Neural Network Ensembles and Combinatorial Optimization with Applications in Medicine

Henrik Haraldsson

Department of Theoretical Physics
Lund University, Sweden

Thesis for the degree of Doctor of Philosophy

Thesis Advisor: Mattias Ohlsson

Faculty Opponent: Tom Heskes

To be presented, with the permission of the Faculty of Natural Sciences of Lund University, for public criticism in Lecture Hall F of the Department of Theoretical Physics on Friday, the 6th of June 2003, at 10.30 a.m.

ISBN 91-628-5685-5
Printed in Sweden by KFS AB, Lund 2003

Till Mamma, Pappa och Morfar

This thesis is based on the following publications:

I Henrik Haraldsson and Mattias Ohlsson,
A New Learning Scheme for Neural Network Ensembles
LU TP 02-04, submitted to *IEEE Transactions on Neural Networks.*

II Henrik Haraldsson, Mattias Ohlsson and Lars Edenbrandt,
Value of Exercise Data for the Interpretation of Myocardial Perfusion SPECT
Journal of Nuclear Cardiology **9**, 169-173 (2002).

III Henrik Haraldsson, Lars Edenbrandt and Mattias Ohlsson,
Detecting Acute Myocardial Infarction in the 12-lead ECG using Hermite Expansions and Neural Networks
LU TP 03-16, submitted to *Artificial Intelligence in Medicine.*

IV Henrik Haraldsson and Mattias Ohlsson,
A Fuzzy Matching Approach to Multiple Structure Alignment of Proteins
LU TP 03-17.

Contents

Introduction

“What’s this physicist doing, talking about hearts and proteins?” Well, the question may certainly seem justified – many people might think of physics as calculating the motion of little colliding wagons, sliding along without friction. Or, perhaps, as the theory of electrons and quarks, and quasars and black holes, that they’ve read about in some book they didn’t really understand everything of.

The plot thickens as I mention that I work with artificial neural networks. “Artificial neu… what??” Or, as one person responded, “Isn’t that something from Star Trek?” Well, artificial intelligence may sound like magic when you hear about it first (at least it did to me), but once you understand it it’s of course not magical at all. “Oh, but *I* wouldn’t understand it”, I hear you say. Well, as a matter of fact you just might. I certainly *intend* to write this introduction in such a way that everybody can understand it in the sense of getting a pretty good picture of it. *Really* understanding all of it in full mathematical detail is another matter, of course — if it could be totally explained in just a few pages, what would I be doing spending four years to study it!

Before I get down to it I should perhaps answer the question what this has got to do with physics. The best answer I can give is that as a physicist you develop a kind of “toolbox” — ways to think about things, ways to calculate things, that you can also apply to areas outside of the classical domain of physics. Most physics research is pretty different from the physics you learn in high school. In school, you learn about simple, often idealized, systems where you can perform exact calculations to predict just how the system will evolve. (Wagons rolling without friction or air resistance, ideal gases in pistons, light beams through prisms.) For real physical systems this is usually impossible, because the system contains lots and lots of tiny parts that all interact with each other, and there’s no way (even in principle) that anybody can find a formula that will exactly describe what all the particles are doing, or even measure exactly where those particles are located for that matter. Instead, you need to make a *model* – a simplified description that does not contain all the full detail, but which still (hopefully) gives an intelligible and sufficiently accurate picture of what's going on. Physics is a lot about making such models, studying them to see how well they fit with experiments, improving them etc. Often we make computer programs based on these models, since calculating by hand how these models evolve is too much to handle, even though they are simplifications. Well, and rather than making models of electrons on crystal surfaces or of fields between quarks, you can choose to make models of proteins, hearts or other aspects of living systems, and that's just what we're doing here at the Complex Systems group at the Deptartment of Theoretical Physics.

Now on to the world of these models.

Artificial Neural Networks

Imagine an excitable man living in a room. He has lots of telephones, each of them connected to a small loudspeaker so that he can listen to them all at once. Suddenly all the phones start ringing; he lifts the receivers, and lots of people start talking to him. Some of them are very happy, others are very sad or angry, others are in an everyday mood – and he takes everything they say very personally. He is not really able to make out the different messages, let alone talk back all these people; he just sits in the middle of his room and listens. As most of the people calling to him are in a happy mood, and as he's a very excitable person who is easily influenced by his surroundings, after a while he starts feeling pretty happy himself. At least it *sounds* to him as if most of the people calling are happy – the voices are transmitted more strongly in some telephone lines than in others, and as it happened the most amplified connections were from some of the happiest people, though there were in fact a number of sad people calling him as well.

Suddenly all the people calling him hang up, and the racket stops as abruptly as it started. All thrilled, he picks up a phone, calls a friend, and excitedly starts talking about how happy he's feeling that all these nice people called.

Next door lives another man who has his room furnished in the same peculiar manner. All telephones connected to loudspeakers, wherever you look. He also got lots of phonecalls, just a while ago – from the very *same* people who called the first guy. Not only that – the people who called him tell him exactly the same thing as they told the first person. In fact, those were exactly the same phonecalls – the people who called them have a special arrangement with the phone company, which means that when they make a phonecall, it gets connected to *both* guys with all the telephones… and to a handful of other people as well! In all, there are a dozen people living on this street, all of them excitable, all of them owning lots of telephones, all of them receiving phonecalls at the same time from the same group of people!

But this does not mean that all twelve end up in the same mood. Oh no. The first guy had some of the incoming phonecalls especially amplified... well, the other ones have too, in fact the incoming phonecalls have a wide range of volumes, from very strong to very faint. But while the first guy had certain incoming phonecalls very amplified, the second guy has some other incoming calls amplified, and the third person has yet a different combination of calls amplified... all of the dozen people getting phonecalls have their own unique combination of amplified and softened phone lines. Which means that though the majority of people calling sounded happy to the first guy, the second guy may well have strongly amplified connections to mostly angry people, whereas the third guy might get an even mix of happy and angry voices, and so on. The first guy was happy after the phonecalls, but the second guy is in a bitter mood - he cannot understand what he has done wrong to be punished so by life; he can't get over the fact that nobody likes him.

Just as the first guy picks up the phone and makes a phonecall after being bombarded with phonecalls himself, so does the second guy, and the third, and the fourth. Each is very influenced by what he has just heard; each is letting his feelings out without restricting himself. The second guy calls to... guess who... he calls to the *same* person as the first guy called to! As do the third and the fourth ones... all twelve people who have gotten phonecalls call to the same person. Who is – a very excitable man; after receiving these phonecalls he is very influenced by what he hears, and...

If you close your eyes and let your imagination loose, you may start seeing a whole world of people all getting phonecalls at the same time from lots of other people, and all then simultaneously calling lots of other people in turn, telling them how they feel now that the've listened, whereupon those people call other people, and so on... well, if you *do* get such bizarre fantasies, that's entirely your own responsibility! Myself I'm a very sane and modest person, who would never tell a crazy story like that. Oh no, I'm just trying to do some theoretical physics here, and I've just finished drawing the picture I need for my explanations. It would never occur to me to tell a tall tale of people calling other people, who call other people who in turn call *other* people, and so on and so forth. Such nonsense! Mathematical rigor! – that's what we need more of in this country. I'm just trying to do a job here. Complex Systems is a production unit, mind you; try to pay some attention instead of day-dreaming.

Did I say that I'm not going to expand the picture further? Sorry! I forgot about the people working in the telephone company. One of them is interested in wines, and she's made some amateur measurements on some different brands of wines, measuring sulphur content, how much light these wines reflect when shone upon with light of different wavelengths, etc. She thinks it would be funny to see if one could determine whether it's a Bordeaux or Italian wine, based on these measurements. As an additional twist, she wants to use the strange people with multiple phones as a kind of giant computational machine.

To do this, she starts by recording voices expressing different levels of happiness or sadness. She intends to play these recordings back to the people with multiple phones. Since they just listen to the general mood, and since they don't try to talk back, they probably won't know the difference, she thinks. She has a number of cassette players, and now she connects a phone line from every cassette player to each of the dozen people with many telephones. All these phone lines have their own level of amplification, as before – these amplification levels are set randomly, to start with.

Now, she takes the measurements from the first bottle of wine, and puts happy or sad tapes into the cassette players depending on these measurements. The first measurement is of sulphur content. The wine had a *high* level of sulphur compared to the other wines; thus, she puts the voice of a *highly* happy person in cassette player number one. Measurement number two is on the reflection of red light. The wine reflected a relatively *low* level of red light, and thus she puts the voice of a person with *low* happiness (i.e. an angry person!) into the second cassette player. And so on.

Then she presses a button which activates all the cassette players and the phone lines going out from them. Thus, the twelve people all get a number of phone calls – as many as there are cassette players. As before, they all get the same phonecalls. But depending on the amplification strengths in the different lines, some mostly hear happy voices, whereas others hear angry people more loudly and end up in a rotten mood.

After a while, the telephone company worker stops these phonecalls. The twelve people who got called up then all call one and the same person, as they're used to, each expressing their feelings. This final person is influenced by the moods of the twelve people he's listening to; mostly so by the people who have their phone lines to him very amplified. The telephone worker observes what mood the final person ends up in, by listening to the next phonecall he makes.

The amplification levels in all the various phone lines are kept fixed unless somebody adjusts them, but the telephone worker is going to adjust them. She has this grand scheme: to adjust the amplification levels in such a way, that whenever she feeds the cassette players with tapes corresponding to a wine and calls the excitable twelve people up, *the mood of the final person in the line will say something about the type of wine*!

Arbitrarily, she has decided that Italian wines shall correspond to the final person being happy, and Bordeaux wines shall correspond to him being angry or sad. If she manages to get all the amplifications right, this whole thing would be very neat. She could then ask a friend to give her a bottle of wine with the label removed; make her measurements on the wine; feed the cassette players with happy, neutral or angry tapes depending on each measurement; and then find out whether it was an Italian or Bordeaux wine by listening to the mood of the final guy getting phonecalls!

As it happened, the first bottle of wine was Italian but the last guy in the line got pretty sad – meaning her classification machine gave the wrong answer, since sadness is supposed to mean "Bordeaux". Well, this is only to be expected, since the amplifications were set randomly. She adjusts all the amplifications with the following goal in mind: that she would get a somewhat more accurate answer, if she were to start these same tapes again. In this case, it means that she needs to make the final person happier. To figure out what adjustments to make, she thinks about the moods of the twelve people listening to the tapes – if one of them was very happy, for instance, she'll want turn up the volume on the connection going out from him to the final guy, as this will make the final guy happier.

Now it may seem that making the final person happy is all about increasing the amplifications from the happy tapes and people, and decreasing the amplifications from those that are sad. This would indeed be the case, if it weren't for the fact that some phone lines have a voice distorter connected to them. These distorters make happy people sound sad and vice versa. Now, suppose that the phone worker wants the final guy to be happier. If there's a voice distorter on a connection from a sad tape, she can turn up that connection and the person listening will hear a louder happy voice! That's suitable if she indeed wants this person to become happier – but maybe this person also has a distorter on his outgoing phone line, in which case she'll need to make him sader to make the final guy happier! This may all sound a little complicated, but it provides the necessary freedom to get the combination of connections right.

After she has tuned the all connections somewhat, she proceeds to wine bottle number two, and starts all over: she prepares the cassette players in the same way as before – putting a happy tape in the first player if the sulphur content is high, and an angry tape if the sulphur content is low; the higher the sulphur content, the happier the voice. And so on for the other measurements and cassette players. She starts all the cassette players and activates all the phone lines, and waits for the twelve people to get happy and angry. Then she turns off the cassette players, waits for these twelve people to call the final person in the line, whereupon she observes his resulting mood. All the connections are then tuned so as to put the final person a bit more into the desired mood – unless he is clearly in that mood already, in which case she adjusts nothing.

Then she proceeds with the rest of the wine bottles, all the time preparing cassette players with tapes matching the measurements, and all the time adjusting the connections a little so as to make the "answer" (mood of the final person) more "accurate" (in line with the true type of wine). She only makes small adjustments each time, so she doesn't entirely mess up the adjustments she's made already. She makes the largest adjustments to the connections that will have the largest effect on the final person's mood. The connections that don't produce a noticeable change when tuned are left more or less unchanged. For example, if a person is very happy already, turning up the volume of a happy phonecall reaching him is not going to have any real influence on his already happy mood. When she's done with all her wine bottles, she starts with wine bottle number one again. Sometimes some adjustments she make counteract some adjustments she made previously, as there is no simple and unique way to way to set the connections, but in general most of the answers get more and more accurate.

She goes on, until the answers do not get any better, at which points she stops making adjustments. Then the grand day has come: she makes measurements on a wine she's never seen before, given to her by a friend in a bottle without label. She prepares the cassette players accordingly, opens the phone lines, and *voilà*: the final guy ends up happy, from which she concludes that it's an Italian wine. When she tells her friend, it turns out that she's right! More people supply her with Italian and Bordeaux wines she hasn't seen before, and most of the time she's able to tell what sort of wine it is just by making the measurements and activating the phones. She becomes a local celebrity. The poor impressionable people who get phone calls all the time become nerve wrecks.[1]

— If you've understood this story (which I've made up entirely on my own), you've understood the principles behind Artificial Neural Networks! Of course, when using neural networks we don't use a group of people that call each other up; we write a computer program that performs essentially the same actions within itself.

[1]Legal disclaimer: whilst the author has made every effort to ascertain that the contents of this introduction are correct, he shall accept no liability of damage should you attempt this experiment yourself. Please consult your physician before attempting any of the exercises detailed herein.

Terminology

Before moving on, I'll summarize the picture and name the central concepts. An artificial neural network is made up of a set of units called *nodes*, which have connections called *weights* between them. These nodes are arranged in layers; signals flow from the first layer on to the second, and then on to the third, etc. (I'm talking about feed-forward neural networks now; there are also other kinds of neural networks that I won't go into.) I've used three layers of nodes in my papers, but one can have more or fewer. A network with five input nodes, three hidden nodes, and one output node is shown in Figure 1. The inputs are the cassette players, the hidden nodes are the excitable people listening to the cassette players, and the output node is the final person in the line. The signals flowing in the wine example were levels of happiness – in a computer the signals in a neural network are just numbers, typically in the range -3 to 3.

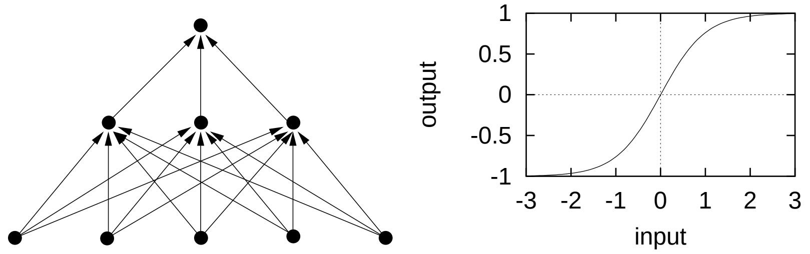

Figure 1: Left: An example of an artificial neural network. Right: a common activation function, tanh(x). This activation function determines how much signal a neuron sends out, given the signal it gets in.

The neurons in the first layer don't do anything complicated – they just send out whatever signal (number) they're told to send out. These numbers sent out by the first layer are simply the inputs given to the neural network. In the example, that would be the sulphur content for node 1, the red light reflection for node 2 and so on. The inputs are normalized; their values are rescaled so that the mean is zero and the standard deviation is one.

In the hidden and output layers, the actual computations are made. Each node here sums the signal it gets – this sum is just a constant *threshold* plus the signals sent from the nodes in the previous layer, multiplied or "weighted" by the strength of the connection in between. (That's why these connections are called weights. The thresholds can be tuned just like the weights, by the way.) This incoming signal to a node – including threshold – then determines how much signal is sent out, through the *activation function.* For small positive or negative input levels, the output is approximately equal to the input. But when the level of input is very high, the output saturates and does not get larger than 1.0 (as can be seen in the right part of Figure 1). It's a bit like the happy and angry people in the story: if a person is very happy because he's listening to a lot of happy people already, then making him listen to yet another happy person is not going to make him very much happier – there's a limit to how happy or sad a person can get.

The whole point of neural network training is this: we have a set of inputs, each consisting of many values $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{iM})$. With each input is associated a single target value t_i.[2] It is assumed that there is no simple, well-known relation between the input and the target; if there were, it would be easier to just write a program that always gives you the right answer based on this rule. Now, the task is to set the connections in such a way, that when the program is given an input it has not seen before, it will produce an output (a guess, a prediction) $y(\mathbf{x}_i)$ that is as close to the target as possible. In the example with telephones, the measurements from one bottle of wine is one input $\mathbf{x}_i$, the target t_i is the type of wine, and the output $y(\mathbf{x}_i)$ is the guess based on the mood of the final person – "Italian" if he's happy, "Bordeaux" if he's not.

In the start, the connections have random values close to zero. When training the neural network (i.e. setting the connections right), we use a *training set* consisting of N *data points* i.e. pairs of inputs and targets $(\mathbf{x}_1, t_1)$, $(\mathbf{x}_2, t_2)$, $\dots$, $(\mathbf{x}_N, t_N)$. To guide us when making calculations on how to set the connections to get the outputs right, we define an *error function* E which measures the mismatch between the targets and the outputs $y(\mathbf{x}_i)$ produced by the neural network when given the inputs $\mathbf{x}_i$ in the training set. Sometimes we call this error function the *energy*, as we're physicists using methods from statistical physics to solve these problems.

One choice of error function is a sum-of-squares error; it is the sum of the squared difference between each output and target.

$$E = \sum_{i=1}^{N} \left(t_i - y(\mathbf{x}_i)\right)^2 \qquad (1)$$

Clearly, this error is never negative, since the square of a number is never less than zero. The error is exactly zero when all the outputs are exactly equal to all the targets. (If you don't understand this formula, or the details of these arguments, don't be depressed – you don't need to understand it to get the

[2]It is simple to generalize to the case of a multi-dimensional target $\mathbf{t}_i$, by giving the neural network many output nodes, one per output component.

big picture. I'm giving it just to give people who do understand formulas some extra flavor.)

There is another set of data points, called the *test set* which is set aside before the training starts; the information in these data points must not be used during the training. After the training is completed, no more adjustments to the connections are made, and the test set is used to evaluate how successful the training is and maybe to compare it to other methods. This procedure is supposed to mimick the scenario when you build a classifying machine and then sell it to someone else, who uses it on newly measured data points you've never seen; that's the real test of a method's usefulness. (It is assumed that the new owner of the machines performs these measurements in exaclty the same way as you did when you built the machine, and on the same type of objects, of course.) In the telephone example, the training set is the wines used to adjust the phone lines, and the test set is the bottles without labels supplied by the friends.

Finally, a remark about the word "neural". The construction of these artificial neural networks was inspired by the way neurons (nerve cells) work in the brain, hence the name. The purpose is not to make a model of the brain – at least not in this thesis, nor in most neural network research, though there are some scientists interested in how artificial neural networks relate to brain function.

Aspects of Training

The training of the neural network is the same thing as making the error E small. When the error is small, meaning that the neural network gives an output that is close to the target most of the time, we say that the neural network has "learned" its task.

How do we set the connections right to make the error small? Well, we can't increase the connections in small steps and try all the combinations – that would take far too long time. (Much longer than the age of the universe, actually, for all but the simplest training sets.) Instead, let's keep all connections but one fixed, and vary that last connection. For each value of that connection, we get a distinct value of the error – the error is a function of the connection's strength. Since we know the training set and exactly how the neural network performs its calculations, it is possible to write down a relatively simple formula of how the connection should be adjusted to reduce the error. (This way the program does not have to experiment by increasing and decreasing the connection to see what happens).

If there were just two connections, we could plot the error for each value of the connections as a surface or "landscape" above the coordinate axes corresponding to the two connections. Finding a low error then corresponds to finding a low valley in this landscape. Now there may be a hundred connections, and it's impossible to visualize a hundred-dimensional coordinate system or draw a landscape above it, but we still speak of an error landscape in analogy with the two-dimensional case. We don't have an overview of this whole landscape (getting an overview would amount to trying all the combinations), but the most obvious way of finding a low valley is to go downhill from the point where you're standing right now, and that's actually the basic principle behind most ways of training neural networks. These methods are called gradient methods, as the direction of steepest slope is called the gradient.

Going in the downhill direction as seen from the turf you're standing on right now may not lead you to the deepest point, if it's nighttime and you're unable to see the whole landscape. It's like letting a little ball roll down a hillside; it might get stuck some place along the slope. (Not to mention the fact that there might be an even deeper valley on the other side of the hill.) To deal with this problem, a large variety of methods is used. Some use a momentum term, which corresponds to making the ball heavy so it picks up speed; even if it happens to fall down into a little hole on the hillside, it rolls up again because of its speed, and then continues downhill. Other methods add noise to the movement, which corresponds to giving the ball kicks in random directions now and then, to prevent it from getting stuck. When looking at the Population method (see page 14 and Paper I), we shall see that there are methods which don't use information about the direction of slope at all.

Noise and the Need for Diversity

OK, so how good can we get at doing this? We do have some measurements, containing information, from which we want to build a neural network to give us good predictions about something we want to know. But these measurements almost never contain the *complete* information about the system we study. Consider the case of making a diagnosis of a patient – perhaps we have an ECG from which we cleverly extract some information which we then plug into our neural network, or maybe we run some other medical tests and use the values of those. Clearly these tests don't tell us *all* there is to know about the patient. If we really were to know if the patient has an infarction or not, we would need to know exactly how his or her heart looks – but there's no way we can make that out by just looking at the ECG. In addition, and equally importantly, you're almost never able to make totally accurate measurements;

there's always some noise involved. Therefore we can never hope to make perfect predictions based on these measurements. We could even in principle imagine the case of two patients displaying identical ECGs, but where one is healthy and one is not.

Thus, the target t_i is not a clean function of the input $\mathbf{x}_i$ in the way we learned about mathematical functions in high school. If we let $f(\mathbf{x}_i)$ denote the true relation between the input $\mathbf{x}_i$ we observe and whatever we're trying to predict, then the observed target t_i always differs by some noise ϵ:

$$t_i = f(\mathbf{x}_i) + \epsilon \tag{2}$$

If we're doing regression, i.e. trying to predict continous observables such as the outdoor temperature or the stock rates tomorrow, we often assume the noise ϵ to be normally distributed. If we're doing classification, i.e. trying to predict observables which take one of a few distinct classes such as "sick" and "healthy", this noise is instead some probability of changing the class label.

The fact that there's noise involved imposes more problems than one might realize immediately, especially as there are never infinitely many data points in the training set. (If we're trying to predict infarction, we've only got as many data points as there have been heart patients in our hospital's emergency department. Getting many data points might be a time-consuming and costly procedure, and so one has to get the most out of what's available.) Consider the artificial situation where the true but unknown relationship is given by

$$f(x) = x^3 - x^2 - x + 1 \tag{3}$$

but where there is normally distributed noise ϵ with standard deviation 0.3. Suppose that we have 10 data points in the interval $x \in [0:2]$ from which we want to create a neural network or some other predictor to get a picture of the underlying function and to be able to predict the output at other values of x. Let's fit the data points we have now, our training set, to some polynomials by minimizing the sum-of-squares error – that's a simple procedure which any mathematical plotting program can do.

When fitting polynomials in one variable, it is always possible to make the error zero and the curve run through all our N points if we use a polynomial of degree $N-1$. Well, isn't that great. So we fit a ninth-order polynomial

$$ax^9 + bx^8 + cx^7 + dx^6 + ex^5 + fx^4 + gx^3 + hx^2 + ix + j \tag{4}$$

to our data points – the result is shown as the dotted line in figure 2. Whoops! As can be seen, it's varying crazily. It matches the points in the training set perfectly, but if it were to be used to give predictions of new data points, it would give lousy results. (For example, for $x = 0.25$ it predicts an output value of ≈ 3, which is far away from the true value of 0.7.)

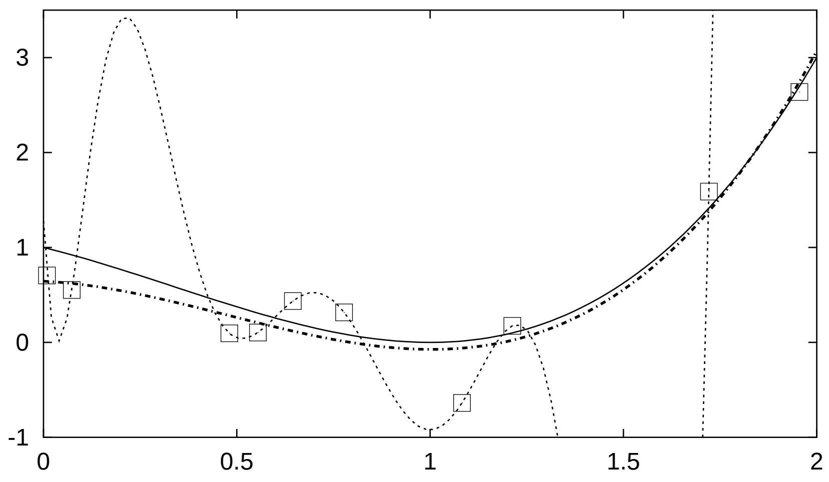

Figure 2: An illustration of the effect of overfitting. The ten data points in the training set (boxes) are noisy samples from the true function (solid line). The dotted line shows a fitted ninth-order polynomial. It is clearly overfitted to the noise in the training data, thus being useless for predicting new data. The thick, dash-dotted line is a fitted third-order polynomial. As can be seen, though it matches the training data less well, the overall curve is much closer to the true function.

A third-order polynomial (the thick, dash-dotted curve) would in this case be a much better choice. It does not have as much freedom to vary and so does not follow the data points as closely as the ninth-order polynomial. In other words the training error is higher, but the predictive power is much better, as can be seen in the picture. For $x = 0.25$ it gives a prediction of 0.5 which is much closer to the true value 0.7. For $x > 1.5$ it follows the true curve almost pefectly.

The phenomenon of the wildly varying ninth-order curve is called overfitting. In this case, the polynomial has adapted too much to the noise inherent in the dataset; it has "learned" the noise and not just the signal. Even for points near the available data points, it provides a worse prediction that the third-order curve with less freedom, and far away from the data points it's obviously not able to interpolate sensibly at all.

The same thing can happen to artificial neural networks. The more hidden neurons you put in, the more freedom the neural network has to adapt to complicated functions, but you also run the risk of overfitting to the noise in the data. On the other hand, if you have too few neurons the network won't be flexible enough to adapt to the true input-output relationship. Therefore it's important to get the number of hidden neurons approximately right. To do this, we make a preliminary training using just two-thirds of the training set. Then we check how good the neural network is at predicting targets of data points it hasn't seen before, by testing its performance on the last third of the training set, called the *validation set.*[3] This is done a few times, using a different number of hidden neurons each time; then the number of neurons that worked the best is chosen for the final training. (It is forbidden to peek at the test set during training, and that's why we have to find the best number of neurons by testing each trained network on a part of the training set.)

Another strategy often used to avoid overfitting, is to add a *regularisation term* to the error during training. This gives a penalty when there are lots of highly amplified connections between the neurons. Because of this penalty term, some connections get more or less turned off; this gives the network less freedom to adapt and reduces the risk of overtraining. The strength of this regularisation term, and some other training parameters as well (such as how fast to go downhill in the error landscape) are tuned during cross-validation.

A totally different approach to deal with the problem of noise in the data, is to create an *ensemble* of neural networks. This means that you train many neural networks and then let them vote, so to speak, by averaging over their outputs in some way and letting this average be the final answer. The idea is that some networks will maybe give an output that's too high on a given datapoint whereas other networks might give an output that's too low, but if you average over lots of such outputs you might end up with an answer that's just right.

This strategy clearly only works if the different networks give different answers. The simplest way to achieve this is to train each network using randomly initialized connections. Another approach is to train each network on a different part of the training set. Other methods give each network a different number of "votes" when averaging their answers together, based on how different

[3]Actually, the parts used for training and validation are then shuffled around, so that we also train on the last two-thirds and use the first third as a validation set, and finally train on the first and last third and use the middle third as a validation set. The performances of these three runs, each using a different third as a validation set, are then weighted together. This process is called *cross-validation.*

its output is from the other networks in the ensemble. This all increases the desired *diversity* among the ensemble members. However, the diversity must not increase so much that the *accuracy* of the members becomes poor, as this will degrade the overall performance. There is a trade-off between the diversity and the accuracy, and this issue is explored further in Paper I. That article also presents a new way of training neural networks to create a diverse and accurate ensemble – the Population method.

The Population Method

Once upon a time there were some mountains where there lived one hundred blind rabbits. Now and then, a rabbit split into two or more rabbits – this was an unusual kind of rabbits, where rabbits multiplied by splitting themselves, rather than by mating. There was also a special curse on the rabbits, which meant that when one rabbit was born, another rabbit had to die, and so their number was kept constant at one hundred.

The rabbits jumped up and down the hills in a random fashion, blind as they were; the rabbits in the valleys had more grass to eat than the rabbits on the mountaintops, and so they split more often. But sometimes a rabbit who had been crossing a hilltop found a thitherto undiscovered valley where the grass was even greener, and so got even more children (or clones) than the rabbits in the other valleys. When a rabbit had just been "born" it was right next to its "parent", but then it immediately started jumping around randomly on its own.

In the beginning of the story it was summer, and so there was lots of grass everywhere. The rabbits on the hilltops had almost as much food to eat as the ones in the valleys, and it was easy for rabbits to wander long distances without starving to death. But then the autumn came, and it grew gradually cooler, and the grass on the hilltops started getting sparse. This favored the rabbits who had found the best spots, so that after a while most of the rabbits were concentrated in the greenest regions. Since there was now less grass on the hilltops, it rarely happened anymore that rabbits found totally new regions by walking long distances. Also, the rabbits started freezing, and so they made shorter leaps than in the summer.

Eventually it became winter, and the rabbits in the higher, less grassy regions were so disadvantaged that eventually all the rabbits were concentrated in one region. There were one hundred of them there (remember, their number was fixed to one hundred by a magic spell), and they were making even shorter leaps as they started freezing even more, and so they in effect explored that green region very thoroughly even though they were blind and jumped in random directions. Eventually, they were all concentrated in some exceptionally green patches here and there – so exceptionally green that it would have been very difficult to find them, even if the rabbits had searched for them by jumping downhill rather going off in any direction. And so it turned out that in effect, the group of rabbits had performed an efficient search, ending up in diverse patches that all were green, though they were only jumping about randomly.

What has this got to do with neural networks? Well, each rabbit is one neural network, the position of each rabbit as seen from above corresponds to the values of two of the connections – though the rabbits can only travel in two dimensions (north/south and east/west), but the weights of the neural networks can wander around in a coordinate system with as many dimensions as there are connections. The altitude corresponds to the training error E of the network in that state. Green valleys are areas with low error, and barren hills are areas with high error.

If you've understood this parable, then you've understood the essentials of the Population Method of Paper I! The fact that the moves are random is in some ways an advantage: it is much quicker for the computer to update the weights of the networks randomly, than to compute the direction of steepest descent.

Feature Extraction

Now you've learned a bit about data sets and some ways to train artificial neural networks. But who decided what these data sets should look like? After all, there are a number of measurements which could reasonably be used as inputs. Who says that it's enough for the neural network to look at ECGs; maybe you should give the network information about the age, weight or blood pressure of the patient as well?

The answer, of course, is that you put whatever measurements in that you think will be most useful for getting the answer you want. Making this decision is not an easy choice, and to some extent you have to use trial and error. Paper II is about this; examining whether the neural network diagnosis of myocardial infarction (heart attack) can be improved by letting the neural network look at more types of measurements. Another difficult question is: suppose that you have a very complicated measurement, like a "bulls-eye" image of the heart taken with a SPECT machine. How do you turn this image into a set of numbers that the neural network can train on? You can't split the image into tiny elements (pixels) and let the intensity in each pixel represent one input – there are thousands and thousands of pixels, and experience shows that you can't make neural networks to work well if you make them this big. (A typical number of input nodes is 5-100.) You need to extract *features* from the image - numbers that tell you something important about the image. Extracting such features is an art in itself.

In the case of ECGs, the most common kind of feature extraction consists of measuring amplitudes (how high the peaks are), durations (how many milliseconds the beats last) and slopes (how sharply the peaks rise). This works very well. But maybe there is an even better way? In Paper III, we tried to extract features by expressing ECGs as an approximate sum of Hermite basis functions. It's a bit complicated, but the idea is that you have a set of wavy-looking curves, shown on page 67 and denoted by $\phi_n(t,\sigma)$, and then you try to write an ECG beat as a sum of such functions. If we call the ECG beat $x(t)$, maybe we get

$$x(t) \approx 1.73 \cdot \phi_0(t,\sigma) - 1.12 \cdot \phi_1(t,\sigma) - 0.44 \cdot \phi_2(t,\sigma) + 0.12 \cdot \phi_3(t,\sigma) + \dots \quad (5)$$

The point of it is this: given an arbitrary function $x(t)$, the coefficients (1,73; -1.12; -0.44...) are uniquely determined, and the approximation gets better and better the more terms you include – look at the figure on page 69.

Now, rather than measuring amplitudes, durations and slopes on ECGs and plugging them into a neural network, we plugged a few Hermite expansion coefficients (1,73; -1.12; -0.44...) into the neural network instead. It is possible to reconstruct a good approximation of the ECG from these coefficients, which means that they contain all the relevant information about the ECG. The neural network should be able to find that information and learn to make good predictions from it, if the algorithm for training it is good. We tried it – nobody ever did that before – and indeed it turned out that it worked. The results were only marginally worse as compared to training with the amplitudes and slopes, and this difference was not statistically significant. Hermite decomposition has the advantage that it's possible to reconstruct the full ECG from the features (coefficients) you extract, something which is not possible if you just measure amplitudes and slopes. Knowing how high a beat is at a couple of points, and how sharp the rise is at the steepest point, is simply not enough information to draw the full ECG. Let's keep this in mind when looking at a different question – how to understand what's going on inside a neural network.

Looking Into the Black Box

OK, so we're able to train these neural networks, and it works really well for giving predictions. But does this mean that we really *understand* what's going on?

Well, not necessarily, in fact. There are lots of theories and theorems regarding how neural networks learn things in general, but when it comes to studying a specific dataset, there is more to it than just writing a computer program that will give a good guess. Suppose you're a doctor who's talking to a patient who has just come to the emergency department of the hospital with chest pains. You run some tests and talk to the patient, and based on your medical schooling and previous experience, you decide that it is likely though not certain that the patient is suffering from a heart attack. But then there is this ECG machine with an integrated state-of-the-art artificial neural network, and the neural network is telling you that the patient is *not* suffering from a heart attack – without giving any clue about why it's giving this answer. If the program is right, you'd better not subject your patient to an unnecessary and expensive treatment that might harm him or her. On the other hand, if the patient *is* really suffering from a heart attack, it's important to start the treatment immediatley. Who are you to believe? You have a tremendous responsibility – you can't just say "well, I think this patient is sick, but this computer does not, so what the heck – I won't bother with any treatment".

Obviously, the neural network would be more useful to the doctor if it could give some hint as to *why* it's giving the answer it is. The doctor could then compare this information with his or her prior knowledge. But extracting this information from the neural network is not an easy task – the network just consists of a lot of connections with various levels of amplification! It's like a black box where we put a question in at one end and get an answer out at the other, but where we don't really know what's going on inside. There's no way a human can understand a lot by looking at these connections or the activation level of the hidden nodes. OK, if there is one input with lots of strong connections going out from it, then one can reasonably guess that this input has a high impact on the output and is generally important. That's something, but it's still a far cry from really understanding what's going on in an individual case. To do so, you need to do something more sophisticated.

We did try something special in Paper III, as a matter of fact. We asked the question: given an ECG and a neural network an existing set of connections, how can you vary a few of the inputs by a limited amount so as to create a change in output (diagnosis) that's as large as possible? If the input is an ECG and the neural network gives the answer "infarction", this amounts to asking the question: "what would the ECG have had to look like for the network to have answered 'healthy' instead, if we can only change the ECG a little?" Having found the answer to that question, we compared the original ECG to the modified one, and found out what parts of the ECG that were most important to the neural network when making its diagnosis. This is certainly interesting for a doctor to know. Now, remember that the reconstruction of a modified ECG from modified neural network inputs is impossible in the traditional approach when using amplitudes and slopes. The possibility of making reconstructions is a distinct advantage with using Hermite decomposition.

Combinatorial Optimization

Now, how do we find the "small" change in neural network input that will produce the largest decrease in output? We have to start by deciding just how large changes to the inputs we permit ourselves to do. In Paper III, we decided to change six out of the 80+ inputs, each by an amount not greater than about one-tenth of the difference between "jumping with joy" and "totally depressed" as expressed in terms of the telephone parable; by not more than half a standard deviation, to be precise. To make our task somewhat easier, we divide the changes of each input into ten steps, so the number of combinations is not infinite. They're still so many there's no way we can try all of them, though.

To be able to solve a problem like this with a computer algorithm, we need a mathematical measure of what we mean by a "good" solution. In the case of training neural networks, it was the sum-of-squares error. In the case of varying a few inputs to make the neural network output *smaller*, chosing the measure of performance is easy: it's just the increase in output. (*Increase* because we're going to minimize this measure i.e. make it very negative, and and a negative increase means a decrease.) But wait – we add another term that is zero when exactly six inputs are modified, and considerably larger than zero otherwise. This way our algorithm can try other combinations where another number of inputs is modified, but if our algorithm is good at minimizing this measure of performance, it should end up in a state with exactly six inputs modified, because that's where the second term is smallest.

If we strip this problem down to the bare basics, what have we got? We have got a set of variables, each defining the change in one input; each variable can be in one of eleven states. (That's ten states for various amounts of change, and the eleventh state meaning "no change".) For each combined state of these variables, there is a well-defined energy. That's it – that's all there is to it. Hey, that's just like a magnet! "Is it???", I hear you say. Yes, I assure you!

A magnetic crystal is made up of a set of molecules; each molecule has a *spin* which is a measure of where its magnetic field is directed. (It is called "spin" because it is generated by the electrons which are thought of as spinning around their own axis.) For some materials, this spin can only be in one out of a definite number of states – at least to some level of approximation; and physics is about making models. For each collective state of these spins, the crystal has a certain energy. Nature functions in such a way so as to minimize this energy. That's just like the problem we're trying to solve:

- There is a set of variables, each of which can be in one of a definite number of states.
- Each collective state of the system has a certain energy.
- The system evolves to the state where this energy is as small as possible.

When studying these models, physicists have developed an algorithm called Mean Field Annealing to simulate these processes and find a state with low energy. This algorithm does not take into account any details of the interactions between the different spins, only the energy of each state. Now that our problem of finding a large decrease in neural network output is formulated in exactly the same terms as a spin system, containing all the information needed by the mean field annealig algorithm, we can use this spin system algorithm to get a good solution to our neural network problem! And that's exactly what we did in Paper III. It turned out to work reasonably well; in about half of the cases, it produced a modified ECG with changes in the same part of the ECG plot as a human expert focused on when asked what part of the ECG indicated infarction. The method certainly needs refinement before it can be used in the emergency room, but we think that what we have already looks promising and much more appealing than the "black box" operation so often characteristic of neural networks.

We used mean field annealing in Paper IV as well, but to solve a completely different problem: to align a number of proteins structures to each other. Imagine that one could place a number of proteins in exactly the same place, so they occupy the same space. Now look for parts of the protein structures that look exactly the same in all proteins, and move those regions "on top of" each other so to speak, so the proteins are oriented in the same way. By looking at how proteins match each other, a biologist can compare them and learn something about their function and evolutionary history.

We solved this problem by defining a set of variables that each encoded whether or not a given amino acid in a protein should be aligned to a certain amino acid in a "consensus structure", which is like a virtual average of all the proteins.[4] These variables could each be in one of a definite number of states. Depending on how well the proteins matched, each state had an energy. We were trying to minimize this energy; thus we used the mean field annealing algorithm. These

[4]A protein is a long chain of a couple of hundred amino acids or so.

matchings were alternated with moving each protein so its amino acids were as close as possible to their matched counterparts in the consensus structure, and then forming a new consensus structure as the average of the current positions of the proteins. Thereafter, the amino acids were matched again, etc.

Explaining this in detail would take us too far afield; the popular introduction ends here. I find it fascinating that two such dissimilar problems as changing neural network inputs and aligning proteins can be solved using the same method, and if you do too, then I've succeeded. If you haven's studied mathematical modelling and artificial intelligence at a university level, I'm afraid that you won't understand the rest of this thesis, except for the acknowledgements a page ahead. It's been a pleasure having you with me this far! – now I will move on to a more exact and rigorous account.

The Papers

Here follows a brief overview of the papers presented in the second part of the thesis. These papers have been discussed in the introduction already; the discussion here is more technical but still with few details.

Paper I

In this paper, the Population method for training an ensemble of neural networks is presented and explored. Particular emphasis is placed on the subject of diversity, and implicit and explicit ways of creating diversity are discussed. The evolution of the population over time, using measures of autocorrelation and replication rate, is also illustrated. The algorithm is tested on a testbed of three datasets, and the performance is seen to be comparable to a Markov Chain Monte Carlo method of training neural networks, and to outperform a Bagging ensemble.

Paper II

Having explored a method for training neural networks in Paper I, the inputs of the datasets being given beforehand, this paper explores the issue of input selection. Artificial neural networks have previously been successfully applied for automated interpretation of myocardial perfusion images. So far the networks have used data from the myocardial perfusion images only. The purpose of this study was to investigate whether the interpretation could be improved by also using other clinical measurements as inputs to the neural network. These measurement were ECG amplitudes, heart rate and maximum achieved workload when exercising on a bicycle ergometer. It was found that including this extra information did not result in improved classification for the neural network ensemble.

Paper III

Here the issue of input selection is taken one step further, by exploring a new way of feature extraction for training neural networks to detect signs of acute myocardial infarction in ECGs. The 12-lead ECG is decomposed into Hermite basis functions, and the resulting coefficients are used as inputs to the neural networks. The resulting predictive performance was marginally worse compared to using the traditional approach measuring amplitudes, slopes and durations; this difference was not statistically significant.

Furthermore, a case-based method of understanding the classificiation made by the neural network was presented. Six inputs corresponding to two ECG leads were perturbed by not more than half a standard deviation each; this perturbation was selected so as to produce the largest possible change in output. To achieve this, the perturbations were discretized and expressed in a Potts spin formulation. Fuzzy spins were used, with a temperature regulating the degree of fuzziness. An energy function was defined, with one term maximizing the output change and two other terms favoring solutions with exactly six inputs from two leads selected. The energy was then minmized through mean field annealing. Because the Hermite decomposition is invertible, it is possible to plot a modified ECG from the modified inputs; this modified ECG can then be compared to the original one to reveal regions critical for neural network response. The leads selected by the algorithm were compared with those selected by a human expert; at least one lead agreed in roughly half of the cases.

Paper IV

In this paper, the Potts spin formulation and mean field annealing was used to solve an entirely different problem: the multiple structure alignment of proteins. For pairwise alignment and fixed positions of the proteins, the optimal alignment of amino acids is achievable in polynomial time with the Needleman-Wunsch algorithm. We used a similar approach, but with soft assignments calculated from a fuzzy generalization of the optimal cost for aligning subchains of the proteins. Naturally, the positions of the proteins should not be fixed but rather updated as the alignment gets better and better, and so the soft alignments were alternated with translations and rotations of the proteins to minimze the RMS distance between the amino acids matched so far. The proteins were not matched against each other directly; rather, each protein was matched to a virtual consensus structure. The matchings and translations/rotations were alternated with the calculation of a new consensus structure, by placing each virtual consensus atom in the middle of the C_α protein atoms it's aligned to.

Five protein families were aligned using our method, and the results were compared against a manual gold standard, the HOMSTRAD database, with good results. The alignments found were in general more accurate than those produced by a competing algorithm based on Monte Carlo. Because our algorithm is based on matching each protein against a consensus structure, and because no initial all-to-all matchings are needed, the CPU cost is modest and only grows linearly with the number of proteins to align, for a fixed chain length.

Acknowledgments

My thanks go to Mattias Ohlsson for always being helpful and forthcoming. I have learned a lot, and as time has passed our co-operation has become more and more cheerful. I also thank Carsten Peterson — few people I know have such an ability to inspire the feeling that something exciting, special and important is really going on. I thank everybody at the department for making it the place it is — in particular Anders Irbäck for providing ample opportunities for off-hour socialization, by arranging the recurrent running trips to Copenhagen and Skrylle.

A New Learning Scheme for Neural Network Ensembles

Paper I

A New Learning Scheme for Neural Network Ensembles

Henrik Haraldsson and Mattias Ohlsson

Complex Systems Division, Department of Theoretical Physics
University of Lund, Sölvegatan 14A, SE-223 62 Lund, Sweden
`http://www.thep.lu.se/complex/`

LU TP 02-04, submitted to *IEEE Transactions on Neural Networks*

We propose a new method for training an ensemble of neural networks. A population of networks is created and maintained such that more probable networks replicate and less probable networks vanish. Each individual network is updated using random weight changes. This produces a diversity among the networks which is important for the ensemble prediction using the population. The method is compared against Bayesian learning for neural networks, Bagging and a simple neural network ensemble, on three datasets. The results show that the population method can be used as an efficient neural network learning algorithm.

1.1 Introduction

Ensemble learning for neural networks is a subject of active research. It enables an increase in generalization performance by combining several individual networks trained on the same task. The ensemble approach has been justified both theoretically [1, 2] and empirically [3]. The creation of an ensemble is often divided into two steps [4], the first being the judicious creation of the individual ensemble members and the second their appropriate combination to produce the ensemble output.

Combining the outputs is clearly only relevant when they disagree on some or several of the inputs. This insight was formalized by Krogh and Vedelsby [2], who showed that the squared error of the ensemble when predicting a single target is equal to the average squared error of the individual networks, minus the diversity defined as the variance of the individual network outputs. Thus, to reduce the ensemble error, one tries to increase the diversity (called the "ambiguity" in [2]) without increasing the individual network errors too much.

The simplest method for creating diverse ensemble members is to train each network using randomly initialized weights. A more elaborate approach is to train the different networks on different subsets of the training set. Examples of this include Bagging [5] where each training set is created by resampling (with replacement) the original one, with uniform probability. Boosting [6] also uses resampling of the training set, but here the data points that were poorly classified by the previous networks receive a higher probability.

There also exist methods that explicitly try to maximize the disagreement among the ensemble members. The ADDEMUP algorithm of Opitz and Shavlik [7], which uses genetic algorithms to create accurate members that disagree, was shown to perform well on four real-world applications. The idea of maximizing the diversity can also be found in the work of Liu [8]. Another approach was taken by Rosen [9], where an error correlation penalty term was added to the network error function to reduce the correlations of individual network errors. More developments and good selections of important ensemble research can be found in [4], [10] and [11].

Since neural network ensembles are easy to use and give better performance than single neural networks models, they are also used in real-world applications. From the medical domain one can find lung cancer cell identification [12], diagnosis of breast cancer [13], diagnosis of small round blue cell tumors using gene expression profiling [14] and detection of acute myocardial infarction using electrocardiograms [15, 16].

Here we present an approach that shares some of its philosophy from evolutionary neural networks [8] and genetic algorithms [17]. The key idea is to obtain set of disagreeing network members by initially populating different parts of state space. The method creates a *population* that consists of many neural networks. The network population is trained using Monte Carlo techniques with a Boltzmann distribution type of statistics. Each network evolves in state space by random weight updates together with a replication step where the most probable networks replicate into several identical copies. The Boltzmann distribution is used when finding probable networks. The ensemble output is obtained by simple averaging of the individual members. Some of the key features of the population method are:

- General approach, where almost any type of model and energy function for the population can be used.
- The training of the population members and the development of the population itself are intimately connected.

The population method was tested against traditional ensemble methods and Bayesian learning for neural network using hybrid Monte Carlo sampling [18, 19]. The test suite consisted of two real-world classification problems and one function approximation task. The results showed that our approach can produce neural network ensembles with good predictive performance, comparable with the other methods.

This paper is organized as follows: In section 1.2 we outline the *population* method and in section 1.3 some of its properties are discussed and exemplified numerically. Section 1.4 contains numerical explorations and comparisons. A summary and discussion is given in section 1.5.

1.2 Method

The purpose is to create a diverse ensemble of networks with good predictive performance. For this purpose, we consider a population of neural networks (in the order of 100 networks). During each iteration in the training process, networks evolve in state space by random moves, followed by a replication step where poorly adapted networks vanish and well-adapted networks replicate. Initially, poorly adapted networks have a relatively large probability to "survive". As the training continues, however, the selections gets stricter as well-adapted networks are favored more and more. The aim is to end up with a population that populates regions of state space where good networks are found.

First we consider the properties of individual networks; then we describe the population algorithm itself.

1.2.1 The ANN models

We consider neural networks in the form of feed-forward multilayer perceptrons (MLP) with one hidden layer and no direct input-output connections. Let $\vec{\omega}_i$ be the weight vector for each neural network (including thresholds). The hidden unit activation function is tanh(); the output activation function is linear for regression problems, logistic for binary classification, and softmax for classification with many classes. For regression, the targets are normalized during training. The inputs are always normalized.

For each model i we define an error function $E^{(i)}_{err}$ that measures the mismatch between the model and the training data. Let P denote the number of data points and K the number of outputs in the dataset. Possible choices of $E^{(i)}_{err}$ are:

$$E^{(i)}_{err} = \frac{1}{PK}\sum_{p=1}^{P}\sum_{k=1}^{K}\left(y^{(i)}_k(p) - t_k(p)\right)^2 \tag{1.1}$$

$$E^{(i)}_{err} = -\frac{1}{PK}\sum_{p=1}^{P}\sum_{k=1}^{K} t_k(p)\log y^{(i)}_k(p) \tag{1.2}$$

$$E^{(i)}_{err} = -\frac{1}{P}\sum_{p=1}^{P} t(p)\log y^{(i)}(p) + (1-t(p))\log\left(1-y^{(i)}(p)\right) \tag{1.3}$$

where $y^{(i)}_k(p)$ is the k:th output of network i using data point p as input, and $t_k(p)$ is the corresponding target. These error functions are used in regression, multi-value classification and binary classification, respectively.

In addition we introduce a weight elimination term [20], controlled by a tunable parameter λ, to regularize the network.

$$E^{(i)}_{reg} = \lambda\sum_{c}\frac{\omega^2_{ic}}{1+\omega^2_{ic}} \tag{1.4}$$

The sum for network i runs over the weights of the connections c between the layers, but not over the thresholds.

We define the total error, henceforth called the energy, of model i as $E_i = E_{err}^{(i)} + E_{reg}^{(i)}$.

1.2.2 Outline of the population method

The population method is executed by repeatedly performing steps 1-3 below.

0) Start with N ANN models that all are initialized by random weight vectors $\vec{\omega}_i$ $(i = 1, \ldots, N)$. As a time-saving preparation for the actual population method updates, the models are moved closer to the interesting regions by a rough back-propagation training for ∼5 epochs.

1) Update each $\vec{\omega}_i$ by making a random move: $\vec{\omega}_i \rightarrow \vec{\omega}_i + \epsilon\vec{\alpha}_i$, where $\vec{\alpha}_i$ is a vector of length 1 with a random direction and ϵ is a small number compared to unity.

2) Introduce a fictitious temperature T which determines the amount of competition between different models. Compute the energies E_i, Boltzmann factors B_i, average Boltzmann factor $\langle B \rangle$ and population (fitness) factors r_i:

$$B_i = e^{-E_i/T} \tag{1.5}$$

$$\langle B \rangle = \frac{1}{N}\sum_{i=1}^{N} B_i \tag{1.6}$$

$$r_i = \frac{B_i}{\langle B \rangle} = N\frac{e^{-E_i/T}}{\sum_j e^{-E_j/T}} \tag{1.7}$$

   From equation 1.7 it follows that $\sum_i r_i = N$.

3) Write r_i as

$$r_i = L_i + \delta_i \tag{1.8}$$

   where $L_i \geq 0$ is the integer part of r_i and δ_i is the remainder, $0 \leq \delta_i < 1$. Place L_i replicas of model i into the new population. Add one more replica with probability δ_i; this stochastic element among other things gives poorly fitted models (with $r_i < 1$) a chance of surviving. Do this for all models. In average this procedure will maintain N models.

   The population size is constrained to be within ±4% of the initial size. If the new population size is outside of these bounds, the proposed population is rejected and a new one is sampled. The ±4% interval is wide enough to keep the rejection rate small, and still narrow enough that the size is roughly constant.

A completion of items 1-3 is called one iteration.

In the first part of the training, the temperature is annealed from the initial value T_i to the final value T_f by multiplying it each iteration by a constant factor < 1. The step size is also lowered from ϵ_i to ϵ_f in the same fashion.

After the annealing, the population continues to evolve according to items 1-3 before the actual network ensemble is sampled. This sampling is performed by recording all models in the population at S different times, with an interval of D_S iterations between each such "snapshot". The default value of S is 10 to get a relatively dense sampling, while D_S is set so that the sampling is performed over the last two-thirds of the post-annealing phase. The final ensemble output $\langle y_k(p) \rangle$ is calculated simply by averaging over all the network outputs in the ensemble.

1.2.3 Similar Approaches

The proposed method is to some extent similar to genetic algorithms [17]. Genetic algorithms evolve a population of models, measure their performance by an objective function f_i and calculate the fitness as $f_i / \langle f \rangle$, whereupon a number of replicas are stochastically created in proportion to the fitness value. This approach is shared with the population method, for which $f_i = B_i = e^{-E_i/T}$. However, unlike genetic algorithms, the population method does not require its models to be represented as binary strings, and does not apply crossover operators to combine different models. In this respect, the population method is more similar to evolutionary programming.

Evolutionary programming is used in [8] to evolve a neural network ensemble of size N. In each iteration, a few extra networks are added by replicating some networks chosen with uniform probability, whereupon all new models make a rather large random move followed by back-propagation training. The training uses a cost term that explicitly favors diversity (cf. sections 1.3.2 and 1.5). Each iteration ends by pruning the the population to the N fittest networks. Though reminiscent of the population method, our method is different in the replication procedure and in the weight updates, which are random rather than gradient-based; the training is essentially performed by the replication step itself.

Another major difference lies in the annealed temperature parameter T that the fitness is based upon. This concept of annealing a temperature and considering a Boltzmann factor $e^{-E/T}$ is also used in simulated annealing [21], but in the latter method a new state is proposed and then accepted or rejected based on the difference in energy; a rejection means that the system returns to its previous state. In the population method, the new states are never rejected in the sense that the networks are returned to the state they had before the random move; rather the temperature is used in the selection between many such updated models.

1.3 Properties of the method

The properties and performance of the algorithm were evaluated using three test sets. The *Robot Arm* data set [22, 18] is an artificial regression problem, in which the outputs are simple trigonometric functions of the inputs, with a small Gaussian noise term added. The *Pima Indians Diabetes Database* [23] and *Myocardial Scintigram* data set [24] are medical classification problems with binary targets. The sizes of these data sets are specified in the Experiments section.

1.3.1 Correlation over time

It is important during all phases of the population learning that the ensemble of networks evolves in state space. Both in the beginning of the learning, in order to obtain well-tuned individual networks, and at the end where we want a diverse enough ensemble. To increase our confidence that the ensemble is evolving considerably, we find it suitable to measure that the population is uncorrelated with its state at a previous time.

To this end, we decided to study the average absolute activation (output) of the hidden units, $\langle |H| \rangle$. To our knowledge, this measure has not been studied by other authors. The average is formed both with respect to the hidden units and the data points in the training set. Averaging over the test set produces a very similar $\langle |H| \rangle$. Since the networks continually vanish and replicate, it seemed impractical to study the autocorrelation of individual networks; rather, we consider the average over all the networks in the population.

Our conjecture is that $\langle |H| \rangle$ will reflect the movement of our population in state space. A significantly fluctuating $\langle |H| \rangle$ indicates that the population is evolving rather than being stuck in a local minimum. However, the fluctuation of $\langle |H| \rangle$ cannot be used as a rule of thumb to indicate that the population has reached the desired distribution; even when we are training with less than optimal parameters, experiencing over-training etc, $\langle |H| \rangle$ still is usually fluctuating.

The qualitative behavior of $\langle|H|\rangle$ depends on the data set studied, and to some extent on the parameters chosen for training. $\langle|H|\rangle$ generally starts at ≈ 0.5 and then rises initially. When there is regularization ($\lambda \neq 0$), as for the Pima (Figure 1.1b) and Scintigram (Figure 1.1c) data sets, $\langle|H|\rangle$ then decreases considerably.

When the annealing is finished, the algorithm is continued, to allow for a detailed exploration. At this stage, with T and ϵ being low, the population cannot be expected to cross high energy barriers or travel long distances in state space. However, when training with the Pima and Scintigram data sets, the population is still mobile enough that $\langle|H|\rangle$ fluctuates significantly – Figure 1.1b and 1.1c. Interestingly, this is true even though the energy for the Pima data set is no longer changing in this phase (cf. Figure 1.4b).

Only for the Robot Arm, where the final values of T and ϵ are exceptionally low, are the fluctuations of $\langle|H|\rangle$ very small in the post-annealing phase (Figure 1.1a). These extreme parameters are required to get a fine-tuned answer for this low-noise problem.

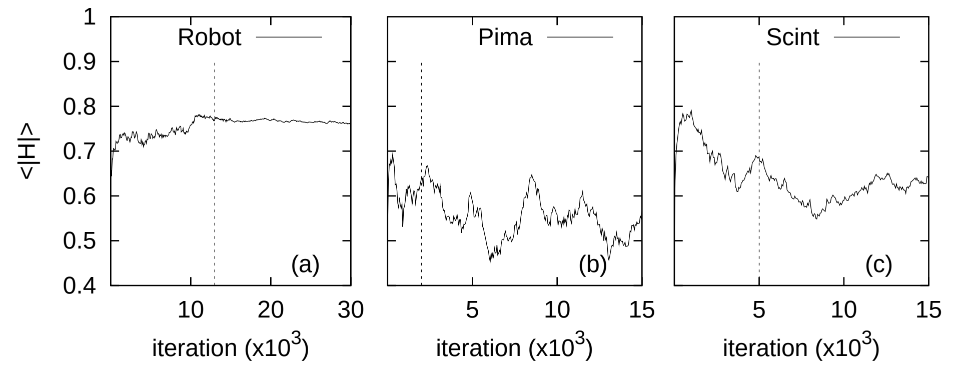

Figure 1.1: The average absolute activation of the hidden units. (a) Robot Arm, (b) Pima and (c) Scintigram data sets. The vertical dashed line represents the iteration where the annealing was finished. (In order to show the long-term behavior of the hidden activation, these plots show a larger number of iterations than what was actually used when measuring the performance.)

As a closing remark regarding $\langle|H|\rangle$, it was noted that our chosen method of starting from a relatively large ϵ and then lowering ϵ gradually makes $\langle|H|\rangle$ rise more quickly to its final region, compared to using a small value of ϵ throughout. (A large constant ϵ cannot be used, as the final fine-tuning requires a small value.) This indicates that a large step size in the beginning of the training indeed makes the population explore larger regions in state space and reach the desired regions more quickly, as was the intention.

Autocorrelation as a stopping criterion

The question arises of how to determine when (and if) the ensemble has converged to the desired distribution. A stopping criterion, expressed in terms of autocorrelations of relevant model quantities, would certainly be desirable. An example of such a criterion would be to sample for 100 times the integrated correlation length τ, defined as

$$\tau = 1 + 2\sum_{s=1}^{\infty} \rho(s) \tag{1.9}$$

Here, $\rho(s)$ is the autocorrelation of the studied quantity $\xi(t)$ at lag s:

$$\rho(s) = \frac{E\left[(\xi(t) - E[\xi])\left(\xi(t-s) - E[\xi]\right)\right]}{Var[\xi]}, \tag{1.10}$$

where t is the iteration number.

We made comparative runs with the Markov Chain Monte Carlo (MCMC) method implemented by Neal [18, 19], and drew the conclusion that for this algorithm as for our own, such a criterion is not reliable.

Neal's method uses Hamiltonian trajectories, discretized into a number of “leapfrog” steps, to sample from the Bayesian posterior distribution of network weights. These trajectories are alternated with Gibbs sampling of hyper-parameters controlling the prior weight distributions. In experiments using the Pima dataset, the smallest correlation lengths of hyper-parameters (measured in units of CPU minutes) were obtained for trajectories of 5 leapfrog steps. If the prescribed aim is to quickly sample for 100 times the correlation length, this would then be the ideal choice. But much longer trajectories are in fact needed to efficiently explore weight space; Neal uses trajectories in the order of 100–10000 steps. Thus we see that the proposed prescription is not reliable. Other authors [25] have performed other types of tests on Neal's method, and were also unable to establish convergence.

For the Population method, the correlation length of $\langle|H|\rangle$ has the same order of magnitude as the total number of iterations found to be appropriate by cross-validation. This being so, it seems impossible in this case also to make a prescription to sample for a certain number of correlation lengths.

In conclusion, given the absence of such a measure of convergence for both these methods, the required number of iterations has to be determined by trial and error and self-consistency checks using cross-validation.

1.3.2 Correlation between networks at a given time, and in the final committee

Diversity

It is important that the ensemble members disagree in their predictions, if anything is to be gained by combining them. A measure of the disagreement is the ensemble diversity $\bar{d}(p)$ on input p, defined as

$$\bar{d}(p) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}(p) - \langle y(p)\rangle\right)^2 \tag{1.11}$$

The diversity is just the variance of the individual network outputs. (For the moment we're considering the case of one output to keep notation simple, but the extension to many outputs is trivial.) Let $\bar{\epsilon}(p)$ denote the average of the individual network errors on input p, and $e_{err}(p)$ the squared error of the ensemble:

$$\bar{\epsilon}(p) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}(p) - t(p)\right)^2 \tag{1.12}$$

$$e_{err}(p) = \left(\langle y(p)\rangle - t(p)\right)^2 \tag{1.13}$$

Then, as demonstrated in [2],

$$e_{err}(p) = \bar{\epsilon}(p) - \bar{d}(p) \tag{1.14}$$

Letting E_{err}, $\bar{\mathcal{E}}$ and $\bar{D}$ be the averages of e_{err}, $\bar{\epsilon}$ and $\bar{d}$ over the input distribution, we obtain

$$E_{err} = \bar{\mathcal{E}} - \bar{D} \tag{1.15}$$

In words, the mean square ensemble error is equal to the average squared error of the individual networks minus the average diversity. Thus, to get a small ensemble error we want the diversity to be large and the individual errors to be small. There is a trade-off; modifying an algorithm to increase the diversity will reduce the ensemble error only if the increase in diversity is larger than the increase in the individual errors. The average diversity $\bar{D}$ is an interesting measure of where the operation of an algorithm is on this scale, and the diversities of the population method along with the diversities of the other studied methods are given in Table 1.1.

	Robot	Pima	Scintigram
Population method	0.00055	0.0012	0.0072
MCMC method	0.00049	0.0082	0.021
Bagging ensemble	—	0.0060	0.069
Bagging, unregularized	0.0011	0.13	0.093
Simple ensemble	0.0015	0.00071	0.00002

Table 1.1: The average diversity $\bar{D}$ of the ensembles produced by the studied methods.

Some algorithms use an explicit mechanism for increasing the diversity; e.g. Bagging trains each network on a different subset of the training set. The population method does not use any explicit method for increasing diversity, and produces ensembles that are less diverse than Bagging, as is evident from Table 1.1. However, there is not a one-to-one correspondence between diversity and good performance; even though the Bagging method produces more diverse ensembles than the population method for all three datasets, the predictive performance of the population method is higher (cf. section 1.4).

Internal correlation

Consider the vector of the activation of the hidden units, for a certain network and data point. We define the hidden activation distance between two networks to be the average Euclidian distance between the hidden activation vectors of these two networks, with the average being formed over the training set. This measure gives an indication of how different two network models are internally; cf. [26].

The hidden activation distance for the final population ensemble, for our choice of architecture and data sets, is typically in the range [0,0.2], whereas for independently trained and for randomly initialized networks it is typically in the range [0.5,3]. Thus, we see that all networks end up in the same approximate region of state space. One possible explanation is that, at some point in the annealing procedure, the group of networks in one region have happened to become significantly better adapted than in other regions, and the other groups vanish. Furthermore, even if the networks in each region are equally well adapted, the number of networks in any given group performs a random walk because of the randomness in the replication, and once a group reaches zero members it is gone.

What does this mean? Remember that there are numerous symmetries in neural networks – under permutation of the hidden unit indexes, and under sign change of all weights going into and out of one hidden unit. Because of this, there are many mutually distant regions in weight space that are symmetric copies of each other and implement essentially the same solution. Nothing is gained by including networks from many different symmetric regions; it is enough to explore one such region sufficiently. Thus, the fact that all models come from the same approximate region is not an essential disadvantage; they are still different enough that the outputs of the different models are significantly diverse (section 1.3.2), which is important for ensemble performance.

The internal distance within the simple and Bagging ensembles is of course large, because these individually trained and randomly initialized networks have equal probability of ending up in either of the symmetric regions. Even though the internal distance is large, however, the diversity for the simple ensemble is small (Table 1.1) and the predictive performance of the simple and Bagging ensembles is worse than for the Population method (section 1.4).

The internal distance between the networks in the MCMC ensemble is also large, because the Hamiltonian trajectories of Hybrid Monte Carlo travel long distances in state space. However, because many of the far-away regions are just symmetric copies of each other, visiting many of them is not really essential for good prediction. Doing so might still be an advantage, as an algorithm that only explores one symmetry-region runs the risk of exploring too small a part of that region, and moving between regions could protect against this problem. This is however at the cost of CPU-intensive calculations of the Hamiltonian equations of motion.

Neal rejects random updates in favor of hybrid Monte Carlo, arguing that random walk is an unsystematic and inefficient way of exploring state space [18]. This is indeed true if but a single network is used, but in the population algorithm the vanishing and replication of networks in effect moves poorly adapted networks long distances to the well-adapted ones where they continue their search, thus making the exploration systematic.

1.3.3 Age and Replication histograms

Each model has an age, defined as follows: in the beginning of the algorithm, all models have age 0. When a number of replicas is created from a model of age A, the first of these replicas has age $A + 1$ and the rest (if any) have age 1. Figure 1.2 shows a histogram of how old the models were when they vanished (i.e. were replicated zero times), during a window of iterations at different stages of the algorithm. Figure 1.3 correspondingly shows a histogram of how many replicas were created from each model.

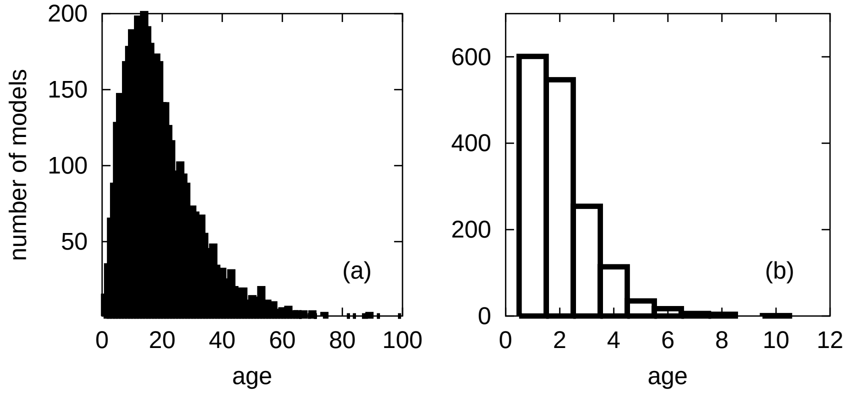

Figure 1.2: The age distribution of the models when they vanish, for the Robot Arm data set. (a) First 300 iterations. The competition is low and some models grow very old. (b) Last 10 iterations; the competition is strong.

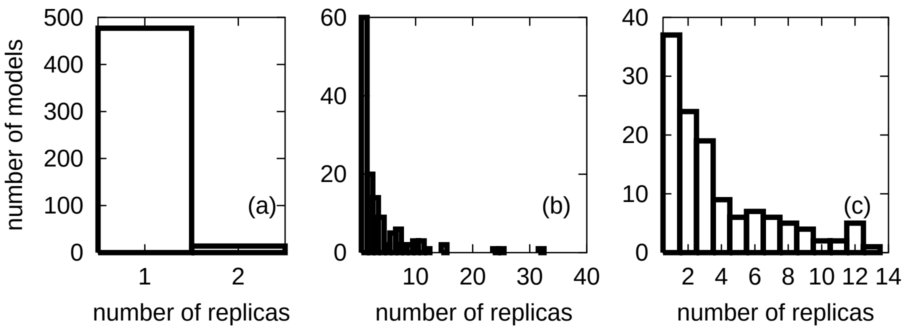

Figure 1.3: Replication histograms, showing the number of replicas produced by the replicating networks, for the Robot Arm data set. (a) Start of training. (b) End of annealing; the energy is still decreasing. (c) End of training.

At the beginning of the procedure, T is high. Our way of normalizing the targets (for regression) and the error means that the approximate magnitude of the initial energy is the same for all data sets, and the default value $T_i = 3.0$ is usually appropriate. A high T implies that all $r_i \approx 1$, and thus most models are replicated exactly once every iteration, as can be seen in Figure 1.3a. This in turn means that there are many models that reach a high age. The typical lifetime of a model is 10–30 as its population factor fluctuates around unity, and some models survive for as much as 100 iterations.

As T is lowered, the tail of the replication histogram grows longer and longer as the best models are more and more favored. When the annealing is almost finished, T is very low which means that the competition is strong. The energies of the models lie within a rather narrow band; a rather small change in energy is enough to give a model a tiny or huge population factor. The energy is still decreasing, i.e. often there is a model that finds a better configuration than what's been achieved before. This model is then replicated a very large number of times – Figure 1.3b.

After the annealing is finished, the energy eventually stops decreasing, and the population spreads out over the best regions found. The competition is still strong, but as there is now a fair number of networks that have reached very low and similar energies, no network is favored greatly over the others and the maximum number of replicas is moderate – Figure 1.3c. Remember that each individual model performs a random walk, and it might easily move to a configuration with significantly higher energy, after which the model is likely to vanish. Thus, we cannot expect models to become all that old. This can be seen in Figure 1.2b, which shows the ages during the last 10 iterations.

The final temperature T_f needed to archive good performance is problem dependent; there is a trade-off between forcing all the models to have very low energies by setting a low T_f, and increasing the diversity by setting T_f higher. As a rule of thumb we used $T_f = 10^{-3}$.

1.4 Experiments

To evaluate the performance of the Population algorithm we compared against the MCMC method by Neal [18, 19], Bagging [5] and a simple averaging ensemble, formed by training 100 MLPs on the full training set using backpropagation with random weight initialization. We also used 100 networks for the Bagging ensemble.

In preliminary runs, parameters were chosen using trial and error based on three-fold cross-validation on the training set. Reasonable effort was used in finding suitable parameters to minimize the error and CPU consumption, but the purpose here is only to give an indication of the merits of the methods, not to make a rigorous test of the general performance. The chosen parameters for the Population method are specified in Table 1.2. The parameters used for the other methods are not specified here, but the number of hidden units used for a

	Robot	Pima	Scintigram
Number of hidden units	10	8	8
Population size	500	100	100
Number of annealing iterations	13000	2000	5000
Total number of iterations	16000	2600	5500
Weight elimination parameter	0	5.0	20.0
Initial stepsize	0.1	0.5	0.5
Final stepsize	0.035	0.07	0.06
Initial temperature	0.08	3.0	3.0
Final temperature	0.00006	0.001	0.001

Table 1.2: Training parameters used for the Population method.

given data set was the same for all four methods. Other values for the number of hidden units were tested for the four methods, but this did not increase predictive performance. Furthermore, much larger ensemble sizes were tried for all methods, without any increase in performance. Apart from the Robot Arm data set that requires unusual parameters, the ones specified in Table 1.2 will also serve as an appropriate set of initial parameters for other problems. However, parameters such as the one for the weight elimination always requires tuning for each specific data set.

For the classification tasks, the performance is given in terms of area under the receiver operating characteristic (ROC) curve. The ROC curve displays diagnostic accuracy expressed in terms of sensitivity against 1 - specificity at all possible output threshold values. The sensitivity is the fraction of correctly classified "one" cases and the specificity is the corresponding fraction for the "zero" cases. The area under the ROC curve is a commonly used performance measure with unity (100%) being a perfect classifier and the value 0.5 being equivalent to random guessing. The total performance, i.e. the fraction of correctly classified examples in the test set, is also given at the threshold level that maximized the fraction of correctly classified examples during cross-validation. For the regression task, performance is given in terms of the mean square error (equation 1.1).

1.4.1 Test results

A final training was made using the whole training set, and the performance was computed using the test set. In each case, ten runs with identical parameters were started. The results were stable; the performances with standard

	Robot	**Pima**		**Scintigram**	
Problem type	Regression	Classification		Classification	
Number of inputs	2	7		30	
Number of targets	2	1		1	
Size of training set	200	200		153	
Size of test set	200	332		76	
	Mean Sq. Error	ROC area (%)	Total perf. (%)	ROC area (%)	Total perf. (%)
Population method	0.00308(11)	86.51(4)	80.2(2)	80.8(1.0)	75.5(1.3)
MCMC method	0.00279(2)	86.30(13)	80.1(3)	80.8(7)	75.1(1.2)
Bagging ensemble	—	86.18(7)	79.3(2)	78.4(7)	71.7(1.3)
Bagging, unreg.	0.00335(4)	83.52(16)	77.8(6)	77.2(4)	71.3(1.4)
Simple ensemble	0.00367(7)	86.11(4)	79.6(4)	75.5(1)	75.0(0)
	CPU min.	CPU minutes		CPU minutes	
Population method	435	19		93	
MCMC method	58	79		386	
Bagging ensemble	3.3	2.9		4.6	
Simple ensemble	3.8	2.8		3.2	

Table 1.3: Properties of the data sets (top); test results (middle), and CPU consumption (bottom). The standard deviations, corresponding to the one or two last digits of the test results, are given within parentheses.

deviations are as specified in Table 1.3. All runs were made on a 800 MHz Pentium III computer running Linux.

The Robot Arm is an artificial data set with very low noise. When starting the population updates from random configurations with high energies, the population often did not reach the lowest energies during the annealing. Therefore a very long initial back-propagation training was used, in order to start with a low energy. The initial temperature was also low; otherwise there would have been practically no competition between the networks as they are randomly updated, and the energy would have increased rapidly. The fine-tuning required of the networks for this problem necessitated an extremely low final temperature, which means that the population is relatively fixed after the annealing is finished, with the hidden activation $\langle |H| \rangle$ almost constant (Figure 1.1a). The energy also settles and fluctuates within a rather narrow interval; Figure 1.4a. — Even though these unusual parameters were used, the MCMC method achieves a better result in this case.

The Pima dataset is a real-world problem requiring less extreme parameters. As can be seen in Figure 1.4b, the training and test errors settle after the annealing and stay practically constant. This does not mean that the population is stuck in a local minimum, however; unlike for the Robot Arm, $\langle |H| \rangle$ is fluctuating (Figure 1.1b), indicating that the population is changing significantly. The performance for the Population and MCMC method is practically the same.

When cross-validating with the Scintigram dataset, there was often over-training; the validation error decreased initially but often subsequently increased; cf. Figure 1.4c. The validation curve could be made to level out by increasing the weight elimination parameter λ, but this resulted in worse predictive performance. This problem was solved by early stopping. An overall validation curve was created by averaging over the validation curves of several cross-validation runs with different dataset partitions; the best region of this curve was found to be the interval 1900–5500. In the final run, this was the interval over which the ensemble was sampled. Again, the performance of the Population and MCMC method is the same.

The tested ensemble methods used a non-zero regularization term for the two classification problems, as a result of the cross-validation process. As a reference we have included large unregularized ensembles (500 networks) created with Bagging. As can be seen in Table 1.3, the performance for the unregularized Bagging ensembles on the classification tasks were worse than for the regularized ones.

In all, the results for the Population method and Neal's Bayesian software were comparable for the real-world classification tasks, whereas Neal's software was better than the Population method on the artificial Robot Arm data set. The Population method was however faster than the MCMC method on the classification problems. Compared to the two other ensemble methods, Bagging and the simple ANN ensemble, the Population method was better, at least for the Robot and the Scintigram datasets. Bagging and the simple ANN ensemble were on the other hand the overall fastest methods.

1.5 Summary and Discussion

We have presented a new method for training an ensemble of neural networks. A population of networks is created and maintained such that probable networks replicates and less probable networks vanish. The Boltzmann distribution is used when defining the probability of a network. The "temperature" appearing in the Boltzmann distribution signifies the level of competition among

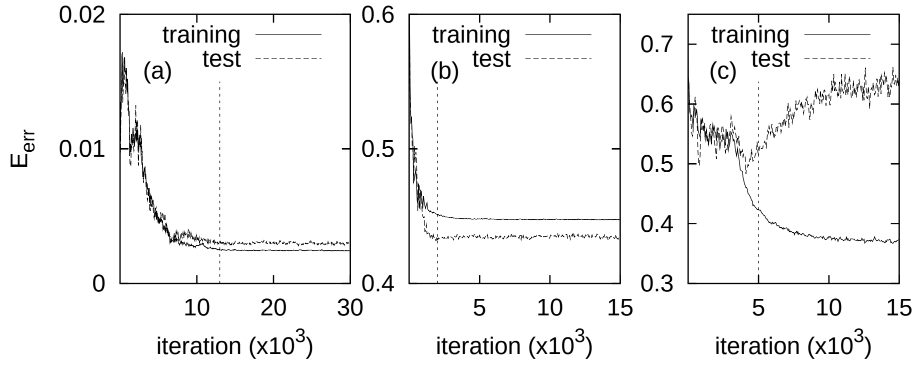

Figure 1.4: Training curves: (a) Robot Arm. (b) Pima. (c) Scintigram. These training curves show more iterations than were actually used when measuring performance, as in Figure 1.1.

the networks, where a high temperature allows for networks that have both small and large errors. A small temperature, on the other hand, selects only well adapted networks. The annealing present in the population method is used to transform an initial (random) population to a set of fine-tuned networks. This population then continues to evolve in state space. The whole procedure creates the necessary diversity among the individual members that is important for the ensemble itself.

The method is simple in the sense that is does not require any complicated algorithmic implementations. Furthermore, no gradient information, with respect to the error function, is used when updating individual networks, which makes the method fast even when maintaining a population of a hundred networks or more.

We tried using an explicit mechanism for increasing the diversity of the population, by introducing a term in the energy equal to the diversity $\bar{D}$ (equation 1.11) times a negative, tunable parameter. Such a term has been used by other authors when training new networks that are to be added to an existing ensemble [8], or when selecting the best networks trained by other means [7]. For our method, this term did not improve the predictive performance for any of the studied datasets. The only noticeable non-detrimental effect was on the Pima dataset, where moderate values of the tunable parameter resulted in doubled average diversity $\bar{D}$ at unchanged ROC area. In conclusion, nothing is gained in the population algorithm by trying to push the trade-off between diversity and individual network errors in this way.

To summarize, the algorithm has been tested on three datasets and the results from the experiments section indeed show that the population method can be used as an efficient neural network learning algorithm.

1.6 Acknowledgments

We thank Carsten Peterson for fruitful discussions. This work was in part supported by the Swedish Natural Science Research Council and the Swedish Foundation for Strategic Research.

References

[1] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 993–1001, 1990.

[2] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 2. San Mateo, CA: Morgan Kaufman, 1995, pp. 650–659.

[3] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[4] A. J. C. Sharkey, "On combining artificial neural nets," *Connection Science*, vol. 8, pp. 299–314, 1996.

[5] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[6] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996, pp. 148–156.

[7] D. W. Opitz and J. W. Shavlik, "Actively searching for an effective neural-network ensemble," *Connection Science*, vol. 8, pp. 337–353, 1996.

[8] Y. Liu, X. Yao, and T. Higuchi, "Evolutionary ensembles with negative correlation learning," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 295–304, 2000.

[9] B. E. Rosen, "Ensemble learning using decorrelated neural networks," *Connection Science*, vol. 8, pp. 373–384, 1996.

[10] A. Sharkey, Ed., *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems.* London: Springer-Verlag, 1999.

[11] V. Tresp, "Committee machines," in *Handbook for Neural Network Signal Processing*, Y. H. Hu and J.-N. Hwang, Eds. CRC Press, 2001.

[12] Z.-H. Zhou, Y. Jiang, Y.-B. Yang, and S.-F. Chen, "Lung cancer cell identification based on artificial neural network ensembles," *Artificial Intelligence in Medicine*, vol. 24, no. 1, pp. 25–36, 2002.

[13] A. J. C. Sharkey, N. E. Sharkey, and S. S. Cross, "Adapting an ensemble approach for the diagnosis of breast cancer," in *Proceedings of ICANN 98.* Springer-Verlag, 1998, pp. 281–286.

[14] J. Khan, J. Wei, M. Ringnér, L. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. Atonescu, C. Peterson, and P. Meltzer, "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks," *Nature Medicine*, vol. 7, pp. 673–679, 2001.

[15] M. Ohlsson, H. Öhlin, S. Wallerstedt, and L. Edenbrandt, "Usefulness of serial electrocardiograms for diagnosis of acute myocardial infarction," *The American Journal of Cardiology*, vol. 88, pp. 478–481, 2001.

[16] S.-E. Olsson, H. Öhlin, M. Ohlsson, and L. Edenbrandt, "Neural networks - a diagnostic tool in acute myocardial infarction with concomitant left bundle branch block," *Clinical Physiology and Functional Imaging*, vol. 22, pp. 295–299, 2002.

[17] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.

[18] R. M. Neal, *Bayesian Learning for Neural Networks.* New York: Springer-Verlag, 1996.

[19] R. M. Neal, "Software for flexible Bayesian modeling," 1999, http://www.cs.toronto.edu/˜radford/fbm.1999-03-13.doc/index.html.

[20] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back–propagation," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 177–185.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[22] D. J. C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[23] B. D. Ripley, *Pattern Recognition and Neural Networks.* Cambridge, Great Britain: Cambridge University Press, 1996.

[24] H. Haraldsson, M. Ohlsson, and L. Edenbrandt, "Value of exercise data for the interpretation of myocardial perfusion SPECT," *Journal of Nuclear Cardiology*, vol. 9, pp. 169–173, 2002.

[25] F. Vivarelli and C. K. I. Williams, "Comparing Bayesian neural network algorithms for classifying segmented outdoor images," *Neural Networks*, vol. 14, no. 4–5, pp. 427–437, 2001.

[26] R. Caruana, "Case-based explanation for artificial neural nets," in *Artificial Neural Networks in Medicine and Biology*, H. Malmgren, M. Borga, and L. Niklasson, Eds. London: Springer-Verlag, 2000, pp. 303–308.

Value of Exercise Data for the Interpretation of Myocardial Perfusion SPECT

Paper II

Value of Exercise Data for the Interpretation of Myocardial Perfusion SPECT

Henrik Haraldsson, Mattias Ohlsson

Complex Systems Division, Department of Theoretical Physics
University of Lund, Sölvegatan 14A, SE-223 62 Lund, Sweden
`http://www.thep.lu.se/complex/`

Lars Edenbrandt

Department of Clinical Physiology, Lund University Hospital
SE-221 85 Lund, Sweden
`http://www.weaidu.com/`

Journal of Nuclear Cardiology **9**, 169–173 (2002)

Background

Artificial neural networks have successfully been applied for automated interpretation of myocardial perfusion images. So far the networks have used data from the myocardial perfusion images only. The purpose of this study was to investigate whether the automated interpretation of myocardial perfusion images using artificial neural networks was improved if clinical data was assessed in addition to the perfusion images.

Methods and Results

A population of 229 patients who had undergone both a rest-stress myocardial perfusion scintigraphy in conjunction with an exercise test and coronary angiography, with no more than three months elapsing between the two examinations, were studied.

The networks were trained to detect coronary artery disease or myocardial ischemia using two different gold standards. The first was based on coronary angiography and the second was based on all data available, including perfusion scintigrams, coronary angiography, exercise test, resting ECG, patient history etc.

The performance of the neural networks was quantified as areas under the receiver operating characteristic curves.

The results show that the neural networks trained with perfusion images performed better than that trained with exercise data (0.78 vs. 0.55, $p<0.0001$), using coronary angiography as gold standard. Furthermore, the networks did not improve when data from the exercise test was used as input in addition to the perfusion images (0.78 vs. 0.77, $p=0.6$).

Conclusions

The results show that the clinically important information in a combined exercise test and myocardial scintigraphy could be found in the perfusion images. Exercise test information did not improve upon the accuracy of automated neural network interpretation of myocardial perfusion images in a receiver operator characteristic analysis of test accuracy.

2.1 Introduction

Artificial neural networks have been used to interpret myocardial perfusion images in several studies [1, 2, 3, 4]. The best of these networks has shown to perform as well as or even better than human experts. Despite their high performance, this type of neural networks will not take over the decision-making process from the physician in clinical practice. Rather, the computer can assist the physician by proposing an interpretation of the scintigram. A recent study showed that physicians benefit from the advice of neural networks both in terms of an improved performance and a decreased intra- and inter-observer variability [5]. It has also been shown that this type of neural networks can maintain a high accuracy in a hospital separate from that in which they were developed [6]. Based on the results of these studies and on modern information technology this type of neural networks may soon become widely available via the Internet [7].

The performance of the above mentioned neural networks has shown to be high but there may still be room for improvement. The purpose of this study was to investigate whether the automated interpretation of myocardial perfusion images using artificial neural networks was improved if clinical data was assessed in addition to the perfusion images. The clinical data was objective and extracted from the exercise test performed in conjunction with the perfusion imaging.

2.2 Materials and methods

2.2.1 Overall study design

Bayesian neural networks were trained to classify myocardial perfusion images and clinical data from the exercise test. Two gold standards were used; one based on a clinical evaluation of all data available, i.e. myocardial perfusion images, coronary angiography, exercise test, resting ECG, patient history etc, and the other based on the result from coronary angiography only. The area under the receiver operating characteristics (ROC) curve was used as measure of neural network performance, in order to compare the importance of different sets of inputs to the neural networks.

2.2.2 Patient population

The study was based on patients from Lund University Hospital who, during the period from November 1992 to June 1997 had undergone both a rest-stress myocardial perfusion scintigraphy and coronary angiography, with no more than three months elapsing between the two examinations. Furthermore, for all patients the exercise test was performed on a bicycle ergometer. From the total material, patients were excluded because they had undergone angioplasty or coronary artery bypass surgery or had signs of progressive coronary artery disease (CAD) between the scintigraphy and angiography. Patients with pacemaker were also excluded form this study. The final material consisted of 229 patients (161 males and 68 females, age mean 57.7, range 21-82 years).

2.2.3 Myocardial scintigraphy

In 124 patients the rest and stress studies were performed in a one-day $^{99\mathrm{m}}$Tc-sestamibi protocol, using 300 MBq at rest, and after a delay of about three hours, 900 MBq at stress. The remaining 105 patients used a one-day $^{99\mathrm{m}}$Tc-tetrofosmin protocol, with 350 MBq at rest and 900 MBq at stress. For both protocols, the time period between injection and imaging was one to two hours for the rest studies and 30 minutes to one hour for the stress studies.

The scintigraphy data were acquired in continuous single photon emission computed tomography (SPECT) over 180 degrees during 20-30 minutes. The first protocol ($^{99\mathrm{m}}$Tc-sestamibi) used a Toshiba gamma camera (GCA-901A/SA) and for the $^{99\mathrm{m}}$Tc-tetrofosmin protocol, data were acquired using an ADAC gamma camera (Vertex Classic). The data processing technique has been described in more detail in a previous study [4]. The three-dimensional alignment of the rest and stress images was performed on an interactive basis by experienced clinical operators. A set of short-axis slices were used as input to the bull's-eye program which then generated the bull's-eye images.

2.2.4 Exercise test

All patients underwent exercise on a bicycle ergometer. Exercise was symptom-limited (anginal pain, severe dyspnea or severe fatigue) unless malignant arrhythmia or exercise hypotension occurred ($>$ 10 mmHg drop between exercise stages). Workload was increased in a stepwise manner by 10 W/min. A standard 12-lead ECG was recorded at rest and during the exercise test. Measurements from the ECG consisted of the ST60 amplitude from lead V4-V6. ST60 denotes the point 60 milliseconds after the ST-J point. The maximal heart rate and the workload at the end of the exercise and were denoted *maximal* heart rate and *final* workload. The work load at the end of the exercise was on average 86% of predicted (range 37-160). One-hundred seventeen of the 229 patients reached a maximal heart rate $> 85\%$ of predicted [8]. Among the remaining 112 patients, 66 were treated with beta blockers. Severe anginal pain or exercise hypotension limited the exercise in 17 patients with low maximal heart rate and without beta blockers.

2.2.5 Coronary angiography

The patients were examined using the standard Judkins technique. Angiograms were performed and interpreted by experienced cardiac radiologists. Each coronary artery was examined in 4-6 projections, of which at least two were orthogonal. Significant CAD was defined as luminal cross area reduction of more than 75%, in a major coronary artery. The severity of a coronary stenosis was determined by visual assessment. In 72 patients CAD was found in both the LAD and the RCA/LCX territories and in 84 patients CAD was found in one territory only (36 LAD and 48 RCA/LCX). Coronary angiography did not reveal significant CAD in the remaining 73 patients.

2.2.6 Gold standard

Two different gold standards were used in this project, one based on the independent method coronary angiography only and the other based on a comprehensive clinical evaluation. Coronary angiography was used to classify the patients as having CAD or not. CAD was found in 156 of the 229 patients. The clinical evaluation was used to classify the patients as having myocardial ischemia or not. The evaluation was made by an experienced physician who retrospectively studied all patient cases. Both the data and the clinical reports from the myocardial perfusion test, the exercise test and the coronary angiography were available to the physician as was information regarding age, gender and history presented in the referral.

2.2.7 Bayesian neural networks

In this study we have used Bayesian inference techniques to neural networks learning. Bayesian training of neural networks consists of picking a number of independent neural networks, each defined by its set of weights $\mathbf{w}$, from a probability distribution. This distribution $p(\mathbf{w}|D)$, which is a function of the weights $\mathbf{w}$, represents the degree of belief for the different neural networks using the observed patient data D. The *posterior* $p(\mathbf{w}|D)$ is constructed using Bayes' theorem. To make a prediction for a test case one has to, theoretically, *integrate* over a multi-dimensional weight space. Practically, one approximates the integration with a finite sum, i.e. the average network output $\langle y(\mathbf{x}_{\text{test}})\rangle$ for a test case $\mathbf{x}_{\text{test}}$, is computed as

$$\langle y(\mathbf{x}_{\text{test}})\rangle = \frac{1}{L}\sum_{i=1}^{L} y(\mathbf{w}_i, \mathbf{x}_{\text{test}}).$$

$y(\mathbf{w}_i, \mathbf{x}_{\text{test}})$ is the output of the network defined by the set of weights $\mathbf{w}_i$ that is drawn from the probability distribution $p(\mathbf{w}|D)$ using Markov Chain Monte Carlo methods [9]. All calculations in this study were done using the software package of Neal [10, 11].

In order to obtain the generalization performance, the full patient material was divided into three equally sized parts. Each of these parts was once used as test set and the remaining two parts were used as training set, for the Bayesian neural network learning, in a 3-fold cross-validation scheme.

The neural networks consisted of one input layer, one hidden layer and one output layer. The number of input nodes ranged from 11 to 41 depending on the set of input variables used. The hidden layer contained 20 nodes and the output layer consisted of one node that encoded whether the patient suffered from CAD/ischemia or not.

2.2.8 Neural network input data

Myocardial perfusion images and data from the exercise test, in different combinations, were used as inputs to the neural networks. The myocardial perfusion images were pre-processed in order to decrease the number of variables and to extract relevant features from the images. This pre-processing was accomplished by a two-dimensional Fourier transform that used both rest and stress images as input [4]. After the transform a selection of 30 values, constituting the real and the imaginary part of the coefficients of the 15 lowest frequencies, were used to represent the two rest and stress images.

From the exercise test, the obtained measurements “final workload” and “maximal heart rate” were used to calculate, for each patient, values of predicted final workload [12] and predicted maximal heart rate [8]. Since the heart rate is affected when the patient is using β-blockers, a zero-one variable indicating the presence of such β-blockers was also used as an input variable to the neural networks. From the ECG three measurements were used, the ST60 amplitude from lead V4-V6, both at rest and at final workload. Table 2.2 shows the input variables for the different sets of input data.

Six different combinations of the sets of input data were used, ranging from 11 variables consisting of exercise measurements, to the full set of 41 variables, that included all exercise data and the perfusion images. Table 2.1 and 2.3 show the different combinations used in this study.

Table 2.1: The result for the Bayesian neural networks using perfusion images and/or all of the exercise data as inputs. The values in parenthesis are the p-values for the difference as compared to the ROC area with only the perfusion images.

Input data set	ROC area (%)	
	CAD	Ischemia
Perfusion images	78.2	88.4
Exercise data	54.9 (< 0.0001)	55.3 (< 0.0001)
Perfusion Images + exercise data	77.4 (0.6)	88.6 (0.8)

2.2.9 Statistical methods

For each combination of inputs to the neural networks an ROC curve was constructed. The difference in performance between the different input sets was measured as the difference in area under the ROC curves. The significance of this area difference was calculated using a permutation test (see e.g. [13]). The test was performed by repeatedly and randomly permuting the test cases in the two classification lists. For each permutation the difference of the two resulting areas were calculated (test statistic). The evidence against the null hypothesis, of no difference between the two original ROC areas, was given by the fraction of area differences of the test statistic larger than the actual difference.

2.3 Results

The results are shown in Table 2.1 and 2.3, which presents the areas under the ROC curves using different sets of inputs to the neural networks. With only the perfusion images as input to the neural networks an area of 0.78 was obtained for the classification of CAD and the corresponding number for ischemia was 0.88. If only the exercise data was used ROC areas of 0.55 was found for both the classification of CAD and ischemia. The difference between these two sets of ROC areas was statistically significant (see Table 2.1).

As can be seen in Table 2.1 and 2.3, using both perfusion images and exercise data did not improve the classification ability of the neural networks significantly. The actual ROC curves corresponding to the CAD results in Table 2.1 are shown in Figure 2.1.

Table 2.2: Description of the input variables for the different sets of input data.

Set of input data	Input variables
Myocardial perfusion images (30)	Fourier frequencies (15 complex numbers)
Exercise test (11)	
Heart rate	maximal heart rate, % of predicted maximal heart rate and β-blockers
Workload	final workload, % of predicted final workload
ST60 amplitudes	ST60 amplitude from leads V4-V6 at both rest and at final workload

A sub-analysis were men and women were studied separately showed the same pattern in both groups, i.e. networks trained with perfusion images only and those trained with perfusion images in combination with exercise data showed the same performance whereas the networks trained with exercise data only showed a worse performance.

Table 2.3: The result for the Bayesian neural networks using perfusion images and/or part of the exercise data as inputs. The values in parenthesis are the p-values for the difference as compared to the ROC area with only the perfusion images.

Input data set	ROC area (%)	
	CAD	Ischemia
Perfusion images	78.2	88.4
Images + ST60 amplitudes	79.4 (0.4)	88.5 (0.9)
Images + heart rate	76.8 (0.4)	88.3 (0.9)
Images + workload	77.0 (0.3)	87.6 (0.5)

2.4 Discussion

The results of this study show that the neural networks fed with perfusion images performed better than those fed with exercise data. Furthermore the neural networks did not improve when data from the exercise test was used as input in addition to the perfusion images. The clinically important information could be found in the perfusion images. This conclusion can be drawn independently of the type of gold standard used. The coronary angiography has the advantage that it is an independent method and it is widely used as gold standard in this type of studies. The disadvantage is that the angiogram shows different phenomena compared to perfusion images and ECG. A stenosis in a coronary artery does not always correlate with a reduction in myocardial perfusion or a ST depression. This problem is well known but since the ideal gold standard is missing, coronary angiography is still the most widely used gold standard. The clinical evaluation used as gold standard has the advantage that this is more close to the clinical reality. The disadvantage with clinical evaluation as gold standard in this study is of course that the results depend on the opinion of the physician who made the classification. A physician who relies heavily on the result of the exercise test would in some cases probably come to another conclusion than a colleague who relies more on the perfusion images. This is a problem which is difficult to tackle. How much a physician relies on different facts when making a clinical classification is probably impossible to describe in detail. Still, even though there are different types of problems with the two gold standard methods, the results point in the same direction.

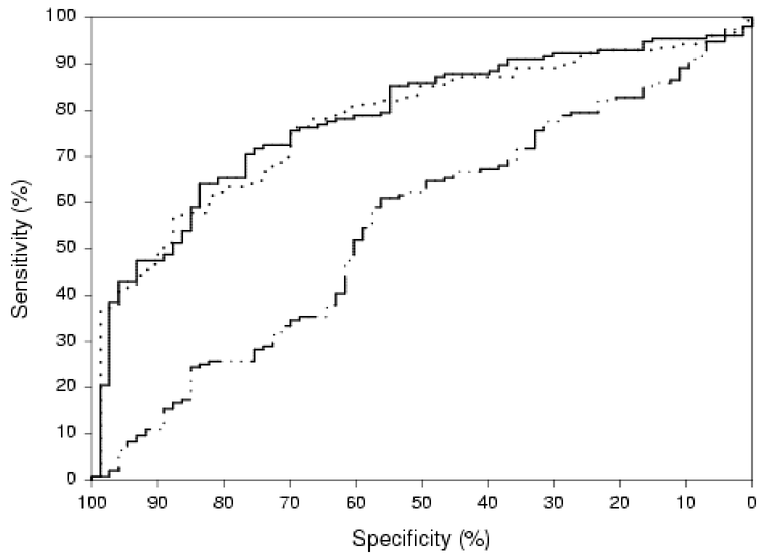

Figure 2.1: ROC curves for the neural network classification of CAD using 3 sets of inputs. Perfusion images (solid line), perfusion images + exercise data (dotted line) and exercise data only (dash dotted line).

2.4.1 Study limitations

This study was based on patients who had undergone both myocardial perfusion imaging and coronary angiography. The study population was a subgroup of patients admitted to myocardial perfusion imaging. If the conclusions from this study could be extrapolated to all these patients remains to be proven. Further studies in other low and high risk populations need to be carried out.

2.4.2 Conclusion

The results show that the clinically important information in a combined exercise test and myocardial scintigraphy could be found in the perfusion images. It was not possible to improve the automated interpretation of myocardial perfusion images by adding data from the exercise test.

2.5 Acknowledgments

This study was supported by grants from the Swedish Foundation for Strategic Research, the Swedish Medical Research Council (K2001-27X-09893-10B), the Medical faculty at Lund University, and the County of Skåne.

References

[1] Fujita H, Katafuchi T, Uehara T, and Nishimura T. Application of artificial neural network to computer-aided diagnosis of coronary artery disease in myocardial SPECT bull's-eye images. J Nucl Med 1992;33:272–276.

[2] Porenta G, Dorffner G, Kundrat S, Petta P, Duit-Schedlmayer J, and Sochor H. Automated interpretation of planar thallium-201-dipyridamole stress-redistribution scintigrams using artificial neural networks. J Nucl Med 1994;35:2041–2047.

[3] Hamilton D, Riley PJ, Miola UJ, and Amro AA. A feed forward network for classification of bull's-eye myocardial perfusion images. Eur J Nucl Med 1995;22:108–115.

[4] Lindahl D, Palmer J, Ohlsson M, Peterson C, Lundin A, and Edenbrandt L. Automated interpretation of spect myocardial perfusion images using artificial neural networks. J Nucl Med 1997;38:1870–1875.

[5] Lindahl D, Lanke J, Lundin A, Palmer J, and Edenbrandt L. Improved classifications of myocardial bulls-eye scintigrams with computer-based decision support system. J Nucl Med 1999;40:96–101.

[6] Lindahl D, Toft J, Hesse B, Palmer J, Ali S, Lundin A, and Edenbrandt L. Scandinavian test of artificial neural network for classification of myocardial perfusion images. Clin Physiol 2000;20:253–261.

[7] Järund A, Edenbrandt L, Ohlsson M, and Borälv E. Internet based artificial neural networks for the interpretation of medical images. In Malmgren H, Borga M, and Niklasson L, editors, Artificial Neural Networks in Medicine and Biology. Springer; 2000. p.87–92.

[8] Åstrand I. Aerobic work capacity in men and women with special reference to age. Acta Physiol. Scand. 1960;49.

[9] Neal RM. Bayesian training of backpropagation networks by the hybrid Monte Carlo method. Technical Report CRG-TR-92-1, Department of Computer Science, University of Toronto, Canada, 1992.

[10] Neal RM. Software for flexible Bayesian modeling and Markov chain sampling. http://www.cs.toronto.edu/˜radford/software.html.

[11] Neal RM. Bayesian learning for neural networks. PhD thesis, Dept. of Computer Science, Univ. of Toronto, 1995.

[12] Nordenfelt I, Adolfsson L, Nilsson JE, and Olsson S. Reference values for exercise tests with continuous increase in load. Clin. Physiol. 1985;5(2):161–172.

[13] Wehrens R, Putter H and Buydens LMC. The bootstrap: A tutorial. Chemom. Intell. Lab. Syst. 2000;54:35-52.

Detecting Acute Myocardial Infarction in the 12-lead ECG using Hermite Expansions and Neural Networks

Paper III

Detecting Acute Myocardial Infarction in the 12-lead ECG using Hermite Expansions and Neural Networks

Henrik Haraldsson[1], Lars Edenbrandt[2] and Mattias Ohlsson[1]

1) Complex Systems Division, Department of Theoretical Physics
University of Lund, Sölvegatan 14A, SE-223 62 Lund, Sweden
`http://www.thep.lu.se/complex/`

2) Department of Clinical Physiology, Lund University Hospital
S-221 85 Lund, Sweden
`http://www.weaidu.com/`

LU TP 03-16, submitted to *Artificial Intelligence in Medicine*

We use Artificial Neural Networks (ANNs) to detect signs of acute myocardial infarction in ECGs. The 12-lead ECG is decomposed into Hermite basis functions, and the resulting coefficients are used as inputs to the ANNs. Furthermore, we present a case-based method that qualitatively explains the operation of the ANNs, by showing regions of each ECG critical for ANN response. Key ingredients in this method are: (i) a cost function used to find local ECG perturbations leading to the largest possible change in ANN output, and (ii) a minimization scheme for this cost function using mean field annealing. Our approach was tested on 2238 ECGs recorded at an emergency department. The obtained area under the receiver operating characteristic curve for ANNs trained with the Hermite representation and standard ECG measurements was 83.4% and 84.3% ($p = 0.4$), respectively. We believe that the proposed method has potential as a decision support system that can provide a good advice for diagnosis, as well as providing the physician with insight into the reason underlying the advice.

3.1 Introduction

Early diagnosis of acute myocardial infarction (AMI) is of vital importance for patients attending the emergency department with chest pain, as there are large benefits for immediate treatment of AMI patients. The 12-lead ECG, together with patient history and biochemical markers, are usually used at the emergency department to diagnose AMI. This diagnosis can be difficult, and for short durations of symptoms the biochemical markers may not show any signs at all. For an early diagnosis of AMI one may therefore have to rely on the 12-lead ECG together with patient history. The 12-lead ECG has the advantage of always being available, but the interpretation can be difficult. Computer-based ECG interpretations for the detection of AMI are therefore of importance as they can improve the early diagnosis of AMI.

Artificial neural networks (ANNs) is a computer paradigm that has turned out to be an efficient method for pattern recognition tasks. It is therefore a very promising new technology for healthcare in general [1] and especially in connection with the diagnosis of AMI. There are several studies where ANNs have been used as a tool for diagnosing AMI. Among the first ones were Baxt [2] and Harrison et al. [3] that used patient history together with ECG data and clinical findings as inputs to the ANNs. A drawback with the ANN of these studies is that some of the input data may be unavailable at the time of initial patient presentation. Later studies [4, 5] have developed ANNs with the aim of making a decision support tool that can be used at the emergency department. Using only 12-lead ECG as inputs to the ANNs, Hedén et al. [6] were able to detect AMI with a performance equal to or even better than an experienced cardiologist. Another study [7] showed the advantage with serial ECG analysis when diagnosing AMI, again only using the 12-lead ECG as input to the ANN. The advantage of using only the 12-lead ECG is the immediate availability and the possibility of an automated interpretation using computer software bundled with the ECG recorder. An additional advantage will certainly be obtained if the ECG interpretation method can explain the reasoning behind its findings, thereby increasing the support for the physician diagnosing the patient.

Case-based sensitivity analysis of ANNs can be defined as a way of finding the most influential inputs for a given test case and a trained ANN. This differs from the objective of overall sensitivity analysis which aims at finding the most important inputs for a given problem and its corresponding data set. A method following the latter approach was developed by Baxt et al. [8] in connection with the diagnosis of AMI. Their method used a bootstrap technique to estimate input variable effects when diagnosing AMI, which resulted in an impact factor for each input variable. These results may differ from a case-based analysis that aims at finding important inputs for a given test case. The advantage of a case-based approach is that we can indirectly explain how the ANN derived its output. A related approach was developed in [9] that aimed at explaining the reasoning behind the ANN, on a case-based level, by finding similar cases. Similarity was measured using the hidden activation of the ANN.

In this paper we have used Bayesian ANNs as a tool for detecting AMI patients using the 12-lead ECG. Furthermore, to explain the reasoning behind the ANN output, a method was developed that aims at showing temporal regions of the ECG important for this particular case. The key ingredients of our approach are: a representation of the ECG using Hermite functions, and a combinatorial optimization problem formulation for finding important ANN inputs.

Each lead of the ECG is expressed as a series of Hermite polynomials [10, 11], and the coefficients in this series are used as inputs for the ANN. After the training, a small number of the coefficients for a given ECG are perturbed within a limited interval; these perturbations are selected so as to maximize the corresponding change in diagnosis (ANN output). A perturbed ECG is reconstructed from the perturbed coefficients, and by comparing this perturbed ECG to the original one some insight can be gained about the reasoning behind this particular ANN output. The possibility of constructing perturbed ECGs from perturbed ANN inputs is one of the virtues of Hermite decomposition; no such reconstruction is possible in the traditional approach where the feature extraction consists of measuring amplitudes and slopes.

The purposes of this study were to:

- Train ANNs to detect AMI using ECGs represented by Hermite expansion coefficients.
- Use a case-based sensitivity analysis to indicate what temporal regions of an ECG are most important for the ANN in calculating its output.

3.2 Materials and Methods

3.2.1 Study Population

This study was based on ECGs recorded at the emergency department of the University Hospital in Lund, Sweden from July 1990 through June 1997. A total of 1119 suitable ECGs were found to originate from patients that were admitted to the coronary care unit and discharged with the diagnosis "acute myocardial infarction". This group of ECGs was defined as the **AMI** group. A random selection of 1119 ECGs among the remaining ones was defined as the **control** group. This group contained ECGs from patients with no diagnosis of "acute myocardial infarction". Patients with pacemaker and ECGs with severe technical deficiencies were excluded from the study.

This study population was also studied in Hedén et.al. [6] where a detailed definition of the criteria for diagnosing acute myocardial infarction can be found.

The acute infarction group consisted of 699 ECGs recorded on men and 420 recorded on women. The mean age for this group was 70.8 ± 12.4. The corresponding numbers for the control group was 578 men and 541 women with a mean age of 63.8 ± 19.0.

The total population of 2238 ECGs was further randomly divided into one training set and one test set of 1499 and 739 ECGs, respectively. The performance of the artificial neural networks classifiers presented in this study is based on the test set.

3.2.2 Electrocardiography

The 12-lead ECGs were recorded by the use of computerized electrocardiographs (Siemens-Elema AB, Solna, Sweden). The following 6 standard measurements, taken from each of the 12 leads, were selected for further analysis using neural networks: ST-J amplitude, ST slope, ST amplitude 2/8, ST amplitude 3/8, positive T amplitude and negative T amplitude. The ST amplitude 2/8 and ST amplitude 3/8 were obtained by dividing the interval between ST-J point and the end of the T wave into 8 parts of equal duration. The amplitudes at the end of the second and the third intervals were denoted ST amplitude 2/8 and ST amplitude 3/8. These measurements were obtained from the computerized ECG recorders using their measurements program.

To facilitate for later modeling of the ECGs using Hermite functions the signals were resampled from the original 500Hz sampling rate to 1kHz. Part of the analysis includes using software for QRS detection [12].

3.2.3 Hermite Decomposition of ECGs

In this section we describe the decomposition of the ECGs using Hermite functions. This method was first used by Sörnmo et al. [10] for evaluation of QRS shape features. It has also been used as a method for ECG data compression [11] and as a preprocessing tool for clustering ECG complexes with

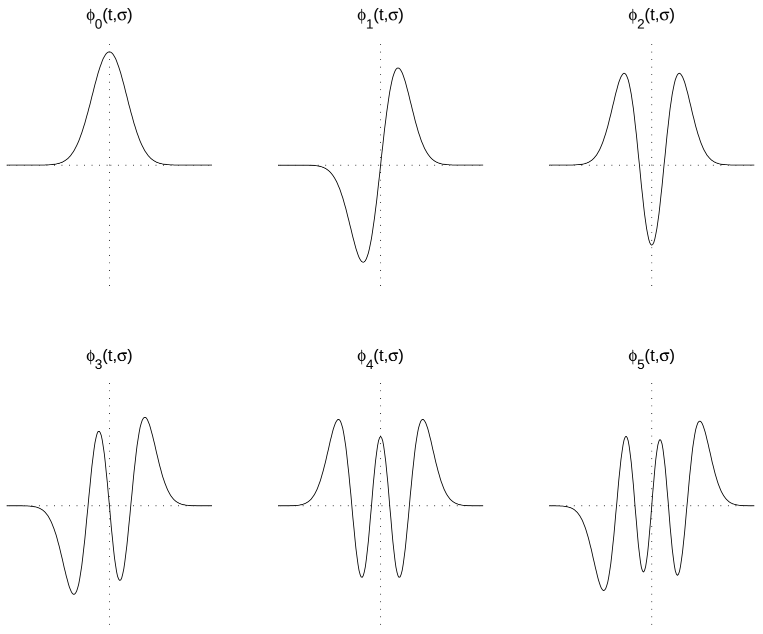

Figure 3.1: The first six Hermite basis functions (Eqs. 3.1, 3.2) plotted as a functions of t, for a fixed value of the width σ. The same scale is used in all figures.

self-organizing maps [13]. Following these ideas we define two windows in each beat, the QRS complex and the ST segment together with the T wave (ST-T complex). For the current application of detecting acute myocardial infarction, the P wave is not important and was consequently not part of the Hermite modeling. To prepare each window for the Hermite expansion a linear trend subtraction was used such that the start and end of each complex was zero. The subtracted amplitude at the start of the ST-T complex was part of the parameter set used to characterize the beats. Furthermore, each window was centered, in time, around the largest amplitude.

Hermite basis functions (Fig. 3.1) have the property that an arbitrary signal which is bounded in time can be approximated by a unique sum of such functions. The error in this approximation can be made arbitrarily small by increasing the number of basis functions used in the expansion.

The Hermite basis functions $\phi_n(t,\sigma)$ are given by the following expression:

$$\phi_n(t,\sigma) = \frac{1}{\sqrt{\sigma 2^n n! \sqrt{\pi}}} e^{-t^2/2\sigma^2} H_n(t/\sigma) \tag{3.1}$$

where the width σ approximates the half-power duration. $H_n(t/\sigma)$ are the Hermite polynomials, given recursively by $H_0(x) = 1$, $H_1(x) = 2x$, and

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x). \tag{3.2}$$

The QRS and ST-T complexes, generically denoted as $x(t)$, can be expressed as a sum of these functions:

$$x(t) = \sum_{n=0}^{N-1} c_n(\sigma)\phi_n(t,\sigma) + e(t,\sigma) \tag{3.3}$$

where the error $e(t,\sigma) \to 0$ as $N \to \infty$. The uniqueness of this expansion is guaranteed by the orthonormality:

$$\sum_{t=-\infty}^{\infty} \phi_n(t,\sigma)\phi_m(t,\sigma) = \delta_{mn} \tag{3.4}$$

This allows for an easy evaluation of the coefficients in Eq. 3.3; multiplying by $\phi_m(t,\sigma)$ and summing over time, one obtains immediately

$$c_n(\sigma) = \sum_{t=-\infty}^{\infty} \phi_n(t,\sigma)x(t). \tag{3.5}$$

The discrepancy between the $x(t)$ and its expansion (Eq. 3.3) is measured by the summed square error

$$\sum_t |e(t,\sigma)|^2 = \sum_t |x(t) - \sum_{n=0}^{N-1} c_n(\sigma)\phi_n(t,\sigma)|^2 \tag{3.6}$$

This error depends on the choice of σ. The optimal σ was found by stepwise incrementation of σ in order to find the expansion that resulted in the lowest possible error.

There is still a choice of how many Hermite functions to use in the expansion (Eq. 3.3). Adding more function will decrease the error between the original

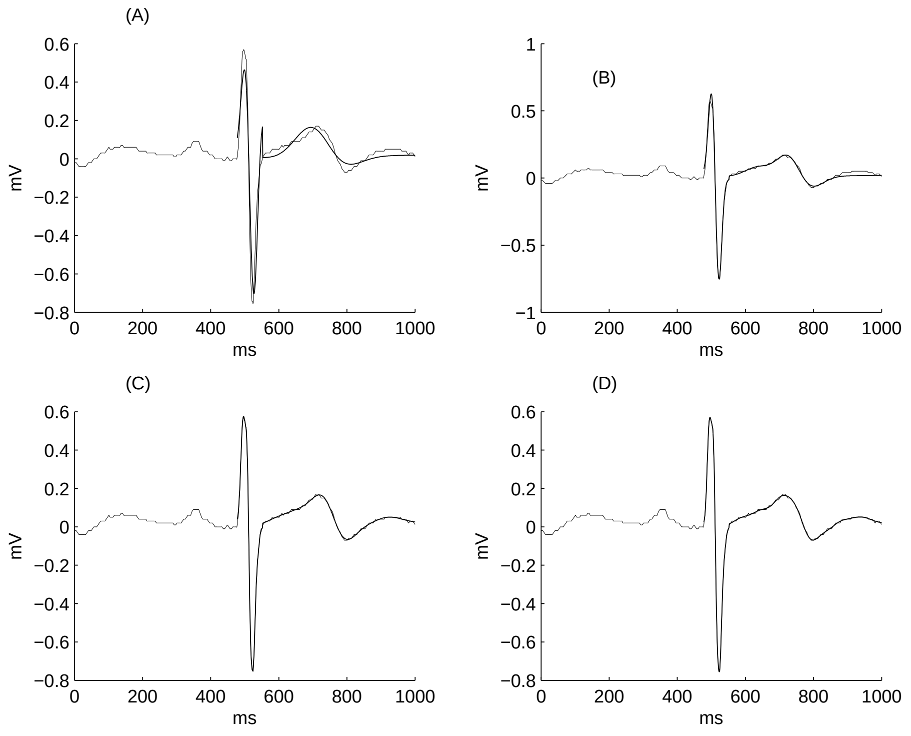

Figure 3.2: Reconstruction of lead V2 (thin line) for an AMI ECG. Graphs A-D shows the reconstruction (thick line) using 3, 5, 11 and 20 Hermite functions, respectively.

and the reconstructed ECG, at the expense of having more expansion coefficients. Figure 3.2 shows the original V2 lead of an AMI ECG together with the reconstructed one, using 3, 5, 11 and 20 Hermite functions for each complex.

We decided to use 11 Hermite functions for both the QRS and the ST-T complex. This generally allowed for reconstructed leads that visibly were very similar to the original ones. However, not all of the parameters were actually used as feature variables in the classification. The reason for using more Hermite functions than actually needed for the classification of the ECG is that reconstructed ECGs will be used in the case-based sensitivity analysis (cf. section 3.2.5).

3.2.4 Artificial Neural Networks

In this study we have used Bayesian inference techniques to ANN learning. Bayesian training of ANNs consists of picking a number of independent ANNs, each defined by its set of weights $\mathbf{w}$, from a probability distribution. This posterior distribution $p(\mathbf{w}|D)$, which is a function of the weights $\mathbf{w}$, represents the degree of belief for the different ANNs using the observed patient data D and is constructed using Bayes' theorem. To make a prediction for a test case one has to, theoretically, *integrate* over a multi-dimensional weight space. Practically, one approximates the integration with a finite sum, i.e. the average output $\langle y(\mathbf{x}_{\text{test}})\rangle$ for a test case $\mathbf{x}_{\text{test}}$, is computed as

$$\langle y(\mathbf{x}_{\text{test}})\rangle = \frac{1}{L}\sum_{i=1}^{L} y(\mathbf{w}_i, \mathbf{x}_{\text{test}}) \tag{3.7}$$

$y(\mathbf{w}_i, \mathbf{x}_{\text{test}})$ is the output of the ANN defined by the set of weights $\mathbf{w}_i$ that is drawn from the probability distribution $p(\mathbf{w}|D)$ using Markov Chain Monte Carlo methods [14]. The training of the ANNs were accomplished using the software package of Neal [15, 16].

The ANNs consisted of one input layer, one hidden layer and one output layer. The number of input nodes ranged from 48 to 108 depending on the set of input variables used. The hidden layer contained 8 nodes and the output layer consisted of one node that encoded whether the ECG originated from a patient that suffered from acute myocardial infarction or not.

When training Bayesian ANNs some parameters have to be set, including the number of networks L to average over and hyper-parameters to use in the posterior distribution. This model selection was accomplished using 3-fold cross-validation on the training set.

Measuring the Performance

For each test case in the test set the ANN classifier presents an output value between 0 and 1. A threshold in this interval was used above which all values were regarded as consistent with acute myocardial infarction. By varying this threshold, a receiver operating characteristic (ROC) curve was obtained. The performance of each ANN classifier was measured as the area under the ROC curve. The 95% confidence limits of the area were estimated by a bootstrap technique [17].

The difference in performance between two ANN classifiers was measured as the difference in area under the ROC curves. The statistical significance of such an observed area difference was assessed by means of a permutation test [17] as follows:

A new classification list was created by randomly selecting for each of the 739 test cases either the classification made with the first ANN or the classification made with the other ANN. A second list was created from the classification not included in the first one. The two lists were used to construct two ROC curves, and the areas under the curves were calculated, as was the area difference (test statistic). The procedure was repeated 50,000 times. The relative frequency of area differences that had an absolute value greater than the actual difference was taken as the probability of obtaining at least the actual area difference if no true difference existed.

3.2.5 Case-Based Sensitivity Analysis

In a case-based analysis we are interested in a single ECG and the diagnosis it received using a trained ANN classifier. By case-based *sensitivity* analysis for ECGs we aim at finding temporal regions of a 12-lead ECG that are critical to the output of the trained ANN classifier. There are situations where such a case-based analysis of ECGs may be of interest, namely:

- Explaining the functioning of the ANN classifier by showing regions of the ECG critical to the ANN output.
- Understanding misclassified ECGs by "reverse engineering".
- Feature selection.

The approach taken in this paper for the sensitivity analysis is based on the notion of *causal importance*, since the method will monitor the ANN output response when manipulating the ANN inputs. This is different from the notion of *predictive importance* which aims at monitoring the generalization performance when deleting ANN inputs, where the latter often requires retraining of the ANN classifier.

The proposed method will perturb a small number, N_{mod}, of ANN inputs. The size of each perturbation, measured in units of standard deviation over the ANN training set, is constrained not to be larger than some constant. The aim is to select the perturbation, following these constraints, that maximizes the change in ANN output.

To see why this is useful, consider the case when the ANN classifier has diagnosed a certain ECG as showing signs of myocardial infarction. By producing a perturbed ECG which the ANN ensemble classifies as healthy, and comparing this perturbed ECG to the original one, one may reveal information about what parts of the ECG that the ANN classifier found critical when computing the output. Notice that this reconstruction of perturbed ECGs from perturbed ANN inputs is made possible by the fact that the Hermite decomposition is invertible; no such reconstruction is possible in the traditional approach where the feature extraction consists of measuring amplitudes and slopes.

To proceed in finding the optimal perturbation, we discretize the perturbation into M steps. M should be large enough that the output varies smoothly as the the inputs move from one perturbation to the next. For the current application, $M = 10$ was used.

Let ϵ_{il} denote the set of possible perturbations of input i:

$$\epsilon_{il} = \left(-\frac{M}{2} + l\right)\frac{\sigma_i}{M} \qquad (l = 0 \ldots M) \tag{3.8}$$

where σ_i is the standard deviation of input i. Note that the extremal perturbations are $\epsilon_{i0} = -\sigma_i/2$ and $\epsilon_{iM} = \sigma_i/2$, and that $\epsilon_{i\frac{M}{2}} = 0$ which means that $l = \frac{M}{2}$ corresponds to no perturbation.

The actual perturbation dx_i of input i is expressed as

$$dx_i = \sum_{l=0}^{M} s_{il}\epsilon_{il} \tag{3.9}$$

where s_{il} are binary decision variables defined such that only the one corresponding to the chosen perturbation is turned on:

$$s_{il} = \begin{cases} 1 & \text{if} \quad x_i \to x_i + \epsilon_{il} \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

Obviously, the variables must fulfill the *Potts* normalization condition:

$$\sum_{l=1}^{M} s_{il} = 1 \quad (i = 1, ..., N) \tag{3.11}$$

By this construction, the input vector x can be changed into one out of possible $(M+1)^N$ new input vectors.

This is a NP-hard combinatorial optimization problem, which we attack by heuristic methods. An energy (or cost) function is defined:

$$E(\{s_{il}\}) = \pm\left[y(\mathbf{x} + d\mathbf{x}(\{s_{il}\})) - y(\mathbf{x})\right] + \alpha\left(\sum_{i=1}^{N} s_{i\frac{M}{2}} - (N - N_{\text{mod}})\right)^2 \tag{3.12}$$

The first term serves to maximize the decrease of the output $y(\mathbf{x})$ (or increase, in the case of the minus sign). The second term favors solutions where N_{mod} inputs are perturbed; α is a tunable parameter.

To find a good minimum in this energy landscape, i.e. to find a good solution, we change the binary decision variables s_{il} into fuzzy, real-valued decision variables, and perform the optimization by means of Mean Field Annealing [18]. The result of the minimization of Eq. 3.12 is a set of binary variables that defines the optimal perturbation, i.e. the perturbation of the original input that will have the largest effect on the output of the ANN ensemble. This modified input also corresponds to a new ECG by means of the Hermite expansion.

Apart from the requirement that exactly N_{mod} inputs be perturbed, we imposed the additional constraint that the perturbed inputs must all come from two leads. This constraint is optional; gathering all the perturbations into two leads has the advantage of making these perturbations easier to see, and of increasing the detail in the perturbation of the beat. To impose this constraint, the energy function (Eq. 3.12) is modified by adding the term

$$\beta \left(\Theta - \sum_{e=1}^{12} \theta(n_e) \right)^2 \qquad , \tag{3.13}$$

where β is another tunable parameter and n_e is the number of modified inputs for lead e:

$$n_e = \frac{N}{12} - \sum_{i \in lead\ e} s_{i\frac{M}{2}} . \tag{3.14}$$

Recall that $N/12$ is the number of inputs per lead. $\theta(n_e)$ is a *soft* measure of whether lead e is perturbed or not, defined as

$$\theta(n_e) = \frac{1}{1 + \exp(-2(n_e - 0.5))} \tag{3.15}$$

The reason for using this soft sigmoid rather than a step function with threshold 0.5, comes from the mean field annealing procedure that uses fuzzy decision variables. Θ is the value of $\sum_{e=1}^{12} \theta(n_e)$ when the desired solution is reached, $\Theta = 2\ \theta(N_{mod}/2) + 10\ \theta(0)$, for the 2-lead constraint.

Table 3.1: Number of input variables used, for each lead, when training different ANN classifiers.

Input variables	QRS complex	ST-T complex
	Hermite expansion ANNs	
Hermite components	2 (c_0, c_1)	5 $(c_0, c_1, c_2, c_3, c_4)$
Amplitude shift	-	1
QRS duration	1	-
	ST-T amplitude ANNs	
Amplitudes	-	5 (ST-J amp, ST-2/8 amp, ST-3/8 amp, T_+ amp and T_- amp)
Slope	-	1 (ST slope)

3.3 Results

3.3.1 Selecting input variables

The QRS and ST-T complexes were decomposed using 11 Hermite components each. Of these, only a few were used when training the ANN classifiers. The selection of inputs to the ANN was performed using three-fold cross-validation on the training set which resulted in the following set: $\{c_0,\ c_1\}$ for the QRS beat and $\{c_0,\ c_1,\ c_2,\ c_3,\ c_4\}$ for the ST-T beat (see Table 3.1). The amplitude shift defined in the preprocessing step prior to the Hermite expansion and the QRS duration were also part of the input variable set. The variables not used in the training were, however, stored in order to be used when reconstructing the modified ECGs.

To compare with the results found by Hedén et al. [6] we have also used amplitude and slope measurements from the ST-T complex (see Table 3.1).

3.3.2 Comparing ROC area performance

Table 3.2 shows the performance of the ANN classifiers, in terms of the area under the ROC curve. We present results using all 12 leads and results using a subset of 8 leads only. This reduction is supported by the fact that the extremity leads III, aVF, aVL and aVR are constructed using leads I and II,

Table 3.2: The results of the ANN classifiers using different sets of inputs. The 95% confidence intervals, given in parentheses, are estimated using a bootstrap technique. For each input set 10 ensembles of ANN classifiers were trained. The median ROC area and its corresponding confidence interval are presented in this table.

Input dataset	ROC area (%)	
Hermite expansion (12 leads)	83.4	(80.6 86.3)
ST-T amplitudes (12 leads)	84.3	(81.6 87.1)
Hermite expansion (8 leads)	82.7	(79.8 85.6)
ST-T amplitudes (8 leads)	82.2	(79.4 85.3)

e.g. Lead III = II - I.

When using all 12 leads, a marginally larger ROC area (84.3%) is obtained for the representation using ST-T amplitudes compared to the Hermite expansion representation (83.4%). This difference is not statistically significant ($p = 0.4$). Hedén et al. [6] reported an ROC area of 86% using the ST-T amplitudes as inputs. The data set used in their study contained a larger group of normals (10452 ECGs), which may explain their somewhat better performance.

Using only 8 leads (V1-V6, I and II) as inputs to the ANN classifiers, the performance is almost the same for the Hermite and the ST-T representation, 82.7% and 82.2% respectively ($p = 0.8$). For ANNs trained with ST-T amplitudes, the ROC area is significantly larger when using 12 leads than when using 8 leads ($p = 0.04$).

One can conclude that representing the ECGs in terms of Hermite expansion coefficients together with preprocessing parameters (QRS duration and the amplitude shift) is almost as good the ST-T amplitudes, when detecting AMI using ANN classifiers. The former representation however has the advantage of being invertible, i.e. one can recover the complete ECG signal from it. This is further pursued in the case-based sensitivity analysis below.

3.3.3 Sensitivity analysis

Here we present some results using the case-based sensitivity method described in the methods section. Our optimization procedure perturbs a small number of the Hermite components fed to the ANN, such that the change of the ANN response is as large as possible. These modified Hermite coefficients, along with the unperturbed components, are then used to reconstruct a modified ECG. This modified ECG is displayed along with the original ECG reconstructed from the unperturbed Hermite coefficients. The modifications are clearly visible, and indicate the temporal ECG regions critical for ANN performance.

The ANNs trained with 12 leads were used throughout; 6 inputs from 2 leads were allowed to be modified by up to half a standard deviation each (cf. section 3.2.5). The sign in Eq. 3.12 was selected so as to *change* the diagnosis; ECGs that gave an ANN output larger than 0.5 were modified so as to decrease the output, and ECGs that gave an output smaller than 0.5 were modified so as to increase the output. Only the 5 Hermite coefficients corresponding to the ST-T complex, and the amplitude shift, were allowed to be modified. The QRS durations were kept constant as changing them would compress or expand the entire beat in time in an curious manner. The 2 Hermite coefficients corresponding to the QRS complex were also kept fixed as they are less important for the diagnosis of AMI; we chose to focus on the ST-T complex in this analysis.

Figure 3.3 shows an AMI ECG from the test set that was correctly classified by the ANN. The original ANN output was 0.94; the ANN output when fed with the optimally perturbed coefficients was 0.76.

The interpretation is as follows: reducing the ST-depression of the ST complex in the V2 and V3 leads produces an ECG that is, according to the ANN, more "healthy" i.e. showing less signs of AMI. These are thus the regions that were, according to our automatic sensitivity analysis, most important for the ANN in making its decision. This corresponds very well to what an experienced ECG reader would select as being the important part of the ECG for a diagnosis of AMI.

Another example can be found in Fig. 3.4 which shows another AMI ECG correctly classified by the ANN. For this ECG the case-based sensitivity method resulted in a modified ECG with a noticeable difference in lead III. (There is also a small, hardly visible, change in lead V2.) The difference corresponds to a lowering of the ST elevation in lead III, thus making the signs of AMI less pronounced. (The ANN output was lowered from 0.93 to 0.84.) This is again in accordance with what a human expert would say.

To what extent do the regions found by the case-based sensitivity method correspond to regions that an experienced ECG reader would classify as important? To answer that question an experienced ECG reader examined all AMI cases from the test set. The ECG reader was instructed to select the two leads where

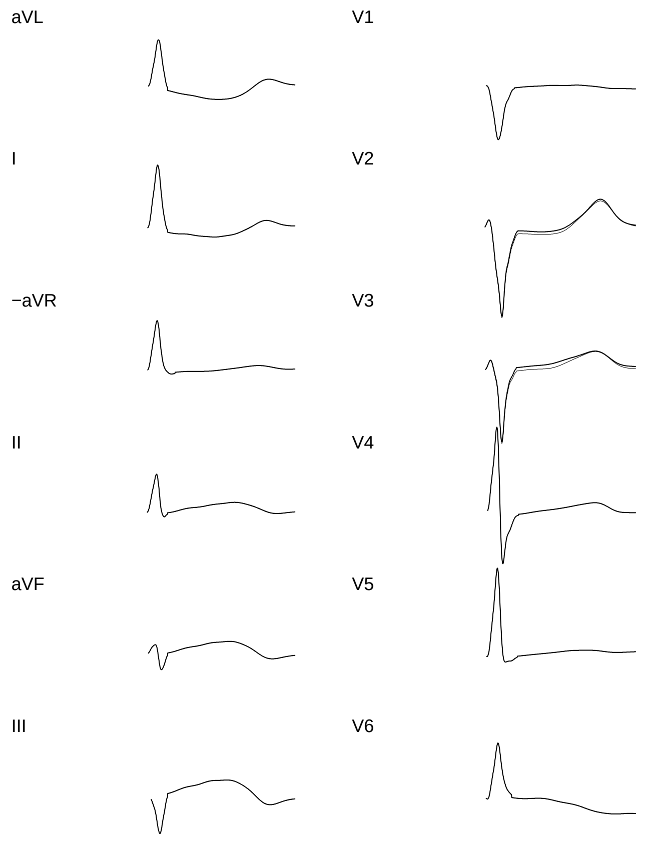

Figure 3.3: An ECG from the AMI test set. The thick line corresponds the the modified ECG and the thin line is the ECG seen by the ANN classifier. A difference can be seen in the ST-T complex of lead V2 and V3, which are the regions sensitive to the ANN classification.

the most important signs of AMI was found. The ECG reader managed this task for 146 of the 371 AMI cases in the test set; the others were dismissed as inconclusive.

These selected leads were then compared to the two leads that the case-based sensitivity method found for the same set of test ECGs. In 64 of these 146 cases (46%) at least one of the leads agreed. If one examines the full confusion matrix one can see that the experienced ECG reader focused on leads V2, V3, III and aVF, whereas our method focused on all pre-cordial leads together with

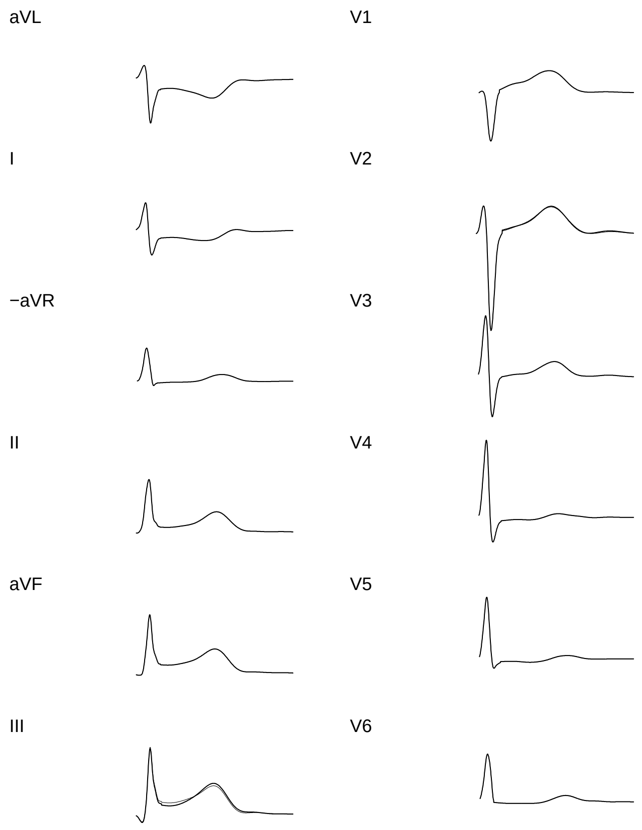

Figure 3.4: An ECG from the AMI test set. The thick line corresponds the the modified ECG and the thin line is the ECG seen by the ANN classifier. A noticeable difference can be seen in lead III.

lead III.

3.4 Discussion

3.4.1 Main findings

The results show that the 12-lead ECG can be represented using Hermite polynomials when diagnosing acute myocardial infarction using artificial neural network classifiers. The performance was essentially the same as using traditional amplitude (and slope) measurements from the ST-T complex. The performance is also in accordance with previous findings [6].

An important aspect of ANN learning is to be able to explain the way ANNs work. In this paper we have developed an approach that implicitly tries to explain the functioning of the ANN by finding small perturbations of the inputs that correspond to large changes in ANN output. This together with a Hermite representation of the leads results in a modified ECG that can be visualized. Differences between the original and the modified ECG represent critical regions for the ANN classifier.

The comparison between the experienced ECG reader and the case-based sensitivity method showed only a limited agreement (46% where at least one of the two leads agreed). This may seem like a large discrepancy, but bear in mind that the leads are correlated, the AMI information is not resident in only one or two of the directly measured leads, and there is additional redundancy in the ECG as four of the leads are constructed as linear combinations of leads I and II. Therefore, one can never hope to obtain an exact or near-exact match.

What leads focused on by the human reader depends to some extent on his or her habits; the physician is used to focus on leads V2 and V3, but in some cases there may be more information in the other pre-cordial leads, picked up by the ANN but not easily discernible for the human eye. A more systematic investigation with at least two experienced ECG readers is needed in order find intra-observer and inter-observer variabilities among them. As for the ANNs, there is a semi-degeneracy in the optimal perturbations: there is one perturbation found to give the largest change in ANN output, but as the network uses information from all twelve leads there are often other perturbations involving other leads that also give a fairly large change in output. What's more, there is nothing magic about our selection of the number of ANN inputs to vary or the maximum perturbation of each input, and changing these parameters sometimes results in different leads being selected. In addition, the minimization of the cost function (Eq. 3.12) is a difficult problem, and one may not always find a good solution; further developments with respect to the construction and minimization of this cost function may therefore be needed.

Further refinements of the method are needed, but we believe that what we have already looks promising and much more attractive than the "black box" operation often characteristic of ANNs and other data mining tools.

3.4.2 Clinical Implications

Previous studies (e.g. [6, 19, 20]) indicate that ANN can be used to improve automated ECG interpretation for AMI. Even experienced cardiologists can benefit from using these ANNs as decision support [6]. The lack of an explanation for each diagnosis made by the ANN classifier has often been a regarded as a drawback. The proposed case-based sensitivity method together with the Hermite representation of the 12-lead ECG is an approach to eliminate that drawback.

In an emergency department a decision support system for the interpretation of ECGs has the advantage of being fully automated and the proposed method for explanation support is not a limitation in that respect. A disadvantage of this type of decision support lies in the fact that the ECG is only used in a limited part of the diagnostic decision, regarding AMI. In a recent retrospective study by Baxt et al. [5] a ANN system was constructed for the diagnosis of AMI. Their system used 40 variables from patient histories, physical examination, ECG recordings and chemical cardiac markers as inputs to the ANN. The results found in this study was excellent. This system is, however, not automated and requires pre-classifications by physicians for some of the input variables.

3.4.3 Conclusions

We have presented a method for automated detection of AMI patients using the 12-lead ECG. Each lead was decomposed using Hermite basis functions and the resulting coefficients were used as inputs to ANN ensembles that were trained to detect AMI. The performance was compared to that of using traditional ST-T amplitudes and slopes on a study population of 2238 ECGs. No significant difference was found in favor for the amplitude/slope measurements. The area under the ROC curve was 83.4% for the Hermite representation using 12 leads.

Furthermore, we have also presented a novel method for case-based sensitivity analysis that was used to show regions of the ECG critical for the ANN response. This method was based on: (i) a cost function used to find local ECG perturbations leading to the largest possible change in ANN output, and (ii) a minimization scheme for this cost function using mean field annealing.

With proper further developments, we believe the proposed method has potential as a decision support system that can provide a good suggestion for diagnosis, as well as providing the physician with insight into why the system gave the answer it did.

3.5 Acknowledgments

We thank Carsten Peterson for fruitful discussions. This work was in part supported by the Swedish Natural Science Research Council and the Swedish Foundation for Strategic Research.

References

[1] P. Lisboa, E. Ifeachor, P. Szczepaniak (Eds.), Artificial neural networks in biomedicine, Springer-Verlag, London, 2000.

[2] W. Baxt, Use of an artificial neural network for the diagnosis of myocardial infarction, Ann Intern Med 115 (1991) 843–848.

[3] R. Harrison, S. Marshall, R. Kennedy, The early diagnosis of heart attacks: a neurocomputational approach, in: Proceedings of the International Joint Conference on Neural Networks, Seattle WA, 1991, pp. 1–5.

[4] R. Kennedy, R. Harrison, A. Burtonc, H. Fraser, W. Hamer, D. MacArthur, R. McAllum, D. Steedman, An artificial neural network system for diagnosis of acute myocardial infarction (AMI) in the accident and emergency department: evaluation and comparison with serum myoglobin measurements, Comput Methods Programs Biomed 52 (1997) 93–103.

[5] W. Baxt, F. Shofer, F. Sites, J. Hollander, A neural computational aid to the diagnosis of acute myocardial infarction, Ann Emerg Med 34 (2002) 366–373.

[6] B. Hedén, H. Öhlin, R. Rittner, L. Edenbrandt, Acute myocardial infarction detected in the 12-lead ECG by artificial neural networks, Circulation 96 (6) (1997) 1798–1802.

[7] M. Ohlsson, H. Öhlin, S. Wallerstedt, L. Edenbrandt, Usefulness of serial electrocardiograms for diagnosis of acute myocardial infarction, Am J Cardiol 88 (2001) 478–481.

[8] W. G. Baxt, H. White, Bootstrapping confidence intervals for clinical input variable effects in a network trained to identify the presence of acute myocardial infarction, Neural Comput 4 (1995) 772–780.

[9] R. Caruana, Case-based explanation for artificial neural nets, in: H. Malmgren, M. Borga, L. Niklasson (Eds.), Artificial Neural Networks in Medicine and Biology, Springer-Verlag, London, 2000, pp. 303–308.

[10] L. Sörnmo, P. Börjesson, M. Nygårds, O. Pahlm, A method for evaluation of QRS shape features using a mathematical model for the ECG, IEEE Trans Biomed Eng 28 (1981) 713–717.

[11] R. Jane, S. Olmos, P. Laguna, P. Caminal, Adaptive Hermite models for ECG data compression: performance and evaluation with automatic wave detection, in: Proc. of Computers in Cardiology, IEEE Computer Society, 1993, pp. 389–392.

[12] M. Nygårds, L. Sörnmo, Delineation of the QRS complex using the envelope of the ECG, Med Biol Eng Comput 21 (1983) 583–547.

[13] M. Lagerholm, C. Peterson, G. Braccini, L. Edenbrandt, L. Sörnmo, Clustering ECG complexes using Hermite functions and self-organizing maps, IEEE Trans Biomed Eng 47 (2000) 838–848.

[14] R. Neal, Bayesian training of backpropagation networks by the hybrid Monte Carlo method, Tech. Rep. CRG-TR-92-1, Department of Computer, Science University of Toronto, Canada (1992).

[15] R. M. Neal, Software for flexible Bayesian modeling, http://www.cs.toronto.edu/˜radford/fbm.1999-03-13.doc/index.html (1999).

[16] R. M. Neal, Bayesian learning for neural networks, Springer-Verlag, New York, 1996.

[17] R. Wehrens, H. Putter, L. Buydens, The bootstrap: A tutorial, Chemometr Intell Lab Syst 54 (2000) 35–52.

[18] C. Peterson, B. Söderberg, A new method for mapping optimization problems onto neural networks, Int J Neural Syst 1 (1989) 3–22.

[19] G. Bortolan, J. Willems, Diagnostic ECG classification based on neural networks, J Electrocardiol 26(suppl) (1993) 75–79.

[20] S.-E. Olsson, H. Öhlin, M. Ohlsson, L. Edenbrandt, Neural networks - a diagnostic tool in acute myocardial infarction with concomitant left bundle branch block, Clinical Physiology and Functional Imaging 22 (2002) 295–299.

A Fuzzy Matching Approach to Multiple Structure Alignment of Proteins

Paper IV

A Fuzzy Matching Approach to Multiple Structure Alignment of Proteins

Henrik Haraldsson and Mattias Ohlsson

Complex Systems Division, Department of Theoretical Physics
University of Lund, Sölvegatan 14A, SE-223 62 Lund, Sweden
`http://www.thep.lu.se/complex/`

LU TP 03-17

We present a novel approach for multiple structure alignment of proteins based on fuzzy pairwise alignments of each protein to a virtual consensus chain. These alignments are alternated with translations and rotations of the proteins onto the consensus structure, and with updating each consensus atom by placing it in the middle of the protein atoms it's aligned to. The pairwise alignments use mean-field annealing optimization of fuzzy alignment variables, based on a cost expressed in terms of distances between aligned atoms and of gaps. No initialization in terms of all-to-all protein alignments is needed, and the only information required is the 3-D coordinates of the C_α atoms, with the optional inclusion of secondary structure information or other matching preferences. The degree of fuzziness is controlled by a temperature parameter, which is lowered in an annealing scheme. The CPU consumption is modest, and scales approximately linearly with the number of proteins to align. Our approach is tested against a set of protein families from the HOMSTRAD database, and against a multiple structure alignment algorithm based on Monte Carlo techniques, with good results.

4.1 Introduction

Comparative analysis of protein structures is a subject of utmost relevance. It enables the study of functional relationships between proteins and is very important for homology and threading methods in structure prediction. Multiple structure alignment of proteins is needed in order to group proteins into families, which enables a subsequent analysis of evolutionary issues. It is also important in order to build a consensus structure that encapsulates common re-occurring substructures among the structures. However, multiple structure alignment is not an easy task.

There exists many methods for pairwise structure alignment (see e.g. [1, 2, 3, 4]). Few of these have been further developed for the alignment of multiple structures. The ones that exist fall into two broad groups. Methods in the first group perform all *pairwise* structure alignments and define the structure most similar to the others. Multiple structure alignment is then achieved by starting with an initial alignment of all structures to this selected one. Methods in the second group do not rely on all-to-all pairwise alignments as a starting point, instead they try to solve the multiple alignment problem in a more direct fashion.

Methods that fall within the first group can be found in the work of Gerstein et al. [1] and Guda et al. [5]. The latter uses Monte Carlo optimization techniques to refine the initial alignment found by pairwise structure alignment using the Combinatorial Extension algorithm [3]. Orengo et al. [6] use double dynamic programming in their SSAPM algorithm, where they form an initial consensus structure from the two protein structures found to match each other the closest. Then they tentatively align all remaining structures to this consensus, select the one with the closest match and merge it into the consensus. This matching and merging is repeated until all protein structures have been merged into the consensus structure. The COMPARER algorithm developed by Šali et al. [7] also performs an all-to-all pairwise alignment of all proteins as a first step in their multiple alignment procedure.

A method that does not rely on an initial all-to-all pairwise alignment is used in Leibowitz et al. [8]. This algorithm finds geometric substructures common for all structures to be aligned. Structural alignment of these cores then induces alignments of the full molecules. The starting point for their algorithm is only the 3-D coordinates for the C_α-atoms defining the structures to be aligned.

In this paper we have developed an approach for multiple structure alignment, that falls within the second group, with two main ingredients: (i) the construction of a *virtual consensus chain* that each individual protein is aligned to, and (ii) the use of an efficient fuzzy pairwise structure alignment method. The alignment is carried out in an iterative procedure. In each iteration step, the proteins are rotated and translated towards the consensus chain, guided by the fuzzy pairwise alignment method. Also, in each step a new consensus chain is computed from the alignments of the individual proteins.

The method, which requires no initial alignment from e.g. multiple sequence alignment, uses only 3-D coordinates of the C_α-atoms along the backbone and optionally secondary structure information for each amino acid. The proposed method, which is very general, has some appealing properties:

- **Scaling properties.** The method works by aligning each of the proteins to a common consensus chain and no initial alignment taken from an all-to-all pairwise alignment is needed. Hence, the CPU time needed grows only (approximately) linearly with the number of structures to align.

- **Generality of formulation.** Almost arbitrary additional constraints are easily incorporated into the formalism including, for example, different functional forms of gap penalties and sequence matching preferences.

The proposed method was tested on structures taken from the HOMSTRAD database [9] of protein structure alignments for homologous families. Furthermore the results of our method were compared to those produced by the Monte Carlo method of Guda et al. [5]. The selected datasets were chosen to contain sets of proteins of various lengths, different structural classes and different average percentages of identities. For most cases our algorithm produced alignments in good agreement with the HOMSTRAD database and comparable to results obtained by the algorithm of Guda et al.

4.2 Methods

4.2.1 Multiple Alignment procedure

Consider K proteins that are to be structurally aligned to each other. In our approach this will be accomplished by aligning each of the K structures to a common *consensus* chain. Thus, the multiple alignment problem has been reduced to the construction of the common consensus chain and to perform K pairwise structure alignments. These two steps are not performed independently of each other; rather, we use an iterative 3-step procedure as follows:

- Fuzzy pairwise structure alignment of each of the K chains to the common consensus chain.
- Weighted rigid body rotations and translations for each of the K chains.
- Calculation of a new consensus chain.

The first two steps are based on the work by Blankenbecler et al. [10] and will only be reviewed briefly.

In what follows we denote by $\mathbf{x}_i^{(k)}$ $(i = 1, ..., N_k)$ the atom coordinates of protein k, with length N_k. The phrase "atom" is here used in a generic sense – it could represent individual atoms but also groups of atoms. In this paper it will mean C_α-atoms along the backbone. Let $\mathbf{y}_j$ $(j = 1, ..., M)$ denote the atoms (hypothetical C_α positions) of the consensus chain (of length M). We use a square distance metric between the atoms in the proteins and atoms in the consensus chain,

$$d_{i,j}^{(k)} = |\mathbf{x}_i^{(k)} - \mathbf{y}_j|^2 \tag{4.1}$$

but the formalism is not confined to this choice.

4.2.2 Weighted rigid body rotations and translations

The objective of the rigid body rotation and translation of the proteins is to minimize the distance between the matched atoms for each of the K proteins and the consensus chain. Let $\tilde{\mathbf{x}}_i^{(k)}$ be the coordinates of the translated and rotated protein k, i.e., $\tilde{\mathbf{x}}_i^{(k)} = \mathbf{a}^{(k)} + \mathcal{R}^{(k)}\mathbf{x}_i^{(k)}$. Based on the fuzzy assignment matrix $\mathcal{W}^{(k)}$ we determine the translation vector $\mathbf{a}^{(k)}$ and the rotation matrix $\mathcal{R}^{(k)}$, by minimizing the following chain error function,

$$E_{\text{chain}}^{(k)} = \sum_{i=1}^{M}\sum_{j=1}^{N} \mathcal{W}_{i,j}^{(k)} \left(\mathbf{a}^{(k)} + \mathcal{R}^{(k)}\mathbf{x}_i^{(k)} - \mathbf{y}_j\right)^2 \ . \tag{4.2}$$

Matched atoms for protein k and the consensus chain are given by the elements $\mathcal{W}_{i,j}$, of the fuzzy assignment matrix. These elements are not restricted to be integers, leading to an interpretation of fuzzy assignments between atoms. The minimization of $E_{\text{chain}}^{(k)}$ can be solved exactly [11], with closed-form expressions for $\mathcal{R}^{(k)}$ and $\mathbf{a}^{(k)}$. It should be noted that this solution is rotationally invariant (independent of $\mathcal{R}^{(k)}$) for the special case when the atoms in the two chains match each other with the same weight, i.e. when $\mathcal{W}_{i,j}^{(k)}$ is constant for all i and j.

4.2.3 Fuzzy pairwise structure alignment

Our approach for multiple structure alignment is an iterative procedure involving pairwise alignments to a common consensus chain. Each pairwise alignment is obtained using the fuzzy alignment method developed by Blankenbecler et al. [10]. This method is similar to the dynamical programming method for global sequence alignment [12], but with two important differences. First, instead of using a score between aligned atoms, a cost formulation is used. This cost, which depends on the distances between the atoms and on the number of gaps and their locations, is changing throughout the alignment procedure. Second, in the original Needleman–Wunsch algorithm [12] an optimal alignment path is calculated, whereas fuzzy alignment paths are computed in the method by Blankenbecler et al. The rest of this section gives a small review of the fuzzy alignment method. For readability we have dropped the (k) superscript that indicates one of the k proteins to align.

The structure alignment of two proteins is carried out in an annealing procedure, controlled by a *temperature* parameter T. Let $\mathcal{D}_{i,j}$ denote a fuzzy generalization of the optimal alignment cost at node (i,j) in the *dot-matrix* (cf. [10]), used to represent all possible alignments of two proteins. We have

$$\mathcal{D}_{i,j} = \sum_{l=1}^{3} v_{i,j;\ l}\, \widetilde{\mathcal{D}}_{i,j;\ l}\ , \tag{4.3}$$

where $\widetilde{\mathcal{D}}_{i,j;\ l}$ is the corresponding generalized fuzzy alignment cost if the alignment path is forced to pass through the preceeding node given by l. In the Needleman–Wunsch algorithm only the optimal direction l is used, which implies that $v_{i,j;\ l}$ are integers and $\sum_l v_{i,j;\ l} = 1$. This restriction is relaxed in the fuzzy alignment method where $v_{i,j;\ l} \in [0,1]$, but still sum up to unity. These so called *mean field* variables are calculated according to,

$$v_{i,j;\ l} = \frac{e^{-\widetilde{\mathcal{D}}_{i,j;\ l}/T}}{\sum_{l'} e^{-\widetilde{\mathcal{D}}_{i,j;\ l'}/T}}\ . \tag{4.4}$$

The generalized fuzzy alignment costs $\widetilde{\mathcal{D}}_{i,j;\ l}$ are then calculated using the following recursive relation,

$$\begin{aligned}
\widetilde{\mathcal{D}}_{i,j;\ 1} &= \mathcal{D}_{i,j-1} + \lambda_j^{(2)}(1 - v_{i,j-1;\ 1}) + \lambda_{\text{ext}} v_{i,j-1;\ 1}\ , \\
\widetilde{\mathcal{D}}_{i,j;\ 2} &= \mathcal{D}_{i-1,j-1} + d_{i,j}\ , \\
\widetilde{\mathcal{D}}_{i,j;\ 3} &= \mathcal{D}_{i-1,j} + \lambda_i^{(1)}(1 - v_{i-1,j;\ 3}) + \lambda_{\text{ext}} v_{i-1,j;\ 3}\ .
\end{aligned} \tag{4.5}$$

Here, $\lambda_a^{(n)}$ is the penalty for matching atom a in chain n to a gap, λ_{ext} is the gap extension penalty.

At each iteration in the annealing procedure of lowering T, a fuzzy assignment matrix $\mathcal{W}_{i,j}$ is calculated as

$$\mathcal{W}_{i,j} = P_{i,j} v_{i,j;\ 2}\ , \tag{4.6}$$

where $P_{i,j}$ is the probability that node (i,j) is part of the optimal path and $v_{i,j;\ 2}$ is the probability that atoms i and j are locally matched. $P_{i,j}$ can be calculated with a similar recursive relation as for $\mathcal{D}_{i,j}$. With the obvious initial value $P_{M,N} = 1$, one has

$$\begin{aligned} P_{i,j} &= v_{i,j+1;\ 1} P_{i,j+1} \\ &+ v_{i+1,j+1;\ 2} P_{i+1,j+1} \\ &+ v_{i+1,j;\ 3} P_{i+1,j}\ . \end{aligned} \tag{4.7}$$

The fuzzy assignment matrix $\mathcal{W}_{i,j}$ is used when moving one of the proteins onto the other according to the minimum of Eq. 4.2.

4.2.4 Calculation of the consensus chain

The coordinates $\mathbf{y}_j$ for the consensus chain are calculated using the coordinates of all the K proteins and the K fuzzy assignment matrices from each of the pairwise alignments. In our approach we construct $\mathbf{y}_j$ according to

$$\mathbf{y}_j = \sum_{k=1}^{K} \alpha_j^{(k)} \tilde{\mathbf{y}}_j^{(k)}\ , \quad \text{where} \tag{4.8}$$

$$\tilde{\mathbf{y}}_j^{(k)} = \sum_{i=1}^{N_k} \tilde{\mathcal{W}}_{i,j}^{(k)} \mathbf{x}_i^{(k)}. \tag{4.9}$$

The last expression (Eq. 4.9) computes the (weighted) average position of the atoms in protein k that are matched to atom j in the consensus chain. The weights $\tilde{\mathcal{W}}_{i,j}^{(k)}$ are normalized columns of the fuzzy assignment matrix $\mathcal{W}_{i,j}^{(k)}$:

$$\tilde{\mathcal{W}}_{i,j}^{(k)} = \frac{\mathcal{W}_{i,j}^{(k)}}{\sum_{i'=1}^{N_k} \mathcal{W}_{i',j}^{(k)}}. \tag{4.10}$$

The final position of consensus atom j is given in Eq. 4.8, which is a weighted sum of all $\tilde{\mathbf{y}}_j^{(k)}$; chains with a large $\alpha_j^{(k)}$ contribute more to the final consensus atom $\mathbf{y}_j$, compared to chains with a low $\alpha_j^{(k)}$. These weights are given by

$$\alpha_j^{(k)} = \frac{e^{-\beta_j^{(k)}/\psi}}{\sum_{k'=1}^{K} e^{-\beta_j^{(k')}/\psi}}, \tag{4.11}$$

where ψ is a parameter and $\beta_j^{(k)}$ is the distance from the center of mass of the positions $\tilde{\mathbf{y}}_j^{(k')}, (k' = 1, ..., K)$ to $\tilde{\mathbf{y}}_j^{(k)}$ itself. This expression (Eq. 4.11) gives approximately equal weights to positions $\tilde{\mathbf{y}}_j^{(k')}$ that are within the characteristic distance ψ from the center of mass, whereas positions $\tilde{\mathbf{y}}_j^{(k')}$ that are much further away receive a negligible weight. When calculating the center of mass of the positions $\tilde{\mathbf{y}}_j^{(k')}$, only proteins where the total matching $\sum_{i=1}^{N_k} \mathcal{W}_{i,j}^{(k)}$ to consensus atom j is larger than 10^{-4}, are considered.

4.2.5 Initialization of the consensus chain

The multiple alignment procedure starts with the definition of an initial consensus chain of length M, where M is a parameter for the algorithm, chosen to be $M = \kappa \max_k\{N_k\}$, where $\kappa \geq 1$. The consensus chain is then initiated as the longest of the K proteins to align, with random coordinates for the left-over tails (if $\kappa > 1$).

A pre-alignment step is then performed where all structures are moved towards the consensus chain using fuzzy assignment matrices $\mathcal{W}_{i,j}^{(k)}$, with a band diagonal structure. The consensus chain is then updated once using Eq. 4.8. It should be noted that the performance of the algorithm does not depend on the details of the initialization of the consensus chain. This is due to the fact that the fuzzy pairwise alignment procedure starts with a large value for the T-parameter (see Eq. 4.4), making the initial 3-D coordinates for the consensus chain less important.

4.2.6 Summary of the method

As stated in the beginning, the multiple alignment of K protein structures is accomplished by iteratively aligning each of the K proteins to a consensus chain using fuzzy assignment matrices, followed by re-computation of the consensus chain. The T-parameter that controls the degree of fuzziness of the assignment matrices is *annealed* during the iterative procedure from a large value to a small value. This ensures the necessary transition from a fuzzy assignment matrix to a binary one, where uniquely matched atoms to the consensus chain are identified. The major algorithmic steps are summarized in Fig. 4.1

1. Initialization.
   (a) Move all proteins to their common center of mass and rescale coordinates such that the largest distance between atoms within the chains is unity.
   (b) Initiate the consensus chain.
   (c) Initiate the temperature, $T = 10$.
2. Alignment procedure.
   (a) For each protein k, proceed row by row $i \to i+1$, $i = 1, \dots, M$, and in each row column by column, $j \to j+1$, $j = 1, \dots, N_k$, and calculate in the given order:
      i. $v^{(k)}_{i,j;\ l}$ from Eq. 4.4.
      ii. $\widetilde{\mathcal{D}}^{(k)}_{i,j;\ l}$ from Eq. 4.5.
      iii. $\mathcal{D}^{(k)}_{i,j}$ from Eq. 4.3.
   (b) Compute the fuzzy matching matrices from Eq. 4.6.
   (c) For each protein k, minimize Eq. 4.2 by rotation and translation of the protein to the consensus chain.
   (d) Repeat items 2(a)-2(c) 3 times.
   (e) Update the consensus chain according to Eq. 4.8
   (f) Lower the T-parameter, e.g. $T \to \epsilon * T$.
   (g) Repeat items 2(a)-2(f) until
   $(1/K) \sum_k \left(\sum_{i,j} (\mathcal{W}^{(k)}_{i,j})^2 / \sum_{i,j} \mathcal{W}^{(k)}_{i,j} \right) < 1 - 10^{-10}$.
3. Extract the multiple structure alignment by examining the alignment of each protein to the consensus chain.

Figure 4.1: The essential algorithmic steps in the multiple structure alignment algorithm.

The final multiple alignment of the K proteins is defined entirely by what is matched to the virtual consensus atoms in the K pairwise alignments, but the consensus atoms themselves are not considered part of the final alignment.

Protein atoms that are matched to one and the same consensus atom, are considered to be matched to each other and form an *alignment column.* If a consensus atom is matched to a gap in one of the pairwise alignments, this protein will naturally have a gap at this position in the multiple alignment as well. Any protein atom matched to a gap in the consensus chain, and any protein atom that is matched to a consensus atom without any other protein atoms matched to it, is considered to be unaligned. Consensus atoms that are entirely unmatched do not contribute to the final multiple alignment.

4.3 Results and Discussion

4.3.1 Data sets used

To test the quality of our alignment algorithm, we have compared alignments on a set of protein families from the HOMSTRAD database [9]. Here, the goal was not a full investigation of all families, but rather to explore a limited set with representative variation. Table 4.1 lists the protein families from HOMSTRAD used in our comparison. We denote proteins by their PDB [13] identifier.[1]

Table 4.1: Protein families from the HOMSTRAD database used in our comparison.

Family	PDB entries for the proteins	Class	N. of prot.	Ave. length	Ave. % identity
ghf7	1cel, 1egl, 2ovw, 2a39	all β	4	399	47
tim	1amk, 5timA, 1htiA, 1timA, 1ypiA, 1treA, 1ydvA, 1aw2A, 2btmA, 1tcdA	α/β barrel	10	249	47
cyt3	2cdv, 2cym, 1wad, 3cyr, 2cy3, 1age	all α	6	110	40
intb	1afcA, 2afgA, 2fgf, 2mib, 1i1b, 1iraX	all β	6	137	30
ricin	1apa, 1qciA, 1abrA, 1fmp, 1mrj, 1mrg, 1cf5A	α plus β	7	254	38

[1]PDB, Protein Data Bank, located at http://www.rcsb.org/pdb/ .

4.3.2 Heuristic post-processing

The atoms in the different protein chains are matched through the mediation of the virtual consensus atoms. If there are too few consensus atoms at a given position, protein atoms will be unaligned there. Conversely, if there are many consensus atoms close to a given position, protein atoms which really should be aligned will split up between the consensus atoms, leading to unnecessary gaps in the assignment.

At high temperatures, the consensus atoms move to their approximate positions, but as the temperature decreases, there is a tendency that some positions end up with too few or to many consensus atoms. For this situation to be remedied, a whole range of consensus atoms has to be moved one or more steps along the chain, in order to shift extra consensus atoms into or out of the interior of the alignment. There is no force in the alignment procedure that makes this happen. The pairwise protein-consensus alignments are optimized with respect to the *current* position of the consensus atoms, and the consensus atoms are likewise moved to *currently* matched protein atoms. These two steps happen sequentially rather than simultaneously. Therefore, the algorithm does not "know" that a globally superior alignment would be achieved if the consensus atoms were shifted along the sequence.

The above situation is taken care of by a series of heuristic procedures, that insert or delete consensus atoms near the end of the annealing process. The following criteria are checked for:

1. Consider two consecutive consensus atoms. If these are matched to at least one gap in each of the K proteins, then the two consensus atoms are merged by translating one of them to the midpoint between the original consensus atom positions, and deleting the other one. (Only consensus atoms actually matched to something are considered here. There is often a pool of unmatched consensus atoms, but these are excluded in this analysis.)
2. If an atom in one of the proteins is unaligned, a consensus atom is inserted at this position, unless there is another consensus atom within the distance 1.5 Å.
3. If two consensus atoms are within the distance 1.0 Å from each other, one of them is removed. This deletion is performed so as to make the average distance between the remaining consensus atoms as close to 3.8 Å as possible.

The above heuristics are performed every fifth iteration, when $\Sigma > 0.99$. The saturation Σ is a measure of how close to zero or unity the elements in the assignment matrices $\mathcal{W}_{i,j}^{(k)}$ are, and is defined as

$$\Sigma = \frac{1}{K} \sum_k \left(\frac{\sum_{i,j} (\mathcal{W}_{i,j}^{(k)})^2}{\sum_{i,j} \mathcal{W}_{i,j}^{(k)}} \right). \tag{4.12}$$

4.3.3 Comparing with HOMSTRAD alignments

Besides HOMSTRAD [9], we have compared our result with the Monte Carlo method of Guda et al. [5] which is available through a WWW-server.[2] When comparing either of the two methods against HOMSTRAD, we have counted the number of aligned columns that exactly matched that of the manual alignment; here the columns with only one atom are also counted. For each method we also present the number of aligned columns with two atoms or more, and the average column distance in the final geometrical configuration. The column distance is defined as the average of all pairwise geometric distances between atoms in a column. Columns with only one atom obviously have no column distance, and are not included in the average.

Table 4.2: Summary of the results from the comparison.

	Our method			Guda et al.		
Fam.	No. aligned columns	Ave. column dist. (Å)	No. correct columns	No. aligned columns	Ave. column dist. (Å)	No. correct columns
ghf7	403	1.30	402	402	1.30	406
tim	261	1.08	236	249	1.04	229
cyt3	118	1.81	98	111	1.69	89
intb	158	1.63	126	155	1.83	121
ricin	279	1.29	238	251	1.20	221

Table 4.2 shows the result obtained for the families listed in Table 4.1. Our method produces alignments in good agreement with the manual alignment from HOMSTRAD as can be seen from the number of matched columns or by visual inspection of the alignments. Compared to the Monte Carlo method by Guda et al. we usually have more correctly aligned columns. The average column distance for the final geometric configuration is, on the average, essentially the same for the two methods.

[2]http://cl.sdsc.edu/mc/mc.html

It is our hope that the average column distances produced by our method might be further reduced by finding even better heuristic criteria for when and where to insert or delete consensus atoms. Recall the problem of getting the right number of consensus atoms in the right place (cf. section 4.3.2). Often there are too few consensus atoms at various places; therefore unaligned protein atoms are given a consensus atom in the heuristic post-processing, unless there is a consensus atom close by already. This, however, has an unfortunate side effect: some protein atoms that really should be unaligned get a consensus atom that then aligns itself to some other unaligned atom within a distance of a few Ångströms. This causes a false alignment between the two proteins. These false alignments mainly occur in the loops and do not disrupt the overall alignment, but they can imply a high column distance for some columns. Thus, there appears to be room for improvement in terms of finding better heuristic post-processing methods; such refinements might lead to fewer false alignments, and to a lower average column distance.

The result of the multiple alignment of the triosephosphate isomerase (tim) family [14] is shown in Figure 4.2. This family consists of 10 proteins belonging to the α/β-barrel structure class. Comparing to the manual alignment from HOMSTRAD, we align 236 of the total 260 columns correctly. A detailed investigation shows that all α-helix and β-sheet structures are correctly aligned. Misalignments occurs in loop regions between α-helix and β-sheets.

Our multiple alignment algorithm depends on a number of parameters for its operation. Table 4.3 shows the values, used in this paper, for important parameters. We have used the same set of parameters for all the families tested.

Table 4.3: Parameters used in our multiple alignment method.

Parameter	Value	Legend
λ	7.0	The gap cost in the pairwise alignment.
λ_{ext}	0.7	The gap extension cost in the pairwise alignment.
κ	1.1	The multiplicative factor used when defining the length of the consensus chain (cf. section 4.2.5).
ϵ	0.7	The annealing rate of T (see Figure 4.1).
ψ	0.9	The width used in the weighting of the different chains when updating consensus atoms (see Eq. 4.11).

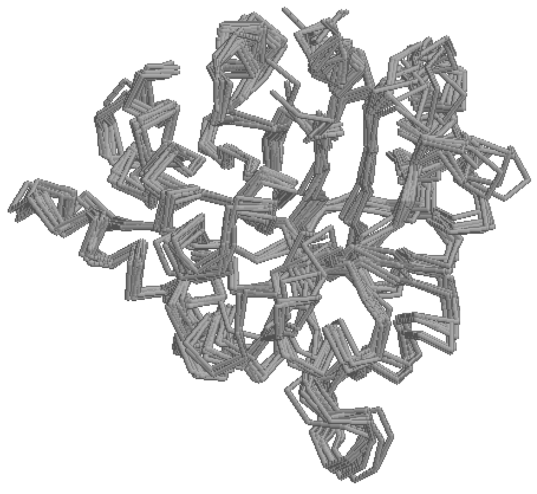

Figure 4.2: Alignment of the 10 proteins in the triose phosphate isomerase (tim) family from Homstrad using our method. Each protein is optimally aligned to the consensus chain (which is not shown). The proteins are: 1amk, 5timA, 1htiA, 1timA,1ypiA, 1treA, 1ydvA, 1aw2A, 2btmA and 1tcdA.

4.3.4 Scaling properties

Our algorithm for multiple alignment of protein structures was implemented in C++ on a PC running the Linux operating system. The running times ranged from 11 seconds for the cytochrome-c3 (cyt3) family to 140 seconds for the glycosyl hydrolase family 7 (ghf7) on a 2.4 GHz processor (Intel). The required CPU time depends on the lengths of the proteins and the number of structures to align. For the pairwise alignment algorithm it scales like $N * M$ for two structures of length N and M. However, for a common and fixed length of a set of structures, the required CPU time for the multiple alignment algorithm grows approximately linearly with the number of structures to align. This is due to the consensus chain that all structures are aligned to, and that no initial all-to-all pairwise alignment is required.

4.4 Conclusions

In summary, we have presented a novel method for multiple alignment of proteins structures. It has been tested on several protein families with encouraging results. Our approach has appealing properties, such as:

- It is fully automatic, requiring (in principle) only the 3-D coordinates of the C_α-atoms as input and no initialization using all-to-all pairwise alignments.
- With the definition of a consensus chain to which all structures are aligned, it gives rise to a scaling property, where the CPU consumption grows (approximately) linearly with the number of structures to align.
- With the underlying fuzzy pairwise alignment, the method can easily be extended to handle more detailed chain representations (e.g. side chain orientation) and additional user-provided constraints of almost any kind.

4.5 Acknowledgments

We thank Carsten Peterson for fruitful discussions. This work was in part supported by the Swedish Research Council.

References

[1] M. Gerstein and M. Levitt. Using iterative dynamic programming to obtain accurate pairwise and multiple alignments of protein structures. In D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and R. F. Smith, editors, *Proceedings of the Fourth International Conference on Intelligent Systems in Molecular Biology*, pages 59–67, Menlo Park, CA, 1996. AAAI Press.

[2] L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *J. Mol. Biol.*, **233**:123–138, 1993.

[3] I. N. Shindyalov and P. E. Bourne. Protein structure by incremental combinatorial extension of the optimal path. *Protein Eng.*, 11:739–747, 1998.

[4] J.D. Szustakowski and Z. Weng. Protein structure alignment using a genetic algorithm. *Proteins*, 38:428–440, 2000.

[5] C. Guda, E.D. Scheeff, P.E. Bourne, and I.N. Shindyalov. A new algorithm for the alignment of multiple protein structures using monte carlo optimization. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 6, pages 275–286, 2001.

[6] C.A. Orengo and W.R. Taylor. SSAP: sequential structure alignment program for protein structure comparison. *Methods Enzymol*, 266:617–635, 1996.

[7] A. Šali and T.L. Blundell. Definition of general topological equivalence in protein structures: A procedure involving comparison of properties and relationships through simulated annealing and dynamic programming. *J. Mol. Biol.*, 212:403–428, 1990.

[8] N. Leibowitz, Z. Y. Fligelman, R. Nussinov, and H. J. Wolfson. Automated multiple structure alignment and detection of a common substructural motif. *Proteins*, 43:235–245, 2001.

[9] K. Mizuguchi, C.M. Deane, T.L. Blundell, and J.P. Overington. HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Sci*, 7:2469–2471, 1998.

[10] R. Blankenbecler, M. Ohlsson, C. Peterson, and M. Ringnér. Matching protein structures with fuzzy alignments. Lund University preprint LU TP 02-39 available at <http://www.thep.lu.se/complex/>, submitted to *Proc. Natl. Acad. Sci. USA*, 2003.

[11] J. von Neumann. Some matrix-inequalities and metrization of matric-space. *Tomsk Univ. Rev.*, **1**:286–300, 1937.

[12] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**:443–453, 1970.

[13] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Res.*, **28**:235–242, 2000.

[14] E. Lolis, T. Alber, R.C. Davenport, D. Rose, F.C. Hartman, and G.A. Petsko. Structure of yeast triosephosphate isomerase at 1.9-A resolution. *Biochemistry*, 29:6609–6618, 1990.