Okay. I was informed that you have posted presentations today and was there another submission due today as well, or was it just the posters? Gosh. Okay, so I'll take it. Not too personally. But I think I'll spend maybe 2 or 3 minutes doing small talk with you in case anyone else would like to join us. You have to remember not to start running with the mic on, because apparently they are tacked to the room they belong in. So if you walk too far away, they actually give up some kind of alarm and then a technician is running after you in case of trying to steal the mic or just accidentally took it to your office. So I have to try to not walk too far away from the room. I don't know how far I can walk. Okay, so considering this is the last lecture, we'll have a bit of a relaxed one. But we still have one more model to cover today. And I think it's one most people actually look, we're looking forward to hearing more about because it's one of those that have been present in the news a lot, especially when we talk about deep learning as a concept. So we will talk about neural networks today I will talk briefly about deep learning, what it actually means in that context and in which kind of settings it's useful, or when it's a bit. Yeah, overhyped maybe, or not as useful as one might think. So it all comes back to this idea that we were talking about in the very first lecture, that each model really is only as good as the context that you're using it in. So each problem that you encounter will depend on what kind of data you have, what kind of objective you have, what kind of relationship you're thinking there might be present in the data that you're doing, trying to predict and depending on what kind of what kind of answer you give to all of these questions, different models will give you better fits, different fits, be computationally more efficient, or just not make any sense to use at all. So I've seen, for example, deep learning be applied to problems where you have small data sets and very simple linear relationships. And we'll see why that doesn't make any sense. Because the big advantage of neural networks, and what I want you to keep in mind when you hear about them and when you think about applying them, is they are inherently nonlinear. And that is a really, really big advantage of them. We'll see why, how exactly they are structured, but in that sense they are similar to support vector machines and to things like K and N random forest decision trees in general. All of these methods were nonlinear. So linear relationships are actually, I wouldn't say quite rare, but purely linear relationships is not something you will encounter very often. And if you do encounter a very kind of straightforward linear relationship, in most cases, a linear regression is all you need, because you will also see that many of the more complex methods, like neural networks, are just kind of linear regression on steroids. So we will see that neural networks take this concept of a linear regression, start combining it in all kinds of sorts of weird ways, pushing it through nonlinear functions. And that gives you a non-linear result. But a lot of the concepts that they use are familiar to you. So if you understand linear regression, then neural networks are kind of a step further from that and a way of making them very much nonlinear. So last week we were talking about support vector machines. I hope you remember that. I was very proud of my way of explaining the kernel trick. I think I'm on to something there. Maybe I should become some kind of YouTube or TikTok professor or something like that. Yeah, I don't know whether that's a good idea, but you might see me in a couple of years doing exactly that. So you remember support vector machines, this idea of kind of finding finding this linear decision boundary so it can be a line or kind of a hyperplane. And still being able to divide non linear relationships in the data by using the kernel trick. This whole idea of kind of pushing some kind of function through your linear space, pushing some part of the data into a new dimension, additional dimension which allows you to use a linear decision boundary. In that context, we were talking about maximum margin classifiers, optimisation classifiers in more general, and we were talking about the support vector classifiers and support vector machines. You'll remember I said support vector classifiers are basically SVMs, but typically we use that term when we have a linear relationship. So the strict term would be a linear support vector classifier. But no one really says that. You just say it's an SVM and kind of using a linear kernel or no kernel at all. So this week we'll talk about Anns a neural networks. We'll talk about single layer feedforward networks, the most simple and basic structure. We'll talk about deep multilayer networks and how they relate to deep learning. And we'll talk about network regularisation and tuning. So keep in mind this is a very basic introductory lecture to neural networks. There are methods which are much more advanced which built on top of them. So you might have heard about for example, recurrent neural network. Works or long term short term memory networks, all of these more kind of advanced structures. But in order to use them and understand them, you first have to understand the basic structures, and we will put a bit of a focus on when to use them and how to use them in different contexts. See, it was worth. Kind of doing a bit of small talk for a couple of minutes and waiting a little bit. So good timing. I said earlier, neural networks are inherently nonlinear, so they can be used for both classification and regression. And we will see how exactly. But you can think about it a bit like linear regression for regression sense and then a bit like logistic regression for a classification case, because we'll use a very similar trick and basically a sigmoid function or something very similar to a sigmoid function to get a probability for different class labels. And that way we can use them for classification. So think about it like linear regression. Logistic regression just big. The name neural network actually does relate to a brain. So we are trying to emulate how our brain functions. And a lot of the terminology that we use for that is similar to neurology. So we'll talk about kind of nodes and that are connected in this neural network in your brain. And that being said, I'm not a neurologist. I have no idea how a brain works. I don't think most biologists really do. So don't think that this is actually how your brain works. It probably isn't. It's an extremely simplified version of how similar it might look in theory for very simple tasks. To the surprise of many neural networks were actually really famous in the 1980s. So that's 50 years ago, and people were really excited about them. So everyone thought, oh, this is the next big thing. It's like a brain. So surely this is AI, as is peak, and this is what we should all do. And then very quickly they kind of fell out of favour. So I think for the 90s, early 2000, no one was using neural networks. The very simple reason for that is they have really expensive. They are so computationally expensive. You'll see later why that when we had things like SVMs and random forests and all of that, they worked just as well, sometimes better, and they were cheaper. So why should we used as neural network, even if it looks like a brain, if it doesn't work very well? So we didn't really kind of use them for a long time. And I think two reasons. One was I as a right here, they require a lot of tinkering. And this is related to computational cost. Neural networks have a lot of parameters. And I mean a lot of them like thousands even for small networks and kind of SVMs, as you've seen earlier, really had these two things that you can think of with, which is the kernel that you want to choose, and you kind of work your evaluation or costing parameter c. So it's much, much easier to tune them to suit your problem versus a neural network where you have to adapt so many small parameters and it's just more expensive. And in many cases SVMs just perform much better, as do random forests. So recently, and I think recently I have to change that for next year because it's not that recently anymore. And people started pushing this method again. Big reason for that. We just have more data, we have better computation, we have better computing architecture. And it's just kind of getting more feasible to train neural networks. I think another reason for that is a lot of the open source coding community started pushing coding for that. So in our lab we will use mostly Keras, which is based on the TensorFlow kind of library or environment, and making this open source and public kind of as a coding environment and library for people to use, made suddenly made neural networks really accessible for people. So it's really kind of, you will see quite easy to implement your own neural network as long as that relatively simple structure, like a simple feed forward network. And it's basically you can just specify the different layers, how many nodes you want in them, and then it's really kind of five lines of code. And you have a perfectly working neural network for your problem. So this kind of environment, these kind of libraries really pushed people to to start tinkering with them and trying them out again. So yes, they resurfaced around 2010, maybe a bit later than that after increased interest in deep learning in particular. So new architectures. We had a few success stories, especially in the area of image classification, video classification. I think this is where neural networks really shine, these extremely kind of complex problems, and also in speech and text modelling. So the AI community, especially in natural language processing, these are they are often interested in these very non-linear complex approaches like neural networks. So let's look at one. I said earlier that a neural network. As it's at its core. It's very similar to a linear regression. And you'll see why. Let's say we have some kind of input vector. That's just your data again right. It's your x. And we we're looking for some kind of non-linear function with which we're trying to predict some output value. So it's a regression problem right. We have data. We're putting it in. We want some kind of value a number. And the structure of a neural network now looks like this. So you can see we have something which we call an input layer in yellow here. That's your data. And we have our output. That's your value y here at the very end and some kind of function. And we think this function will then transform x into our value y. The interesting thing about our new network is this layer in the middle, and how it's connected to both of these sites. So you can see this is actually quite complex. Already we have each of these input values going through each of these hidden layers, the nodes within that hidden layer, and then being transformed through some kind of non-linear function into our output. You can also see that these arrows are directional. So we're talking about a feed forward network because we are feeding data forward through the network from the left to the right. There's a bit of kind of backtracking and back propagation going on with the optimisation step. But in this very simple idea, taking data, pushing it through the network, getting a value out at the very end of it. So here's an example. We have four predictors. So these are our kind of four values. Our input values that we're putting in. We're feeding them through. We have k hidden units here we have five. That's the number you can choose. So you can choose how many of these hidden units you want in your layer. More units make it more complex can be better for more complex nonlinear relationships. Less units make it cheaper and easier. Better for simpler relationships. It's very similar to other models that we've seen in the past. So if we actually have a look at what goes on computationally, it becomes more clear why we say or why I say it's similar to a linear regression in its core. So you can see that we still have a sort of linear combination of factors. So you can see these vectors again they can remind you of your linear regression betters similar idea. So we are estimating some kind of coefficient. And we have our input value x here. And we're trying to estimate some kind of output value y with that. And the problem is that we then kind of starting to layer multiple of these linear combinations within each other. So for example we have our hidden notes a. So these are our five hidden nodes. Within each of these nodes we have our linear computations going on. So each of these vectors or each of these values is fed into each of these nodes neurones. Then computation and compute it through linear combinations and then fed forward through that again. So the idea is relatively linear. However we are using a nonlinear activation function within each of these nodes. And this is where the non-linearity comes in. So this nonlinear activation function can be for example a sigmoid function or a polynomial function. So the core is it has to be some kind of non-linear function that is being applied to a linear combination of our input features. And this is the core. What kind of differentiates it from a nested system of very complex linear regression? So. And then in the end, we then feed this hidden layer, the last hidden layer into our output layer, and we predict some kind of value. So remember our A here where all our hidden layers, all our hidden nodes. So each of these little A's we have again some kind of parameters because we're combining all of these A's together to predict a value. So yes, it's similar to a linear regression model. But we have five neurones. And these neurones are in turn nonlinear transformations of linear combinations of your input data. So that's kind of why it gets so complex. Because suddenly we have already quite a few parameters. We have the parameters better which you can see here. For our output layer the very end. And we have all of these W's which you can think about as weights. You can think about them as kind of how much weight or how much proportion of your X value is in each of your nodes. So it doesn't it's not always equal. So it's also very much unclear. And you don't really know what exactly is going on. But you know that each of these A's has some proportion and some impact for each of our X. And these are the W's that you're then estimating. So you can see that a lot of estimations, because it's already a lot of parameters for a really small network. So remember we said we have four input values five nodes and one value we're trying to predict. So we're always estimating a lot of parameters for a very very small network. I mentioned earlier that these activation functions, that's kind of the nonlinear part in each of these linear combinations can be, for example, a sigmoid. You remember sigmoid functions. That's what we were using for logistic regression. That was what we use to do to convert these linear functions into probabilities instead. Nowadays we typically use ReLU which stands for rectified linear unit activation functions. They're very similar to a sigmoid, so don't worry too much about the difference. I put a difference up here to explain it. You can see it has basically the same idea. So you can have a value of zero until some kind of certain threshold, and then you rise up rapidly to the value one after the threshold. So it's similar to a sigmoid or sigmoid here in green it's just a bit smoother. The main difference and the main reason why we use a ray nowadays is it's cheaper to store computationally. Because you can see here all these values are zero. And then here we would just continue that all of these values are one. So you have these values here for example less to store. So it's computationally cheaper. And it has almost the same effect in kind of the sum of it. So that's why we would use this function instead of the sigmoid. That being said, if you use a sigmoid instead it doesn't really matter that much. Some people still hold on to those old ways of doing it. And as I mentioned earlier, it is a relatively rapidly evolving area in field. So when I talk about, oh, it's kind of 2010 is so long ago and probably is, it feels like yesterday, but it's a long time ago and we're still developing is even further. So probably in five years we've changed our minds again. But for now this is what we stick with. So he is. Yes. In summary, the model is a network in this case with five hidden neurones. Here is this kind of terminology. Again you can say nodes or units or neurones or whatever you prefer. And we derive five new features by computing five linear combinations of x squashing them through some kind of activation function. And then the final model is linear in these new derived variables. So remember that this is linear again. So you basically have linear combinations of non linear transformations of linear combinations if that makes sense. So this is where this non-linearity comes in. So I actually have a computation example here. And I think the main reason why I have that is because I want to show you why it's actually nonlinear from a mathematical standpoint. So here we have quite simple example where we have only two input variables. So x one and x two. We have two hidden units. These are our kind of neurones in the middle. And we call them H1X and H2X. And we define some kind of non linear transformation squashing function or activation function. I like to call them squashing function because I just think it's kind of a funny idea to squash your linear combinations. Let's say we already estimated all of our different parameters. So you will see that estimation of these is computationally quite expensive. Let's say we have one for all of that because we want to see what these calculations would look like. So we know this is our non linear function g. And we're transforming with that our linear combination. We have our weights w. Remember this is basically the impact of each of our variables on our neurones. And we we are calculating this for both H1 and H2. So these are our two hidden neurones. So you will see that if we actually do that do all of this pretty calculation step where we just plugging in our beta values, which we've already said we've estimated magically. You will see that in the end you actually get x one times x two, which of course is non linear. So you can see that the sum of these two non linear transformations gives us an interaction between these two variables instead of just a simple linear combination of them. So suddenly if we're estimating our value y we can do so as this interaction between the two variables instead of just for example x one plus x two, which which you would get in a linear regression. So by having these non linear activation functions in there which transform your linear combinations, you are able to actually capture these non-linearities and these really interesting interaction effects. So yes, we just said we have these parameters magically appear. But of course this is not the case. We actually have to estimate them. The good news is the process the same as you've already used to? So we still have estimation through some kind of error loss function. In regression problems we still minimise our error. So this is still our true value minus our estimated value. We square these errors. We sum them all up. Same thing we always do for classification. It would also be the same thing. So you still can estimate just the error in the classification. Let's have a look at what this looks like. Multilayered, because I think this is the interesting part. You've seen now just a single layer neural network. And you already saw this is a bit of a complex thing, but nowadays most neural networks will have more than one of these layers. So more than one hidden layer. And in many cases you have a lot of units. So a lot of nodes in each of these layers as well. So this is a bit of a generalisation. Networks with multiple layers can be referred to as deep neural networks. The deep refers to the depth as in how many hidden layers you have, so people will have differing opinions on how many layers does it take to call network deep? Obviously, because no one can actually agree on that. And then there are people who come around the corner and say, yes, well, this is a very deep neural network and an even more deep neural network, etcetera, etcetera. But generally the number of layers refers to the depth. And very generally, as soon as you have more than one, you could call it a deep neural network, depending on your definition of deep. So in theory, what we just saw was the single layer. You can already model most of these complexities and most of the non-linearities, but in some cases it's actually easier to use multiple layers with fewer nodes instead of one layer with a lot of nodes. So in many cases, in order to actually get to the level of complexity that you desire, you would have to have a very kind of broad layer instead of instead of a deep one. So instead of having kind of a lot of nodes, you can have smaller layers and then just stack a couple of them and push through all of these, your data and it will have a similar effect. And in some cases it's actually giving you better results as well for your solution. So I do have a figure for such a multi-layer network. And I explained this with a very famous example for neural networks, which is image classification. I mentioned earlier, neural networks have a very kind of specialised niche that they really work well in. One of those is time series. So time series prediction is actually one of the areas where neural networks perform really well. You will often see them in a business context, for example, in finance where you have very long time series and you're trying to predict the next step in that time series, for example, for stock market prediction and new networks can perform quite well on such tasks. The other area is image classification. So this is a really famous example where we're trying to predict handwriting. In particular, we're trying to predict the numbers that someone is writing on paper by looking at the images of those numbers. So here I give you a rather big two layered network. So we have two hidden layers in there. One has 256 units, the other has 128 units. Now you ask me why exactly these numbers, and you can probably imagine what my answer will be. While it kind of worked out, in that case, you will have to cross, validate and hope for the best. And so this is a rather big, big one compared to what we had earlier with just five, right? So we will see how computationally expensive it gets. I also said we have numbers so we will have ten output variables. This is actually a classification problem. We're not trying to predict a single value as an aggregation. We're trying to classify for the numbers 0 to 9 what the handwriting refers to. So these kind of multitask learning systems can be really formed well with neural networks in one go. We're talking about SVMs and how they kind of can be a bit tricky if you have more than two classes, so you can use them, but it's typically good for binary case logistic regression, same thing. Logistic regression. Typically binary output can be kind of stacked and combined for multiclass but kind of one goal. Multiclass prediction is really good with a neural network. This is what it would look like. Obviously, it's not a full one because I can't draw that many notes for you, but this is actually the structure that it would take. And then this would just have 200. What did I say 56 and 128 of these blue neurones. So you can see we have our input layer. This is our data that we're pushing through first training then the test. Same thing that you always do. And then we have some kind of output layer which I want nine numbers. So probability that this is number zero and probability that this is number nine and everything in between. So we're trying to predict a probability again as I said earlier in linear regression kind of refers to a regression network logistic regression similar to a classification neural network. You can also see these our weights. Remember W was the contribution of the input to each of these nodes. And then we have our betas. And that was how we then actually transform that to get our output value at the very end. Otherwise it's very similar to what we just had. The big difference is that you can see first we start feeding in the input data into each of our layer, into each of our nodes here in the first layer as before. But now the output from that is not taken as the direct output for the number, but actually as the input for the next layer. So this is the key difference. And this is where we kind of start stacking non-linearities within each other. So it's a nested system of non-linearities. And then at the end we do the same thing again. And we actually have our output layer where we have our non linear transformation. In this case we would for example use sigmoid or something similar because we want the probability for that. So the first layer will look familiar, because that is the same thing that we've already did earlier. We have our weights this time. I index them with a little one to indicate this is our first hidden layer that we're feeding that in. We have our normal linear combination, some kind of weight system. You can think about them as the betas in your linear regression, our input data X and our non linear transformation function g. So this is all being fed into our first hidden layer which we call a one. Remember k is the number of nodes that you had in this layer. Then the second layer treats the output from this first layer as the inputs. So in this case you can see that this basically adds a bit of complexity into that, but it doesn't look too bad. The only difference for now is that this a is this term. So I wrote the simplified version here, but you can imagine that this a here a one is actually the whole expression that you have up here. So that's why we say this is a kind of nested system of non-linearities. So you have nonlinear transformation G1A nonlinear transformation g two basically. Otherwise same system. We have our WS indexed with two because they are now the weights for our second hidden layer a two. So yes, the important thing is that each of these units in the second layer is still a function of your input vector x. So you're still kind of feeding x into the second layer, but you're feeding a non linear transformation of a linear combination of x into that layer. So it is a function of A. But these are functions of x. So if you think about that what I just said is if you take this whole term and put it back in here, you again have your X as your input data even in your second hidden layer. And that would also be the case if you have more and more layers so you don't have to stick to two. You could have hundreds of layers in theory, and they would all be nested combinations. So it would just go on and on and on. And you can also see that this will lead to very complex transformations of your X. And in the very end, you still have this feeding into the output layer that you had before. It's just now more complex transformations. This is also the reason why neural networks are black box models. Honestly, as soon as you start feeding the data through so many layers of non linear transformations, different weights, different kind of amounts of x in each of these nodes, you have no idea what's going on. At some point, it's just impossible to track how X is being transformed through all of these steps, and then kind of finally ends up in your value y. So neural networks are one of these black box models where you're not really sure why they work, but you know how well they work because you can still calculate the error of the model at the very end. The only way of kind of translating a little bit is if you look at things like feature importance. So we were talking about that for other models as well a little bit. But for neural networks this is really the key kind of the key concept or the key way of still understanding to an extent what is going on. Feature importance can be thought a little bit in my opinion. Like when we were talking about PCA, we were talking about this kind of loadings. So what was the loading of each of these variables on the principal components that we were constructing? And I like to think about feature importance in a similar way. What are the loadings of the variables in the model. So what kind of impact does each of these variables have on the construction of your Y at the very end? You can't know that for certain, but it can give you a general idea. So we'll look into feature importance a little bit more during the doing. The computer lab. That's the word during the computer lab tomorrow. Gosh, I'm just running on caffeine today. Not enough sleep. Okay. So let's look at what this would look like for our number example. So we said we have this two layer network 258 or something. 285 I don't remember, and 128 something like that. So these are our two hidden layers that we have. We're trying to classify digit images. So someone wrote numbers. And our neural network is trying to predict or trying to understand and model what these numbers are. And. The way we do it is we have a look at these images. And in this case they measured 28 by 28 pixels. So this is images of size 28 by 28, which gives you 784 pixels for each of your samples. So you can already see okay, this is this is a lot of data. And this is also the reason why neural networks work so well with image classification. You have a lot of data in those. So let's say our first hidden layer has 284 units. Then we calculate the weight of the matrix of weights w one, and that will have 285 number of pixels plus one times 200 miles, or 256. Probably because I can't count. What did we say earlier? What is the size of a layer? 256 MB. So it should be 256. So 285 times 256. Number of units in this layer will give you 200,960 WS. That's a lot. So note we have 785 here instead of 784 because we have to account for the intercept. Remember in linear regression when we are feeding the whole data in that and sometimes the model tells you I can't do that because I need an intercept to actually function. Same thing. You have to add an intercept to that. So this is just like your random term or your constant term with basically how your model is anchoring itself. So we set each of these elements so each of these nodes to 184 of them and feeds into a second hidden layer. So we have 128 in that. So we have our second double here of size 32,896 elements. So this is your W two. So now we also have our output layer. That's the actual prediction part right. So it's very similar to a logistic regression basically where we have ten responses rather than one. And we the way we do it is we calculate ten different linear models for each of these possible responses. And in order to actually find out what all of our better values could be. So we have 1290 1290 beta values just for the kind of final estimation step. So. We said earlier, we're trying to predict probabilities. So very similar to a logistic regression. We need some kind of squashing function, something to actually translate it into probabilities. Here we use a softmax activation function looks very similar to a sigmoid. It's just very small shape differences between all of these. The reason why we use softmax instead of I don't know any anything else really is people just started doing it and then other people adopted it. And now it's just what be using. Okay. So don't try to question it too much. Sometimes research works in mysterious ways. It works. And you can think about it just like you did about logistic regression. Okay. So we are looking for class probability. So how likely is the sample to belong to this class of number one or number six or whatever. And we're calculating these probabilities using this activation function. And so yes our goal in this case is to build a classifier. So we're trying to predict an estimate probability for each of our classes. And then we assign the image to that class which is the most likely. So the highest probability gets the sample. They are basically voting on who is the most likely to actually get the sample for it. The other thing that I wanted to mention here is how do we actually estimate these? These coefficients. This is. In this case because it's a classification problem. We would use something called cross entropy. If you haven't heard about cross entropy before, you might have heard about maximum likelihood instead. So maximum likelihood. Cross entropy is basically two sides of the same coin. So they're basically a very similar concept. The basic idea behind cross entropy is that we're looking at the difference between two probability functions. So in this case here we basically look at the difference between the probability functions of all of all of all of our class probabilities. Gosh. And we're trying to minimise that. So we're trying to find the most likely class for it. Think about this concept very similar to how you think about minimising the error when you're fitting a linear regression. So it's all about minimising some kind of cost function in order to find the best fit model in terms of its parameters. The parameters are a little screws that you can kind of change in order to change the fit of the model, and obviously you want to fit it in such a way that it fits closely to your example data, but not too closely, because in that case you would have an overfit model. So why did I tell you all of these numbers? The reason is that I want you to have a look at this number. 200,000 235,000 and 146 parameters to estimate for your image classification problem. So I don't think I have to tell you that this is a lot of parameters and it's very computationally expensive. The other thing I want you to think about is I think we briefly touched upon that, but maybe not enough detail. In order to estimate this high of a number of parameters, you need a lot of data. Typically, you want the number of samples that you have for your training to be higher than the number of predictors or estimators or whatever. You have parameters, whatever you have to estimate. So typically you want more data. Then you have to do estimations. That's not always possible. Some models are more robust to that. I think k and n for example, is more robust to having smaller sets of data compared to a number of predictors. But typically you want. A lot of data. If you have a lot of parameters, that's kind of the baseline that you should look for. So in order to estimate over 200,000 parameters, you would also need a lot of data. And in order to train a lot of data you need a lot of computational power. So all of this is contributing together to why neural networks are so expensive to run. Okay, so I think this is a fairly good place to take a little break. So let's take ten minutes and then come back for the second part where we talk a bit more about fitting the neural networks. And then at the very end, I also wanted to take maybe ten minutes to go over the exam. Given that this is our last lecture and any questions that you have. Okay. So let's meet here again in ten minutes. I'm going to get. Away with that one. Should we get started again? Let's spend a little more time with our neural networks for the next half an hour or so, and then there will be plenty of time for questions about the exam and anything else that you have that you would like to ask me about the cause in general. So what we know. Okay, so we just said. If we imagine this very simple single layer neural network that we had at the very beginning. So you remember that we had these better parameters that were kind of feeding into the last output layer, and we had our w parameters which were feeding between the input and the hidden layer that we had. So you can see that we will have some kind of error minimisation problem again. Right. So I said earlier in a classification problem we're trying to minimise in this case for example cross entropy. And and if you have a regression problem we would return to least squares as our optimisation method. So it is very similar to linear regression but it's non linear. So we're looking at a method of nonlinear least squares. The idea is still the same. We are still trying to minimise our error. And our error is still defined by how much or how far away is our estimation. That's our f of x from our known true value y. So we're still doing the same thing. We're still squaring all of these errors. We're still summing all of them up. The big difference is that you can see here we suddenly have a lot of parameters that we can kind of tune a little bit to find the best fit. So in the past this was only your better for your linear regression. You could just change your betters. Now you can also change your W's and all of your betters. And another big difference is that this f of x is non linear as we've seen earlier. So this is the formula we had at the very beginning where we had this non linear squashing function g which was doing something with our linear combination of weights and data input. So very similar idea. A big problem is that this objective function looks quite simple. As I just said, it's the same as linear regression, right? But the problem is that these parameters are so nested. So the more layers you are actually adding, the more we have these nested combinations of parameters which basically lead to this becoming quite a complex problem. The big problem in that is that this function of minimal combination of parameters, which we're looking at here, is non convex. What does that mean. It's very difficult to actually find the perfect minimum, the perfect kind of combination of parameters which would minimise our ABA function. So if you imagine here for example we have a combination of parameters. So all of our parameters. And if imagine we're all combining them because we're all estimating them in the same kind of in the same goal. Right. So if we have all of these and we're looking for whatever is the minimum, the resulting minimum of our loss function, you can see that there's basically two possible minimums that you might reach and one is better than the other. So what we say to that is there are multiple local minima and it's difficult to find the global minima. So the overall best combination of all of these parameters. This is the big challenge that we have. There are two general ways which we can use to kind of avoid this being stuck in this local minimum, never finding the perfect parameters. One is what we call slow learning and in combination with early stopping. So we are kind of using a very slow, gradual search for the best parameter combination. And we stop early to avoid overfitting. And the other is our good friend regularisation. I told you it's so similar to linear regression. We also look at regularisation again in this context. But let's start with this idea of slow learning and early stopping. I think I mentioned gradient descent in our lecture on linear regression. So that's lecture five. I probably mentioned it very briefly that the idea of gradient descent is we're looking kind of for this hill that we're going down to find the minimum, right. So we have some kind of best guess for the parameters. And then we're iterating until the objective function doesn't decrease anymore. So we're finding some kind of vector which finds a small change in our graph in our error rate. How do we express explain that. The idea is basically we're finding some kind of combination. For example, here the are the first are here, and then we're finding a way of going down the hill until we reach hopefully the best value, the hopefully the best combination of parameters that we have. So the question is obviously, how do we find the best direction to move in? And the idea for that is that we're calculating the gradient. Why the gradient? Because it tells you basically in which direction you should follow the hill to minimise your function. So that was what I was doing here in this point here, for example, I would calculate last opportunity for me to draw in this course. Looking forward to that. So I would calculate the gradient in this point and I would see okay this gradient is telling me go into this direction and I would follow that, calculate the next gradient, keep going into this direction. And I would go and go and go and go. And if I, for example, go a bit too far, then a new gradient in this area would actually tell me no no no no no, go back, go back, go back. So hopefully going to kind of following this back and forth at some point will give me our minimum, which is exactly what you're looking for. So remember we're finding the minimum in the error. So the minimum and this term here minimisation minimum in the error as a combination of your parameters. So how do we change the parameters in such a way that we're minimising our error. And yes, the idea is that we move a little bit into the direction which allows us to go downhill, so downwards towards the minimum of our error. You've also just seen what happens if our learning rate. So step size is a little bit too large. So I said okay what happens if I go a bit too far. And suddenly on the other side of the, of the, of the, of the valley that I was looking for. So in that case I was kind of jumping a bit too far for that. So you want to find the step size in such a way that you actually find this local minimum without always jumping back and forth, kind of above this valley that you're looking for. So remember here, if my step size is to back to pick of a jump from one to the other side and never truly find this value here. So that's the that's the danger of having a too big of a step size. On the other hand, if you have a very small step size, it would allow you to go down here, but it would take very long because you would have to calculate and calculate again and again again. So choose wisely this kind of value. And I always say choose wisely. And it's such a difficult thing to do, isn't it. But in most cases it's a little bit of a matter of experience and testing different values, seeing where you end up with. So if you have a step size and then you reduce it a little bit and suddenly you find a better minimum value, you know, okay, you were jumping across the valley. The other thing is. I was quite lucky to start here, wasn't I? Because if I had started here, it would have told me to go into the other direction. So this is the big problem which we have with gradient descent. It often only finds the local minima, and it's dependent on this random first value that you start with. So what did I want to say? Yes. So? This random first value. Because the problem is that we then sometimes only reach the local minima. But mostly doing in real life is we're not calculating that just once, but we're actually we're actually taking a sample instead. So here, for example, you can see that we don't look at all of our parameter combinations for all of our observations in, but instead we only use a small sample of some kind of batch of those observations and then compute our gradient steps. And then we basically. By combining these different steps or by combining these different fractions of the data, we would create something which we call stochastic gradient descent, which allows you to basically introduce randomness into the process. And as we've discussed multiple times, introducing randomness typically improves in some way your optimisation. And that is the case here as well. So stochastic gradient descent has the big advantage that it can help you overcome the problem of local minima. So I've just jumped over a couple of slides. And the reason for that is. We said the previous calculation. So basically finding your error, finding your. Finding your gradients, following your gradients, and recalculating your average at each of these steps is not very complicated. And the problem is basically just that it takes a lot of small calculations to do. I put these beautiful calculations here up for you in order to demonstrate how relatively easy each of these gradient calculations actually are. And I've also done so because I want you to have a look at this term here. So this is the interesting part because it shows you that in each of these calculations we are considering the residuals. So the errors that we're actually taking into account, which makes sense because we're trying to minimise the error rate. So the act of differentiation. Still keeps this fraction of the residual in each of these. In each of these calculations. For each of these parameters. And this is basically what we call backpropagation in the literature. You might have heard about back propagation, and I hated this weird explanation for it, but it's the mathematically correct. However, instead of taking my word for it, I will. Use the wonders of technology because I could draw all of this, but someone else has done it for me, so I will utilise their explanations. Um, in person. So this is a really, really good video on how backpropagation works, colour coded in a network, which is the reason why I'm showing it here. Let's have a look. So I'm going to start things off here with a complete disregard for the notation and just step through those effects that each training example is having on the weights and biases. Because the cost function involves averaging a certain cost per example over all the tens of thousands of training examples, the way that we adjust the weights and biases for a single gradient descent step also depends on every single example, or rather, in principle it should. But for computational efficiency, we're going to do a little trick later to keep you from needing to hit every single example for every single step. So this is what I meant by we're taking the residual into account in each of our gradient descent calculations. Right. That's what he said. We take into account each of our samples and each of these steps in theory, but computationally, we're not doing that really. In either case, right now, all we're going to do is focus our attention on one single example, this image of a two. What effect should this one training example have on how the weights and biases get adjusted? Let's say we're at a point where the network is not well trained yet, so the activations in the output are going to look pretty random. Maybe something like 0.5, 0.8, 0.2. On and on. So remember these are your your probabilities. So we were talking in classification. We had these probabilities. So for an example of two you have for example the probability 0.5 and one. So you can see these are not really good probabilities. It's actually not estimating that this is a sample for number two. And you will see now how the how this single training example is going through the network and adjusting the parameters. That's the tuning part in order to improve a little bit. And this is then done with each of these training examples. Now we can't directly change those activations. We only have influence on the weights and biases, but it is helpful to keep track of which adjustments we wish should take place to that output layer. And since we want it to classify the image as a two, we want that third value to get nudged up while all of the others get nudged down. Moreover, the sizes of these nudges should be proportional to how far away each current value is from its target value. For example, the increase to that number two neurones activation is in a sense more important than the decrease to the number eight neurone, which is already pretty close to where it should be. So zooming in further, let's focus just on this one. Neurone, the one whose activation we wish to increase. Remember that activation is defined as a certain weighted sum of all of the activations in the previous layer, plus a bias, which is all then plugged in to something like the sigmoid squish ification function or a ReLU. So there are three different avenues that can team up together to help increase that activation. You can increase the bias, you can increase the weights, and you can change the activations from the previous layer. Focusing just on how the weights should be adjusted. Notice how the weights actually have differing levels of influence. The connections with the brightest neurones from the preceding layer have the biggest effect, since those weights are multiplied by larger activation values. So if you were to increase one of those weights, it actually has a stronger influence on the ultimate cost function. Then increasing the weights of connections with dimmer neurones, at least as far as this one training example is concerned. Remember, when we talk about gradient descent, we don't just care about whether each component should get nudged up or down. We care about which ones give you the most bang for your buck. This, by the way, is at least somewhat reminiscent of a theory in neuroscience for how biological networks of neurones learn. Hebbian theory often summed up in the phrase neurones that fire together, wire together. Here, the biggest increases to weights, the biggest strengthening of connections happens between neurones, which are the most active and the ones which we wish to become more active. In a sense, the neurones that are firing while seeing a two get more strongly linked to those firing when thinking about it too. To be clear, I really am not in a position to make statements one way or another about whether artificial networks of neurones behave anything like biological brains, and this fires together, wired together idea comes with a couple meaningful asterisks. But taken as a very loose analogy, I do find it interesting to note. Anyway, the third way that we can help increase this neurones activation is by changing all the activations in the previous layer, namely, if everything connected to that digit two neurone with a positive weight got brighter, and if everything connected with a negative weight got dimmer, then that digit two neurone would become more active. And similar to the weight changes, you're going to get the most bang for your buck by seeking changes that are proportional to the size of the corresponding weights. Now, of course, we cannot directly influence those activations. We only have control over the weights and biases. But just as with the last layer, it's helpful to just keep a note of what those desired changes are. But keep in mind, zooming out one step here. This is only what that digit two output neurone wants. Remember, we also want all of the other neurones in the last layer to become less active. And each of those other output neurones has its own thoughts about what should happen to that second to last layer. So the desire of this digit two neurone is added together with the desires of all the other output neurones, for what should happen to this second to last layer, again in proportion to the corresponding weights, and in proportion to how much each of those neurones needs to change. This right here is where the idea of propagating backwards comes in by adding together all these desired effects. You basically get a list of nudges that you want to happen to this second to last layer. And once you have those, you can recursively apply the same process to the relevant weights and biases that determine those values. Repeating the same process I just walked through and moving backwards through the network. And zooming out a bit further, remember that this is all just how a single training example wishes to nudge each one of those weights and biases. If we only listen to what that two wanted, the network would ultimately be incentivised just to classify all images as a two. So what you do is you go through this same backprop routine for every other training example, recording how each of them would like to change the weights in the biases, and you average together those desired changes. This collection here of the average nudges to each weight and bias is, loosely speaking, the negative gradient of the cost function referenced in the last video, or at least something proportional to it. I say, loosely speaking, only because I have yet to get quantitatively precise about those nudges. But if you understood every change that I just referenced, why some are proportionally bigger than others and how they all need to be added together, you understand the mechanics for what backpropagation is actually doing. Okay, I hope you can see why I could not draw that. The key thing that I would like to take away from this is one. Those videos are really good, so check them out. And the other thing is, the idea of back propagation really is about how. The structure of the network impacts your optimisation process. So we could see that it's relatively easy if you think about it in a linear regression where everything is just kind of fed forward once. So all of these weights, all of your betters can just be adjusted by you once. There is no need to think about what happened in the previous step or anything like that, but in a new network because because each of these nodes is a linear, non and non linear transformation of a linear combination of what happened before it, suddenly you have to think about how do we take into consideration all of that as well. So a neural network in theory, as I said in a very beginning is feeding data forward through it, at least the structure of a neural network we're discussing here. But then we are adjusting the neural network. We're optimising it backwards because that is how the relationship of each of these nodes work with each other with the previous one. So this is the basic idea of backpropagation feeding data forward, adjusting weights, thinking backwards. And I think this video also gave you a very good idea of how complex that is computationally. So that was the example for one training example the one number two. So that training example. In that case what's classification. It works the same way for regression task. The only change is to the cost function which you're trying to optimise. Otherwise it's the same idea. So all of these calculations are straightforward in themselves, but become complex when combined and when trying to keep track of all of that. Okay, so that was the idea of gradient descent, of how to optimise the combination of parameters that we have in order to minimise some kind of cost or whatever or whatever we have. We also said that similar to regression, we can have a look at regularisation instead, and there are a couple of ways of doing that. We've seen earlier in our example for digit recognition, we had over 200,000 of these weights. So you can imagine if we're actually optimising that and fitting that perfectly to our data, there is a huge risk of overfitting to our data. So neural networks are really prone to overfitting, especially if you're not careful with regards to regularisation for example. So similarly to decision trees, similar structure really as well where you have kind of a tree like structure, we're training it to the side very similar and same idea. If you have a single neural network and you're just optimising it perfectly, it's probably really overfitted the same as a single decision tree, but you're virtually optimising for probably overfitted. So one way of avoiding that is using our good friend which or lasso regularisation. Again, you remember that for linear regression all of that comes back to haunt you. And so we are introducing some kind of error term again. So we have some kind of penalty term. You remember lambda lambda was your regularisation parameter. And you choose that to minimise basically the number of parameters. In a way you can also use lasso regularisation, which the same as linear regression can lead to some of these parameters actually becoming zero. So some of these weights will then just have zero impact in your neural network, which reduces just the amount of overfitting that the network might do. So this is the same principle that we had for linear regression. You will recognise that. The only difference is that here we have cross entropy to optimise instead of our instead of our error term. Otherwise it's the same principle as regression. The other thing I wanted to mention is to about learning, because this is relatively, I mean, not new anymore, probably, but still extremely popular because it works very well. And the idea here is that we basically just randomly remove parts of the network. So we said the network itself is really complex. I was asked in the break, do each of these nodes connect to each of these nodes? Yes, they actually do. So you can see this node here, for example, is connected to all of the nodes in the hidden layer. In some cases it makes sense to deactivate some of these nodes. So here for example we are deactivating two of these neurones in the hidden layer and one of the neurones in our input layer. And we still receive a really good estimate probably which is not as overfit. So you can think about this a bit like this is basically feature selection isn't it. So we are selecting which part of the input vector are actually fed into our network. And this is a bit like lasso not really like lasso regularisation, but we are basically removing some of these parts, some of these combinations of features that we have. So we're moving complexity. It's like pruning your decision tree. We're removing parts of your neural network. So yeah, these surviving units then take on the work of those which went missing. So in that case, we would scale up their weights to compensate. Similarly simply because they have to take on more work. I think this is also, again, very, very closely related to how brains work. So I don't know whether you've heard that in some people with brain injury, for example, it's known that in other parts of their brains take on work that the injured parts used to take on. So this is again similar principle. These surviving units stand in for the ones that were removed. And yes, this prevents new ones from becoming over specialised. We talked in this example earlier. This is especially the case if you have an unbalanced training data. Again, so unbalanced training data can lead to your to your neural network and itself becoming over specialised. I've seen that. I've seen it happen to a colleague of mine who was very frustrated about his neural network. Just predicting always the class. No, even though he was looking for a good mix of yes and no for binary classification task. And the simple solution to that was just rebalancing your training data. So going back to the basics, looking at your pre-processing, trying it again. And then usually it helps. The other thing that I wanted to mention here as a summarisation to that is we've seen network tuning in itself sounds straightforward, but it's really complex. So we have the number of hidden layers that you can choose and the number of units per layer that you can choose. I was asked earlier whether it was on purpose that my multi layer network, which we had here, had fewer nodes and fewer nodes than training examples, and then again, fewer nodes in the next hidden layer versus my single single network, which actually had more nodes in the hidden layer than in the input layer. The answer is yes and no. So it was on purpose for the lecture series, but it's not always the case. The reason why you might do that is you have basically two ways of introducing complexity into your model. You can increase the number of nodes of neurones that you have per layer, or you can increase the number of layers, which also increases the number of nodes. So both are completely valid. Both work well and which one works better typically depends on your on your problem. So there's not really one way to to decide on that. Modern thinking is that we can have quite a high number of units per layer, because we are now at a stage where it's computationally efficient for us. And then you can control overfitting through regularisation, for example, dropout learning, which then just deactivate some of those and still gives you a good result. And the other way is to increase the number of layers, which is more the thought process behind deep learning. So some people believe that adding multiple layers which are not have don't have as many nodes performs better for very complex tasks. Which really works better depends on basically the amount of non-linearity and complexity that you have. You have to try it out yourself. Depends on your problem. Both are valid. Both were computationally similar. So number of hidden layers, number of units per layers. Then you have to choose regularisation tuning parameters. So your dropout rate for example, which is basically how many of these nodes to deactivate or the intensity of your lasso which regularisation. So all of that you have to then set separately for each of these layers. So there's a lot of decisions that you have to choose for your gradient descent or your stochastic gradient descent, which is the preferred method to avoid your local local optima would be the learning rate, batch size, and how many times you actually run it. So all of these parameters add to the complexity. And this is not even considering the parameter estimation which we've just seen is very complex. So lots of choices, lots of estimation, complex model. But I hope you can agree. But the idea is relatively simple. So it's a relatively simple idea of having multiple layers for which you feed the data forward. It's just underfitting and the tuning, which makes it so complicated. I also hope I don't sound too negative. I've heard in the past that people complain about me being too negative about neural networks, because I think they have very specialised, but I hope I gave you a good idea of there are really. Tasks for which they work exceptionally well, and they're just sometimes a bit overused because you need a lot of data for them. Okay, so that was our brief discourse about neural networks. As you know, this was the last lecture. So don't come here to mobile AI next week. I won't be here next week. I will be here tomorrow in your computer lab over in the other building. And we will work on implementing a neural network. You also have a deadline for the group coursework coming up. I know you don't want to hear about deadlines right now. I can imagine you're very tired of them. Please keep it in mind nevertheless. And please double check the time and location of your exam before the exam period. You should get an email if there are any updates about it, but don't risk it. Just double check the website before you go to the building. Okay, so I've added a link again here to the exam website. So what I wanted to do now is actually have a brief look at the exam together, because I think this is the perfect time for that exam information. Where have I added it? Here we go. Okay. So this is do you have any other exams or is this the only one? Just the one. Okay. Lucky you. So this will be the first the first time that you see an exam paper. And the last time apparently for this semester, which is good. So this is what it will look like. This is actually the scan of it. So this is exactly what it would look like obviously with slightly different times and dates. I can't stress this enough. Please read it carefully. You won't believe how many people each year don't read the instructions carefully and you will be very frustrated when that happens to you, because it's so easy to not read it carefully and then do something wrong, and then you're losing points just by being nervous, so you will be nervous. Take your time. You have two hours, which is more than enough time to produce a really good answer. Trust in that. So here, for example, it will tell you answer all questions and part one and answer one of the questions in part two. So you will have a multiple choice question again this time. Or a section multiple choice section which will be part one. And then there will be an open section for part two. So the structure will be the same this year. Obviously the questions will be different. I hope you know that. So for the multiple choice question cheat, you will get kind of an extra booklet on which you have to fill that out. And because they are machine read. So read carefully how you have to fill them out so that the computer and the machine can actually then read your answers. And we don't have the teaching administration staff having to decipher which option you take. Okay. We use their workload. They are working very hard in December. So that's the multiple choice questions sheet. You will have some kind of kind of booklet. And then you have to clearly mark which answer you have. And you have to use a pencil. It will be provided for you. And that's so that the machine can properly read it. So yes, very thin marks might not be read. So that's why you provide a soft one. Etcetera, etcetera. One mark is given for a correct answer. No marks will be subtracted for one answer. So please attempt, even if you're not entirely sure. And no mark is given or subtracted. If there's no attempt, or if a line is placed with more than one option. That is because all of these multiple choice questions will have one correct answer. So this is what it will look like. Attempt all questions. You will have 35 marks in total. Each question will give you five marks if answered correctly. And these are some of the examples that you will have. So they will have something like what is an example for linear regression problem for options. And then you have your separate sheet of paper and you clearly mark which one it would be. Who would like to answer question one. Can you see that. Can you do that? What is an example for linear regression problem? Classifying machine parts as faulty or not faulty. Predicting whether a person will default or not, segmenting a customer base based on the behaviour, or predicting the number of attendees at a concert. The exactly. We're predicting a number. And then you would mark very clearly the number. Did someone say it was going to last on the last one? Okay. Unrelated Giggling that's all right. Okay. So you will have seven questions total 35 marks. And then the second part will be open ended discussion questions. Again, I put in bold. Please answer one of the two 1 or 2. The reason why I put that in bold is because some people then try to answer both of these, and then you obviously don't have enough time to develop the question, so please read it carefully. Okay, so here in this year you were allowed to choose between logistic regression. Decision trees are two methods that describe them, compare them and discuss them. Or building a training machine. Learning models come with pitfalls. Explain what how to avoid overfitting. Talk about resampling. Talk about regularisation so you can see how these marks with this were. Then divide it here for example 35 and 30 marks. So in total you were able to get 65 marks. I was asked, how much should I write for an answer? There is not really a rule for that. You should answer like enough, if that makes sense. Concise answers are fine, so I don't have to read like a book report or something like that. As long as you tick all the boxes, you answer the question properly and concisely. I'm happy with that. Think about the timing. So you have two hours in total. You can estimate two hours for 100 marks. So an hour for 50 marks, like estimate how much time you should spend per per question or per section roughly. Based on that, I hope you're all practising your handwriting. I have to read it. Please make sure it's legible. If not, I can't give you marks. Okay? If I can't read it, then you're not getting marks. The other thing is what I would like you to have a look at is these keywords that I use here describe, compare and discuss. So think about what these words mean. And what I'm looking for describe is to actually describe what the methods do. So that's a description. You can use formulas but you don't have to. You can use numeric examples if you want but you don't have to. You can just describe the methods in your own words. The way it kind of makes sense and it's concise and correct. Compare and discuss then the critical analysis part. So I would like you to think about those methods, not just what they do but how they do it. So how do these two methods, how are they different? When would you use one or the other? So what kind of examples can you give me for? Yeah. For example, when a logistic regression is better than a decision tree. ET cetera. I say using examples. If I say that, please use examples okay. Please don't ignore that. So these are kind of examples for this question. Here we have explained as a keyword. Here we have discussed. So again we have kind of a descriptive part. And then we have a critical part okay. So I hope this is relatively clear. And that's the end of the paper. So you're given kind of booklets little kind of paper booklets. You can I think ask for more if you need more paper but don't go overboard okay. And yes you can draw in them as well. Pictures can be nice nice nice for me to look at over the Christmas holidays. Good. Any questions about the exam? Yes. Right? Or is it just. I believe it's lined. Yeah. Huh? Yeah. I'm going to see that. Announcer. Oh, gosh. Yes, that would be interesting. No, I think that line, I mean, honestly, they should be, like, squared, in my opinion, but I think you'll get lined. Yeah. Any other questions? You were very scared about the exam in the very beginning of the course. Remember that. Do you feel a bit more comfortable about it now? Yeah. No. Oh, gosh. The other thing that you can think about is the computer labs prepared you for the coursework. Basically the implementation I did mention theory in them, which hopefully explained methods better or further. But basically the lectures will feed into the exam and then the computer labs feed into the coursework. So that's how you can think about how these two elements work together. And the other thing is you can have a look at the learning objectives of the course again, because they really explain you how I constructed the exam, because that's what I want to test. I'm not doing that just for fun. So I'm not writing exams just for fun. I'm writing them because I want to see whether you have reached the learning objectives of the course. That's the formal way of assessing whether you've reached. Whether you've reached the end of the course. Does that make sense? Yes. Anything else. Questions about the group assignment. Any other questions about anything else? You have ten more minutes to pass to me with questions. Then I'm gone forever. No. I will still be available via email. If anything else comes up, send me an email. I'm happy to talk about that. And if you have questions about the exam or the course, or you would like to double check some material that we have covered during your review. Just email me and I'm happy to help with that. Anything else? No, I'm pretty happy. Are you pretty happy? Very stressed, but happy. Okay, I, I can't change that. Unfortunately, I. Would, I would if. I could. But you will get through it. I think we had a good time together. So I hope you enjoyed the lectures in general and found them helpful and interesting. I will see you before your exam, because I will be there to deliver the papers and then collect them after that. You can complain directly to me in person after the exam if you didn't like it, but it will be too late by then. So there you go. Okay, if I don't see you, then because I'm too quick or you're too quick. Happy holidays if you're celebrating, Happy New Year and it was a pleasure teaching you and I hope to see you again in the future. Thank you very much. You.