Neural network programming python pdf

Image source: PixabayArtificial intelligence (AI) is a buzz word that you see almost everywhere around you, even when you're not looking. It completely dominates tech media, newsrooms, and is even credited with the success of many modern applications. But does it really work, or is it just hype? The truth is, yes. While there might be some hype around its capabilities, AI has been shown to work really well in research and industry for various tasks and use cases. There are many techniques to make computers learn intelligently, but neural networks are one of the most popular and effective methods, especially in complex tasks such as image recognition, language translation, audio transcription, and so on. In this two-part series, I'm going to walk you through building a neural network from scratch. Even if you don't build one from scratch in a real environment, it's a good idea to work through this process at least once in a lifetime as an AI engineer. This can really help you better understand how neural networks work. In this first article you will learn: - What is artificial intelligence - What is deep learning - What does the neural network mean - Why are neural networks popular - Building a neural network from scratchRead python for AI? Understanding neural network layers Benefits and biasActivation functionCommand: Forward PropagationA Step Back: BackpropagationOptimization and Training of the Neural NetworkMaking PredictionsPutting It All TogetherAnd once you've had a chance to work through this tutorial, head over to part 2 where we actually train and test the network we build here: What is AIArtificial Intelligence (AI) is an umbrella term used to describe the intelligence displayed by machines (computers) , including their ability to imitate humans in areas such as learning and problem solving. This means that with AI, you can automate how you think, think and make decisions. As such, you can teach your computer to do what people do without explicitly programming it. Despite this simple explanation above, it is not as simple as it sounds. And while many scientists and researchers have been able to teach machines to behave like humans in areas like computer vision and natural language processing, there's still serious work to be done before we can have effective and fully functional AI systems. I guess that's why you're here-to learn how neural networks and AI work in general, and how you can use them to automate your own processes, create customized user experiences, and more. AI is wide and has many subfields that include machine learning. Machine learning itself has many techniques, of which neural networks are one (albeit a very successful technique). The difference between AI, ML and DL (image source)What is Deep Learning-learning and Learning Technique-is an effective way of learning that relies on large amounts of data where features that can help the machine map input to output is automatically extracted from layers of neurons. Deep learning is a major technology: Driverless cars Big-scale recommendation engines like Spotify, YouTube and AmazonLanguage translation services such as Google TranslateChatbots like Siri and Google Assistant. The neural network is a type of deep learning architecture and is our main focus in this course. Some specific architectures for deep neural networks include convolutional neural networks (CNN) for computer vision use, repetitive neural networks (RNN) for language and time series modeling, and others, such as generative dispute networks (GAN) for cases of using generative computer vision. The future of machine learning is on edge. Subscribe to the Fritz AI newsletter to discover the possibilities and benefits of embedding ML models into mobile applications. What is neural networkneural networks consist of simple building blocks called neurons. While many people try to draw correlations between neuron neuron neurons neurons neurons neurons neurons neurons neurons neurons, I will simply state the obvious here: The neuron is a mathematical function that takes data as input, performs transformation to them, and produces output. This means that neurons can represent any mathematical function; however, we usually use nonlinear functions in neural networks. One neuron in the networkYes looking for the neuron above, you can see that it consists of two main parts: summary and activation function. The neuron takes data (x_1, x_2, $x_▶$) as input, marries each with a certain mass (w_1, w_2, $w_▶$), and then overstees the result of a nonlinear function called activation function to produce output. The neural network combines multiple neurons stacking them vertically/horizontally to create a network of neurons—hence the name of the neural network. A simple one-neuron network is called perceptron and is the easiest network ever. Another important concept I will explain in later sections of this tutorial how the neural network actually learns the scales it assigns to each input function. In neural networks, weights are everything. If you know the correct weight, you can easily output the correct prediction. In short, what machine learning and deep learning really boils down to is actually trying to find the right weights to generalize to any input. In the previous section, I introduced neural networks and briefly explained the building blocks. Now we will examine why neural networks are popular today. Neural networks have been around for a really long time-a few big problems with them, and the reasons why people didn't use them before were due to that: They were notoriously difficult to train, in the sense that it can be get the right scales that generalize to new inputs. They need a huge amount of data. Computing power was still low and expensive. When these barriers were overcome, the neural networks cooled again, and many applications appeared. Neural networks are also very popular now because of their effectiveness on a wide range of tasks. They can automatically extract features from unstructured data, such as texts, images, and sounds, and deep learning has significantly reduced the time spent manually creating functions. To illustrate, I'll tell you a short story about Google Translate. In the early days of Google Translate, thousands of engineers, language experts, and computer scientists had to work all day to manually extract and create features from texts. These manual functions had to be inserted into machine learning models. Even with this time-consuming and costly task, the performance of these systems was nothing like a human being. But when Geoff Hilton's team showed that the neural network can be trained using a technique called backpropagation, Google switched from manual engineering functions to the use of deep neural networks, and this significantly improved performance. This anecdote shows that with enough data and computing power, neural networks can do better than other machine learning algorithms - that is, their growing popularity. Now that you've got a brief introduction to AI, deep learning and neural networks, including some reasons why they work well, you'll build your own neural network from scratch. To do this, you will use Python and its effective scientific library Numpy.Why Python for AI? Python is a high-level, interpreted, and universal language that can be used for a wide range of tasks. It's one of the easiest languages to learn, and it makes it the go-to for new programmers. Python is popular among AI engineers - in fact, most AI applications are built using Python and Python-related tools. There are many reasons for this, some of which include: There is a large ecosystem of pre-built libraries for scientific calculations. Libraries like NumPy, SciPy, and Pandas make scientific calculations easier and faster, because most of these libraries are well optimized for common ML and DL workloads. This means python is easily cross-platform compatible and can be deployed almost anywhere. Python has a useful and supportive community built around it and this community provides lots of guides and tutorials to work with the language. You can be sure that most of the problems you encounter have already been resolved. I love the comic below, which shows a flying programmer *winked. It shows why Python learns easily, with many libraries that you can import and use for almost any task – including antigravity 😳! Flying Programmer (image source)Before it is essential that you set the machine learning environment correctly. If not, you should first visit this page before moving on to the next section. What are you trying to solve? Before you write code, let's talk about the problem you'll be solving, because a more complete understanding of the problem will help you create a solution. In this tutorial, you will create a neural network that predicts whether a person will have heart disease or not. You will use a heart disease dataset from the UCL data store. You can download it here. UCL heart disease dataset pageNamed dataset page, click on the data folder and download the heart.dat file. It comes in .dat format. Create a new directory where your notebook and Jupyter data will live. Then copy the heart.dat file to the folder. Next, you can create a new notebook and add the following lines of code:In the code block above, you first set the headers, which are the column names for the data set. You can also get these names from the dataset description file on the data access page. Note that the sep parameters passed to pandas read the function? it tells pandas that the data is separated by spaces and not the default commas. In the next block of code, you print the data header:The first ten rows of the dataset From the data head, you can see the features present, and you can start imagining the kind of analysis that you will need to perform in the dataset. Next, print the shape of the data:(270, 14)There are 270 observations. This means that the neural network will have input data in the form of 270 x 13, except for the target variable (heart_disease). The properties present in the dataset are: agesexchest type of pain (4 values)resting blood pressure cholesterol in mg / dlfasting blood sugar > 120 mg / dlresting electrocardiographic results (values 0,1,2) maximum cardiac rate achievedexercise induced anginaoldpeak (ST depression induced by exercise due to rest) inclination of the peak of st numbers of the main blood vessels (0-3) colored fluoroscopy (3 = normal; 6 = solid defect; 7 = reversible defect)heart_disease: absence (1) or presence (2) of heart diseaseNew, you can check for missing values, as well as data types. The neural network expects all functions to be numeric and do not contain missing values. No values are missing from the dataset, and all functions are numeric. Next, you separate the target from the data, split it into a training and test kit, and then standardize the data. In the code block above, you first dropped the target from the training dataset and replaced classes 0 and 1. Note that you have reshaped the y_label into a 1D array. This is important when you start making products with a dot. Next, you used the practical train_test_split from sklearn to split the data into a train and test kit, with the test kit taking 20 percent Data. Finally, you standardized the dataset using the StandardScaler sklearn module. Now that you have downloaded and prepared the dataset, we begin to build a neural network to make predictions. To do this, you first need to understand the concept of layers. Remember when I said that the neural network folds multiple neurons together to create really large and complex mathematical functions? Well, the official name for it is layer. A layer is a collection of nodes at different stages of calculation in a neural network. Each node acts as a neuron and performs calculations on the data that is passed to it. Look at the illustration of the 3-layer neural network below: 3 layers of neural networkJa know you see four layers-but in deep learning, you do not have to count the first layer. The first layer is called the input layer, and the number of nodes will depend on the number of functions present in the dataset. In our case, it will be 13 knots, because we have 13 functions. The last layer of the neural network is called the output layer, and the number depends on what you are trying to predict. For regression and binary classification tasks, you can use a single node; while for multi-class problems, you will use multiple nodes, depending on the number of classes. In this article, you will use one node for the final layer because you are working on a binary classification task. Layers between the input and the last layer are where spells happen— they're called hidden layers. Hidden layers can be as deep or wide as you want, and while a deeper network is better, computing time also increases when you go deeper. To make it relatively simple, you design and time a 2-layer neural network. Below is a preview of the architecture:2 layers of neural net neural net above will have one hidden layer and the final output layer. The input layer will have 13 nodes, because we have 13 functions, except for the target. The hidden layer can accept any number of nodes, but you start with 8, and the final layer that makes predictions will have 1 node. Next, we talk about the weight and distortion that each layer must have. Scales and BiasesWeights and biases are teachable parameters that help the neural network properly learn function. Think of scales as a measure of how confident you are that a function contributes to prediction and distortion as the base value from which your predictions must begin. I'll give you an illustration. Suppose you're a machine learning model, and you want to predict whether a person is rich or not, and you've been given the following clues to help you make this decision: Age of a personHeight personHeight personEnding above are what we call features in machine learning, and what you want to predict is called goal/label/country truth. The label can be one of two (rich, not rich)—in other words, binary classification. Basically, what you want to do is combine the features in such a way that they help you more accurately predict the result.y (rich, not rich) = Age + Height + salary + [base]Provided that we have set the base salary of $ 3000 and person 1 has the following functions; age = 18, height = 5.6 feet, salary = $ 2000, then you calculate wealth as follows: y (rich, not rich) = 18 + 5.6 + 2000 + 3000 = ~ 5024For this example, we can define the threshold for wealth as any value greater than $ 40,000. Judging by these criteria, you can conclude that person 1 is not rich. Let's look at the next example. Person 2 has the following characteristics; age = 26, height = 5.2 ft and salary = $50,000. Your prediction will be calculated as:y(rich, not rich) = 26 + 5.2 + 50000 + 3000 = ~ 53031Then, according to the threshold previously stated, person 2 is rich. Obviously, some clues are more important than others. Can you guess which one is most important? Yes! Salary. This is perhaps unsurprisingly, an important factor that indicates whether a person is rich or not. You can use this idea to assign importance to functions. For example, you can assign weights as follows: y (rich, not rich) = (2 * Age) + (1 * Height) + (8 * Salary) + baseIntuitively assign a higher value to the pay function. The meaning of the value can be any number, but it must be representative of the scale. You may be wondering what the base value of 3000 is and why we're adding it to predictions. This value is called distortion. It is the base value that each prediction must have, even if nothing else is given. Now, if you make a prediction for person 1 and 2 again, you will have the following:Person 1: (2 * 18) + (1 * 5.6) (8 * 2000) + 3000 = ~ 19041 (still bad) Person 2 : (2 * 26) + (1 * 5,2) + (8 * 5000) + 3000 = ~43057 (Still rich)What if a person has no value for age, height, and salary, then your prediction will be?y(rich, not rich) = (2 * 0) + (1 * 0) (8 * 0) + 3000 = 3000 Now you see where the distortion value comes from. What you should take away from the examples above is that the importance values assigned to functions are called scales, and the base value is called distortion. The machine learning model uses many examples to learn the correct weights and distortions assigned to each function in the dataset in order for it to correctly predict the outputs. Back to our proposed solution. You now know that each function in our dataset must be assigned a weight, and that after you make a weighted total, you add the term distortion. In the code block below, you create a neural network class and initialize these scales and distortions:First, you create a neural network class, and then during initialization, you create some variables to store intermediate calculations. An argument layer is a list in which the network architecture is stored. You can see that it accepts 13 input functions, uses 8 nodes hidden layers (as we mentioned earlier) and eventually uses 1 node in the output layer. We will talk about other parameters such as learning rate, sample size and iteration in other sections. Moving to the next piece of code, you have created a function (init_weights) to initialize scales and distortion as random numbers. These scales are initialized from a uniform random distribution and stored in a dictionary called paramy. Note that there are two weight and distortion fields. The first weighing field (W1) will have dimensions of 13 by 8 – this is because you have 13 input functions and 8 hidden nodes, while the first distortion (b1) will be a size 8 vector because you have 8 hidden nodes. The second field weight (W2) will be 10 by 1-dimensional array because you have 10 hidden nodes and 1 output node, and finally, the second bias (b2) will be a vector of size because you have only 1 output. I'm guessing you see a pattern here. This means that if you have a neural network with the following architecture [20,30,2], then you know that you will have the following dimensions for your scales and distortions: W1 = (20,30) , b1 = (30,)W2 = (30, 2), b2 = (2,)And if you have 3 layers of architecture as [5,7,8,2], then you know that you will have 3 scales and 3 distortions with the following shapes: W1 = (5,7) , b1 = (7,)W2 = (7,8), b2 = (8 ,)W3 = (8,2), b3 = (2,)So what will be the dimensions for a neural network with this architecture [20, 23, 2]? Activation functionNow that you initialized scales and distortions, let's talk about activation functions. Activations are nonlinear calculations performed on each node of the neural network. Remember when I told you that every knot does some kind of mathematical calculation? Well, the calculation happens in two stages. First, make a weighted sum of input and weights, add distortion, and pass the result using the activation function. I'll explain why we're doing it below. Activation function is what makes the neural network capable of learning complex nonlinear functions. Nonlinear functions are difficult to learn for traditional machine learning algorithms such as logistics and linear regression. Activation function is what makes the neural network able to understand these functions. There are many types of activation features used in deep learning – some popular are Sigmoid, ReLU, tanh, Leaky ReLU and so on. Each activation feature has its advantages and disadvantages, but relu has been shown to work very well, so in this article you will use the ReLU function. Various activation functions used in deep learning (image source)The activation function is calculated by each node in the hidden layers of the neural network. This means that you will need to pass the weighted amounts through the ReLU function. But what is ReLU? Retified Linear Unit (Retified Linear Unit) is a simple function that compares a value with zero. That returns a value that is passed to it if it is greater than zero; otherwise, it returns zero. The code for the ReLU function is listed below:Add it inside the NeuralNetwork class. This feature performs field-wise ReLU because you will deal mainly with fields, not individual values. In short, the hidden layer receives values from the input layer, calculates the weighted sum, adds the term distortion, and then relays each result through the activation function — in our case, ReLU. The result from ReLU is then passed to the output layer, where the next weighted sum is performed using the second weight and distortion. But then instead of passing the result through another activation function, it is passed what I wanted to call the output function. The output function will depend on what you are trying to predict. You can use the sigmoid function if you have a two-class problem (binary classification) and you can use a function called softmax for multi-class issues. In this tutorial, you will use the sigmoid function for the output layer. It's because you're predicting one of two classes. Sigmoid function (image source)The sigmoid function takes a real number and crushes it to a value between 0 and 1. In other words, it outputs probability scores for each real number. This is useful for a given task because you don't want your model to predict only yes (1) or no (0) — you want it to anticipate probabilities that can help you measure how confident its predictions are. Let's add Sigmoid to our NeuralNetwork class: You use the exponential Numpy function to code the sigmoid function. This allows you to perform an operation on an array instead of individual values. Also, the implementation of Numpy is faster than pure Python, as written in C.The Loss FunctionNext, let's talk about the function of loss of the neural network. The loss feature is a way to measure how good a prediction model is so that it can adjust weights and distortions. The loss function must be properly designed to correctly penalize a model that is flawed and reward a model that is correct. That is, you want the loss to tell if the forecast is far or near the actual prediction. The choice of loss depends on the task — and you can use cross-loss entropy for classification issues. Loss of cross-entropyWhich C is the number of classes, y is the actual value and y_hat is the predicted value. For binary classification task (i.e. C = 2), the function of cross entropy loss happens:Now, let's put it in the code:Note the sum and distribution by sample size in the code block above? This means that you are considering the average loss with respect to all inputs. This means that you are concerned about the combined loss from all samples and not about individual losses. Going Forward: Forward PromotionNow You have some basic building blocks for your neural network, you move to a very important part of the process called forward propagation. Forward propagation is the name of a given series of calculations performed by the neural network before prediction is made. In a two-tier network, you perform the following calculation for forward propagation:Calculate the weighted sum between the input and the weight of the first layer, and then add the distortion: Z1 = (W1 * X) + bPass result through relu activation function: A1 = Relu(Z1)Compute weighted sum between output (A1) previous step and weight of the second layer – also add distortion: Z2 = (W2 * A1) + b2Compute output function by passing the result through the sigmoid function: A2 = sigmoid(Z2)And finally, calculate the loss between the expected output and the actual labels: loss (A2, And there you have forward propagation for your two-layer neural network. For a three-layer neural network, you would have to calculate Z3 and A2 using W3 and b3 before the output layer. Now let's encode it. Be sure to add code to the NeuralNetwork class:In the code cell above, you first make all the products dots and add the weights and distortions that you initialized earlier, calculate the loss by calling entropy_loss, save the calculated parameters, and then return the estimated values and loss. These values will be used during backpropagation. Ugh! That's a lot, I'm happy to report that you're halfway to completing your neural network. Take a moment to smile! Backpropagation is the name of the neural network training process by updating its weight and bias. The neural network learns to predict correct values by constantly testing different values for weights and then comparing losses. If the loss function decreases, then the current weight is better than the previous one or vice versa. This means that the neural network must go through many training (forward propagation) and update (backpropagation) cycles in order to gain the best weights and prejudices. This cycle is what we generally refer to as the training phase, and the process of finding the right weight is called optimization. The question is how do you load the neural network to correctly adjust its weight in view of the loss it calculates. Well, thanks to math, we can use calculus to do it effectively. So the number you learned at school is important after all, 😊. The primer on CalculusCalculus helps us understand how a change in one variable affects another variable. This means that you can use the count to calculate how much weight change/distortion affects the loss function. So basically we use calculus to understand how much and in what direction to update weights and bias to reduce loss. Suppose you have a function y = This function tells you that the y value is 2 times higher than the value of x. i.e. we call it derivation y with respect to x.You can easily know the rate of change of many functions through special calculated derivatives. These derivatives are formulas that have been studied and can be quickly used to calculate complex derivatives. For example, in $y = x^2$, the derivation is 2x. This means that the rate of change is 2 times higher than the x value. How was it calculated? By number, you can calculate the derivation of the form function $y = x^n$ using the formula:For example, if y = X, then n = 4, so that the derivative will be calculated as:What if you have special functions such as sigmoid, ReLU, tanh, Sin, or a combination of multiple functions such as $3x + 2x^2$ - how do you calculate the derivative? The good news is that most of these features are built from a combination of smaller functions, so you can use the chaining concept to aggregate derivatives – enter the string rule! So if you have two functions that depend on each other – say $2x^2 + 3x$ – then the derivative becomes the addition of individual derivatives:But if two functions are multiple, the calculation varies. Here is an instance - assuming you need to find × derivation of this function $y = 2x$- and $4x^2$, you can calculate the derivation as follows: First assign the alphabet to each function; so let's say that and the first part is assigned, ie. Then the derivation happens:That is, for two functions multiplied together, first you take the derivation of the first part a (Δa) and multiply it by the second part b, then you take the derivation of the second part b (Δb) and multiply it by the first part, and finally summarize the result. So the derivative becomes: and from the laws of indices, it decreases to:Now, I do not plan to show you all derivatives available in number, but basically you should know that most derivatives you use when encoding backpropagation algorithm have already been calculated, so just use formulas. A couple of great resources to learn more about derivatives here:Using Calculus in Backpropagation After calculating output and loss in the forward propagation layer, you'll move into the backpropagation phase where you calculate derivatives back, from loss all the way up to first weight and bias. To perform backpropagation in your neural network, you will follow the steps below:Starting with the last layer, calculate the derivation of losses with respect to output yhat as:But how did you get lost? Well, you calculated sigmoid(Z2). Now what is a loss derivative with regard to sigmoid(Z2)? Sigmoid(Z2) is a combination of two functions, so you need to calculate two derivatives:First calculate the derivation of sigmoid activation with respect to (wrt) loss:Then you calculate the derivation of loss WRT Z2: Now how did you get Z2? You have calculated a dot product between A1 and W2 and added distortion b2. This means that you have to calculate the loss with respect to all these variables:And how did you get to A1? You have performed ReLU(Z1). So you take a derivative of ReLU and Z1 wrt at a loss as well. ReLU derivation is 1 if the input is greater than 1 and 0 otherwise. You create a function to calculate this and call it dRelu: Next, how did you get the Z1? You calculated the dotted product between X and W1 and added distortion b1. So you calculate the derivation of all variables involved, except for the input of X.Note: dl_wrt is to read the loss with respect toPheeewww! Now you have all your derivatives for the backpropagation algorithm. If you want a detailed overview of how these derivatives are calculated from scratch, this Medium post is a great guide. Next, we write the code backpropagation:In the backpropagation function, you first create a function for calculating ReLU derivatives, then you calculate and save the derivation of each parameter with respect to the loss function. Note that we use a common naming scheme (dl_wrt). This helps keep the code clean and easy to read. Once you have calculated these derivatives, you must update the previous weights. This is the essence of computational derivatives - basically, you want to know how to update the scales to minimize loss. Optimization and training of the neural networkIn the previous section, you used the count to calculate the derivation of weights and prejudices with respect to loss. The model now knows how to change them. To automatically use this information to update weight and distortion, the neural network must perform hundreds, thousands, and even millions of propagation back and forth. This means that in the training phase, the neural network must perform the following:Forward propagationBackpropagationWeight update with calculated gradientsRepeatLet is to write code that updates weights and distortions. In the backpropagation function, add the following lines of code: What you're basically doing here is subtracting a derivative multiplied by a small value called a learning rate. The learning rate is a value that tells our neural network how big the update should be. Now that you've added lines of code to make updates, you create a new feature called fit that takes input (X) and labels (Y) and repeatedly calls forward and backpropagation for a specified number of iterations:The customization feature takes 2 parameters: X (input dataset) and y (labels). First, it saves the train and the target on and then initializes the scales and distortions by calling the init_weights. It then loops through a specified number of iterations, performs forward and backpropagation, saves a loss. To make predictions to flow predictions, simply perform a data transfer test. This means that you are using stored weights and distortion from the training phase. To make the process easier, you add a function to the NeuralNetwork class named predict:The function passes data through the propagation layer forward and calculates the prediction using stored weights and distortions. Predictions are probability values ranging from 0 to 1. To interpret these probabilities, you can either round up values or use the threshold function. To make it simple, we just rounded up the probability. Putting them togetherlet is to put all the code together:Congratulations, now you have a fully functional, 2-layer neural network for binary classification task. You should get a pat on the back. But before declaring a complete victory, let's see if your network actually works. In the next post, you'll make predictions and also compare your network's predictions with your favorite deep learning libraries. If you have any questions, suggestions or feedback, feel free to use the comments section below. Connect with me on Twitter.Connect with me on LinkedIn.Editor Note: Heartbeat is a contributor-driven online publication and community dedicated to exploring the emerging intersection of mobile app development and machine learning. We are committed to supporting and inspiring developers and engineers from all walks of life. The redactionally independent Heartbeat is sponsored and published by Fritz AI, a machine learning platform that helps developers teach devices to see, hear, perceive, and think. We pay our contributors and we don't sell ads. If you would like to contribute, please take our call for contributors. You can also subscribe to our weekly newsletters (Deep Learning Weekly and Fritz AI Newsletter), join us at Slack, and follow Fritz AI on Twitter for all the latest in mobile machine learning. Learning.