Deep Learning

Cheng Guan

VISION@OUC

July 22, 2018

1 Research problem

During this week, I spend about four hours a day to learn deep learning course [1] and continue to practice programs in leetcode.

2 Research progress

2.1 Neural Network Representation

As shown in Fig. 1, the input features x_1, x_2, x_3 stacked vertically were called the input layer of neural network, which contains the inputs of neural network. The another layer of circles is called a hidden layer of neural network. The final layer that is called output layer is just one node, which is responsible for generating the predicted value $\hat{y}$. The meaning of hidden layer is that in the training set the true values for these nodes in the middle are not observed. In the training set, I don't know its value and just know the value of input and output.

So this is so-called hidden layer. $a^{[0]}$ refers to the values that different of the neural network are passing on to the subsequent layers. So the input layer passes on the value x to the hidden layer. $a^{[1]}$ is a four dimensional vector. The input layer will generate value $a^{[2]}$ which is a real number and so $\hat{y}$ is equal to the value of $a^{[2]}$. Generally speaking, Fig. 1 is a two-layer neural network, because the input layer is regarded as the 0th layer. As shown in Fig. 2, the circle images represents two steps of computation.

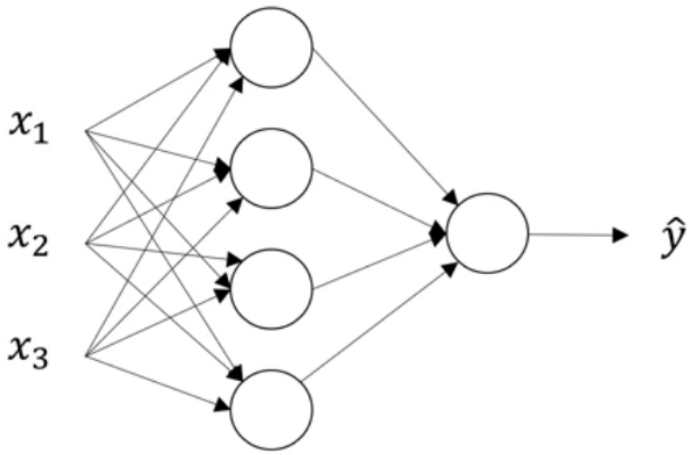

Figure 1: Neural network

The first step is to compute z and the second step is to compute activation. So the neural network just does this more times. As $a_i^{[l]}$ shown, l is the layer of neural network and i refers to the nodes in that layer. As shown as the Eq.1, we can calculate the output of a hidden layer neural network.

$$\begin{aligned} z^{[1]} &= W^{[1]}X + b^{[1]} \\ a^{[1]} &= \sigma\left(z^{[1]}\right) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma\left(z^{[2]}\right) \end{aligned} \tag{1}$$

2.2 Vectorizing across Multiple Examples

Take the training examples and stack them in columns to form a matrix. As shown in Eq.2, l refers to the layer of neural network and i refers to the ith example. Actually if we use for loop to implement the algorithm, that will be a difficult process of computation. As shown in Fig.3, In m training samples, each

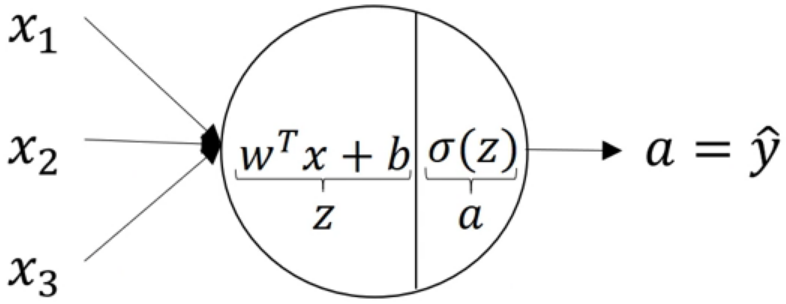

Figure 2: Neural network

calculation is repeated in the same process, and the output of the same size and structure is obtained, so a single sample is merged into a matrix by the idea of vectorization.

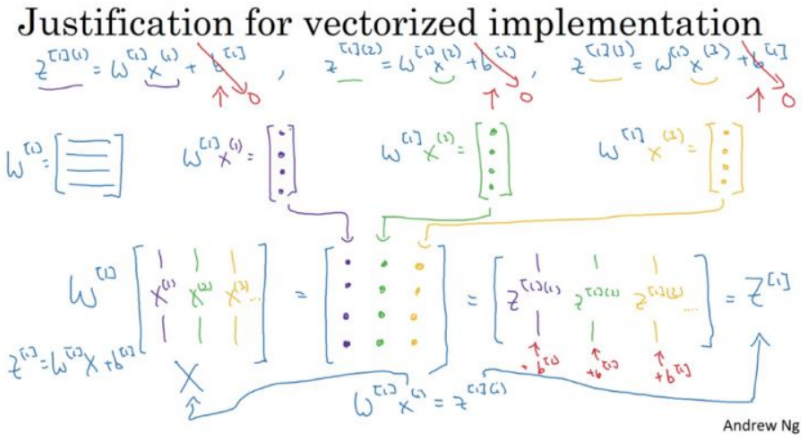

Figure 3: Justification for vectorized implementation

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ a^{[l](1)} & a^{[l](2)} & \cdots & a^{[l](i)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{2}$$

2.3 Activation Function

So far, there are four activation function in Fig. 4 the sigmoid activation is regarded as the activation function. it's just one of the activation functions and sometimes other activation function can work much better. The sigmoid function goes between 0 and 1 but the tanh can almost always work better than the sigmoid function, which goes between -1 and +1 as shown in Eq.3:

$$\frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3}$$

It turns out that for hidden units if the activation function is equals to $tanh\,(z)$, which has the effect of centering the data and makes learning for the next layer a little bit easier.

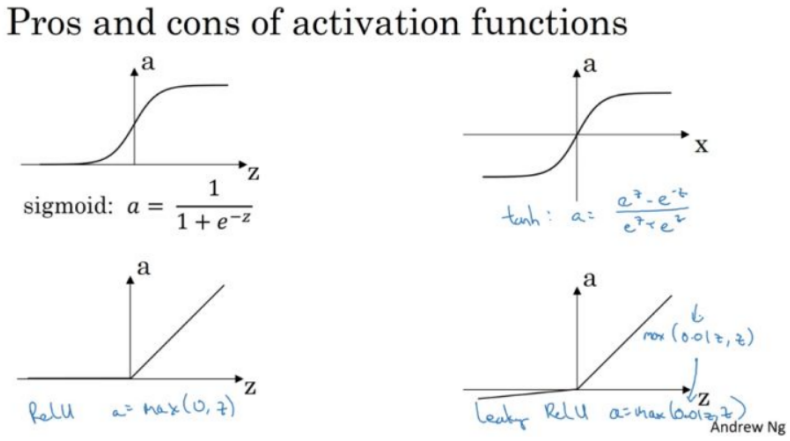

Figure 4: Activation functions

2.4 Derivatives of Activation Functions

When we implement back-propagation for the neural network, we need to compute the derivative of the activation functions. So it's important to choose activation function and compute the slope of these functions. Take the sigmoid function as an example, we can easily compute the its derivative as shown in Eq.4.

$$\sigma'(z) = \sigma(z)\left[1 - \sigma(z)\right] \tag{4}$$

According to the knowledge about calculus, we can take derivatives and show that it simplifies to the Eq.5:

$$g'(z) = 1 - (\tanh(z))^2 \tag{5}$$

As for the activation function ReLU, we can calculate the derivatives similarly as shown in Eq.6. It doesn't matter to set the derivative to be equal to a certain value when when z is equal to zero. So arm with these formulas, we can compute the slopes or the derivatives of the activation functions and implement gradient descent for teh neural network.

$$g'(z) = \begin{cases} 0, & if \quad z < 0 \\ 1, & if \quad z > 0 \\ undefined & if \quad z = 0 \end{cases} \tag{6}$$

2.5 Gradient Descent for Neural Network

According the knowledge of the calculus, we can deduce the formulas of the back propagation as shown in Eq.7:

$$\begin{aligned} dz^{[2]} &= a^{[2]} - y \\ dW^{[2]} &= dz^{[2]} a^{[1]^{\mathrm{T}}} \\ db^{[2]} &= dz^{[2]} \\ dz^{[1]} &= W^{[2]^{\mathrm{T}}} dz^{[2]} * g^{[1]\prime}\left(z^{[1]}\right) \\ dW^{[1]} &= dz^{[1]} x^{\mathrm{T}} \\ db^{[1]} &= dz^{[1]} \end{aligned} \tag{7}$$

2.6 Random Initialization

For the logistic regression it was fine to initialize the weights to zero, but for a neural network of initializing the arrays of the parameters to all zero and apply gradient descent that will failed. As shown in Fig. 5, x_1, x_2 are two input features and two hidden units. So the $W^{[1]}$ ia a matrix of 2×2. If we

initiate the matrix to zero and $b^{[1]}$ is also a zero matrix. It turns out that initializing the bias terms to zero is actually fine, however initializing W to all zero is a problem.

Table 1: Initialize the parameters

w = np.random.rand((2,2))∗0.01
b = np.zero((2,1))

If the parameters of the two hidden neurons are set to the same size at the beginning, then the two hidden neurons have the same effect on the output unit. The same gradient size will be obtained when the back gradient descends to the calculation, so the two hidden layer units are still the same after repeated iteration. So the solution is to initialize the parameters randomly. We can initialize the parameter as shown in Table.1.

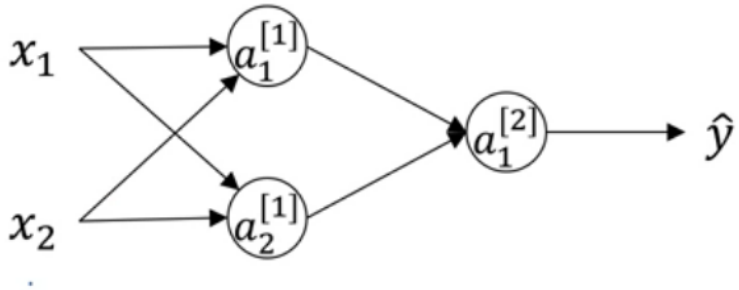

Figure 5: A two layer neural network

Multiply the value of W by 0.01 to make the weight W initialize to a smaller value as much as possible. This is because if the sigmoid function or the tanh function is used as an activation function, the W is smaller, then the value of $Z = WX + b$ is also relatively small. The proximity gradient of the 0 point area is larger, which can greatly improve of speed of updateing of the algorithm. If the W is too large, the gradient will be smaller, so the training process will become very slow.

3 Deep Neural Network

3.1 What it Deep Neural Network

As shown in Fig.6, they are the logistic regression, two layer neural network, three layer neural network and five layer neural network respectively. Technically logistic regression is a one layer neural network. On the neural network of L layer, the dimension of each parameters in a single example are in the Tab.2: The Fig.7 is the four layer neural network that has three hidden layers and the number

Table 2: The dimension of each parameter.

$W^{[l]} : \left(n^{[l]}, n^{[l-1]}\right)$
$b^{[l]} : \left(n^{[l]}, 1\right)$
$dW^{[l]} : \left(n^{[l]}, n^{[l-1]}\right)$
$db^{[l]} : \left(n^{[l]}, 1\right)$
$Z^{[l]} : \left(n^{[l]}, 1\right)$
$A^{[l]} = Z^{[l]} : \left(n^{[l]}, 1\right)$

of the hidden layer is 4, 4 and 3, respectively. $n^{[l]}$ denotes the number of nodes or the number of units in layer l. So if we index the input as layer zero, $n^{[1]}$ that is the first hidden layer would be equal to 5. Similarly, the $n^{[2]}$ that is the second hidden layer would be equal to 5, $n^{[3]}$ would be equal to 5, $n^{[4]}$ would be equal to 3, $n^{[5]}$ would be equal to 1. $a^{[l]}$ denotes the activations in layer l and the $W^{[l]}$ denote

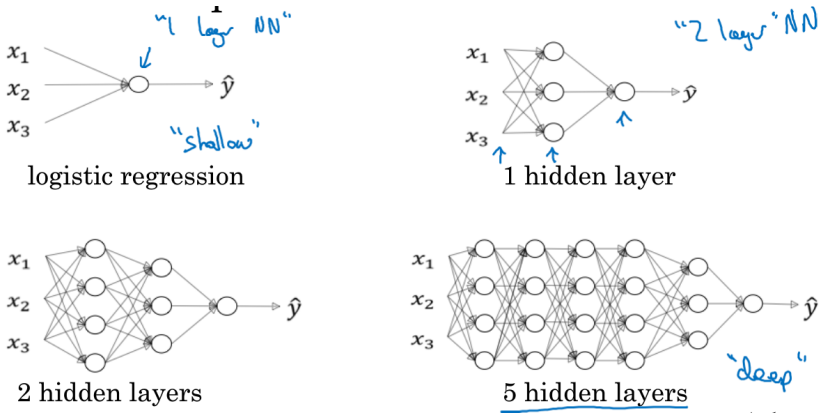

Figure 6: What is a deep neural network? They are the logistic regression, two layer neural network, three layer neural network and five layer neural network, respectively.

the weights in layer l. The input features are called x which is the activation function of the layer zero and the activation function of final layer is equal to $\hat{y}$.

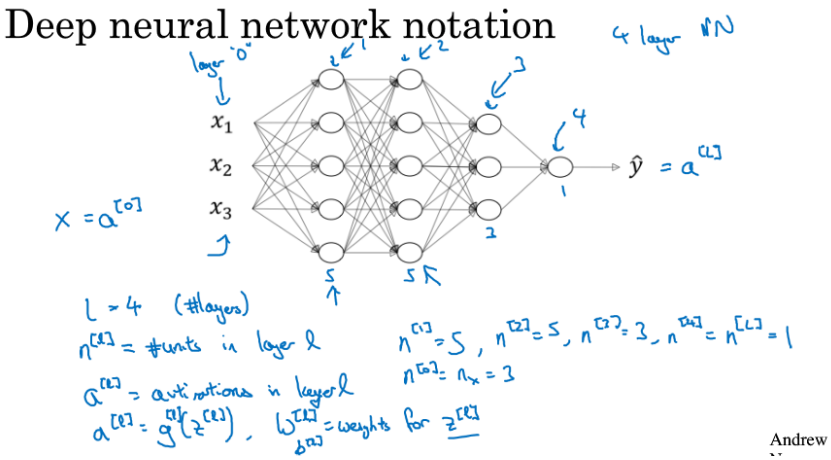

Figure 7: The five layer neural network

3.2 The Purpose of Using DNN

In a system for face recognition or face detection, maybe we input a picture of a face. We regard the first layer of the neural network as a feature detector or an edge detector.

As shown in Fig. 8, for face recognition, the first layer of the neural network extracts the outline and edge of the face from the original picture. Each neuron learns the information of the different edges; the second layer of the network combines the first layer of the learned edge information to form some local features of the face, such as the eyes, the mouth and so on. The latter layers gradually combine the features of the previous layer to form the appearance of the human face. With the increase of the number of neural networks, the features extend from the original edge to the whole of the face, from the whole to the part, from simple to complex. The more layers there are, the more accurate the learning effect will be.

For speech recognition, the first layer of neural network can learn some tones of the language pronunciation. The deeper network can detect the basic phonemes, then to the word information, and gradually deepen the learning of phrases and sentences. So from the above two examples, I can see that as the depth of neural network deepens, the model can learn more complicated problems and function more powerful.

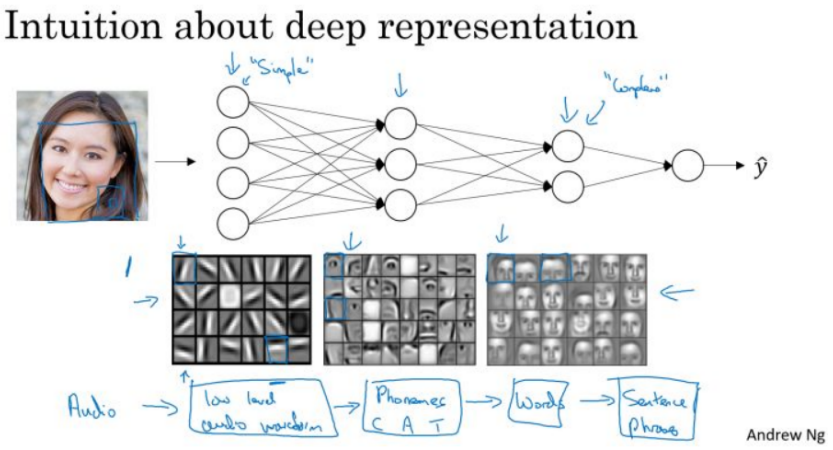

Figure 8: The process of face recognition.

4 Train/Dev/Test sets

When training a neural network, I should have to make a lot of decisions, such as how many layer the neural network have, how many hidden units we want each layer to have and what the activation functions we want to use to for the different layers. It's almost impossible to correctly guess the right value for all of these and for other hyperparameter choices. So in practice applied machine learning is a highly iterative process. Setting up the data sets well in terms of the train, development and test sets can make the process of training much more efficient. Even very experienced deep learning people find it almost impossible to correctly, and then applied deep leanring is a very iterative process where we just have to go around this cycle many times to hopefully find a good choice of network for application. As

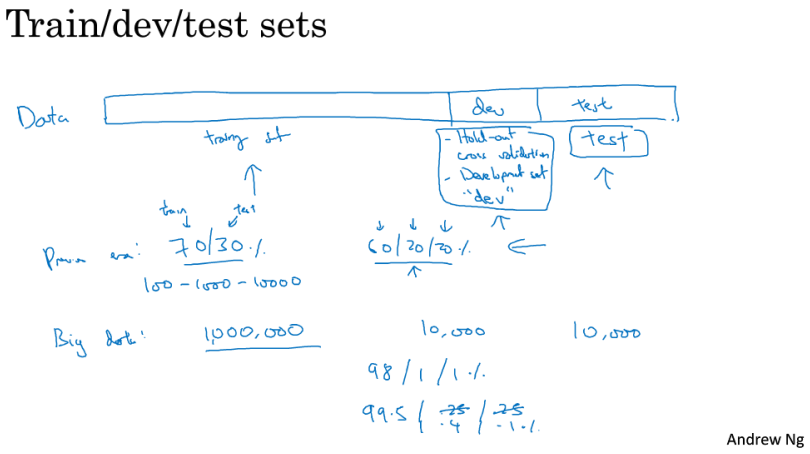

Figure 9: Train/dev/test sets.

shown in Fig.9, training data is divided into some portion of it to be training set. And some portion of it to be hold-out cross validation set and sometimes it also is called development set. Then some final portion of it is the test set. In order to get an unbiased estimate of how well algorithm is doing, in the previous era of machine learning, it was common practice to take all data and split it according to maybe a 70/30% in terms of people often talk about 70% train and 30% test splits or maybe a 60% train, 20% dev, and 20% test which is widely considered best practice serveral years ago. However, we now have a million examples in total, then the trend is that, we might with 99.5% train, 0.25% dev and 0.25% test. When setting up the machining learning problem, usually set it up into a train, dev and test sets. if we have a relatively small dataset, these traditional ratios might be fine. But if we have a much larger data set, it's also fine to set the dev and test sets to be much smaller than the 20% or even 10% of the data.

From this section, I learn the practical aspects to make neural network work well and know how to set up data and make sure optimization algorithm runs quickly, in order to make our learning algorithm to learn in a suitable time.

5 Progress in this week

After a week of deep learning of Andrew Ng, I finish the practicing of programs. I have a deeper understanding of deep learning, including shallow neural networks, deep neural networks, and their forward propagation and backward propagation.

6 Plan

Objective: Finishing improving deep neural networks courses
Deadline: 2018.07.30

2018.07.18—2018.07.24 Finish neural networks and Deep Learning.

2018.07.24—2018.07.30 Finish improving deep neural networks courses.

2018.07.31—2018.08.06 Finish structuring machine learning projects courses.

2018.08.07—2018.08.13 Finish convolutional neural networks courses.

2018.08.14—2018.08.20 Finish sequence models courses.

References

[1] A.Ng. Neutral network and deep learning. http://mooc.study.163.com/. 1