Chapter 7

Neural Networks

We have studied, linear and logistic regression models in Chapter 5. In linear regression models we fixed the basis and then try to find an optimal combination of these basis. Linear Regression model do not work well in more complex cases. One solution around these problem is to have a fixed number of adaptive basis where the basis and their combination is jointly learned during training. The most popular method of this type is Neural Network. In this chapter we will study the Neural Networks and important considerations for their implementation.

7.1 Basic Blocks of Neural Networks

The linear regression models we have seen so far are liner functions of some basis functions of the form:

$$f(x) = \sum_{i=0}^{N} w_i x^i \tag{7.1}$$

More generally we can write,

$$f(x) = \sum_{i=0}^{N} w_i \phi_i(x) \tag{7.2}$$

where ϕ_i are the fixed basis functions.

In neural networks, we write the basis as a composition of linear operation, where each linear operation is followed by a non-linearity. Each of these linear operation followed by a non-linearity is called a layer. Below we provide the linear operation in the first layer

$$a = Wx + b \tag{7.3}$$

where, x is in the input, W is a weight matrix, b is a vector of biases, and a is the output, known as activation. The non-linearity in the neural network is a non-linear activation function. There are several activation functions, the most popular are rectified linear unit (RELU) and sigmoids and softmax (details of activation function will follow below)

$$z = h(a) \tag{7.4}$$

where $h(.)$ is the non-linear activation function and it is applied element-wise to the input a and z is the output of the first layer.

More generally er can write, the first layer as $f^{[1]}(x) = h(W^{[1]}x + b^{[1]})$, then an N layer neural network would be, the composition of N layers

$$F(x) = f^{[N]} \circ f^{[N-1]} \circ ... \circ f^{[2]} \circ f^{[1]}(x). \tag{7.5}$$

where each layer $f^{(}[i])$ is parameterised by unknown parameters $W^{[i]}, b^{[i]}$.

Question: Can we remove the non-linearities in the neural networks? If we remove them what will happen?

?

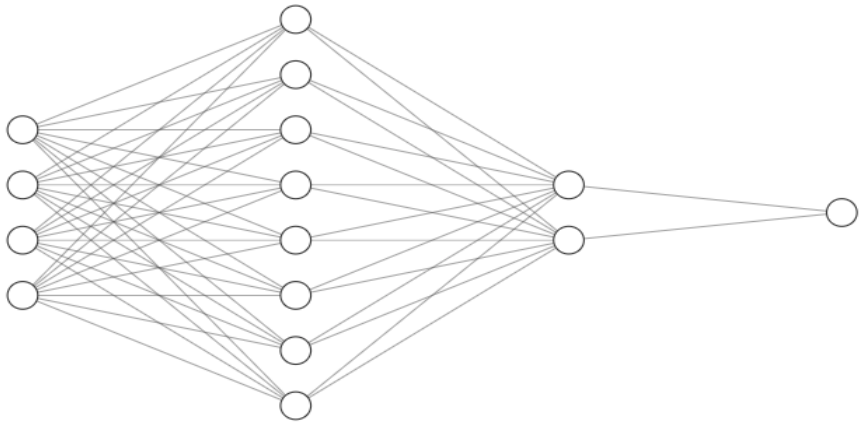

Figure 7.1: A three layer neural network

7.2 Activation Function

Activation functions are the non-linearities that are applied to the output of the linear operation of each layer. Some of the most commonly used activation functions are:

Rectified Linear Unit

Rectified linear unit is one of the most commonly used activation functions in machine learning and it is one of the reasons that we are able to train deep networks (network with many layers). A rectified linear unit layer is defined as

$$f(x) = max(0, x) \tag{7.6}$$

The derivative of the Relu layer is defined as

$$f'(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases} \tag{7.7}$$

Notice that the derivative is not defined at $x = 0$.

Sigmoid Function

Neural networks have proved very successful in classification applications, where the desired outputs are discrete class labels rather than continuous values. Instead of the network output being discrete labels (calculating derivatives for which will be difficult as the output will not be continuous), we design the network such that the outputs are the probability of input belonging to a class. Sigmoid function is useful in these scenarios. A sigmoid is defined as:

$$f(x) = \frac{1}{1 + exp(-x)} \tag{7.8}$$

The derivative of the sigmoid function is defined as

$$f'(x) = f(x) * (1 - f(x)) \tag{7.9}$$

The important property of the sigmoid function is that it's output lies in [0,1) which can be interpreted as probability of belonging to a class.

Tanh Function

Another common non-linearity used in the neural networks is tanh. Tanh is defined as:

$$f(x) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)} \tag{7.10}$$

The derivative of the tanh function is defined as

$$f'(x) = 1 - f^2(x) \tag{7.11}$$

Softmax function

In most classification applications rather then dividing the problem into a set of binary classification problems (where we can use sigmoid), we want the output to be probability of belonging to each class, where the sum over all probability over classes is equal to 1. For these applications we used softmax function. Softmax function is defined as:

$$f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{7.12}$$

The derivative of the softmax function is defined as:

$$\frac{\partial f(x_i)}{x_i} = \frac{exp(x_i)}{\sum_j exp(x_j)} - \frac{exp(x_i)^2}{(\sum_j exp(x_j))^2} \tag{7.13}$$

$$\frac{\partial f(x_i)}{x_k} = -\frac{exp(x_i) * exp(x_k)}{(\sum_j exp(x_k))^2} \tag{7.14}$$

for $k \neq i$

7.3 Back-propagation and Weights Update in Neural Networks

Using equation (7.5) and the nonlinear functions defined above we can design a neural network of any shape and size. Calculating the output of the neural network amount to composition of layers in order as in (7.5). But what are the right values of the weights of the neural network and how can we train for the right values of the parameters of the network, given that we can have an arbitrarily complex network.

To answer this question we will first give a brief introduction of the chain rule we used in our calculus courses. Generally speaking we can define a derivative over composition of function as:

$$\frac{d}{dx} f(g(x)) = \frac{df(g(x))}{dg(x)} * \frac{dg(x)}{dx} \tag{7.15}$$

Calculating, the gradients of the loss function of the neural network with respect to the weights of the networks, translate of application of the chain rule.

Let's show it with and example. Let's assume that our neural network is defined as

$$F(x) = s \circ f^{[4]} \circ f^{[3]} \circ f^{[2]} \circ f^{[1]}(x). \tag{7.16}$$

where, $f^{[i]} = r * W^{[i]} + b^{[i]}$, and let's assume that we want to find the gradient of the loss function of the network with respect to $W^{[2]}$. Let's assume that the loss function is some function $L(F(x))$, here we will not concern our-self with what L is, we just know that the derivative of L is defined.

Now, Let's apply the chain rule to the loss function and try to find the gradient with respect to $W^{[2]}$

$$\frac{\partial L(F(x))}{\partial W^{[2]}} = \frac{\partial L(F(x))}{F(x)} * \frac{\partial F(x)}{\partial F^4(x)} * \frac{\partial F^4(x)}{\partial F^3(x)} * \frac{\partial F^3(x)}{\partial F^2(x)} * \frac{\partial F^2(x)}{\partial W^{[2]}} \tag{7.17}$$

where $F^i(x) = f^{[i]} \circ f^{[i-1]} ... \circ f^{[2]} \circ f^{[1]}(x)$ is the output of the ith layer. All the partial derivative in the above expression are either derivative of a linear function or of a non-linearity. We have calculated the derivatives of both types of functions above. So we have all the tools necessary to calculate the gradient of the loss function with respect to the parameters of the neural network.

Note: Notice, that in practical implementation, we don't need to store the output before and after all the activation functions, it saves us memory.

To Do: Some networks for typical problems, figure and exercises

Exercises

Exercise 7.1: Write a numpy based program to compute the forward and backward pass of a two layer neural network. Use this code to train a classifier for 2D data.

Exercise 7.2: Implement the above using PyTorch using SGD optimizer.