UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Master in Innovation and Research in Informatics

Algorithm Data Mining

Deep Learning

First Assignment

Mohana Fathollahi

March 26, 2022

1 Deep learning

Deep learning is a subset of machine learning that try to simulate behaviour of human brain. It is a neural network with three or more layers that can optimize its performance and increase accuracy. while in neural network with single layer we can find an approximation of prediction. In the figure 2 we can clearly see the difference between these two neural networks [1]

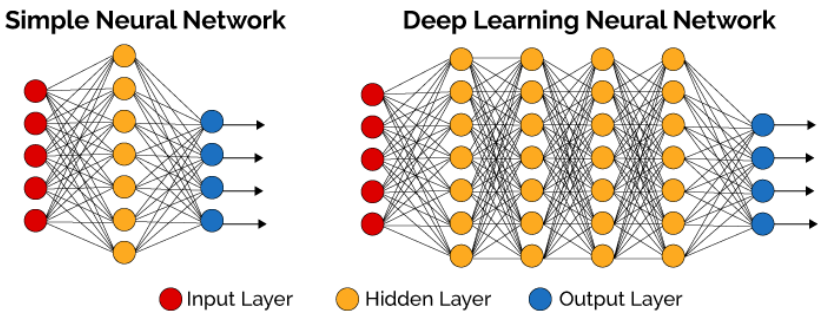

Figure 1: Compare two kind of Neural Networks

In this assignment, first structure of neural network and algorithm for dealing with cost function will be described and then approaches for weight initialization will be discused. At the end different activation functions and approaches to deal with overfiting in deep learning will be explained.

2 Structure of Neural Network

Neural network consist of small individual units called neurons. Neurons are similar to a biological neuron, receives input from other neurons, do some processing and produce a result. These similarity has been shown in figure 2.

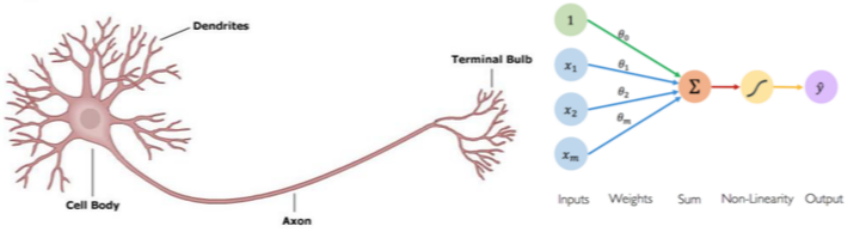

Figure 2: Similarity between biological and artificial Neural Networks

A neural network typically consists of three types of layers: Input Layer, Hidden Layer(s) and Output Layer [2].

- Input layer: The main role of input layer is receiving explanatory attributes for each observation. This layer has number of input nodes that is equal to the number of explanatory variables. For this layer we will not apply any calculations and data move to hidden layer.
- Hidden layer: The values that are coming from input layer or other hidden layers should multiply by weights. Therefore, we have vector of input values $x_1, .., x_d$ and combines it with some weights that are local to the neuron $(w_0, w_1, .., w_d)$ and Input for the next neuron will be based on below formula:

$$w_0 + \sum_{i=1}^{d} w_i x_i$$

  The special weight w_0 also known as bias. the question can be raised is that how can we define these weights? this question will be answered in Weight Initialization Techniques.

  Another thing that we should consider in hidden layers is selecting activation functions. Choosing appropriate activation function will help us to learn training data set in efficient way and get better accuracy.

  The choice of activation function in the output layer will define the type of predictions the model can make.
- Output layer: Output layer receives connections from hidden layers or from the input layer. Choosing activation function for this layers depends on the type of prediction problem.

3 Optimization

In the neural network like other algorithms we should try to minimize error, difference between grand truth and result of output layer. We need to have an optimization algorithm that can help us to find minimum value for loss function and one of straightforward algorithms is Gradient decent. It starts with initial point and calculates the gradient and makes a tiny step in the direction that optimizes the function. This is repeated over and over until we find the optimum. In the figure 3 these steps have been shown. Additionally, different approaches in gradient descent can be applied for neural network in this section some of these approaches are provided [3].

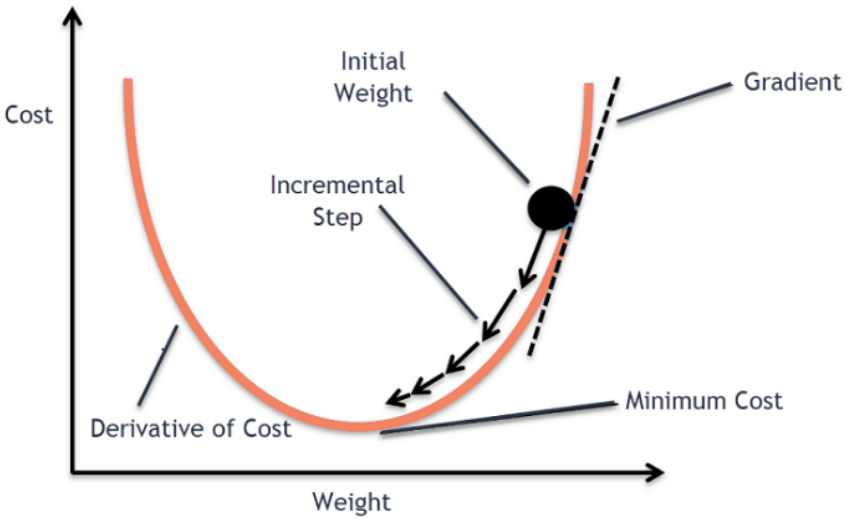

Figure 3: Gradient descent

3.1 Batch gradient descent

In batch gradient descent we use all training samples and calculate cumulative error, and then back propagate and adjust weight. This approach is useful when we have small training set. For example when we have 2 features and 10 million samples, we should find 20 million derivatives. But when we have 200 features, computation will be too much and needs a lot of time. Therefore, When number of training examples is large, then batch gradient descent is not a good option.

3.2 Stochastic gradient descent

In stochastic gradient descent SGD instead of using all the samples we can randomly pick single data and adjust weight. SGD is good when training set is very big and we do not want too much computation. Although, SGD often converges much faster compared to GD but the error function is not as well minimized as in the GD.

3.3 Mini batch gradient descent

Mini batch is like SGD but instead of choosing one randomly picked training sample, we will use a batch of randomly picked training samples. When we have a large training set, it process in batches of b and it will be faster than batch gradient descent and sgd.
In the figure 4 comparison for these approaches has been provide;

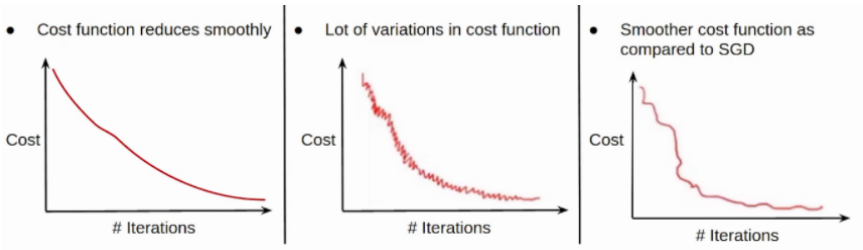

Figure 4: Comparing cost functions

In the left side figure Batch GD has been applied and because we used entire data set, cost function reduced smoothly.
For the plot in the middle, sgd has been used. Because just one example considered at a time the cost will fluctuate over the training example and it will not necessarily decrease but in the long run it will happen.
In the right side figure Mini-batch GD has been applied and the cost function is smoother than SGD because instead of considering a single observation a batch of observation considered.

3.4 Batch normalization

When we have imbalanced non-normalized data set, it will be drastically harder to train and decrease training speed. If one of weights in neural network is larger than another weights, imbalance continue to cascade through neural network that causing instability [4]. Steps that batch normalization will do when it is applying to a layer:

- Normalize output from activation function before being passed to the next layer as input

$$z = (x - m)/s$$

- multiply this normalized output by some arbitrary parameter

$$z * g$$

- add another arbitrary parameter to the result of previous step

$$(z * g) + b$$

Parameters such as mean, standard deviation, g and b are trainable and can become optimized during training process. Therefore, the weights do not become imbalanced with extremely high or low value since the normalization has been included in the gradient process. Applying batch norm can greatly increase the speed in which training occurs and reduce the ability of outlying large weights. Therefore, normalization can be applied in data before entering to input layer and output data from the activation functions for individual layers.

Moreover, to efficiently decrease the cost we may need to change all weights and biases. Backpropagation is an algorithm for tuning weights and find a better accuracy. In Backpropagation, we calculate partial derivative of cost function with respect to any weights and biases, therefore we will know that how changes in weights and biases can change the overall behaviour of the network. In the figure 5 by using gradients of loss function, values of the parameters from the last layer to the first layer have been updated [5].

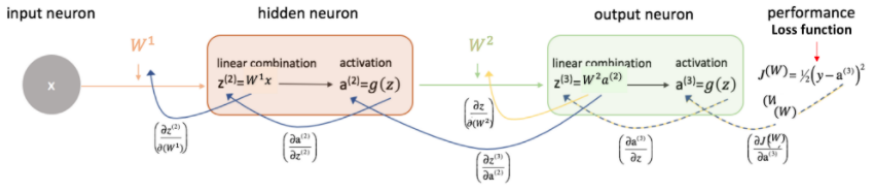

Figure 5: Backpropagation

In the following, effect of selecting different values for weights will be discussed.

4 Weight initialization techniques

If we do not initialize appropriate weights for neural network, loss gradients will either be too large or too small, and the network will take more time to converge if it is even able to do so at all. on the other side, if we initialized the weights correctly, optimization of loss function will be achieved in the least time [6], [7]. There are different techniques to initialize weights:

- Zero Initialization: If we assign zero to all weights, derivative with respect to loss function is the same for every weight. Hidden layers will be symmetric and result of neural network will not be better than a linear model.
- Random Initialization: In this technique, random values except 0 will be assigned to weights, therefore we will not have symmetric hidden layers and our loss function can be converged while for zero initialization loss function could not converge. To see effect of these two approaches on neural network, we can check this link[1].
  Two points that should be considered in random initialization are risk of selecting high or low values. For example if weights are initialized with very high values the multiplication of variables and wights will be high and if sigmoid activation function has been used, the slope of gradient changes slowly and learning rate will take time, this is often referred exploding gradient problem. When low values have been selected for weights, we will have same problem again in this case called vanishing gradient.
- Xavier Initialization: To mitigate the chances of exploding or vanishing gradients, this initialization can be used. In this method random initialization will be multiplied by $\sqrt{1/n}$, where n is the number of weights connected to this node from the previous layer. It is used for tanh() activation function.

[1] `https://www.deeplearning.ai/ai-notes/initialization/`

5 Activation functions

The performance of a neural network model will significantly depend on the type of activation function that we are going to use. Most of the time we should learn complex pattern, therefore we need to introduce some non-linear activation functions inside hidden layers [5]. There are some requirements for activation function that are described in below:

- Vanishing Gradient problem: this problem causes major difficulty when training a neural network. During training, gradient descent works to calculate the gradient of the loss with respect to weight in the network. Sometimes, gradient with respect to weights in earlier layers of the network becomes very small. Therefore, updating weights barely moved from its original value and weights will be stuck and do not move to optimal value. Therefore, we want a activation function to not shift the gradient towards low values.
- Differentiable: As mentioned before gradient descent will be used in neural network and it is based on derivative, therefore activation function should be differentiable.
- Zero-Centered: To avoid shifting in gradient to a particular direction, output of activation function should be symmetrical at zero.
- Computational Expense: In deep networks we may need to calculate activation function millions of times, therefore it should be inexpensive to be calculated.

In below some of non-linear activation functions have been described.

1. Logistic (Sigmoid):
   Output of sigmoid or logestic activation function will be between 0 and 1. This method is generally used for binary classification problems. But it does not cover all requirements that were mentioned before such as, computational expense, vanishing gradient problem and zero-centred.

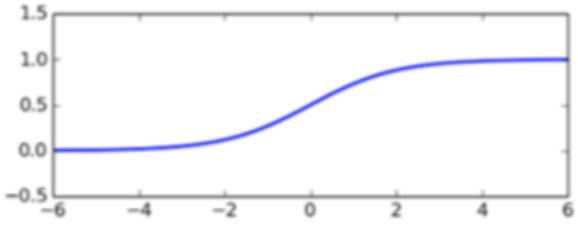

Figure 6: Sigmoid Activation function

2. Softmax:
   The more generalized form of Sigmoid is softmax and it used for multi-class classification problems. Similar to sigmoid, the range of output is in range of 0 and 1 and it is used as the final layer in classification models.
3. Hyperbolic Tangent (Tanh):
   Output will be in range of -1 and +1. Tanhs is a bit more robust regarding vanish-

ing gradients compare to sigmoid therefore using Xavier initilization can be a small improvement for initialization of weights for Tanh.

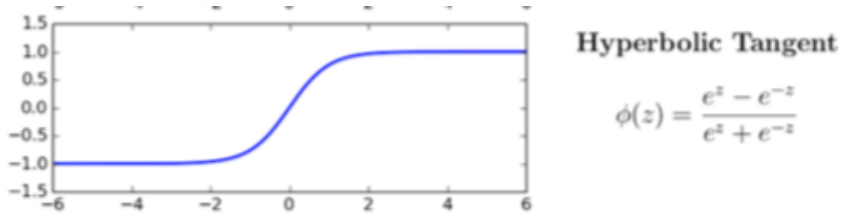

Figure 7: Tanh Activation function

4. ReLU (Rectified Linear Unit):
   It is one of popular activation function that is so useful in Convolutional Neural networks [2]. Its computation is easy and does not cause the Vanishing Gradient Problem. It has two problems; not being zero centred, the output is zero for all negative inputs. Therefore, some nodes will completely die and do not learn anything. second problem is related to higher limit that is inf and again leads to unusable nodes.

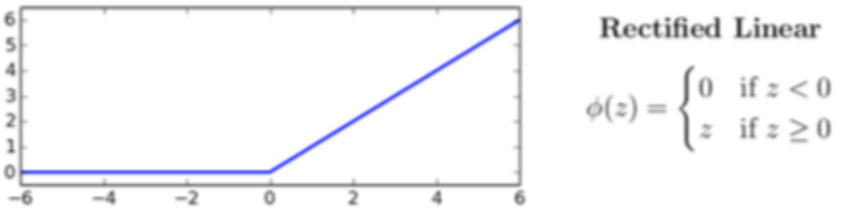

Figure 8: ReLU Activation function

5. Leaky ReLU: It is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope. In this function, problem of dying some nodes has been solved and the value that multiply with x can be considered as a hyperparameter for each neuron separately, and will give us parametric ReLU or PReLU.

[2] They are especially prevalent in image and video processing projects.

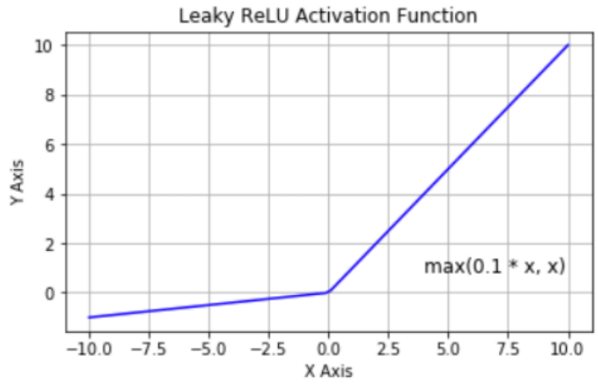

Figure 9: Leaky ReLU Activation function

5.1 Overfitting in deep learning

Deep learning neural networks are likely to overfit a training dataset with few examples. In this part some techniques for overfiting will be described [8].

1. L1 / L2 regularization:
   In L1 and L2, we will add a penalty term on the loss function, for example L2 regularization allows weights to decay toward zero and in L1 regularization allows weights to zero.

2. Remove layers / number of units per layer:
   In over complex models, with reducing complexity we can overcome overfiting. For example by removing layers and reduce the size of model we can create a balance between underfitting and overfitting.

3. Dropout:
   Dropout tries to randomly drop some neurons during the training phase and convert their weight to zero. When we drop different sets of neurons, it's equivalent to training different neural networks. So, the dropout procedure is like averaging the effects of large number of different networks. It considered as a form of ensemble learning. In ensemble learning; each classifier has been trained separately, it has learned different 'aspects' of the data and their mistakes are different. Combining them helps to produce an stronger classifier, which is less prone to overfitting.

References

[1] *Deep learning vs Machine Learning.* URL: https://levity.ai/blog/difference-machine-learning-deep-learning#:~:text=Machine%20learning%20means%20computers%20learning,as%20documents%2C%20images%20and%20text..

[2] *Structure of neural network.* URL: https://medium.com/@rinu.gour123/artificial-neural-network-for-machine-learning-structure-layers-2a275f73f473.

[3] *Gradient Descent.* URL: https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a#:~:text=Batch%20Gradient%20Descent%20can%20be,converges%20faster%20for%20larger%20datasets..

[4] *Batch normalization.* URL: https://en.wikipedia.org/wiki/Batch_normalization.

[5] *Activation functions.* URL: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253#:~:text=Differentiable%3A%20As%20mentioned%2C%20neural%20networks,work%20as%20activation%20function%20layer..

[6] *weights initialization.* URL: https://www.analyticsvidhya.com/blog/2021/05/how-to-initialize-weights-in-neural-networks/.

[7] *weight-initialization-techniques.* URL: https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78.

[8] *Overfiting.* URL: https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d#87f3.