**COMP 642 — Machine Learning** Jan 27, 2022

# Lecture 6: Deep Learning

*Lecturer: Anshumali Shrivastava*
*Scribe By: Chris Hager, Gafar Alli, Nonso Nwadialo, Yemi Dada*

**Disclaimer:** *These lecture notes are intended to develop the thought process and intuition in machine learning. The materials are not thoroughly reviewed and can contain errors.*

## 1 Evolution of Machine Learning – From Single Layer to Multilayer Perceptrons

In traditional ML techniques, the role of the domain expert was critical in feature selection. Possessing the required knowledge provided the modeler with some intuition on which features would affect the dependent variable, and which were redundant. The idea of neural networks obviated the need for feature selection as the multiple layers learn more complicated and abstract representations of the input data. In deep learning, the idea is to learn future levels of increasing abstraction with minimum human contribution (Bengio 2009). In most complex problems, there is no way to fully define the structures and inter-dependencies present in the domain.

### 1.1 Artificial Neural Networks

**Building blocks:**

- Input Layer – number of neurons equals the number of input features.
- Hidden Layer – one or many hidden layers with variable number of neurons between input and output layers. In general, each neuron in the hidden layer is connected to all neurons in the adjacent layers ("fully connected").
- Output Layer – number of neurons equals the number of target features.

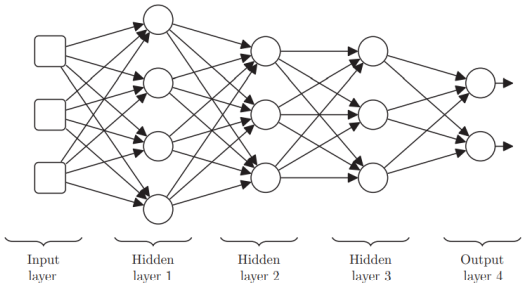


Figure 1: Neural network architecture

[1]An artificial neural network consists of a network of simple information processing units, called neurons. The power of neural networks to model complex relationships is not the result of complex mathematical models, but rather emerges from the interactions between a large set of simple neurons. Figure 1 illustrates the structure of a neural network. It is standard to think of the neurons in a neural network as organized into layers. The depicted network has five layers: one input layer, three hidden layers, and one output layer. A hidden layer is just a layer that is neither the input nor the output layer. Deep learning networks are neural networks that have many hidden layers of neurons. The minimum number of hidden layers necessary to be considered deep is two. However, most deep learning networks have many more than two hidden layers. The important point is that the depth of a network is measured in terms of the number of hidden layers, plus the output layer. In figure 1, the squares in the input layer represent locations in memory that are used to present inputs to the network. These locations can be thought of as sensing neurons. There is no processing of information in these sensing neurons; the output of each of these neurons is simply the value of the data stored at the memory location. Deep learning networks are neural networks that have many hidden layers of neurons.

The circles in the figure represent the information processing neurons in the network. Each of these neurons takes a set of numeric values as input and maps them to a single output value. Each input to a processing neuron is either the output of a sensing neuron or the output of another processing neuron. The arrows in figure 1 illustrate how information flows through the network from the output of one neuron to the input of another neuron. Each connection in a network connects two neurons and each connection is directed, which means that information carried along a connection only flows in one direction. Each of the connections in a network has a weight associated with it. A connection weight is simply a number, but these weights are very important. The weight of a connection affects how a neuron processes the information it receives along the connection, and, in fact, training an artificial neural network, essentially, involves searching for the best (or optimal) set of weights

## 1.2 Neural Network Deep Learning as Feature Learning:

A well-known intuition about deep learning is that it finds more abstract and useful representations of a feature vector (X) that sheds redundancies, while retaining as much information about the output vector (Y) as possible.

- Extracts important features by iteratively transforming the data, going deeper towards meaningful patterns with each transformation
- The modeler does not need to select or encode features ahead of time
- Deep learning automatically learns features that matter. It is a part of the larger field of work called 'representation learning'

## 1.3 Forward Propagation

Before discussing the back-propagation algorithm, it is important to discuss how the neural network calculates its output given a set of inputs. Forward propagation can be visualized as a long series of nested equations. The deeper you go, the more abstract, the representations of the input get.

[1]Deep Learning, John Kelleher, MIT Press Essential Knowledge Series 2019

$$F(x) = A(B(C(x)))$$

Let's assume we have a network architecture as shown in Fig.2 and our input data is a [i, j] matrix where i equals the number of records and j equals the number of input features. Each neuron in the input layer represents a [i, 1] vector of a specific feature (X1, X2).

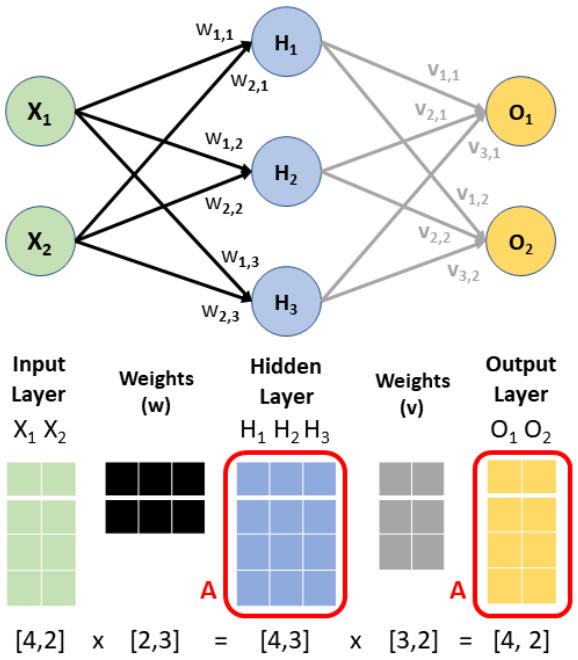


Figure 2: Forward propagation across NN layers

The operations to get from input to output layer are as follows:

1. Multiply the input feature matrix [i, j] with the weights matrix w [j, n], where n equals the number of neurons in the hidden layer. The resulting matrix has the dimensions [i, n]. See Fig.3.

2. Pass this matrix through the activation function (A) to calculate the values for the neurons in the hidden layer (H1, H2, H3).

3. Multiply the hidden layer matrix [i, n] with the weights matrix v [n, m], where m equals the number of neurons in the output layer. The resulting matrix has the dimensions [i, m].

4. Again, pass this matrix through the activation function (A) to calculate the values for the neurons in the output layer (O1, O2).

In mathematical terms we can write the result of this forward propagation step, which is just a series of matrix multiplications with element-wise application of an activation function (A), as:

$$A(v \cdot (A(w \cdot X)))$$

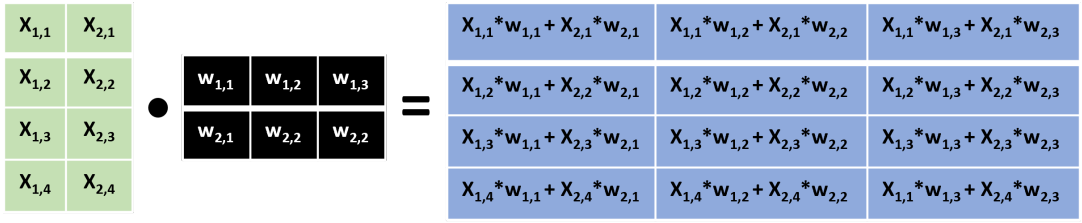


Figure 3: Example of matrix multiplication in Step 1

For more complex networks with more than one hidden layer and variable number of neurons, we repeat steps 1 and 2 until we arrive at the output layer. All the layer matrices will have dimensions of [i, k] where k equals the number of neurons of the current layer. The associated weight matrices, to calculate values for the next layer, will have dimensions of [k, l] where l equals the number of neurons in the next layer.
Adding bias to the network can be accomplished by just adding another feature with a constant value across all records (see Fig.4). The bias is just another node in the layer and the weight matrix will have an additional row for the bias weights. Bias neurons can be added at each layer of the network.

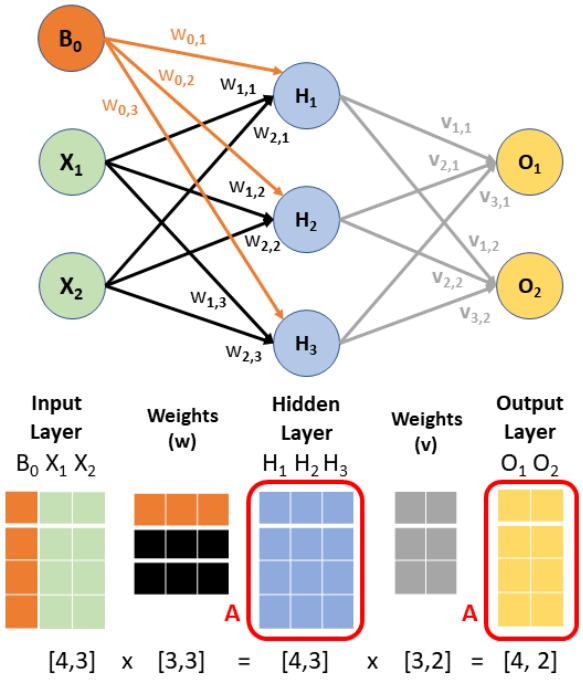


Figure 4: Forward propagation with bias

Activation functions do not have to be the same for all layers. Typically, all hidden layers use the same activation function (e.g. ReLu is a popular choice) but the activation function for the output layer depends on the type of prediction required by the model. Classification models require activation functions that transform the numerical values to probabilities (ranging from

0-1, e.g. sigmoid, softmax) whereas any type of activation function can be used in regression models.

**Why do we need to add non-linearity to network through activation functions?**

As shown above, forward propagation is nothing more than the weighted sum of the inputs which is equal to linear regression. If all layers have linear activation functions, then the entire network is just an accumulation of linear regressions which results in a linear mapping of input to output. As a consequence, the network "collapses" to a simple input-output architecture with no hidden layers. It follows that the network will not be able to find a function that explains any non-linear behavior of our data. Adding non-linearity to the network brings us closer to the goal of a "universal function approximator". In linear regression we tried to accomplish this goal by feature engineering (increasing the dimensions of our input data to be able to fit a boundary between classes).

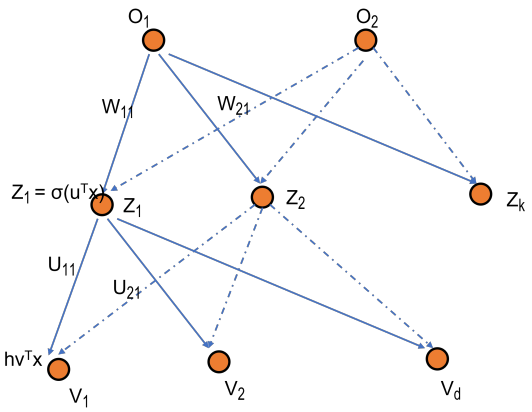


Figure 5: Neural network with non-linearity ($\sigma$)

## 1.4 Back Propagation

It is a common misconception to think that the backpropagation algorithm is the whole learning algorithm for multi-layer neural networks. Backpropagation is simply the algorithm that is used to calculate gradients for the parameters in each layer. Gradient Descent is the optimization algorithm that is used to train the model, and it needs gradients to be calculated for each parameter before new values can be calculated. The gradients are supplied using the backpropagation algorithm, relying heavily on the chain rule of calculus.

## 1.5 GPUs and Matrix Multiplication

Graphics Processing Units (GPUs) were developed originally to meet the computational needs of algorithms for rendering computer graphics. The rapid and enormous growth in sophistication of graphics applications such as computer games has resulted in the availability of GPUs that have hundreds of processors and peak performance near a teraflop and that sell for hundreds of dollars to a few thousand dollars. Although GPUs are optimized for graphics calculations, their low cost per gigaflop has motivated significant research into their efficient use for non-graphics applications. The effort being expended in this direction has long-lasting potential because the widespread use of GPUs in the vibrant computer games industry almost ensures the longevity of GPUs in other applications. GPUs have found favor in the field of neural networks as they allow us perform multiplication of huge matrices at a fraction of the time and cost, compared to using CPUs. The distributed nature of neural networks make them the perfect paradigm for parallel processing.