

# NLP4Free



A Free Natural Language Processing Microcourse

<https://github.com/nlpfromscratch/nlp4free>

## Part 5 – Deep Learning for Natural Language



Myles Harrison,  
Founder and Trainer

# Agenda

**01** Front Matter

**02** Neural Network Fundamentals

**03** Training a Feed-Forward Network

**04** Conclusion

# Licensing and Usage

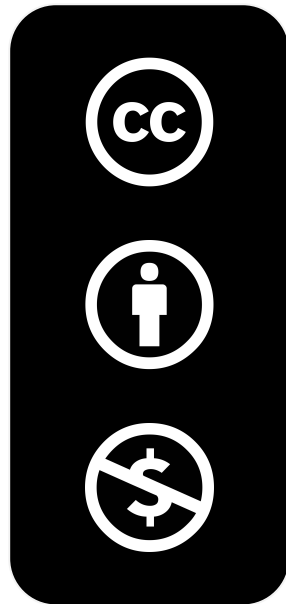
This work is licensed under a [Creative Commons Attribution Non-Commercial License](#).

You are free to:

- **Share:** copy and redistribute the material in any medium or format
- **Adapt:** remix, transform, and build upon the material

Under the following terms:

- **Attribution:** You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **NonCommercial:** You may not use the material for commercial purposes.



# Credit Where Due

Though this work is based upon my experiences consulting and teaching data and machine learning, including that for natural language, all materials within have been compiled or created by myself.

When I have included or relied upon others' materials such as images, code, or text, I have done my best to cite as appropriate and provide links to the source.

# **Neural Network Fundamentals**

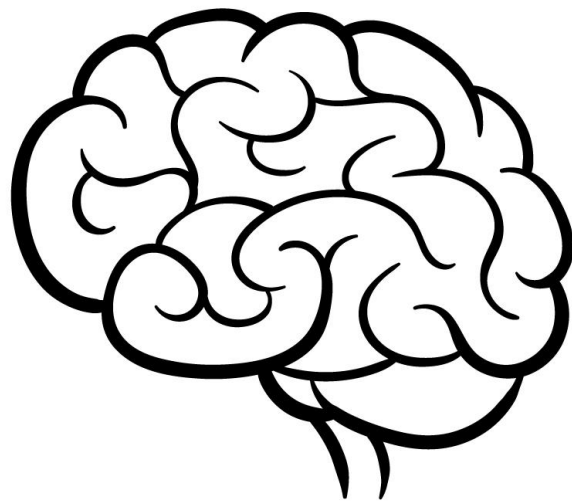
# What is Deep Learning?

Deep Learning is a specialized type of machine learning that takes motivations from the structure of the human brain.

Unlike other machine learning models, deep learning models - or artificial neural networks - are composed of many nodes which can be viewed as individual "sub-models"

The theoretical foundations for deep learning have existed since the 1960s (or even earlier), but it only recently been realized with the rise of inexpensive and powerful computing available at scale.

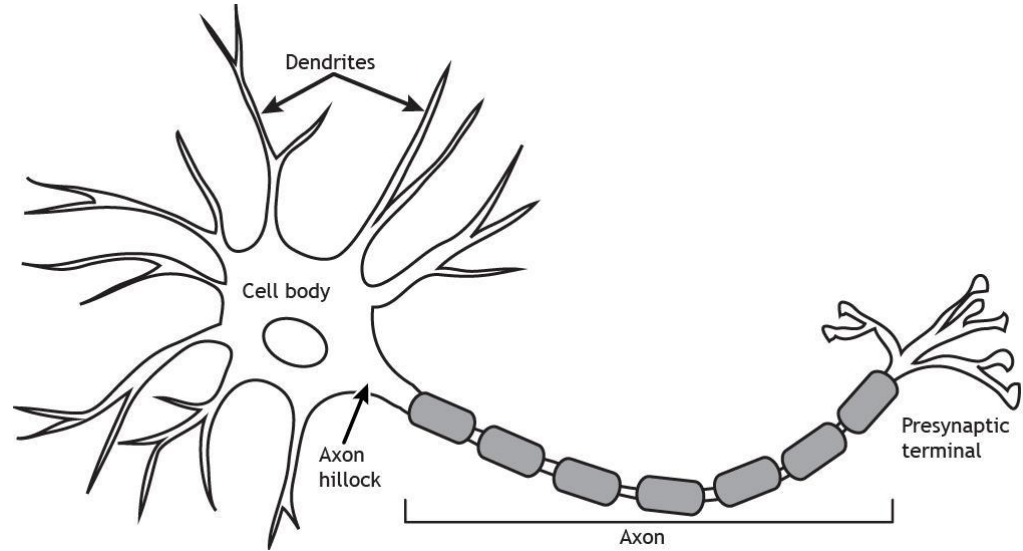
In NLP, deep learning models form the basis for state-of-the-art large language models (LLMs).



# A Neuron in the Brain

The human brain is composed of billions of neurons, electrically excitable cells composed of a cell body, dendrites, an axon, and terminal.

Neurons receive input through their dendrites, and when firing, an electrical impulse travels down the axon to the terminal and release neurotransmitters to the next cell.



# An Artificial Neuron (Perceptron)

An artificial neuron, referred to as a *perceptron*, is structured similarly: inputs to the model are the data plus a constant which are then multiplied by a set of *weights* (corresponding to dendrites in a physical neuron).

These together make a weighted sum of the inputs, which are processed through an *activation function* producing the perceptron output or *activation*, analogous to the axon and terminal in a physical neuron firing.

Many perceptrons combined together make up what was historically referred to as a Multilayer Perceptron (MLP) that we now refer to as a feed-forward, or fully connected, neural network.

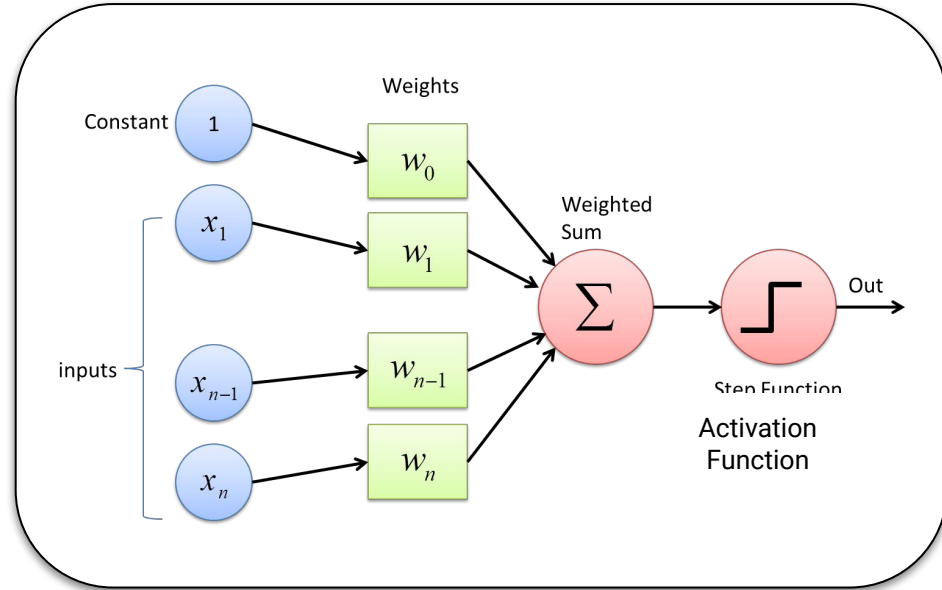


Image from:

<https://deepai.org/machine-learning-glossary-and-terms/perceptron>



# Structure of A Neural Network

Multiple perceptrons are put together into *layers* composed of *nodes* (each perceptron) to create a *neural network*.

The outputs, or *activations*, which come out of previous layers become the inputs of the following layer.

The number of layers and number of nodes in the network - known as its *architecture* - is arbitrary and up to the modeler. There are also specific architectures that are well suited to particular types of problems.

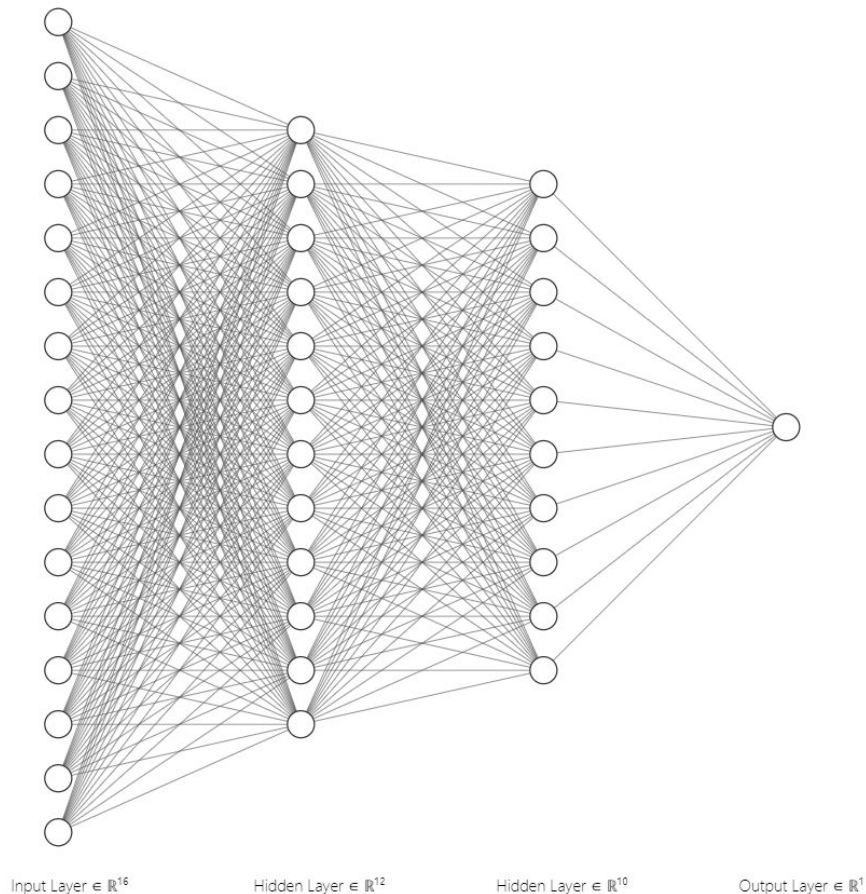



Image from: <https://alexlenail.me/NN-SVG/>

NLP from scratch 

# Structure of A Neural Network

Multiple perceptrons are put together into *layers* composed of *nodes* (each perceptron) to create a *neural network*.

The outputs, or *activations*, which come out of previous layers become the inputs of the following layer.

The number of layers and number of nodes in the network - known as its *architecture* - is arbitrary and up to the modeler. There are also specific architectures that are well suited to particular types of problems.

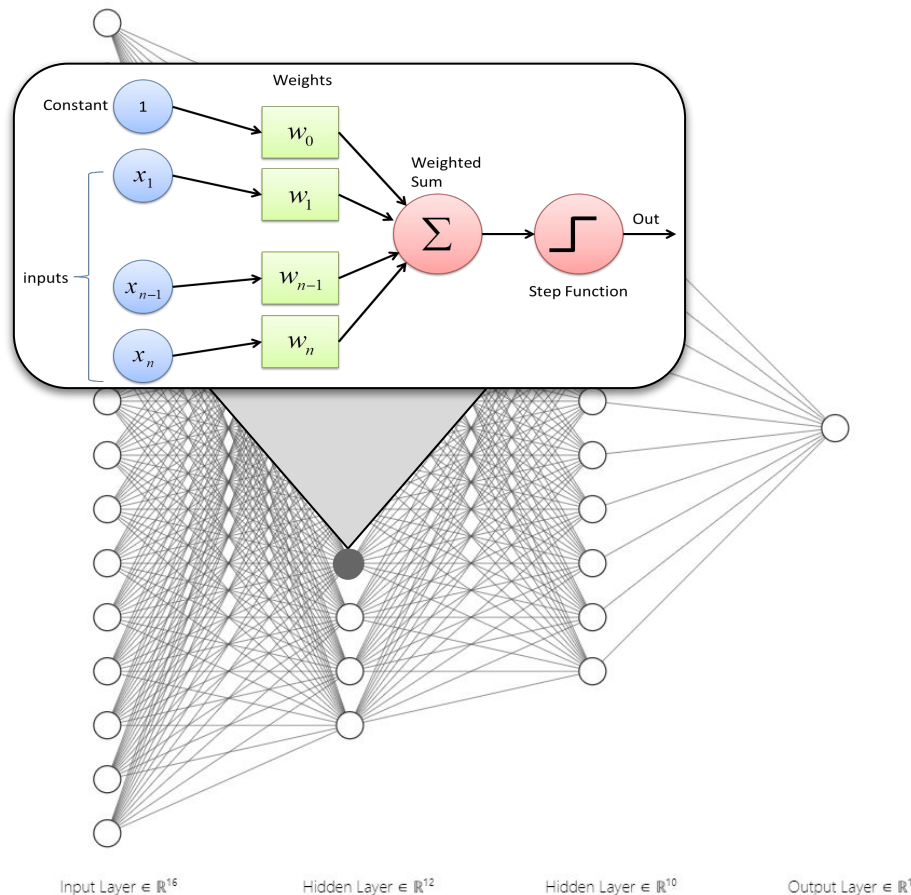


Image from: <https://alexlenail.me/NN-SVG/>

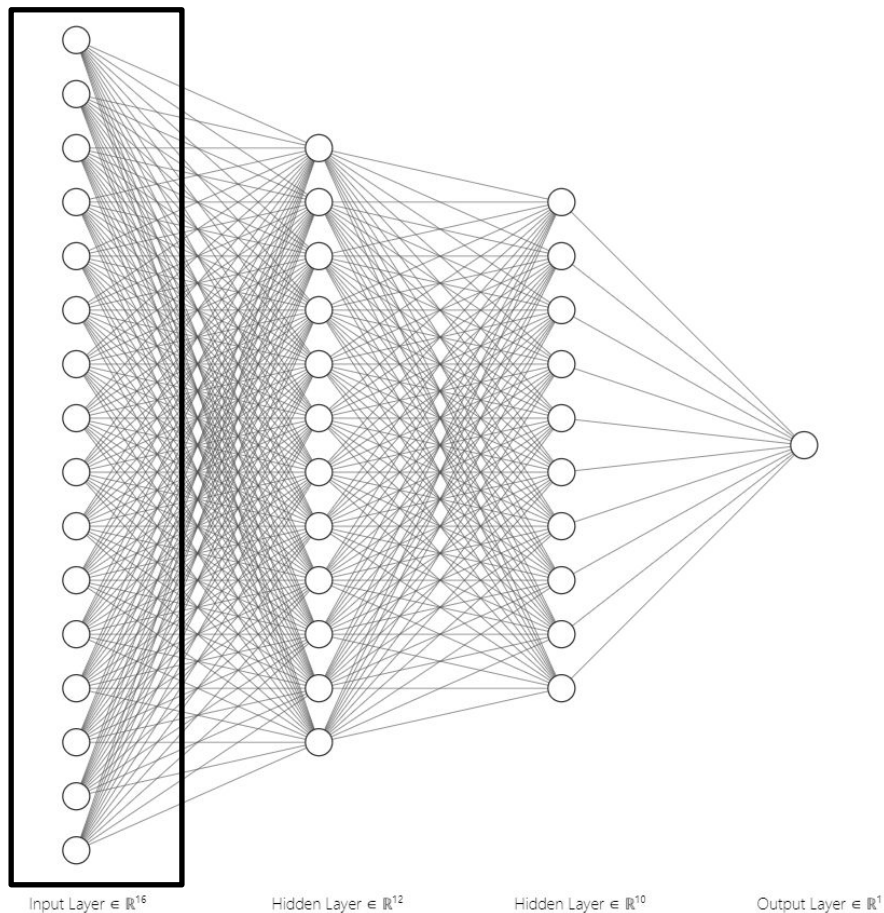
NLP from scratch

# Input Layer

The *input layer* is not a “true” layer but just passes the data through to the following layers - we say either that there is no activation or that the activation function here is a linear passthrough (the function  $y = x$ ).

Each neuron in the input layer represents a feature of the data, so there will be an equal number of nodes in the input layer as there are features in the data.

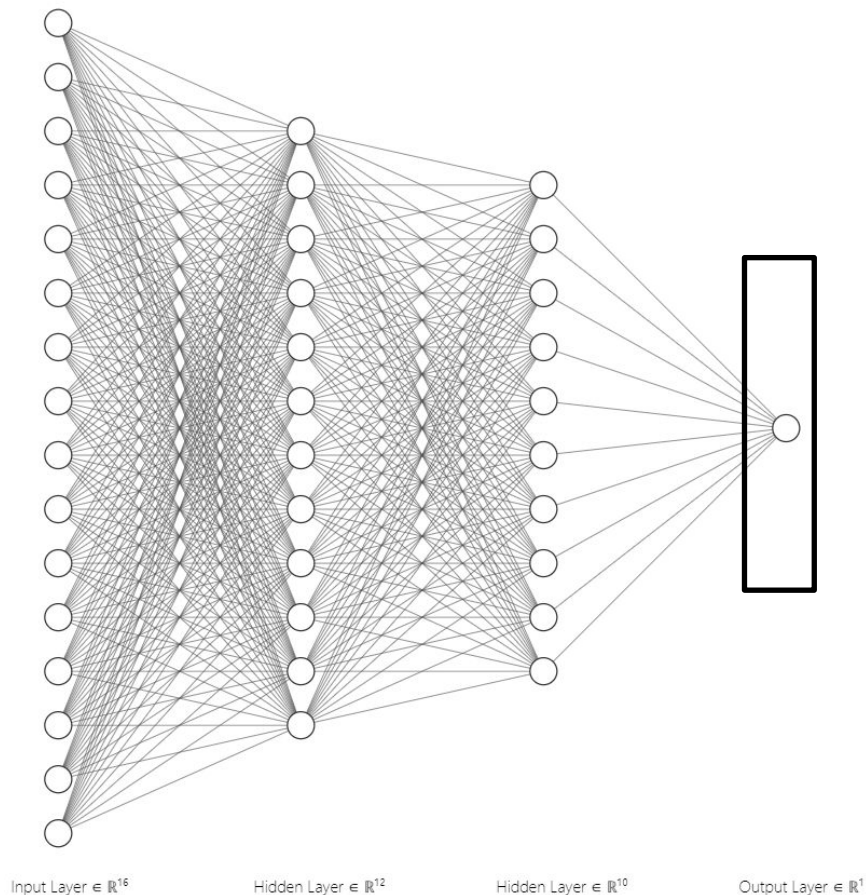
For example, the neural network depicted on the right could be used to make predictions if we tabular data with 16 features (columns) describing each observations (row) in the dataset.



# Output Layer

On the other hand, the *output layer* is the final layer of a neural network that produces the network's predictions (output). The number of nodes in the output layer is dependent on the problem type.

A single node can be used for binary classification or regression, since for each observation of input there will only be one output: a probability between 0 and 1 of lying in the positive class (class 1) in the case of binary classification, and a single numeric value in the case of regression - predicting a continuous value associated with each observation of the input data.

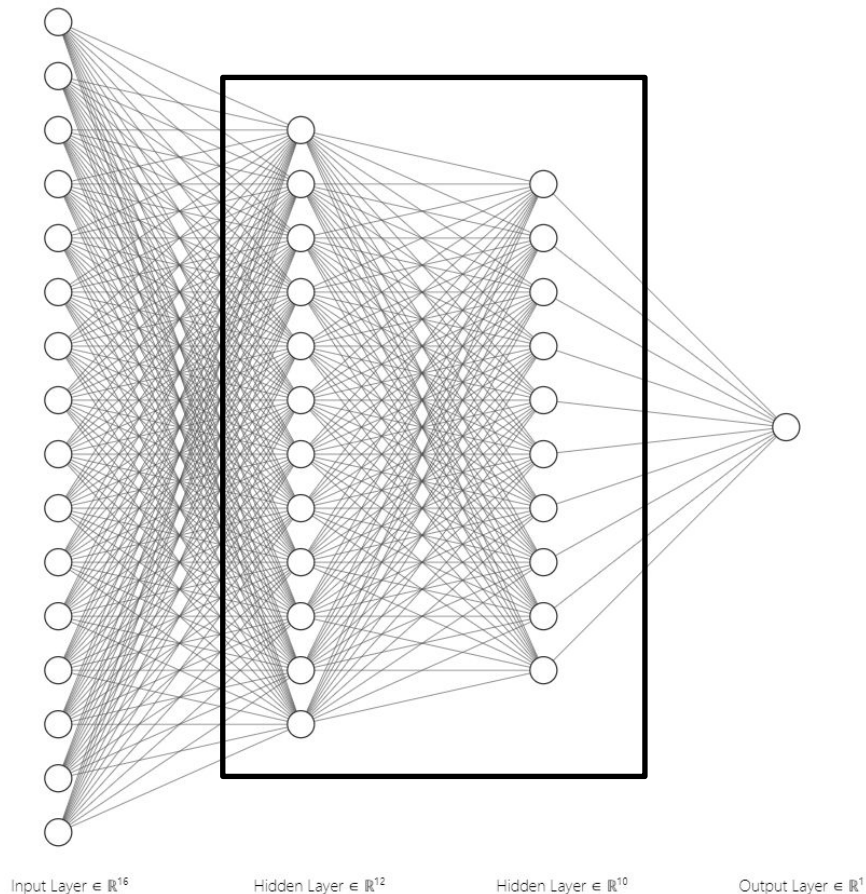


# Hidden Layers

The intermediate layers between the input and output of the network are the so-called *hidden layers*.

Each hidden layers perform computations on outputs of previous layer, a linear combination of these multiplied by coefficients - or *weights*. The activation function is then applied to this result. The weights are what is learned by the model in training.

Here we are speaking only of *fully-connected (feed-forward)* networks, the simplest type of neural network. There are other layer types for different types of models specifically suited to certain types of tasks (e.g. computer vision)



# Activation Functions

The ability for neural networks to learn highly complex, non-linear relationships is greatly due to activation functions. As these are applied at each layer, as the information flows through the network from left to right, the functions are applied atop one another in previous outputs.

Each layer may have a different type of activation function, though it is not uncommon to use the same for most, if not all of the layers. Again, these choices are up to the modeler.

There exist families of well-known mathematical functions that perform well and have desirable mathematical properties and serve as standard choices to use when training deep learning models.

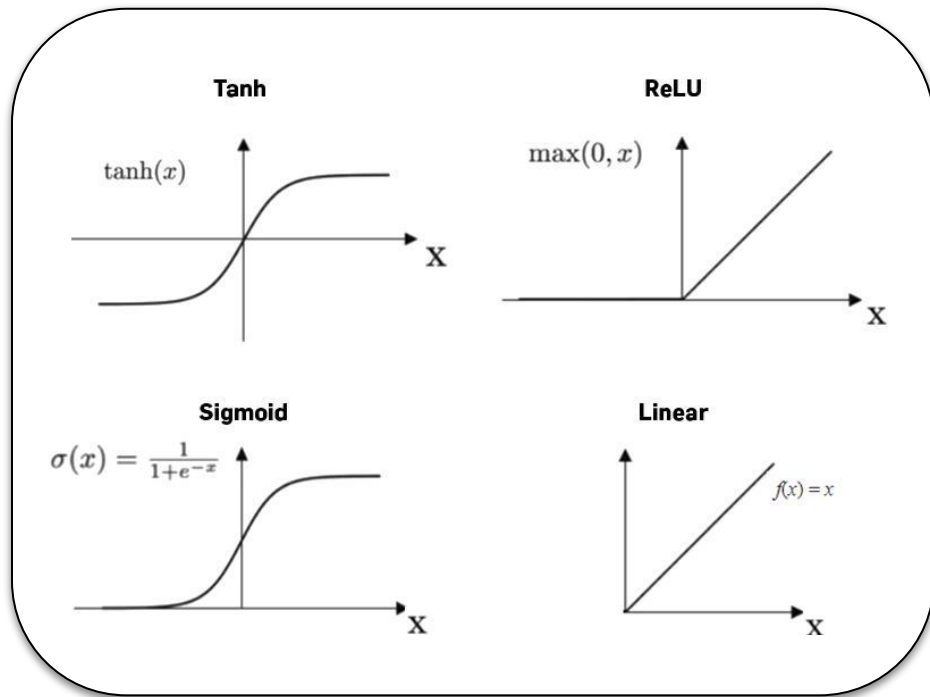


Image source: <https://machine-learning.paperspace.com/wiki/activation-function>

# Loss Functions

Every neural network also has a *loss function*. This is just a technical term for a function which measures the model's error. They compares the values output from the network to those expected or known to compute the error.

Depending on the type of problem the network is being applied to, there are different families of loss functions which are used.

Any machine learning model is never "right" - it is only less wrong, and the goal of training a neural network is to find the optimal values for the weights such that the loss (error) is minimized.

But how do we find these values for the weights? That brings us to the details of how neural networks are trained.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$L(\hat{y}, y) = - \sum_k y^{(k)} \log \hat{y}^{(k)}$$

# How do Neural Networks Learn?

The details of how neural networks are trained and the optimal values for the model weights found are quite mathematically complex.

At a high level, it can be viewed as an optimization problem where we want to minimize the error (loss) as a function of how much we should change the weights.

Imagine you wish to find the lowest point in a mountain range - which way should you walk to make the steepest descent and reach the bottom?

Analogously, there are numerical methods which can perform computations over many millions of parameters in a neural network to determine how to adjust the weights to those which minimize the error.

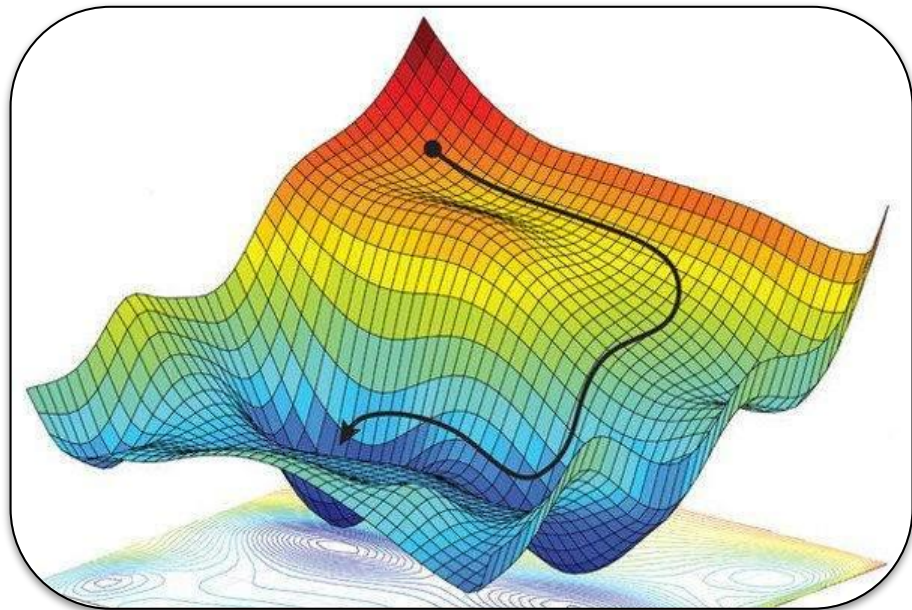


Image from:  
<https://www.mygreatlearning.com/academy/learn-for-free/courses/stochastic-gradient-descent>

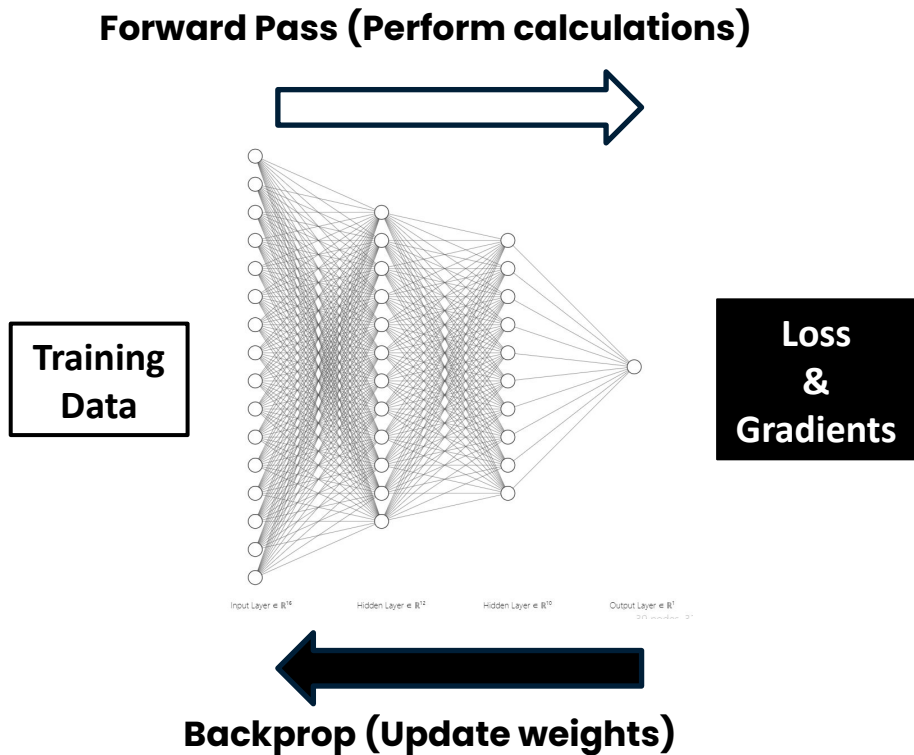


# Forward Pass and Backpropagation

This logic is applied during neural network training. Unlike other types of machine learning models, neural networks are trained in two steps which are repeated many times.

In the *forward pass*, data is run through the network to compute the output, and error calculated from the loss function comparing this output against the known true value associated with the input.

*Backpropagation* ("backprop") follows, and applies changes to weights' values in the network as determined from *gradients*, or slopes, the direction of greatest decrease of error.



*NLP from scratch*

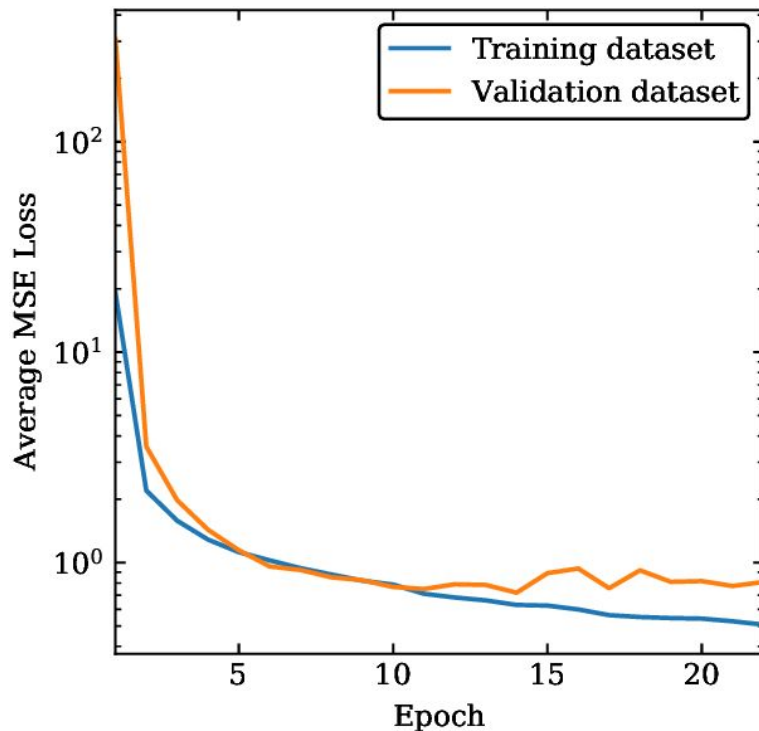
# Epochs and Batches

In addition to differing from other types of machine learning based on these two steps, deep learning also differs in that data is not passed through the network once in its entirety, but in smaller subsets known as *batches*.

When all the data has gone through the network once, this is referred to as a single *epoch* of training

One epoch is composed of many batches, and networks are trained for many epochs and see the whole training dataset multiple, even hundreds or thousands of times, depending on the problem.

With each epoch of training, the model weights are adjusted and the network better learns the relationships between the features and target variable.



# (Python) Neural Network Frameworks



TensorFlow

- Google product
- Graph-based computation, GPU training
- Other deployment options (Tensorflow Lite, TF.js)
- Easy with integration of Keras into TF 2.x



PyTorch

- Facebook product
- Graph-based computation, GPU training
- Pytorch Mobile for embedded, no web (ONNX?)
- OOP dev focus (ML eng), Lightning equivalent to Keras

# Deep Learning for Sentiment

Let's see what we covered in action!

Download the Jupyter Notebook to run  
locally, or open in Colab.



Run in Google Colab



Download as .ipynb

# End of Part 5

## Deep Learning for Natural Language

# So long!

This concludes the course.

If you enjoyed it, please share with anyone else you'd think would benefit from it.

You can also [connect with me on LinkedIn](#).

If you'd like to learn more about natural language processing and dive deeper, please check out the workshops I offer at [NLP from scratch](#).

Thanks for reading!



 Thanks!



[nlpfromscratch.com](https://nlpfromscratch.com)



[linkedin.com/company/nlp-from-scratch](https://linkedin.com/company/nlp-from-scratch)



[youtube.com/@nlpfromscratch](https://youtube.com/@nlpfromscratch)