Neural Networks

* Loosely modeled after our brains
* Layers of nodes
* Input layer, hidden layer(s), output layer
* Data is transmitted across layers
* Each node has incoming streams and outgoing streams
* Nodes may send data across these streams to a node in a layer above and may receive data from nodes in a layer below
* For every input stream, the node assigns a weight to it
* Data is multiplied by the weight and all products are summed up
* If this value is more than a threshold value, the value is pushed across all its output streams to nodes in the next layer
* Networks are trained to find the optimal weights and threshold values
* A model is considered correct if it outputs the correct label (for an object recognition network, for example) consistently for its training examples.
* A fully connected model means each node connects to every other node in layers on either side.
* Deep learning refers to the number of layers (depth) of a neural network
* Feed-forward network refers to the way data moves through the network (from input to hidden to output)
* Perceptron is the first neural network, with only one hidden layer
* Optimized settings (weights/ threshold values) which output the correct label do not help us in any way (these numbers cannot be demystified)
* Their mechanisms were mysterious; we moved to support vector machines – an alternative to neural networks which rely on elegant and clean mathematics
* Gaming industry with its huge requirement of computing power (GPU’s) brought back neural networks
* **Backward propagation** is used to help train a neural network
* <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>

Data:

* **Integer Encoding:** Giving integer values to each categorical value (blue = 1, red = 2, green = 3)
* **One-Hot Encoding**: Some categorical variables do not have such a hierarchy, and integer encoding places importance on some values over others (higher values mean “more importance”). So, we give red = (1,0,0), blue = (0,1,0), and green = (0,0,1), where each place represents a different color. The 0/1 in a particular place tells us if the corresponding color exists; for this, the places are (red, blue, green)

Picking a Neural Network:

* Loss Functions:
  + **Regression**: (Minimum) Mean Squared Error Loss (minimizing MSE)
    - Similar to Least Squares, except we take the average of the squares and aim to minimize that quantity
  + **Classification**: Cross Entropy Loss
    - Analogous to log loss (used for binary classification), where the model output is compared to the true label.
    - Error increases when we are “confidently incorrect”, or when the model strongly believes an incorrect label (the opposite of the true label) and dwindles as the model predicts the true label
* <https://rohanvarma.me/Loss-Functions/#:~:text=The%20main%20difference%20between%20the,from%20a%20maximum%20likelihood%20estimate>
* <https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#hinge>

Backward Propagation:

* A technique used to train the model

Perceptron:

* Single-level neural network (one hidden layer) is a perceptron
* A multi-level perceptron is a neural network
* A binary linear classifier
* Consists of 4 parts:
  + Input layer
  + Weights and Bias
  + Net Sum
  + Activation Function
* Bias is used to shift activation function up/down
* Weights are used to show the strength of a node
* Activation functions are used to map the input (the weighted sum) between required values ([0,1] or [-1,1] for example)
* **This output is used when determining if the node should “fire” or not**
* Usually used to classify data into 2 parts
* <https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53>

Gradient Descent:

* Iterative optimization algorithm used to find the parameters for the minimum of a loss function (error function)
* Compute the gradient of the loss function (partial derivatives for each parameter)
* Begin with arbitrary parameter values
* Find gradient by plugging in parameter values into the corresponding partial derivative
* Take bigger steps when the gradient is very large and smaller steps as you get closer to the minimum (gradient begins to level out) (this happens automatically as the gradient is scaled by the learning rate -> larger gradient means large step sizes)
* This helps make sure to not overshoot the minimum
* The learning rate helps with this (start bigger, then become small as gradient becomes small) -> too big and you overshoot the minimum, too small and it may be inefficient
* Step size = gradient × learning rate
* New parameter value = old parameter value – step size
* Loss function can be anything (for linear regression, maybe use sum of squared residuals)
* Gradient descent ends when step size is or is close to 0 (which means gradient must be very small)
* Stochastic gradient descent:
  + With big data, there are millions of data points
  + As a result, the loss function becomes enormous and calculating its gradient for each parameter value is time-consuming
  + Solution is to only use a single data point per epoch
  + Short-term, the loss function may not decrease (considering only a single data point at a time), but over the long run, there is a downward trend
* Batch gradient descent:
  + Take the average of gradients of all training data for each epoch; this is the traditional way of gradient descent
  + Used for smooth curves
  + Use the entire data set per epoch
* Mini-Batch Gradient descent:
  + Combination of batch and stochastic gradient descent which uses a random subset of the dataset on each iteration in order to minimize the calculus and still incorporate all the data (not just one data point per epoch but also not the entire data set)
  + The loss function still fluctuates like stochastic gradient descent but now we can update our parameters frequently (unlike batch) and still use a vectorized implementation (more than a single data point) (unlike stochastic)
* <https://www.youtube.com/watch?v=sDv4f4s2SB8>
* <https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a>