Introduction to Neural Networks

Neural Networks and Deep Learning are behind most of the AI that shapes our everyday life. Think of how you interact everyday with these technologies just by using the greatest features in our phones (face-recognition, autocorrect, text-autocomplete, voicemail-to-text previews), finding what we need on the internet (predictive internet searches, content or product recommendations), or using self-driving cars. Also, some of the classification and regression problems you need to solve, are good candidates for Neural Networks and Deep Learning as well.

Some basic facts of Neural Networks:

- Use biology as inspiration for mathematical models
- Get signals from previous neurons
- Generate signals according to inputs
- Pass signals on to next neurons
- You can create a complex model by layering many neurons

The basic syntax of Multi-Layer Perceptrons in scikit learn is:

```
# Import Scikit-Learn model

from sklearn.neural_network import MLPClassifier

# Specify an activation function

mlp = MLPClassifier(hidden_layer_sizes=(5,2), activation= 'logistic')

# Fit and predict data (similar to approach for other sklearn models)

mlp.fit(X_train, y_train)

mlp.predict(X_test)
```

These are the main parts of MLP:

- Weights
- Input layer
- Hidden Layer
- Weights
- Net Input
- Activation

Deep Learning Use Cases Summary

Method	Use case
Neural Network Models: Multi-Layer Perceptron, Feedforward Networks	Applied to many traditional predictive problems (classification and regression, tabular data)
Recurrent Neural Networks (RNN, LSTM)	Useful for modeling sequences (time-series forecasting, sentence prediction)
Convolutional Neural Networks (CNN)	Useful for feature and object recognition in visual data (images, video). Also applied in other contexts (forecasting)
Unsupervised Pre-trained Networks: Autoencoders, Deep Belief Networks, and Generative Adversarial Networks.	Many uses including generating images, labeling outcomes, dimensionality reduction

Training a Neural Network

In a nutshell this is the process to train a neural network:

- Put in Training inputs, get the output.
- Compare output to correct answers: Look at loss function J.
- Adjust and repeat.
- Backpropagation tells us how to make a single adjustment using calculus.

The vanishing gradient problem is caused due to the fact that as you have more layers, the gradient gets very small at the early layers. For this reason, other activations (such as ReLU) have become more common

The right activation function depends on the application, and there are no hard and fast rules. These are the some of the most used activation functions and their most common use cases:

Method	Use case
Sigmoid Activation	Useful when outcomes should be in (0, 1), suffers from vanishing gradient issues
Hyperbolic Tangent	Useful when outcomes should be in (-1, 1), suffers from vanishing gradient issues
ReLU	Useful to capture large effects, doesn’t suffer from vanishing gradient
Leaky ReLU	Acts like ReLU, but allows negative outcomes