Chapter 10: Introduction to Artificial Neural Networks with Keras

- From Biological to Artificial Neurons
  - Biological Neurons
  - Logical Computations with Neurons
    - McCulloch and Pitts: network of neurons can compute any logical proposition
  - The Perceptron
    - Threshold logic unit, linear threshold unit
    - Weighted sum of inputs then step function
    - Step functions
      - Heaviside: 0/1
      - Sign: [-1,0,1]
    - Fully connected layer
    - Hebb's rule: cells that fire together, wire together
    - Perceptron learning rule:
      - $w_{i,j}^{(next\ step)} = w_{i,j} + \eta \left(y_j - \hat{y}_j \right) x_i$
      - η is the learning rate
    - Decision boundary of each output neuron is linear
      - Perceptrons incapable of learning complex patterns
    - Similar to Stochastic Gradient Descent
    - Limitations: unable to solve Exclusive OR (XOR) classification problem → MLPs can solve
  - The Multilayer Perceptron and Backpropagation
  - Regression MLPs
  - Classification MLPs
- Implementing MLPs with Keras
  - Installing TF2
  - Building an image classifier using the Sequential API
    - Sequential model: simplest model, single stack of layers connected sequentially
    - Layers:
      - Flatten: X.reshape(-1,1)
      - Dense: hidden layers, or output, initializes weights randomly
      - Can set kernel_initializer or bias_initializer
    - Compiling the model
      - Sparse_categorical_crossentropy for target class index
      - Categorical_crossentropy for one-hot encoded
      - Convert sparse labels (e.g., class indices) to one-hot vector labels use keras.utils.to_categorical(), to get back np.argmax()
      - Validation_split set instead of passing validation_data
      - Class_weight in fit() method to give larger weights to underrepresented classes
      - Sample_weight is per instance weights

      - History.history contains loss and other metrics
      - Training error is running mean during epoch, validation error @ end of epoch
      - Not happy with model results:
        - Change learning rate then optimizer
        - Then # of layers, # of neurons, activation function @ each layer
        - Then batch size (default batch_size=32)
  - Building a regression MLP using the Sequential API
    - keras.layers.Dense(1)
  - Building complex models using the Functional API
    - Wide & Deep NN: connects all or part of inputs to output layer
      - Learns deep patterns and simple rules
      - Sequential forces all data to flow through full stack → simple patterns could get distorted
    - Called functional b/c calling layers like functions
    - Use case for multiple outputs:
      - Task driven: locate and classify main object in a picture
      - Multiple independent tasks based on same data
      - Regularization
  - Using the Subclassing API to build dynamic models
    - Sequential and Functional are declarative → static
  - Saving and restoring a model
    - Code: model.save(‘my_model.h5’), keras.models.load_model(‘my_model.h5’)
  - Using callbacks
    - check_pt = keras.callbacks.ModelCheckpoint(‘my_mode.h5’
      - model.fit(X_train, y_train, epochs = #, call_backs = [check_pt])
    - Early stopping:
      - early_stopping = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)
      - model.fit(…., call_backs=[check_pt, early_stopping])
  - Using TensorBoard for Visualization
    - Visualize learning curves
- Fine-tuning Neural Network Hyperparameters
  - Number of Hidden Layers
    - Parameter efficiency: deep networks can model more complex functions with fewer neurons than shallow ones
    - Learn hierarchical structure
  - Number of Neurons per Hidden Layer
    - Common to size to form a pyramid
    - Simpler to pick model with more layers and neurons than need and use early stopping
    - More bang from increasing layers than neurons

- Learning rate, batch size, and other hyperparameters
    - Learning rate: most important
        - If start small and increase, optimal rate just b4 loss function ↑
    - Optimizer: chap 11 discusses in detail
    - Batch size: affects performance and training time
        - GPU's can process efficiently → training algo sees more instances per second, but may lead to instability, poor generalization
    - Activation function: ReLU usually good default
    - Number of iterations: don't need to tweak, stop early!