Guide for Building Neural Network Models in Keras

<https://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/>

<https://keras.io/>

5 step cycle

1. Define Network

* Choice of activation function for the output layer is the most important layer since it will define what the output will look like
* General Rules of Thumb to Follow
* **Regression**
  + Linear activation function
  + Ensure number of output layers match input (same dimension)
* **Binary Classification (2-class)**
  + Logistic activation function or “sigmoid and one neurone in the output layer
    - Don’t need 2 since value of 0 can correspond to one class and 1 can correspond to the other class
* **Multiclass Classification (>2 class)**
  + Softmax activation function and one output neuron per class, assuming a one-hot encoded output pattern
* To define networks in Keras
* model = Sequential()
  + Must create an instance of the Sequential class, allowing you create your layers and add to the graph
* Add layers using model.add(#define instance of whatever layer you are trying to add) e.g. model.add(Dense(2))
  + Adds a dense layer with 2 nodes to the model
* Note you don’t need to add a separate layer for the input, in the first hidden layer, specify input\_dim = x as a parameter
  + E.g:
  + model=Sequential()
  + model.add(Dense(2, input\_dim=2))
  + #add activation layers after each layer as such
  + model.add(Activation(‘relu’))
* Can do model.summary() to get a brief overview of what the computation graph looks like

1. Compile Network

* Efficiency step – converts simple graph into an efficient set of matrix transformations which can be executed with a GPU
* **Required step** before training/loading pre-trained weights
* Must provide the optimizer (usually use Adam which is a variant of gradient descent) and the loss function (specific to the type of problem)
* Also allows certain metrics e.g. a useful one for classification is accuracy – provided as an array passed in (metrics=[‘accuracy’, etc.])
* Some standard loss functions for different perspective model types
  + **Regression**: Mean squared error
  + **Binary Classification:** Logarithmic loss or cross entropy
  + **Multiclass Classification:** Multiclass Logarithmic loss
* Common optimizers
  + **Stochastic gradient descent:** requires tuning of learning rate and momentum
  + **Adam:** requires tuning of learning rate
  + **RMSpropl:** requires tuning of learning rate

1. Fit Network

* Train on the training data and adjust weights using backpropagation
* model.fit(x, y, batch\_size=.., epochs=..)
* Input and target sizes must match the corresponding network
* Batch size defines the number of patterns the network is exposed to before the weights are updated within an epoch

1. Evaluate Network

* Can evaluate the network on a separate dataset
  + **NOT accurate** to let the model test on the training data since it has seen it before
* The mode will evaluate the loss across all test patterns and any other metrics specified when the model was compiled such as accuracy for a classification task
* loss, accuracy = model.evaluate(X, y)

1. Make Predictions

* To predict on new input data – easy as:
* Predictions = model.predict(x)
* This will be returned in the format provided by the network
* **Linear Regression:** will be in the format of the problem directly
* **Binary Classification:** probability between 0 and 1 that can be converted to the desired output by rounding
* **Multiclass Classification:** array of probabilities that can be converted to the desired output using the argmax function