Open in Colab

T81-558: Applications of Deep Neural Networks

Module 8: Kaggle Data Sets

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 8 Material

- Part 8.1: Introduction to Kaggle [Video] [Notebook]
- Part 8.2: Building Ensembles with Scikit-Learn and Keras [Video] [Notebook]
- **Part 8.3: How Should you Architect Your Keras Neural Network: Hyperparameters** [Video] [Notebook]
- Part 8.4: Bayesian Hyperparameter Optimization for Keras [Video] [Notebook]
- Part 8.5: Current Semester's Kaggle [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
# Startup CoLab
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False


# Nicely formatted time string
def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return "{}:{:>02}:{:>05.2f}".format(h, m, s)
```

```
Note: not using Google CoLab
```

Part 8.3: Architecting Network: Hyperparameters

You have probably noticed several hyperparameters introduced previously in this course that you need to choose for your neural network. The number of layers, neuron counts per layer, layer types, and activation functions are all choices you must make to optimize your neural network. Some of the categories of hyperparameters for you to choose from coming from the following list:

- Number of Hidden Layers and Neuron Counts
- Activation Functions
- Advanced Activation Functions
- Regularization: L1, L2, Dropout
- Batch Normalization
- Training Parameters

The following sections will introduce each of these categories for Keras. While I will provide some general guidelines for hyperparameter selection, no two tasks are the same. You will benefit from experimentation with these values to determine what works best for your neural network. In the next part, we will see how machine learning can select some of these values independently.

Number of Hidden Layers and Neuron Counts

The structure of Keras layers is perhaps the hyperparameters that most become aware of first. How many layers should you have? How many neurons are on each layer? What activation function and layer type should you use? These are all questions that come up when designing a neural network. There are many different types of layer in Keras, listed here:

- **Activation** - You can also add activation functions as layers. Using the activation layer is the same as specifying the activation function as part of a Dense (or other) layer type.
- **ActivityRegularization** Used to add L1/L2 regularization outside of a layer. You can specify L1 and L2 as part of a Dense (or other) layer type.
- **Dense** - The original neural network layer type. In this layer type, every neuron connects to the next layer. The input vector is one-dimensional, and placing specific inputs next does not affect each other.
- **Dropout** - Dropout consists of randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. Dropout only occurs during training.

- **Flatten** - Flattens the input to 1D and does not affect the batch size.
- **Input** - A Keras tensor is a tensor object from the underlying back end (Theano, TensorFlow, or CNTK), which we augment with specific attributes to build a Keras by knowing the inputs and outputs of the model.
- **Lambda** - Wraps arbitrary expression as a Layer object.
- **Masking** - Masks a sequence using a mask value to skip timesteps.
- **Permute** - Permutes the input dimensions according to a given pattern. Useful for tasks such as connecting RNNs and convolutional networks.
- **RepeatVector** - Repeats the input n times.
- **Reshape** - Similar to Numpy reshapes.
- **SpatialDropout1D** - This version performs the same function as Dropout; however, it drops entire 1D feature maps instead of individual elements.
- **SpatialDropout2D** - This version performs the same function as Dropout; however, it drops entire 2D feature maps instead of individual elements
- **SpatialDropout3D** - This version performs the same function as Dropout; however, it drops entire 3D feature maps instead of individual elements.

There is always trial and error for choosing a good number of neurons and hidden layers. Generally, the number of neurons on each layer will be larger closer to the hidden layer and smaller towards the output layer. This configuration gives the neural network a somewhat triangular or trapezoid appearance.

Activation Functions

Activation functions are a choice that you must make for each layer. Generally, you can follow this guideline:

- Hidden Layers - RELU
- Output Layer - Softmax for classification, linear for regression.

Some of the common activation functions in Keras are listed here:

- **softmax** - Used for multi-class classification. Ensures all output neurons behave as probabilities and sum to 1.0.
- **elu** - Exponential linear unit. Exponential Linear Unit or its widely known name ELU is a function that tends to converge cost to zero faster and produce more accurate results. Can produce negative outputs.
- **selu** - Scaled Exponential Linear Unit (SELU), essentially **elu** multiplied by a scaling constant.
- **softplus** - Softplus activation function. $log(exp(x) + 1)$ Introduced in 2001.
- **softsign** Softsign activation function. $x/(abs(x) + 1)$ Similar to tanh, but not widely used.

- **relu** - Very popular neural network activation function. Used for hidden layers, cannot output negative values. No trainable parameters.
- **tanh** Classic neural network activation function, though often replaced by relu family on modern networks.
- **sigmoid** - Classic neural network activation. Often used on output layer of a binary classifier.
- **hard_sigmoid** - Less computationally expensive variant of sigmoid.
- **exponential** - Exponential (base e) activation function.
- **linear** - Pass-through activation function. Usually used on the output layer of a regression neural network.

For more information about Keras activation functions refer to the following:

- Keras Activation Functions
- Activation Function Cheat Sheets

Advanced Activation Functions

Hyperparameters are not changed when the neural network trains. You, the network designer, must define the hyperparameters. The neural network learns regular parameters during neural network training. Neural network weights are the most common type of regular parameter. The "advanced activation functions," as Keras call them, also contain parameters that the network will learn during training. These activation functions may give you better performance than RELU.

- **LeakyReLU** - Leaky version of a Rectified Linear Unit. It allows a small gradient when the unit is not active, controlled by alpha hyperparameter.
- **PReLU** - Parametric Rectified Linear Unit, learns the alpha hyperparameter.

Regularization: L1, L2, Dropout

- Keras Regularization
- Keras Dropout

Batch Normalization

- Keras Batch Normalization

- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. Can allow learning rate to be larger.

Training Parameters

- Keras Optimizers
- **Batch Size** - Usually small, such as 32 or so.
- **Learning Rate** - Usually small, 1e-3 or so.

In [ ]: