Notebook Assignment 4 - Due Date TBD:

Deep Learning Software Frameworks (updated: November 14, 2024)

TensorFlow Keras Libraries: There are several python software llibraries used for deep learning. One of these is the TensorFlow/Keras library. This assignments has you work through a textbook example that instantiates and trains a convolutional neural network using samples from the MNIST database.

MNIST is a large database of handwritten digits $(0-9)$ that is used to train various image processing systems. The database contains 60,000 training images and 10,000 testing images. All images are 28×28 monochrome images where each pixel value is an unsigned 8 bit integer $(0-255)$. This section will demonstrate how TensorFlow can be used to instantiate, train, and evaluate a simple 2D convolution neural network model that takes the pixel image and recognizes which digit that image represents.

(1) Load the MNIST database

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels),(test_images,test_labels) = mnist.load_data()
```

The basic data type used in TensorFlow is a *tensor*. A tensor is a multi-dimensional array where the number of dimensions is called the tensor's *rank*. We usually specify a rank-n tensor's *shape* by the n-tuple, $(x_1, x_2, \ldots, x_n)$, where x_i is the number of array components along that ith dimension. A vector $x = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$, may therefore be seen as a rank-1 tensor with shape $(3,)$. A matrix $x = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ is a rank-2 tensor with shape $(3, 3)$. The database `train_images` is initially a rank-3 tensor of shape $(60000, 28, 28)$. Write a script that takes 5000 of the training images and reshapes them into a rank-4 tensor of shape $(5000, 28, 28, 1)$. Your script should take 5000 `train_labels` and reshape it into into a tensor of shape $(5000, 1)$. Repeat for the testing data, but only extracting 1000 samples.

(2) The data arrays generated in the preceding script are actually Numpy arrays. Training of neural network models will be faster if we use these Numpy arrays to form *dataset* objects. Neural network training is usually not done on the whole dataset at once. Instead, we update a model's weights using a smaller *batch* of inputs in the training set. This is called *mini-batch* training and it can be done more efficiently

by the computer if we create a *dataset* object to be used in training. A dataset object is an *iterator* that can be called recursively by the model's training method. One advantage of the dataset object is that it can divide up the dataset into batches, that can be called more quickly at training time. The original image data array consists of unsigned 8 bit integers in the range $0 - 255$. Neural network models train better if the data type is floating point and if they are scaled to $[0, 1]$. Write a script that uses TensorFlow's Dataset object to build the training and test dataset objects with batches of size 32 and where the data has been retyped as "float32" and scaled to be between 0 and 1.

(3) We instantiate a TensorFlow `model` object by declaring the layers in the model and then chaining them together. The following TensorFlow script shows how to do this for a simple sequential model with two convolutional layers. This model consists of 5 layers

$$\text{input} \to \text{Conv2D} \to \text{Conv2D} \to \text{Flatten} \to \text{Dense}.$$

The input layer fixes the shape of the input tensors to $(28, 28, 1)$. The first two dimensions are called *spatial dimensions* and the third dimension is called a *channel*. For MNIST images, the channel only has 1 component because all images are monochrome. If these images had been RGB color images, then this third dimension would have 3 components. These inputs are fed to convolutional layers. Convolutional layers (`Conv2D`) are special layers that perform spatial convolutions on their input tensors. The output from these layers will be rank-3 tensors where the number of channels is specified as an argument in the layer's constructor. In general the spatial dimensions of this output tensor will be slightly smaller due to the convolution operation. These layers also specify an activation function, ReLu. The output of the convolutional layers is a rank-3 tensor, but the output of the model is going to be a rank-1 tensor of shape (10), one component for each of the 10 digits we want to recognize. The last two layers (`Flatten` and `Dense`) reshape the convolutional layer's outputs into the shape. The `Dense` layer is a fully connected layer similar to what we see in an MLP's hidden layer. In this example, we use a softmax activation function so the model outputs are real-valued numbers between 0 and 1.

(4) We are now ready to train the model declared above using the dataset objects we created. Training is done by evaluating the gradient of the empirical risk evaluated

for a mini-batch of data and then using that gradient to update the trainable weights. This procedure is done multiple times as part of a gradient search algorithm. Before conducting the gradient search, we need to configure the structures used to compute the gradient. This is done through the model's `compile` method. Write a script that compiles the model to use sparse categorical crossentropy loss function with the rmsprop optimizer using accuracy as the main performance metric.

(5) Train the model using the `fit` method for 30 epochs with the training dataset, `train_ds`, and using the testing data, `test_ds`, for validation. We also want you to use a callback in during training that saves the model with the smallest validation (testing) loss. Your function call should return an array `history` that can be used to plot the model's training losses as a function of epoch.