Assignment #1
ADL, Spring 2019
Due Feb 14th

Note: updated the EC on Feb 3.

Hello World in Two Styles

About: In this assignment, you will gain experience implementing neural networks to classify the Fashion MNIST data, using two different development styles. I thought it'd be helpful for you to see both of these early (if you're familiar with both, you can branch out to any major framework that exists today).

- In part one, you'll write code using a Sequential style (your model is defined by a graph of layers.

- In part two, you'll use a Subclassing style (your model is defined by extending a class, then writing the forward pass imperatively).

Submission instructions: Please submit this assignment on Piazza by uploading a Jupyter notebook. You may submit a single notebook for the entire assignment, or you may submit a zip including one notebook for each part. Reminder, please be sure your notebooks include saved output that shows the results of training your models.

Extra credit: There's an extra credit section at the bottom of this assignment. As always, it's optional. If you choose to complete some or all of it, please include your work with your submission.

Part 1 (50 points)**:** Implement and evaluate two models to classify the Fashion MNIST dataset using the Keras Sequential API.

Tip: You should base your work off of this tutorial, and/or this example. We recommend doing this assignment in Colab (you can download a Jupyter notebook from there for your submission - be sure to save your output first). For this part of the assignment, it does not matter whether you have TensorFlow v1 or v2.0 preview installed (TensorFlow v1 is installed on Colab by default).

1. Implement and train a linear model to classify this dataset. Evaluate it by producing a plot that compares the training and validation accuracy. Include this plot with your submission.

2. Implement and train a deep neural network to classify this dataset. No need to produce plots for this part, just try to get the validation accuracy as high as you can.

3. Produce a diagram that visualizes your linear and deep models using plot_model. You can find a complete example of how to use plot model here. Include these plots with your submission.

4. Produce a confusion matrix for one of your models. You can reuse the sklearn code (it's great). You'll need to use your model to make predictions on the images in the test set, and compare them with the correct answer. Include the confusion matrix in your submission.

Part 2 (50 points)**:** Implement and evaluate two models to classify the Fashion MNIST dataset using the Keras Subclassing API.

Tip: You should base your work off this example (which shows how to install TensorFlow 2.0 in Colab, and demonstrates how to use the Subclassing API on a similar dataset). For this part of the assignment, you will need to install the TensorFlow 2.0 preview (we recommend using Colab, as in the notebook above, so you can install it easily). If you ever need to reset your Colab environment, you can go to *Runtime -> Reset all runtimes*.

5. Implement and train a linear model to classify this dataset. Evaluate it by producing a plot that compares the training and validation accuracy. Include this plot with your submission.

6. Implement and train a deep model to classify this dataset. No need to produce plots for this part, just try to get the validation accuracy as high as you can.

Extra credit

You may complete some of all of these problems, in any order.

EC1: Provide your own implementation of a squared error loss function, and run an experiment to compare it to cross entropy. What do you find?

Note: this part was updated on Feb 3. Originally, we were going to write our own implementations of softmax and cross entropy loss, but as the docs for TF 2.0 are still underway, decided to upload a complete example of that, for you to learn from. If you did the previous version of this, no worries, you'll get credit for it. If not, try this one.

We talked a little bit in class about cross entropy loss, and why it's preferable to classification error. But, we didn't compare it squared error. Could you use squared error to train a DNN for classification? Why or why not? Using the custom loss function example as starter code, write your own loss function for squared error (do not use built-in methods). Next, design and run an experiment to compare it cross entropy. What do you find?

EC2: Provide your own implementation of a Dense layer.

So far we've used high-level building blocks to define our layers and loss. If you look at example 2.2 on GitHub, you'll find this block of code:

```
class MyModel(Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.flatten = Flatten()
    self.d1 = Dense(128)
    self.d2 = Dense(10)
```

The built-in Dense layers are cool, but can you write one of your own? Please do so, and use it to create your model (as opposed to the existing Dense layer class). Run a short experiment comparing your implementation to the built-in one, and include saved results in your submission.

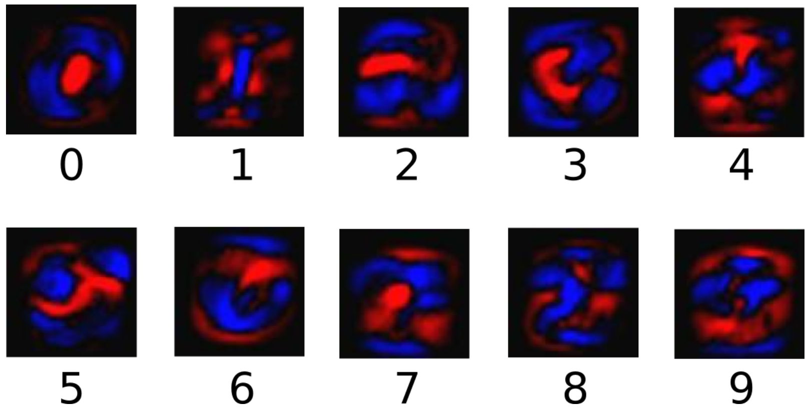
Tips: Use the code from part two as a starting point. See this example to learn how to write a custom layer.

EC3: Visualize the learned weights.

Using the Subclassing API as in part two, write and train a linear model to classify the MNIST dataset. After it's trained, extract the learned weights from the Dense layer, and produce a visualization similar to the one below using Matplotlib.

Hint: recall MNIST images are 28x28 = 784 unrolled pixels. These are fed into your Dense layer, which connects to 10 outputs (so you have 784 x 10 = 7840 weights). The import thing is to realize you have 784 weights per output digit (one for each pixel). What would you see if you reshaped those weights back into a 28x28 image?

Write code to do so, and produce a visualization similar to the one below. Include it with your submission.


0
1
2
3
4
5
6
7
8
9