Assignment 10

Cloth Classification using RNN and CNN

Computing Lab II, Spring 2021

The objective of this assignment is to train a Recurrent Neural Network (RNN) and a Convolutional Neural Network (CNN) using Tensorflow, for the task of image classification. You may use Tensorflow 2, but for this assignment we recommend using v1 functionality as follows:

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

Dataset: Fashion MNIST Dataset. You can load the dataset from tf.keras directly as shown in the following tutorial: https://www.tensorflow.org/tutorials/keras/classification with the following lines of code:

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_data, train_labels), (test_data, test_labels) =
fashion_mnist.load_data()
```

Loading the dataset returns four NumPy arrays:

- The train_data and train_labels arrays are the *training set*—the data the model uses to learn.
- The model is tested against the *test set*, the test_data, and test_labels arrays.

The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255.

Class Labels: Each training and test example is assigned to one of the following labels:

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag

- 9 Ankle boot

You should further split the training data into training and validation sets using Scikit Learn's train_test_split method: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
You should record both the training and validation errors at the end of each epoch (do not train on the validation set, just record the error).

You don't have to submit any data files in your final submission.

You may use the **early stopping** technique to decide when to finish training. A simple algorithm for this would be as follows:

1. Train on training data.
2. Find validation set performance at regular intervals.
3. If validation set performance doesn't improve over k iterations (called patience), stop training. This is called **early stopping.**
4. Load best model weights based on validation set.
5. Find performance on the test set.

Task 1

You will create a recurrent neural network with a tensorflow.compat.v1.nn.rnn_cell.RNNCell as a basic component. Use tensorflow.compat.v1.nn.static_rnn to create the RNN. Train it end to end.

RNN specifics:

1. Hidden state size: 100
2. Timesteps for unfolding the RNN: 28
3. Non-linear activation: tanh
4. Loss function: categorical softmax cross entropy with logits
5. Regularization: Use dropout with retention probability of 0.8
6. Optimizer: Adam

Save the weights corresponding to the best validation accuracy in a folder 'logs_rnn/' and report the results on the validation and test set.

Task 2

You will create a 2D convolutional neural network using tensorflow.compat.v1.layers.conv2d as a basic component. Train it end to end.

CNN specifics:

1. 1st convolutional layer: 32 filters of dimension 5x5 followed by batch normalization, relu activation and max pooling (with 2x2 subsampling).
2. 2nd convolutional layer: 64 filters of dimension 5x5 followed by batch normalization, relu activation and max pooling (with 2x2 subsampling).
3. Flatten above layer and feed it to a densely connected layer with 1024 hidden units
4. Add an output dense layer with 10 logits followed by batch normalization and softmax activation.
5. Loss function: categorical softmax cross entropy with logits
6. Regularization: Use dropout with retention probability of 0.6
7. Optimizer: Adam

Save the weights corresponding to the best validation accuracy in a folder ‘logs_cnn/’ and report the results on the validation and test set.

Submit a brief report on what inferences you could draw from the experiments (in pdf only). The report should contain information on model hyperparameters and the training, validation and test accuracies.

Constraints and Considerations:

1. You must NOT use tensorflow.keras functions other than for loading the dataset. Define the RNN using the functions in tensorflow.compat.v1.nn only. Similarly use only the functions in tensorflow.compat.v1.layers for the CNN.
2. Apart from the above restriction you have the freedom to choose the details of your architecture e.g. learning rate, variable initializers, strides, zero padding etc.
3. You should create a train-validation split on the training set and train and test on appropriate data only. You must train properly with a proper stopping criterion. All decisions you take for model parameters must be logical.
4. Do NOT hardcode any paths.
5. Include all training and testing codes. Two files named ‘rnn.py’ and ‘cnn.py’ must be present. They will take some command line arguments. ‘python rnn.py --train’ should train the RNN model, ‘python rnn.py --test’ should load all model weights, test the model and report the accuracy on the test data. Similarly for the file ‘cnn.py’.
6. Name the ZIP file as well the folder that you compressed to get the zip as “Assignment10_<Roll No.>.zip” and “Assignment10_<Roll No.>” respectively. Upload the zip file to moodle. Note that the zip file should contain your python files, model weights (i.e. the ‘logs_rnn/’ and the ‘logs_cnn/’ folders), and a pdf file explaining your model hyperparameters and results/inferences. Please note that all explanations should be short (1-3 sentences).
7. Small differences in your method's accuracy would not be penalized (larger may be; remember to set a fixed random seed for reproducing your result). Your experimentation technique should be sound (like, do not test on training data or train on validation data).