Homework 2: Neural Networks with TensorFlow
Large-Scale Data Analysis 2020

Shahriyar Mahdi Robbani (XQR418)

May 13, 2020

1 Standard Neural Network

1.1 Adding a layer

```
tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, activation='sigmoid', input_shape=(1,), name=
    'hidden1'),
    tf.keras.layers.Dense(32, activation='sigmoid', name='hidden2'),
    tf.keras.layers.Dense(1, activation='linear', name='output')
])
```

1.2 Functional API

```
inputs = keras.Input(shape=(1,))
h1 = layers.Dense(64, activation='sigmoid')(inputs)
h2 = layers.Dense(32, activation='sigmoid')(h1)
outputs = layers.Dense(1, activation='linear')(h2)
model = keras.Model(inputs=inputs, outputs=outputs, name='functional_model'
    )
```

1.3 Shortcut connections

```
inputs = keras.Input(shape=(1,))
h1 = layers.Dense(64, activation='sigmoid')(inputs)
h2 = layers.Dense(32, activation='sigmoid')(h1)
h2_cat = layers.concatenate([h2, inputs])
outputs = layers.Dense(1, activation='linear')(h2_cat)
model = keras.Model(inputs=inputs, outputs=outputs, name='shortcut_model')
```

The resulting model has the following trainable parameters:

```
Model: "shortcut_model"
________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
input_1 (InputLayer)            [(None, 1)]          0
________________________________________________________________________________
dense (Dense)                   (None, 64)           128         input_1[0][0]
________________________________________________________________________________
dense_1 (Dense)                 (None, 32)           2080        dense[0][0]
________________________________________________________________________________
concatenate (Concatenate)       (None, 33)           0           dense_1[0][0]
                                                                 input_1[0][0]
________________________________________________________________________________
```

```
dense_2 (Dense)                (None, 1)            34          concatenate[0][0]
================================================================================
Total params: 2,242
Trainable params: 2,242
Non-trainable params: 0
```

2 Convolutional neural network for traffic sign recognition

2.1 Adding batch normalization

I added batch normalization after each convolutional layer, but before the corresponding activation layer, as mentioned in the paper. This resulted in a total of two batch normalizations. The model ran for 800 epochs and resulted in a test accuracy of 0.971.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
   img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
   initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.BatchNormalization(), #added
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
   tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.BatchNormalization(), #added
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

2.2 Trainable parameters

1. The 1st layer is the input layer which has 0 trainable parameters.
2. The 2nd layer is a 2D convolutional layer. Each neuron in the layer computes the output of the 2D cross-correlation function on each input and a 5×5 filter. The input to each neuron is a vector of shape $28 \times 28 \times 3$ (28 length, width and 3 channels). Thus, the total number of trainable parameters are the size of each filter (5×5) times the number of channels (3), plus a bias for every filter, multiplied by the number of filters (32) which results in $((5 \times 3) + 1) \times 32 = 2432$ trainable parameters.
3. The 3rd layer is a batch normalization layer. Each neuron in this layer normalizes the input before scaling and shifting it. The parameters are μ and σ^2, which are required to normalize the input and β and γ which shift and scale respectively. The input to this layer are the 32 feature maps produced by the 2nd layer. Thus, the total number of trainable parameters are $4 \times 32 = 128$.
4. The 4th layer is an ELU activation layer which simply performs the ELU operation so there are 0 trainable parameters.
5. The 5th layer is a pooling layer which simply reduces the dimensionality of the previous layer so there are 0 trainable parameters.
6. The 6th layer is another 2D convlutional layer. This layer also applies a 5×5 filter but has 64 filters instead. The input to this layer is a vector of shape $12 \times 12 \times 32$ (12 length, width and 32 channels). Thus the total number of trainable parameters are $((5 \times 5 \times 32) + 1) \times 64 = 51264$.
7. The 7th layer is another batch normalization layer. The input to this layer are the 64 feature maps from the previous layer, so the total number of trainable parameters are $64 \times 4 = 256$.

8. The 8th layer is another ELU layer so it has 0 trainable parameters.

9. The 9th layer is another pooling layer so it has 0 trainable parameters.

10. The 10th layer is a flatten layer which reshapes the $4 \times 4 \times 64$ input vector to a 1024×1 vector so it has 0 trainable parameters.

11. The 11th layer is a dense layer. Each neuron in the layer computes the function $W(x)+b$, where W is weight, x is an input and b is a bias. This layer has 43 outputs, corresponding to the number of classes. The input to this layer are the 1024 values from the previous layer. Thus the overall number of trainable parameters are inputs (1024) times outputs (43) plus weights (43) which results in $(1024 \times 43) + 43 = 44075$ trainable parameters.

2.3 Data augmentation

The image is randomly cropped, flipped from left to right and the brightness is randomly changed. The transformation that corresponds to flipping the image from left the the right depends on the label and is only applied to sign that do not have any text, numbers or directional arrows, because this transformation would result in flipped letters, numbers or arrows that will never actually occur on traffic signs so our model will be given incorrect training data and labels.

One transformation that can be added is to randomly change the contrast of the image:

```
image = tf.image.random_contrast(image, lower=0.5, upper=2.)
```

This change multiplies all values of the image with a random contrast factor between 0.5 and 2. This adds more variety in the input data without changing fundamental features of the image such as the numbers, text or direction of arrows,

3 Experimental architecture comparison

I created 6 models in order to test the effect of the dropout layer position. All models were run for 800 epochs and a dropout value of 0.5 was used. The following models were used:

The basic model where no changes were made:

```
default = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
   img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
   initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
   tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

A model where dropout was added after the first convolutional layer.

```
dropout_1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
   img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
   initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
   tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

A model where dropout was added after the first pooling layer.

```
dropout1_pool = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

A model where dropout was added after both convolutional layers.

```
dropout_2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

A model where dropout was added after both pooling layers.

```
dropout2_pool = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

A model where dropout was added after the flatten layer.

```
flatten = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

A model where dropout was added after each pooling layer and the flatten layer.

```
full = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

The following plot show the loss plotted against the epoch for all the models:

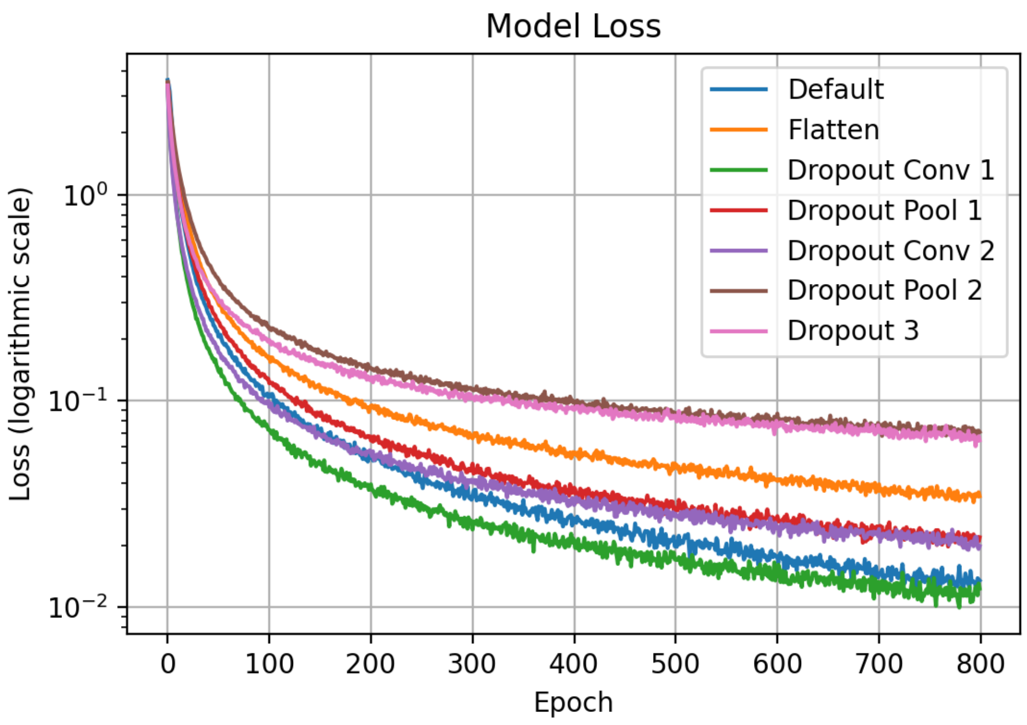

Figure 1: Loss for different dropout layers

The average test accuracy was recorded for the various layers:

```
Default
Average test accuracy: 0.9377
Flatten
Average test accuracy: 0.9726
Dropout 1
Average test accuracy: 0.9416
Dropout 1 pool
Average test accuracy: 0.9593
Dropout 2
Average test accuracy: 0.9397
```

```
Dropout 2 pool
Average test accuracy: 0.9772
Dropout 3
Average test accuracy: 0.9824
```

It can be seen that accuracy increased much more when the dropout layer was added after the pooling layer instead of before it. Increasing the number of dropout layers also increased accuracy, with the 3 dropout layers having the best accuracy. However, with the addition of more dropout layers, training slowed down, and the loss was much higher at 800 epochs. The overall shape of the loss curves were similar.

I then created 3 more models to test different dropout values (0.5, 0.2, 0.8), Each model had 3 dropout layers with the same value. These models are shown below:

```
full_05 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5) #added
    tf.keras.layers.Dense(no_classes, activation='softmax')])

full_02 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.2) #added
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.2) #added
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.2) #added
    tf.keras.layers.Dense(no_classes, activation='softmax')])

full_08 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation=None, input_shape=(
    img_width_crop, img_height_crop, no_channels), bias_initializer=tf.
    initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.8) #added
    tf.keras.layers.Conv2D(64, (5, 5), activation=None,  bias_initializer=
    tf.initializers.TruncatedNormal(mean=sd_init, stddev=sd_init)),
    tf.keras.layers.ELU(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(0.8) #added
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.8) #added
    tf.keras.layers.Dense(no_classes, activation='softmax')])
```

The following plot shows the loss plotted against the epochs for all 3 models:

The average test accuracy for the models are given below:

```
Dropout 0.5
```

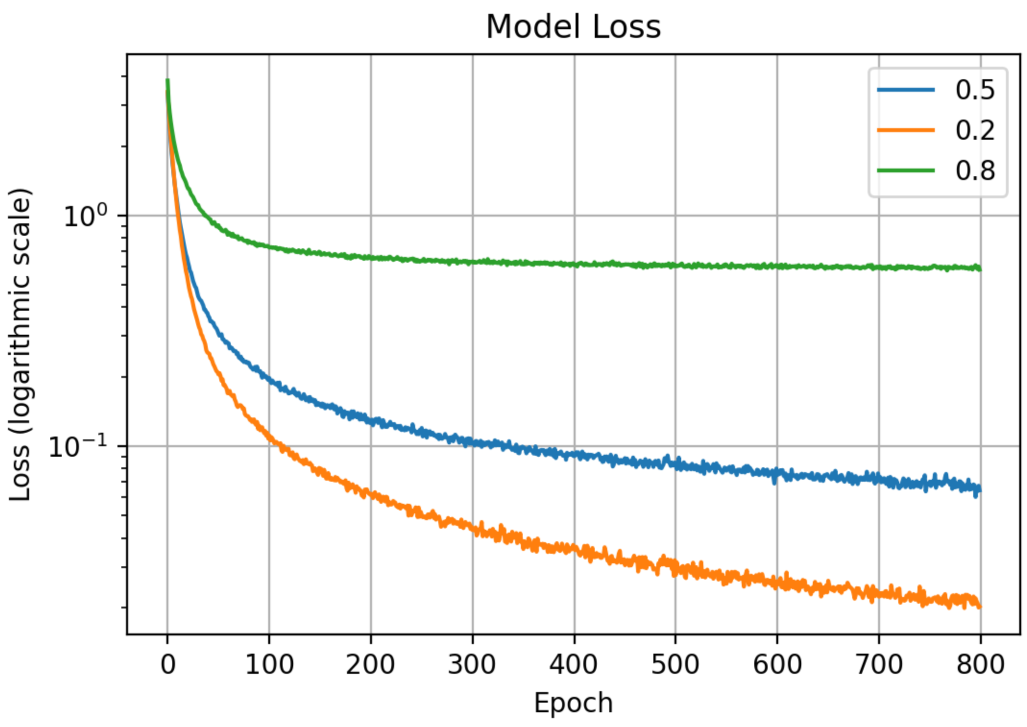

Figure 2: Loss for different dropout values

```
Average test accuracy: 0.9824
Dropout 0.2
Average test accuracy: 0.9651
Dropout 0.8
Average test accuracy: 0.8385
```

It can be seen that the training slows down with higher dropout values. With a dropout of 0.8 training is significantly slower and the model performs much worse at 800 epochs. While a dropout of 0.2 increases the training speed (a lower loss is obtained at 800 epochs), the test accuracy decreases.

From this basic analysis we can see that the addition of dropout can help increase test accuracy at the cost of requiring a longer training time. The best placement of the dropout layer is after pooling layers and a moderate dropout value, close to 0.5, seems to yield the best results for a given epoch constraint.

4 Keras training and testing mode

Models and layers in Keras have a Boolean flag that can put them in training and testing mode by specifying `training=True` and `training=False`, respectively.

4.1 Dropout

When dropout is being used to train a model, it randomly removes nodes in a layer, however when evaluating a model, we do not want this behavior to occur because this would result in random results. Therefore specifying `training=True` ensures dropout randomly removes nodes during the training phase, while `training=False` ensures no nodes are removed and all weights are modified instead during the evaluation phase.

4.2 Batch Normalization

When a model is trained, a batch of data is used. The mean and variance for this batch is calculated during the training process. When evaluating the model, only a single data point is used therefore no mean and variance is calculated for it. Thus, `training=True` is used to perform batch normalization on the training data and `training=False` is used to prevent batch normalization of the evaluation data.

5 Challenge (optional)

I did the following modification to the model and the training process:

The results are shown in . . .

Please find the code attached in the file . . .