digits_classification

December 14, 2020

1 MNIST digits classification with TensorFlow

```
In [35]: import numpy as np
         from sklearn.metrics import accuracy_score
         from matplotlib import pyplot as plt
         %matplotlib inline
         import tensorflow as tf
         print("We're using TF", tf.__version__)
```

```
We're using TF 1.2.1
```

```
In [36]: import sys
         sys.path.append("../..")
         import grading

         import matplotlib_utils
         from importlib import reload
         reload(matplotlib_utils)

         import grading_utils
         reload(grading_utils)

         import keras_utils
         from keras_utils import reset_tf_session
```

2 Fill in your Coursera token and email

To successfully submit your answers to our grader, please fill in your Coursera submission token and email

```
In [37]: grader = grading.Grader(assignment_key="XtD7ho3TEeiHQBLWejjYAA",
                                 all_parts=["9XaAS", "vmogZ", "RMv95", "i8bgs", "rE763"])
```

```
In [105]: # token expires every 30 min
          COURSERA_TOKEN = "PNwzO8jImtYeIdns"
          COURSERA_EMAIL = "knowtech94@gmail.com"
```

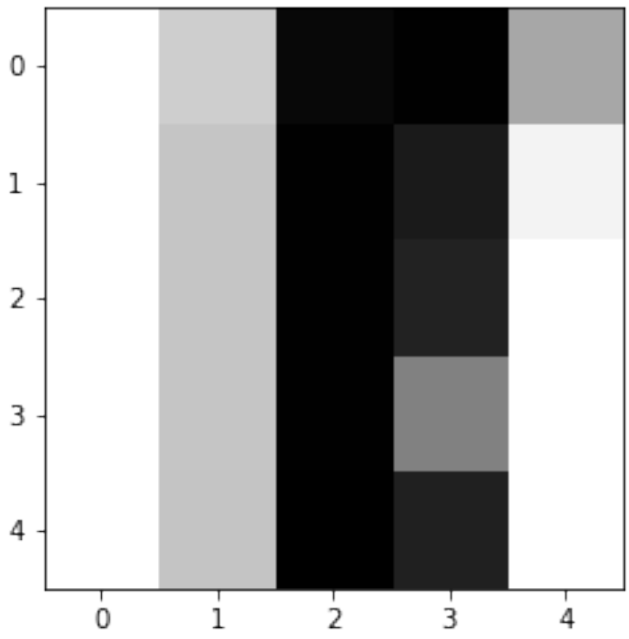
3 Look at the data

In this task we have 50000 28x28 images of digits from 0 to 9. We will train a classifier on this data.

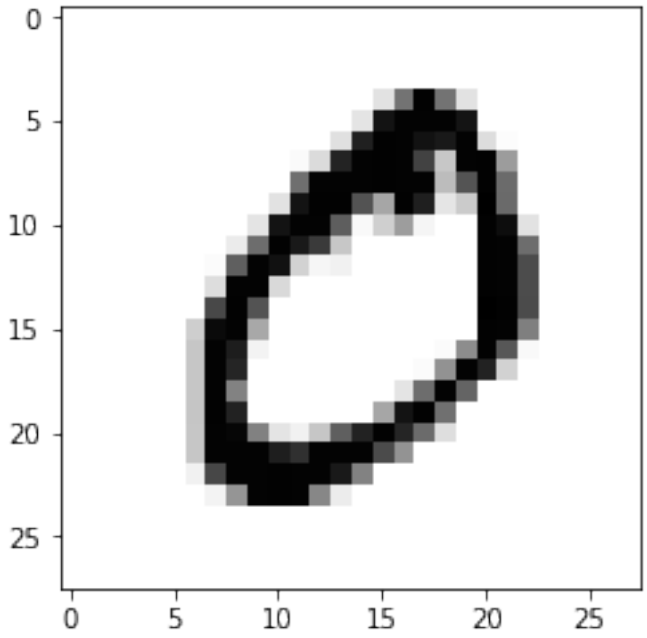
```
In [39]: import preprocessed_mnist
         X_train, y_train, X_val, y_val, X_test, y_test = preprocessed_mnist.load_dataset_from_f

In [40]: # X contains rgb values divided by 255
         print("X_train [shape %s] sample patch:\n" % (str(X_train.shape)), X_train[1, 15:20, 5:
         print("A closeup of a sample patch:")
         plt.imshow(X_train[1, 15:20, 5:10], cmap="Greys")
         plt.show()
         print("And the whole sample:")
         plt.imshow(X_train[1], cmap="Greys")
         plt.show()
         print("y_train [shape %s] 10 samples:\n" % (str(y_train.shape)), y_train[:10])
```

```
X_train [shape (50000, 28, 28)] sample patch:
 [[ 0.          0.29803922  0.96470588  0.98823529  0.43921569]
 [ 0.          0.33333333  0.98823529  0.90196078  0.09803922]
 [ 0.          0.33333333  0.98823529  0.8745098   0.        ]
 [ 0.          0.33333333  0.98823529  0.56862745  0.        ]
 [ 0.          0.3372549   0.99215686  0.88235294  0.        ]]
A closeup of a sample patch:
```



```
And the whole sample:
```




```
y_train [shape (50000,)] 10 samples:
 [5 0 4 1 9 2 1 3 1 4]
```

4 Linear model

Your task is to train a linear classifier $\vec{x} \to y$ with SGD using TensorFlow.

You will need to calculate a logit (a linear transformation) z_k for each class:

$$z_k = \vec{x} \cdot \vec{w}_k + b_k \quad k = 0..9$$

And transform logits z_k to valid probabilities p_k with softmax:

$$p_k = \frac{e^{z_k}}{\sum_{i=0}^{9} e^{z_i}} \quad k = 0..9$$

We will use a cross-entropy loss to train our multi-class classifier:

$$\text{cross-entropy}(y, p) = -\sum_{k=0}^{9} \log(p_k)[y = k]$$

where

$$[x] = \begin{cases} 1, & \text{if } x \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

Cross-entropy minimization pushes p_k close to 1 when $y = k$, which is what we want.

Here's the plan: * Flatten the images (28x28 -> 784) with `X_train.reshape((X_train.shape[0], -1))` to simplify our linear model implementation * Use a matrix placeholder for flattened `X_train` * Convert `y_train` to one-hot encoded vectors that are needed for cross-entropy * Use a shared variable `W` for all weights (a column $\vec{w}_k$ per class) and `b` for all biases. * Aim for ~0.93 validation accuracy

```
In [41]: X_train_flat = X_train.reshape((X_train.shape[0], -1))
         print(X_train_flat.shape)

         X_val_flat = X_val.reshape((X_val.shape[0], -1))
         print(X_val_flat.shape)
```

```
(50000, 784)
(10000, 784)
```

```
In [42]: import keras
         #one-hot encoder variables
         y_train_oh = keras.utils.to_categorical(y_train, 10)
         y_val_oh = keras.utils.to_categorical(y_val, 10)

         print(y_train_oh.shape)
         print(y_train_oh[:3], y_train[:3])
```

```
(50000, 10)
[[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]] [5 0 4]
```

```
In [88]: # run this again if you remake your graph
         s = reset_tf_session()
```

```
In [89]: # Model parameters: W and b
         #W = ### YOUR CODE HERE ### tf.get_variable(...) with shape[0] = 784
         #b = ### YOUR CODE HERE ### tf.get_variable(...)
         W = tf.get_variable('weights', shape = (784, 10), dtype = tf.float32)
         b = tf.get_variable('bias', shape = (10, ), dtype = tf.float32)
```

```
In [90]: # Placeholders for the input data
         #input_X = ### YOUR CODE HERE ### tf.placeholder(...) for flat X with shape[0] = None f
         #input_y = ### YOUR CODE HERE ### tf.placeholder(...) for one-hot encoded true labels
         input_X = tf.placeholder(tf.float32, shape = (None, 784))
         input_y = tf.placeholder(tf.float32, shape = (None, 10))
```

```
In [91]: # Compute predictions
         #logits = ### YOUR CODE HERE ### logits for input_X, resulting shape should be [input_X
         #probas = ### YOUR CODE HERE ### apply tf.nn.softmax to logits
```

```
#classes = ### YOUR CODE HERE ### apply tf.argmax to find a class index with highest pr
logits = tf.matmul(input_X, W) + b
probas = tf.nn.softmax(logits)
classes = tf.argmax(probas, axis = 1)

# Loss should be a scalar number: average loss over all the objects with tf.reduce_mean
# Use tf.nn.softmax_cross_entropy_with_logits on top of one-hot encoded input_y and log
# It is identical to calculating cross-entropy on top of probas, but is more numericall
#loss = ### YOUR CODE HERE ### cross-entropy loss
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = logits, labels =

# Use a default tf.train.AdamOptimizer to get an SGD step
#step = ### YOUR CODE HERE ### optimizer step that minimizes the loss
optimizer = tf.train.AdamOptimizer(0.003)
step = optimizer.minimize(loss)
```

In [92]:
```
s.run(tf.global_variables_initializer())

BATCH_SIZE = 512
EPOCHS = 40

# for logging the progress right here in Jupyter (for those who don't have TensorBoard)
simpleTrainingCurves = matplotlib_utils.SimpleTrainingCurves("cross-entropy", "accuracy

for epoch in range(EPOCHS):  # we finish an epoch when we've looked at all training sam

    batch_losses = []
    for batch_start in range(0, X_train_flat.shape[0], BATCH_SIZE):  # data is already
        _, batch_loss = s.run([step, loss], {input_X: X_train_flat[batch_start:batch_st
                                             input_y: y_train_oh[batch_start:batch_star
        # collect batch losses, this is almost free as we need a forward pass for backp
        batch_losses.append(batch_loss)

    train_loss = np.mean(batch_losses)
    val_loss = s.run(loss, {input_X: X_val_flat, input_y: y_val_oh})  # this part is us
    train_accuracy = accuracy_score(y_train, s.run(classes, {input_X: X_train_flat}))
    valid_accuracy = accuracy_score(y_val, s.run(classes, {input_X: X_val_flat}))
    simpleTrainingCurves.add(train_loss, val_loss, train_accuracy, valid_accuracy)
```

5 Submit a linear model

In [93]:
```
## GRADED PART, DO NOT CHANGE!
# Testing shapes
grader.set_answer("9XaAS", grading_utils.get_tensors_shapes_string([W, b, input_X, inpu
# Validation loss
grader.set_answer("vmogZ", s.run(loss, {input_X: X_val_flat, input_y: y_val_oh}))
# Validation accuracy
grader.set_answer("RMv95", accuracy_score(y_val, s.run(classes, {input_X: X_val_flat}))
```

In [94]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

6 MLP with hidden layers

Previously we've coded a dense layer with matrix multiplication by hand. But this is not convenient, you have to create a lot of variables and your code becomes a mess. In TensorFlow there's an easier way to make a dense layer:

```
hidden1 = tf.layers.dense(inputs, 256, activation=tf.nn.sigmoid)
```

That will create all the necessary variables automatically. Here you can also choose an activation function (remember that we need it for a hidden layer!).

Now define the MLP with 2 hidden layers and restart training with the cell above.

You're aiming for ~0.97 validation accuracy here.

In [103]:
```
# write the code here to get a new `step` operation and then run the cell with trainin
# name your variables in the same way (e.g. logits, probas, classes, etc) for safety.
### YOUR CODE HERE ###
#reset session
s = reset_tf_session()
#define variables
input_X = tf.placeholder(tf.float32, shape = (None, 784))
input_y = tf.placeholder(tf.float32, shape = (None, 10))
#make architecture of mlp
hidden1 = tf.layers.dense(inputs = input_X, units = 256, activation = tf.nn.sigmoid)
logits = tf.layers.dense(inputs = hidden1, units = 10)
#compute high probabilities
probas = tf.nn.softmax(logits)
classes = tf.argmax(probas, axis = 1)
#loss function
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = logits, labels
#optimizer
optimizer = tf.train.AdamOptimizer(0.001)
step = optimizer.minimize(loss)
```

In [104]:
```
#define the loop to train
s.run(tf.global_variables_initializer())

BATCH_SIZE = 512
EPOCHS = 40

# for logging the progress right here in Jupyter (for those who don't have TensorBoard
simpleTrainingCurves = matplotlib_utils.SimpleTrainingCurves("cross-entropy", "accurac
```

```
for epoch in range(EPOCHS):  # we finish an epoch when we've looked at all training sa

    batch_losses = []
    for batch_start in range(0, X_train_flat.shape[0], BATCH_SIZE):  # data is already
        _, batch_loss = s.run([step, loss], {input_X: X_train_flat[batch_start:batch_s
                                             input_y: y_train_oh[batch_start:batch_sta
        # collect batch losses, this is almost free as we need a forward pass for back
        batch_losses.append(batch_loss)

    train_loss = np.mean(batch_losses)
    val_loss = s.run(loss, {input_X: X_val_flat, input_y: y_val_oh})  # this part is u
    train_accuracy = accuracy_score(y_train, s.run(classes, {input_X: X_train_flat}))
    valid_accuracy = accuracy_score(y_val, s.run(classes, {input_X: X_val_flat}))
    simpleTrainingCurves.add(train_loss, val_loss, train_accuracy, valid_accuracy)
```

7 Submit the MLP with 2 hidden layers

Run these cells after training the MLP with 2 hidden layers

In [106]:
```
## GRADED PART, DO NOT CHANGE!
# Validation loss for MLP
grader.set_answer("i8bgs", s.run(loss, {input_X: X_val_flat, input_y: y_val_oh}))
# Validation accuracy for MLP
grader.set_answer("rE763", accuracy_score(y_val, s.run(classes, {input_X: X_val_flat}))
```

In [107]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

In [ ]: