Solution

March 12, 2023

1 imports

[ ]:
```python
import numpy as np
from conv import Conv3x3
from maxpool import MaxPool2
from softmax import Softmax

from matplotlib import pyplot as plt

# load mnist handwritten dataset from tensorflow
import tensorflow as tf
mnist = tf.keras.datasets.mnist
```

```
2023-03-12 23:14:11.232703: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

[ ]:
```python
# split the data into train and test
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images[:1000]
train_labels = train_labels[:1000]
test_images = test_images[:1000]
test_labels = test_labels[:1000]
```

[ ]:
```python
# Normalize the images.
train_images = train_images / 255.0
test_images = test_images / 255.0
```

[ ]:
```python
# making a model

conv = Conv3x3(num_filters=9)                    # 9 filters, of size 3x3 and␣
 ↪stride 1 without padding
pool = MaxPool2()                   # Pooling layer with size 2x2 and stride 2
```

```python
softmax = Softmax(input_len=13 * 13 * 9, nodes=10)  # Softmax layer with 10␣
↪classes
```

[ ]:
```python
# forward pass
def forward(image, label):
    '''
    Completes a forward pass of the CNN and calculates the accuracy and
    cross-entropy loss.
    - image is a 2d numpy array
    - label is a digit
    '''
    # Transform the grayscale image from [0, 255] to [-0.5, 0.5] to make it easier
    # to work with.
    out = conv.forward(image )
    out = pool.forward( out )
    out = softmax.forward( out )

    # Compute cross-entropy loss and accuracy.
    loss =  -np.log(out[label])
    acc =  1 if np.argmax(out) == label else 0

    return out, loss, acc
```

[ ]:
```python
def train(im, label, lr=.001):
    '''
    A training step on the given image and label.
    Shall return the cross-entropy loss and accuracy.
    - image is a 2d numpy array
    - label is a digit
    - lr is the learning rate
    '''
    # Forward
    out, loss, acc = forward(im, label)

    # Calculate initial gradient
    gradient = np.zeros(10)
    gradient[label] = -1 / out[label]

    # Backprop
    gradient = softmax.backprop(gradient, lr)
    gradient = pool.backprop(gradient)
    gradient = conv.backprop(gradient, lr)

    return loss, acc
```

[ ]:
```python
# training the model
for epoch in range(3):
```

```python
  print('--- Epoch %d ---' % (epoch + 1))

  # Shuffle the training data
  permutation = np.random.permutation(len(train_images))
  train_images = train_images[permutation]
  train_labels = train_labels[permutation]

  # Train!
  loss = 0
  num_correct = 0

  for i, (im, label) in enumerate(zip(train_images, train_labels)):
    if i % 100 == 0:
      print(
        '[Step %d] Past 100 steps: Average Loss %.3f | Accuracy: %d%%' %
        (i, loss / 100, num_correct)
      )
      loss = 0
      num_correct = 0

    l, acc = train(im, label)
    loss += l
    num_correct  += acc
```

```
--- Epoch 1 ---
[Step 0] Past 100 steps: Average Loss 0.000 | Accuracy: 0%
[Step 100] Past 100 steps: Average Loss 2.294 | Accuracy: 25%
[Step 200] Past 100 steps: Average Loss 2.270 | Accuracy: 33%
[Step 300] Past 100 steps: Average Loss 2.256 | Accuracy: 38%
[Step 400] Past 100 steps: Average Loss 2.235 | Accuracy: 38%
[Step 500] Past 100 steps: Average Loss 2.208 | Accuracy: 49%
[Step 600] Past 100 steps: Average Loss 2.178 | Accuracy: 60%
[Step 700] Past 100 steps: Average Loss 2.141 | Accuracy: 67%
[Step 800] Past 100 steps: Average Loss 2.110 | Accuracy: 66%
[Step 900] Past 100 steps: Average Loss 2.055 | Accuracy: 64%
--- Epoch 2 ---
[Step 0] Past 100 steps: Average Loss 0.000 | Accuracy: 0%
[Step 100] Past 100 steps: Average Loss 1.931 | Accuracy: 64%
[Step 200] Past 100 steps: Average Loss 1.790 | Accuracy: 72%
[Step 300] Past 100 steps: Average Loss 1.702 | Accuracy: 63%
[Step 400] Past 100 steps: Average Loss 1.517 | Accuracy: 72%
[Step 500] Past 100 steps: Average Loss 1.484 | Accuracy: 72%
[Step 600] Past 100 steps: Average Loss 1.358 | Accuracy: 79%
[Step 700] Past 100 steps: Average Loss 1.284 | Accuracy: 79%
[Step 800] Past 100 steps: Average Loss 1.181 | Accuracy: 78%
[Step 900] Past 100 steps: Average Loss 1.067 | Accuracy: 79%
--- Epoch 3 ---
```

```
[Step 0] Past 100 steps: Average Loss 0.000 | Accuracy: 0%
[Step 100] Past 100 steps: Average Loss 0.885 | Accuracy: 86%
[Step 200] Past 100 steps: Average Loss 0.975 | Accuracy: 74%
[Step 300] Past 100 steps: Average Loss 0.828 | Accuracy: 84%
[Step 400] Past 100 steps: Average Loss 0.802 | Accuracy: 82%
[Step 500] Past 100 steps: Average Loss 0.771 | Accuracy: 80%
[Step 600] Past 100 steps: Average Loss 0.670 | Accuracy: 83%
[Step 700] Past 100 steps: Average Loss 0.744 | Accuracy: 79%
[Step 800] Past 100 steps: Average Loss 0.679 | Accuracy: 86%
[Step 900] Past 100 steps: Average Loss 0.501 | Accuracy: 89%
```

1.0.1 Training Set Accuracy

[ ]:
```python
loss = 0
num_correct = 0
for im, label in zip(train_images, train_labels):
  _, l, acc = forward(im, label)
  loss += l
  num_correct += acc

num_trains = len(train_images)
print('Train Accuracy:', num_correct / num_trains)
```

```
Train Accuracy: 0.851
```

1.0.2 Test Set Accuracy

[ ]:
```python
loss = 0
num_correct = 0
for im, label in zip(test_images, test_labels):
  _, l, acc = forward(im, label)
  loss += l
  num_correct += acc

num_tests = len(test_images)
print('Test Accuracy:', num_correct / num_tests)
```

```
Test Accuracy: 0.77
```

2 I have also written a Full Neural Netwrok Module from Scratch.

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from  nn.nn import NeuralNetwork as network
from  nn import layers as layers
```

```
from  nn import losses as losses
from nn import activations as activations
from nn.metrics import Metrics as metrics
from nn import optimizers as optimizers
```

[ ]:
```
# Changing the dimesions of the input
x_train = train_images.reshape(-1,1,28,28)
x_test = test_images.reshape(-1,1,28,28)

# one hot encoding the y labels
y_train = pd.get_dummies(train_labels).values
y_test = pd.get_dummies(test_labels).values
```

[ ]:
```
digits = network()

digits.add(layers.ConvLayer(input_shape=(28,28,1), filters=9,␣
 ↪kernel_size=(3,3), padding='valid', use_bias=True, strides=1))
digits.add(layers.ActivationLayer(activations.Tanh))
digits.add(layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))


digits.add(layers.FlattenLayer())
digits.add(layers.DenseLayer( units=10, activation="softmax"))

digits.compile(loss=losses.CategoricalCrossentropy(), optimizer=optimizers.
 ↪Adam(lr=0.01), initializer="glorot_uniform", metrics=["accuracy",␣
 ↪"precision"])

digits.summary()
digits.fit(x_train, y_train, epochs=20, batch_size=32, verbose=3)
```

```
Summary of the Neural Network
--------------------------------------------------------------------------------
-----------------------------------
Layer (type)        Neurons #      Input Shape    Output Shape   Weights shape
Bias shape        Param #
================================================================================
===================================
Input               3              -              (3, None)      -
-                      0

ConvLayer           9              (28, 28, 1)    ((26, 26, 9),) (3, 3, 1, 9)
(9, 1)                90

ActivationLayer     -              activavtion    Tanh           -
-                      0
```

```
MaxPool2D          -              (26, 26, 9)    ((13, 13, 9),) -
-                      0

FlattenLayer       -              -              -              -
-                      0

DenseLayer         10             (1521,)        (10,)          (10, 1521)
(10, 1)            15220

================================================================================
===================================
Total params
15310
Epoch 1-20 Batch 1-31 ======================> cost: 2.2395 accuracy: 0.1740
precision: 0.1375
Epoch 2-20 Batch 1-31 ======================> cost: 0.9249 accuracy: 0.7170
precision: 0.7192
Epoch 3-20 Batch 1-31 ======================> cost: 0.7180 accuracy: 0.7830
precision: 0.7845
Epoch 4-20 Batch 1-31 ======================> cost: 0.5622 accuracy: 0.8490
precision: 0.8468
Epoch 5-20 Batch 1-31 ======================> cost: 0.5028 accuracy: 0.8340
precision: 0.8389
Epoch 6-20 Batch 1-31 ======================> cost: 0.4146 accuracy: 0.8820
precision: 0.8796
Epoch 7-20 Batch 1-31 ======================> cost: 0.3885 accuracy: 0.8850
precision: 0.8855
Epoch 8-20 Batch 1-31 ======================> cost: 0.3201 accuracy: 0.9210
precision: 0.9218
Epoch 9-20 Batch 1-31 ======================> cost: 0.3131 accuracy: 0.9080
precision: 0.9158
Epoch 10-20 Batch 1-31 ======================> cost: 0.2799 accuracy: 0.9290
precision: 0.9298
Epoch 11-20 Batch 1-31 ======================> cost: 0.2586 accuracy: 0.9310
precision: 0.9289
Epoch 12-20 Batch 1-31 ======================> cost: 0.2379 accuracy: 0.9360
precision: 0.9378
Epoch 13-20 Batch 1-31 ======================> cost: 0.2383 accuracy: 0.9350
precision: 0.9355
Epoch 14-20 Batch 1-31 ======================> cost: 0.2162 accuracy: 0.9510
precision: 0.9517
Epoch 15-20 Batch 1-31 ======================> cost: 0.2128 accuracy: 0.9460
precision: 0.9466
Epoch 16-20 Batch 1-31 ======================> cost: 0.1783 accuracy: 0.9590
precision: 0.9600
Epoch 17-20 Batch 1-31 ======================> cost: 0.1619 accuracy: 0.9620
precision: 0.9636
Epoch 18-20 Batch 1-31 ======================> cost: 0.1697 accuracy: 0.9630
```

```
precision: 0.9622
Epoch 19-20 Batch 1-31 ======================> cost: 0.1565 accuracy: 0.9610
precision: 0.9622
Epoch 20-20 Batch 1-31 ======================> cost: 0.1272 accuracy: 0.9770
precision: 0.9768
```

2.0.1 Train Set Accuracy

[ ]:
```python
pred_train_preb = digits.predict(x_train)

# get the index of the max value in each column
pred_train = np.argmax(pred_train_preb, axis=1)
y_train_ = np.argmax(y_train, axis=1)
print("Train Accuracy: ", metrics.accuracy(y_train_, pred_train))
```

```
Train Accuracy:  0.977
```

2.0.2 Test Set Accuracy

[ ]:
```python
pred_test_preb = digits.predict(x_test)

# get the index of the max value in each column
pred_test = np.argmax(pred_test_preb, axis=1)
y_test_ = np.argmax(y_test, axis=1)
print("Test Accuracy: ", metrics.accuracy(y_test_, pred_test))
```

```
Test Accuracy:  0.766
```