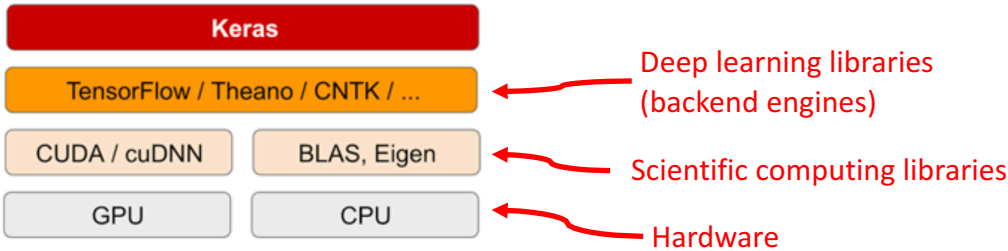
Train a Neural Networks using Keras

Shusen Wang

Implement a Softmax Classifier Using Keras

Deep Learning Systems



Softmax Classifier

The MNIST Dataset

- $n = 60{,}000$ training samples $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_j$ is a 28×28 image.
- Each y_j is an integer in $\{0, 1, 2, \cdots, 9\}$.

Softmax Classifier

The MNIST Dataset

- $n = 60{,}000$ training samples $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_j$ is a 28×28 image.
- Each y_j is an integer in $\{0, 1, 2, \cdots, 9\}$.

Softmax classifier ($\mathbf{f}: \mathbb{R}^{784} \mapsto \mathbb{R}^{10}$) for MNIST:

- **Input**: vector $\mathbf{x} \in \mathbb{R}^{784}$.
- $\mathbf{z} = \mathbf{W}\,\mathbf{x} + \mathbf{b} \in \mathbb{R}^{10}$.
- **Output**: $\mathbf{f}(\mathbf{x}) = \mathrm{SoftMax}(\mathbf{z})$.

Trainable parameters:

- $\mathbf{W} \in \mathbb{R}^{10 \times 784}$
- $\mathbf{b} \in \mathbb{R}^{10}$

1. Load and Process the MNIST Dataset

Load data

```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

print('Shape of x_train: ' + str(x_train.shape))
print('Shape of x_test: ' + str(x_test.shape))
print('Shape of y_train: ' + str(y_train.shape))
print('Shape of y_test: ' + str(y_test.shape))
```

```
Shape of x_train: (60000, 28, 28)
Shape of x_test: (10000, 28, 28)
Shape of y_train: (60000,)
Shape of y_test: (10000,)
```

1. Load and Process the MNIST Dataset

Vectorization: convert the 28×28 images to 784-dim vectors.

```
x_train_vec = x_train.reshape(60000, 784)
x_test_vec = x_test.reshape(10000, 784)

print('Shape of x_train_vec is ' + str(x_train_vec.shape))
```

```
Shape of x_train_vec is (60000, 784)
```

1. Load and Process the MNIST Dataset

Vectorization: convert the labels (an integer in $\{0, 1, \cdots, 9\}$) to 10-dim vectors (by the one-hot encode).

```
def to_one_hot(labels, dimension=10):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results

y_train_vec = to_one_hot(y_train)
y_test_vec = to_one_hot(y_test)
print('Shape of y_train_vec is ' + str(y_train_vec.shape))
```

```
Shape of y_train_vec is (60000, 10)
```

1. Load and Process the MNIST Dataset

Partition the **training** set to **training** and **validation** sets.

```
rand_indices = np.random.permutation(60000)
train_indices = rand_indices[0:50000]
valid_indices = rand_indices[50000:60000]

x_valid_vec = x_train_vec[valid_indices, :]
y_valid_vec = y_train_vec[valid_indices, :]

x_train_vec = x_train_vec[train_indices, :]
y_train_vec = y_train_vec[train_indices, :]

print('Shape of x_valid_vec: ' + str(x_valid_vec.shape))
print('Shape of y_valid_vec: ' + str(y_valid_vec.shape))
print('Shape of x_train_vec: ' + str(x_train_vec.shape))
print('Shape of y_train_vec: ' + str(y_train_vec.shape))
```

```
Shape of x_valid_vec: (10000, 784)
Shape of y_valid_vec: (10000, 10)
Shape of x_train_vec: (50000, 784)
Shape of y_train_vec: (50000, 10)
```

2. Build the Softmax Classifier

$$\mathbf{f}(\mathbf{x}) = \text{SoftMax}(\mathbf{W}\,\mathbf{x} + \mathbf{b})$$

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(10, activation='softmax', input_shape=(784,)))

# print the summary of the model.
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

3. Train the Softmax Classifier

Specify: optimization algorithm, learning rate (LR), loss function, and metric.

```
from keras import optimizers
model.compile(optimizers.RMSprop(lr=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

3. Train the Softmax Classifier

Specify: batch size and number of epochs.

```
history = model.fit(x_train_vec, y_train_vec,
                    batch_size=128, epochs=50,
                    validation_data=(x_valid_vec, y_valid_vec))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 1s 19us/step - loss: 11.2824 - acc: 0.2816 - val_loss: 8.7464 - val_acc: 0.4398
Epoch 2/50
50000/50000 [==============================] - 1s 13us/step - loss: 7.1525 - acc: 0.5384 - val_loss: 5.7830 - val_acc: 0.6221
Epoch 3/50
50000/50000 [==============================] - 1s 13us/step - loss: 5.1013 - acc: 0.6661 - val_loss: 4.5210 - val_acc: 0.7036
Epoch 4/50
50000/50000 [==============================] - 1s 13us/step - loss: 4.0385 - acc: 0.7339 - val_loss: 3.7206 - val_acc: 0.7533
```

-
-
-

```
Epoch 49/50
50000/50000 [==============================] - 1s 13us/step - loss: 1.1430 - acc: 0.9228 - val_loss: 1.3567 - val_acc: 0.9088
Epoch 50/50
50000/50000 [==============================] - 1s 14us/step - loss: 1.1407 - acc: 0.9230 - val_loss: 1.3526 - val_acc: 0.9077
```

4. Examine the Results

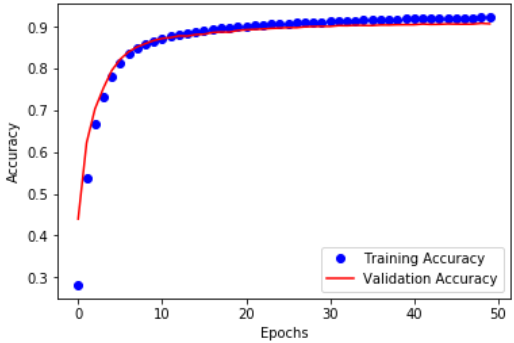
Plot the ***accuracy*** against ***epochs*** (1 epoch = 1 pass over the data).

```
import matplotlib.pyplot as plt
%matplotlib inline

epochs = range(50) # 50 is the number of epochs
train_acc = history.history['acc']
valid_acc = history.history['val_acc']
plt.plot(epochs, train_acc, 'bo', label='Training Accuracy')
plt.plot(epochs, valid_acc, 'r', label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

4. Examine the Results

Plot the ***accuracy*** against ***epochs*** (1 epoch = 1 pass over the data).



Why using the validation set?

- Tune the hyper-parameters, e.g., learning rate, regularization.
- Check whether overfit or underfit.

4. Examine the Results

Evaluate the model on the ***test*** set. (So far, the model has not seen the ***test*** set.)

```
loss_and_acc = model.evaluate(x_test_vec, y_test_vec)
print('loss = ' + str(loss_and_acc[0]))
print('accuracy = ' + str(loss_and_acc[1]))
```

```
10000/10000 [==============================] - 0s 20us/step
loss = 1.2781493856459185
accuracy = 0.9131
```

4. Examine the Results

- Training accuracy: 92.3%
- Validation accuracy: 90.8%
- Test accuracy: 91.3%

Not bad! A random guess has only 10% accuracy.

Implement a Neural Network Using Keras

Full-Connected Neural Network

- **Input**: vector $\mathbf{x}^{(0)} \in \mathbb{R}^{784}$.

- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)}\mathbf{x}^{(0)} + \mathbf{b}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.

Hidden Layer 1

- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)}\mathbf{x}^{(1)} + \mathbf{b}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.

Hidden Layer 2

- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)}\mathbf{x}^{(2)} + \mathbf{b}^{(2)} \in \mathbb{R}^{10}$.
- **Output**: $\mathrm{SoftMax}(\mathbf{z}^{(3)})$.

Output Layer

Trainable parameters:

- $\mathbf{W}^{(0)} \in \mathbb{R}^{d_1 \times 784}$,
- $\mathbf{W}^{(1)} \in \mathbb{R}^{d_2 \times d_1}$,
- $\mathbf{W}^{(2)} \in \mathbb{R}^{10 \times d_2}$,
- $\mathbf{b}^{(0)} \in \mathbb{R}^{d_1}$,
- $\mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$,
- $\mathbf{b}^{(2)} \in \mathbb{R}^{10}$.

2. Build the Softmax Classifier

```
from keras import models
from keras import layers

d1 = 500 # width of the 1st hidden layer
d2 = 500 # width of the 2nd hidden layer

model = models.Sequential()
model.add(layers.Dense(d1, activation='relu', input_shape=(784,)))
model.add(layers.Dense(d2, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# print the summary of the model.
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 500)               392500
_________________________________________________________________
dense_5 (Dense)              (None, 500)               250500
_________________________________________________________________
dense_6 (Dense)              (None, 10)                5010
=================================================================
Total params: 648,010
Trainable params: 648,010
Non-trainable params: 0
_________________________________________________________________
```

3. Train the Softmax Classifier

Specify: optimization algorithm, learning rate (LR), loss function, and metric.

```
from keras import optimizers
model.compile(optimizers.RMSprop(lr=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

3. Train the Softmax Classifier

Specify: batch size and number of epochs.

```
history = model.fit(x_train_vec, y_train_vec,
                    batch_size=128, epochs=50,
                    validation_data=(x_valid_vec, y_valid_vec))
```

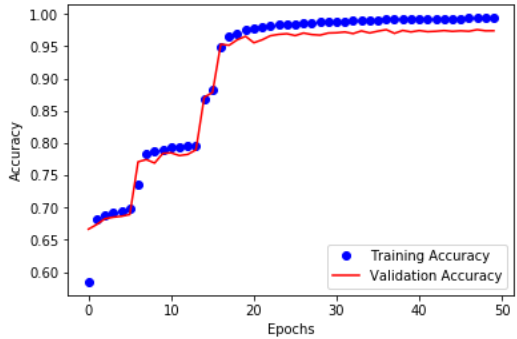
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/50
50000/50000 [==============================] - 5s 92us/step - loss: 6.5323 - acc: 0.5850 - val_loss: 5.2988 - val_acc: 0.6666
Epoch 2/50
50000/50000 [==============================] - 4s 84us/step - loss: 5.0559 - acc: 0.6817 - val_loss: 5.1904 - val_acc: 0.6738
Epoch 3/50
50000/50000 [==============================] - 3s 64us/step - loss: 4.9762 - acc: 0.6881 - val_loss: 5.0556 - val_acc: 0.6828
```

-
-
-

```
Epoch 49/50
50000/50000 [==============================] - 5s 93us/step - loss: 0.0912 - acc: 0.9934 - val_loss: 0.3404 - val_acc: 0.9738
Epoch 50/50
50000/50000 [==============================] - 4s 79us/step - loss: 0.0920 - acc: 0.9934 - val_loss: 0.3327 - val_acc: 0.9739
```

4. Examine the Results

Plot the ***accuracy*** against ***epochs*** (1 epoch = 1 pass over the data).



4. Examine the Results

Evaluate the model on the ***test*** set. (So far, the model has not seen the ***test*** set.)

```
loss_and_acc = model.evaluate(x_test_vec, y_test_vec)
print('loss = ' + str(loss_and_acc[0]))
print('accuracy = ' + str(loss_and_acc[1]))
```

```
10000/10000 [==============================] - 0s 43us/step
loss = 0.3648844491034643
accuracy = 0.9724
```

4. Examine the Results

- Training accuracy: 99.3%
- Validation accuracy: 97.4%
- Test accuracy: 97.2%

Much better than the simple softmax classifier (91.3% accuracy).

Summary