

Training Neural Networks from scratch

February 27, 2024

- Full Name : FELLAH Abdelnour.
- Group : 01.

1 Necessary Packages

```
[1]: import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import os
from tqdm.notebook import trange
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
```

2 Utilities

```
[2]: def get_dataset(m, n):
    X = np.random.normal(loc=0, scale=10, size=(m, n))
    Y = np.sum(X, axis=1)
    Y = 0 * (Y < -10) + 1 * ((Y >= -10) & (Y <= 10)) + 2 * (Y >= 10)
    return X, Y
```

```
[3]: def one_hot(a):
    b = np.zeros((a.size, a.max() + 1))
    b[np.arange(a.size), a] = 1
    return b
```

3 Implementation

3.1 Initialization

```
[4]: def init_params(nx, nh, ny):
    W1 = np.random.normal(loc=0, scale=0.3, size=(nh, nx+1))
    W2 = np.random.normal(loc=0, scale=0.3, size=(ny, nh+1))
    return [W1, W2]
```

3.2 Forward propagation

```
[5]: def tanh(z):  
    exp_z = np.exp(z)  
    exp__z = np.exp(-2 * z)  
    return (exp_z - exp__z) / (exp_z + exp__z)  
  
[6]: def sigmoid(z):  
    return 1 / (1 + np.exp(-z))  
  
[7]: def softmax(z):  
    t = np.exp(z - z.max(axis=0, keepdims=True)) # for numerical stability  
    return t / np.sum(t, axis=0, keepdims=True)  
  
[8]: def forward(params, X, activations):  
  
    Y = X.T  
    outputs = []  
  
    for W, activation in zip(params, activations):  
        Y = np.vstack([np.ones(Y.shape[1]), Y])  
        Z = W @ Y  
        Y = activation(Z)  
        outputs.append([Z,Y])  
  
    return outputs
```

3.3 Loss & Accuracy

```
[9]: def loss_accuracy(y_hat, y):  
    loss = -np.mean(np.log(np.sum(y_hat * y, axis=1)))  
    accuracy = np.mean(np.argmax(y_hat, axis=1) == np.argmax(y, axis=1))  
    return loss, accuracy
```

3.4 Backward propagation

```
[10]: def backward(X, params, outputs, Y):  
  
    # add a new line to all the outputs  
    outputs[-2][1] = np.vstack([np.ones(outputs[-2][1].  
→shape[1]), outputs[-2][1]]) # dA  
    X = np.vstack([np.ones(X.shape[0]), X.T])  
  
    # compute the gradients  
    gradients = {}  
  
    gradients["dZ2"] = outputs[-1][1] - Y.T # (ny,m) - (ny,m) = (ny,m)
```

```

    gradients["dW2"] = gradients["dZ2"] @ outputs[-2][1].T # (ny,m) * (m,nh+1) =
    ↪ (ny,nh+1)

    t = tanh(outputs[-2][0]) # (nh,m)
    gradients["dZ1"] = (params[-1].T[1:] @ gradients["dZ2"]) * (1 - t ** 2) #
    ↪ (nh,ny) @ (ny,m) * (nh,m) = (nh,m)

    # print(gradients["dZ1"].shape,X.shape)
    gradients["dW1"] = gradients["dZ1"] @ X.T # (nh,m) @ (m,nx+1) = (nh,nx+1)

    return gradients

```

3.5 Gradient Descent

```

[11]: def sgd(params, grads, eta):
    params[0] = params[0] - eta * grads["dW1"]
    params[1] = params[1] - eta * grads["dW2"]

```

3.6 predict

```

[12]: def predict(params, X):
    outputs = forward(params, X, [tanh,softmax])
    y_hat = outputs[-1][-1]
    y_hat = np.argmax(y_hat, axis=0)
    return y_hat

```

3.7 Train

```

[13]: def train(X, Y, test_set=None, eta=0.01, epochs=50, batch_size=128, nh=32):

    m,n = X.shape

    ny = len(np.unique(Y))

    Y = one_hot(Y)

    if test_set is not None:
        test_set = (test_set[0], one_hot(test_set[1]))

    params = init_params(n,nh,ny)

    history = {
        "accuracy": [],
        "loss": [],
        "test loss": [],
        "test accuracy": []
    }

```

```

for j in range(epochs):

    # randomize the data
    idx = np.arange(m)
    np.random.shuffle(idx)
    X = X[idx]
    Y = Y[idx]

    # calculate the number of batches
    batches_count = int(np.floor(m / batch_size))

    t = trange(batches_count, desc='Bar desc', leave=True)

    for i in t:

        X_batch = X[i * batch_size:(i+1) * batch_size,:]
        Y_batch = Y[i * batch_size:(i+1) * batch_size,:]

        outputs = forward(params, X_batch, [tanh, softmax])
        grads = backward(X_batch, params, outputs, Y_batch)

        sgd(params, grads, eta=eta)

        if i % 50 == 0:
            Y_hat = outputs[-1][1].T
            loss, accuracy = loss_accuracy(Y_hat, Y_batch)

            msg = f"epoch = {j+1} | loss = {loss:.6f} | accuracy = {100 * _
↪accuracy:.2f}%"
            test_loss, test_accuracy = None, None

            if test_set is not None:
                X_test, y_test = test_set
                y_test_hat = forward(params, X_test, [tanh, softmax])[-1][1].
↪T
                test_loss, test_accuracy = loss_accuracy(y_test, y_test_hat)
                msg += f" | test loss = {test_loss:.6f} | test accuracy = _
↪{100 * test_accuracy:.2f}%"

            if i % 50 == 0:
                t.set_description(msg)
                t.refresh()

            history["loss"].append(loss)
            history["accuracy"].append(accuracy)

```

```

        if test_set is not None:
            history["test loss"].append(test_loss)
            history["test accuracy"].append(test_accuracy)

    return params, history

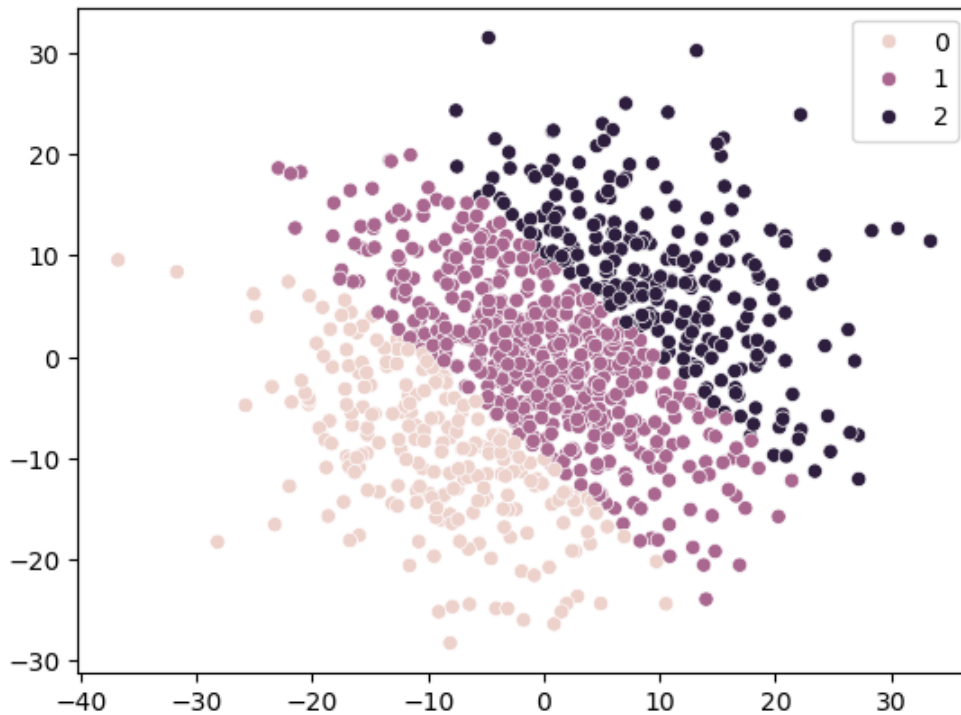
```

4 Test on a random dataset

```
[14]: X, Y = get_dataset(1000, 2)
```

```
[15]: sns.scatterplot(x=X[:,0],y=X[:,1],hue=Y)
```

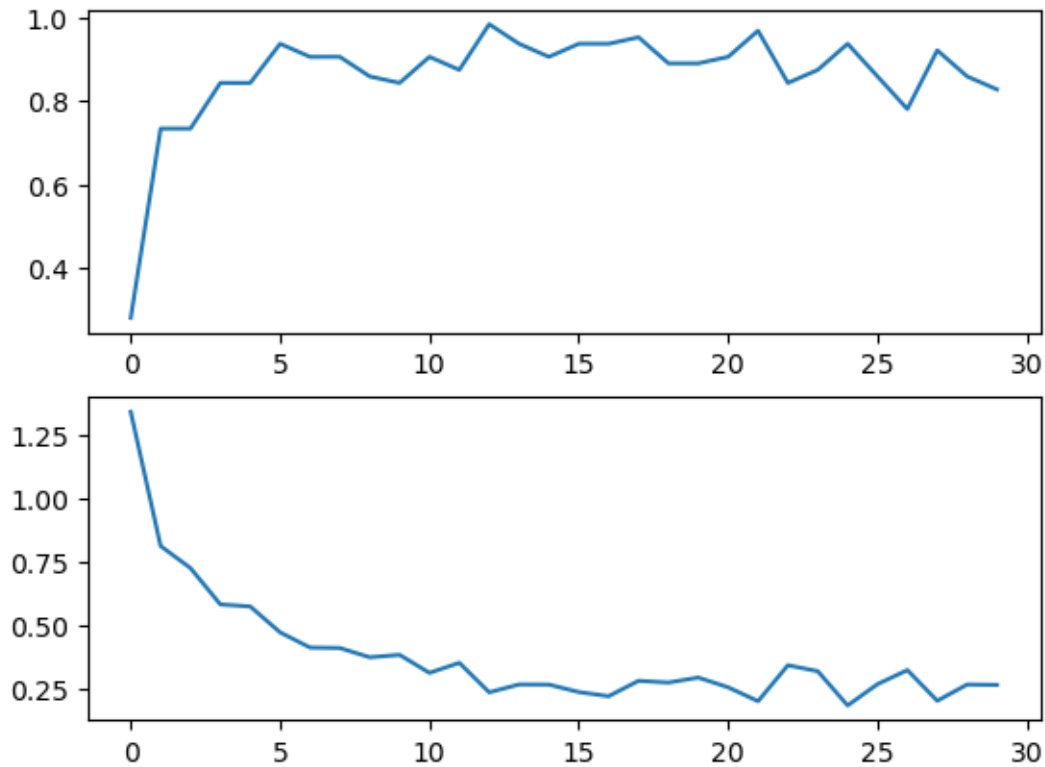
```
[15]: <Axes: >
```



```
[16]: params, history = train(X, Y, eta=10e-4, epochs=30, batch_size=64, nh=8)
```

```
[17]: fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1)
ax1.plot(np.arange(len(history['accuracy'])), history['accuracy'])
ax2.plot(np.arange(len(history['loss'])), history['loss'])
```

```
[17]: [<matplotlib.lines.Line2D at 0x7f131dbf3990>]
```



5 Train on mnist handwritten digits

- Dataset link : <https://www.kaggle.com/datasets/oddrational/mnist-in-csv>.

5.1 Load the dataset

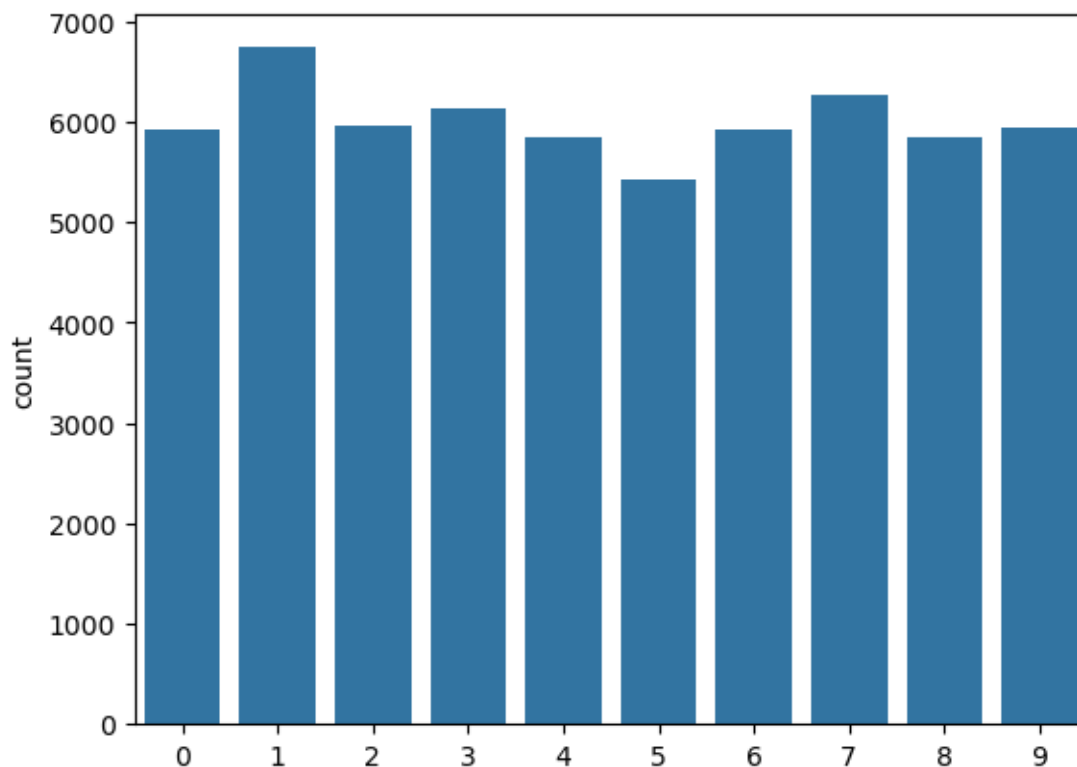
```
[18]: df = pd.read_csv("./data/mnist_train.csv")
```

```
[19]: Y = df["label"].values
      X = df[df.columns[1:]].values
```

5.2 Plot the labels distribution

```
[20]: sns.countplot(x=Y)
```

```
[20]: <Axes: ylabel='count'>
```



5.3 Show some images

```
[21]: def plot_random_images(X,Y, nrows=3, ncols=3, real_labels = None):
```

```
    fig, axes = plt.subplots(nrows=nrows, ncols=ncols)
    n = nrows * ncols
```

```
    idx = np.arange(X.shape[0])
    np.random.shuffle(idx)
    idx = idx[:n]
```

```
    X = X[idx]
    Y = Y[idx]
```

```
    if real_labels is not None:
        real_labels = real_labels[idx]
```

```
    i = 0
```

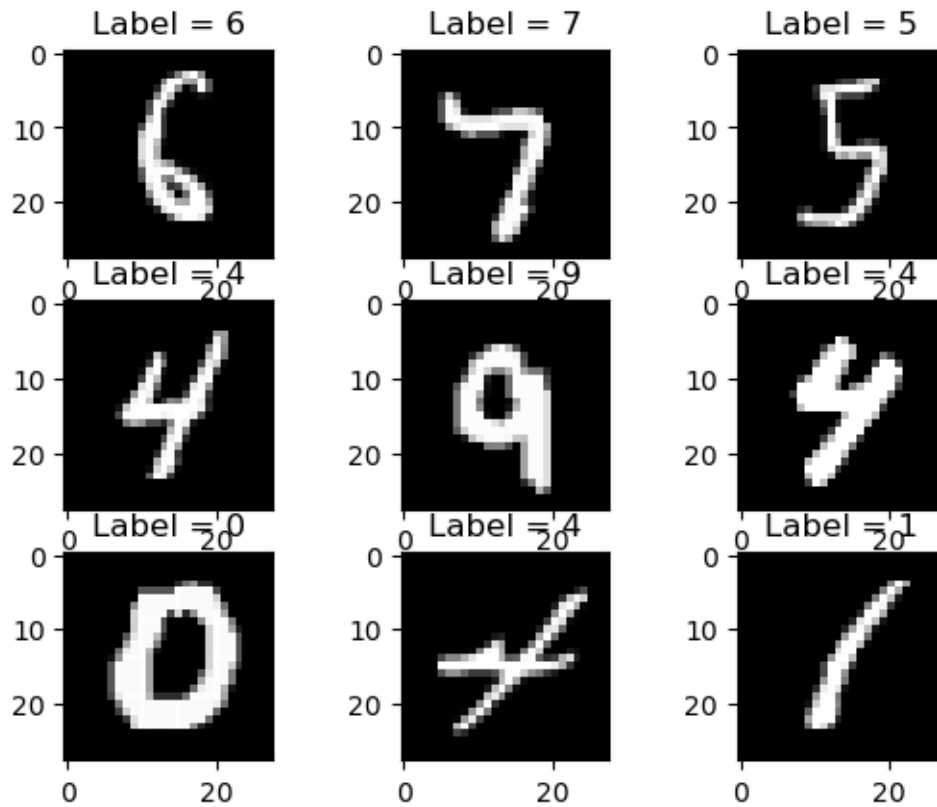
```
    for row in axes:
        for cell in row:
            img = X[i].reshape(28,28)
            cell.imshow(img,cmap="gray")
            i += 1
```

```

        if real_labels is None:
            cell.set_title(f"Label = {Y[i]}")
        else:
            cell.set_title(f"Label = {Y[i]} \n Real Label = \n
↪{real_labels[i]}")
        i += 1

```

```
[22]: plot_random_images(X,Y)
```



5.4 Normalization

```
[23]: X = X / 255.0
```

5.5 Data splitting

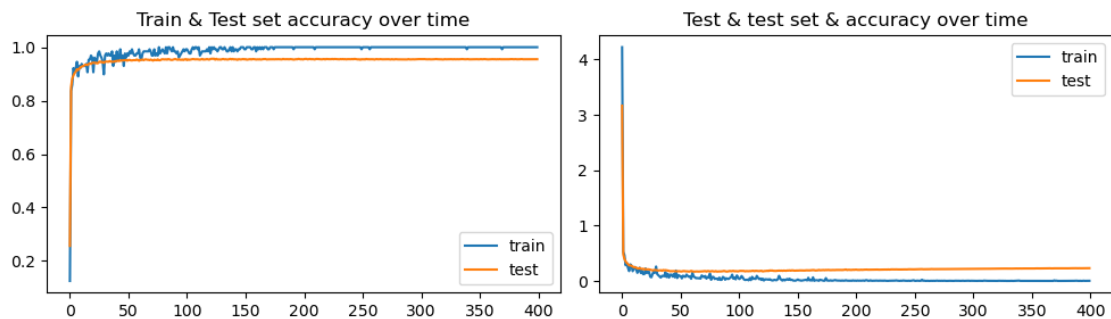
```
[24]: X_train,X_test,y_train,y_test = train_test_split(X, Y, test_size=0.2,stratify=Y)
```


5.6 start the training

```
[25]: params,history = train(  
    X_train,  
    y_train,  
    test_set=(X_test,y_test),  
    eta=0.01,  
    epochs=50,  
    batch_size=128,  
    nh=64  
)
```

5.7 Learning graph

```
[26]: fig, (ax1,ax2) = plt.subplots(nrows=1,ncols=2)  
  
fig.set_size_inches(10,3)  
  
ax1.plot(np.arange(len(history['accuracy'])),history['accuracy'], label="train")  
ax2.plot(np.arange(len(history['loss'])),history['loss'],label="train")  
  
ax1.set_title("Train & Test set accuracy over time")  
  
ax1.plot(np.arange(len(history['test accuracy'])),history['test accuracy'],  
        label='test')  
ax2.plot(np.arange(len(history['test loss'])),history['test loss'], label="test")  
  
ax2.set_title("Test & test set & accuracy over time")  
  
ax1.legend()  
ax2.legend()  
  
plt.tight_layout()
```



5.8 Evaluation

```
[27]: y_train_hat = predict(params, X_train)
      y_test_hat = predict(params, X_test)
```

```
[28]: def get_matrices(y, y_hat):

      accuracy = accuracy_score(y, y_hat)
      f1 = f1_score(y, y_hat, average="macro")
      precision = precision_score(y, y_hat, average="macro")
      recall = recall_score(y, y_hat, average="macro")

      return pd.Series({
          "accuracy":accuracy,
          "f1_score":f1,
          "precision":precision,
          "recall":recall
      })
```

```
[29]: train_metrics = get_matrices(y_train, y_train_hat)
      test_metrics = get_matrices(y_test, y_test_hat)
      metrics = pd.DataFrame(data={
          "train": train_metrics,
          "test":test_metrics
      })
```

```
[30]: metrics
```

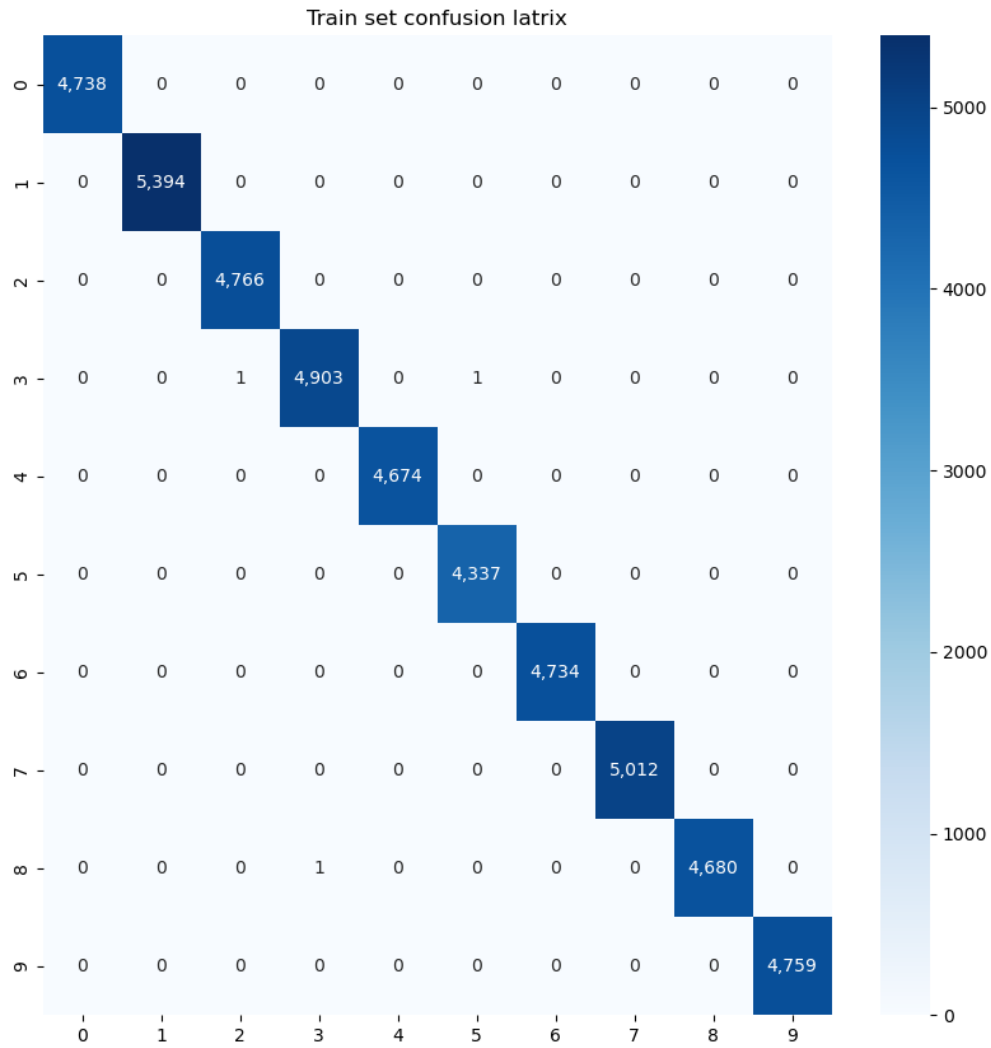
```
[30]:
```

	train	test
accuracy	0.999938	0.955250
f1_score	0.999937	0.954826
precision	0.999936	0.954855
recall	0.999938	0.954849

```
[31]: def plot_confusion_matrix(y, y_hat):
      cm = confusion_matrix(y, y_hat)
      ax = sns.heatmap(data=cm, annot=True, cmap='Blues', fmt=',d')
      ax.get_figure().set_size_inches(10,10)
      return ax
```

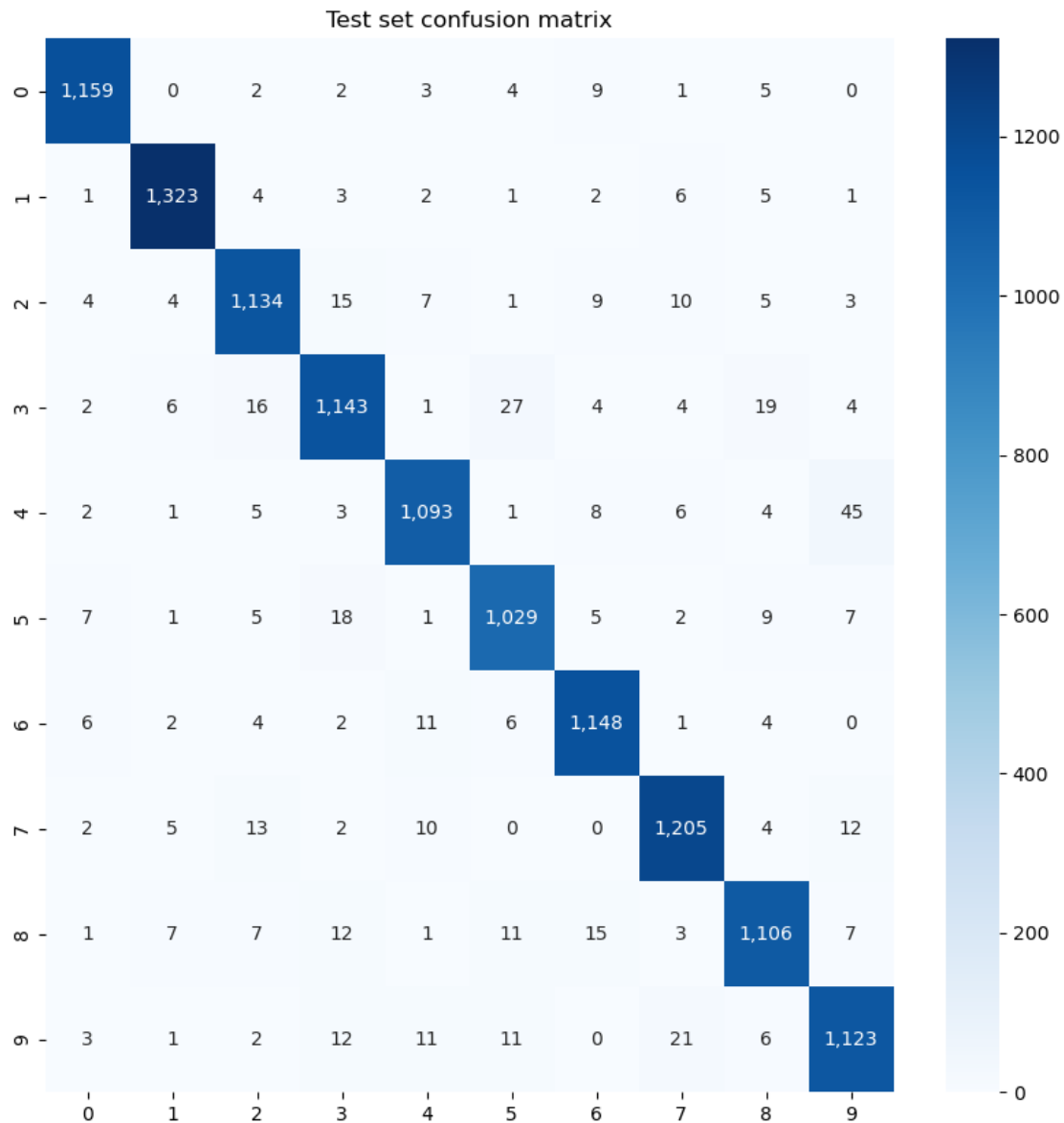
```
[32]: ax = plot_confusion_matrix(y_train, y_train_hat)
      ax.set_title("Train set confusion matrix")
```

```
[32]: Text(0.5, 1.0, 'Train set confusion matrix')
```



```
[33]: ax = plot_confusion_matrix(y_test, y_test_hat)
      ax.set_title("Test set confusion matrix")
```

```
[33]: Text(0.5, 1.0, 'Test set confusion matrix')
```



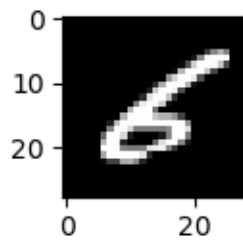
5.9 Error analysis

- show miss-classified data points

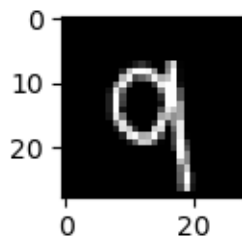
```
[34]: X_test_ = X_test[y_test_hat != y_test]
      y_test_ = y_test[y_test_hat != y_test]
      y_test_hat_ = y_test_hat[y_test_hat != y_test]
```

```
[35]: plot_random_images(X_test_, y_test_hat_, real_labels=y_test_)
      plt.tight_layout(pad=0.25)
```

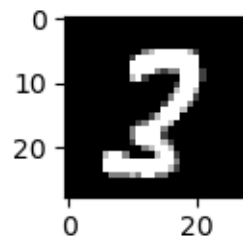
Label = 2
Real Label = 6



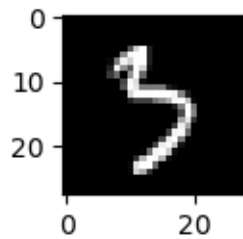
Label = 7
Real Label = 9



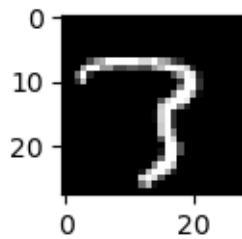
Label = 2
Real Label = 3



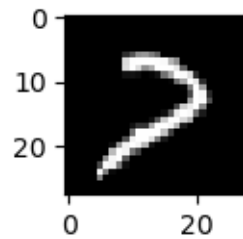
Label = 5
Real Label = 3



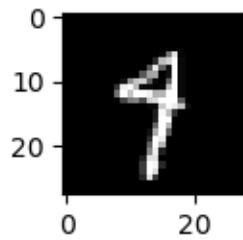
Label = 7
Real Label = 3



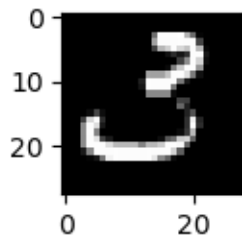
Label = 2
Real Label = 7



Label = 9
Real Label = 4



Label = 5
Real Label = 3



Label = 4
Real Label = 6

