SignLanguage

April 27, 2020

0.0.1 Before nothing please check that we are running python3.6+ and have a GPU

[0]:
```
!python3 --version
```

```
Python 3.6.9
```

0.1 Lets import some libraries so we can see the dataset

[0]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[0]:
```
def dataframe_to_data(raw):
    '''Simple function that takes a datafram and
       split it into 2 parts, the labels col and everything else'''
    labels = raw['label']
    raw.drop('label', axis=1, inplace=True)

    return raw.values, labels.values
```

We are going to download the dataset, and use pandas to read it and parse it, then we are going to use our costume function to split it into labels and data.

Finally we are going to split the test dataset into validation and actual test

With the split been made we show how many elements each section has

[0]:
```
base_url = "https://raw.githubusercontent.com/SoyOscarRH/Learning/master/UNAM/
↪MachineLearning/Sign/dataset/"

data_url = base_url + "sign_mnist_test.csv"
data_raw = pd.read_csv(data_url, sep=",")
```

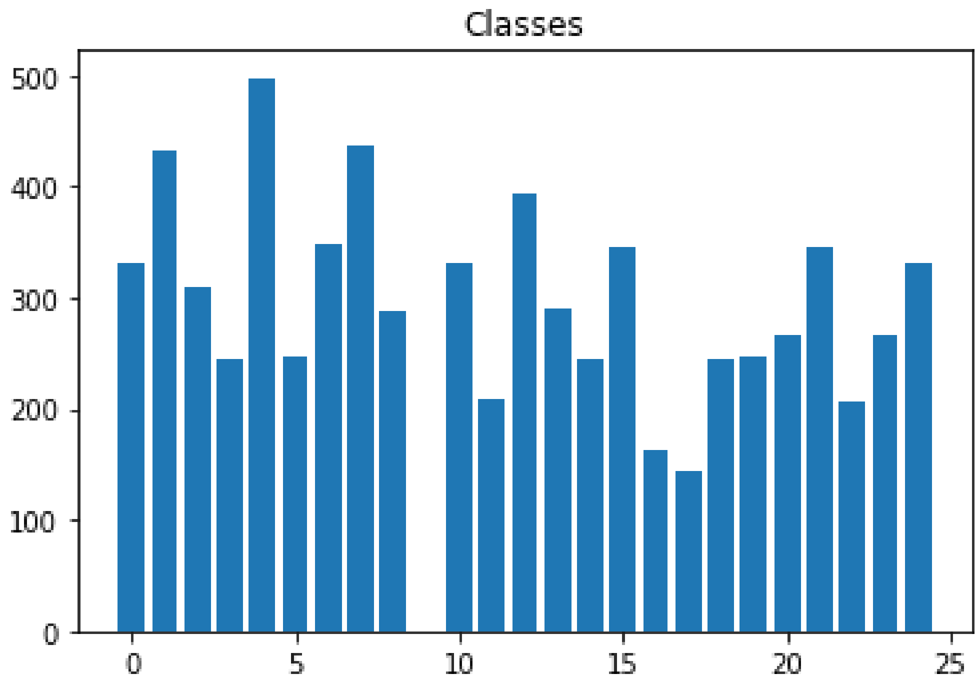
```
data, labels = dataframe_to_data(data_raw)
print("data:", len(data), "elements")

test_url = base_url + "sign_mnist_test.csv"
test_validation_data_raw = pd.read_csv(test_url, sep=",")

n = len(test_validation_data_raw)
test_data_raw       = test_validation_data_raw.loc[:n//2, :].copy()
validation_data_raw = test_validation_data_raw.loc[n//2:, :].copy()

test_data, labels_test = dataframe_to_data(test_data_raw)
print("test:", len(test_data), "elements")

validation_data, labels_validation = dataframe_to_data(validation_data_raw)
print("validation:", len(validation_data), "elements")
```

```
data: 7172 elements
test: 3587 elements
validation: 3586 elements
```

Now we are going to look at the number of elements each class has (in training)

[0]:
```
import collections
counter = collections.Counter(labels)
print(counter)
fig, ax = plt.subplots()
ax.bar(list(counter.keys()), list(counter.values()))
plt.title("Classes")
plt.show()
```

```
Counter({4: 498, 7: 436, 1: 432, 12: 394, 6: 348, 15: 347, 21: 346, 24: 332, 10:
331, 0: 331, 2: 310, 13: 291, 8: 288, 23: 267, 20: 266, 19: 248, 5: 247, 14:
246, 18: 246, 3: 245, 11: 209, 22: 206, 16: 164, 17: 144})
```



[0]:
```
IMAGE_SIZE = 28
```

[0]:
```
def num_to_letter(num: int) -> str:
    """ siple helper function to take an int and transform it into a letter so␣
 ↪we can show it to you"""
    start = ord('a')
    return chr(num + start)

examples = [num_to_letter(i) for i in range(26)]
print(examples)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

0.2 Interact with the dataset

[0]:
```
from ipywidgets import interact

@interact(sample = (0, len(data)))
def show_image_dataset(sample):
    """ Given an index we show in a heatmap the i'nth image in the dataset"""
    info, label = data[sample], labels[sample]
    pixels = info.reshape(IMAGE_SIZE, IMAGE_SIZE)

    _, ax = plt.subplots(figsize=(3.5, 3.5))
    sns.heatmap(data=pixels, cmap="YlGnBu", ax=ax)
```

```
    plt.title(num_to_letter(label), fontsize=30, color="#00008b")

    plt.show()
```

```
interactive(children=(IntSlider(value=3586, description='sample', max=7172), Output()), _dom_cl
```

1 Declare the arquitecture of the neuronal network

[0]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
```

We transform our data into tensors and reshape each element so it is actually a 2D matrix

[0]:
```
data = [pic.reshape(1, IMAGE_SIZE, IMAGE_SIZE) for pic in data]
validation_data = [pic.reshape(1, IMAGE_SIZE, IMAGE_SIZE) for pic in␣
 ↪validation_data]
test_data = [pic.reshape(1, IMAGE_SIZE, IMAGE_SIZE) for pic in test_data]

x                         = torch.FloatTensor(data)
y                         = torch.LongTensor(labels.tolist())

validation_data_formated = torch.FloatTensor(validation_data)
validation_labels         = torch.LongTensor(labels_validation.tolist())

test_data_formated        = torch.FloatTensor(test_data)
test_labels               = torch.LongTensor(labels_test.tolist())
```

Basic hyperparams

[0]:
```
epochs = 25
batch_size = 64
learning_rate = 0.0015
```

[0]:
```
class Network(nn.Module):

    def __init__(self):
        super(Network, self).__init__()

        # convolutional layer (sees 28x28x1 image tensor)
        self.conv1    = nn.Conv2d(in_channels=1, out_channels=10, kernel_size=3)
        self.pool1    = nn.MaxPool2d(2)

        # convolutional layer (sees 10x tensor)
        self.conv2    = nn.Conv2d(in_channels=10, out_channels=20,␣
 ↪kernel_size=3)
```

```
        self.pool2    = nn.MaxPool2d(2)

        # convolutional layer (sees 20x tensor)
        self.conv3    = nn.Conv2d(in_channels=20, out_channels=30,␣
↪kernel_size=3)
        self.dropout1 = nn.Dropout2d(0.4)

        self.fc3      = nn.Linear(30 * 3 * 3, 256)
        self.fc4      = nn.Linear(256, 25)

        self.softmax  = nn.LogSoftmax(dim=1)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = F.relu(x)
        x = self.dropout1(x)

        x = x.view(-1, self.fc3.in_features)

        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))

        return self.softmax(x)


    def evaluate(self, x):
      """Take a element through the nn and out the best candidate class"""
      if torch.cuda.is_available(): x = x.cuda()
      output = self(x)
      return torch.max(output.data, 1)[1]


    def step_train(self, optimizer, loss_fn, x, y):
        """ Do one step in training, return the loss from back prop"""
        x = Variable(x)
        y = Variable(y)
        if torch.cuda.is_available(): x, y = x.cuda(), y.cuda()

        optimizer.zero_grad()
```

```
        loss = loss_fn(self(x), y)
        loss.backward()
        optimizer.step()

        return loss.item()


    def accuracy(self, predictions, labels) -> float:
        """ given 2 iterables representing the actual labels and the prediction␣
↪of our system we output
            our estimated accuracy"""
        correct = 0
        for prediction, label in zip(predictions, labels):
            if prediction == label: correct += 1

        return correct / len(predictions)
```

Instanciate a network, make it reproducible

[0]:
```
torch.manual_seed(0)
np.random.seed(0)
net = Network()
if torch.cuda.is_available(): net.cuda()
print(net)
```

```
Network(
  (conv1): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv3): Conv2d(20, 30, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout2d(p=0.4, inplace=False)
  (fc3): Linear(in_features=270, out_features=256, bias=True)
  (fc4): Linear(in_features=256, out_features=25, bias=True)
  (softmax): LogSoftmax()
)
```

We are going to use stocastic gradient descent with 0.7 as our default momementum

[0]:
```
optimizer = optim.SGD(net.parameters(), learning_rate, momentum=0.7)
loss_fn = nn.CrossEntropyLoss()
```

1.1 Our train loop

We are going to use minibatch to iterate over the training data, on each step we are going to record the training loss and evaluate the current model using the validation set, so we can actually store the best model in terms of validation loss.

[0]:
```python
from statistics import mean

valid_loss_min = np.Inf
train_losses, valid_losses = [], []
for epoch in range(epochs):
    # train the model
    net.train()
    train_loss = []
    for i in range(0, x.shape[0], batch_size):
        loss = net.step_train(optimizer, loss_fn, x[i : i + batch_size], y[i :␣
↪i + batch_size])
        train_loss.append(loss)

    train_loss = mean(train_loss)
    train_losses.append(train_loss)

    # validate the model
    net.eval()
    valid_loss = net.step_train(optimizer, loss_fn, validation_data_formated,␣
↪validation_labels)
    valid_losses.append(valid_loss)

    print(f'Epoch: {epoch + 1} \t', end="")
    print(f'Training Loss: {round(train_loss, 6)} \t Validation Loss:␣
↪{round(valid_loss, 6)}')

    # save model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        before, after = round(valid_loss_min, 6), round(valid_loss, 6)
        print(f'Validation loss min: ({before} --> {after}). \nSaving model')

        torch.save(net.state_dict(), 'best_model.pt')
        valid_loss_min = valid_loss

    print()
```

```
Epoch: 1        Training Loss: 3.12588   Validation Loss: 2.8588
Validation loss min: (inf --> 2.8588).
Saving model

Epoch: 2        Training Loss: 2.708383         Validation Loss: 2.434011
Validation loss min: (2.8588 --> 2.434011).
Saving model

Epoch: 3        Training Loss: 2.260573         Validation Loss: 1.87094
Validation loss min: (2.434011 --> 1.87094).
Saving model
```

```
Epoch: 4        Training Loss: 1.803474         Validation Loss: 1.292004
Validation loss min: (1.87094 --> 1.292004).
Saving model

Epoch: 5        Training Loss: 1.408076         Validation Loss: 0.924011
Validation loss min: (1.292004 --> 0.924011).
Saving model

Epoch: 6        Training Loss: 1.121076         Validation Loss: 0.714124
Validation loss min: (0.924011 --> 0.714124).
Saving model

Epoch: 7        Training Loss: 0.829007         Validation Loss: 0.504813
Validation loss min: (0.714124 --> 0.504813).
Saving model

Epoch: 8        Training Loss: 0.67426  Validation Loss: 0.39776
Validation loss min: (0.504813 --> 0.39776).
Saving model

Epoch: 9        Training Loss: 0.581228         Validation Loss: 0.367483
Validation loss min: (0.39776 --> 0.367483).
Saving model

Epoch: 10       Training Loss: 0.511361         Validation Loss: 0.351019
Validation loss min: (0.367483 --> 0.351019).
Saving model

Epoch: 11       Training Loss: 0.487887         Validation Loss: 0.331776
Validation loss min: (0.351019 --> 0.331776).
Saving model

Epoch: 12       Training Loss: 0.451158         Validation Loss: 0.324673
Validation loss min: (0.331776 --> 0.324673).
Saving model

Epoch: 13       Training Loss: 0.416918         Validation Loss: 0.324102
Validation loss min: (0.324673 --> 0.324102).
Saving model

Epoch: 14       Training Loss: 0.414308         Validation Loss: 0.322166
Validation loss min: (0.324102 --> 0.322166).
Saving model

Epoch: 15       Training Loss: 0.400752         Validation Loss: 0.320976
Validation loss min: (0.322166 --> 0.320976).
Saving model
```

```
Epoch: 16       Training Loss: 0.393943         Validation Loss: 0.319565
Validation loss min: (0.320976 --> 0.319565).
Saving model

Epoch: 17       Training Loss: 0.379305         Validation Loss: 0.321035

Epoch: 18       Training Loss: 0.37971   Validation Loss: 0.318718
Validation loss min: (0.319565 --> 0.318718).
Saving model

Epoch: 19       Training Loss: 0.367212         Validation Loss: 0.318221
Validation loss min: (0.318718 --> 0.318221).
Saving model

Epoch: 20       Training Loss: 0.360242         Validation Loss: 0.319724

Epoch: 21       Training Loss: 0.365743         Validation Loss: 0.318356

Epoch: 22       Training Loss: 0.359045         Validation Loss: 0.318892

Epoch: 23       Training Loss: 0.354851         Validation Loss: 0.317742
Validation loss min: (0.318221 --> 0.317742).
Saving model

Epoch: 24       Training Loss: 0.35308   Validation Loss: 0.318673

Epoch: 25       Training Loss: 0.317728         Validation Loss: 0.216968
Validation loss min: (0.317742 --> 0.216968).
Saving model
```

Recover the "best" model

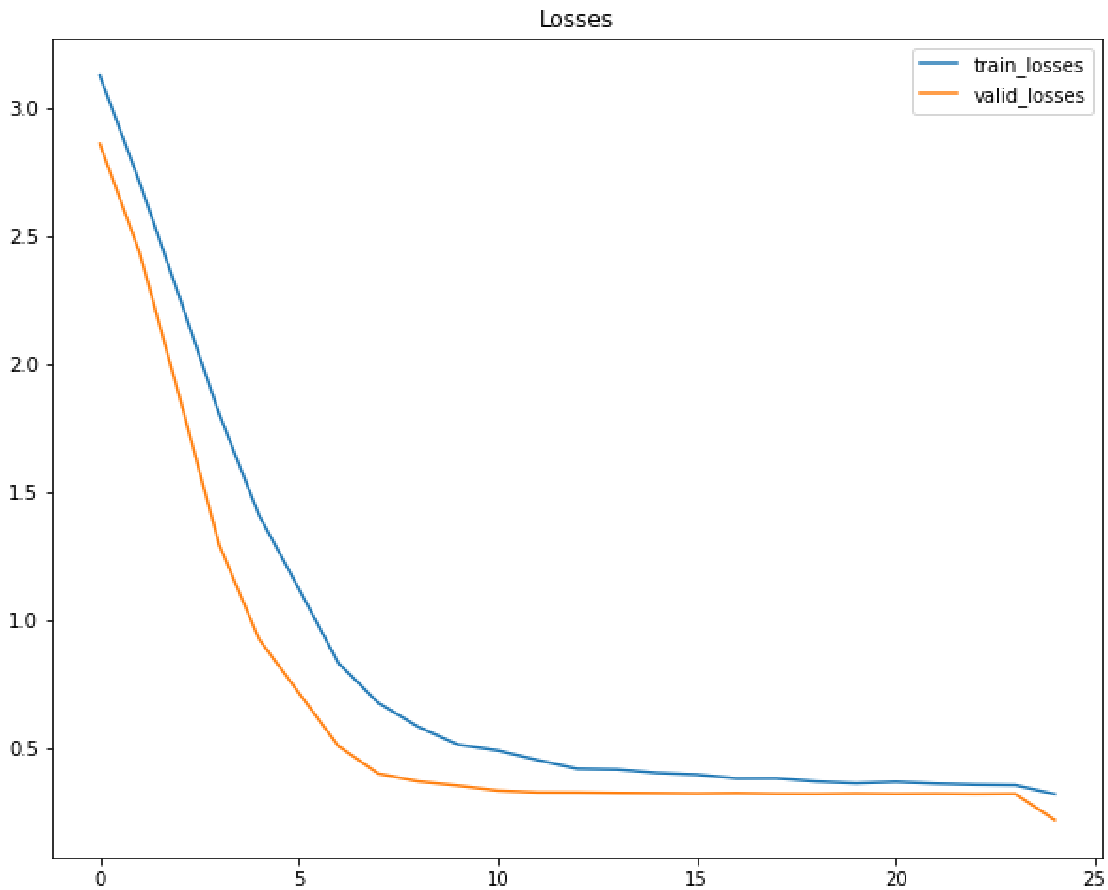
[0]:
```
net.load_state_dict(torch.load('best_model.pt'))
```

[0]:
```
<All keys matched successfully>
```

Show more visually the change in loss

[0]:
```
%matplotlib inline

plt.figure(figsize=(10,8))
plt.plot(train_losses, label="train_losses")
plt.plot(valid_losses, label="valid_losses")
plt.legend()
plt.title("Losses")
```

[0]:
```
Text(0.5, 1.0, 'Losses')
```



[0]:
```
## Switch to evaluate mode, no more backprop and no dropout
net.eval()
```

[0]:
```
Network(
  (conv1): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv3): Conv2d(20, 30, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout2d(p=0.4, inplace=False)
  (fc3): Linear(in_features=270, out_features=256, bias=True)
  (fc4): Linear(in_features=256, out_features=25, bias=True)
  (softmax): LogSoftmax()
)
```

[0]:
```
@interact(sample = (0, len(test_data)))
def show_image_dataset(sample):
```

```
    """Same helper iteractive tool but now we evaluate our model and get the␣
↪actual prediction"""
    info, label  = test_data[sample], labels_test[sample]
    pixels       = info.reshape(IMAGE_SIZE, IMAGE_SIZE)

    input_pixels    = info.reshape(1, IMAGE_SIZE, IMAGE_SIZE).tolist()
    predicted_label = net.evaluate(torch.FloatTensor([input_pixels]))

    _, ax = plt.subplots(figsize=(3.5, 3.5))
    sns.heatmap(data=pixels, cmap="YlGnBu", ax=ax)

    actual    = f"{num_to_letter(label)}  actual"
    predicted = f"{num_to_letter(predicted_label)}  predicted"
    plt.title(actual + "\n" + predicted, fontsize=20, color="#00008b",␣
↪loc="left")

    plt.show()
```

```
interactive(children=(IntSlider(value=1793, description='sample', max=3587), Output()), _dom_cl
```

1.2 Lets measuare the accuracy of the model

[0]:
```
prediction = net.evaluate(Variable(test_data_formated))
accuracy = net.accuracy(prediction, test_labels)

n = len(prediction)
correct = int(n * accuracy)
print(f"Correct predictions: {correct} / {n}: ", end="")
print(f"{round(accuracy, 6) * 100}%")
```

```
Correct predictions: 3367 / 3587: 93.86670000000001%
```