LABO IA ML: TP1

November 28, 2019

1 Load & unpack the dataset

In this tutorial, we will make use the FashionMINST dataset from Zalando. There's 10 types of clothes, thus it is a classification problem.

You can download manually the dataset from their GitHub [1] or use Tensorflow 2.0 (tf) to download it.

```
import tensorflow as tf
(x_train, y_train),(x_test, y_test) = tf.keras.datasets.
                                    fashion_mnist.load_data()
```

In this case, tf will automatically split the dataset in 2. The training dataset and the test dataset. If you make your own dataset, you will have to manually split your data set (80% Training and 20% Testing).

Now you can check out the shape of your x_train.

```
import numpy as np
print(x_train.shape) # (60000, 28, 28)
```

We have 60000 images of size $28x28$ in x_train. Here we have gray scaled images ($28x28$), RBG images would have been of size $28x28x3$ since there is 3 colors. For an image containing the Alpha channel, the size would have been $28x28x4$

2 Prepare the dataset

Now that we have loaded our data set, we need to prepare it before training. We want to organise our X such has we have one line per image. So $x\_train[0]$ must contains all the pixels of the first image. $x\_train[1]$ contains all the pixels of the second image. etc.

You have to reshape your training and testing dataset to a new shape.

```
x_train = x_train.reshape("""FIX ME""")
x_test = x_test.reshape("""FIX ME""")
```

[1] https://github.com/zalandoresearch/fashion-mnist#get-the-data

An other way to do it is to use the *Flatten* function of Tensorflow 2.0, our images are squares, *Flatten* just takes that square and turns it into a 1-dimensional set.

```
model.add(Flatten(input_shape=("""FIX ME""")))
```

3 Linear model

3.1 One Hot Encoder

3.1.1 Prepare labels: y_train $ y_test

y_train has values between $[0-9]$ since it has 10 classes. In a classification model, you will have has much output has classes. So in this model we will need 10 different outputs. Each output will give the information respective of its class.

The "y_train" are the expected classes for each images. But we need 10 expected output for each classes of each image.

```
0 -> [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
...
8 -> [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
9 -> [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

We can manually create you own function to translate the *y_train*, or use tf's *one_hot* [2] function

Modify your *y_train* and y_test with the function.

```
y_train_one_hot = tf.one_hot("""FIX ME""")
y_test_one_hot = tf.one_hot("""FIX ME""")
```

```
print(y_train)
#[9 0 0 ... 3 0 5]

print(y_train_one_hot)
#tf.Tensor(
#[[0. 0. 0. ... 0. 0. 1.]
# [1. 0. 0. ... 0. 0. 0.]
# [1. 0. 0. ... 0. 0. 0.]
# ...
# [0. 0. 0. ... 0. 0. 0.]
# [1. 0. 0. ... 0. 0. 0.]
# [0. 0. 0. ... 0. 0. 0.]], shape=(60000, 10), dtype=float32)
```

[2] https://www.tensorflow.org/api_docs/python/tf/one_hot

3.1.2 Create model

Create a Sequential model with keras tensorflow 2.0. This model should have only one neuron since it is a linear model. Once you added your first layer of one neuron, we will need the output layers. The output layer has as many output as classes. This layer will need a special activation function, the $softmax$. The sum of all output will give 1 while using $softmax$.

For example if we have 3 classes/outputs:

```
print(my_result_of_each_output)
# [0.6590011388859679, 0.2424329707047139, 0.09856589040931818]
print(sum(my_result_of_each_output))
# 1.0
```

In this example we can say that the probability that the given input is the first class is 65.90%, second class is 24.24% and last class is 9.85%.

You should have only 2 layers for this model. One for the linear neuron and one for the outputs.

Now you have to compile your model. You can optimize your model using specific *optimizer* [3]. You will have also to chose a Loss and a Metrics to visualize how you model performs.

4 Fit model

Your model is compiled, you can now feed your model with your inputs. Try to fit you model with the fit function in tensorflow 2.0.

```
model.fit("""FIX ME""")
```

You have to feed your model with your x_train, y_train. You can also give your validation data in argument. The $batch\_size$ is the number of images taken in x_train you will feed in once to your GPU. Having a large batch size will make your model fit faster.

If you have 60000 images in your X, and you chose 100 has $batch\_size$, then the CPU has to transfer 600 times 100 images from your RAM (CPU side) to your VRAM (GPU side) for one epoch. Thus you will lose lot of time to copy the data for only one epoch. If you chose a $batch\_size$ of 20000, you will load 20000 images 3 times for one epoch. It will be faster since you have less data transfer, but you need to have sufficient VRAM in your GPU. So the batch size will depend on the VRAM of your GPU.

It is very useful to save your model to reload it sometimes later. A model can be fitted multiple times.

```
# Create, compile model
model.save("./my_model_path/model_random.h5") # your model has been
                                    save with random weights since
                                   you never fit you model
model.fit("""params""")
```

[3] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

```
model.save("./my_model_path/model_fit.h5") # fitted model.


model_2 = tf.keras.models.load_model("./my_model_path/model_fit.h5"
                                     )
model_2.fit("""params""")
model_2.fit("""params""")
model_2.fit("""params""")
model_2.save("./my_model_path/model_2_3_fits.h5") # copy of model
                                     fitted 3 times.
```

4.1 10 Models

You can also create 10 models. One model to detect the first class and not the others. Y has to be set to 1 to the first class and 0 (or -1, depending your activation function) to the other one. You will have to fit 10 models with the same inputs, but with the appropriate labels.

The model will only need one layer. The first layer will be the last layer since it output will between 0 and 1.

To predict, you will have to feed the 10 models with same image and compare the 10 results. The model with the highest result will be the given class of this image.

So 1 model or 10 models? We don't have the answer, you will have to find it!

5 Normalization

5.1 Min-Max

It is very useful to normalize your data before training. It will make your training faster. Sometimes, depending of the optimizer or the loss function, it will compute squares or exponential. Your pixel are encoded between 0 and 255. Imagine your computer has to make this computation: 255^{10} the result is 116252367019153321289062 5. Now imagine your pixels are encoded in float between 0 and 1: 1^{10} is 1. It will be way faster for the GPU. Having small inputs will make you have small weights (or at least smaller).

```
x_train = x_train / 255
x_test = x_test / 255
```

The general formula is : $x_i = \frac{x_i - max(X)}{max(X) - min(X)}$

6 Neural Network

For the neural Network, you can take your One Hot Linear model, edit your first layers to get more neurons. You can more layers with various amount of

neurons inside it. You will have to change the activation function of all your layers expect the last one (softmax).

Try with different amount of layers, and different amount of neurons in each of your layers.