11/1/2019

Siamese network

Artificial Intelligent

Ruochen Hou
QUEENSLANDE UNIVERCITY OF TECHNOLOGYR

1. Introduction

In theory, Siamese networks are a kind of neural architecture that measures how different two images are. The inputs for the Siamese Network are a pair of images. The images are positive if they are similar and their target value will be set to 0. On the other hand, the images are negative if they are different and their target value will be set to 1.

Each image is processed by exactly the same base_neural network and these subnetworks share the same weight. The output of the base_neural network are two vectors that will be the input to a layer which measures the distance between those two images.

Siamese Networks can be implemented to recognize faces or handwriting or to index documents among other applications.

2. Methodology

2.1 Creation and verification of the dataset

To create our Siamese Network, we have used the dataset called “fashion_mnist”. In total, this dataset contains 10 classes (0-9) and each class will represent a clothing item.

To load the data, we executed the following line of code: *keras.datasets.fashion_mnist.load_data.*

By default, this dataset was divided into (x_train, y_train) and (x_test , y_test) but this is not the way it should be divided according to the requirements for assessment 2 .

Therefore, we combined the dataset to have all the labels together (y_combine) and all the images (x_combine) together. (Figure1)

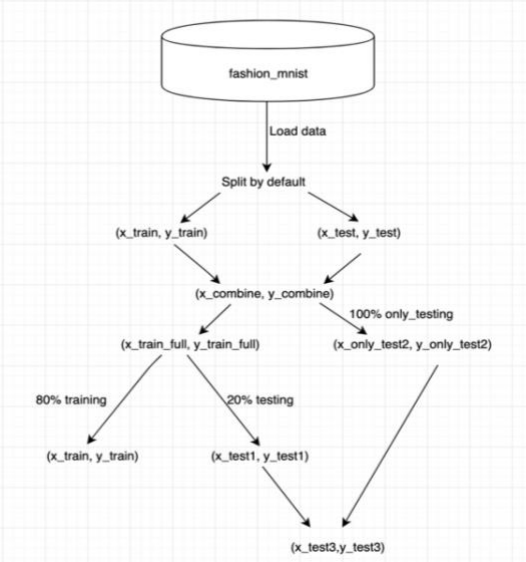

Figure 1. Dividing the datasets.

Then, we can start splitting the datasets according to the requirements for assessment 2 as it is shown in the Table1

Dataset	Classes
x_train **y_train**	80% ["top", "trouser", "pullover", "coat", "sandal", "ankle”, “boot"]
x_test1 **y_test1**	20% ["top", "trouser", "pullover", "coat", "sandal", "ankle", "boot"]
x_only_test2 **y_only_test2**	["dress", "sneaker", "bag", "shirt"].
X_test3 **Y_test3**	["top", "trouser", "pullover", "coat", "sandal", "ankle boot", "dress", "sneaker", "bag", "shirt"].

Table 1.Datasets and classes.

2.2 Implement and test the contrastive loss function

The contrastive loss function that the Siamese model will use for learning is the following:

$$\mathcal{L}_c(P_i, P_j) = \begin{cases} \frac{1}{2} D(P_i, P_j)^2, & \text{if } y_{ij} = 0 \\ \frac{1}{2} \max(0, m - D(P_i, P_j))^2, & \text{if } y_{ij} = 1, \end{cases}$$

where $D(P_i, P_j)$ is the Euclidean distance between $f(P_i)$ and $f(P_j)$. The margin $m > 0$ determines how far the embeddings of a negative pair should be pushed apart.

From the formula, D is the Euclidean distance and therefore, we calculate first the Euclidean Distance with the following functions:

```
def eucl_dist_output_shape(shapes):
    shape1, shape2 = shapes
    return (shape1[0], 1)

def euclidean_distance(vects):
    x, y = vects
    sum_square = K.sum(K.square(x - y), axis=1, keepdims=True)
    return K.sqrt(K.maximum(sum_square, K.epsilon()))
```

The vectors x and y are the outputs of the Siamese network for each pair of images.

Then, we can calculate the *contrastive lost function*.

According to the formula, the margin indicates how distant two negative images should be. As a consequence, the margin should be bigger than 0. We can test the loss function by tuning the margin value. In our function we set that the margin value to 1 and also we tested with the value margin=5. However, we decided that a margin of 1 was more suitable for our model.

```
def contrastive_loss(y_true, y_pred):

    margin = 1
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(margin - y_pred, 0))
    return K.mean((1 - y_true) * square_pred + y_true * margin_square)
def testing_loss(y_true, y_pred):

    margin = 5
    square_pred = K.square(y_pred)
    margin_square = K.square(K.maximum(margin - y_pred, 0))
    return K.mean((1 - y_true) * square_pred + y_true * margin_square)
```

2.3 Build the Siamese network

For our Siamese network we implemented a functional API because it will accept multiple inputs and will generate multiple outputs simultaneously that cannot be achieved using the sequential model.

As an experiment, we have created a Base Model and a Complex Siamese model. The architectures for each model will be shown in the Experimental Results and Evaluation section.

The base model will not have any layer that contains regularization methods. The complex Siamese model will have a dropout layer and a regularized layer. The experiment will observe the behaviour and accuracy of the Base model compared with the complex Siamese model.

2.4 Training of the Siamese network

For this part, we have a function def run_base_network() that include as parameters the training, testing and validation datasets. Inside the function we have called the function that creates the pairs, reshapes the input, trains the network and compute the final accuracy of each dataset. The same process was done for the def run_siamese_network() only that it will use the complex siamese model to train the network.

3. Experimental Results & Evaluation

To test the Siamese Network generalization capability, we have created two models, base model and Complex Siamese model.

3.1 Base Model

To build the base model, for layer 1 Conv2D, we set 32 kernels with size is 3*3 and "relu" activation function to extract the features of training dataset. Next, layer 2 Conv2D, we set 64 kernels with size is 3*3. Then, we use MaxPooling2D as layer 3 to choose the highest value of a 2*2 matrix from previous layer. After that we utilize Flatten function to convert the previous layer data to two-dimension layer. Finally, add two dense layers with 128 output classes (shown in below picture).

```
def create_base_network(input_shape):

    input_layer = Input(input_shape)
    hidden1 = Conv2D(32, kernel_size=(3, 3),activation='relu')(input_layer)
    hidden2 = Conv2D(64, (3, 3), activation='relu')(hidden1)
    hidden3 = MaxPooling2D(pool_size=(2, 2))(hidden2)
    hidden4 = Flatten()(hidden3)
    hidden5 = Dense(128, activation='relu')(hidden4)
    output = Dense(128,activation='relu')(hidden5)
    return Model(input_layer, output)
```

To display the results for the Base Model we have created the following function:

```
def dispaly_base_network():

    train_test,only_testing,total_class = dataset_class()
    epochs = 20
    optimizer = RMSprop()
    x_train,y_train,x_test1,y_test1,x_only_test2,y_only_test2,x_test3,y_test3 = data_preparation()
    x_validation = x_test1
    y_validation = y_test1
    train_sets = train_test

    print('----------------------------Base model---------------------------- ')
    print('-----------------------------Test1------------------------------ ')
    test_sets = train_test
    tr_acc, te_acc,information,score = run_base_network(x_train, y_train, x_test1, y_test1, epochs,
                                                        train_sets, test_sets, x_validation,y_validation,optimizer)
    print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
    print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
    print('* Evaluation test loss: %.2f' % score[0])

    print('-----------------------------only_Test2------------------------------ ')
    test_sets = only_testing
    tr_acc, te_acc,information,score = run_base_network(x_train, y_train, x_only_test2, y_only_test2, epochs,
                                                        train_sets, test_sets, x_validation,y_validation,optimizer)
    print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
    print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
    print('* Evaluation test loss: %.2f' % score[0])

    print('-----------------------------Test3------------------------------ ')
    test_sets = total_class
    tr_acc, te_acc,information,score = run_base_network(x_train, y_train, x_test3, y_test3, epochs,
                                                        train_sets, test_sets, x_validation,y_validation,optimizer)
    print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
    print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
    print('* Evaluation test loss: %.2f' % score[0])
    plot_result(information, epochs, "Base network")
```

We have set the epochs to 20 repetitions and we will display the value for accuracy from 1 to 20.

Accuracy Level for Base Model		
Test Dataset	Training accuracy	Testing accuracy
First pair dataset [0,1,2,4,5,9]	99.85%	95.99%
Second pair dataset [0 - 9]	99.73%	79.85%
Third pair dataset [3,7,8,6]	99.70%	61.43%

Table2. Accuracy Level for Base Network

For the First pair dataset with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"] we had a high accuracy in the training set: 99.85% and in the test set: 95.99% because both the training and testing belong to the same labels and therefore the data is known data.
For the Second pair dataset with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot”] UNION ["dress", "sneaker", "bag", "shirt"] the accuracy in the training set is high: 99.73% However, the accuracy for the test set is slightly lower with 79.85%. Due to the combination of the datasets involved in this experiment.
For the Third pair dataset with labels ["dress", "sneaker", "bag", "shirt"] we had a high accuracy in the training set: 99.70% but significantly low accuracy with the test set: 61.43%. This is because we have use as the test set the unseen data.
Then, we plot the training and the validation datasets vs epochs.

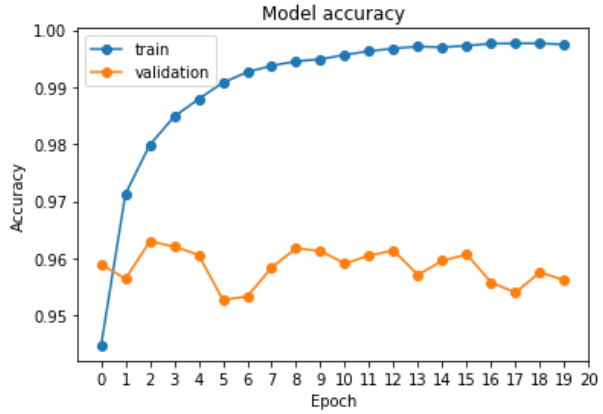

Figure 2. Model Accuracy

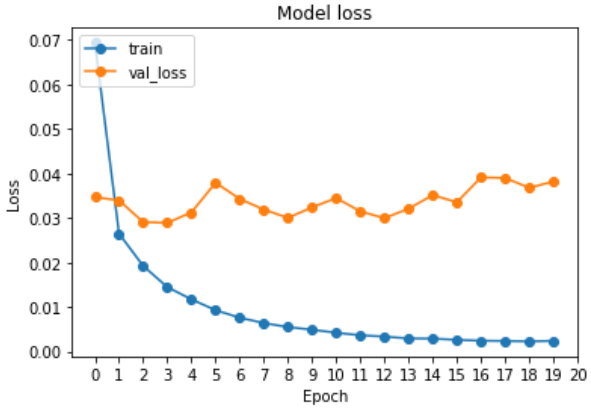

Figure 3. Model Loss

According to the plot for the Model accuracy as the training dataset (Blue line) increased its accuracy nearly to 100%, the validation test oscillates between values from 95% to approximately 96%. That indicates that the model is not improving its accuracy which could imply that the model is not learning.
As for the Model loss, we can clearly see that the training loss (Blue line) was decreasing after each iteration. However, the validation loss (Orange line) remained fluctuating among values 0.03 and 0.04 , which means this model is staring to underfit after the first epoch. This result is basically the same as we were expecting, since in the base model don’t include the dropout and regularization techniques. Therefore, in the next model, we will try to add both dropout and regularization techniques to solve this problem.

3.2 Complex Siamese Model

This model uses layers from the base model and additions dropout and reguarization layers, in order to improve generalization (shown in the picture below).

```
def create_siamese_network(input_shape):

    input_layer = Input(input_shape)
    hidden1 = Conv2D(32, kernel_size=(3, 3),activation='relu')(input_layer)
    hidden2 = Conv2D(64, (3, 3), activation='relu')(hidden1)
    hidden3 = MaxPooling2D(pool_size=(2, 2))(hidden2)
    hidden4 = Flatten()(hidden3)
    hidden5 = Dense(128, activation='relu',
                    kernel_regularizer=regularizers.l2(0.01),
                    bias_regularizer=regularizers.l1(0.01))(hidden4)
    hidden6 = Dropout(0.1)(hidden5)
    output = Dense(128,activation='relu')(hidden6)
    return Model(input_layer, output)
```

To display the results for the Complex Siamese Model we have created the following function:

```
def dispaly_siamese_network():

    train_test,only_testing,total_class = dataset_class()
    epochs = 20
    optimizer = RMSprop()
    x_train,y_train,x_test1,y_test1,x_only_test2,y_only_test2,x_test3,y_test3 = data_preparation()
    x_validation = x_test1
    y_validation = y_test1
    train_sets = train_test


    print('----------------------------Siamese model---------------------------- ')
    print('----------------------------Test1---------------------------- ')
    test_sets = train_test
    tr_acc, te_acc,information,score = run_siamese_network(x_train, y_train, x_test1, y_test1, epochs,
                                                           train_sets, test_sets, x_validation,y_validation,optimizer)
    print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
    print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
    print('* Evaluation test loss: %.2f' % score[0])


    print('----------------------------only_Test2---------------------------- ')
    test_sets = only_testing
    tr_acc, te_acc,information,score = run_siamese_network(x_train, y_train, x_only_test2, y_only_test2, epochs,
                                                           train_sets, test_sets, x_validation,y_validation,optimizer)
    print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
    print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
    print('* Evaluation test loss: %.2f' % score[0])


    print('----------------------------Test3---------------------------- ')
    test_sets = total_class
    tr_acc, te_acc,information,score = run_siamese_network(x_train, y_train, x_test3, y_test3, epochs,
                                                           train_sets, test_sets, x_validation,y_validation,optimizer)
    print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
    print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
    print('* Evaluation test loss: %.2f' % score[0])
    plot_result(information, epochs, "Siamese network")
```

We have set the epochs to 20 repetitions and we will display the value for accuracy from 1 to 20.

Accuracy Level for Complex Siamese Model		
Test Dataset	Training accuracy	Testing accuracy
First pair dataset [0,1,2,4,5,9]	96.75%	96.18%
Second pair dataset [0 - 9]	96.84%	81.69%
Third pair dataset [3,7,8,6]	95.10%	65.32%

Table3. Accuracy Level for Complex Siamese Network

Comparing with the results of the previous module, the values of the training accuracy are decreased, and the values of the testing accuracy are increased in the complex Siamese model.

For the First pair dataset with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"] we had a high accuracy in the training set: 96.75% and in the test set: 96.18%. Comparing with the previous module, this result is similar, However, the difference is the value of training accuracy and testing accuracy is closer.

For the Second pair dataset with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot”] UNION ["dress", "sneaker", "bag", "shirt"] the accuracy in the training set is high: 96.84%. The accuracy for the test set is slightly lower with 81.69%. Due to the combination of the datasets involved in this experiment. However, comparing with the previous module, the testing accuracy has increased by 2.11%.

For the Third pair dataset with labels ["dress", "sneaker", "bag", "shirt"] we had a high accuracy in the training set: 95.10% but significantly low accuracy with the test set: 65.32%. This is because we have use as the test set the unseen data. Comparing with the previous module, the testing accuracy has increased by 3.89%.

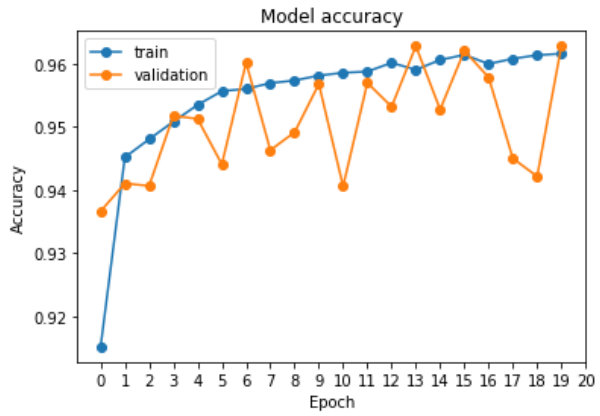

Figure 4. Model Accuracy

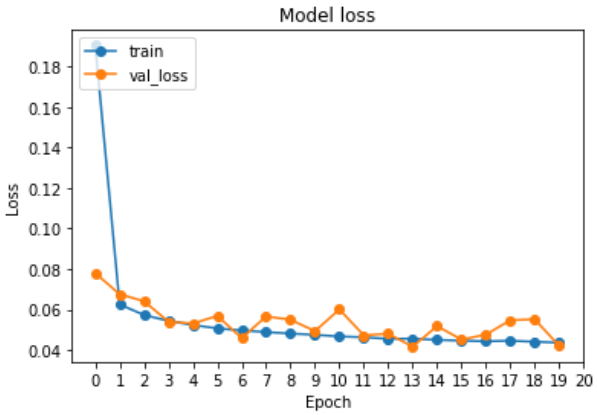

Figure 5. Model Loss

According with the Model Accuracy, we can infer that even though the ,model is not perfect , still the validation dataset is trying to make a good fit over the training dataset. Which means that his model is learning more than the base model.

According the plot for the Model Loss, we can observe that the overfitting issue has been enhanced. Although the validation loss (Orange line) was going up in some positions, such as around epochs 5,10,17,18, the two lines were basically following the same trend throughout the 20 iterations. This could be the effect of the added dropout and regularizers layers.

3.3 Testing Loss Function

For testing the loss function, we simply to adjust the margin value from the m = 1 to m = 5, in order to check the different between both accuracy results.

Complex Siamese Model Accuracy Level (margin = 1)		
Test Dataset	Training accuracy	Testing accuracy
First pair dataset [0,1,2,4,5,9]	96.75%	96.18%
Second pair dataset [0 - 9]	96.84%	81.69%
Third pair dataset [3,7,8,6]	95.10%	65.32%

Table4. Results margin=1 for Complex Siamese Model

Complex Siamese Model Accuracy Level (margin = 5)		
Test Dataset	Training accuracy	Testing accuracy
First pair dataset [0,1,2,4,5,9]	82.58%	81.96%
Second pair dataset [0 - 9]	78.84%	67.78%
Third pair dataset [3,7,8,6]	79.32%	55.43%

Table5. Results margin=5 for Complex Siamese Model

According to the result of the testing, with the margin value increasing to 5, the training and testing accuracy are decreased.

4.0 Discussion

From the results obtained in these experiments we can conclude that for the Base Model , the plot displaying the Model Loss was not giving us good results. In fact, the more epochs we did we could see that the validation test was getting further apart from the train dataset which means that the loss value was growing. Conversely, when we execute the Complex Siamese model to generate the graphic for the Model loss we could observe that on every iteration (epoch) the values for the Model loss were improving from 0.08 to 0.04 decreasing the values for both the train and validation. Over time, the Siamese network using the Complex Siamese model will get better results due to the dropout and regularization techniques added.

The Dropout layer is one method that we used to avoid the overfitting in our Siamese Network. When the model is trained, the dropout layers deactivates some random nodes which will stimulate the network to learn using different nodes. As a consequence, the generalization for unseen data will slightly improve. This was proven when we generated the table of accuracy for both models (Table2 and Table3). For the base model the level of accuracy for the test2 (unseen data) was 61.43% whereas, for the Complex Siamese Model the level of accuracy for the test2 was 65.32%. Still , the Siamese network will have inaccurate results when new images are entered to the network.

The loss function was tested using different values for the margin and then measuring the accuracy of the model with each value. The tunning of the value for the margin =1 provided better results in accuracy than when we set the value for the margin to be equal to 5. It is understandable that different images (negative) must have values far away from 0 but the margin should not be so distant so that the accuracy of the model does not decline.

In our Complex Siamese Model, we use dropout layers along with optimizers such as RMSprop. These kind of optimization algorithms will improve the generalization as well. Some algorithms will work better than others depending on the applications. In this assessment, we evaluated the model using Adadelta along with the model as well. However, the results were not satisfactory compared with RMSprop. For this reason, we decided to run the models using RMSprop as our optimizer.