Assignment 3

Problem 1

Preprocessing

In the preprocessing steps, firstly I load the cifra-10 dataset from keras, and select 20% of the original training set as a new training dadaset. Then I split the original testing data into half for validation and half for test. So finally there are 10000 training samples, 5000 validation samples and 5000 test samples. Then I rescale the original pixel values from 0 to 255, to the range from 0 to 1. Then, since our aim is to solve the multiclass calssificaiton with 10 possible values of labels, I applied one-hot encoding to the labels and encoded them to vectors.

Output layers and loss function

The output layer is a fully connected layer with 10 nerouns, because there are 10 classes in total. Here in this layer I use softmax as the activation function so that we will get a probability of each class whose sum is bound to be one, which is usually suitable for multi-classification tasks, . For loss function, I use 'categorical_crossentropy', which measures the performance of a classification model whose output is a probability value between 0 and 1.

MLP with different layers and neurons

In this part, I change the number of layers and the number of neurons per layer in the MLP, and run each network for 5 epochs to see the result. Firstly I use a 2 hidden layers network with 512 neurons per layer as given. And then adjust number of layers with fixed neurons to find the effect of number of layers, which is shown in the table 1.Then I fixed the hidden layer number

to 2, and change the number of neurons in each hidden layer, the result is shown in table 2.

Number of hidden Layer	Train accuracy	Validation accuracy
1	0.3792	0.3428
2	0.3614	0.3534
3	0.3257	0.3124
4	0.2851	0.2844
5	0.1954	0.1918

Table 1: Train and Validation accuracy with different number of layers

Number of neurons	Train accuracy	Validation accuracy
2	0.1749	0.1692
32	0.3113	0.3122
128	0.3484	0.3412
512	0.3585	0.3464
1024	0.3588	0.3576

Table 2: Train and Validation accuracy with different number neurons

From the tables awe can find: for the number of hidden layers,when the number increase, the accuracy decrease at the same time, so too many layers may lead to bad performance of our network. Since a too deep network will easily cause the gradient disappear and the model is difficult to train. While for the same layer numbers, when increasing the number of neurons, it seems we get better accuracy, but it also raises the computational time as well.

(layers,neurons)	Train accuracy	Validation accuracy
(2,512)	0.3615	0.3378
(3,256)	0.3555	0.3626

Table 3: Train and Validation accuracy with different layers and neurons

I also find that if layer numbers increases, we may need less number of neurons to get better result. For example, in table 3, when we compare a 2-hidden layers network with 512 neurons per layer and a 3-hidden layers network with 256 neurons per layer, they got similar accuracy.

MLP vs CNN

From table 4 we can find the accuracy of these three networks after training 5 epochs.

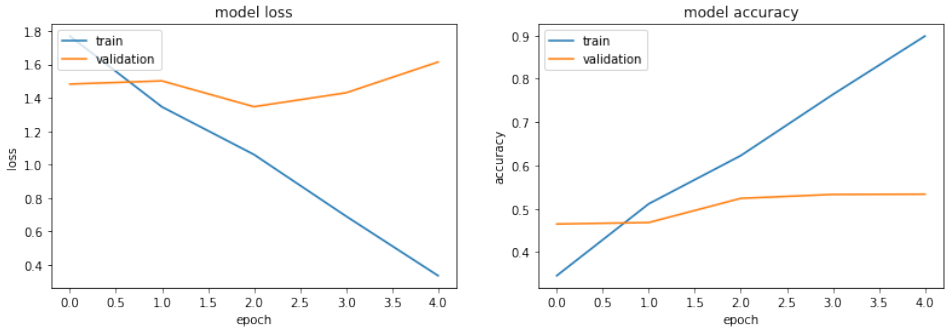
Network	Train accuracy	Test accuracy	Validation accuracy
MPL	0.3577	0.3610	0.3598
CNN1	0.8984	0.5347	0.5336
CNN2	0.5767	0.5666	0.5602

Table 4: Accuracy of different Networks

Among three models, MLP has the lowest accuracy compared with two CNN models. For this kind of multi-class image classification problem, MLP seems to be a rather simple model and it is difficult to get a high accuracy, however CNN is more suitble for such tasks with more complicated image classification. Since in MLP model, all of the layers are fully-connected, when taking images as input, with large number of pixels, there will be huge amount of parameters, what's more, MLP takes vector as input and may cause lost of dimensional information. While CNN takes tensor as input so that it can understand the relation between nearby pixels of image, which lead to better performance.

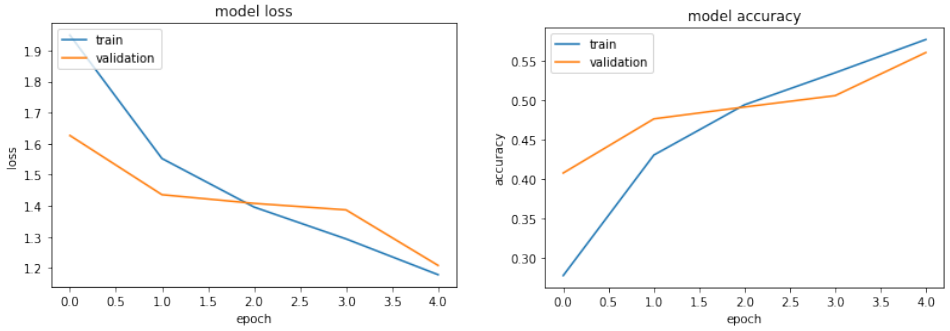
Comparision of two CNNs

Train and Val curves



(a) (b)

Figure 1: CNN1 loss and accuracy



(a) (b)

Figure 2: CNN2 loss and accuracy

Computation time

For CNN1, it takes about 14s per epoch and 1 ms each step in the training process, while for CNN2, it only takes 4s per epoch and 400 *mu*s each step to train.

Architecture

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 30, 30, 64)	1792
conv2d_2 (Conv2D)	(None, 28, 28, 64)	36928
flatten_1 (Flatten)	(None, 50176)	0
dense_8 (Dense)	(None, 512)	25690624
dense_9 (Dense)	(None, 512)	262656
dense_10 (Dense)	(None, 10)	5130

Total params: 25,997,130
Trainable params: 25,997,130
Non-trainable params: 0

(a) CNN1

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 64)	1792
max_pooling2d_1 (MaxPooling2	(None, 15, 15, 64)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 64)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_11 (Dense)	(None, 512)	1180160
dropout_1 (Dropout)	(None, 512)	0
dense_12 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_13 (Dense)	(None, 10)	5130

Total params: 1,486,666
Trainable params: 1,486,666
Non-trainable params: 0

(b) CNN2

Figure 3: CNN1 and CNN2 architecture

Comments

From the comparison of above results, we can clearly see that CNN1 is overfitting, with rather high training accuracy up to 0.9 but val accuracy is only about 0.5, and the training loss keeps decreasing however val loss is not. CNN2 does not show the same result, for 5 epochs, there is an increasing tendency on both training and validation accuracy. And CNN2 also trains and tests quicking than CNN2. We can discuss this from the difference of the architectures of these two networks. Compared to CNN1, CNN2 has additional max pooling layer and dropout layer. The dropout layer randomly drops out nodes during training to avoid overfitting, and pooling layer help to reduce the size of feature matrix and also further reduce the problem of overfitting, additionally, it can reduce the computational cost and training time. If training for more epochs, it seems that in CNN1, the val accuracy will not increase, while in CNN2, both the training accuracy and val accuracy will keeps increasing until they converge at a certain value.

Recommendations to improve networks

After I ran CNN2 more epochs, I found it is still overfitting with rather high training accuracy but low validation accuracy. There are several methods to solve overfitting problems that we can try. For example, we can add more dropout layers, because in CNN2, dropout layers are only added after two dense layers. Or we can apply other regularization methods like weight decay. And we can also do the data augmentation, or use larger dataset, because in this assignment we only use 20% of the original data, which may not be enough for training.

Problem 2

Creating the data

To predict next day's open using the latest three days' open,high,low prices and volume, we need to create a new dataset which contains the features mentioned above. Firstly I load the original dadaset which contains date, close/last, Volume, Open, High, and Low. And I read the values except for the first row, use for loops to save the second column(close/last) as the targets, and save the other four values of latest certain days(here it is 3) as

the features, where each sample has 12 features, and save them together as the new dataset. Then I randomize the cdata and split it into 70% training and 30% testing and save them as .csv file into the data directory respectively so that later on it can be directly loaded from the directory.

Preprocessing

In the preprocessing step, I first split the features and targets from the loaded dataset, and applied MinMaxScaler to the features both on training set and testing set. The purpose of this step is to scale each feature to a given range. After min-max normalization, for every feature, the minimum values is transformed to 0, and the maximum value gets transformed to 1, and every other value is transformed into a decimal between 0 and 1. The targets are remain the real values without normalization.

Design steps

For the network design, I firstly tried simpleRNN layers, and finally I choose a Long Short-Term Memory network for this prediction task, which is a powerful time-series models and it allows to model both long-term and short-term data. LSTM is a special kind of RNN, and it is designed to avoid the long-term dependency problem. And for the unit numbers, I tried to adjust unit number of each layer from small to large, to achieve better results on testing set. To measure the performance, I choose mean square error as loss function and mean absolute error as evaluation metrics. MSE measures the squared average distance between the real data and the predicted data.MAE measures the absolute average distance between the real data and the predicted data. Since the features are normalized while the targets are not, MAE is more accurate to measure whether the predicted value is close to the true value.

Network Architecture

My final network has 2 LSTM layers and the final layer is a fully connected layer. The first LSTM layer has 50 units and the second LSTM layer has 100 units, because I find with more units, it takes less epochs to converge with good prediction performance. I set the number of epochs to 600, because I find basically the network converged at this point. The loss function I choose

for training is mean square error(MSE). And the batch size is set as default value 32, and define the training algorithm as "adam" optimizer, which is a popular version of gradient descent because it automatically tunes itself and gives good results.

Output of training loop

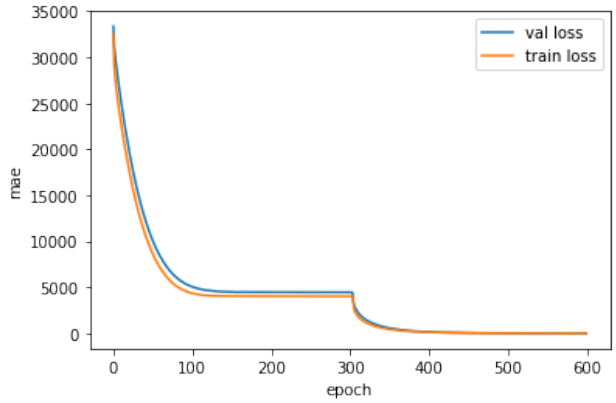


Figure 4: Testing set prediction

Comments

From figure 4 we can see the output of training loop, with the increase of training epochs, the loss decreases gradually. In the beginning, it decreases very fast, however, after a few epochs, it converges to about 5000, and then begins to decrease after that, finally it converged to training loss about 25, validation loss about 20. Training MAE is about 3.2, val MAE is about 3.5. This is possibly caused by the effect of Adam optimizer. Adam update the learning rate, which is different to classical stochastic gradient descent that maintains a single learning rate and the learning rate does not change during training.

Output of testing

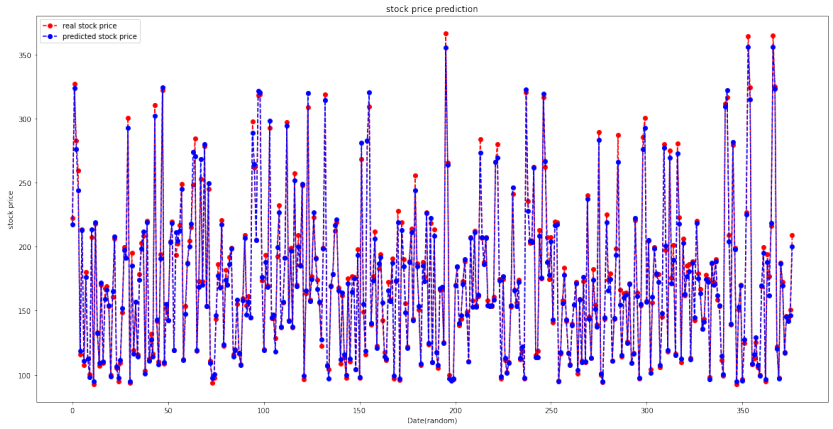


Figure 5: Testing set prediction

Comments

I loaded the saved model after training to give the prediction. Figure 5 shows the output of testing. It gives the prediction of the testing set. The MSE loss on testing dataset is about 22, and the MAE is 3.15. From the figure we can see most predictions are similar to the true value, but there are also a few predictions are not close to the testing targets.

Use more days for features

To use more days as features to train the network, I change the variable "days" in the train python file. With a few more days for features, the performance of the model improved to some extent. However, it is not always the case that more features leads to better prediction results. Although compared to simpleRNN, LSTM is more capable for long-term memory. Actually, if there are the more features, the network structure need to be adjusted, however here the network is the same, so to achieve better performance, there are more work to be done, not only the increase of features.

Problem 3

Preprocessing

Firstly I load the ac1Imdb from the data directory, since the texts of movie reviews are stored as .txt files in /pos and /neg directories, I read the texts from the path and return to the raw texts and their labels. Then we need to clean the data, since there are punctuation and special characters that are not useful for training, so I use regular expressions to remove them. Next we need to vetcorize the text into a list of integers. Then I fit a keras text tokenizer to turn the cleaned text into a sequence of integers, where each integer is the index of a token in the dictionary. The original reviews are of different lengths, I found that the max length of all reviews is 2451, and the average length is about 229. So I use pad sequence to fix the inputs to the same length, since the network need the inputs with the same length. So that sequence that are shorter than max length we set will be padded with zeros, and sequence longer than that length will be cut. After doing all the preprocessing steps, the data is ready to be feed in the neural network.

Network Design

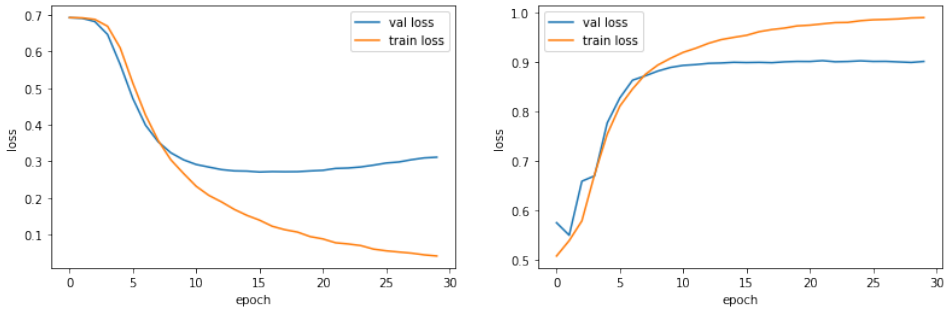
Figure 6 shows the neural network structure. The network is basically a sequential CNN model. Since for sentiment analysis tasks, CNN's ability to extract local features from text is really useful. First of all, there is an embedding layer, which transforms each integer into the ith line of the embedding weights matrix, then the convolution layer with 16 filters and kerneal size is 2 with relu activation function, and a global average pooling layer is added, and there are two fully connected layers, one is with 32 nodes and relu activation function and the final layer is with one output node and sigmoid activation function.I also added Dropout layers after operation layers to avoid overfitting.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, None, 16)	2284368
dropout_3 (Dropout)	(None, None, 16)	0
conv1d_1 (Conv1D)	(None, None, 16)	528
global_average_pooling1d_1 (	(None, 16)	0
dropout_4 (Dropout)	(None, 16)	0
dense_2 (Dense)	(None, 32)	544
dropout_5 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 1)	33

Total params: 2,285,473
Trainable params: 2,285,473
Non-trainable params: 0

Figure 6: network structure

Training and testing accuracies



(a) train and val loss (b) train and val accuracy

Figure 7: loss and accuracy

As is shown in figure 7,after training for 20 epochs, the training loss reaches to 0.0920 and the validation reaches to 0.2698. And the training accuracy is 0.9737, and the validation accuracy is 0.9031. I actually trained for 30 epochs but here I tried early stopping since we can see that the training is a

little bit overfitting, so the final model is trained after only 20 epochs. When evaluating on the test dataset, the test accuracy is 0.8853.

Comments

As we can see from the training and testing accuracies, we can easily find the fact that the training accuracy goes to increase over 95%, however, both the validation accuracy and testing accuracy seems not going to increase over 90% and this model has some limits. To try to find a better approach, we may consider about training with more samples, processing the data to be better to train, focusing on adjusting the network and hyper-parameters, or maybe trying combine CNN together with RNN to see if it will get better results.