Cover page for answers.pdf

CSE512 Fall 2018 - Machine Learning - Homework 6

Your Name: Sriram Reddy Kalluri

Solar ID: 111878857

NetID email address: skalluri@cs.stonybrook.edu sriram.kalluri@stonybrook.edu

Names of people whom you discussed the homework with:

1). $\tilde{C} = \frac{1}{n}\tilde{X}\tilde{X}^T$.

We know that. $\tilde{X} = (I - v_1 v_1^T)X$.

$\tilde{X}^T = X^T (I - v_1 v_1^T)^T$ Since $I - v_1 v_1^T$ is symmetric

$= X^T (I - v_1 v_1^T)$ $\Rightarrow (I - v_1 v_1^T)^T = (I - v_1 v_1^T)$

$$\tilde{C} = \frac{1}{n}(I - v_1 v_1^T) X X^T (I - v_1 v_1^T).$$

$$= \frac{1}{n}\left(XX^T - v_1 v_1^T XX^T - XX^T v_1 v_1^T + v_1 v_1^T XX^T v_1 v_1^T\right) \text{ — } ①$$

Given that; $XX^T v_1 = n\lambda v_1$.

Applying transpose on both sides.

$$(XX^T v_1)^T = (n\lambda_1 v_1)^T$$

$$v_1^T (XX^T)^T = n\lambda_1 v_1^T$$

$$v_1^T (X^T)^T X^T = n\lambda_1 v_1^T$$

$$v_1^T (XX^T) = n\lambda_1 v_1^T \text{ — } ②$$

Substituting ② in ① and $v_1^T v_1 = 1$.

$$\tilde{C} = \frac{1}{n}\left(XX^T - v_1 v_1^T XX^T - XX^T v_1 v_1^T + v_1 v_1^T XX^T v_1 v_1^T\right).$$

$$= \frac{1}{n}\left(XX^T - \cancel{n\lambda_1 v_1 v_1^T} - n\lambda_1 v_1 v_1^T + \cancel{v_1 (n\lambda_1 v_1^T)} (1)\right)$$

$$= \frac{1}{n}XX^T - \lambda_1 v_1 v_1^T$$

2). To prove! $j \neq 1$, if v_j is a principal eigen vector of C with corresponding eigen value λ_j (i.e $Cv_j = \lambda_j v_j$).,

then v_j is also a principal eigenvector of $\tilde{C}$ with the same eigen value λ_j

Given $\tilde{C} = \frac{1}{n} \tilde{X} \tilde{X}^T$

From 1, $\tilde{C} = \frac{1}{n} XX^T - \lambda_1 v_1 v_1^T$.

$$\tilde{C} v_j = \left[\frac{1}{n} XX^T - \lambda_1 v_1 v_1^T\right] v_j$$

We know that $XX^T = n\lambda_j$ and $v_i^T v_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$

$$= \frac{1}{n}\left[(n\lambda_j v_j) - \lambda_1 v_1 (0)\right]$$

$$\tilde{C} v_j = \lambda_j v_j \quad - ③ \quad \text{for } j \neq 1.$$

Hence proved that v_j is also a principal eigen vector of $\tilde{C}$ with same eigen value λ_j for $j \neq 1$.

3). Let u be the first principal eigenvector of $\tilde{C}$.

To prove $u = v_2$.

Eigen values for the eigen vectors will be in decreasing order i.e. $\lambda_1 > \lambda_2 \dots > \lambda_K$ for $v_1, v_2 \dots v_K$.

From ② $\tilde{C} v_1 = \lambda_1 v_1 - \lambda_1 v_1 v_1^T v_1$.

we know $v_i^T v_j = 1$ if $i = j$

$\Rightarrow v_1^T v_1 = 1$.

Scanned by CamScanner

$\Rightarrow$. $\tilde{C} v_1 = 0$

Therefore, the eigen value of v_1 for $\tilde{C}$ is 0.

which also satisfies.

For $v_2, v_3 \dots v_k$ is $\lambda_2 \lambda_3 \dots \lambda_k$

$$\lambda_2 > \lambda_3 \dots > \lambda_k$$

Thus, I can say that λ_2 is the largest eigen value of $\tilde{C}$.

This makes v_2 the first principle eigen vector of $\tilde{C}$.

Hence proved. $u = v_2$.

4). $[\lambda, u] = f(C)$.

The pseudocode using python format is as given below:

```
def  k_eigenvec (C, k, f) :
        λ = [ ]
        v = [ ]
        for j in range(1, k+1):
            λj, vj = f(C).
            C = C - λj vj vj^T.
            λ = λ + [λj]       } appending λ and
            v = v + [vj].        v lists.

    return v, λ.
```

v is the list of first k eigen vectors.

λ is the list of first k eigen values.

I have properly filled the 8 ‘ToDos’ spots in the note book.
Please find them as it is easy to evaluate.

1)

```
nn.Conv2d(3, 8, kernel_size = 7, stride = 1),
nn.ReLU(inplace = True),
nn.MaxPool2d(2, stride = 2),
nn.Conv2d(8, 16, kernel_size = 7, stride = 1),
nn.ReLU(inplace = True),
nn.MaxPool2d(2, stride = 2),
Flatten(),
nn.ReLU(inplace = True),
nn.Linear(1936, 10)
```

2)

```
optimizer=optim.RMSprop(fixed_model.parameters(), lr=1e-4)
loss_fn=nn.CrossEntropyLoss().type(dtype)
```

3)

I have tried the following things:

1)Training the network using the CNN layer filter from 7 and varying to 5 and 3 by increasing the CNN layers.
I have observed the training the network is faster for using filter size 3 as the number of parameters in the network is reduced drastically.

2)I have trained the network with and without the batch normalization. I have seen little improvement if I'm using the network with Batch Normalization.

3)I used data augmentation techniques by adding salt and pepper noise and flipping the pictures. I have uploaded the script for data augmentation in my github. Please find the link below:
https://github.com/sri123098/Fruit-Image-Classification-CNN-SVM/blob/master/data_augmentation.py

It had improved the performance but only 1%.

4)I have played with different Optimization techniques and properly tuned the learning rate. RMSprop and Adam gave decent results.

5)Similarly,I have seen the benefits of drop out as it is helping to reduce the overfit problem because of the training for more epochs. Moreover, I'm fascianted behind the idea of dropout as it came from reproduction in biology.
Using the layers, conv2d, relu, maxpool,Batchnorm and Affine layers, I have constructed seven networks. Out of which, I got the cross validation accuracy i.e best for the model which is given below.

CONV2D -> RELU -> CONV2D -> RELU -> MAXPOOL2D -> BATCH NORM -> DROP OUT -> CONV2D -> RELU -> MAXPOOL2D ->BATCHNORM -> DROPOUT ->CONV2D -> RELU -> MAX POOL2D ->
BATCHNORM 2D ->FLATTEN -> AFFINE -> RELU -> DROP OUT -> AFFINE -> RELU -> SOFTMAX

Affine transformation is nothing but the linear layer with dense connections similar to that of tensorflow.

Exact description is given below:

1. 3x3 Convolutional Layer with 64 filters and stride of 1
2. ReLU Activation Layer
3. 3x3 Convolutional Layer with 64 filters and stride of 1
4. ReLU Activation Layer
5. 2x2 Max Pooling layer with a stride of 2
6. Batch Normalization Layer
7. Drop out layer of 0.1
8. 3x3 Convolutional Layer with 128 filters and stride of 1
9. ReLU Activation Layer
10. 2x2 Max Pooling layer with a stride of 2
11. Batch Normalization Layer
12. Drop out layer of 0.2
13. 5x5 Convolutional Layer with 256 filters and stride of 1
14. ReLU Activation Layer
15. 2x2 Max Pooling layer with a stride of 2
16. Batch Normalization Layer
17. Flatten
18. Affine layer
19. ReLU Activation Layer
20. Drop out layer of 0.2
21. Affine layer
22. ReLU Activation Layer
23. Softmax layer

The above network is some what similar to VGG Net 19 architecture.

With this, I got cross validation accuracy of 71.45%.

Cross validation accuracy on the validation test is:

```
In [71]: check_accuracy(model, image_dataloader_val)

Got 4780 / 6690 correct (71.45)

/Users/sriramreddy/Downloads/ML/hw6_data/.env/lib/python3.6/site-packages/torch/nn/modules/container.py:91: UserWarni
ng: Implicit dimension choice for log_softmax has been deprecated. Change the call to include dim=X as an argument.
  input = module(input)
```

Please find the Kaggle score below:

19	▾ 4	**Sriram Reddy Kalluri**	0.70010	9	now

Your Best Entry ↑

You advanced 15 places on the leaderboard!

Your submission scored 0.70010, which is an improvement of your previous score of 0.67910. Great job! Tweet this!

VIDEO 3D

I have used three different architectures. Out of them, the following two architectures gave decent results.

Please find the best architecture below:

1. Conv3d with 32 channels kernel_size=(1,5,5), stride=(1,1,1)
2. Re LU Activation Layer
3. Max pooling 3d with filter size 1,2,2 and stride 1,2,2
4. Batch Normalization Layer
5. Drop out layer with 0.1 percentage
6. Conv3d with 24 channels kernel_size=(1,5,5), stride=(1,1,1)
7. Re LU Activation Layer
8. Max pooling 3d with filter size 1,2,2 and stride 1,2,2
9. Batch Normalization Layer
10. Drop out layer with 0.2 percentage
11. Conv3d with 20 channels kernel_size=(1,4,4), stride=(1,1,1)
12. Re LU Activation Layer
13. Max pooling 3d with filter size 1,2,2 and stride 1,2,2
14. Batch Normalization Layer
15. Flatten 3d
16. Affine layer connecting 1500 to 500
17. Affine layer connecting 500 to 100
18. Affine layer connecting 100 to 50
19. Affine layer connecting 50 to 10
20. Soft max layer

Please find the Kaggle score below: