Deep Learning 2019

Assignment # 3

Report

Topic: Convolutional Neural Networks

➢Task 1:

Use the Filter Banks to get the features for the Neuron with Sigmoid and cross entropy loss.

Steps of training of Convolution:

1. load_dataset function is used to get train and test input from the given data. Each image is of size 28x28. Training data is saved in tuples of size (60000,784) after flattening the images. Labels are saved in (60000, 1). Testing data is saved in tuples of size (10000,784) after flattening the images. Labels are saved in (10000, 1).
2. Data is normalized by subtracting mean and dividing by standard deviation.
3. Convolutional is performed on training and testing images by using the given filters. Each filter is of size 17x17. 17x17 patches are taken from the image, multiplied with filter element-wise and summation is taken. Same procedure is done for every filter. Output volume is 12,12,36 as there are 36 feature.
4. Then max-pooling is performed with filter_size = 2 and stride=2. Resultant output volume is 6,6,36.
5. Then feature volume is flattened to get vector of 1296 for each image.
6. This feature volume is then given to the fully-connected network implemented in assignment 2, task 3. 256,128,10 is the number of neurons in the FC layers. Steps of training of FC layers are given below are given below:

Steps of training of fully-connected layers:

Note: I've performed all experiments using *SGD*.

Feedforward:

1. Randomly initialize weights and biases.

   $w^2 = w_h$, $w^3 = w_o$ (h for hidden and o for output)
   $b^2 = b_h$, $b^3 = b_o$ (h for hidden and o for output)

2. Compute z matrices by:

$$z_h = {w_h}^T x + b_h$$
$$z_o = {w_o}^T x + b_o$$

3. Apply activation function on z to get a_h and a_o.

   Sigmoid:

$$a = Sigm(z) = \frac{1}{1 + e^{-z}}$$

   tanh:

$$a = tanh(z) = \frac{2}{1+e^{-2z}} - 1$$

Relu:

$$a = \max(0, z)$$

Back-propagate:

1. Calculate the gradients for back-propagation of error. We need to find:

$$\frac{\partial \mathrm{L}}{\partial \mathrm{w_o}}, \frac{\partial \mathrm{L}}{\partial \mathrm{w_h}}, \frac{\partial \mathrm{L}}{\partial \mathrm{b_o}}, \frac{\partial \mathrm{L}}{\partial \mathrm{b_h}}$$

where L is binary cross-entropy loss, w is set of weights and b is bias.
Always using sigmoid at the output node (to get probabilities), we get:

Output layer:

$$dz_o = \frac{\partial \mathrm{L}}{\partial \mathrm{z_o}} = (a_o - y\,) = \ \delta_o$$

$$\frac{\partial \mathrm{L}}{\partial \mathrm{w_o}} = \ \delta_o * a \quad \& \quad \frac{\partial \mathrm{L}}{\partial \mathrm{b_o}} = \delta_o$$

Hidden Layer:

$$dz_h = \frac{\partial \mathrm{L}}{\partial \mathrm{z_h}} = \ \delta_o * {w_o}^T * \ a_h = \ \delta_h$$

Note: * is not dot product. It is just element-wise multiplication of two vectors.

$$\frac{\partial \mathrm{L}}{\partial \mathrm{w_o}} = \ \delta_h \mathbf{x}\, x$$

Note: $\mathbf{x}$ is outer product and x is the input vector.

$$\frac{\partial \mathrm{L}}{\partial \mathrm{b_o}} = \ \delta_h$$

Update Weights and Bias:

1. Update Weights:

$$w_o = w_o - alpha * \frac{\partial L}{\partial w_o}$$

$$w_h = w_h - alpha * \frac{\partial L}{\partial w_h}$$

2. Update Biases:

$$b_o = b_o - alpha * \frac{\partial L}{\partial b_o}$$

$$b_h = b_h - alpha * \frac{\partial L}{\partial b_h}$$

Alpha is the learning rate.

Steps of SGD:

1. For every data point:
   - Perform feedforward.
   - Perform back-propagation i.e. find gradients and back propagate the error, update the weights and bias simultaneously.
2. Compute Cross-entropy loss L:

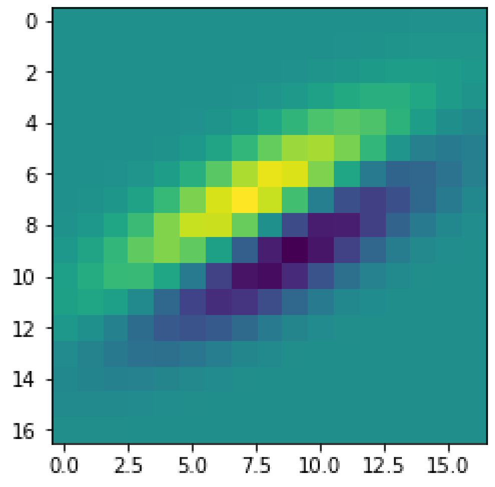
$$L = \frac{1}{n}\sum_{i=1}^{n}(-(y_i * \log(a_{oi}) + \ (1 - y_i) * \log(1 - a_{oi}))$$

   where y_i is the actual label and n is the total number of training examples.
3. Repeat the above steps till L converges.

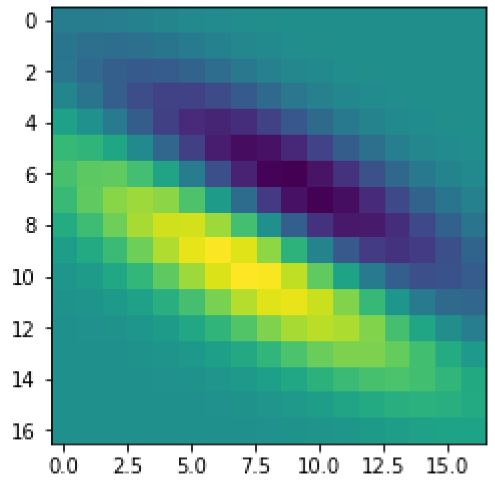
Results:

Viewing Filters:

```
plt.imshow(np.reshape(filters[:,:,5],(17, 17)))
plt.show()
```

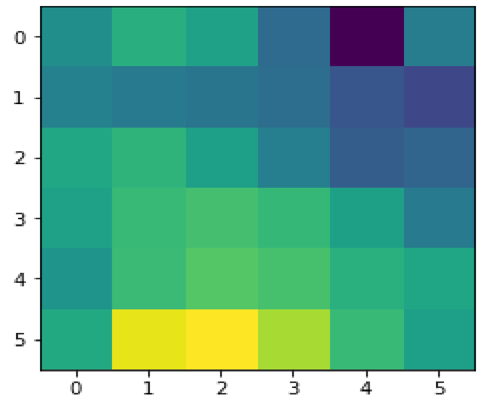


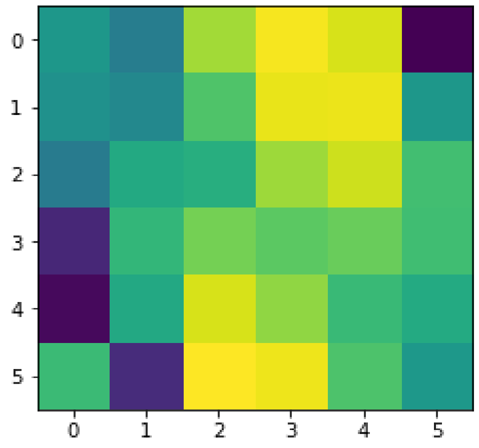
```
plt.imshow(np.reshape(filters[:,:,7],(17, 17)))
plt.show()
```

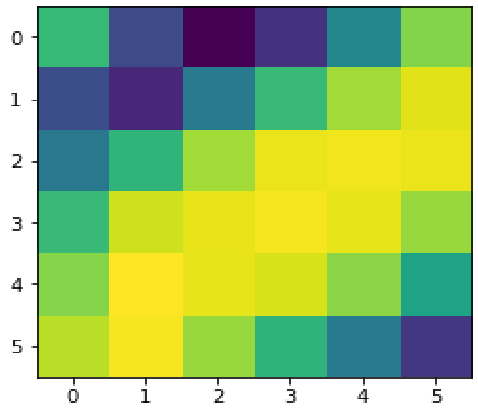


Viewing output of convolution:

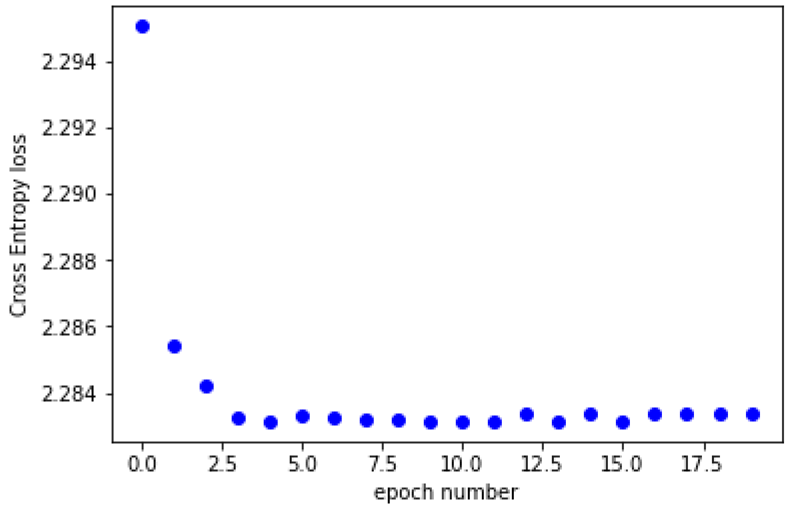
```
plt.imshow(image[:,:,35])
plt.show()
```

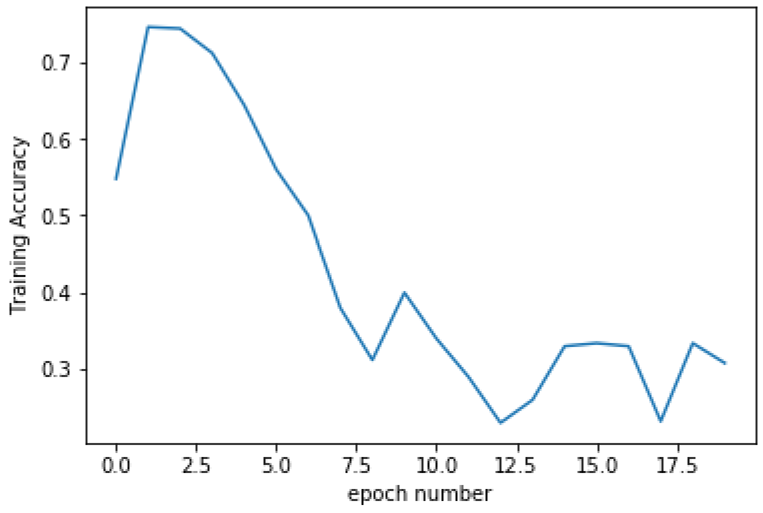






Loss & Accuracy Curves:





Note: I've used the same implementation of FC-network as used in the assignment 2 and it was working but while I was changing the network to add convolution to the network I changed something in the functions which I was unable to figure out later because I was short of time, so I implemented the convolution function out of the Neural_Network class and extracted all convolutional features at once and saved them and moved to the second task.

➢Task 2:

Implement Convolutional Neural Network.

Note: I've performed all experiments using *SGD* with momentum. Also I've performed experiments with subset of 50, 100, 200 and 500 images as I was unable to perform experiments with large number of images on my machine though I tried it quite a lot of time. But my network is overfitting on all the smaller subsets, accuracy was continuously increasing and loss decreasing which shows it is correct.

MNIST:

Implementation details:

FC-network is implemented as given in the task 1, while the convolution part and its back-propagation is performed according to the equations given in the assignment document.

Network Architecture:

1) 2 convolution layers, 1 pooling layer, 2 fc layers.
2) 8 filters of size 5x5 are used in both convolutional layers with stride=1 and padding=0.
3) Pooling is done with filter size= 2 and stride = 2. (Max-pooling and Average-pooling is implemented)
4) 2 fc-layers are used with 128 and 10 neurons respectively.

Filters, Weights and Biases:

1) Filters and weights are initialized from normal distribution,
2) Biases (there are total four bias vectors: 2 for convolutional layers and 2 for fc layers) are initialized with zeros.

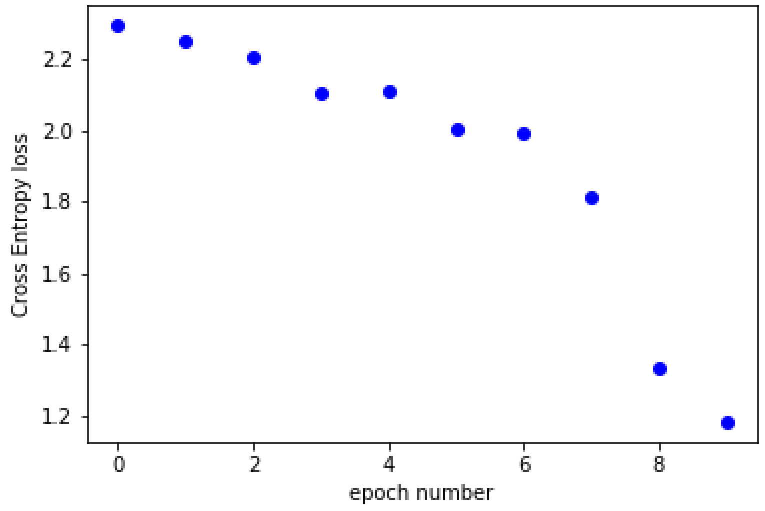
Feature Size:

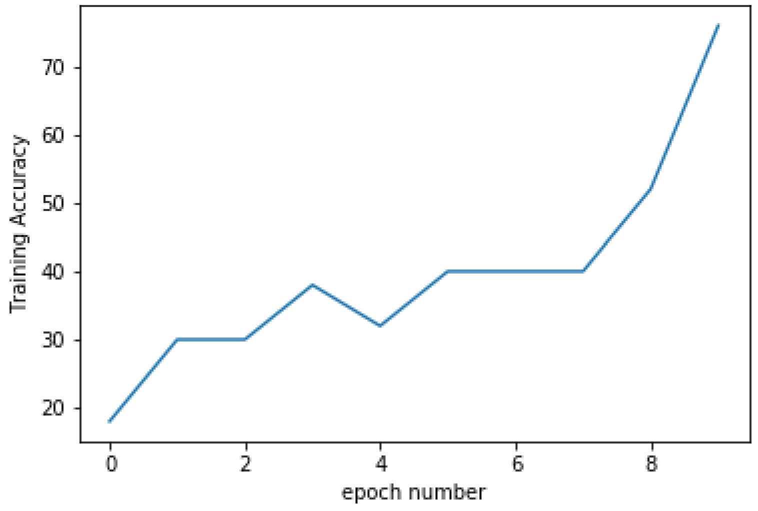
1) After first convolution layer each image of mnist (1*28*28) is changed to (8*24*24), after second convolution output volume is (8*20*20). After pooling, (8*10*10). After flattening (800).

Results:

```
[[58  0  0  0  0  0  0  0  0  0]
 [ 0 60  1  0  0  0  0  0  2  0]
 [ 0  0 40  1  0  0  0  1  0  0]
 [ 0  0  1 46  0  1  0  1  1  0]
 [ 0  1  1  0 43  2  0  0  0  0]
 [ 1  0  1  1  0 35  0  0  1  0]
 [ 0  0  1  0  0  0 47  0  0  0]
 [ 1  0  1  2  0  1  0 52  0  0]
 [ 1  0  2  0  0  2  0  0 41  0]
 [ 2  0  0  0  4  5  0  2  0 37]]
At epoch  1 loss is:  1.0685072381910588 accuracy is:  91.8
```

Accuracy and loss curves after 10 epochs with learning rate = 0.9.





I've performed lots of experiments on MNIST dataset with different learning rates and number filters. These experiments can be found in the notebook.

CIFAR-10:

Implementation details:
FC-network is implemented as given in the task 1, while the convolution part and its back-propagation is performed according to the equations given in the assignment document.

Network Architecture:

5) 2 convolution layers, 1 pooling layer, 2 fc layers.
6) 10 filters of size 5x5 are used in both convolutional layers with stride=1 and padding=0.
7) Pooling is done with filter size= 2 and stride = 2. (Max-pooling and Average-pooling is implemented)
8) 2 fc-layers are used with 128 and 10 neurons respectively.

Filters, Weights and Biases:

3) Filters and weights are initialized from normal distribution,
4) Biases (there are total four bias vectors: 2 for convolutional layers and 2 for fc layers) are initialized with zeros.

Feature Size:

2) After first convolution layer each image of cifar -10(32*32*3) is changed to (28*28*10), after second convolution output volume is (24*24*10). After pooling, (12*12*10). After flattening (1440).

I couldnt perform multiple experiments on cifar-10 because of time limitation.

➢References:

https://www.researchgate.net/figure/Logistic-sigmoid-function-Maps-real-numbers-to-the-interval-between-0-and-1_fig6_234049070

https://www.youtube.com/watch?v=BvrWiL2fd0M

https://www.coursera.org/learn/machine-learning/lecture/1z9WW/backpropagation-algorithm

https://github.com/Alescontrela/Numpy-CNN