University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 10: Deep Neural Networks
October 2, 2018

Lecturer: Alexander Schwing Scribe: CHEN, HONG SHUO

1 Introduction

After finish the last lecture Multiclass Classification and Kernel Methods, we understand multi-class logistic regression and multi-class SVM, and also learn the kernel trick. First, we get the conclusions that although each of the different classifiers uses the different cost function, all of them actually lie in the same framework. Second, we know how to use kernels as non-linearing feature mappings. In section 2, this lecture notes would first briefly introduce deep nets and know how does it extend from the framework we get in the last lecture. In section 3, we would know forward and backward pass which is the way we define the structured functions and compute and also show some general structure and example of Deep Neural Networks. In section 4 and 5, we would thoroughly go through each of the deep net components and give a mathematical introduction. In section 6, we would show an example of Deep Neural Networks implemented by PyTorch.

2 Getting to know deep nets

Our current framework is

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in D} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + w^\top \psi(x^{(i)},\hat{y})}{\epsilon} - w^\top \psi(x^{(i)},y^{(i)})$$

However, there are still some possible issues and limitations. The linearity in the feature space $\psi(x^{(i)},\hat{y})$ may not reach the best accuracy. We could fix it using the kernels we mentioned in the last lecture. Still, the problem here is that we still learning a model linear in the parameters w. Regarding this, we replace $w^\top \psi(x,y)$ to a much more general function $F(w,x,y)$. The output of $w^\top \psi(x,y)$ and $F(w,x,y)$ are both just one-dimensional real number.

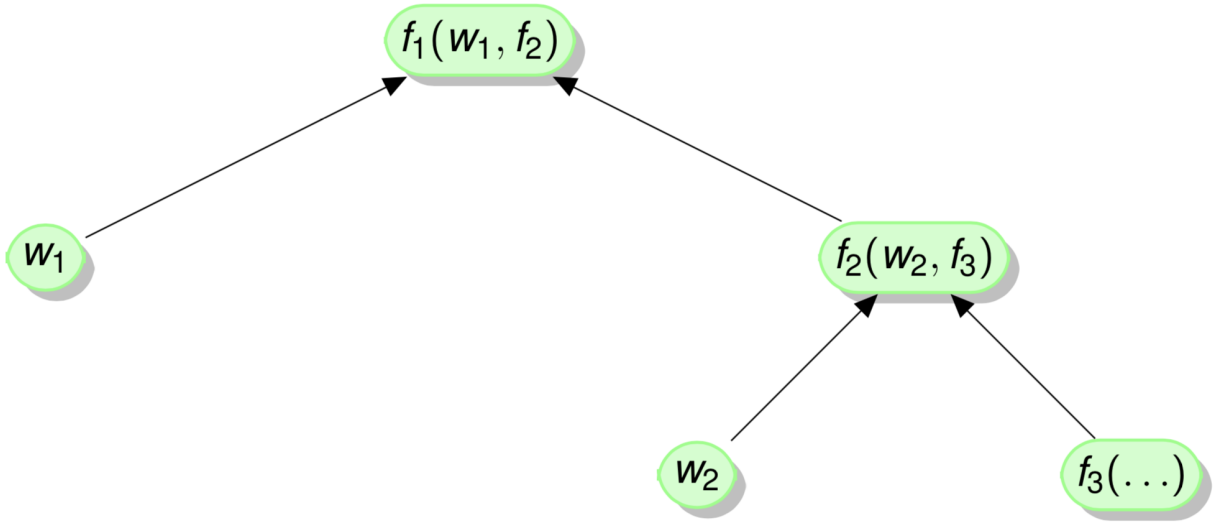
More general framework is

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in D} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + F(w,x^{(i)},\hat{y})}{\epsilon} - F(w,x^{(i)},y^{(i)})$$

where $F(w,x,y) \in \mathbb{R}$.

We can get to this framework by the technique we learn before, such as Logistic regression, Binary SVM, Multiclass regression, Multiclass SVM and most importantly Deep Learning which we want to elaborate in this lecture.

Figure 1: Computation graph: Nodes are weights, data, and functions



2.1 Deep learning

The first question is that there are infinite functions in this world. How do we choose the function as $F(w, x, y)$? The answer is that we could choose any function that could be differentiable and $F(w, x, y)$ could be composed of them. The reason that it has to be differentiable is that we train it and compute it in backpropagation which will mention in the next lecture.

Deep Neural Networks are called network because it is represented by composing many different functions. More generally, we could say that the model is associated with a directed acyclic graph or computation graph describing how the functions are composed together. For example, we might have n functions connected in a chain. Take the first three functions as f_1, f_2 and f_3.

$$F(w, x, y) = f_1(w_1, f_2(w_2, f_3(w_3, ...)))$$

The weight w here is no longer a K by D dimensional vector, where D is the length of the feature vector and K is the number of class. The weight w is a long vector, and the number of parameters of each w_n depends on f_n.

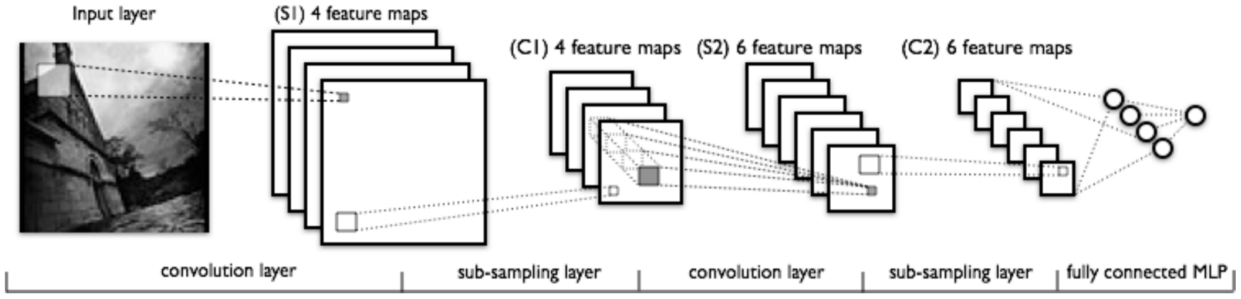
From figure 1, we can see the visualization of $F(w, x, y)$. These chain structures are the most commonly used structures of neural networks. f_1 is the last layer and also called the output layer, f_2 is the last second layer, and f_n is the first layer, and so on.

The overall length of the chain gives the depth of the model, and it is also the reason why we called it "deep learning".

$$F(w, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, ...)))$$

If we put y in the function, which is always put in the last few layers. The benefit is that we could get a much more fast approach to the outcome, while the disadvantage is that it will lose some classes.

Figure 2: LeNet



3 Understanding forward and backward pass

Regarding the last section, what are the individual functions and layers $f1$, $f2$, etc.? They might be Fully connected layers, which we always use it in the output layer, Convolutions, which we use it to extract the features, Rectified linear units (ReLU): max0,x, which is non-linear function and help us approach all the non-linear functions in the world, Maximum-/Average pooling, which help us retain the most important information in the image and downsize the image, Soft-max layer, which we use it to calculate the probability, and Dropout, which prevent the model from overfitting.

In this section, we would introduce several fundamental Deep Neural Networks, which use the components I mentioned above, and see how the information pass forward and backward. We would first introduce the most earlier work of deep nets LeNet, AlexNet, and some other deep network.

3.1 LeNet [2]

LeNet is proposed by Yan LeCun in 1994. It is one of the earliest Deep Neural Networks. From the figure 2 given in the lecture, LeNet is composed of the six layers, which the input layer is exclusive. Generally, convolution layer (S1, S2) and sub-sampling layer (C1 C2) are the heart of LeNet family of models. While exact details could be adjusted, some papers use seven layers which have two fully connected layers [2].

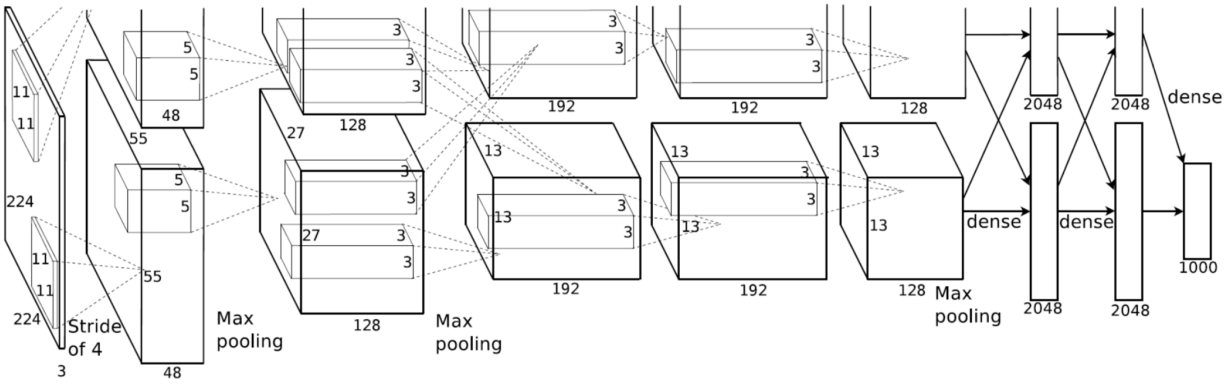
The most important concept we should get here is that we decrease the spatial resolution and increase the number of channels. Figure 2 shows that the channel increase from 3 (RGB) to 4 (S1) and furthermore to 6(S2), while the resolution of image keeps decreasing. The reason why we do that is that the output of the model just a real number. We need to gather all the information in this model.

3.2 AlexNet [1]

The structure of the AlexNet is quite similar to the LeNet. However, the different thing here is that it increases the depth of the networks and adds more convolution layer and pooling layer to the network. We could see another main difference from Figure 3 is that there are two paths. It is meant to express that the network is forwarded by two GPUs to accelerate the computation.

Take the first convolution layer as an example. The input of input layer is a 224 by 224 by 3 RGB images. We would do some pre-processing to make the image become 227 by 227 by 3. In the first convolution layer, we use 96 11 by 11 by 3 filters to do the convolution. We could show that the dimension of the output is 55 by 55 by 48. Because (227-11)/4 + 1 is equal to 55, which the stride here is 4. They separate 96 filters into two parts, which is the upper part and the lower part.

Figure 3: AlexNet



We do the max pooling after getting the outcome. The size of max-pooling is 3 by 3, and the stride is 2, so the output size after max-pooling is (55-3)/2+1 = 27, so the output of this layer is 27 by 27 by 96.

Follow the instruction above we could get the similar thing in the next few convolution layers. The last two layers before the output layer are the fully connected layer to learn all the features.

Why is the output 1000-dimensional?

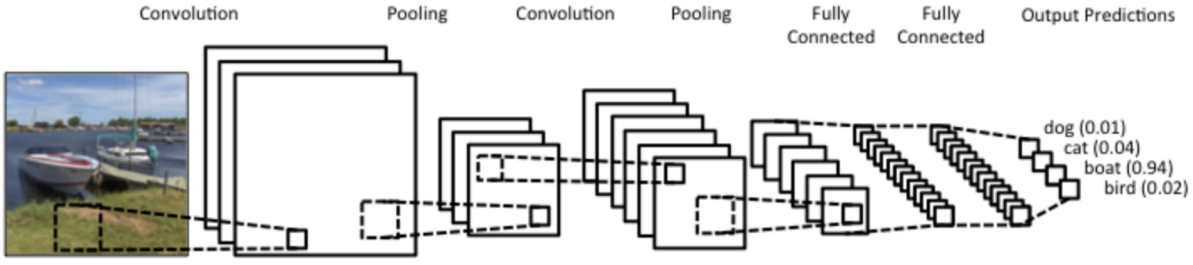
Because there are 1000 classes in this ImageNet dataset.

3.3 Another deep net

From the samples above, those nets are structurally simple that a layers output is used as input for the next layer. However, it is actually not required, we could take the input from other layer, not only from the last layer.

Figure 4 shows a general structure of Deep Neural Networks, and the output is the probability of each class.

Figure 4: General structure



4 Learning about deep net components

In this section, we will go through each functions and layers.

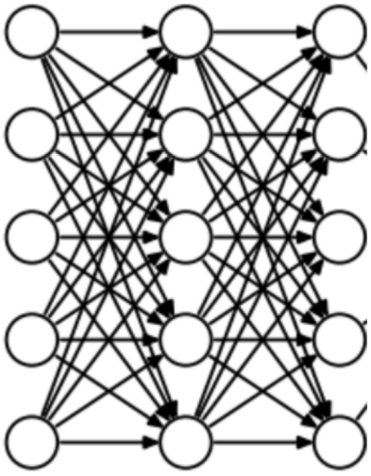
4.1 Fully connected layer

Fully connected layer have connections to all activations in the previous layer, which is like the classification of all the feature vector derived by the precious layers. We could write the formula as:

$$Wx + b$$

There are two trainable parameters, which are W matrix and b bias. The issue of the fully connected layer is that the number of the parameter is too large to train if we take the whole image as an input. Suppose the input is an image of size 256 256 and let the output of this layer have identical size. There are 2 256 256 weights in the layer. We could not train all the weighs. The alternative way is that we map the image to a low effective dimensional vector to reduce the computation and the number of weights we need to train. The way we share the weight is convolution. We would talk in the next subsection.

Figure 5: Fully connected layer



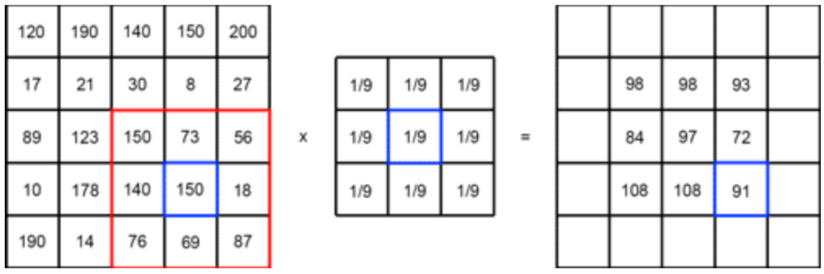
4.2 Convolutions

In order to get a shared weight, we use the convolution. The width, height, depth, and number all are the parameters. A different filter would extract a different feature from images. Figure 6 gives us a visualization of how to do elementary convolution works.

4.3 Maximum-/Average pooling

In order to get the most important information in the image, and downsize the image, we always do the maximum and average pooling with some stride.

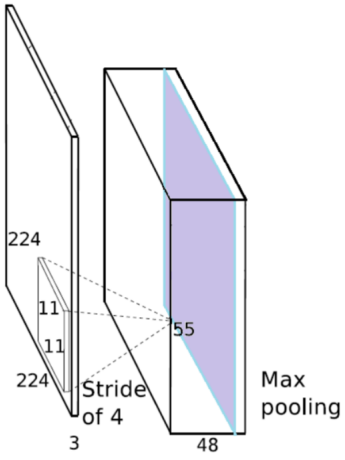
Figure 6: Convolution



The maximum pooling is calculated as the maximum value of the certain region, while the average pooling is calculated as the average value of the certain region.

Figure 7 gives us the example of max pooling it is a certain layer of the AlexNet. Although there is no weight to train, the step is essential that prevent the model from overfitting and help us approach the non-linear function much correctly.

Figure 7: Maximum-/Average pooling



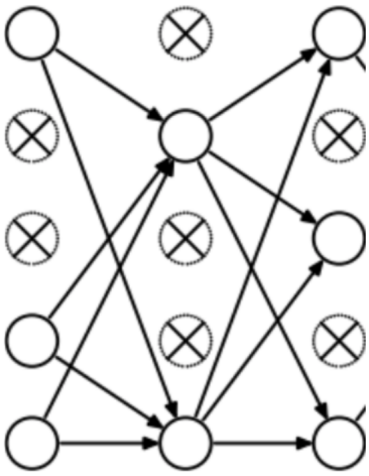
4.4 Soft-max layer

Soft-max layer is the layer that converts the score into probability. We always use a soft-max layer in the last few layers to calculate the probability of each class. There are also no trainable parameters.

The formula here is:

$$x \to \frac{\exp x_i}{\sum_j \exp x_j}$$

Figure 8: Dropout layer



4.5 Dropout layer

The most important thing of the dropout layer is that it used to prevent overfitting. Using dropout layer could somehow increase the accuracy. Figure 8 shows how the dropout layer randomly set activations to zero. There are also no trainable weighting here.

5 Deep net training

Our current general network:

$$\min_w \frac{C}{2}\|w\|_2^2 + \sum_{i\in D} \epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)},\hat{y}) + w^\top \psi(x^{(i)},\hat{y})}{\epsilon} - w^\top \psi(x^{(i)},y^{(i)})$$

Often also referred to as maximizing the regularized cross entropy

$$\max_w -\frac{C}{2}\|w\|_2^2 + \sum_{i\in D}\sum_{\hat{y}} p_{GT}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)}) with \begin{cases} p_{GT}^{(i)}(\hat{y}) = \delta\hat{y} = y^{(i)} \\ p(\hat{y}|x^{(i)}) \propto \exp F(w,x,\hat{y}) \end{cases}$$

where C is Weight decay(aka regularization constant)

$$\min_w \frac{C}{2}\|w\|_2^2 - \sum_{i\in D}\sum_{\hat{y}} p_{GT}^{(i)}(\hat{y}) \ln p(\hat{y}|x^{(i)})$$

We have two design choices. We design a composite function $F(w,x,y)$ or use an appropriate loss function, and then know what you are doing, i.e., know all the dimensions.

There are a lot of loss function we could use in different situation.

- CrossEntropyLoss

$$loss(x,class) = -log(exp(x[class])/(\sum_j exp(x[j]))) = -x[class] + log(\sum_j exp(x[j]))$$

- NLLLoss

$$loss(x, class) = -x[class]$$

- MSELoss

$$loss(x, y) = 1/n \sum_i |x_i - y_i|^2$$

- BCELoss

$$loss(o, t) = -1/n \sum_i i(t[i] * log(o[i]) + (1 - t[i]) * log(1 - o[i]))$$

- BCEWithLogitsLoss

$$loss(o, t) = -1/n \sum_i (t[i] * log(sigmoid(o[i])) + (1 - t[i]) * log(1 - sigmoid(o[i])))$$

- L1Loss
- KLDivLoss

We could see that why the form for the NLLLoss is $-x[class]$. Because it intended to be used in combination with 'LogSoftmax':

$$f(x) = \log \frac{\exp x_i}{\sum_j \exp x_j}$$

It will be much more numerical robustness (log-sum-exp trick).

$$\log \sum_j \exp x_j = c + \log \sum_j \exp (x_j - c)$$

6 PyTorch Example

This is an simple example for deep net which is implemented in Pytorch.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

From $self.conv1$, we could see that input channel is 1 dimension, output channel is 6 dimension and kernel size is 5.

7 Conclusion

In this lecture, we go through the deep nets, and all of its components, such as convolution layer, fully connected layer, max pooling layer and so on. The framework is similar to SVMs and logistic regression. We could train this framework by the loss function we define above.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.