University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 11: October 2

Lecturer: Alexander Schwing Scribe: Houyi Du

1 Introduction

This lecture mainly focus on the general framework applied at deep nets and one critical part of deep nets: backpropagation. We also discussed different loss functions and mainly focus on cross entropy loss.

2 Limitation of earlier framework

2.1 The limitation

Our earlier framework:

$$\min_{\mathbf{w}} \frac{C}{2} \|\boldsymbol{w}\|_2^2 + \sum_{i \in D} \left(\epsilon \ln \sum_{\hat{y}} \exp \frac{L(y^{(i)}, \hat{y}) + \boldsymbol{w}^T \psi(x^{(i)}, \hat{y})}{\epsilon} - \boldsymbol{w}^T \psi(x^{(i)}, y^{(i)}) \right)$$

recall that:

$$\psi(x^{(i)}, y^{(i)}) = \begin{bmatrix} \phi(x^{(i)}) \delta(y^{(i)} = 0) \\ ... \\ \phi(x^{(i)}) \delta(y^{(i)} = K - 1) \end{bmatrix}$$

So the feature space $\psi(x, y)$ is linear. In this way, we can only separate different classes linearly. Even if we use kernels (change $\psi(x, y)$ to $f(x, y)$), it is still a model linear in the parameters $\boldsymbol{w}$. Thus, We will replace $\boldsymbol{w}^T \psi(x, y)$ with a general function $F(\mathbf{w}, \mathbf{x}, y)$ to add non-linearity to the framework because F is a general function rather than just multiplication of matrix.

2.2 Result of replacement

After replacement, our current general framework look like this:

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

So how to inference (in other words, when we have feature vector $\boldsymbol{x}$, how to get corresponded label y)?

$$y^* = argmax F(\mathbf{w}, \mathbf{x}, y), y \in \{1, ..., K\}$$

where y^* is the intended label.

By using exhaustive search to find the best label because $\hat{y}$ is discrete(For the same reason we cannot use gradient decent). In fact, there are many other methods could be used to find the most suitable label y^* which will be introduced in later lectures.

3 Explanation of F

Now we replace $\boldsymbol{w}^T\psi(x,y)$ by function F, so what kind of composite functions we can use? First, let's figure out what is F.

3.1 What is F

- Any differentiable composite function$(f_1, f_2, f_3, ...)$

$$F(\mathbf{w}, x, y) = f_1(\mathbf{w}_1, y, f_2(\mathbf{w}_2, f_3(...f_n(\mathbf{w}_n, x)...))) \in \mathbb{R}$$

  Notice that we put y in the last function. We do this because y is just used to inference and we can avoid unnecessary computation.

- More generally, functions which can be represented by an acyclic graph(computation graph).

  A acyclic graph is shown below, nodes of the graph could be weights, data, and functions:

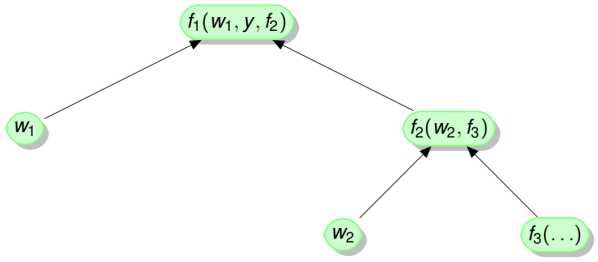

Figure 1: acyclic graph

3.2 Choices for composite functions

- Fully connected layers
- Convolutions
- Rectified linear units(ReLU): $max\{0, x\}$
- Maximum-/Average pooling
- Soft-max layer
- Dropout

3.3 Examples of composite function Architecture

3.3.1 LeNet

Figure 2 is just the combinations of convolutions and sub-sampling layers. During the forward pass applying convolutions (from left to right), we slide (more precisely, convolve) each filter across the width and height of the input volume(which is 1 because this input only have 1 channel or feature map) and compute dot products between the entries of the filter and the input at any position. So after applying the first convolution, we get four feature maps because we use four filters to slide over the input(only 1 layer). Sub-sampling is simply average pooling with learnable weights per feature map. A pooling or subsampling layer often immediately follows a convolution layer in CNN. Its role is to downsample the output of a convolution layer along both the spatial dimensions of height and width [3]. So it will not change the number of feature maps. In the second convolution layer, it used six filters and we will get six feature maps.

Finally, it applied a fully-connected layer whose neurons have full connections to all activations in the previous layer.

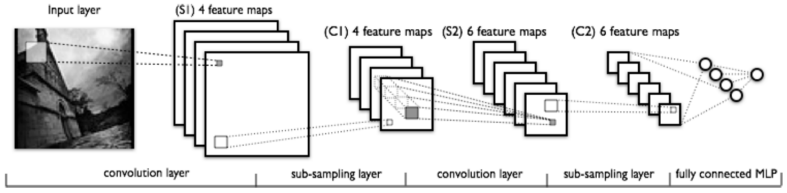

Figure 2: LeNet

3.3.2 AlexNet

The architecture of AlexNet is a combination of convolutions, max-pooling, and dense layer(fully connected layer). A much more detailed explanation of this Net work could be find here [1]. The final output is 1000-dimensional is because it is a soft-max layer with 1000-way which means there are 1000 classes/categories of this data set.

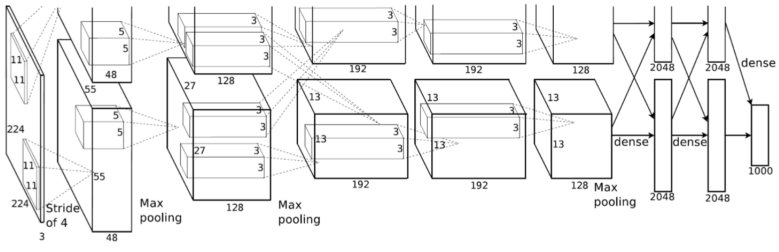

Figure 3: AlexNet

3.3.3 Another deep net

The architecture of this net is similar to LeNet. As the result of the image shows, probability is used as the criterion to classify.

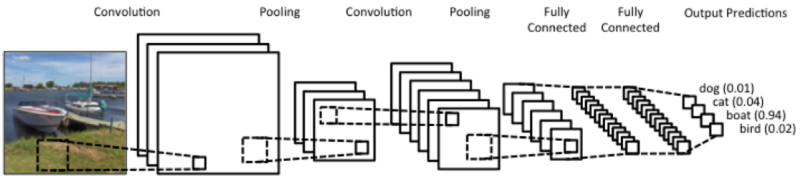

Figure 4: Another net

4 Applying the framework on deep net training

4.1 What should we optimize

In the scenario of Deep net learning, since the y is only used when we inference so we can get rid of the L. Besides, we set $\epsilon = 1$ for simplicity. Thus, the general framework change to:

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in D} \left(\ln \sum_{\hat{y}} \exp F(\mathbf{w}, x^{(i)}, \hat{y}) - F(\mathbf{w}, x^{(i)}, y^{(i)}) \right)$$

which is the same stuff as maximizing the regularized cross entropy(The explanation is at section 5):

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

where GT stands for ground truth and p_{GT} is just an indicator function which means when we classify it right, the $p_{GT}^{(i)}(\hat{y})$ will be 1, otherwise, it will be 0; And recap of soft-max, we have:

$$p(\hat{y}|x^{(i)}) = p(\hat{y}; x^{(i)}, \mathbf{w}) = \frac{\exp F(\mathbf{w}, x^{(i)}, \hat{y})}{\sum_{\tilde{y}} \exp F(\mathbf{w}, x^{(i)}, \tilde{y})} \text{ where, } \tilde{y} \text{ is all the all the classes.}$$

We will use the stochastic gradient descent with momentum to optimize the program. The gradient of program:

$$\min_{\mathbf{w}} \frac{C}{2} \|\mathbf{w}\|_2^2 + \sum_{i \in D} \left(\ln \sum_{\hat{y}} \exp F(\mathbf{w}, x^{(i)}, \hat{y}) - F(\mathbf{w}, x^{(i)}, y^{(i)}) \right)$$

is:

$$C\mathbf{w} + \sum_{i \in D} \sum_{\hat{y}} \left(p(\hat{y}|x^{(i)}) - \delta(\hat{y} = y^{(i)}) \right) \frac{\partial F(\mathbf{w}, x^{(i)}, \hat{y})}{\partial \mathbf{w}}$$

4.2 Algorithm to optimize

As we know the fomula of gradient decent, now we have the way to update the parameters $\mathbf{w}$. Below is the algorithm that we use to optimize:

- Forward pass to compute $F(\mathbf{w}, x^{(i)}, \hat{y})$ which means get the F via calculation from the bottom to the top of the acyclic graph.
- Compute $p(\hat{y}|x)$ via soft-max mentioned before.
- Compute $\frac{\partial F(\mathbf{w}, x^{(i)}, \hat{y})}{\partial \mathbf{w}}$ via Backpropagation.
- Update parameters $\mathbf{w}$

5 Explanation of Backpropagation

As the algorithm show, one thing we need to do is to get the derivative of F with respect to $\mathbf{w}$. As we get the $F(\mathbf{w}, x^{(i)}, \hat{y})$ through forward computation. We now use the backpropagation to compute the $\frac{\partial F(\mathbf{w}, x^{(i)}, \hat{y})}{\partial \mathbf{w}}$. The reason we use backpropagation is that we want to avoid repeated computation. Here is the example to show the process of backpropagation.

5.1 An example of backpropagation

Let

$$F(\mathbf{w}, x, y) = f_1(w_1, y, f_2(w_2, f_3(w_3, x))) \text{ with activations } \begin{cases} x_2 = f_3(w_3, x) \\ x_1 = f_2(w_2, x_2) \end{cases}$$

And now we have:

$$\frac{\partial F(\mathbf{w}, x^{(i)}, \hat{y})}{\partial \mathbf{w_3}} = \frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial x_2} \cdot \frac{\partial x_2}{\partial w_3} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_3} \cdot \frac{\partial f_3}{\partial w_3}$$

$$\frac{\partial F(\mathbf{w}, x^{(i)}, \hat{y})}{\partial \mathbf{w_2}} = \frac{\partial f_1}{\partial x_1} \cdot \frac{\partial x_1}{\partial w_2} = \frac{\partial f_1}{\partial f_2} \cdot \frac{\partial f_2}{\partial w_2}$$

As you may notice that, the above two equations have two common item: $\frac{\partial f_1}{\partial f_2}$. To avoid computation, we need to store the intermediate result of derivative during training process. An illustration in acyclic graph:

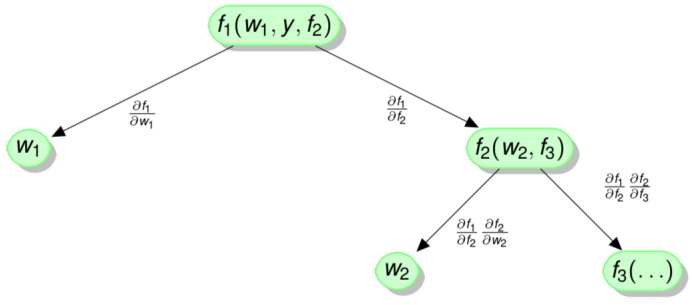

Figure 5: Backpropagation

We will store the derivative with respect to composite functions during the training process. For example, if we want to get the derivative with respect to $\mathbf{w_3}$, we can use the intermediate result ($\frac{\partial f_1}{\partial f_2}\frac{\partial f_2}{\partial f_3}$) stored in node $f_3(...)$ in the graph. As former mentioned, we will repeated use of chain rule for efficient computation of all gradients.

As the above example shows, we first calculate $\frac{\partial f_1}{\partial \mathbf{w_1}}$, then we calculate $\frac{\partial f_1}{\partial f_2}$ and store the result for further utilizing. When we calculate $\frac{\partial f_1}{\partial f_2}\frac{\partial f_2}{\partial \mathbf{w_2}}$ and $\frac{\partial f_1}{\partial f_2}\frac{\partial f_2}{\partial f_3}$, we can use $\frac{\partial f_1}{\partial f_2}$ directly.

6 Deep learning choice

Now we have the algorithm to update parameters $\mathbf{w}$, we still need to build the architecture of our neural net work. There are two essential steps:

- Design a composite function $F(\mathbf{w}, x, y)$, or in practice, build the architecture of the neural network.
- Use an appropriate loss function

6.1 Choices of loss functions

- CrossEntropyLoss

  It is useful to train a classification problem with C classes.

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

  As mentioned in the section 2, the whole loss function which is our general framework often refereed as regularized cross entropy loss. We call it regularized is because that we have the regularization term $\frac{C}{2}\left\|\boldsymbol{w}\right\|_2^2$ where C is the *weight decay*. The second item of the general framework is the cross entropy loss.

- The negative log likelihood loss (NLLLoss)

  It is useful to train a classification problem with C classes. The input given through a forward call is expected to contain log-probabilities of each class. Obtaining log-probabilities in a neural network is easily achieved by adding a LogSoftmax layer in the last layer of the network [2].

- MSELoss

  Measures the mean squared error between n elements in the input x and target y.

$$l(x, y) = \frac{1}{n}\sum_i (x_i - y_i)^2$$

- Binary Cross Entropy Loss (BCELoss) In the slides, we have:
  loss(o,t) = -1/n\sum_i i(t[i]*log(o[i]) + (1-t[i])*log(1-o[i]))

  where o has to be positive and t is the class label.

  The o is the x, t is y in the following. Since we have the log item, we have to make sure that x_n to be positive

$$l(x, y) = \frac{1}{n}\sum_i y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)$$

There are many different loss functions and each of them tackle different problems, below are the loss functions mentioned in class.

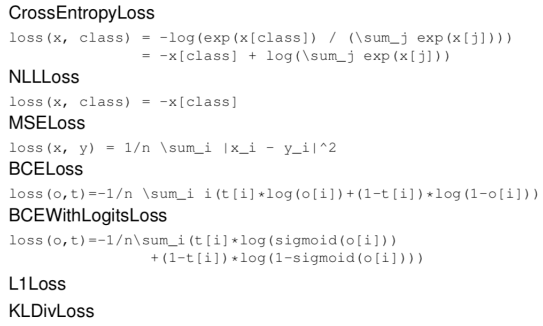

Figure 6: Loss functions

7 Remark

7.1 Information need to be stored

- During the inference process

  We need to store nothing since we just using exhaustive search or other methods to calculate the value of F.

- During the training process

  Store intermediate results for fully connected layer, convolution. Some functions can be combined to reduce storage of intermediate.

7.2 Troubles when we use F

Since $F(\mathbf{w}, x, y)$ is no longer constrained in any form, the loss function is generally no longer convex. Thus, we are not longer guaranteed to find the global optimum. For the same reason, the initialization of $\mathbf{w}$ matters. If we initialize $\mathbf{w}$ poorly, we may fall into local minimum easily. Besides, the initialization of $\mathbf{w}$ is not well understood in general now.

7.3 Some other remark

- A deep net with a single fully connected layer is equivalent to logistic regression
- Deep nets automatically learn features space transformation. Thus we do need to it manually. However, deep nets are computationally demanding and require significant amounts of training data.

7.4 Algorithmic advances

- Rectified linear unit ($max\{0, x\}$) activation as opposed to sigmoid
- Dropout

  Decorrelates different units. According to the implementation of dropout layer, only keeping a neuron active with some probability, otherwise set it to zero. Since different neuron learns different features and we drop some neuron [5]. Thus, we can say that we decorrelate different units since some units or neurons are droped(set to 0).

- Good initialization heuristics

  As aforementioned, it is critical to give the parameters a good initialization. There are some values that we should not choose. For example, all zero initialization [4]. According to the same source, Small random numbers, and Calibrating the variances with 1/sqrt(n) are great method to initialize.

- Batch-Normalization during training

  1. Normalizes data when training really deep nets
  2. Normalize by subtracting mean and dividing by standard deviation

  In general, Normalization forcing the activations of a network to take on a unit gaussian distribution at the beginning of the training. The reason that we do this is because this could solve a lot of problems during training and more information could find here [6].

References

[1] *ImageNet Classification with Deep Convolutional Neural Networks*, 2015. Available at http://vision.stanford.edu/teaching/cs231b$_{s}pring1415/slides/alexnet_{t}ugce_{k}yunghee.pdf$.

[2] *Loss functions*, 2018. Available at https://pytorch.org/docs/stable/nn.htmlloss-functions.

[3] W. D. Jianing Wei, Anurag Bhardwaj. *Deep Learning Essentials*. Packt Publishing, 2018.

[4] F.-F. Li. *Convolutional Neural Networks for Vision Recognition*, 2017. Available at http://cs231n.github.io/neural-networks-2/.

[5] A. K. I. S. R. S. Nitish Srivastava, Geoffrey Hinton. Dropout: A simple way to prevent neural networks from overfitting, 2014.

[6] C. S. Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.