Deep Learning — Assignment 3

Anirudhan J. Rajagopalan
Department of Computer Science
New York University
New York, NY.
ajr619@nyu.edu

1 General Questions

1.1 Distilling a network into one module

A linear activation function can be defined as

$$a^i = w^{(i)}x + b^i$$

A linear combination of such layers can be expressed as a sum of all the individual activations. For example, given two activation layers

$$\begin{aligned} y =& w^{(2)}x_{out} + b^{(2)} \\ =& w^{(2)}w^{(1)}x_{in} + w^{(2)}b^{(1)} + b^{(2)} \end{aligned}$$

Generalizing this, we can write

$$\begin{aligned} y =& Wx_{in} + b \\ W =& \prod_{i=1}^{l} w_i \\ b =& \sum_{i=1}^{l}(\prod_{l}^{i} w_i)b_{i-1} + b_n \end{aligned}$$

1.2 Dictionary used in Sparse coding vs Autoencoders

An autoencoder is a model which tries to reconstruct the input using some sort of constraint. Without any constraints an autoencoder can learn just the identity function which will will always provide the least reconstruction loss. Thus a dictionary learnt by an autoencoder need not be sparse. Additionally there are different type of encoding techniques, like denoising autoencoders which can be used to remove noise from an input. Sparse coding is a encoding method which can forces the model to find a sparse and overcomplete representation of the input. Thus a dictionary learnt with denoising autoencoder will always be sparse (there will be lot of features with zeros).

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

2 Softmax regression gradient calculation

2.1 Derive $\frac{\partial l}{\partial W_{i,j}}$

Given:

$$\begin{aligned}\hat{y} =& \sigma(Wx + b)\\ \sigma(a)_i =& \frac{\exp(a_i)}{\sum_j \exp(a_i)}\\ l(y, \hat{y}) =& -\sum_i y_i \log \hat{y}_i\end{aligned}$$

By using chain rule,

$$\frac{\partial l}{\partial W_{ij}} = \frac{\partial l}{\partial \sigma(a)k} \frac{\partial \sigma(a)_k}{\partial W_{ij}}$$

also,

$$\begin{aligned}\frac{\partial l}{\partial \sigma(a)_k} =& -\frac{yk}{\sigma(a)_k}\\ \frac{\partial a_l}{\partial W_{ij}} =& \delta_{il} x_j\\ \implies \frac{\partial \sigma(a)_k}{\partial W_{ij}} =& \frac{\partial \sigma(a)_k}{\partial (a)_l} \frac{\partial (a)_l}{\partial W_{ij}}\\ =& \frac{\partial \sigma(a)_k}{\partial (a)_l} \delta_{il} x_j\\ =& \frac{\partial \sigma(a)_k}{\partial (a)_i} x_j\end{aligned}$$

Using chain rule:

$$\frac{\partial l}{\partial W_{ij}} = -\frac{yk}{\sigma(a)_k} \frac{\partial \sigma(a)_k}{\partial (a)_i} x_j$$

By using the equation obtained in assignment (2).

$$\frac{\partial \sigma(a)_k}{\partial (a)_i} = \sigma(a)_k (\delta_{ik} - \sigma(a)_i)$$

By substituting $y_i = \sigma(a)_i$

$$\begin{aligned}\frac{\partial l}{\partial W_{ij}} =& -\frac{yk}{\sigma(a)_k} \sigma(a)_k (\delta_{ik} - \sigma(a)_i)\\ =& -yk(\delta_{ik} - \sigma(a)_i) x_j\\ =& (\hat{y}_i - y_i) x_j\end{aligned}$$

2.2 Loss functions and gradients

2.2.1 When $y_{c1} = 1, \hat{y_{c2}} = 1, c_1 \neq c_2$

When $y_{c1} = 1$,

$$\begin{aligned}l =& -\log(\hat{y}_{c1})\\ \frac{\partial l}{\partial W_{ij}} =& -(\delta_{ic1} - \hat{y}_i) x_j\end{aligned}$$

When $\hat{y_{c2}} = 1, c_1 \neq c_2$,

$$\frac{\partial l}{\partial W_{c1j}} = -x_j$$
$$\frac{\partial l}{\partial W_{c2j}} = x_j$$

2.2.2 Why there is no need to worry about this situation

Since the weights of the hidden layers are initialized randomly, the possibility of $\hat{y}_{c2} = 1$ is negligible. When the weights changes during learning, it moves away from this point. Hencec we need not worry about this situation.

3 Chain rule

3.1 Calculating gradients using chain rule

Given

$$f(x, y) = \frac{x^2 + \sigma(y)}{3x + y - \sigma(x)}$$

This can be considered as a equation in $x, y, \sigma(x), \sigma(x)$

$$f(x, y) = E(x, y, \sigma(x), \sigma(y))$$

The gradient calculated using the chain rule is:

$$\frac{\partial f}{\partial x} = \frac{\partial E}{\partial x} + \frac{\partial E}{\partial \sigma(x)} \frac{\partial \sigma(x)}{\partial x}$$
$$\frac{\partial f}{\partial y} = \frac{\partial E}{\partial y} + \frac{\partial E}{\partial \sigma(y)} \frac{\partial \sigma(y)}{\partial y}$$

Where:

$$\frac{\partial E}{\partial x} = \frac{2x(3x + y - \sigma(x)) - (x^2 + \sigma(y))3}{(3x + y - \sigma(x))^2}$$
$$\frac{\partial E}{\partial y} = -\frac{x^2 + \sigma(y)}{(3x + y - \sigma(x))^2}$$
$$\frac{\partial E}{\partial \sigma(x)} = \frac{x^2 + \sigma(y)}{(3x + y - \sigma(x))^2}$$
$$\frac{\partial E}{\partial \sigma(y)} = \frac{1}{(3x + y - \sigma(x))}$$
$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1 + e^{-x})^2}$$
$$\frac{\partial \sigma(y)}{\partial y} = \frac{e^{-y}}{(1 + e^{-y})^2}$$

3.2 Gradients at $x = 1, y = 0$

$$\frac{\partial E}{\partial x}(x = 1, y = 0) = 0.007$$
$$\frac{\partial E}{\partial y}(x = 1, y = 0) = -0.29$$
$$\frac{\partial E}{\partial \sigma(x)} = 0.29$$
$$\frac{\partial E}{\partial \sigma(y)} = 0.44$$
$$\frac{\partial \sigma(x)}{\partial x} = 0.19$$
$$\frac{\partial \sigma(y)}{\partial y} = 0.25$$

Substituting the gradients:

$$\frac{\partial f}{\partial x} = 0.064$$
$$\frac{\partial f}{\partial y} = -0.18$$

4 Variants of Pooling

Pooling is an operation in Convolutional Neural Networks which is performed after a convolutional layer. We decide on a size of the region to do pool our convolutional filters over. Then we divide the convolved features into disjoint regions and take the max, mean or norm of the feature activations depending on the type of pooling operation. Generally the stride length used in pooling is equal to the pooling window size.

4.1 Three types of pooling

There are three types of pooling operations that are used

Max pooling Max pooling takes the maximum of the feature activations. Implemented by SpatialMaxPooling.

Average pooling Average pooling takes the average of the feature activations. Implemented by SpatialAveragePooling.

Lp pooling LpPooling takes the lp norm of the feature activations. Implemented by SpatialLPPooling.

4.2 Mathematical formulas for pooling

Suppose we have a NxN convolutional layer (filters) and a pooling layer of size mxm where $m < N$ The pooling operation is generally implemented using a stride of m. Therefore the size of filters after pooling will be $\frac{N}{m} * \frac{N}{m}$ and each of the mxm pooling operations gives a single value. The single value for a pooling operation can be represented by

$$y = \sigma(x_{ij})$$

where $1 \leq i \leq m$ and $1 \leq j \leq m$.

4.2.1 MaxPooling

Incase of max pooling

$$\sigma = \max(x_{ij}) \qquad 1 \leq i \leq mand1 \leq j \leq m$$

4.2.2 Average Pooling

Incase of average pooling

$$\sigma = \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij}}{m * m}$$

4.2.3 LPPooling

Incase of LP pooling where $p \geq 2$.

$$\sigma = (\sum_{i=1}^{m} \sum_{j=1}^{m} x_{ij}^{p})^{\frac{1}{p}}$$

4.3 Reason for using in deep learning

Pooling is used for a number of reasons.

1. Computational tractability. The number of features after convolution increases by $numFIltersx(N-m+1)x(N-m+1)$. Generally it will be computationally intractable to perform learning on this huge data.
2. Overfitting. Since the number of features are high, this will lead to overfitting. Pooling helps in reducing the number of features.
3. Stationary property. Generally images have stationary property. This means that features that are useful in one region will be useful in other regions also. Pooling operation helps in aggregating the features at various locations.

We generally use Max-pooling for images as the performance of Max pooling has been emperically shown to be better than Average or Lp pooling for image workflows. This can be explained by the stationary property of the image as the most important activation in a small patch can be found by using max pooling.

5 Convolution

5.1 Number of values after forward propogation

When we apply a convolution on a $N \times N$ image with a kerenel of size $m \times m$ we get $(N - m + 1) \times (N - m + 1)$ output values after the convolution.

Here, N = 5, m = 3. Therefore we will get $(5 - 3 + 1) \times (5 - 3 + 1) = 9$ values.

5.2 Values after forward propogation

The values of filters were hand calculated and verified by the code in [2]. The resulting filter obtained by applying the convolutional kernel is:

$$\begin{matrix} 109 & 92 & 72 \\ 108 & 85 & 74 \\ 110 & 74 & 79 \end{matrix}$$

5.3 Value of gradient

The values of gradient were hand calculated and verified by the code in [2]

4	7	10	6	3
9	17	25	16	8
11	23	34	23	11
7	16	24	17	8
2	6	9	7	3

6 Optimization

6.1 Mathematical formula for reconstruction loss

An autoencoder is a model which tries to reconstruct the input. This reconstruction loss for real valued input is generally given by

$$\begin{aligned} L =& \frac{1}{2}\sum_k ||\hat{x}_k - x_k||^2 \\ \hat{x}_k =& \sigma_2(W'\sigma_1(WX+b)+b') \end{aligned}$$

Here, σ_2 denotes the decoding function and σ denotes the encoding function.

When the input are real valued, we can use cross entropy loss.

$$L(X,Z) = -\sum_{k=1}^{d}[X_k \log Z_k + (1-X_k)\log(1-Z_k)]$$

6.2 Gradient of the loss with respect to the loss

$$\begin{aligned} A_1 =& Wx \\ A_2 =& W^*(\sigma_1 A_1) \\ \frac{\partial l}{\partial W} =& \frac{\partial l}{\partial \sigma_2}\frac{\partial \sigma_2}{\partial A_2}\frac{\partial A_2}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial A_1}\frac{\partial A_1}{\partial W} \\ \frac{\partial l}{\partial W^*} =& \frac{\partial l}{\partial \sigma_2}\frac{\partial \sigma_2}{\partial A_2}\frac{\partial A_2}{\partial W^*} \\ \frac{\partial l}{\partial W} =& -2(x-\sigma_2(A_2))\frac{\partial \sigma_2}{\partial A_2}W^*\frac{\partial \sigma_1}{\partial A_1}x \\ \frac{\partial l}{\partial W^*} =& -2(x-\sigma_2(A_2))\frac{\partial \sigma_2}{\partial A_2}(\sigma A_1) \end{aligned}$$

6.3 Gradient descent step

The gradient descent is eta η multiplied by the gradient of loss with respect to the parameters

$$\begin{aligned} w*^t =& w*^{t-1} - \eta\frac{\partial l}{\partial W*^{t-1}} \\ w^t =& w^{t-1} - \eta\frac{\partial l}{\partial W^{t-1}} \end{aligned}$$

6.4 Gradient step with momentum

Momentum is used to help the network out of local minimas. By adding the momentum term, the udpate step is now

$$\begin{aligned} w*^t =& w*^{t-1} - \eta\frac{\partial l}{\partial W*^{t-1}} - m\times\eta\frac{\partial l^{t-1}}{\partial W*^{t-2}} \\ w^t =& w^{t-1} - \eta\frac{\partial l}{\partial W^{t-1}} - m\times\eta\frac{\partial l^{t-1}}{\partial W^{t-2}} \end{aligned}$$

for a fixed eta.

7 Top-k error

7.1 Definition

Top-k error is an error metric used in ranking problems. This is the number of samples, x_i with correct labels y_i that doesn't appear in the top-k predicted results of the model in the order of the probability measure or score used for ranking. According to LVSRC 2014 by Imagenet [3], the error is defined as

$$e = \frac{1}{n}. \sum_k \min_i d(c_i, C_k)$$

where C_k is the ground truth of the image with k = 1, ..., n labels. and c_i is the labels generated by the classification algorithm with i $\in$ 1, ... 5.

Given this formulation, using the top-5 error helps us not to penalize a classification algorithm if it finds an object in an image which is not present in ground truth.

Using top-1 error ensures how close the object identified in the image by an algorithm is with respect to the original image in imagenet database.

8 t-SNE

8.1 Crowding problem

The volume of a sphere centered on datapoint i scales as r and m, where r is the radius and m the dimensionality of the sphere. If the data is uniformly distributed around i in the high dimensional manifold, and when we try to project them into two dimensional manifold, the area in the lower dimensional manifold available to map distant data points will not be large enough compared to the area available to map nearby datapoints. This leads to a large number of data points collecting to the center of the map. This is called as crowding problem.

In t-SNE we solve this problem by using gaussian distribution for high dimensional data and student-t distribution for the two dimensional data. [4]

8.2 Derive $\frac{\partial C}{\partial y_i}$

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_i \sum_j (p_{ij} \log p_{ij} - p_{ij} \log q_{ij}) \tag{1}$$

Lets define $d_{ij} = ||y_i - y_j||$, then the expression of q_{ij} is:

$$q_{ij} = \frac{e^{-d_{ij}^2}}{\sum_{k \neq l} e^{-d_{kl}^2}} \tag{2}$$

the dependency of C from y_i is going to be only through d_{ij}^2 and d_{ji}^2:

$$\frac{\partial C}{\partial y_i} = \sum_j (\frac{\partial C}{\partial d_{ij}^2} \frac{\partial d_{ij}^2}{\partial y_i} + \frac{\partial C}{\partial d_{ji}^2} \frac{\partial d_{ji}^2}{\partial y_i})$$

Because of the symmetry for exchange of y_i and y_j of d_{ij}^2:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j \frac{\partial C}{\partial d_{ij}^2} (y_i - y_j) \tag{3}$$

Let:

$$Z = (\sum_{k \neq l} e^{-d_{kl}^2}) \tag{4}$$

And then from (2):

$$q_{ij} = \frac{e^{-d_{ij}^2}}{Z} \Rightarrow q_{ij}Z = e^{-d_{ij}^2} \tag{5}$$

The first part of (1) is a constant with respect to d_{ij} so:

$$\frac{\partial C}{\partial d_{ij}^2} = -\sum_{k\neq l} p_{kl}\frac{\partial(\log q_{kl})}{\partial d_{ij}^2} = -\sum_{k\neq l} p_{kl}\frac{\partial(\log q_{kl}Z - \log Z)}{\partial d_{ij}^2}$$

after multiplying and dividing p_{kl} for Z and split the logarithm. Using (5):

$$\frac{\partial C}{\partial d_{ij}^2} = -\sum_{k\neq l} p_{kl}[(\frac{1}{q_{kl}Z})\frac{\partial e^{-d_{kl}^2}}{\partial d_{ij}^2} - \frac{1}{Z}\frac{\partial Z}{\partial d_{ij}^2}] \tag{6}$$

Calculating derivatives in this equation:

$$\begin{aligned}\frac{\partial e^{-d_{kl}^2}}{\partial d_{ij}^2} &= -e^{-d_{ij}^2}\delta_{ik}\delta_{jl} \\ \frac{\partial Z}{\partial d_{ij}^2} &= -e^{-d_{ij}^2}\end{aligned}$$

where we used the Kronecker delta δ_{ik} that is 1 for $i = k$ and 0 otherwise. Then substituting in (6). The only terms of the sum that remains is the one with $i = k$ and $j = l$:

$$\frac{\partial C}{\partial d_{ij}^2} = p_{ij}(\frac{1}{q_{ij}Z})e^{-d_{ij}^2} - \sum_{k\neq l} p_{kl}\frac{1}{Z}e^{-d_{ij}^2} = (p_{ij} - q_{ij})$$

where we used the (5) and the fact that $\sum_{k\neq l} p_{kl} = 1$. Substituting in (8.2):

$$\frac{\partial C}{\partial y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

9 Proximal gradient descent

9.1 A

$$\frac{\partial}{\partial z}(\frac{1}{2}||z - x||_2^2 + t||z||_1) = z_i - |x| + \text{sign}(||z||_1)t = 0$$

Solving for z:

$$z = |x| - \text{sign}(||z||_1)t$$

We have three possible values depending on the value of z being positive, negative or null:

$$\begin{aligned} z &= |x| - t \text{ for } x > t \\ z &= |x| + t \text{ for } x < -t \\ z &= 0 \text{ for } |x| < t \end{aligned}$$

That is the soft-thresholding function:

$$S_t(x) = (|x| - t)_+ \odot \text{sign}(x)$$

And then for a vector:

$$S_t(\mathbf{x}) = (|\mathbf{x}| - t)_+ \odot \text{sign}(\mathbf{x}) \tag{7}$$

9.2 B

By substituting $\nabla g(x_k) = -X^T(y - Xx_k)$ in:

$$x_{k+1} = \text{prox}_{h,\alpha_k}(x_k - \alpha_k \nabla g(x_k)) = S_{\alpha_k}(x_k - \alpha_k \nabla g(x_k))$$

$$x_{k+1} = S_{\alpha_k}(x_k - \alpha_k X^T(Xx_k - y))$$

This is exactly the update rule used for ISTA, as in [1] with a few variable changes.

9.3 C

$$u = \text{prox}_{h,t} \Rightarrow \partial_u[\frac{1}{2}||u-x||_2^2 + th(u)] \ \in \ 0$$
$$u - x + t\partial h(u) \ \in \ 0$$

:

$$\implies \ \frac{x-u}{t} \ \in \ \partial h(u)$$

9.4 D

From $G_{\alpha_k}(x_k)$ we know that,

$$\text{prox}_{h,t}(x_k - \alpha_k \nabla g(x_k)) = x_k - \alpha_k G_{\alpha_k}(x_k) = x_{k+1}$$

and

$$\text{prox}_{h,t}(x_k - \alpha_k \nabla g(x_k)) = \text{argmin}_{x_{k+1}}(\frac{1}{2}||x_{k+1} - x_k + \alpha_k \nabla g(x_k)||_2^2 + th(x_{k+1}))$$

$$\implies \partial(\frac{1}{2}||x_{k+1} - x_k + \alpha_k \nabla g(x_k)||_2^2 + th(x_{k+1})) = 0$$

By solving the above equations,

$$x_{k+1} = x_k - \alpha_k \nabla g(x_k) - t\partial h(x_{k+1}) \tag{8}$$

Evaluating

$$G_{\alpha_k}(x_k) - \nabla g(x_k) = \frac{x_k}{\alpha_k} - \frac{1}{\alpha_k}\text{prox}_{h,t}(x_k - \alpha_k \nabla g(x_k)) - \nabla g(x_k)$$

using (8):

$$G_{\alpha_k}(x_k) - \nabla g(x_k) \in \frac{x_k}{\alpha_k} - \frac{1}{\alpha_k}(x_k - \alpha_k \nabla g(x_k) - t\partial h(x_{k+1})) - \nabla g(x_k)$$

we get:

$$G_{\alpha_k}(x_k) - \nabla g(x_k) \in \partial h(x_{k+1})$$

References

[1] Julien Mairal, Francis Bach, and Jean Ponce. Sparse modeling for image and vision processing. *arXiv preprint arXiv:1411.3230*, 2014.

[2] Anirudhan J. Rajagopalan. Homework 3 code. `https://github.com/rajegannathan/Deep-Learning/tree/master`

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[4] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.