A technical report about Deep Learning

Tao Hu
Department of Computer Science
Peking University
No.5 Yiheyuan Road Haidian District, Beijing, P.R.China
taohu@pku.edu.cn

Abstract

This report mainly talks about back propagation principle, batch normalization, fine tuning, bayesian optimization of Deep Learning. some ablation study are listed. different activation function ablation study also is listed.

1 Back Propogation Study

Derivation of backprop for MLP with ReLU, Sigmoid,and try an implementation.

Solution

Basic Back-propagation procedure:

$$z^{(2)} = w^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = w^{(2)}a^{(2)} + b^{(2)}$$
$$\text{in the middle: } h_{w,b}(x) = a^{(3)} = f(z^{(3)})$$

Notations:

$$w^{(i)} \text{ : weight w of the i-th layer}$$
$$b^{(i)} \text{ :bias b of the i-th layer}$$
$$z^{(i)} \text{ : result of w,b}$$
$$a^{(i)} \text{ :result of activation, and } a^{i+1} = x^i$$

the optimizer target function is :

$$J(W,b;x,y) = \frac{1}{2} \parallel h_{w,b}(x) - y \parallel_2^2$$
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} J(w,b;x,y) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$
$$= \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2} \parallel h_{w,b}(x) - y \parallel_2^2 + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

the total back-propagation procedure is as follows:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(w,b)$$
$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial}{\partial b_i^{(l)}}J(w,b)$$

Author Info: Taohu ,Peking University

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(w,b,x^{(i)},y^{(i)}) + \lambda W_{ij}^{l}$$

$$\frac{\partial}{\partial b_{i}^{(l)}} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b_{i}^{(l)}} J(w,b,x^{(i)},y^{(i)})$$

let $\delta_i^{(n_l)} = \frac{\partial}{\partial Z_i^{n_l}} J(W,b;x,y)$:

$$\begin{aligned}
\delta_i^{(n_l)} &= \frac{\partial}{\partial Z_i^{n_l}} J(W,b;x,y) \\
&= \frac{\partial}{\partial Z_i^{n_l}} \frac{1}{2}\sum_{j=1}^{S^{n_l}} (y_j - a_j^{n_l})^2 \\
&= \frac{\partial}{\partial Z_i^{n_l}} \frac{1}{2}\sum_{j=1}^{S^{n_l}} (y_j - f(Z_j^{n_l}))^2 \\
&= -(y_i - f(Z_i^{n_l}))f^{'}(Z_i^{(n_l)}) \\
&= -(y_i - a_i^{(n_l)})f^{'}(Z_i^{(n_l)})
\end{aligned}$$

the deduction above can facilitate we acquire δ from the bottom(close to the loss function side) of the network.

Now Let's deduce how to acquire δ iteratively .

$$\begin{aligned}
\delta_i^{n_l-1} &= \frac{\partial}{\partial Z_i^{n_l-1}} J(W,b;x,y) = \frac{\partial}{\partial Z_i^{n_l-1}} \partial 12 \parallel y - h_{w,b}(x) \parallel_2^2 \\
&= \frac{\partial}{\partial Z_i^{n_l-1}} \frac{1}{2}\sum_{j=1}^{s_{n_l}} (y_i - a_j^{n_l})^2 \\
&= \frac{1}{2}\sum_{j=1}^{s_{n_l}} \frac{\partial}{\partial Z_i^{n_l-1}} (y_i - a_j^{n_l})^2 \\
&= \sum_{j=1}^{S_{nl}} -(y_j - f(z_j^{nl}) \frac{\partial}{\partial Z_i^{n_l-1}} f(z_j^{nl}) \\
&= \sum_{j=1}^{S_{nl}} -(y_j - f(z_j^{nl}) \frac{\partial}{\partial Z_i^{n_l-1}} f^{'}(z_j^{nl}) \frac{\partial Z_j^{nl}}{Z_i^{nl-1}} \\
&= \sum_{j=1}^{S_{nl}} \delta_j^{nl} \frac{\partial Z_j^{nl}}{Z_i^{nl-1}} \\
&= \sum_{j=1}^{S_{nl}} \delta_j^{nl} \frac{\partial}{Z_i^{nl-1}} \sum_{k=1}^{S_{nl-1}} f(Z_k^{nl-1}) W_{jk}^{nl-1} \\
&= \sum_{j=1}^{S_{nl}} \delta_j^{nl} W_{ji}^{nl} f^{'}(Z_i^{nl-1}) \\
&= (\sum_{j=1}^{S_{nl}} \delta_j^{nl} W_{ji}^{nl}) f^{'}(Z_i^{nl-1})
\end{aligned}$$

thus we can acquire every layer's δ,which is convenient for latter back-propagation.

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(w,b;x,y) = \frac{\partial}{\partial Z_i^{(l+1)}} * \frac{\partial Z_i^{(l+1)}}{\partial W_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)} \quad \frac{\partial}{\partial b_{ij}^{(l)}} J(w,b;x,y) = \delta_i^{(l+1)}$$

Finally:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{l}} J(w,b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{l}} J(w,b)$$

The Final back-propagation procedure is:

1,forward propagation.

$$f(z) = \frac{1}{1+exp(-z)}$$

$$z^{(l+1)} = w^{(l)} a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f(z^{(l+1)})$$

2, calculate the nl-layer's neural unit's $\delta_i^{(nl)}$:

$$\delta_i^{nl} = -(y_i - a_i^{nl}) * f'(z_i^{(l)})$$

3,calculate every layer's neural unit $\delta_i^{(l)}$ iteratively from output layer to input layer.

$$\delta_i^{(l)} = (\sum_{j=1}^{S_l+1} w_{ji}^{(l)} \delta_i^{(l+1)}) * f'(z_i^{(l)})$$

4,calculate $\frac{\partial}{\partial W_{ij}^{(l)}} J(w,b;x,y)$ and $\frac{\partial}{\partial b_{ij}^{(l)}} J(w,b;x,y)$ of every layer:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(w,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_{ij}^{(l)}} J(w,b;x,y) = \delta_i^{(l+1)}$$

5, calculate $\frac{\partial}{\partial W_{ij}^{(l)}} J(w,b)$ and $\frac{\partial}{\partial b_i^{(l)}} J(w,b)$ of every layer.

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(w,b,x^{(i)},y^{(i)}) + \lambda W_{ij}^l$$
$$\frac{\partial}{\partial b_i^{(l)}} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(w,b,x^{(i)},y^{(i)})$$

6,update $W_{ij}^{(l)}$, $b_i^{(l)}$ of every layer

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^l} J(w,b)$$
$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^l} J(w,b)$$

7,until J(w,b) is small enough:

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} J(w,b;x,y) + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ji}^{(l)})^2$$

this is the deduction of the sigmoid activation function, the deduction of ReLu,LeakyReLu is the same. the only difference is the derivation of the activation function, the other place just keeps the same reduction according the chain rule.

Run a Mnist Classifer based on tensorflow

Solution

just run the python script by "python multilayer_perceptron.py".

During the script, we construct the model with layer input layer size 784(flattend by the input image),two intermediate layers with size 256,and output size 10(equal to the final class size).

and then we define the loss and optimizer.

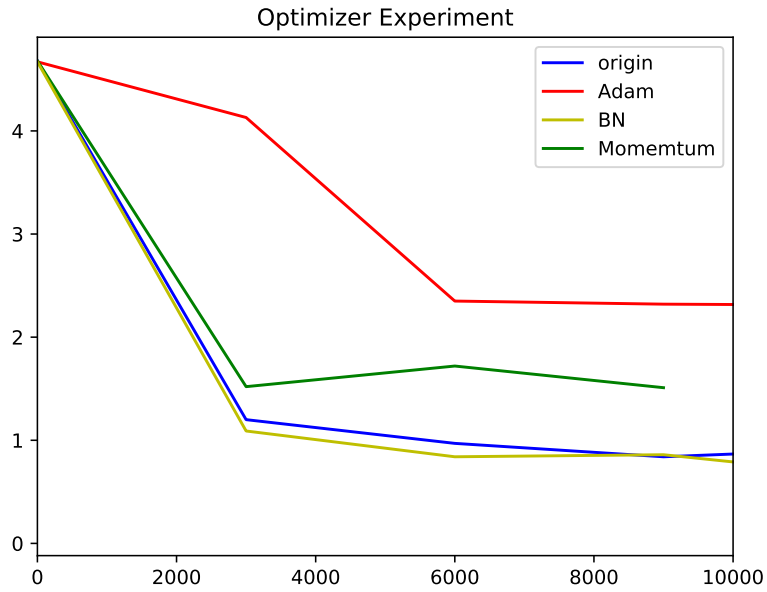
finally we set the batch size = 100,and output epoch of the result.after the training.the test data inference result will print as follows.

```
Epoch: 0001 cost= 212.434582488
Epoch: 0002 cost= 43.698304957
Epoch: 0003 cost= 27.303758452
Epoch: 0004 cost= 19.144715998
Epoch: 0005 cost= 13.905557823
Epoch: 0006 cost= 10.521803028
Epoch: 0007 cost= 7.942286914
Epoch: 0008 cost= 6.117937308
Epoch: 0009 cost= 4.404222389
Epoch: 0010 cost= 3.519790992
Epoch: 0011 cost= 2.575894599
Epoch: 0012 cost= 1.885512865
Epoch: 0013 cost= 1.441038678
Epoch: 0014 cost= 1.136289034
Epoch: 0015 cost= 0.908172189
Optimization Finished!
Accuracy: 0.9474

Process finished with exit code 0
```

2 Different Optimizer

the network structure is as follows:



3 Batch Normalization really matters ?

we compare diverse normalization method include batch normalization[], local responsive normalization[], non-normalization.

different bn location cannot be tested due to time limit.

4 Magical SeLU

The experiment is based on the code in `https://github.com/shaohua0116/Activation-Visualization-Histogram`.

5 Bayesian Optimization

The experiment is based on the `https://github.com/fmfn/BayesianOptimization`.

References