CSE 291 I00: Machine Learning for 3D Data Homework 1

Qiaojun Feng
A53235331
Department of Electrical and Computer Engineering
University of California, San Diego
qjfeng@ucsd.edu

1

Compute the derivative of the cross-entropy loss L w.r.t o, i.e compute $\frac{\partial L}{\partial o}$.

$$
\begin{aligned}
\frac{\partial L}{\partial o} &= \frac{\partial L}{\partial p}\frac{\partial p}{\partial o} \\
L &= -\sum_{\text{for all class } c} y_c log(p_c) \\
\frac{\partial L}{\partial p_c} &= -\frac{y_c}{p_c} \\
p_c &= \frac{e^{o_c}}{\sum_{j=0}^{K} e^{o_j}} \\
\frac{\partial p_c}{\partial o_i} &= \begin{cases} p_c(1-p_i) & i = c \\ -p_c p_i & i \neq c \end{cases} \\
\frac{\partial L}{\partial o_i} &= \sum_{\text{for all class } c} \frac{\partial L}{\partial p_c}\frac{\partial p_c}{\partial o_i} \\
&= -y_i(1-p_i) + \sum_{c \neq i} y_c p_i \\
&= -y_i + (y_i + \sum_{c \neq i} y_c)p_i \\
&= p_i - y_i
\end{aligned}
$$

The detailed derivation of softmax's derivative:
For $i = c$:

$$
\begin{aligned}
\frac{\partial p_c}{\partial o_i} &= \frac{e^{o_c}\sum_{j=0}^{K} e^{o_j} - e^{o_c}e^{o_i}}{(\sum_{j=0}^{K} e^{o_j})^2} \\
&= \frac{e^{o_c}}{\sum_{j=0}^{K} e^{o_j}} \times \frac{\sum_{j=0}^{K} e^{o_j} - e^{o_i}}{\sum_{j=0}^{K} e^{o_j}} \\
&= p_c(1-p_i)
\end{aligned}
$$

For $i \neq c$:

$$
\begin{aligned}
\frac{\partial p_c}{\partial o_i} &= \frac{-e^{o_c}e^{o_i}}{(\sum_{j=0}^{K} e^{o_j})^2} \\
&= -\frac{e^{o_c}}{\sum_{j=0}^{K} e^{o_j}} \times \frac{e^{o_i}}{\sum_{j=0}^{K} e^{o_j}} \\
&= -p_c p_i
\end{aligned}
$$

2

(1)Draw the computational graph of the network.

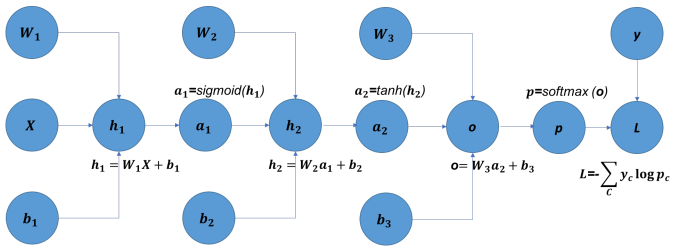
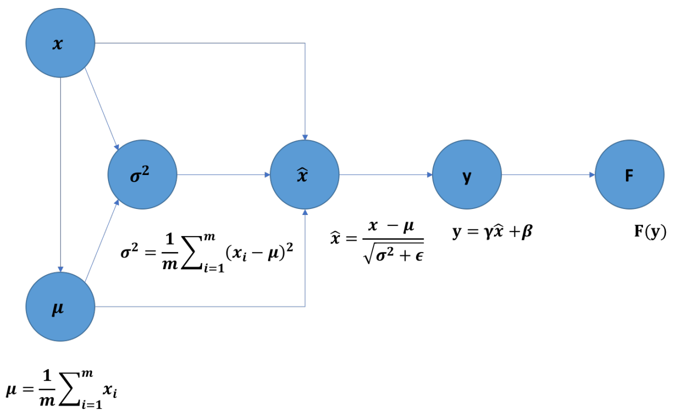


Fig. 1. computational graph of network

(2)Perform Backpropagation to compute the gradients of loss L w.r.t all the weights and biases,
i.e compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial W_3}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_3}$.

$$
\begin{aligned}
\frac{\partial L}{\partial W_3} &= \sum_{i=0}^{K} \frac{\partial L}{\partial o_i}\frac{\partial o_i}{\partial W_3} \\
&= \sum_{i=0}^{K} (p_i - y_i)a_2^T \\
\frac{\partial L}{\partial b_3} &= \sum_{i=0}^{K} \frac{\partial L}{\partial o_i}\frac{\partial o_i}{\partial b_3} \\
&= \sum_{i=0}^{K} (p_i - y_i) \\
\frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial h_2}\frac{\partial h_2}{\partial W_2} \\
&= W_3^T \sum_{i=0}^{K} (p_i - y_i)(1 - a_2^2)a_1^T \\
\frac{\partial L}{\partial b_2} &= \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial h_2}\frac{\partial h_2}{\partial b_2} \\
&= W_3^T \sum_{i=0}^{K} (p_i - y_i)(1 - a_2^2) \\
\frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial h_1}\frac{\partial h_1}{\partial W_1} \\
&= W_2^T W_3^T \sum_{i=0}^{K} (p_i - y_i)(1 - a_2^2)a_1(1 - a_1)X^T
\end{aligned}
$$



x
σ²
x̂
y
F
μ
$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$
$\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}$
$y = \gamma\hat{x} + \beta$
F(y)
$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$

Fig. 2. computational graph of BatchNorm layer

$$
\begin{aligned}
\frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial h_1}\frac{\partial h_1}{\partial b_1} \\
&= W_2^T W_3^T \sum_{i=0}^{K}(p_i - y_i)(1-a_2^2)a_1(1-a_1)
\end{aligned}
$$

3

(1)Draw the computational graph of the batchnorm layer. (2)Consider a function $F(y_i)$ and compute the derivatives of $F(y_i)$ w.r.t γ, β and x, i.e compute $\frac{\partial F}{\partial \gamma}, \frac{\partial F}{\partial \beta}, \frac{\partial F}{\partial x}$. Assume $\frac{\partial F}{\partial y_i}$is given.

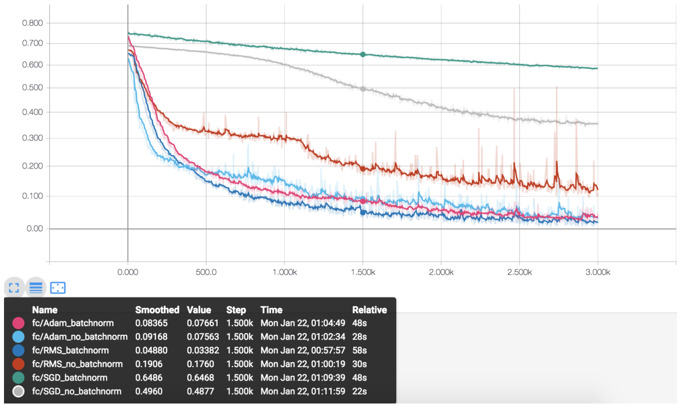
If we want derive the derivative from the batch of data. That is, function $F(y_1, \ldots, y_m)$. Then we should have:

$$
\begin{aligned}
\frac{\partial F}{\partial \gamma} &= \sum_{i=1}^{m}\frac{\partial F}{\partial y_i}\frac{\partial y_i}{\partial \gamma} \\
&= \sum_{i=1}^{m}\frac{\partial F}{\partial y_i}\hat{x}_i \\
\frac{\partial F}{\partial \beta} &= \sum_{i=1}^{m}\frac{\partial F}{\partial y_i}\frac{\partial y_i}{\partial \beta} \\
&= \sum_{i=1}^{m}\frac{\partial F}{\partial y_i} \\
\frac{\partial F}{\partial \hat{x}_i} &= \frac{\partial F}{\partial y_i}\gamma \\
\frac{\partial F}{\partial \sigma_{\mathcal{B}}^2} &= \sum_{i=1}^{m}\frac{\partial F}{\partial \hat{x}_i}(x_i - \mu_{\mathcal{B}})\frac{-1}{2}(\sigma_{\mathcal{B}}^2+\epsilon)^{-2/3}) \\
\frac{\partial F}{\partial \mu_{\mathcal{B}}} &= \sum_{i=1}^{m}\frac{\partial F}{\partial \hat{x}_i}\frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} \\
\frac{\partial F}{\partial x_i} &= \frac{\partial F}{\partial \hat{x}_i}\frac{1}{\sqrt{\sigma_{\mathcal{B}}^2+\epsilon}} + \frac{\partial F}{\partial \sigma_{\mathcal{B}}^2}\frac{2(x_i-\mu_{\mathcal{B}})}{m} + \frac{\partial F}{\partial \mu_{\mathcal{B}}}\frac{1}{m}
\end{aligned}
$$

4

I referred to Udacity's github deep-learning repo and built the network on its fundamental.

(1)(2)Construct a model using fully connected layers and ReLu layers to solve mnist classification problem. Plot a graph demonstrating how the loss function decreases over the number of iterations. Compare with and without batch normalization layers. Compare different optimization methods.



	Name	Smoothed	Value	Step	Time	Relative
●	fc/Adam_batchnorm	0.08365	0.07661	1.500k	Mon Jan 22, 01:04:49	48s
●	fc/Adam_no_batchnorm	0.09168	0.07563	1.500k	Mon Jan 22, 01:02:34	28s
●	fc/RMS_batchnorm	0.04880	0.03382	1.500k	Mon Jan 22, 00:57:57	58s
●	fc/RMS_no_batchnorm	0.1906	0.1760	1.500k	Mon Jan 22, 01:00:19	30s
●	fc/SGD_batchnorm	0.6486	0.6468	1.500k	Mon Jan 22, 01:09:39	48s
●	fc/SGD_no_batchnorm	0.4960	0.4877	1.500k	Mon Jan 22, 01:11:59	22s

Fig. 3. loss of fully connected network

For the dataset, there are 60000 as train set and 10000 as test set. We split the train set and use 5000 of them as validation set. So the remaining train set has 55000 images. Use batch size 110, we need 500 batches to go through the whole train set.

We designed a 8-fully-connected-layer network. The numbers of neurons in each layer are 256,128,64,32,16,8,4,10. Also we tried stochastic gradient descent, RMSProp and Adam optimizer. We can conclude that RMSProp and Adam are better than SGD, while batch normalization helps all of them.

Batch normalization helps because it normalize the input to each layer and make the model training easier. The weight will be changed in a smaller range without broadly search. Also since on each layer the derivative will be normalized, the back-propagation is more robust. The derivative is not likely to explode or vanish any more.

TABLE I
FULLY CONNECTED NETWORK TEST ACCURACY

Optimizer	SGD	RMSProp	Adam
no batchnorm	9.58%	86.23%	95.17%
batchnorm	17.60%	97.31%	97.21%

(3)Construct a Convolutional Neural Network and do the same things above.

Here we built a network with 7 convolution layers. Each layer's size is different from one another.

Again we compared the performance with and without batch normalization. Besides, we also tried different optimization methods, different conv layer numbers and different learning rate.

TABLE II
CNN TEST ACCURACY WITH DIFFERENT OPTIMIZERS

Optimizer	SGD	RMSProp	Adam
no batchnorm	60.65%	98.85%	98.98%
batchnorm	80.69%	99.02%	98.82%

TABLE III
CNN TEST ACCURACY WITH DIFFERENT CONVOLUTION LAYERS

conv_layers	3	5	7
no batchnorm	63.18%	78.09%	60.65%
batchnorm	83.57%	86.28%	80.69%

TABLE IV
CNN TEST ACCURACY WITH DIFFERENT LEARNING RATE

learning_rate	0.001	0.002	0.005
no batchnorm	66.94%	79.41%	93.04%
batchnorm	81.25%	88.98%	94.79%
learning_rate	0.01	0.02	0.1
no batchnorm	94.54%	96.73%	98.52%
batchnorm	96.11%	97.66%	98.99%

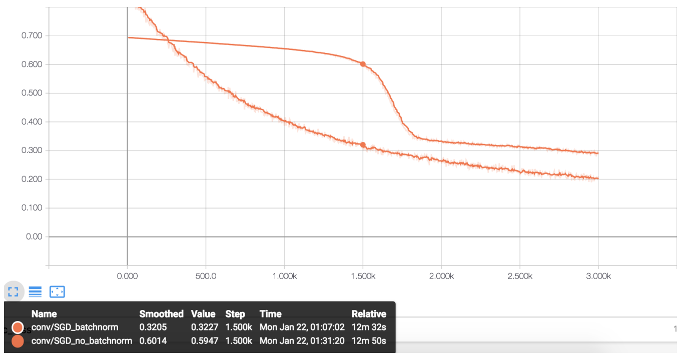
Easy to tell that batch normalization helps in most of the cases. And the reasons just like we said in (1)(2). Also, we may want to explain the batch normalization's role in CNN in another way. It is clear that batch normalization will make full use of more convolution kernel and layer, preventing some of them taking the dominant place and neglecting all the others.

Also for different optimization, SGD is the worst and RMSProp and Adam are really good.

For different structure of this network, the training results may be different. Intuitively, model with more layers has better approximation ability, but there are harder to train if play time is limited. Here we actually stopped before we can reach the best result since the loss is continuously decreasing, but we can also tell that some specific structure is better for this not-so-complex dataset.

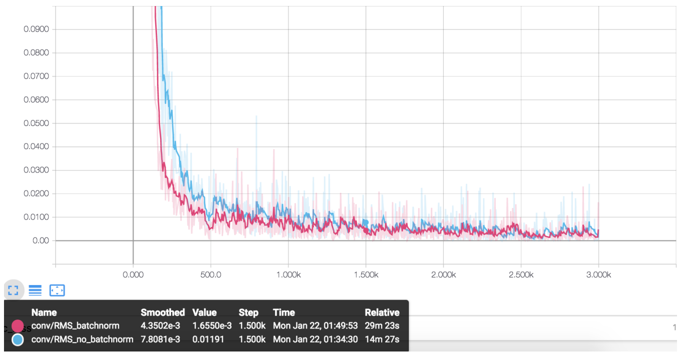
Finally for the learning rate part, we can see at least for this model, we should use a larger learning rate. Also larger learning rate can prevent from being trapped in some local optimal but global sub-optimal.

Finally we can tell batch normalization helps a lot. Also RMSProp and Adam optimizer are good optimizer compared with SGD.



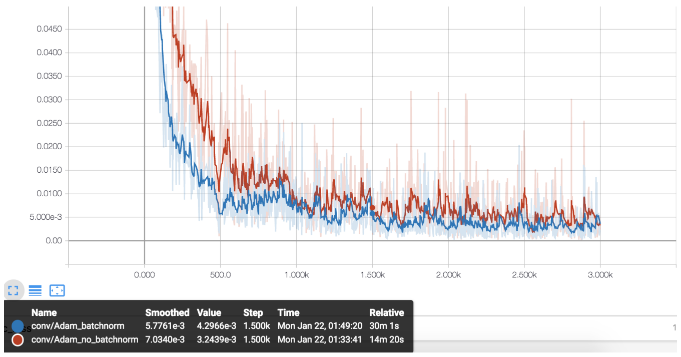
	Name	Smoothed	Value	Step	Time	Relative
●	conv/SGD_batchnorm	0.3205	0.3227	1.500k	Mon Jan 22, 01:07:02	12m 32s
●	conv/SGD_no_batchnorm	0.6014	0.5947	1.500k	Mon Jan 22, 01:31:20	12m 50s

Fig. 4. loss of CNN using SGD with or without batch norm



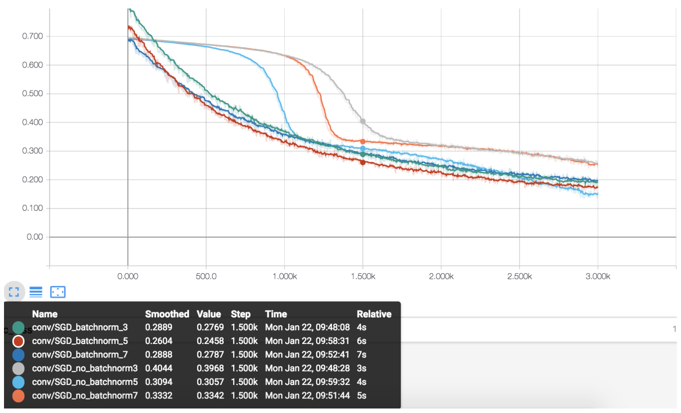
	Name	Smoothed	Value	Step	Time	Relative
●	conv/RMS_batchnorm	4.3502e-3	1.6550e-3	1.500k	Mon Jan 22, 01:49:53	29m 23s
●	conv/RMS_no_batchnorm	7.8081e-3	0.01191	1.500k	Mon Jan 22, 01:34:30	14m 27s

Fig. 5. loss of CNN using RMSProp with or without batch norm



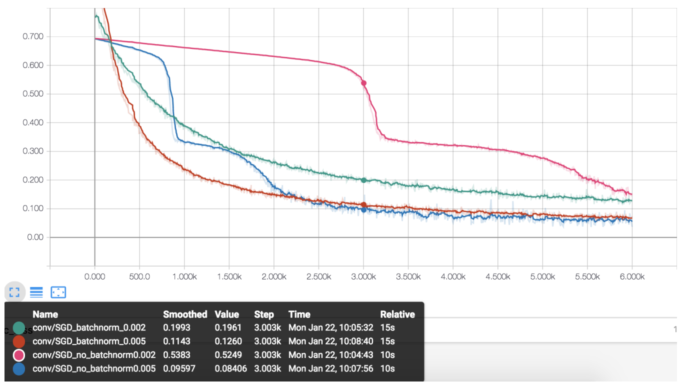
	Name	Smoothed	Value	Step	Time	Relative
●	conv/Adam_batchnorm	5.7761e-3	4.2966e-3	1.500k	Mon Jan 22, 01:49:20	30m 1s
●	conv/Adam_no_batchnorm	7.0340e-3	3.2439e-3	1.500k	Mon Jan 22, 01:33:41	14m 20s

Fig. 6. loss of CNN using Adam with or without batch norm



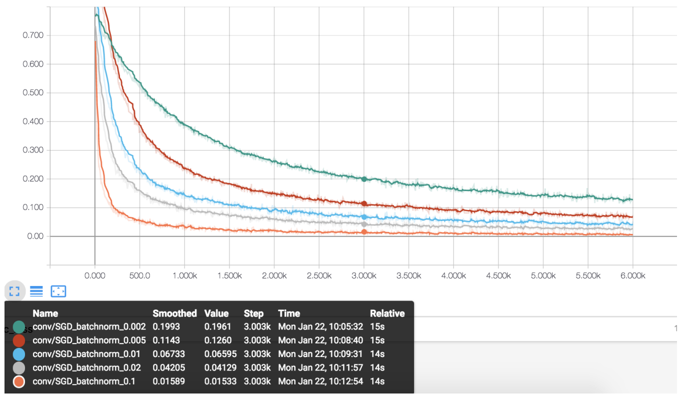
	Name	Smoothed	Value	Step	Time	Relative
●	conv/SGD_batchnorm_3	0.2889	0.2769	1.500k	Mon Jan 22, 09:48:08	4s
●	conv/SGD_batchnorm_5	0.2604	0.2458	1.500k	Mon Jan 22, 09:58:31	6s
●	conv/SGD_batchnorm_7	0.2888	0.2787	1.500k	Mon Jan 22, 09:52:41	7s
●	conv/SGD_no_batchnorm3	0.4044	0.3968	1.500k	Mon Jan 22, 09:48:28	3s
●	conv/SGD_no_batchnorm5	0.3094	0.3057	1.500k	Mon Jan 22, 09:59:32	4s
●	conv/SGD_no_batchnorm7	0.3332	0.3342	1.500k	Mon Jan 22, 09:51:44	5s

Fig. 7. loss of CNN with different layers with or without batch norm



	Name	Smoothed	Value	Step	Time	Relative
●	conv/SGD_batchnorm_0.002	0.1993	0.1961	3.003k	Mon Jan 22, 10:05:32	15s
●	conv/SGD_batchnorm_0.005	0.1143	0.1260	3.003k	Mon Jan 22, 10:08:40	15s
●	conv/SGD_no_batchnorm0.002	0.5383	0.5249	3.003k	Mon Jan 22, 10:04:43	10s
●	conv/SGD_no_batchnorm0.005	0.09597	0.08406	3.003k	Mon Jan 22, 10:07:56	10s

Fig. 8. loss of CNN with different learning rate with or without batch norm



	Name	Smoothed	Value	Step	Time	Relative
●	conv/SGD_batchnorm_0.002	0.1993	0.1961	3.003k	Mon Jan 22, 10:05:32	15s
●	conv/SGD_batchnorm_0.005	0.1143	0.1260	3.003k	Mon Jan 22, 10:08:40	15s
●	conv/SGD_batchnorm_0.01	0.06733	0.06595	3.003k	Mon Jan 22, 10:09:31	14s
●	conv/SGD_batchnorm_0.02	0.04205	0.04129	3.003k	Mon Jan 22, 10:11:57	14s
●	conv/SGD_batchnorm_0.1	0.01589	0.01533	3.003k	Mon Jan 22, 10:12:54	14s

Fig. 9. loss of CNN with different learning rate with batch norm

REFERENCES

[1] https://deepnotes.io/softmax-crossentropy
[2] Yes you should understand backprop - Andrej Karpathy - Medium
[3] https://en.wikipedia.org/wiki/Matrix_calculus
[4] https://deepnotes.io/batchnorm
[5] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International Conference on Machine Learning. 2015.
[6] TensorBoard: Visualizing Learning TensorFlow
[7] https://github.com/udacity/deep-learning/tree/master/batch-norm