Deep Learning and Practice

Lab 4: VA + BN + WI on Cifar-10

李韡

0556157

fm.bigballon@gmail.com

Introduction

In Lab 3, we learned how to build an NIN (Network in Network), and how to use Tensorflow (Keras or tflearn) to implement the model. In this Lab, we will be asked to use various activation functions, batch normalization, an weight initialization in NIN, and train it on Cifar-10 dataset.

I used Keras to implement these models and test the following **requirements**:

- Use **ReLU**, Leaky Rectifier Linear Unit(**LReLU**) [1], Exponential Linear Unit(**ELU**) [2] and **Maxout** activation function
- implement **weight initialization** method of He's paper [3]
- use **batch normalization** [4] in NIN model.
- Train NIN using three different activation functions and compare
  - 2(ReLU, ELU,) X 2(w/wo BN) X 2(w/wo weight initial)
  - 2(LReLU, Maxout) X 1(with BN) X 1(with weight initial)

Lab Description:

- Dying ReLUs

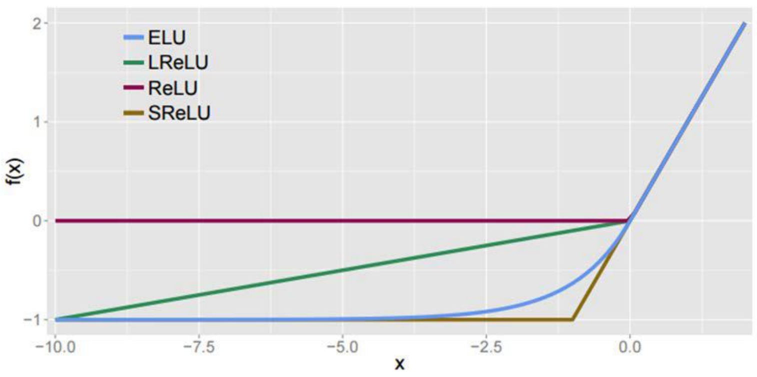


Figure 1: Dying ReLU activations

- Leaky Rectifier Linear Unit:

$$f(y_i) = \begin{cases} y_i & if \ \ y_i > 0 \\ 0.01y_i & if \ \ y_i \le 0 \end{cases}$$

$$f'(y_i) = \begin{cases} 1 & if \ \ y_i > 0 \\ 0.01 & if \ \ y_i \le 0 \end{cases}$$

- Exponential Linear Unit:

$$f(y_i) = \begin{cases} y_i & if \;\; y_i > 0 \\ \alpha(\exp(y_i) - 1) & if \;\; y_i \le 0 \end{cases}$$

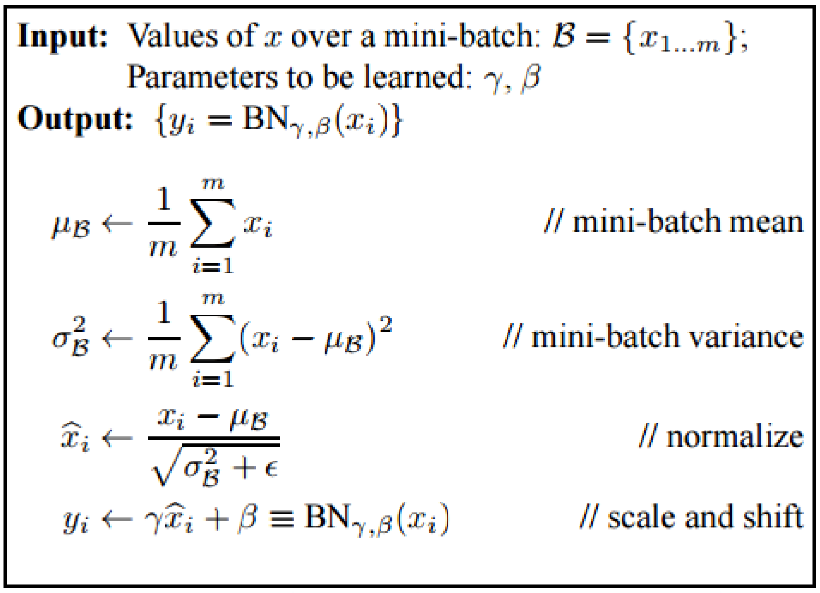
$$f'(y_i) = \begin{cases} 1 & if \;\; y_i > 0 \\ f(y_i) + \alpha & if \;\; y_i \le 0 \end{cases}$$

$\alpha = 1$ in the experiments of the original paper

- He's Weight initialization:

WI $=$ **normal**$(\sqrt{2/n_l})$, where n_l is $n_l = k^2 c$, input size $\mathrm{k} \times \mathrm{k}$ with c filters at $l\text{–}th \;\; layer$

- Batch Normalization:



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 2: Batch Normalization algorithms

In CNN:

Original CNN: $z = f(\boldsymbol{W}x + b)$

CNN with BN: $z = f(BN(\boldsymbol{W}x + b))$

Benefit:

Accelerating training

Regularization

Experiment setup

Architecture Details:

Table 1: ConvPool-CNN Architecture

Conv 1	5x5 conv. 192 pad = 2 stride = 1	w/wo BN	ReLU/ELU/LReLU
Mlp 1	1x1 conv. 160 pad = 0 stride = 1	w/wo BN	ReLU/ELU/LReLU
Mlp 2	1x1 conv. 96 pad = 0 stride = 1	w/wo BN	ReLU/ELU/LReLU
Pool 1	3x3 max pooling, stride = 2		
	Dropout 0.5		
Conv 2	5x5 conv. 192 pad = 2 stride = 1	w/wo BN	ReLU/ELU/LReLU
Mlp 2	1x1 conv. 192 pad = 0 stride = 1	w/wo BN	ReLU/ELU/LReLU
Mlp 2	1x1 conv. 192 pad = 0 stride = 1	w/wo BN	ReLU/ELU/LReLU
Pool 2	3x3 max pooling, stride = 2		
	Dropout 0.5		
Conv 3	3x3 conv. 192 pad = 1 stride = 1	w/wo BN	ReLU/ELU/LReLU
Mlp 3	1x1 conv. 192 pad = 0 stride = 1	w/wo BN	ReLU/ELU/LReLU
Mlp 3	1x1 conv. 10 pad = 0 stride = 1	w/wo BN	ReLU/ELU/LReLU
Global Pool	8x8 average pooling, stride =1		
	Softmax		

- **Training hyperparameters:**
  - Method: SGD with Nesterov momentum momentum=0.9, nesterov=True
  - Data preprocessing: Color normalization
  - Weight initialization: **He's WI**
  - batch size: 128 (391 iterations for each epoch)
  - Total epochs: 164
  - Loss function: cross-entropy
  - Initial learning rate: 0.1, divide by 10 at 81, 122 epoch
  - Weight Decay: kernel_regularizer=keras.regularizers.l2(0.0001)
  - Dropout: 0.5

- **Data augmentation:**
  - Translation: Pad **4** zeros in each side and random cropping back to 32x32 size
  - Horizontal flipping with probability 0.5

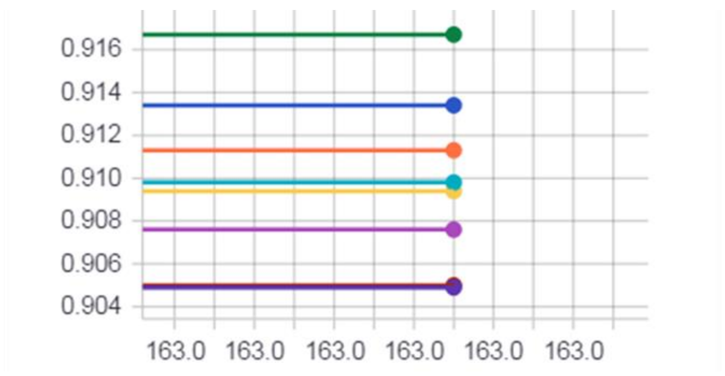
Result

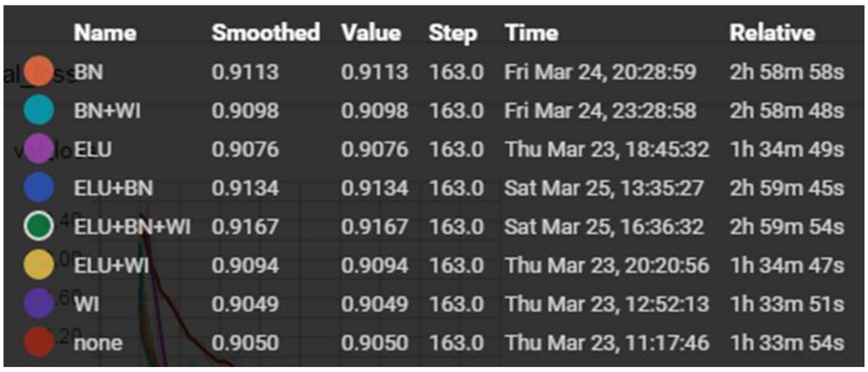
■ Final Accuracy (test error)

I just test all of the 8 models, it cost about 3 hours(2h59m) to train a model (ELU+BN+WI), (ELU+BN), (BN) and (BN+WI), 1h35min for the other models.
All of the 8 CNN models' final test accuracy are more than 90%.
The **ELU+BN+WI** model gets **91.67%**, and the **ELU+BN** model gets **91.34%**, it seems nice. We can learn that as the data flows through a deep network, the weights and parameters adjust those values, sometimes making the data too big or too small again. By normalizing the data in each mini-batch, this problem is largely avoided.
See the table and figures below for detailed results

Table 2: Test accuracy for all 8 models

ELU+BN+WI	**91.67%**
ELU+BN	**91.34%**
BN	**91.13%**
BN+WI	**90.98%**
ELU+WI	**90.94%**
ELU	**90.76%**
None	**90.50%**
WI	**90.49%**





	Name	Smoothed	Value	Step	Time	Relative
●	BN	0.9113	0.9113	163.0	Fri Mar 24, 20:28:59	2h 58m 58s
●	BN+WI	0.9098	0.9098	163.0	Fri Mar 24, 23:28:58	2h 58m 48s
●	ELU	0.9076	0.9076	163.0	Thu Mar 23, 18:45:32	1h 34m 49s
●	ELU+BN	0.9134	0.9134	163.0	Sat Mar 25, 13:35:27	2h 59m 45s
○	ELU+BN+WI	0.9167	0.9167	163.0	Sat Mar 25, 16:36:32	2h 59m 54s
●	ELU+WI	0.9094	0.9094	163.0	Thu Mar 23, 20:20:56	1h 34m 47s
●	WI	0.9049	0.9049	163.0	Thu Mar 23, 12:52:13	1h 33m 51s
●	none	0.9050	0.9050	163.0	Thu Mar 23, 11:17:46	1h 33m 54s

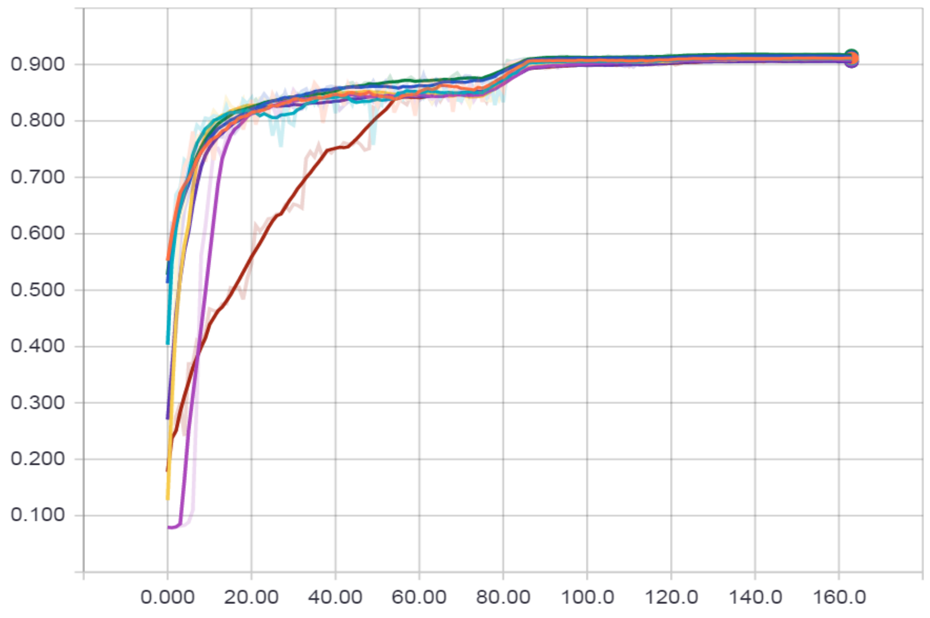
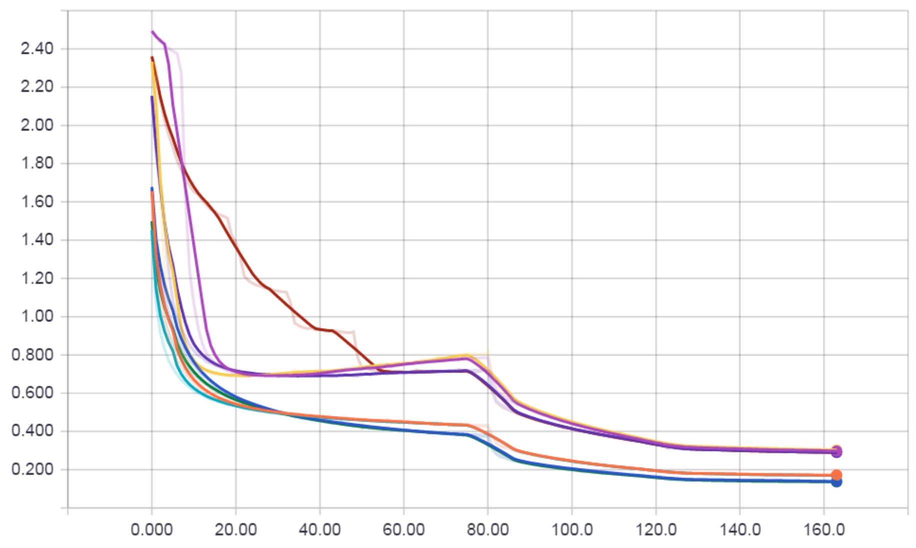
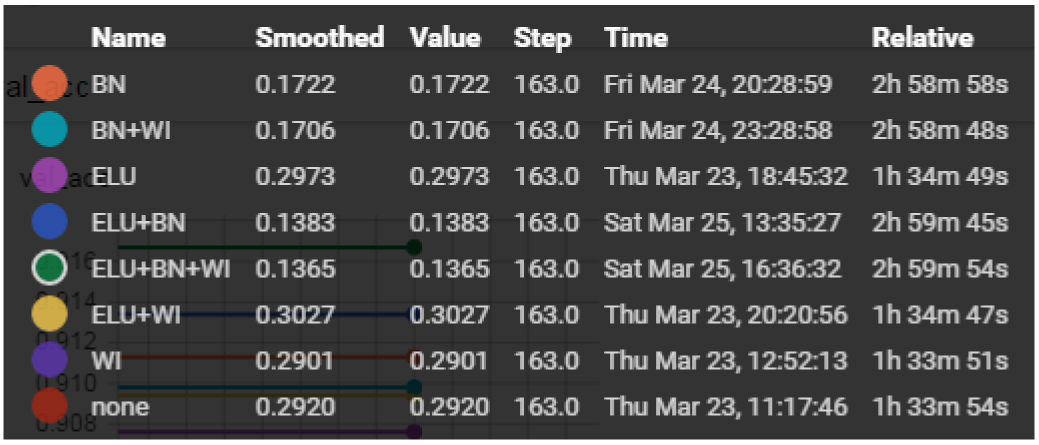


Figure 3: Test accuracy curve for all 8 models

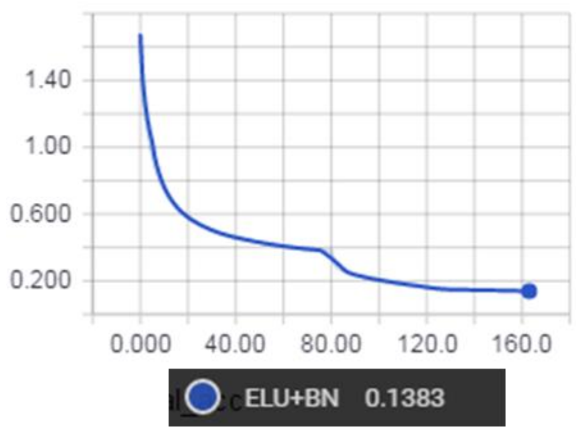
- **Training loss curve:**





	Name	Smoothed	Value	Step	Time	Relative
●	BN	0.1722	0.1722	163.0	Fri Mar 24, 20:28:59	2h 58m 58s
●	BN+WI	0.1706	0.1706	163.0	Fri Mar 24, 23:28:58	2h 58m 48s
●	ELU	0.2973	0.2973	163.0	Thu Mar 23, 18:45:32	1h 34m 49s
●	ELU+BN	0.1383	0.1383	163.0	Sat Mar 25, 13:35:27	2h 59m 45s
●	ELU+BN+WI	0.1365	0.1365	163.0	Sat Mar 25, 16:36:32	2h 59m 54s
●	ELU+WI	0.3027	0.3027	163.0	Thu Mar 23, 20:20:56	1h 34m 47s
●	WI	0.2901	0.2901	163.0	Thu Mar 23, 12:52:13	1h 33m 51s
●	none	0.2920	0.2920	163.0	Thu Mar 23, 11:17:46	1h 33m 54s

Figure 4: Training loss curve for all 8 models



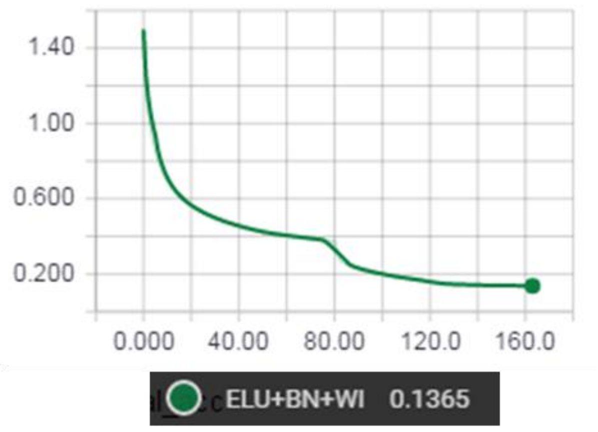
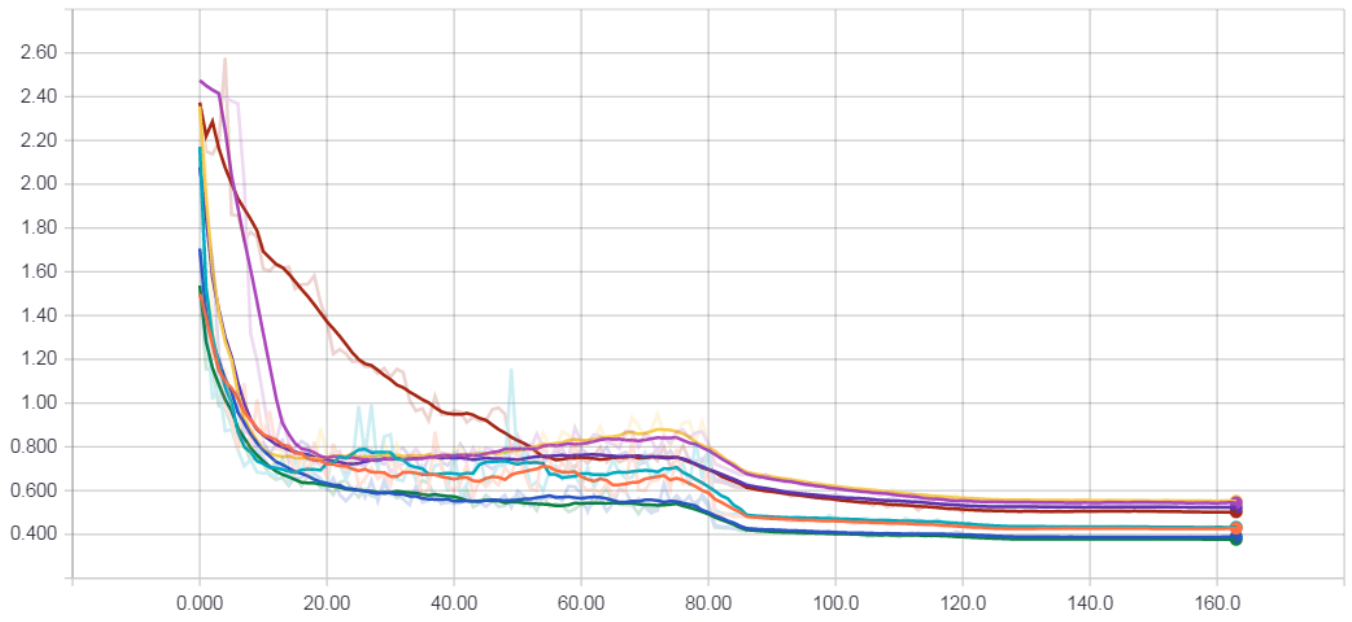
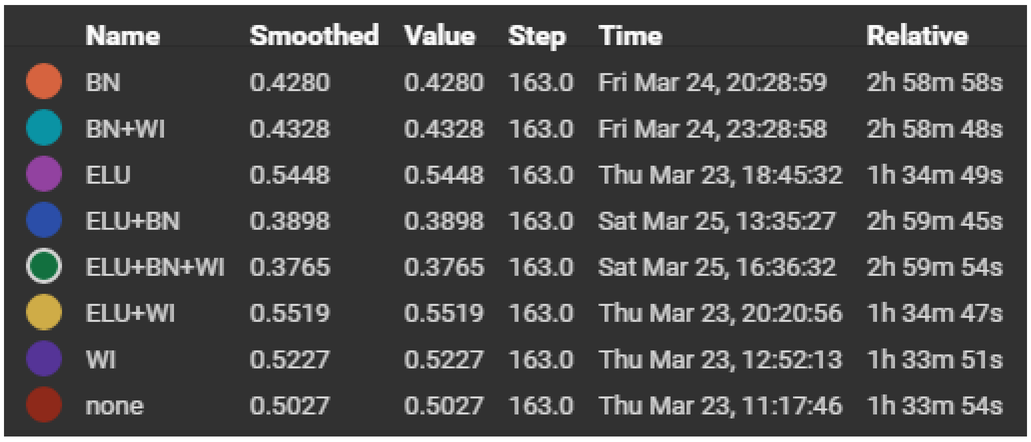


Figure 5: The smallest training loss error

■ **Test loss curve**





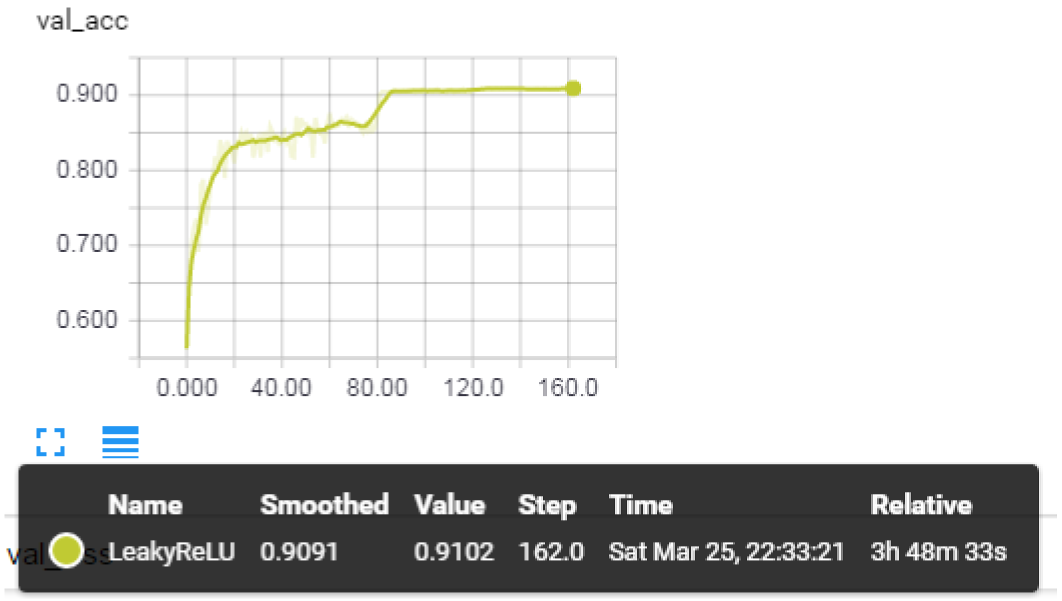
	Name	Smoothed	Value	Step	Time	Relative
●	BN	0.4280	0.4280	163.0	Fri Mar 24, 20:28:59	2h 58m 58s
●	BN+WI	0.4328	0.4328	163.0	Fri Mar 24, 23:28:58	2h 58m 48s
●	ELU	0.5448	0.5448	163.0	Thu Mar 23, 18:45:32	1h 34m 49s
●	ELU+BN	0.3898	0.3898	163.0	Sat Mar 25, 13:35:27	2h 59m 45s
●	ELU+BN+WI	0.3765	0.3765	163.0	Sat Mar 25, 16:36:32	2h 59m 54s
●	ELU+WI	0.5519	0.5519	163.0	Thu Mar 23, 20:20:56	1h 34m 47s
●	WI	0.5227	0.5227	163.0	Thu Mar 23, 12:52:13	1h 33m 51s
●	none	0.5027	0.5027	163.0	Thu Mar 23, 11:17:46	1h 33m 54s

Figure 6: Test loss curve for all 8 models

Other experiments

■ **LeakyReLU**

I change the activation function to LeakyReLU and set the alpha=**0.01**, after training (about 3h50min), its test accuracy is **90.91%**



	Name	Smoothed	Value	Step	Time	Relative
●	LeakyReLU	0.9091	0.9102	162.0	Sat Mar 25, 22:33:21	3h 48m 33s

Figure 7: The accuracy for LeakyReLU

Discussion

In lab4, we used lots of tricks and method to improve our model's accuracy.

BN (Batch normalization) faster learning and higher overall accuracy. The improved method also allows us to use a higher learning rate, potentially providing another boost in speed. (Initial learning rate 0.1 is too large to train if we did not use special tricks in Lab3, but BN can avoid this problem, it makes loss drops very fast.)

WI (ReLU + He's weight initial + without BN) got the worst accuracy compared with other models. And sometimes WI may not converge.

Different activation function will have different effect, **ELU** and **LeakyReLU** have better performance than **ReLU**. As for maxout, I tried to use it, but it seems that my code doesn't work.

References:

[1] Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013, June). Rectifier nonlinearities improve neural network acoustic models. In Proc. ICML (Vol. 30, No. 1).
[2] Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289.
[3] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1026-1034).
[4] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
[5] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. C., & Bengio, Y. (2013). Maxout networks. ICML (3), 28, 1319-1327.