Lab 4: Various activations + Batch normalization + Weight Initialization

Lab Objective:

In this lab, you will be asked to use various activation functions, batch normalization, and weight initialization in NIN, and train it on Cifar-10 dataset.

Turn in:

1. Experiment Report (3/28(二))
2. Demo date (3/28(二))

Requirements:

- Use "Exponential Linear Unit" (ELU) [2] 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)

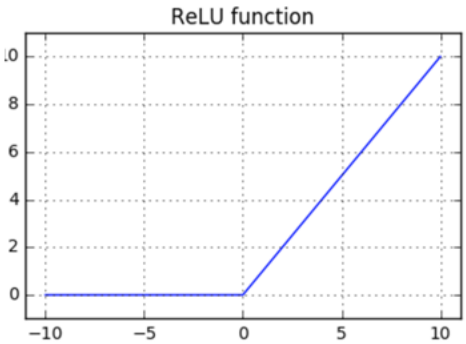
    (It may not converge: ReLU + He's weight initial + without BN)

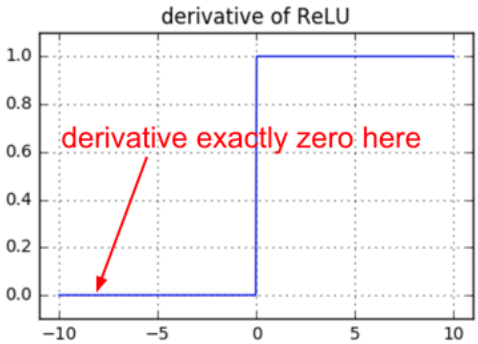
Environment:

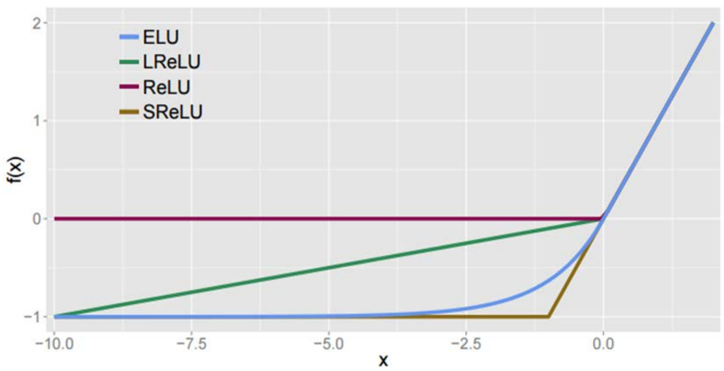
Use Cifar-10 as same as Lab 3.

Lab Description:

- Dying ReLUs







- Leaky Rectifier Linear Unit [1]

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

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

- Exponential Linear Unit [2]

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

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

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

tf.nn.elu(features, name=None)

https://www.tensorflow.org/api_docs/python/tf/nn/elu

- Weight initialization [3]

All you need is a good initial?

Initial from a random normal is really bad.

How to keep that the output is unit variance (Var=1) ?

Weight initialization = $\text{normal}(0, \sqrt{2/n_l})$, where n_l is $n_l = k^2 c$, input size $\text{k} \times \text{k}$ with c filters at $l - th\ layer$

Example: 1-st layer of NIN used 192 of 5×5 convolutional filters

Those convolutional filter's weights were initialized by He's method.

The weights should be initialized from the distribution of $N(0, \sqrt{2/(5 \times 5 \times 192)})$ equal to $N(0, 0.0204)$.

- Batch Normalization [4]

Idea: prevent internal covariance shift

Method: normalize features in every layer over a mini-batch

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

In CNN:

Original CNN $\quad z = f(\mathbf{W}x + b)$

with BN: $\quad z = f(\mathrm{BN}(\mathbf{W}x))$

Benefit:

1. Accelerating training
2. Regularization

BN usage:

`tf.contrib.layers.batch_norm(*args, **kwargs)`

Args:

- `inputs` : a tensor with 2 or more dimensions, where the first dimension has `batch_size` . The normalization is over all but the last dimension if `data_format` is `NHWC` and the second dimension if `data_format` is `NCHW` .
- `decay` : decay for the moving average. Reasonable values for `decay` are close to 1.0, typically in the multiple-nines range: 0.999, 0.99, 0.9, etc. Lower `decay` value (recommend trying `decay` =0.9) if model experiences reasonably good training performance but poor validation and/or test performance. Try zero_debias_moving_mean=True for improved stability.
- `center` : If True, add offset of `beta` to normalized tensor. If False, `beta` is ignored.
- `scale` : If True, multiply by `gamma` . If False, `gamma` is not used. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling can be done by the next layer.
- `epsilon` : small float added to variance to avoid dividing by zero.
- `activation_fn` : activation function, default set to None to skip it and maintain a linear activation.
- `param_initializers` : optional initializers for beta, gamma, moving mean and moving variance.
- `updates_collections` : collections to collect the update ops for computation. The updates_ops need to be executed with the train_op. If None, a control dependency would be added to make sure the updates are computed in place.
- `is_training` : whether or not the layer is in training mode. In training mode it would accumulate the statistics of the moments into `moving_mean` and `moving_variance` using an exponential moving average with the given `decay` . When it is not in training mode then it would use the values of the `moving_mean` and the `moving_variance` .

Implementation Details:

- Training Hyperparameters:
  - Method: SGD with momentum
  - Mini-batch size: 128 (391 iterations for each epoch)
  - Total epochs: 164, momentum 0.9
  - Initial learning rate: 0.1, divide by 10 at 81, 122 epoch
  - Loss function: cross-entropy
  - Use data augmentation

- You should compare all the condition
  - 3 (activations) x 2 (w/wo BN) x 2 (w/wo weight initialization)
- Data augmentation parameters:
  - Translation: Pad **4** zeros in each side and random cropping back to 32x32 size
  - Horizontal flipping: With probability **0.5**

Methodology:

- 91.67% accuracy with data augmentation in my implementation (NIN+BN)

Extra Bonus:

- Implement “Maxout” [5] activation function and apply to NIN
- Use ELU

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.

Report Spec: [black: Demo, Gray: No Demo]

1. Introduction (15%)
2. Experiment setup (15%)
   - The detail of your model
   - Report all your training hyper-parameters
3. Result
   - The comparison of 2(RELU, ELU)×2(w/wo BN)×2(w/wo weight initial)
     - Final Test error (10%, 15%)
     - Training loss curve (you need to record training loss every epoch) (10%, 15%)
     - Test error curve (you need to record test error every epoch) (10%, 15%)
4. Discussion (20%, 25%)

Demo (20%) [抽 20 人 DEMO]

-------實驗結果標準 (with data augmentation)----

Accuracy: (92.0~90.0)% = 100%

Accuracy: (90.0~87.0)% = 90%

Accuracy: (87.0~84.0)% = 80%

Accuracy: below 84.0% = 70%

Accuracy: 10% = 0%

評分標準: 40%*實驗結果 + 60%*(報告+DEMO)