بسم الله الرحمن الرحیم

Deep Learning

دانشگاه صنعتی مالک اشتر

Mohammad Ali Keyvanrad

Lecture 4:A Review of Artificial Neural Networks (3)

OUTLINE

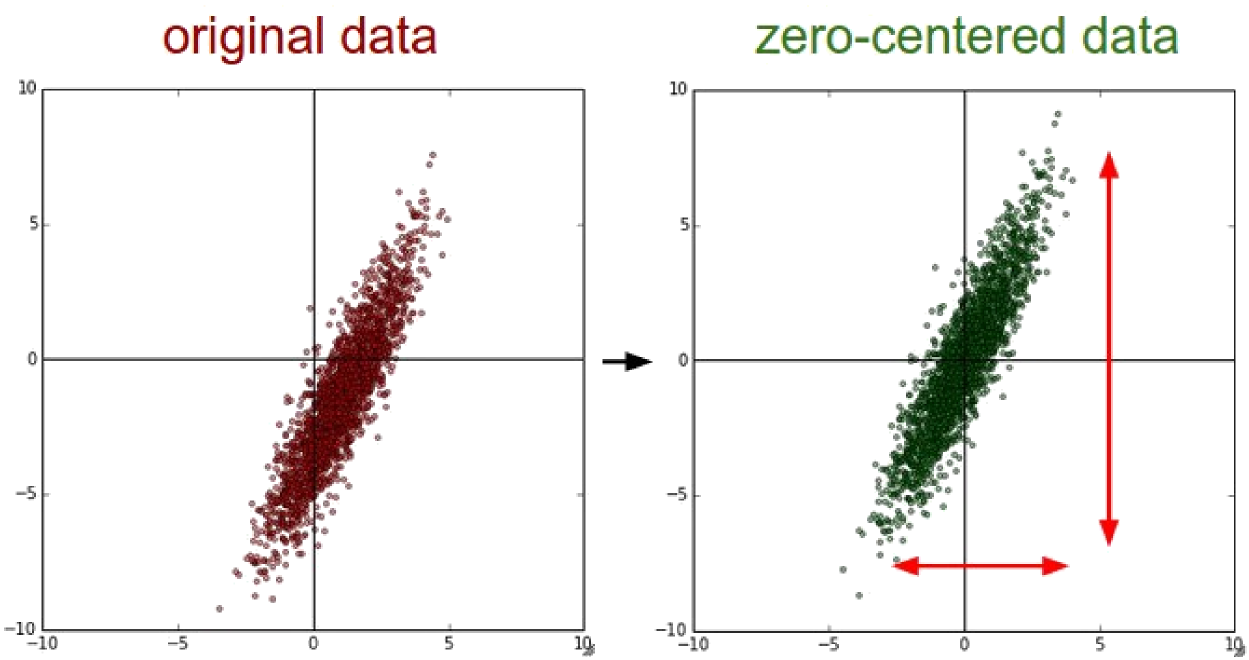
- Data Preprocessing
- Weight Initialization
- Batch Normalization
  - Normalization via Mini-Batch Statistics

OUTLINE

- **Data Preprocessing**
- Weight Initialization
- Batch Normalization
  - Normalization via Mini-Batch Statistics

Data Preprocessing

- Mean subtraction
  - Subtracting the mean across every individual feature



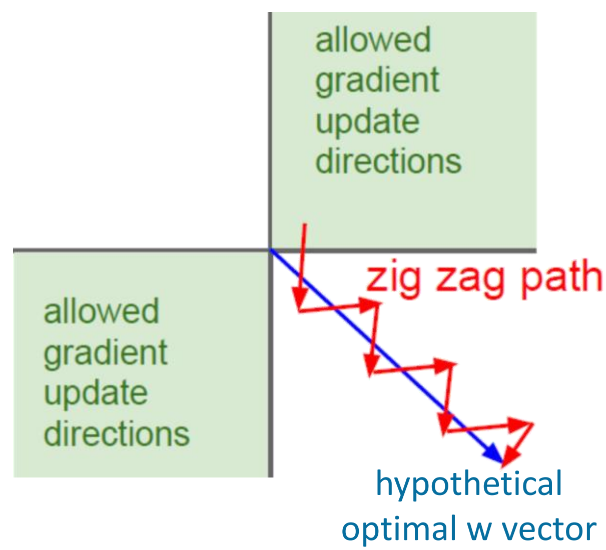

```
X -= np.mean(X, axis=0)
```

Assume $X\ [N \times D]$ is data matrix, each example in a row

zero-mean data

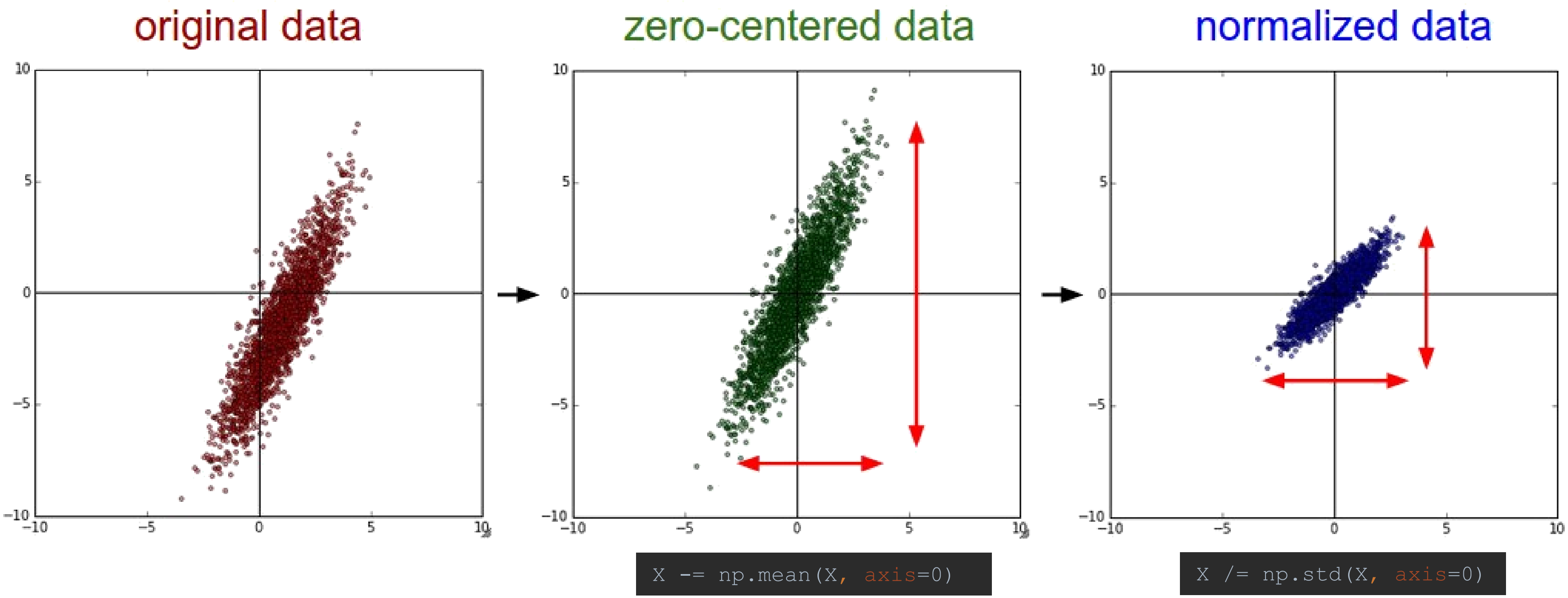
- Consider what happens when the input to a neuron is always positive
  - The gradient on the weights $\boldsymbol{w}$ become either all be positive, or all negative.
  - introduce zig-zagging dynamics in the gradient updates

$$f\left(\sum_i w_i x_i + b\right)$$



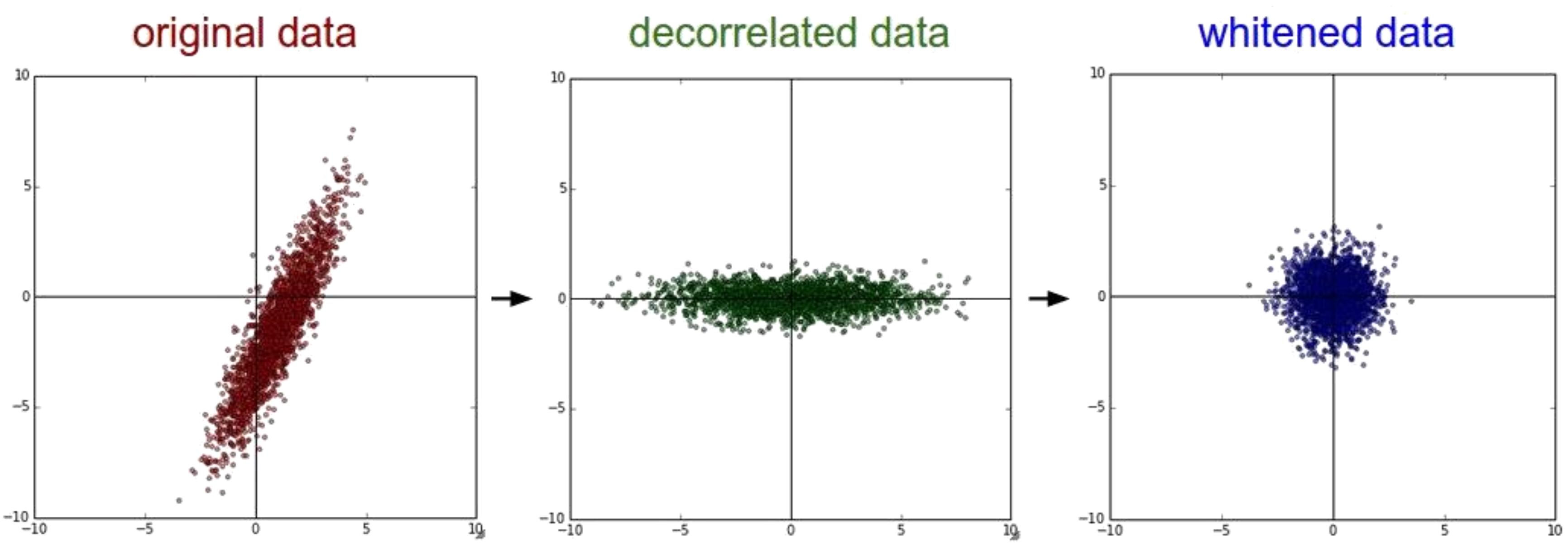
Data Preprocessing

- Normalization
  - Normalizing the data dimensions so that they are of approximately the same scale (by std or min/max)



Data Preprocessing

- PCA and Whitening
  - PCA: the data rotated into the eigenbasis and decorrelates the data
  - Whitening: Each dimension is scaled by the eigenvalues



TLDR; In practice

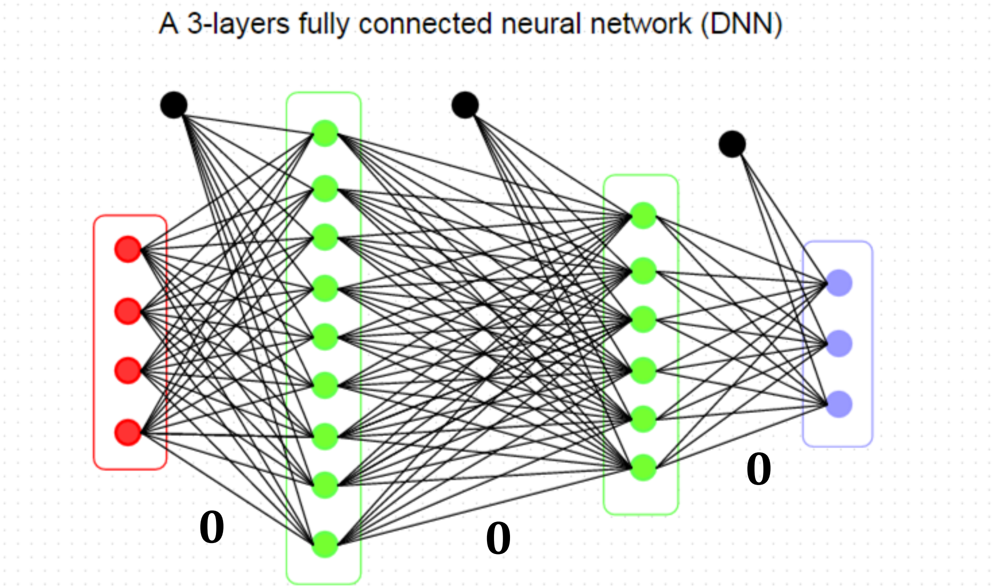
- It is very important to zero-center the data.
- It is common normalization of data.
- Not common using PCA or whitening.
- Case study : CIFAR-10 example with [32,32,3] images
  - AlexNet: Subtract the mean image (mean: [32,32,3] array)
  - VGGNet: Subtract per-channel mean (mean: 3 numbers)
- **Common pitfall**
  - The preprocessing must only be computed on the training data.
  - Then applied to the validation / test data.

OUTLINE

- Data Preprocessing
- **Weight Initialization**
- Batch Normalization
  - Normalization via Mini-Batch Statistics

Weight Initialization

- **Zero initialization**
  - Every neuron in the network computes the same output
  - They will also all compute the same gradients
  - No source of asymmetry between neurons



Weight Initialization

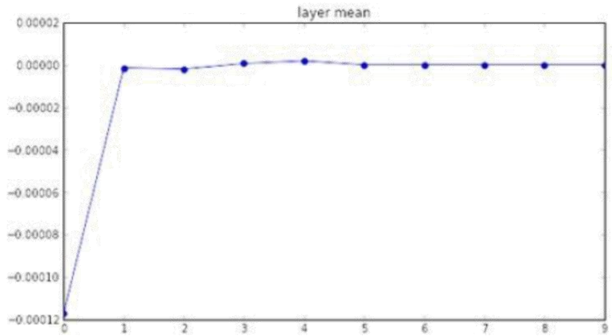
- **Small random numbers**
  - symmetry breaking
  - $W = 0.01 * np.random.randn(D, H)$
  - This could greatly diminish the "gradient signal" flowing backward through a network
    - This could become a concern for deep networks.
  - All activations become zero!

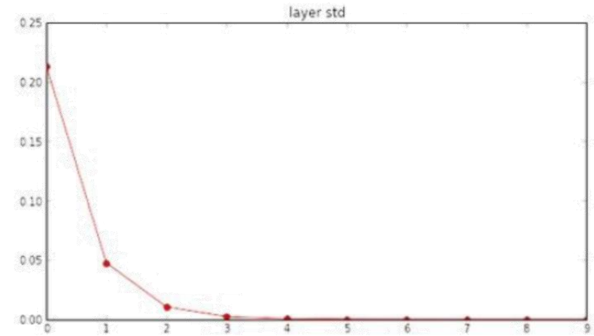
Weight Initialization

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```

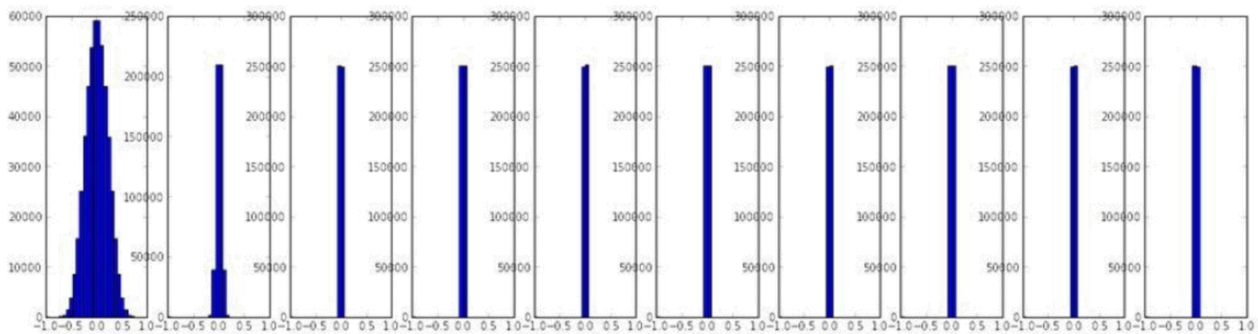
Example: 10-layer net with 500 neurons on each layer

- Activation Function: tanh non-linearities
- Initialization: $W = 0.01 * np.random.randn(.)$





All activations become zero!



W = **1** $* \ np.random.randn(.)$ → Almost all neurons completely saturated

Weight Initialization

- **Calibrating the variances**
  - Randomly initialized neuron has a variance that grows with the number of inputs
  - An Idea: $w = np.random.randn(n) \; / \; sqrt(n)$
  - n is the number of its inputs

Weight Initialization

- Consider the inner product $s = \sum_i^n w_i x_i$ between the weights w and input x

[Glorot et al., 2010]

$$\text{Var}(s) = Var\left(\sum_i^n w_i x_i\right)$$

$$\text{Var}\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n \text{Var}(X_i).$$

$$= \sum_i^n Var(w_i x_i)$$

$$\text{Var}(XY) = [E(X)]^2 \,\text{Var}(Y) + [E(Y)]^2 \,\text{Var}(X) + \text{Var}(X)\,\text{Var}(Y).$$

$$= \sum_i^n [E(w_i)]^2 \, Var(x_i) + E[(x_i)]^2 \, Var(w_i) + Var(x_i)\, Var(w_i)$$

$$E[x_i] = E[w_i] = 0$$

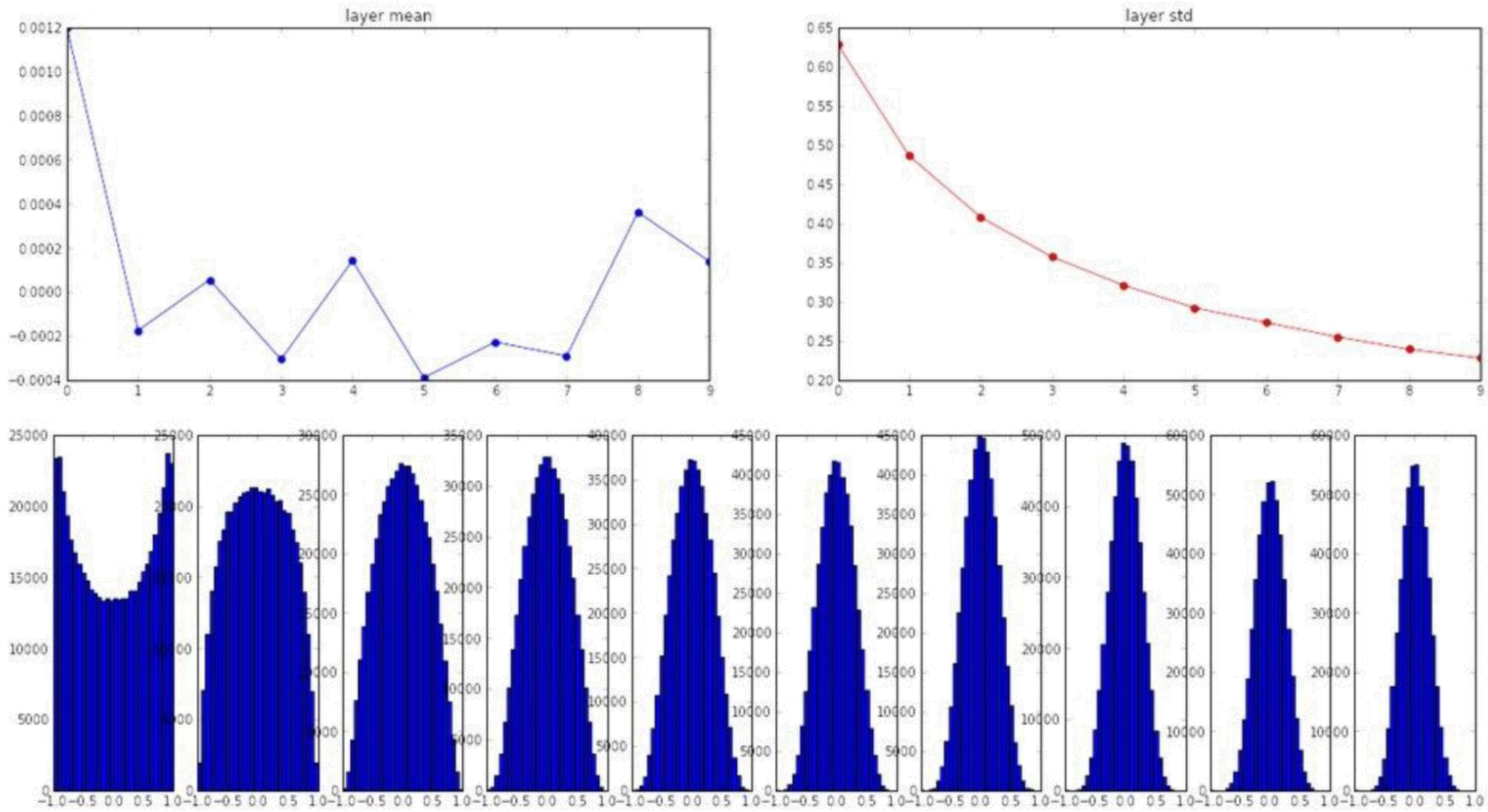
$$= \sum_i^n Var(x_i)\, Var(w_i)$$

$$= \big(n\, Var(w)\big)\, Var(x)$$

$$\xrightarrow{\text{Var}(aX) = a^2\, Var(x)} a = \sqrt{1/n} \rightarrow w = np.random.randn(n) \,/\, sqrt(n)$$

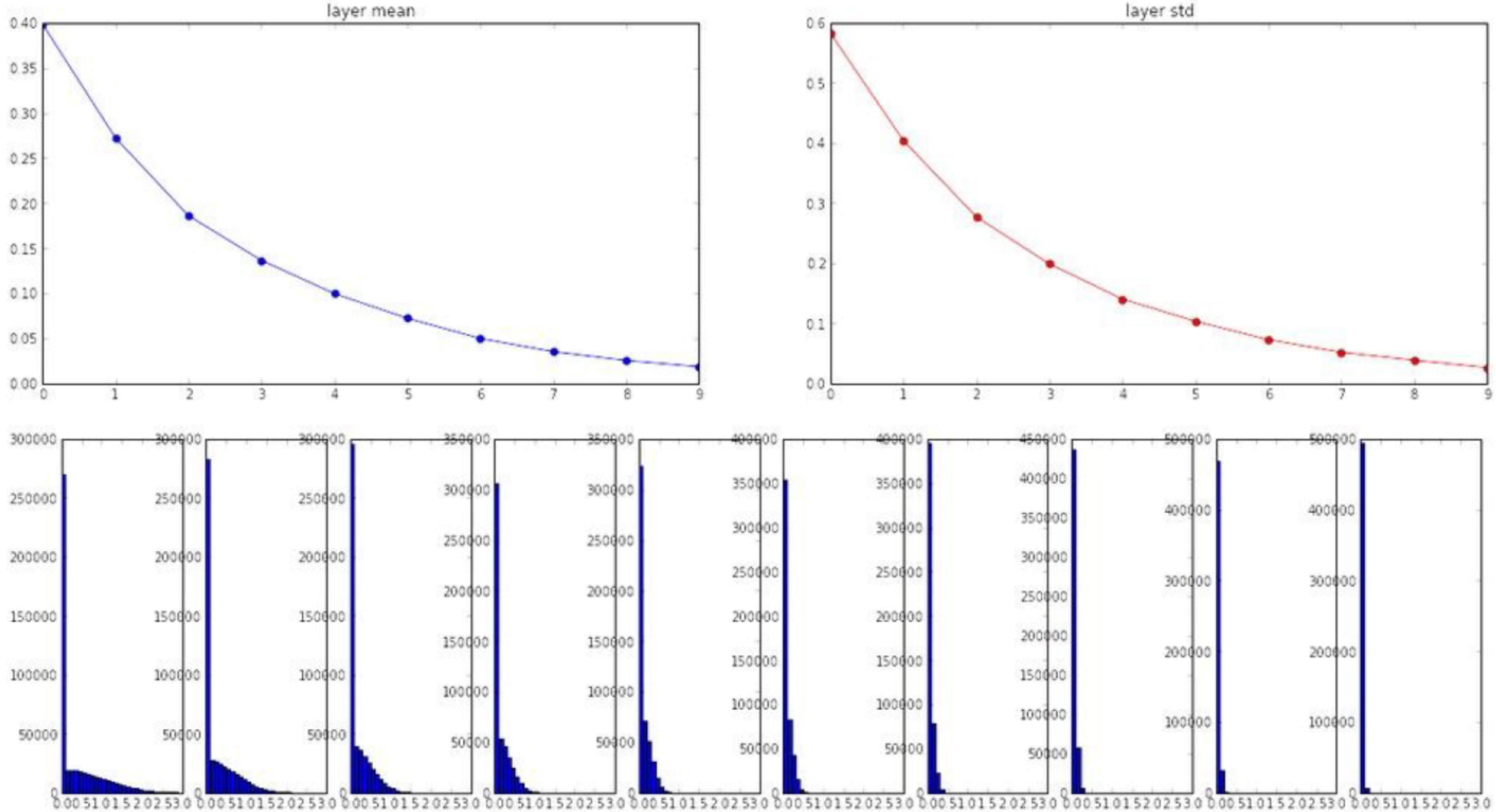
Weight Initialization

- Using $w = np.random.randn(n) / sqrt(n)$



Weight Initialization

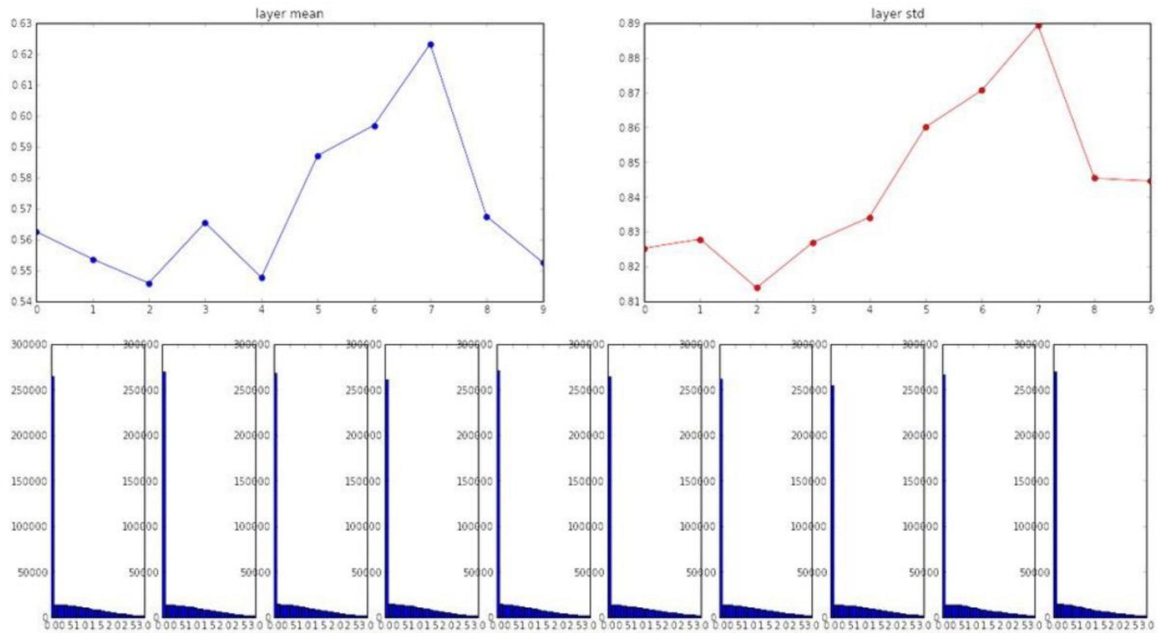
- Using $w = np.random.randn(n) \,/\, sqrt(n)$
  - when using the ReLU activation function

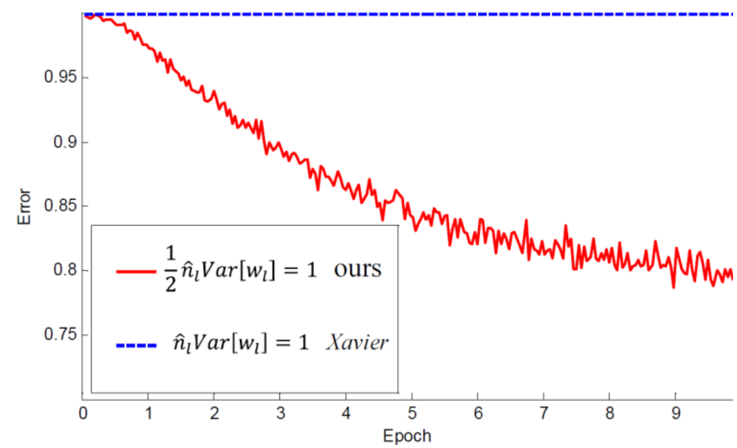


Weight Initialization

- Using $w = np.random.randn(n) \ / \ sqrt(n/2)$

[He et al., 2015]





TLDR; In practice

- Proper initialization is an active area of research
  - Mishkin, Dmytro, and Jiri Matas. "All you need is a good init." (2015).
  - Krähenbühl, Philipp, et al. "Data-dependent initializations of convolutional neural networks." (2015).
- Recommendation
  - Using ReLU units
  - Using $w = np.random.randn(n) \,/\, sqrt(n/2)$ for initialization

OUTLINE

- Data Preprocessing
- Weight Initialization
- **Batch Normalization**
  - Normalization via Mini-Batch Statistics

Batch Normalization

- Internal Covariate Shift
  - Change in the distribution of network activations due to the change in network parameters during training.
  - We seek to reduce the internal covariate shift. By fixing the distribution of the layer inputs x.
- The network training converges faster if its inputs are whitened. (LeCun et al., 1998b; Wiesler & Ney,2011)

The full whitening of each layer's inputs is costly and not everywhere differentiable.

OUTLINE

- Data Preprocessing
- Weight Initialization
- Batch Normalization
  - **Normalization via Mini-Batch Statistics**

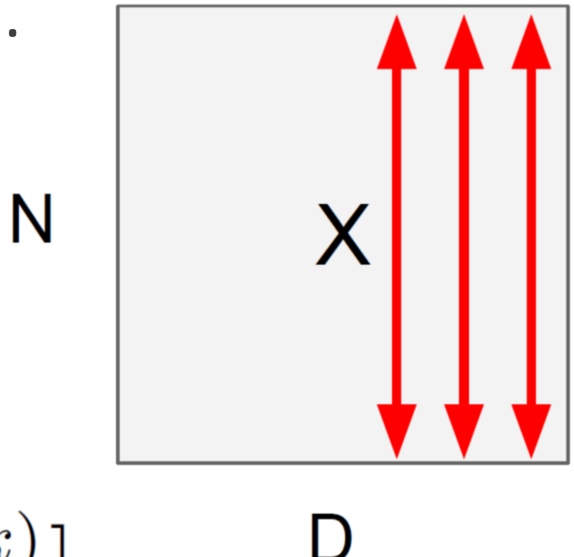
Normalization via Mini-Batch Statistics

- Two necessary simplifications
  - Normalizing each scalar feature independently
  - Using mini-batches in stochastic gradient training
- you want unit gaussian activations?
  - Consider a batch of activations at some layer.
  - To make each dimension unit gaussian, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Normalization via Mini-Batch Statistics

- Normalization
  1. Compute the empirical mean and variance independently for each dimension.



  2. Normalize

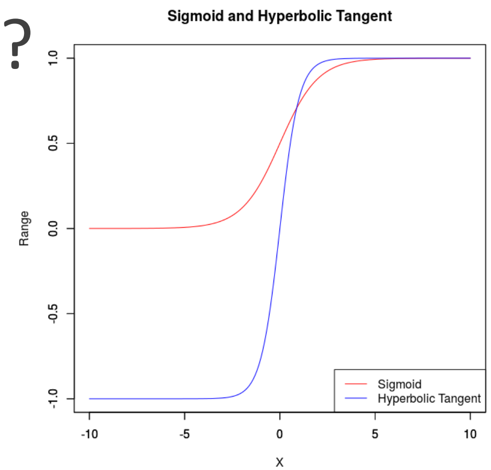
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

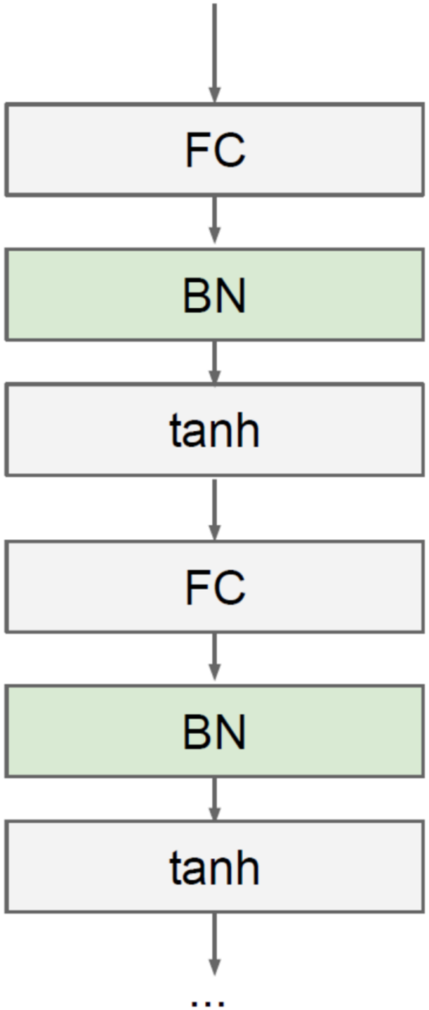
Normalization via Mini-Batch Statistics

- **Batch Normalization (BN)** usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Can BN help sig() or tanh()?





Normalization via Mini-Batch Statistics

Is the power of the network diminished?

- we introduce, for each activation $x(k)$, a pair of parameters $\gamma(k)$, $\beta(k)$, which scale and shift the normalized value.
- allow the network to squash the range if it wants.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \qquad y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

- We could recover the original activations, if that were the optimal thing to do.

$$\beta^{(k)} = \mathrm{E}[x^{(k)}] \quad \gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

Normalization via Mini-Batch Statistics

- Batch Normalizing Transform

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

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

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

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

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

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

Normalization via Mini-Batch Statistics

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout

Normalization via Mini-Batch Statistics

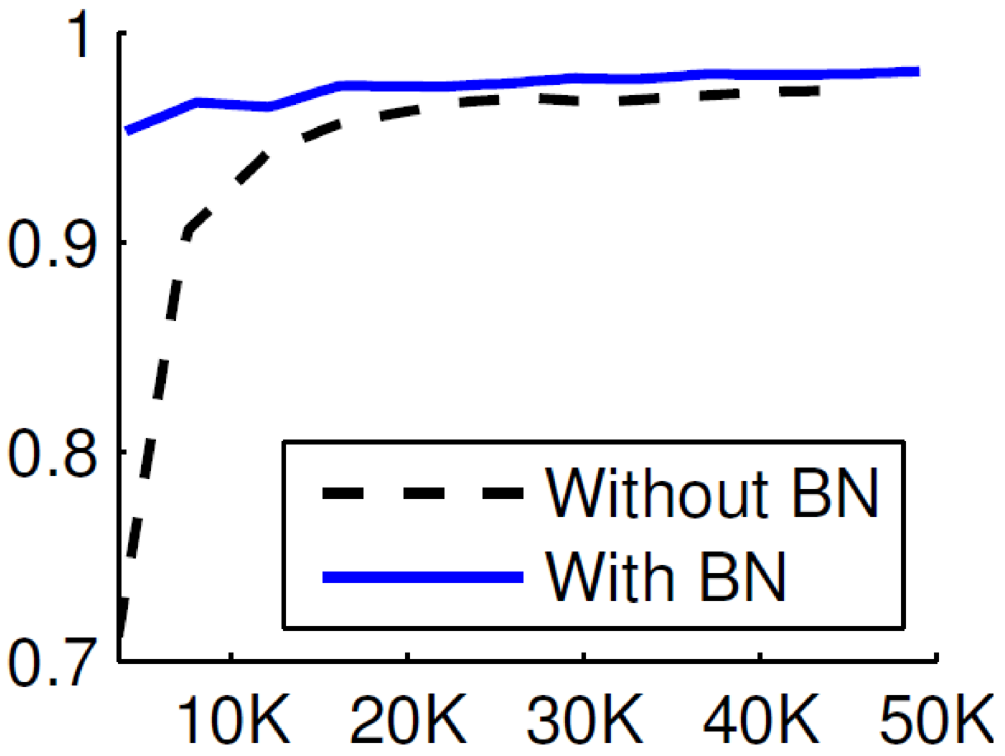
- At test time BN layer, functions differently
  - The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.
    - Can be estimated during training with moving averages

$$\bar{x}_n = \frac{(n-1)\,\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$\sigma_n^2 = \frac{(n-1)\,\sigma_{n-1}^2 + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)}{n}.$$

Normalization via Mini-Batch Statistics

- In MNIST Dataset Sig() activation function



References

- Stanford "Convolutional Neural Networks for Visual Recognition" course (Training Neural Networks, part I)
- Stanford "Convolutional Neural Networks for Visual Recognition" course (Neural Nets notes 2)
- Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *International Conference on Machine Learning*. 2015.

رسول خدا (ص):

إذا وُقِعَ فی الرَّجُلِ وَأَنْتَ فی مَلَأٍ فَکُنْ لِلرَّجُلِ ناصِراً وَلِلْقَوْمِ زاجِراً وَقُمْ عَنْهُم.

اگر در میان جمعی بودی و از کسی غیبت شد، آن فرد را یاری کن و آن جمع را از بدگویی بازدار و از میانشان برخیز و برو.

If you were among a group of people who are backbiting someone, you should help the backbitten person, prevent them from backbiting him, and leave them.

کنز العمّال، ج ۳، ص ۵۸۶، ح ۸۰۲۸