Indian Institute of Information Technology, Allahabad

Data Preparation and Weight Initialization

By

Dr. Shiv Ram Dubey
Assistant Professor
Computer Vision And Biometrics Lab (CVBL)
Department Of Information Technology
Indian Institute Of Information Technology, Allahabad

Email: srdubey@iiita.ac.in   Web: https://profile.iiita.ac.in/srdubey/

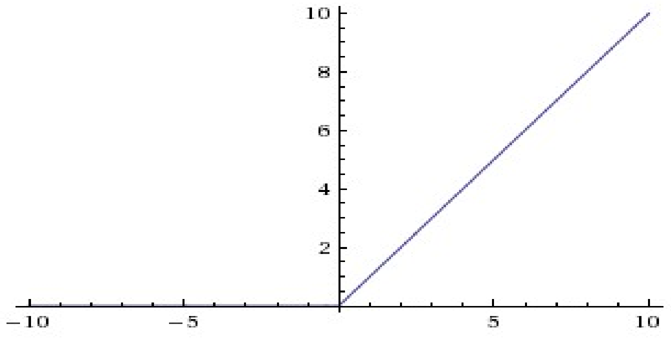
DISCLAIMER

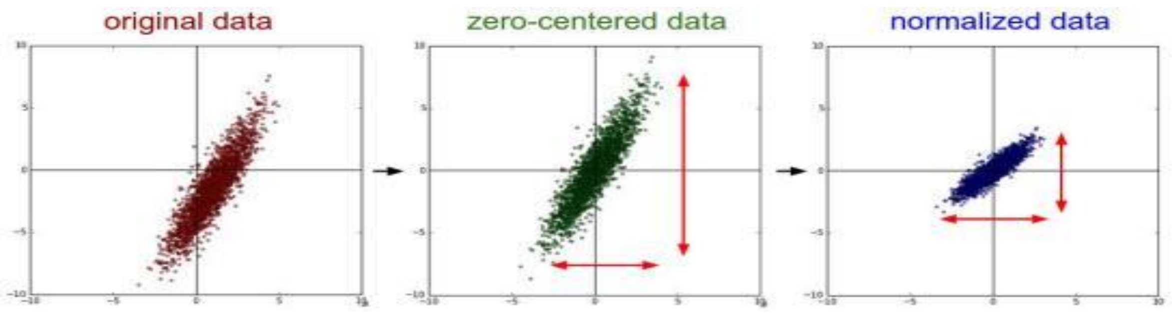
The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

This Class

Data and Weight Setup

- Dataset Preparation
- Data Preprocessing
- Weight Initialization





Dataset Preparation
Train/Val/Test sets

In General People Do: Train/Test

- Split data into train and test,
- Choose hyperparameters that work best on test data

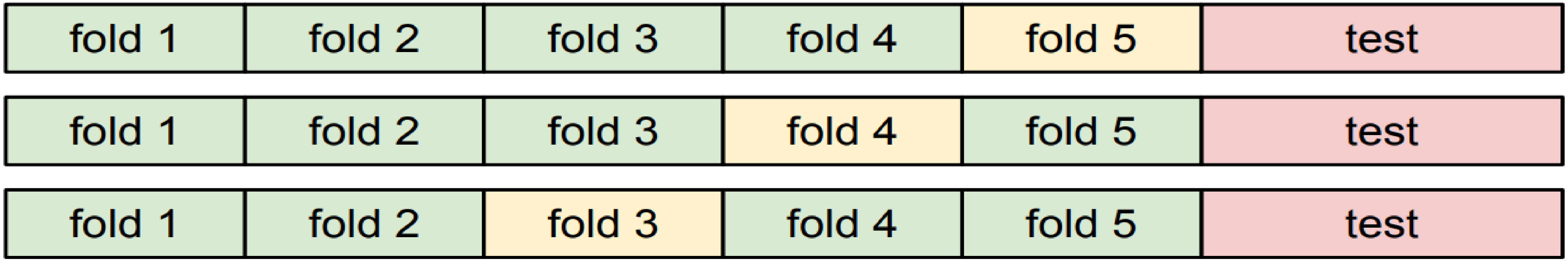
In General People Do: Train/Test

- Split data into train and test,
- Choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

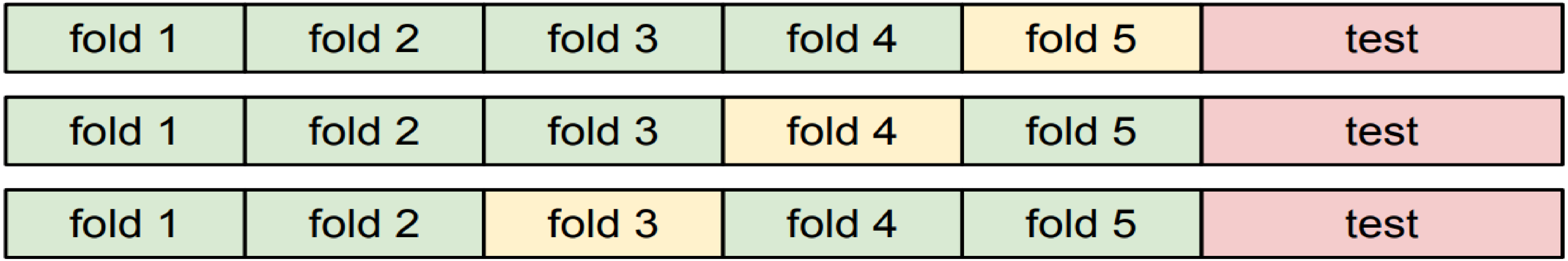
K-Fold Validation

- Split data into folds,
- Try each fold as validation and average the results



K-Fold Validation

- Split data into folds,
- Try each fold as validation and average the results



Useful for small datasets, but not used too frequently in deep learning

Better Approach: Train/Val/Test sets

- Split data into **train**, **val**, and **test**;
- Choose hyperparameters on val and evaluate on test

Better Approach: Train/Val/Test sets

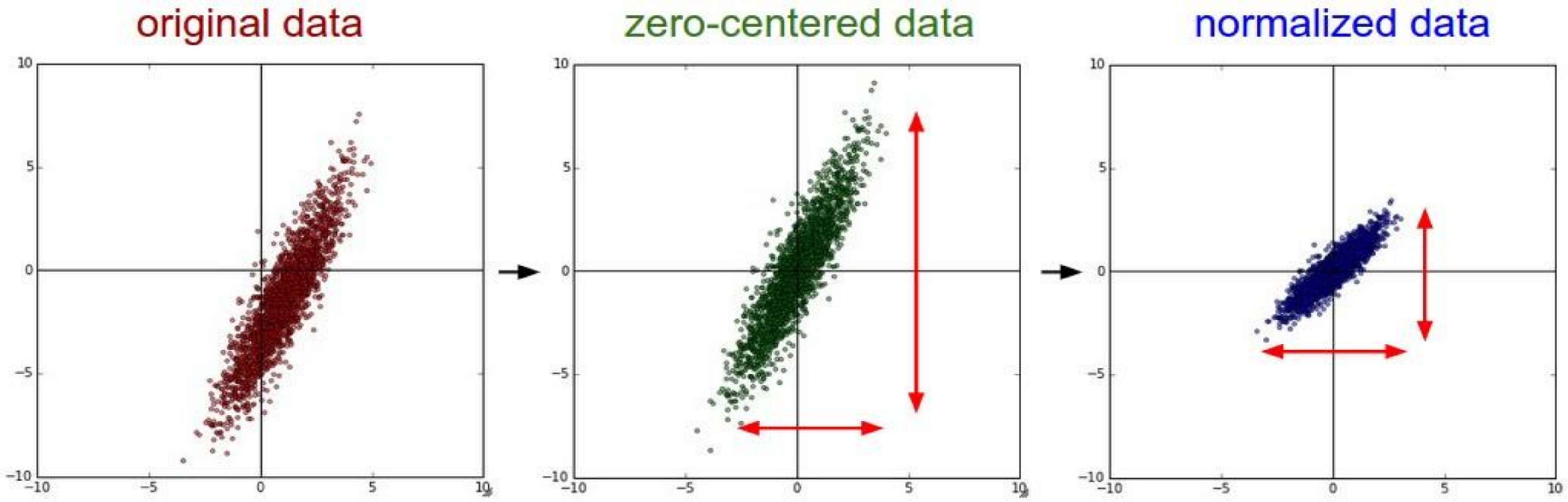
- Split data into **train**, **val**, and **test**;
- Choose hyperparameters on val and evaluate on test

Division can be done based on the size of dataset:

- Roughly 10k or 10% whichever is less for val and test sets.
- Rest in train set.

Data Preprocessing

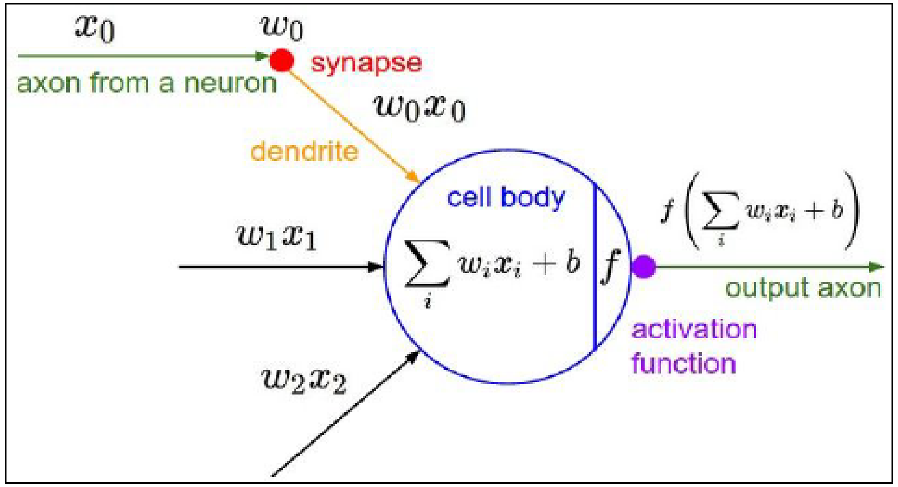
Data Preprocessing



Source: cs231n

Data Preprocessing

Consider what happens when the input to a neuron (x) is always positive:



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

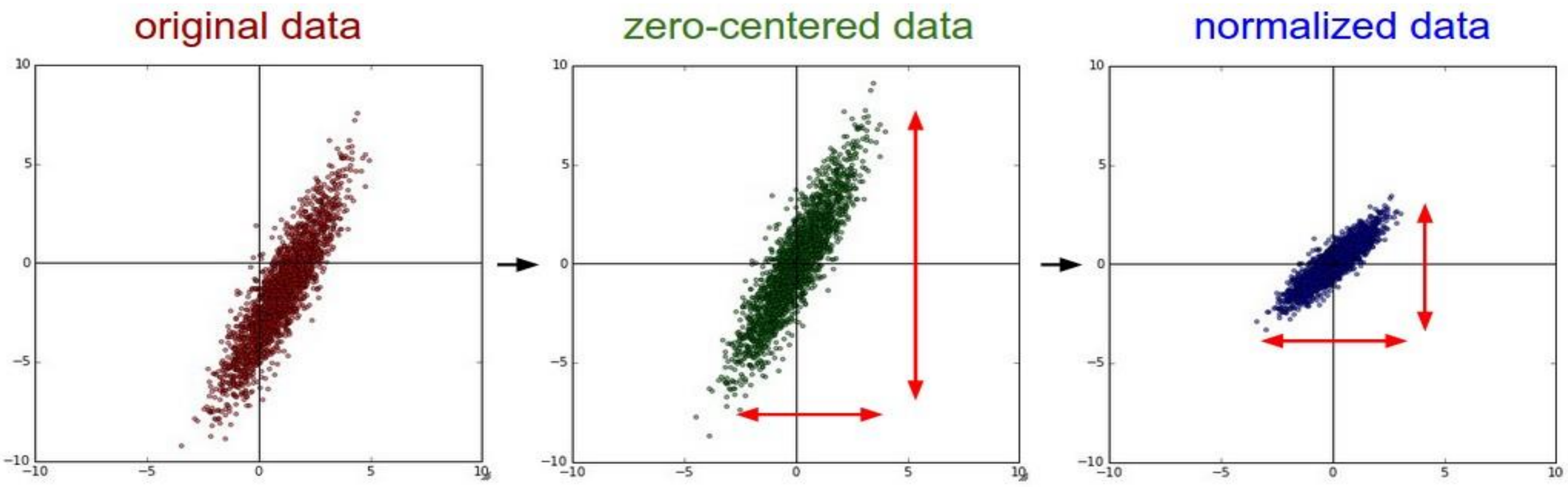
What can we say about the gradients on **w**?

Always all positive or all negative

(this is also why you want zero-mean data!)

Source: cs231n, Stanford University

Data Preprocessing



In practice for Images: only centering is preferred

e.g. consider CIFAR-10 example with [32,32,3] images

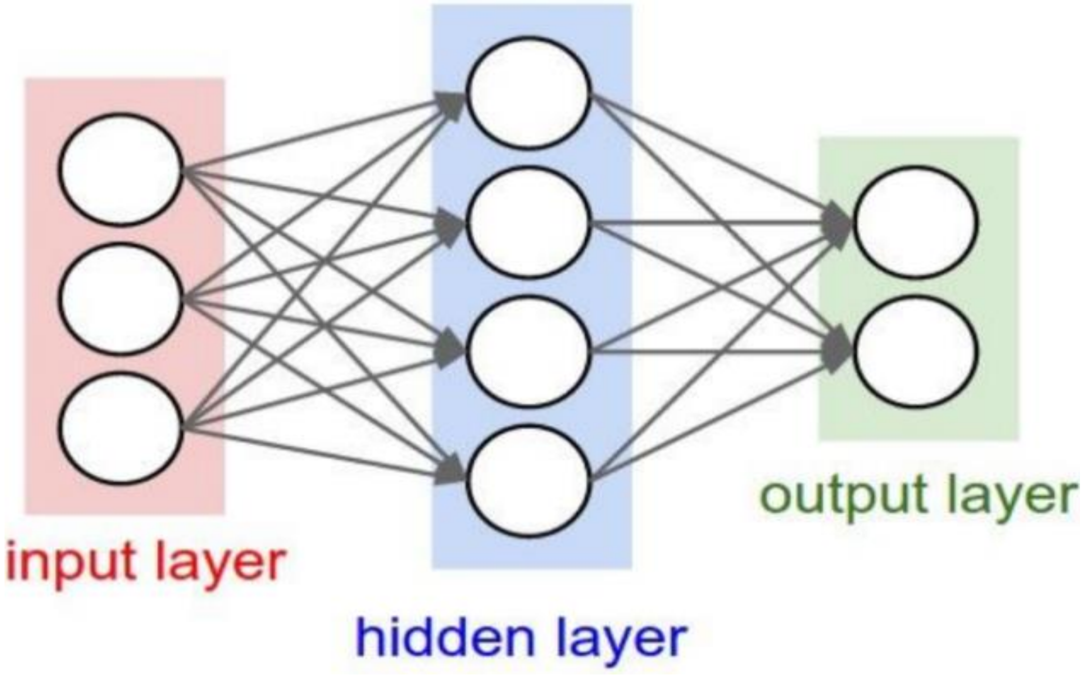
- Subtract the mean image (e.g. AlexNet)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet, ResNet, etc.)
  (mean along each channel = 3 numbers)

Source: cs231n

Weight Initialization

Weight Initialization: Constant

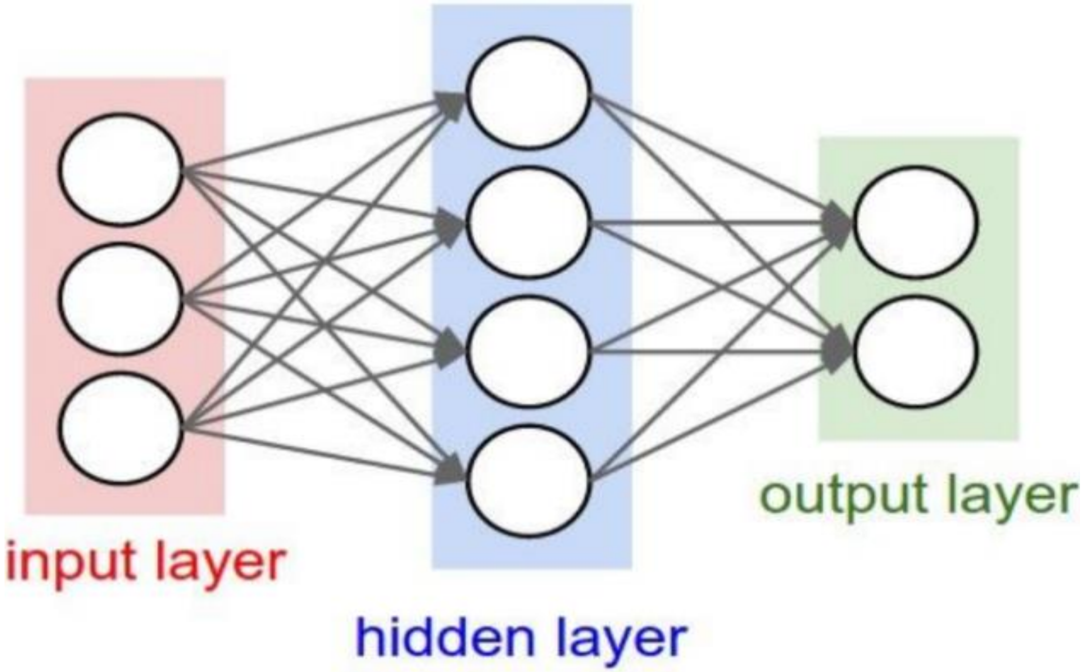
Q: what happens when W=Constant init is used?



Weight Initialization: Constant

Q: what happens when W=Constant init is used?

- Every neuron will compute the same output and undergo the exact same parameter updates.
- There is no source of asymmetry between neurons if their weights are initialized to be the same.



Source: cs231n

Weight Initialization: Gaussian

First idea: **Small random numbers**
(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for different neurons

Source: cs231n

Weight Initialization: Gaussian

First idea: **Small random numbers**
(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for different neurons

Works ~okay for small networks, but problems with deeper networks, i.e. Almost all neurons will become zero

-> gradient diminishing problem.

Source: cs231n

Weight Initialization: Gaussian

First idea: **Small random numbers**
(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for different neurons

Works ~okay for small networks, but problems with deeper networks, i.e. Almost all neurons will become zero

-> gradient diminishing problem.

Increase the standard deviation to 1

Source: cs231n

Weight Initialization: Gaussian

First idea: **Small random numbers**
(Gaussian with zero mean and 1e-2 standard deviation)

Symmetry breaking: Weights are different for different neurons

Works ~okay for small networks, but problems with deeper networks, i.e. Almost all neurons will become zero

-> gradient diminishing problem.

Increase the standard deviation to 1

Almost all neurons completely saturated, either -1 or 1. Gradients will be all zero.

-> gradient diminishing problem.

Source: cs231n

Weight Initialization: Gaussian

Lets look at some activation statistics

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and initializing as described in last slide.

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

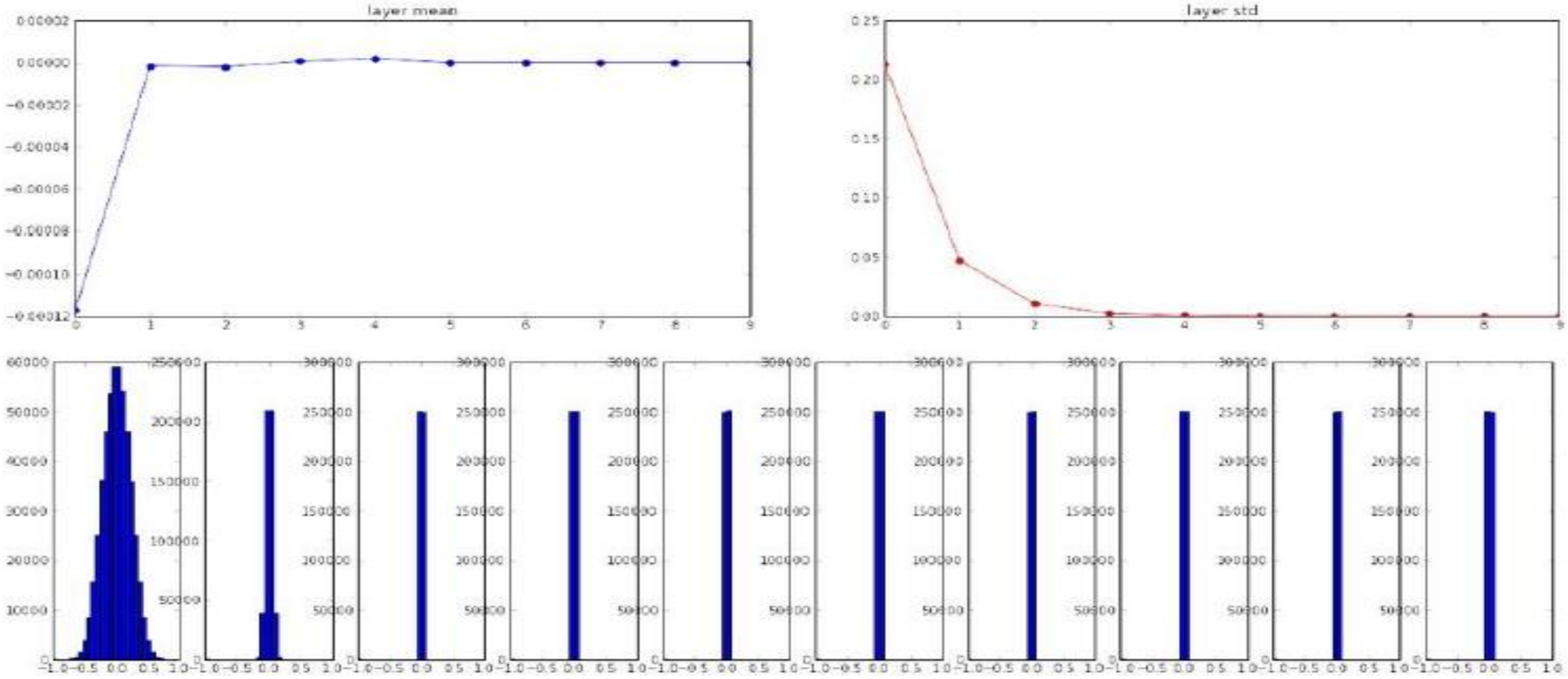
```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer
```

```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

Source: cs231n

Weight Initialization: Gaussian

```
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
```



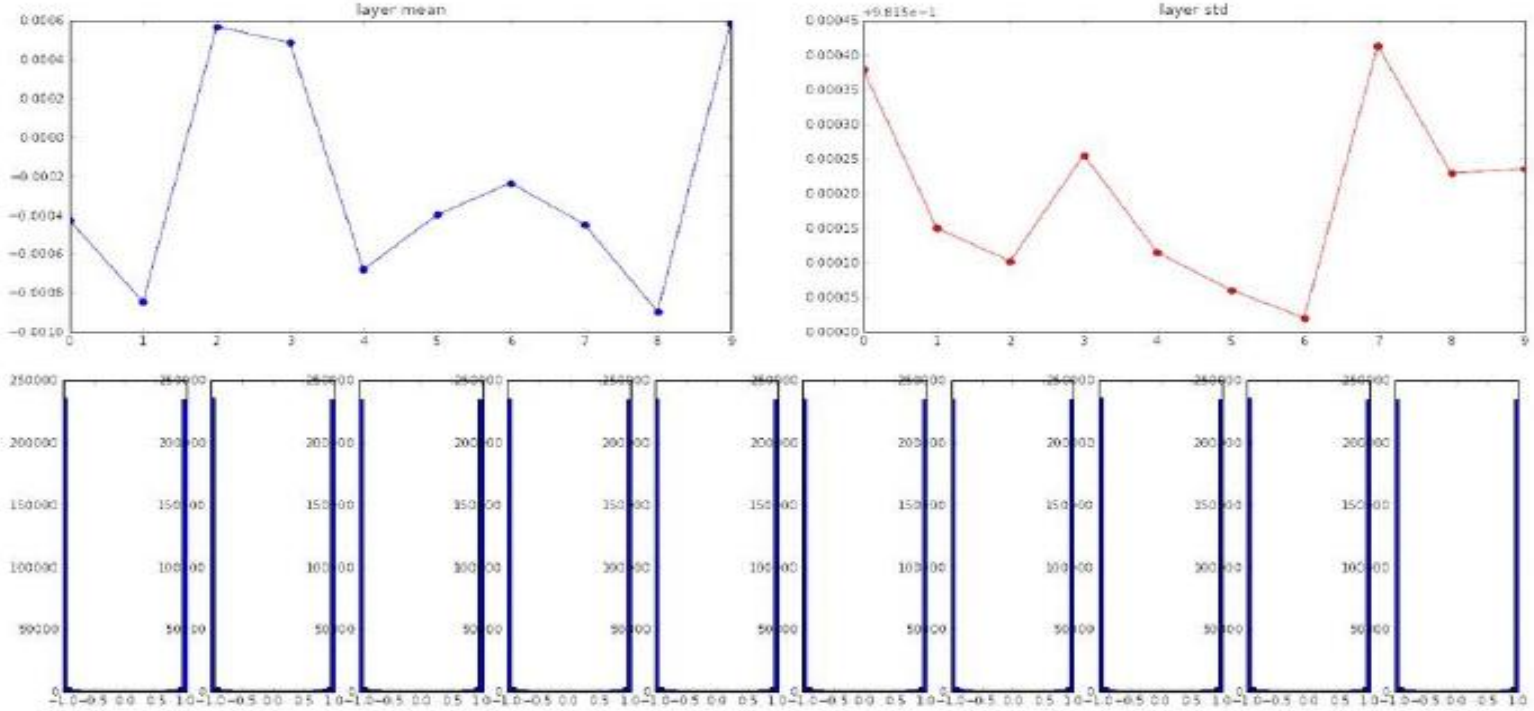
Source: cs231n

Weight Initialization: Gaussian

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

*1.0 instead of *0.01



Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

Source: cs231n

[Glorot et al., 2010]

Weight Initialization: Xavier

Calibrating the variances with 1/sqrt(fan_in)

$$W = np.random.randn(fan\_in, fan\_out)/np.sqrt(fan\_in)$$

Reasonable initialization.
(Mathematical derivation assumes linear activations)

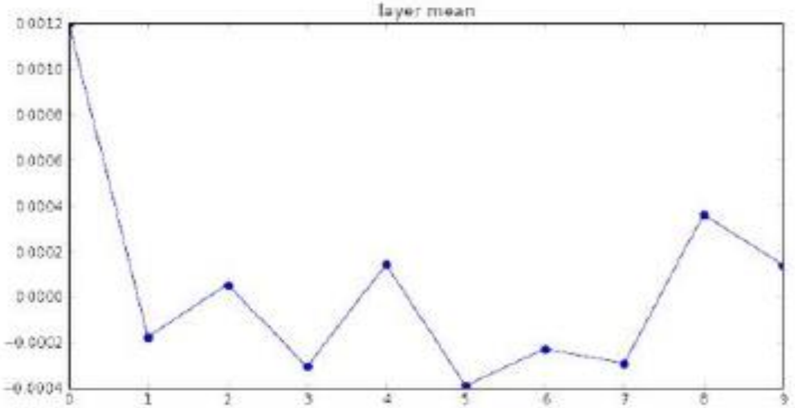
Source: cs231n

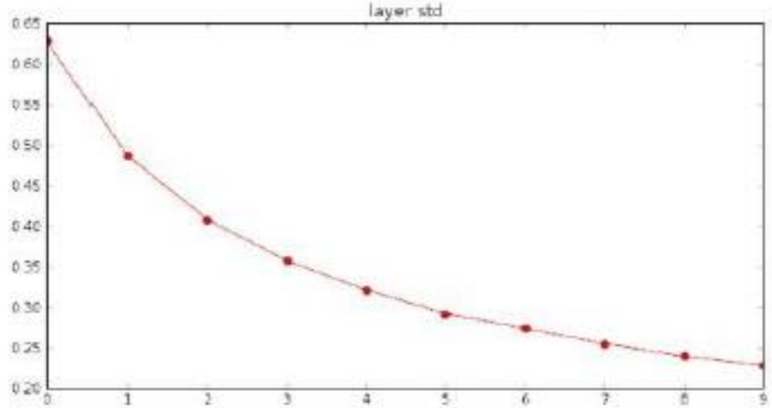
Weight Initialization: Xavier

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

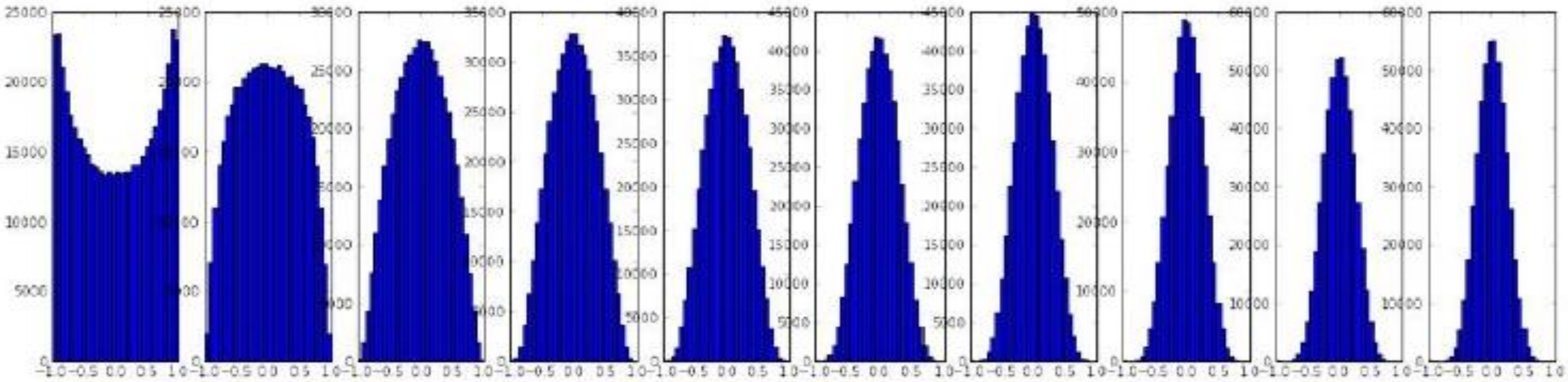
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

"Xavier initialization"
[Glorot et al., 2010]





Reasonable initialization.
(Mathematical derivation assumes linear activations)



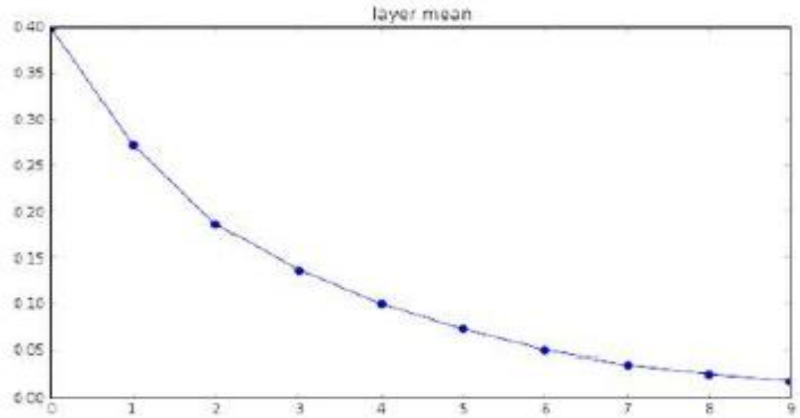
Source: cs231n

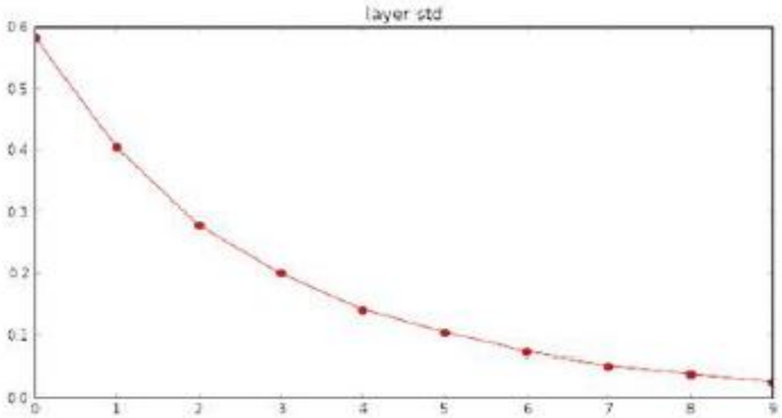
Weight Initialization: Xavier

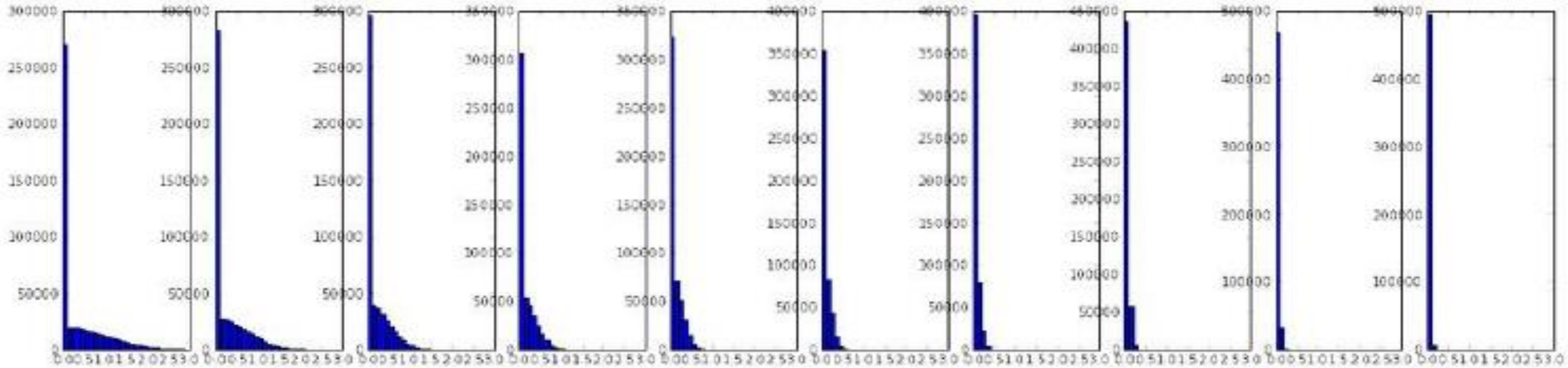
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

but when using the ReLU nonlinearity it breaks.







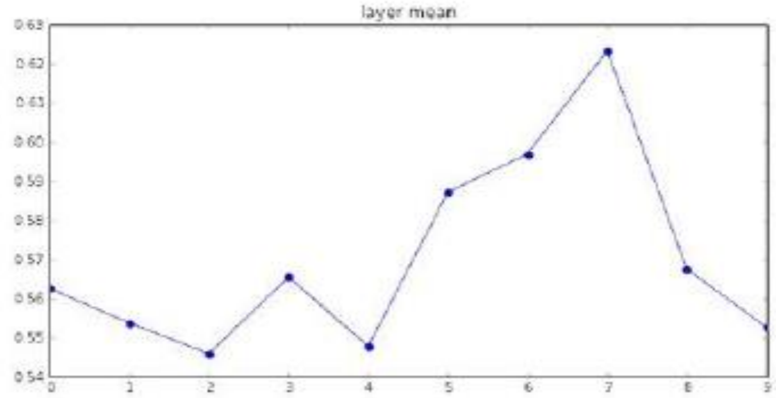
Source: cs231n

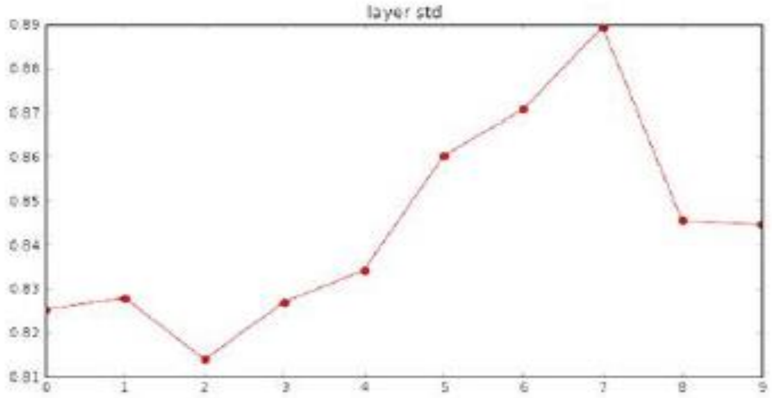
Weight Initialization: XavierImproved

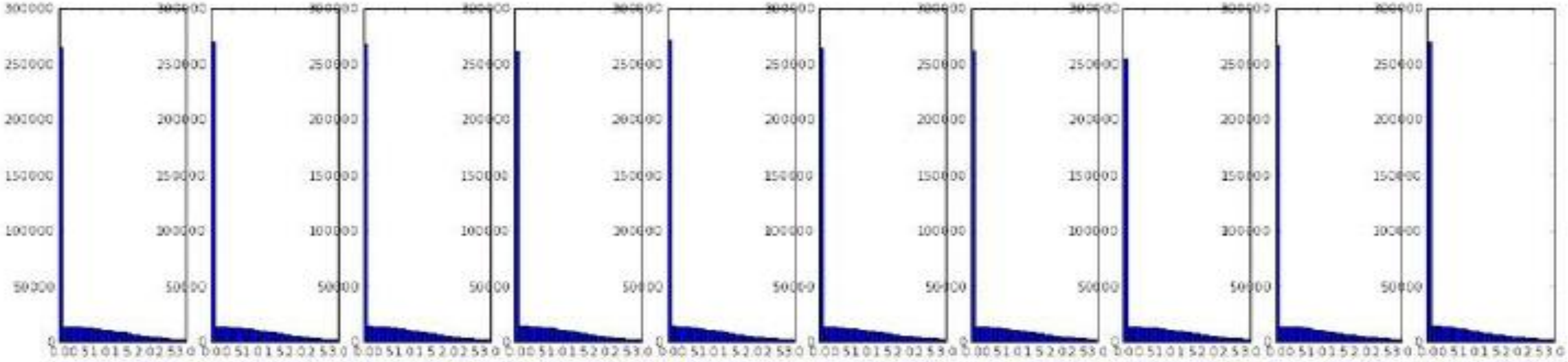
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```

```python
W = np.random.randn(fan_in, fan_out) / np.sqrt(2/fan_in) # layer initialization
```

He et al., 2015
(note additional 2/)







Source: cs231n

Proper initialization is an active area of research…

Understanding the difficulty of training deep feedforward neural networks by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init by Mishkin and Matas, 2015
…

Source: cs231n

Acknowledgement

Thanks to the following courses and corresponding researchers for making their teaching/research material online

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University
- And Many More ......