Xavier Initialization

HAI 소성민

2020.04.18

to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future.

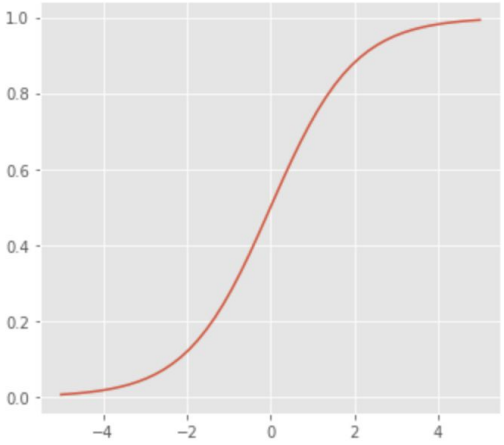
Experimental settings

- 1~5 hidden layer
- 1,000 hidden units per layer
- softmax logistic regression
- negative log-likelihood(-*log* P(y|x)) as cost function (x: input images, y: target class)

Activation Functions

- Sigmoid
- Tanh(hyperbolic tangent)
- Softsign

Sigmoid



$$\sigma(x) = \frac{1}{1 + exp(-x)}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

range : [0, 1]

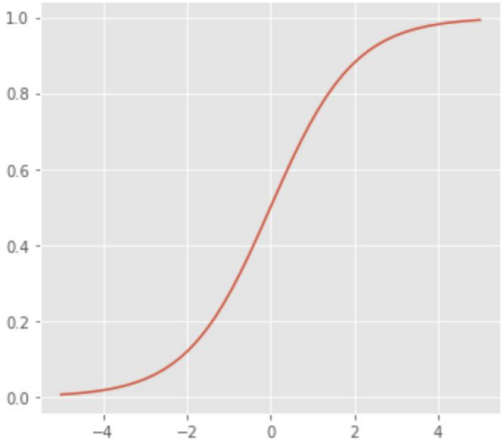
신경의 “활성화” 구조를 나타내기 때문에 유명하고 많이 사용된 함수

장점:

- 출력값의 범위가 0에서 1 사이로 제한되어, 정규화 관점에서 exploding gradient 문제 방지
- 미분 식의 단순한 형태

증명:
https://math.stackexchange.com/questions/78575/derivative-of-sigmoid-function-sigma-x-frac11e-x/1225116#1225116

Sigmoid

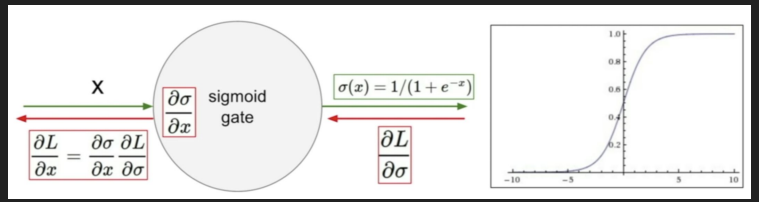


$$\sigma(x) = \frac{1}{1 + exp(-x)}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

단점:

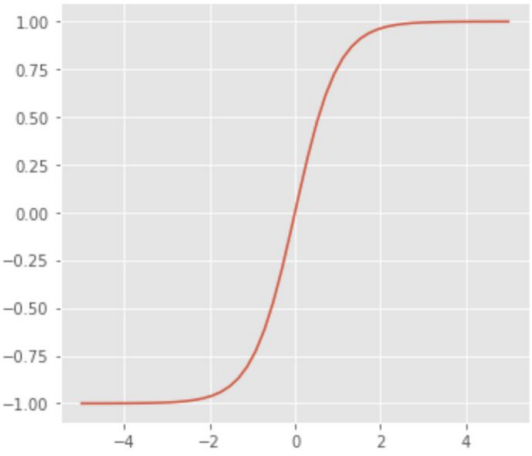
- 포화된(saturated) 노드들이 gradient를 죽임
  - ex) 입력값 x가 10 또는 -10일 경우



- 출력값의 중앙값이 0이 아님
  - 참고: https://datascience.stackexchange.com/questions/14349/difference-of-activation-functions-in-neural-networks-in-general

- exp()의 연산 비용이 큼

Tanh



$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$tanh'(x) = 1 - tanh^{2}(x)$$

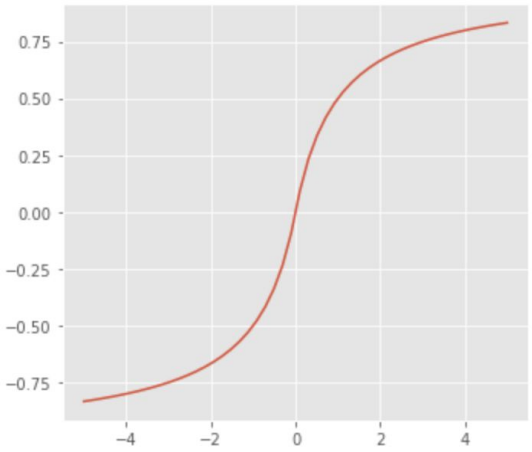
range: [-1, 1]

쌍곡선 함수(hyperbolic). Sigmoid 변형으로 이용가능

장점:
- 출력값의 중앙값이 0
- 이외 장점은 Sigmoid와 동일

단점:
- 포화된(saturated) 노드들이 gradient를 죽임

Softsign



$$f(x) = \frac{x}{1 + |x|}$$

$$f'(x) = \frac{1}{(1 + |x|)^2}$$

range: [-1, 1]

tanh와 비슷한 함수

tanh와는 다르게 함수의 끝이(극한부분이) 지수 꼴이 아닌 이차 다항식 꼴(quadratic polynomial)

참고:
https://www.quora.com/Why-is-the-softsign-activation-function-rarely-used-or-mentioned-in-Deep-Learning

Tanh vs Softsign

Red Line → Soft Sign Activation Function

Blue Line → Tanh Activation Function

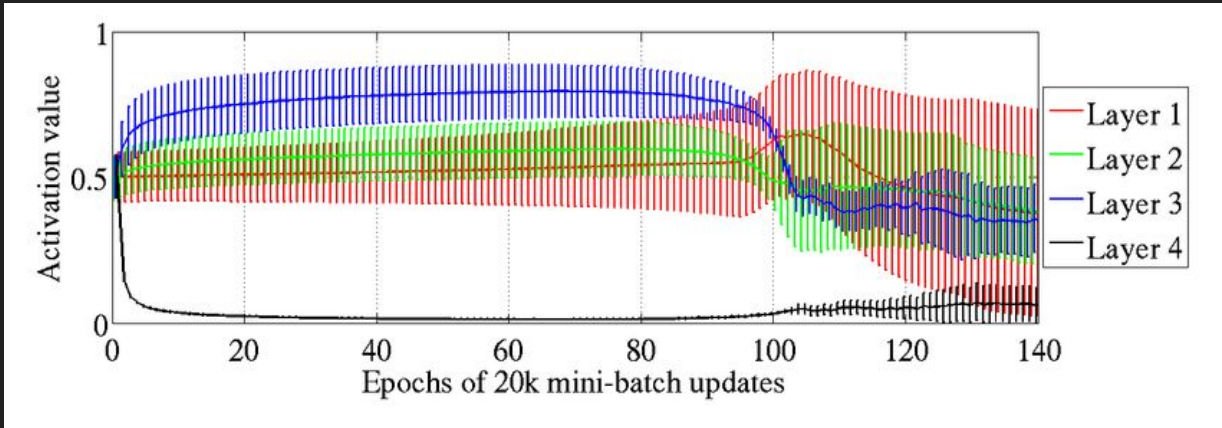
Green Line → Derivative for Soft Sign Function

Orange Line → Derivative for Tanh Activation Function

Experiment with Sigmoid

bias = 0, initialization = standard

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right],$$



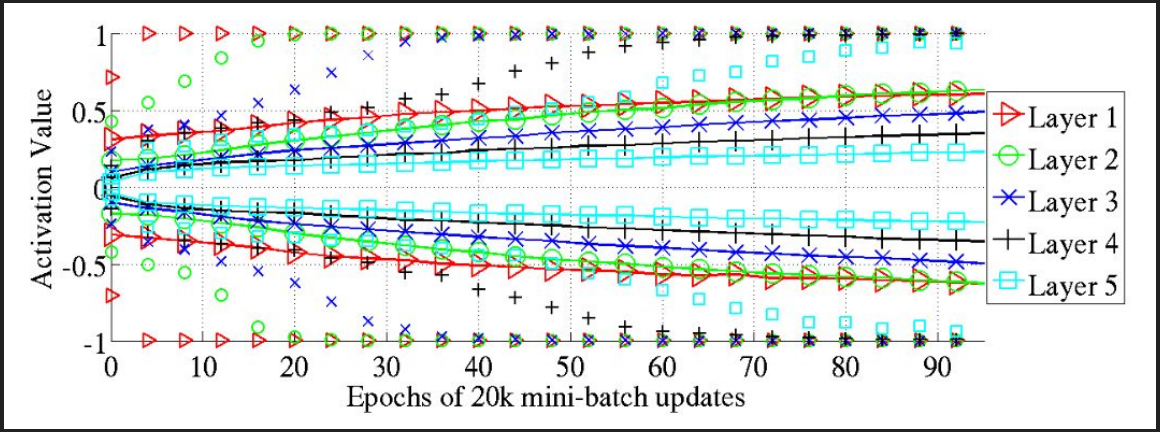
- 가장 마지막 hidden layer에서 급격히 0으로 포화되는 현상 발견
- epoch 100 이후로는 천천히 포화가 해결되는 현상 발견

=> 5 이상의 layer network 모델에서는 해당 현상 극복 불가능으로 결론

Experiment with Tanh

bias = 0, initialization = standard

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right],$$

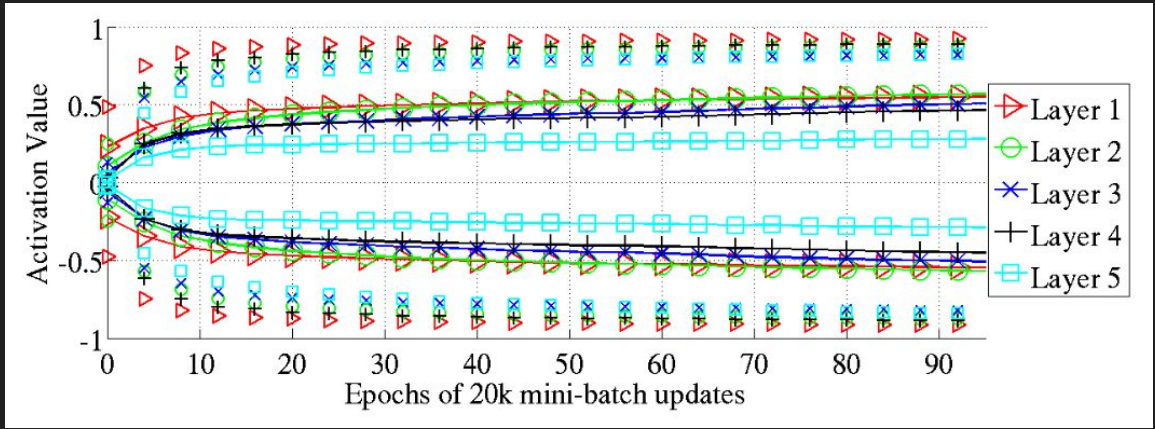


- Sigmoid의 경우처럼 가장 마지막 hidden layer에서 급격히 0으로 포화되는 현상은 없음
- 첫번째 layer부터 점점 포화현상이 발생해서 학습을 시킬때마다 전체적으로 포화되는 현상 발견
- > 이유를 찾지 못해 TODO로 남김

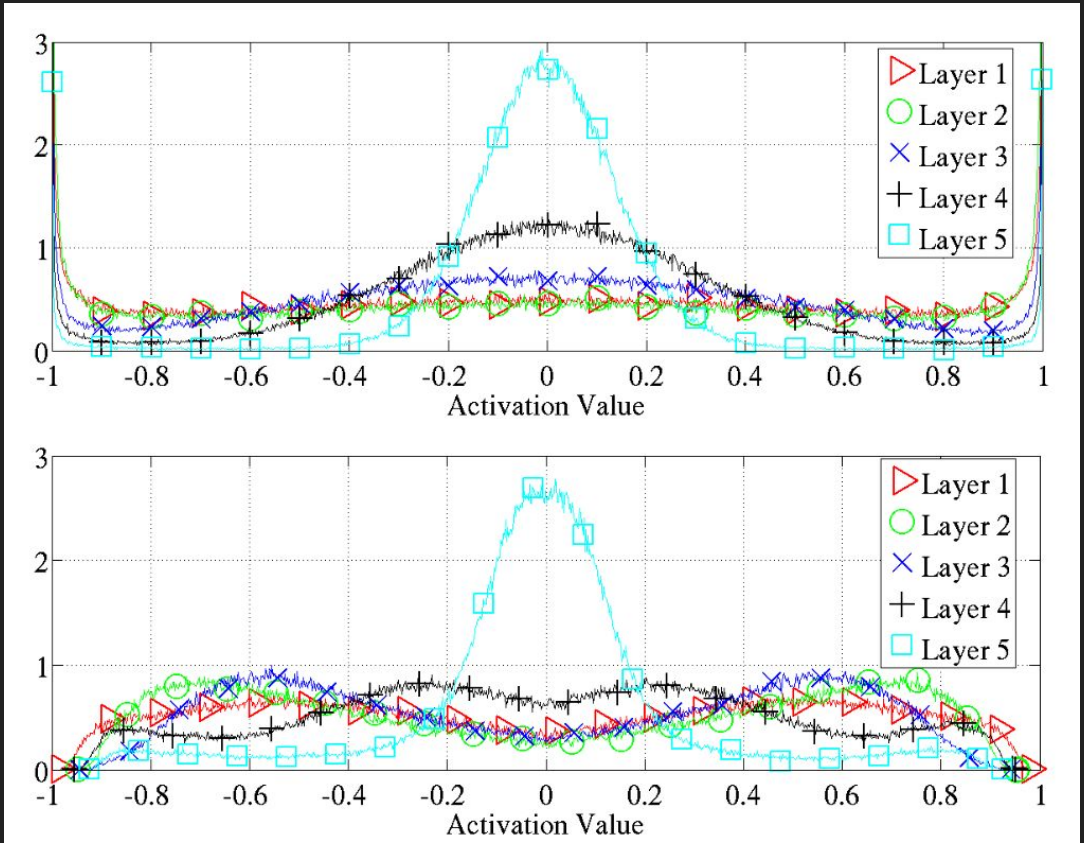
Experiment with Softsign

bias = 0, initialization = standard

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right],$$



- tanh의 경우와 비슷한 결과를 보임
- tanh와는 달리 포화현상을 보이지는 않았는데 그 이유는 tanh보다 부드러운 이차 다항식이기 때문


Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Activation Value
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Activation Value

효율적으로 초기화된 weights?

- *keep the same variance of the weights gradient across layers, during training*

- 즉, 모델의 네트워크 층이 깊어지더라도 gradient를 유실하지 않고 일정한 분산 분포를 가지도록 하는 것

weights를 0으로 초기화

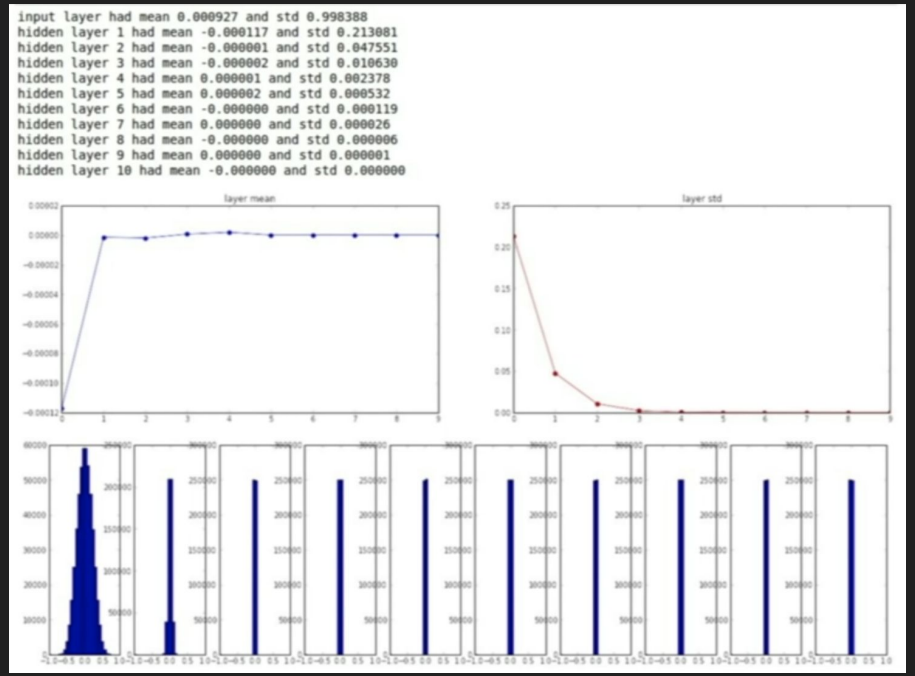
모든 노드가 비슷한 값으로 업데이트

- weights가 0이면, 입력값이 들어올 때 기본적으로 같은 계산을 하기 때문에 비슷하거나 같은 결과값을 가짐
- 또 비슷하거나 같은 gradient를 가짐
- 결과적으로 전체적인 값들이 거의 동일하게 됨

weights를 small random number로 초기화

```
D = np.random.randn[1000, 500]
hidden_layer_sizes = [500]*10
for i in range(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
```

- 평균값은 거의 0
- 표준편차값은 급격히 0으로 떨어짐
- activation function 출력값도 거의 0

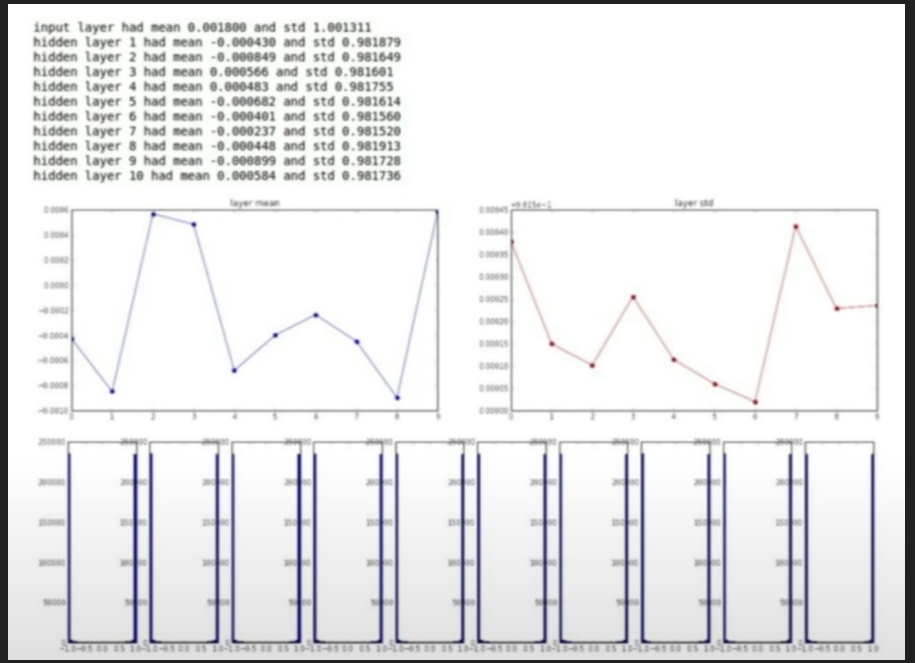


출처: https://www.youtube.com/watch?v=wEoyxE0GP2M&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=7&t=2935s

weights를 big random number로 초기화

```
D = np.random.randn[1000, 500]
hidden_layer_sizes = [500]*10
for i in range(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) * 1.0 # layer initialization
```

- weight값이 크기 때문에 항상 saturated
- gradient 값이 항상 0이기 때문에 weight값의 업데이트가 없음



출처: https://www.youtube.com/watch?v=wEoyxE0GP2M&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=7&t=2935s

$$\mathbf{s}^i = \mathbf{z}^i W^i + \mathbf{b}^i$$

$$\mathbf{z}^{i+1} = f(\mathbf{s}^i)$$

$$\frac{\partial Cost}{\partial s_k^i} = f'(s_k^i) W_{k,\bullet}^{i+1} \frac{\partial Cost}{\partial \mathbf{s}^{i+1}} \tag{2}$$

$$\frac{\partial Cost}{\partial w_{l,k}^i} = z_l^i \frac{\partial Cost}{\partial s_k^i} \tag{3}$$

$$f'(s_k^i) \approx 1, \tag{4}$$

$$Var[z^i] = Var[x] \prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}], \tag{5}$$

$$Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^d}\right] \prod_{i'=i}^{d} n_{i'+1} Var[W^{i'}], \tag{6}$$

$$Var\left[\frac{\partial Cost}{\partial w^i}\right] = \prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}] \prod_{i'=i}^{d-1} n_{i'+1} Var[W^{i'}] \times Var[x] Var\left[\frac{\partial Cost}{\partial s^d}\right]. \tag{7}$$

$$\forall (i, i'), Var[z^i] = Var[z^{i'}]. \tag{8}$$

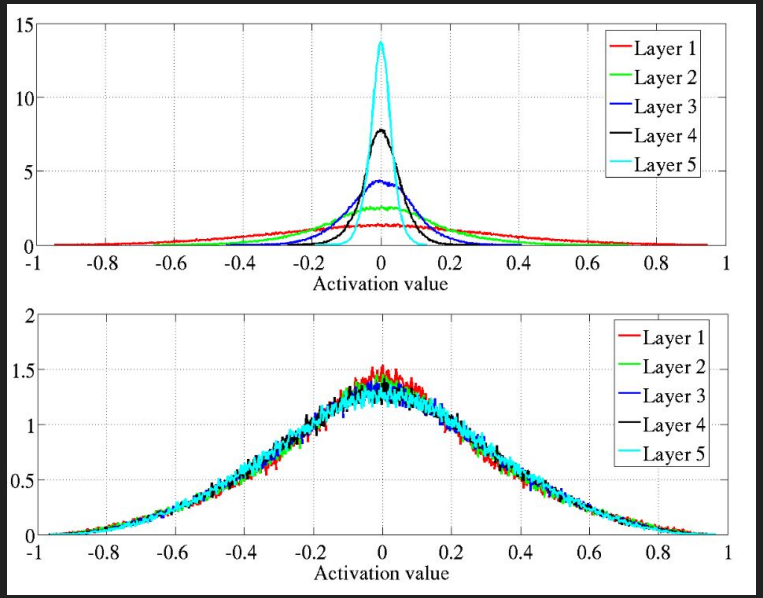
$$\forall (i, i'), Var\left[\frac{\partial Cost}{\partial s^i}\right] = Var\left[\frac{\partial Cost}{\partial s^{i'}}\right]. \tag{9}$$

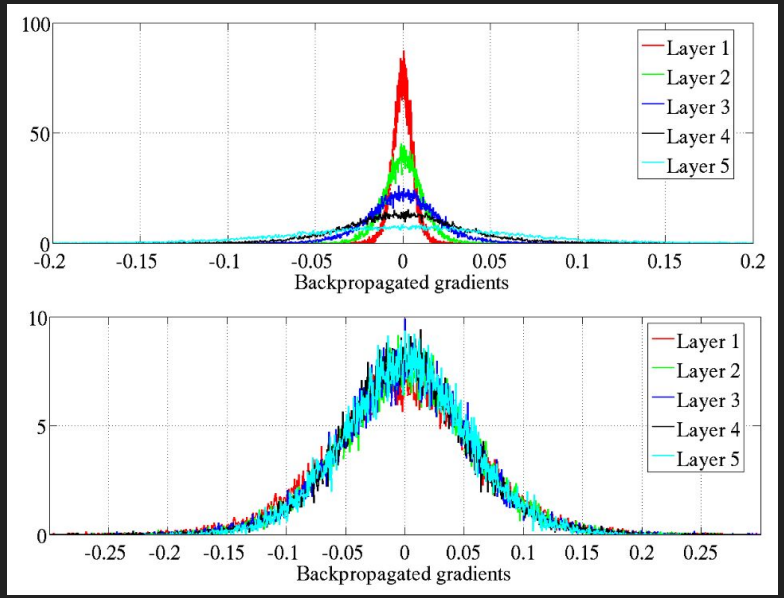
$$\forall i, \quad n_i Var[W^i] = 1 \tag{10}$$

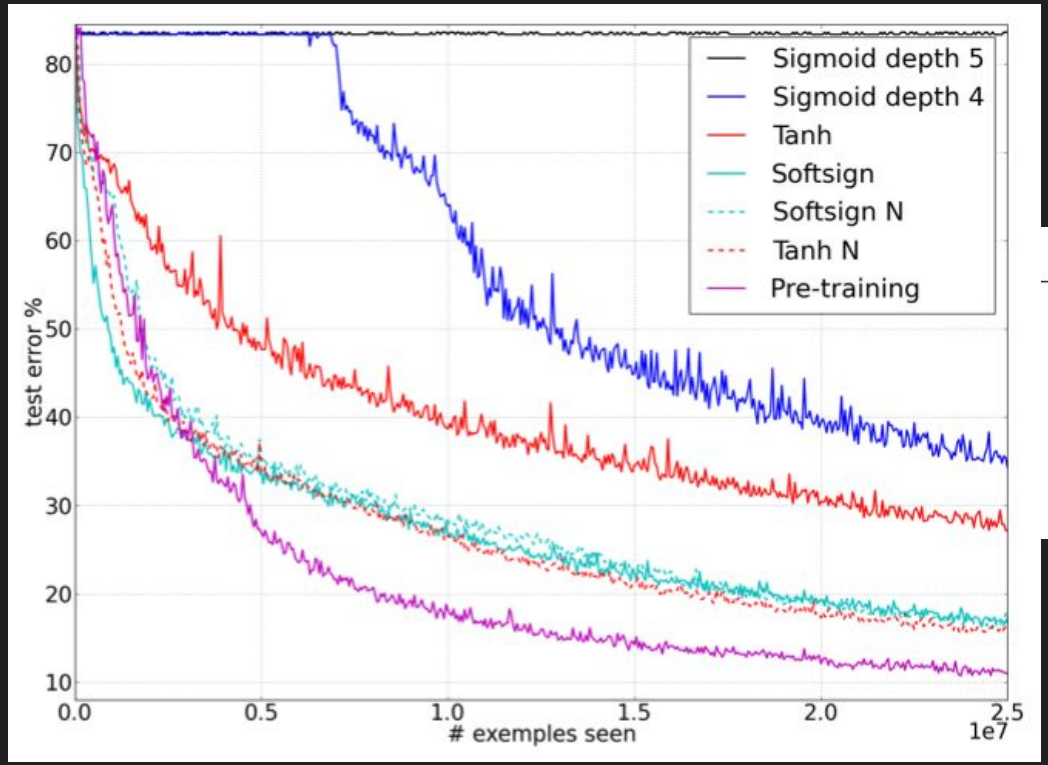
$$\forall i, \quad n_{i+1} Var[W^i] = 1 \tag{11}$$

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}} \tag{12}$$

$$W \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \Big] \tag{16}$$


Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Activation value
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Activation value


Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Backpropagated gradients
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
Backpropagated gradients



TYPE	Shapeset	MNIST	CIFAR-10	ImageNet
Softsign	**16.27**	**1.64**	55.78	**69.14**
Softsign N	**16.06**	**1.72**	**53.8**	**68.13**
Tanh	27.15	**1.76**	55.9	70.58
Tanh N	**15.60**	**1.64**	**52.92**	**68.57**
Sigmoid	82.61	2.21	57.28	70.66

Results in bold are statistically different from non-bold ones under the null hypothesis test with p=0.005

```
# 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 range(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) / np.sqrt(fan_in) # layer initialization

  H = np.dot(X, W) #matrix multiply
  H = act[nonlinearities[i]](H)
  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.means(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_std[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))
```

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, Mishkin and Matas, 2015

…