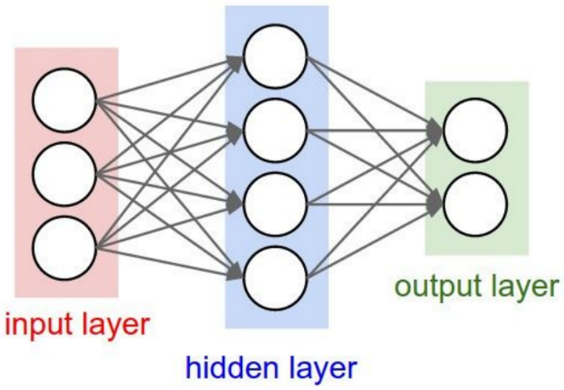
Lecture 6-3

Title	Training Neural Networks I
slide	http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

Weight Initialization



“Two Layer Neural Network" 예시를 한번 보자. 맨 처음에 어떤 초기 가중치들이 있고, 우리는gradient를 계산하여 가중치를 업데이트할 것이다. 만약 "모든 가중치 = 0" 이면, 즉 모든 파라미터를 0으로 세팅하면 어떻게 될까? 가중치가 0이므로 모든 뉴런은 같은 연산을 수행하게 된다. 따라서 출력도 모두 같게 되고, gradient도 서로 같게 되어, 모든 가중치가 똑같은 값으로 업데이트 된다. 이러면 모든 뉴런이 똑같이 생기게 되고, 이것은 우리가 원하는 것이 아니다. 이렇듯 모든 가중치를 동일한 값으로 초기화시키면 "Symmetry breaking"이 일어날 수 없다.

1. small random numbers

초기화 문제를 해결하는 첫번째 방법은 임의의 작은 값으로 초기화하는 것이다. 초기 W를 표준정규분포(standard gaussian)에서 샘플링하고, 좀 더 작은 값을 위해 스케일링을 해준다. 가령 0.01을 나눠 표준편차를 1e-2 즉 0.01로 만들어 둔다. 이런 식으로 모든 가중치를 임의의 값으로 초기화하면 작은 네트워크의 경우에는 충분히 Symmetry breaking이 해결된다.

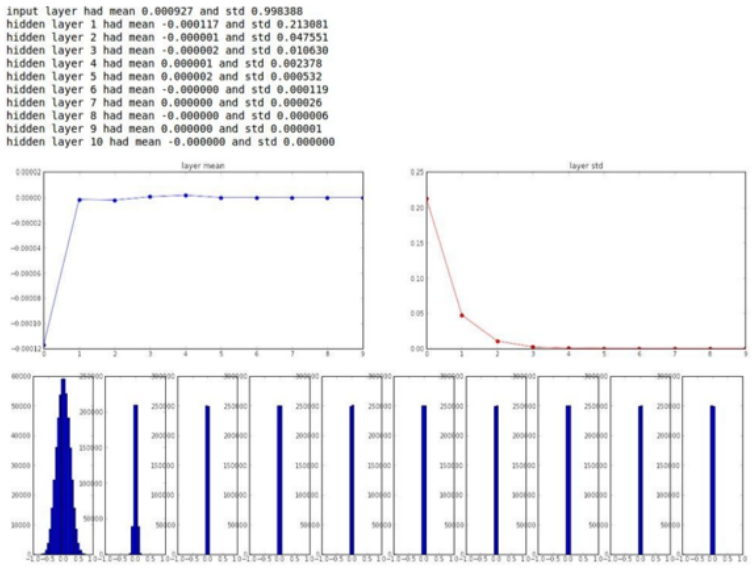
Lets look at some activation statistics

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))
```

그러나 더 깊은 네트워크에서는 문제가 생길 수 있다. 위와 같이 10개의 레이어로 이루어진 네트워크가 있고, 레이어당 500개의 뉴런이 있고, nonlinearities로는 tanh를 사용한다고 할 때, 가중치를 "임의의 작은 값" 으로 초기화시킨다고 하자. 그러면 X와 W를 내적한 값에 tanh를 취하고, 그 값을 저장한다.

```
w = np.random.randn(fan_in, fan_out) * 0.01
```

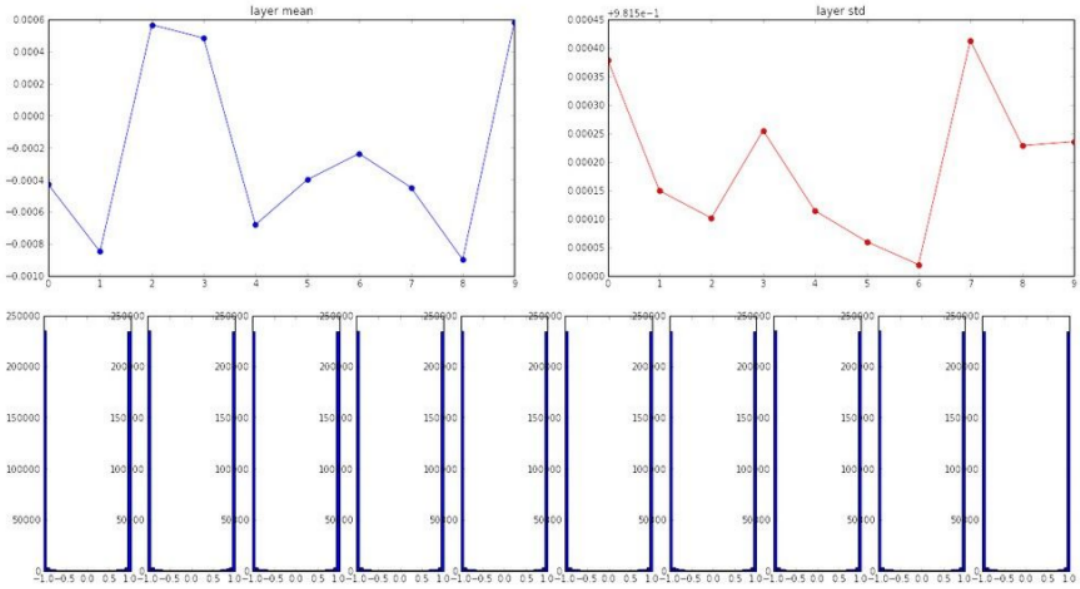


데이터를 랜덤으로 만들어주고 forward pass 시켜서 각 레이어별 activations 수치를 통계화 시켜서 평균과 표준편차를 계산하여 나타냈다. 첫번째 레이어를 보면, tanh가 zero-centered 이므로 평균은 항상 0 근처에 있다. 나머지 출력 분포들도 평균은 항상 0근처에 있다. 그러나 표준편차는 아주 가파르게 줄어들어서 0에 수렴하는 것을 볼 수 있다. 위의 그래

프를 보면, 첫 번째 레이어에서는 가우시안처럼 생긴 좋은 분포를 형성하고 있지만, W를 곱하면 곱할수록 W가 너무 작은 값이라서 출력 값이 급격히 줄어들어 모든 활성함수의 결과가 0이 되는 문제가 발생한다.

이번에는 backwards pass하여 gradient를 구하는 것을 생각해보자.Backporp에서는 "upstream gradient"가 전파되고, 현재 가중치를 업데이트하려면 "upstream gradient"에 local gradient를 곱해준다. 우리의 예시에서 우선 WX를 W에 대해 미분해보면, local gradient는 입력 값인 X가 된다. 이렇게 되면, .X가 엄청 작은 값이기 때문에 gradient도 작을 것이고 결국 업데이트가 잘 일어나지 않는다.위의 과정을 통해 다양한 입력 타입에 따라 weight와 gradient가 어떤 영향을 미치게 되는지, 그리고 gradients가 연결되면서 (chaining) 어떤 식으로 전파(flowing back)가 되는지 생각해 볼 수 있다. gradient를 backprop하는 과정은 그 반대의 과정이다. upstream gradient에 W의 gradient인 X를 곱하고 backward pass의 과정에서 upstream gradient를 구하는 것은 현재 upstream에 가중치를 곱하는 것이다. W를 계속해서 곱하기 때문에 Backwork pass에서도 Forward에서처럼 점점 gradient값이 작아지게 되고, 따라서 upstream gradients는 0으로 수렴하게 된다.

2. lager random numbers



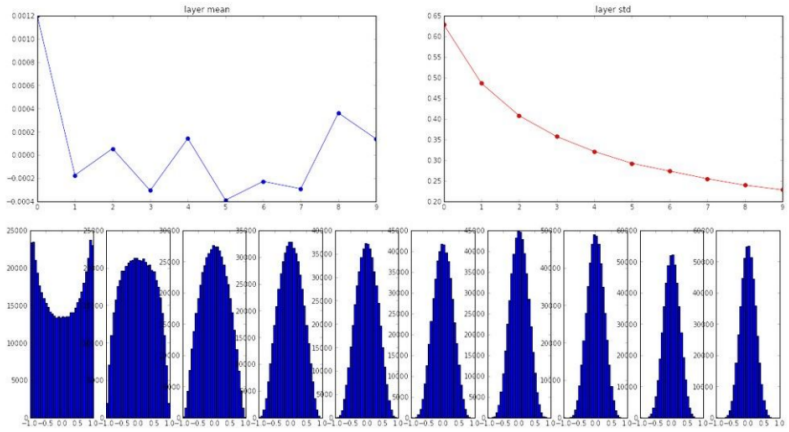
가중치를 좀 더 큰 값으로 초기화하면 어떻게 될까? 가중치의 편차를 0.01 이 아니라 1로 하여 생각해보자. 큰 가중치를 통과한 출력 WX를 구하고, 이를 tanh를 거치게 된다면 값들이 saturation 되게 된다. 그렇게 되면 위와 같이 출력이 항상 -1 이거나 +1이게 될 것이고, gradient는 0이 되어 가중치 업데이트가 일어나지 않는다. 가중치가 너무 작으면 사라져버리고, 너무 크면 saturation되어 버려서 적절한 가중치를 얻기는 너무 어렵다.

Xavier initialization

```
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]



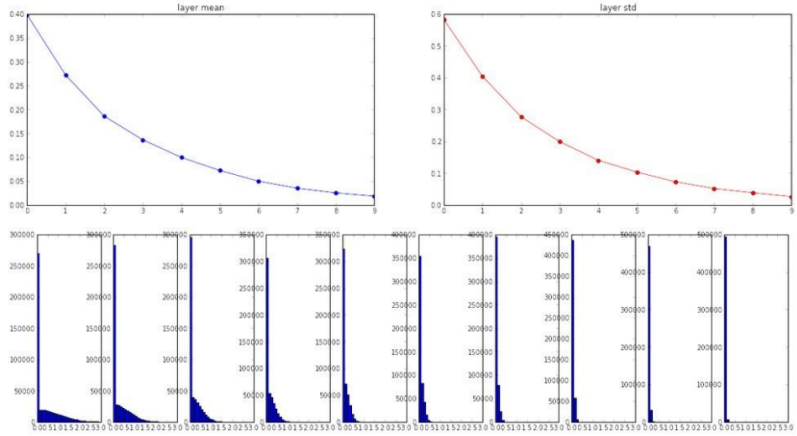
Reasonable initialization.
(Mathematical derivation
assumes linear activations)

가중치 초기화를 잘 할 수 있는 좋은 방법 중 하나는 Glorot가 2010년에 발표한 논문의 Xavier initialization이다. 기본적으로 Xavier initialization이 하는 일은 입/출력의 분산을 맞춰주는 것이다. 가장 위에 W의 공식을 보면, Standard gaussian으로 뽑은 값을 "입력의 수"로 스케일링해준다. 이 수식을 통해서 직관적으로 이해할 수 있는 것은 입력의 수가 작으면 더 작은 값으로 나누고 좀 더 큰 값을 얻게 된다는 것이다. 작은 입력의 수가 가중치와 곱해지기 때문에, 가중치가 커야 출력의 분산 만큼 큰 값을 얻을 수 있기 때문에 우리는 더 큰 가중치가 필요하다. 이와 반대로 입력의 수가 많은 경우에는 더 작은 가중치가 필요하다. 각 레이어의 입력이 Unit gaussian이길 원한다면 이런 류의 초기화 기법을 사용할 수 있다. 여기서 가정하는 것은 현재 Linear activation이 있다고 가정하는 것이다. 가령 tanh의 경우에 우리가 지금 tanh의 active region안에 있다고 가정하는 것이다.

```
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.



하지만 Xavier initialization에 문제가 하나 있는데, ReLU를 쓰면 잘 동작하지 않는다는 것이다. ReLU를 쓰면 출력의 절반을 죽여서 0이 되고, 결국 출력의 분산을 반토막 낸다. 그러므로 이전과 같은 가정을 해버리면 값이 너무 작아져서 ReLU에서는 잘 작동하지 않는다. 위에 보이는 것이 그런 현상인데 분포가 줄어드는 것을 확인할 수 있다. 점점 더 많은 값들이 0이 되고, 결국은 비활성(deactivated) 되어 버린다.