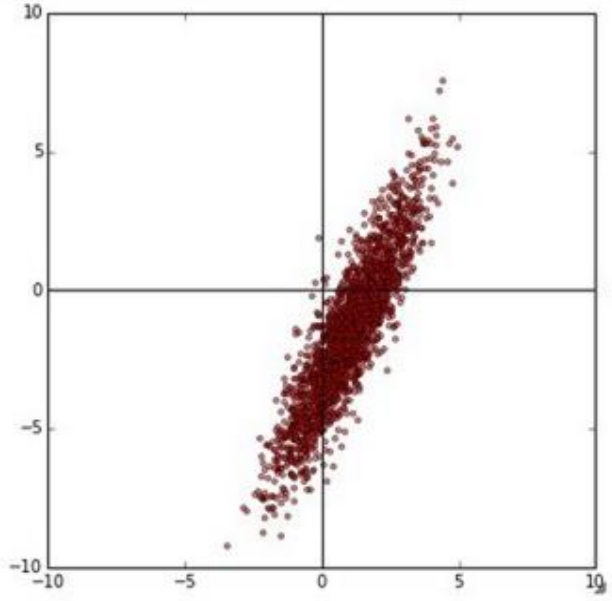
INICIALIZAÇÃO

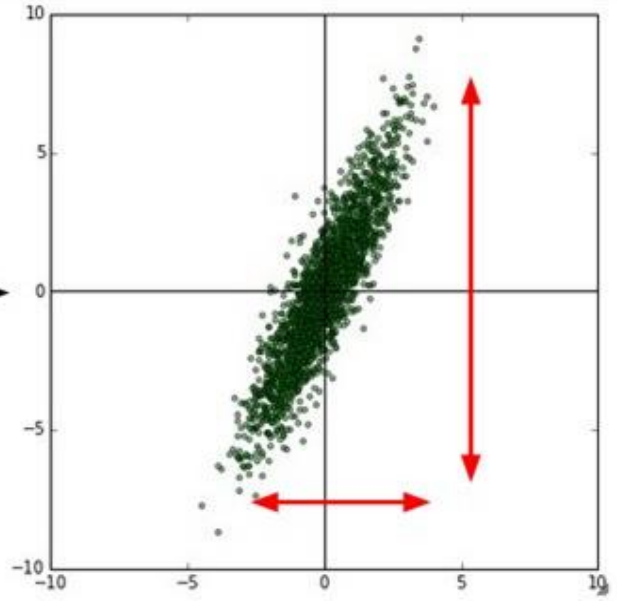
Zenilton K. G. Patrocínio Jr
zenilton@pucminas.br

Pré-Processamento de Dados – Básico

Dados originais

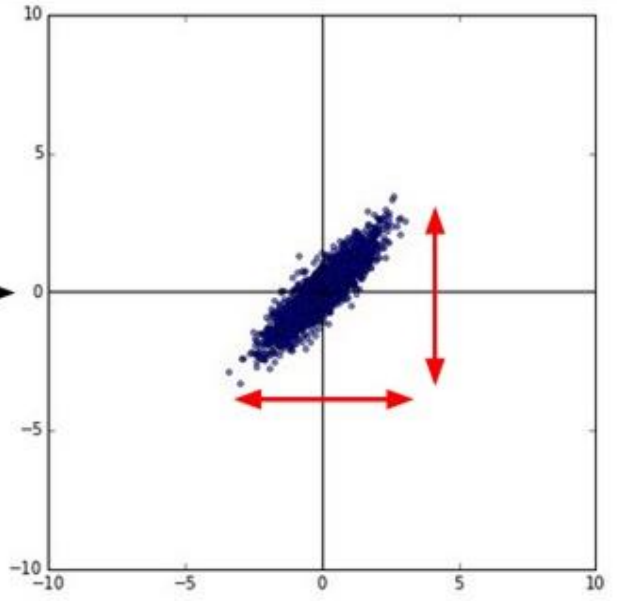


Dados centrados em zero




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

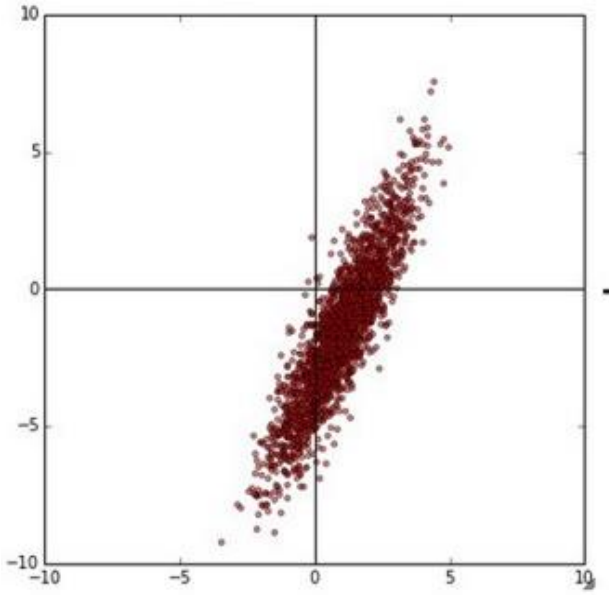
Dados normalizados



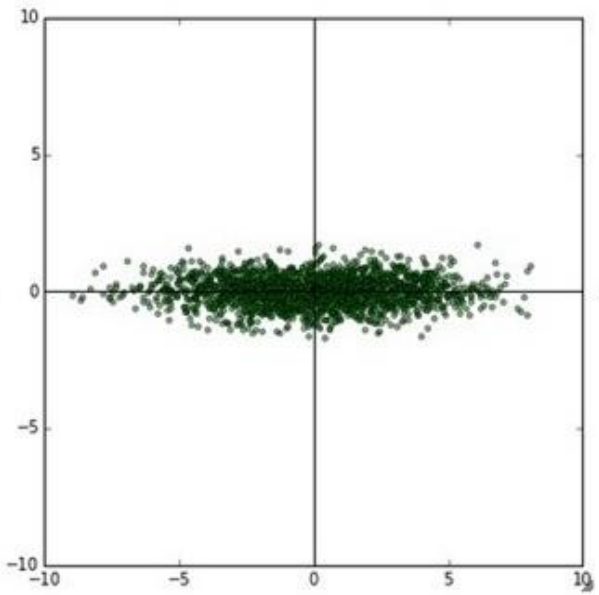

```
X /= np.std(X, axis = 0)
```

Pré-Processamento de Dados – Avançado

Dados originais

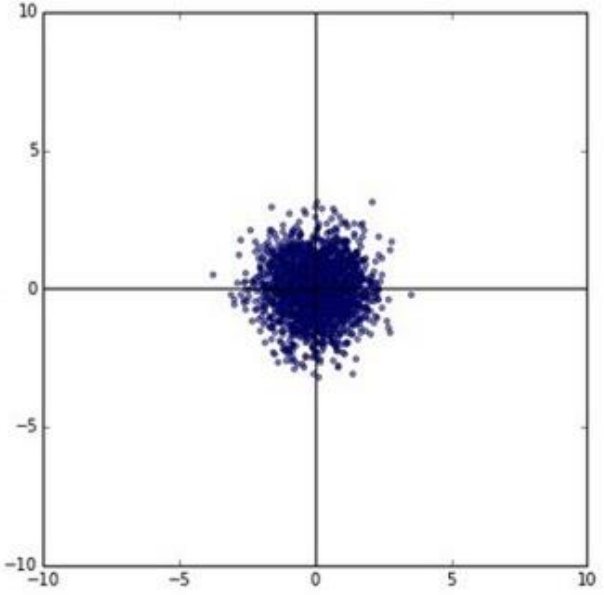


Dados sem correlação



Matriz de covariância diagonal

Dados sem correlação e variância 1



Matriz de covariância é a identidade

Pré-Processamento de Imagens

Apenas centrar os dados em zero

p.ex. considere o dataset CIFAR-10 com imagens [32,32,3]

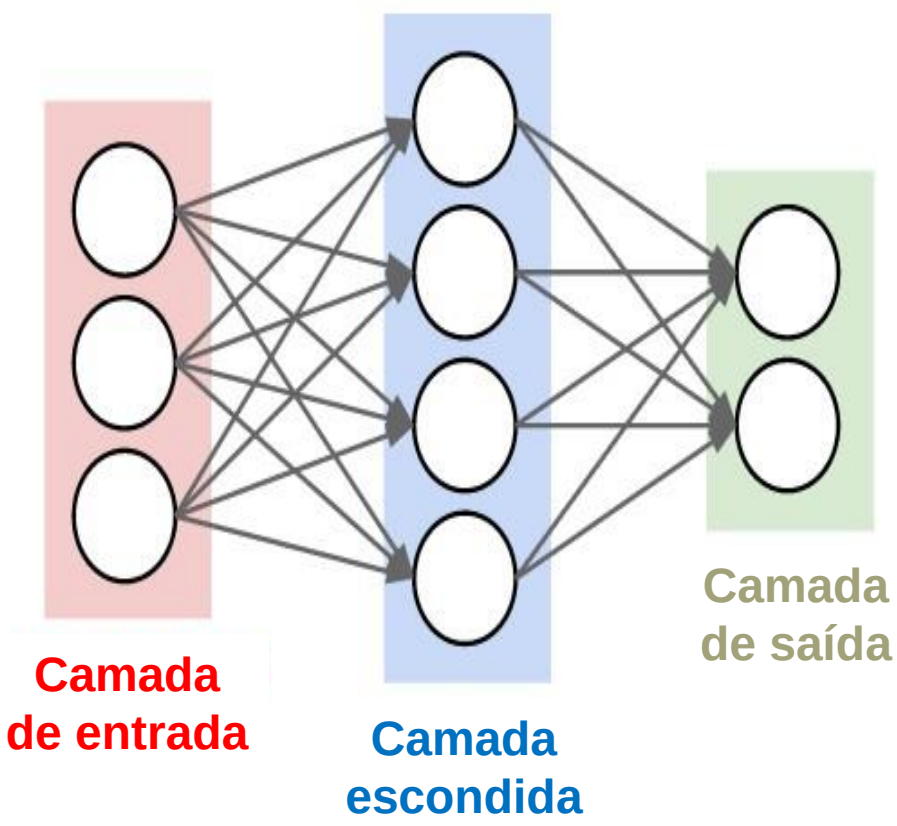
- Subtrair a imagem média (p.ex. AlexNet)
  imagem média = vetor de dimensões [32,32,3]

- Subtrair a média por canal (p.ex. VGGNet)
  média ao longo de cada canal = 3 números

Não é comum normaliza nem decorrelacionar os pixels de uma imagem

Inicialização de Pesos

- Pergunta: que ocorre quando os pesos são inicializados com zero, isto é, $\boldsymbol{W} = 0$?



Inicialização de Pesos

- Uma primeira idea: **Usar números randômicos pequenos**
  Usar uma distribuição normal com média zero e desvio padrão 10^{-2}

```
W = 0.01* np.random.randn(D,H)
```

Funciona bem para redes pequenas, mas pode levar a distribuições não homogêneas de ativações ao longo das camadas de uma rede

Inicialização de Pesos – Estatísticas de Ativação

Exemplo:

- Rede de 10 camadas
- 500 neurônios por camada
- Ativação Tanh
- Pesos inicializados com números randômicos pequenos

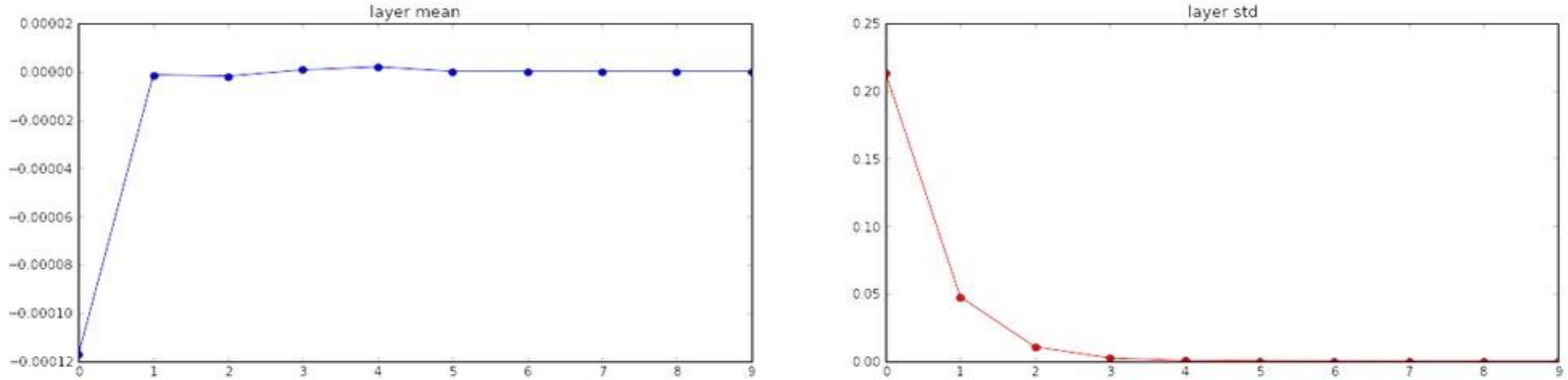
```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

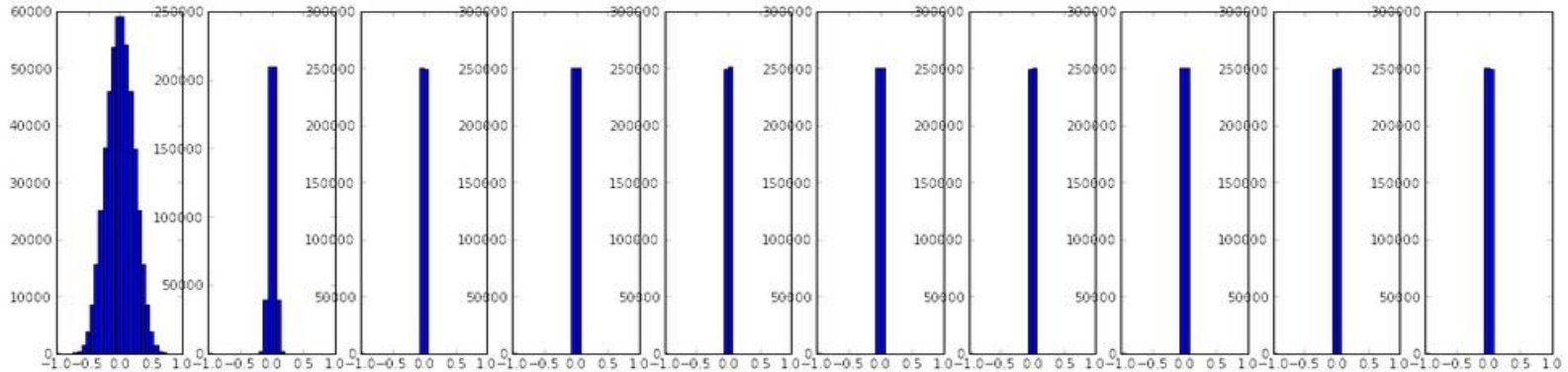
```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    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))
```

Inicialização de Pesos – Estatísticas de Ativação

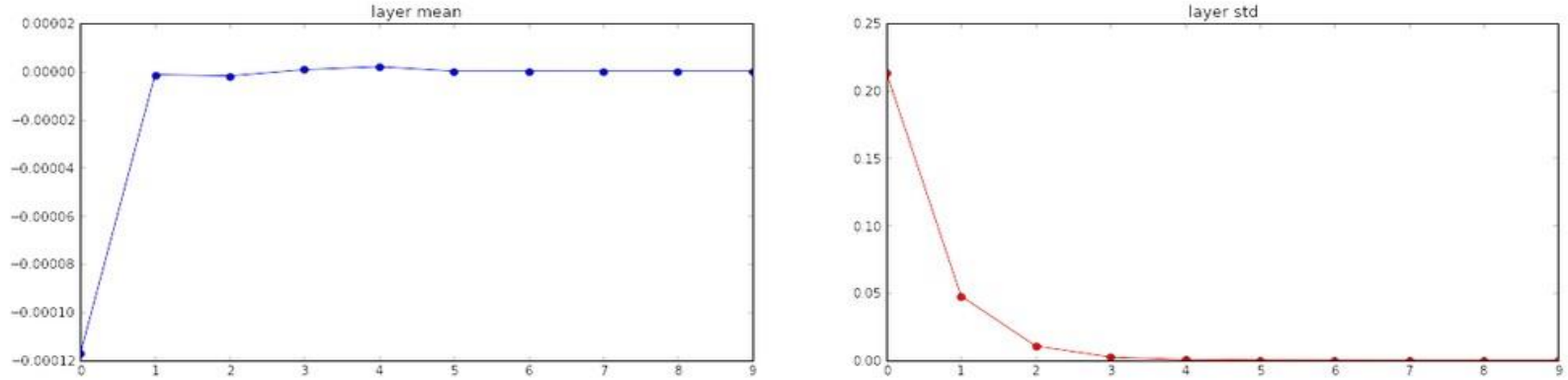
```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
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
```

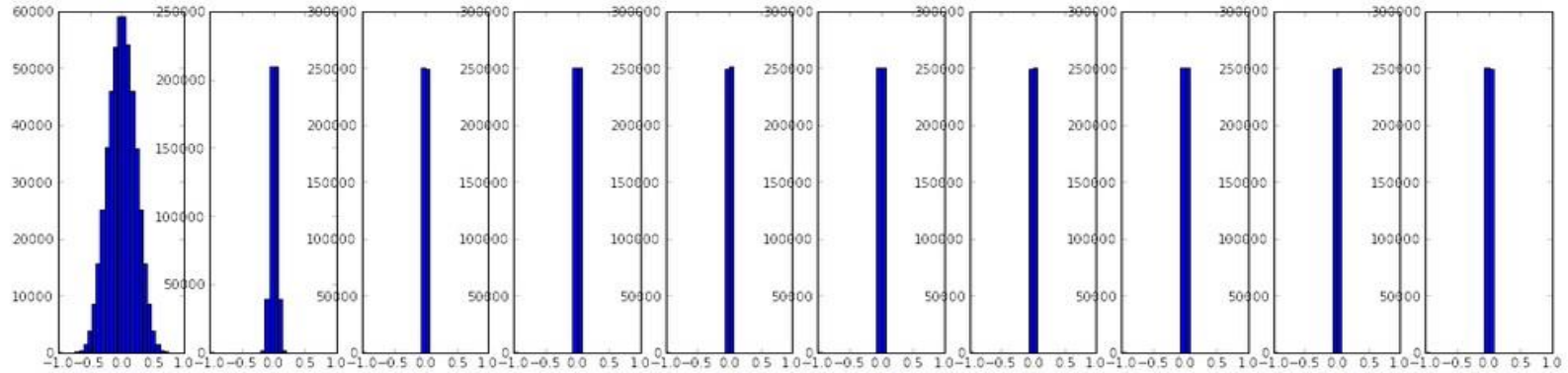




Inicialização de Pesos – Estatísticas de Ativação

```
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
```





Todas as ativações se tornam zero!

P: pense sobre o passo retrógrado. Como são os gradientes?

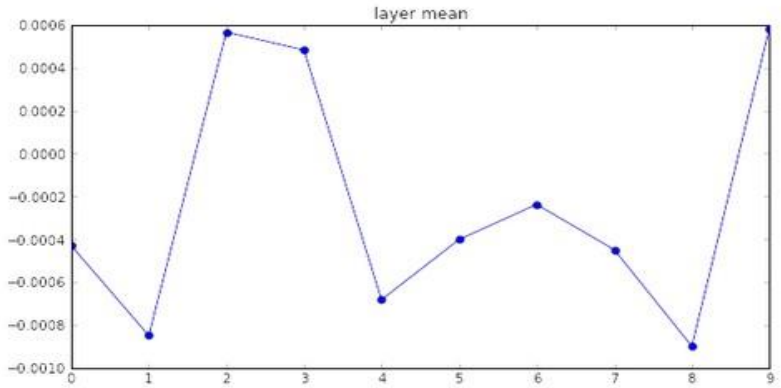
Dica: lembre que a função de ativação é aplicada após o produto interno Wx

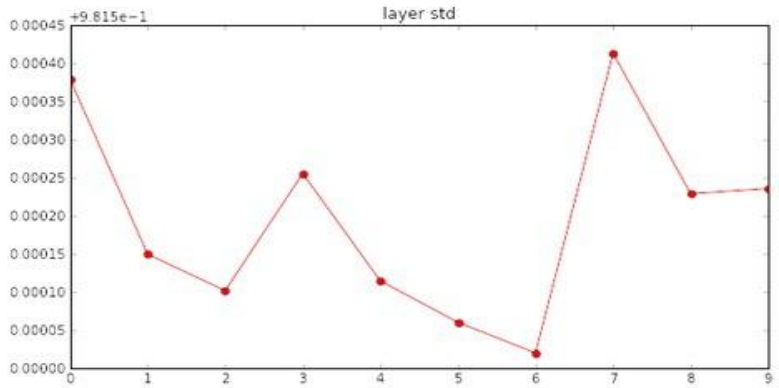
Inicialização de Pesos – Estatísticas de Ativação

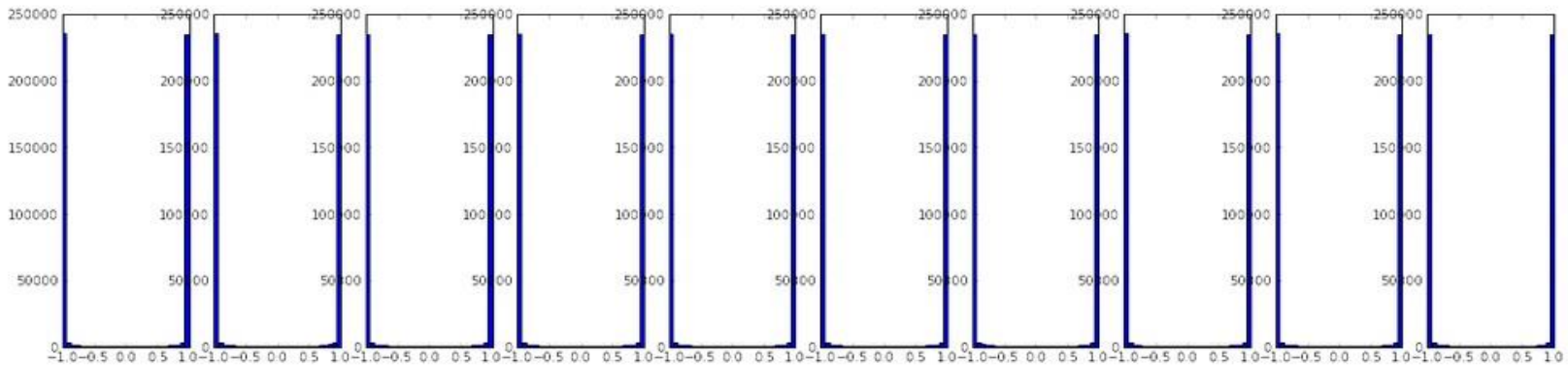
```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
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
```

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

*1.0 ao invés de *0.01





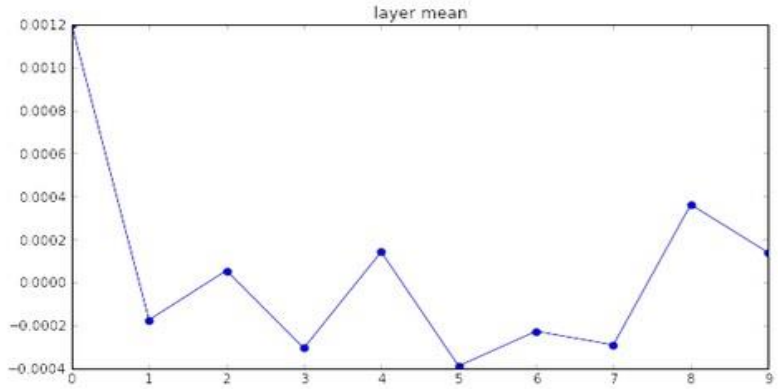


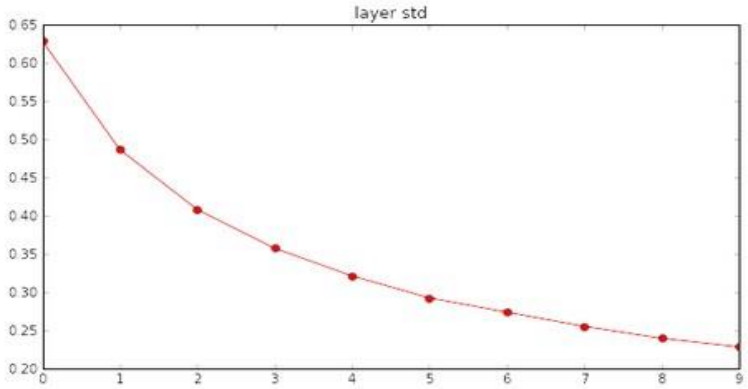
Quase todos os neurônios ficaram completamente saturados em -1 ou +1

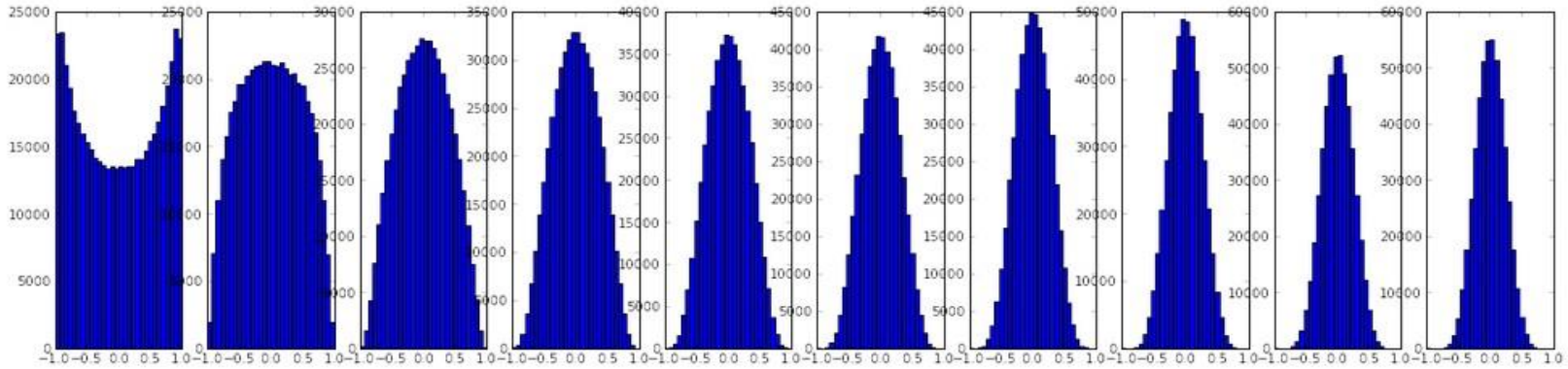
Gradientes serão todos nulos !!!

Inicialização de Pesos – Estatísticas de Ativação

```
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
```

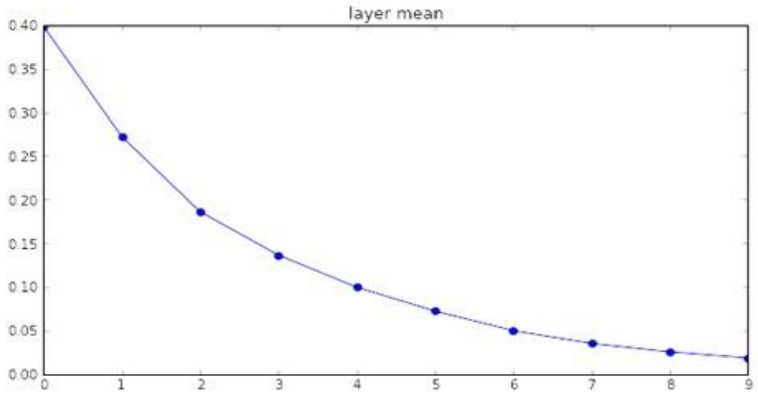
"Inicialização de Xavier"
[Glorot et al., 2010]

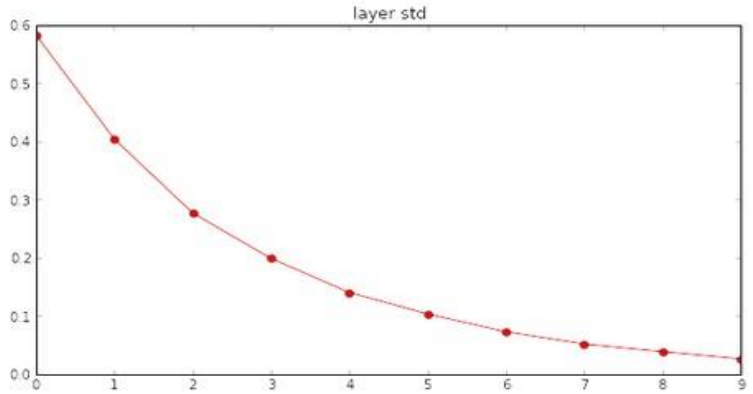
Resultados razoáveis

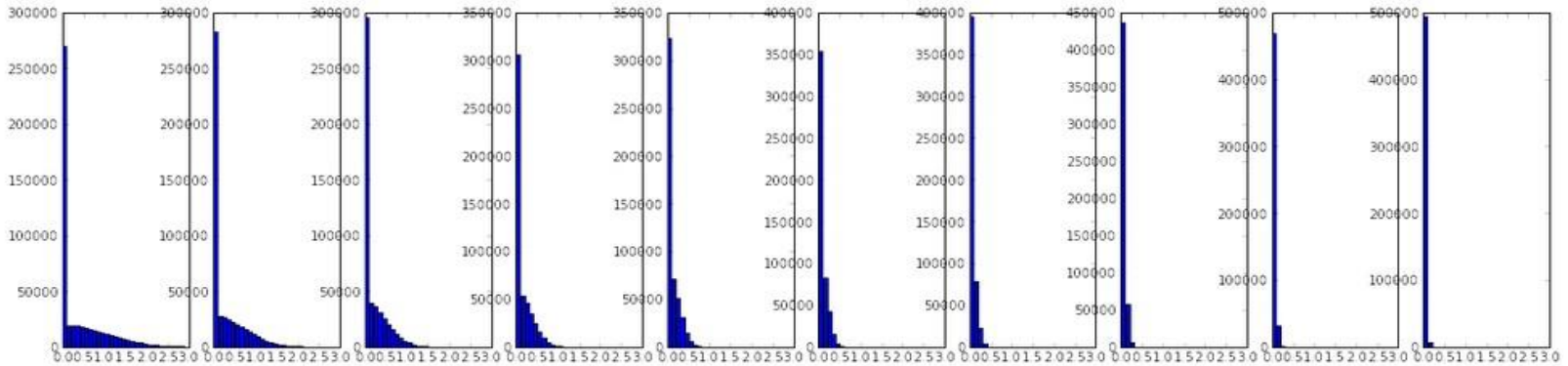
- Derivação matemática supondo ativação linear

Inicialização de Pesos – Estatísticas de Ativação

```
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
```

"Inicialização de Xavier"
[Glorot et al., 2010]

Resultados razoáveis

- Derivação matemática supondo ativação linear

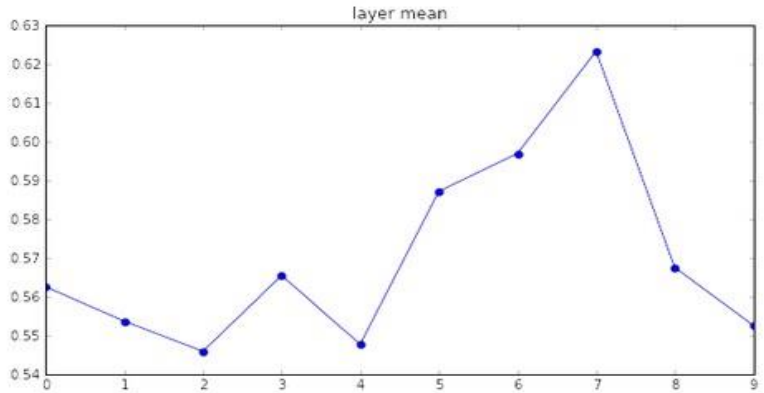
Mas ao usar ReLU, não funciona : (

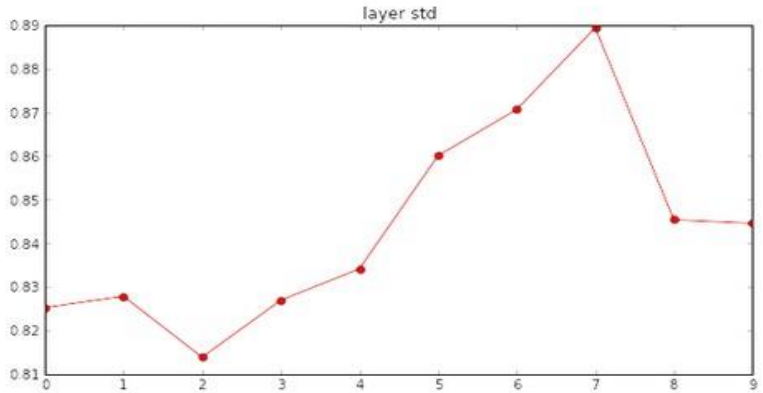
Inicialização de Pesos – Estatísticas de Ativação

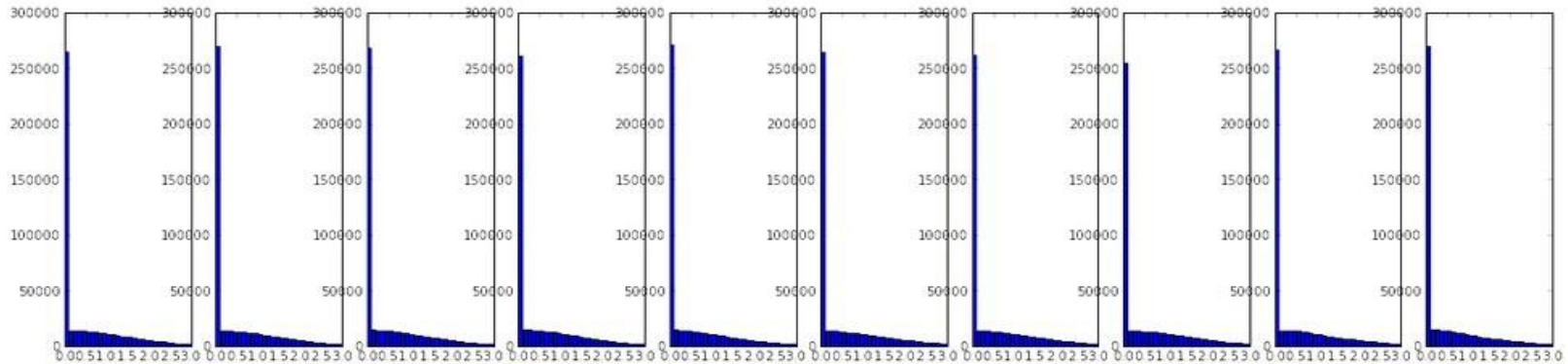
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
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
```

```
W = np.random.randn(fan in, fan out) / np.sqrt(fan in/2) # layer initialization
```

He et al., 2015
(observe a divisão por **2**)

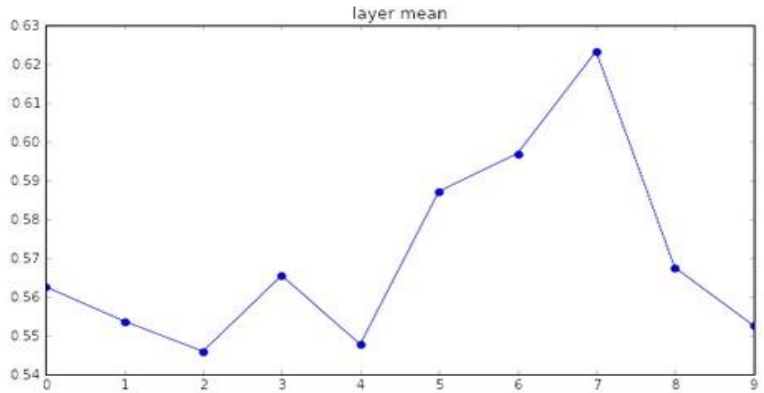


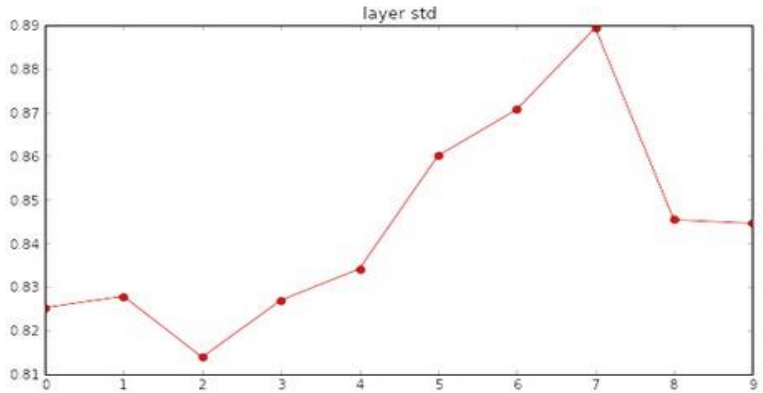


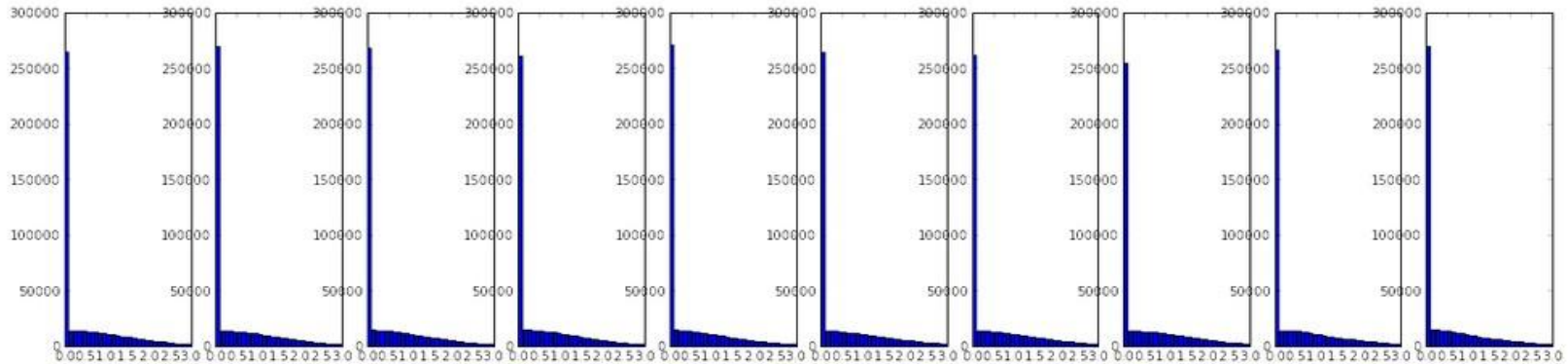


Inicialização de Pesos – Estatísticas de Ativação

```
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
```







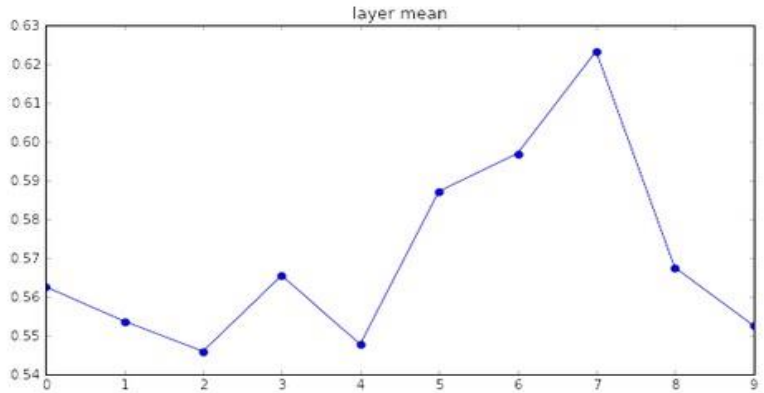
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

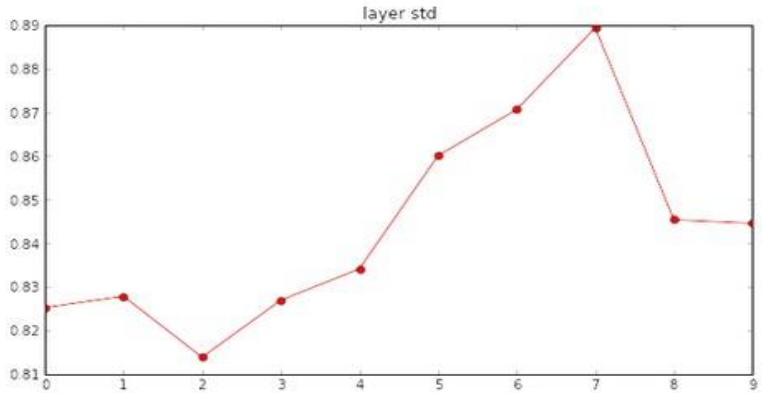
He et al., 2015
(observe a divisão por **2**)

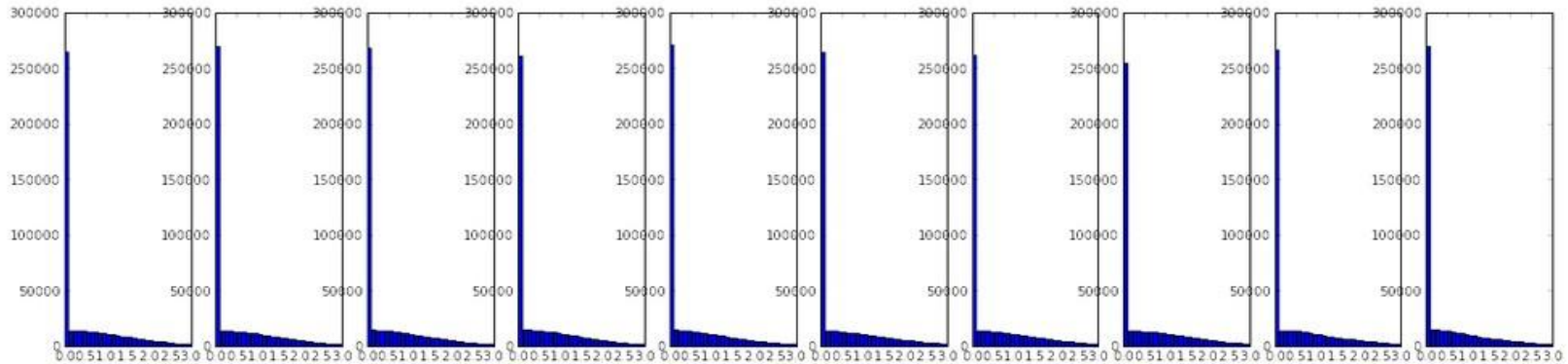
O fator 2 não parece muito, mas lembre-se de que se aplica de forma multiplicativa (por exemplo, 150 vezes na ResNet)

Inicialização de Pesos – Estatísticas de Ativação

```
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
```

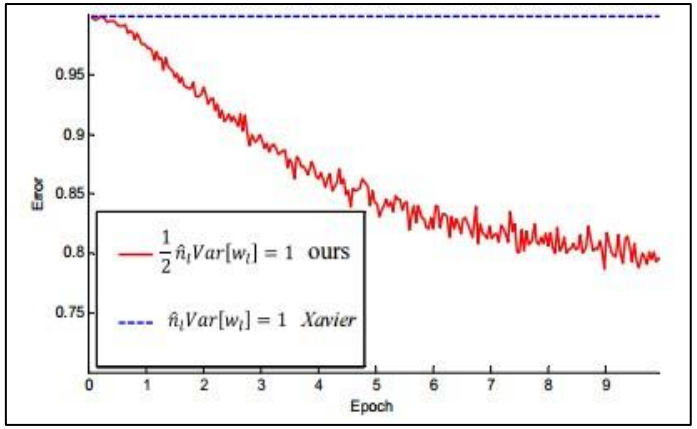






```
W = np.random.randn(fan in, fan out) / np.sqrt(fan in/2) # layer initialization
```

He et al., 2015
(observe a divisão por **2**)



Inicialização de Pesos

Initialization adequada foi uma área ativa de pesquisa…

- ***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
- …