Università degli Studi di Padova

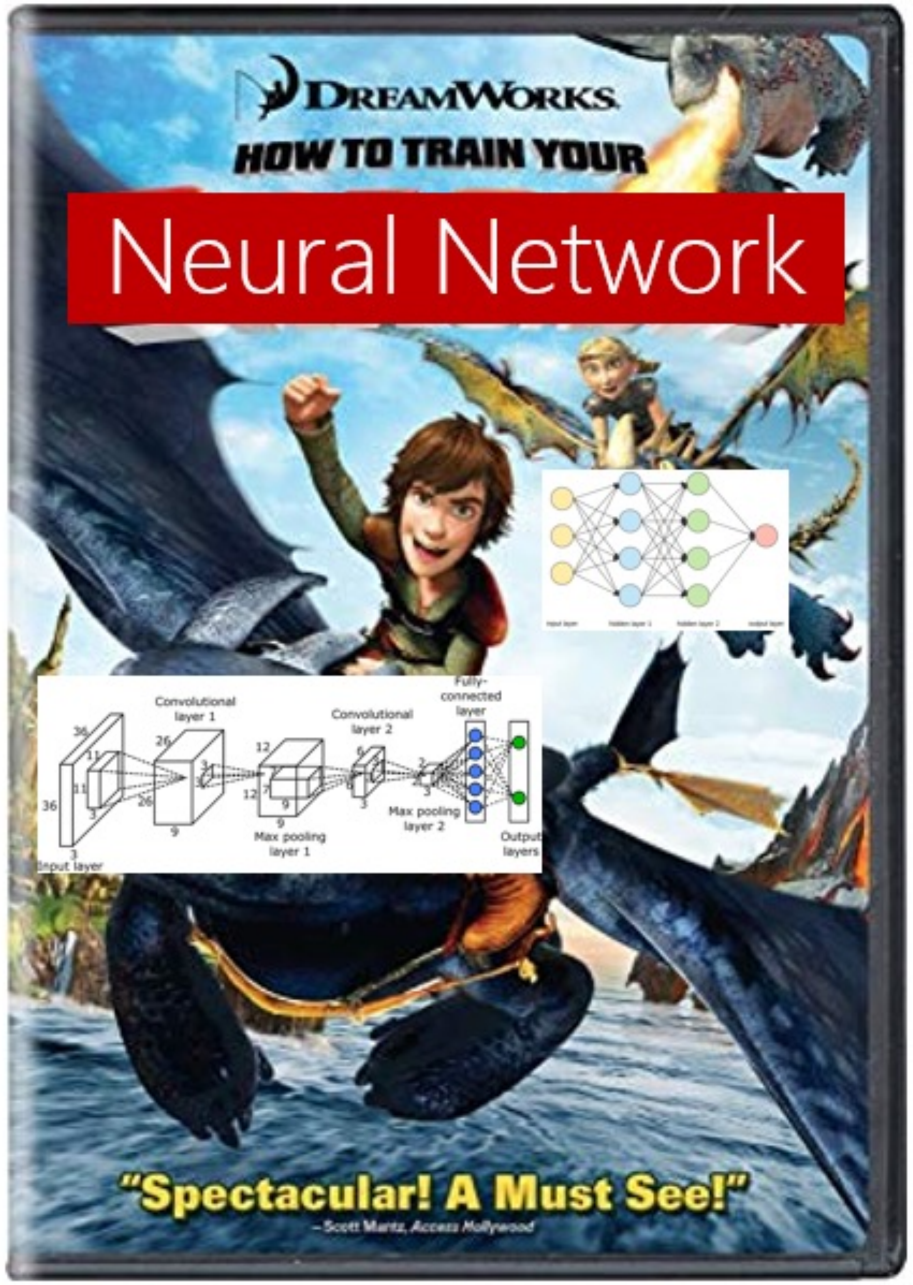
Machine Learning
2024/2025

Artificial Intelligence, Machine Learning and Control Research Group

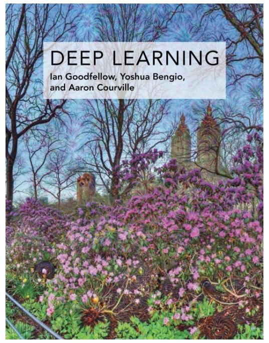
Lecture #25
Neural Networks Training – Part II

Gian Antonio Susto

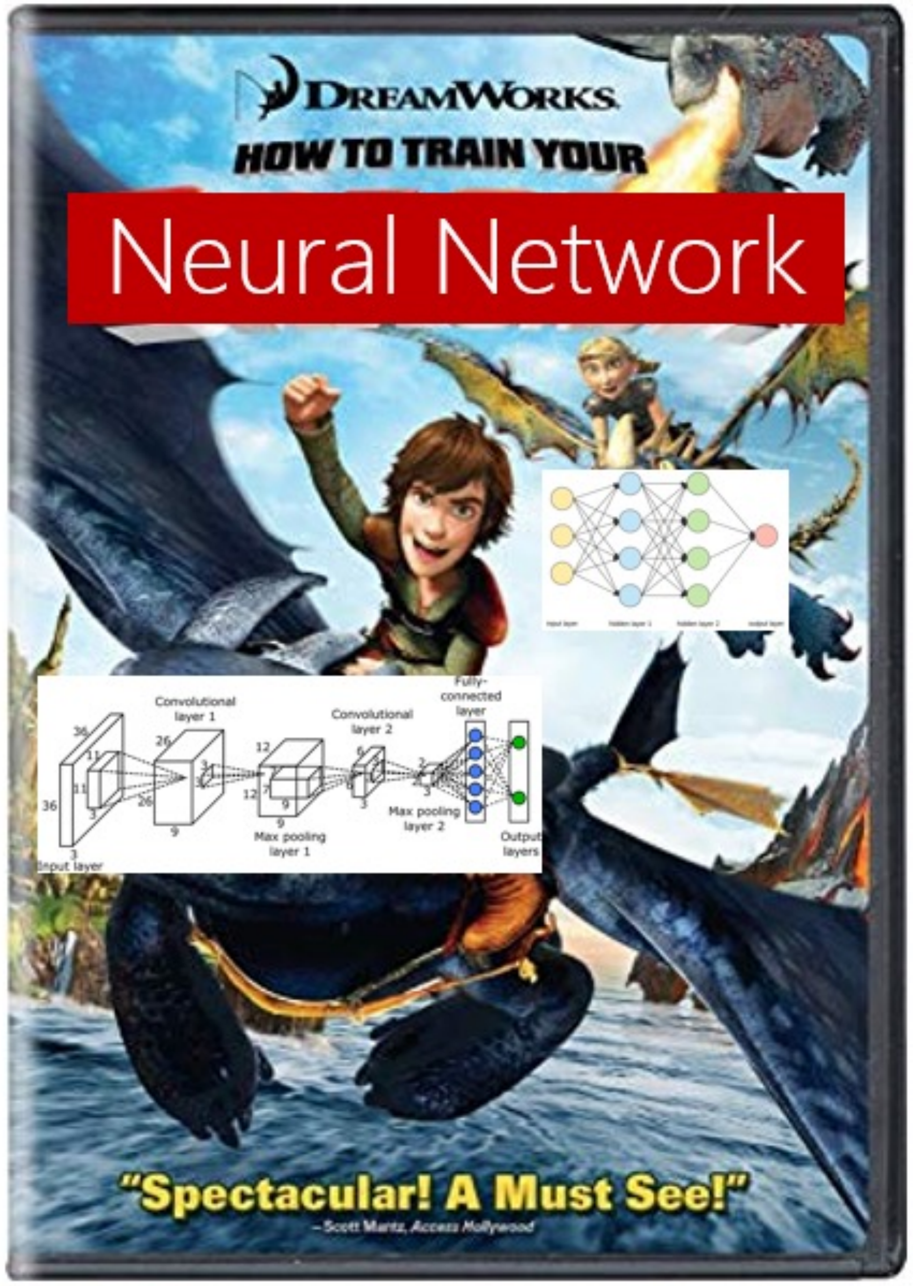
1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization



1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization



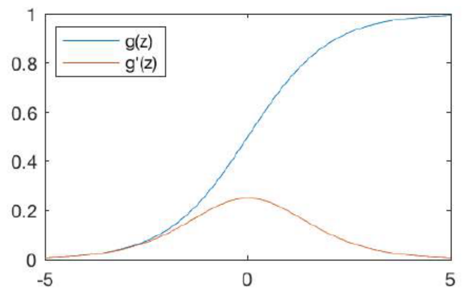
Chapter 6.3 Hidden Units



Recap: Activation functions

Beside introducing non-linearities, it is important for Activation functions to have an easy way to compute the gradients (we need this in backpropagation)

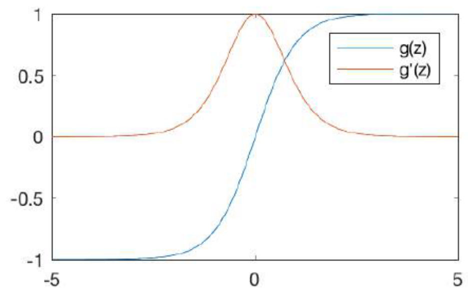
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

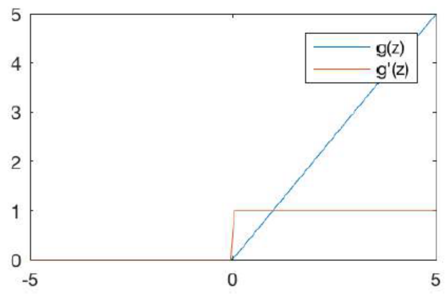
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

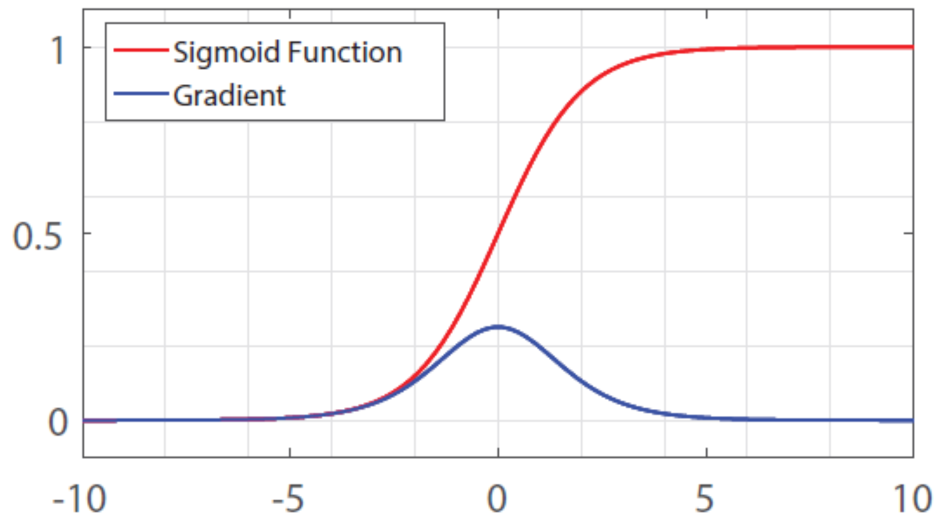
Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Recap: Sigmoid



G. Roffo *Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications*

$$g(z) = \frac{1}{1 + e^{-z}}$$

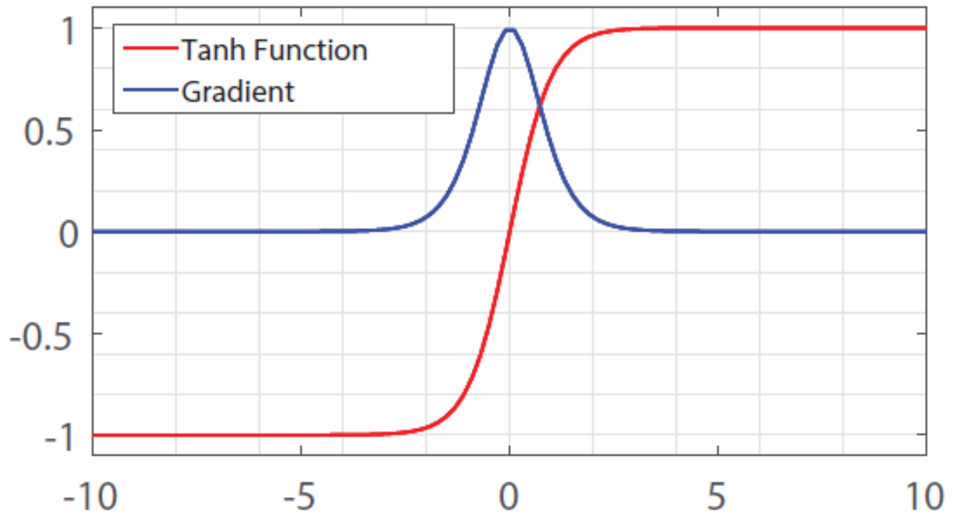
$$g'(z) = g(z)(1 - g(z))$$

Simple interpretation (probability)

Issues:

- Vanishing gradient problem: gradient becomes increasily small as the absolute value of the input increases (it is a problem in backpropagation)
- No zero-centered output (zig-zagging dynamics in the gradient updates)
- Exp function can be expensive to compute

Recap: Tanh



G. Roffo *Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications*

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

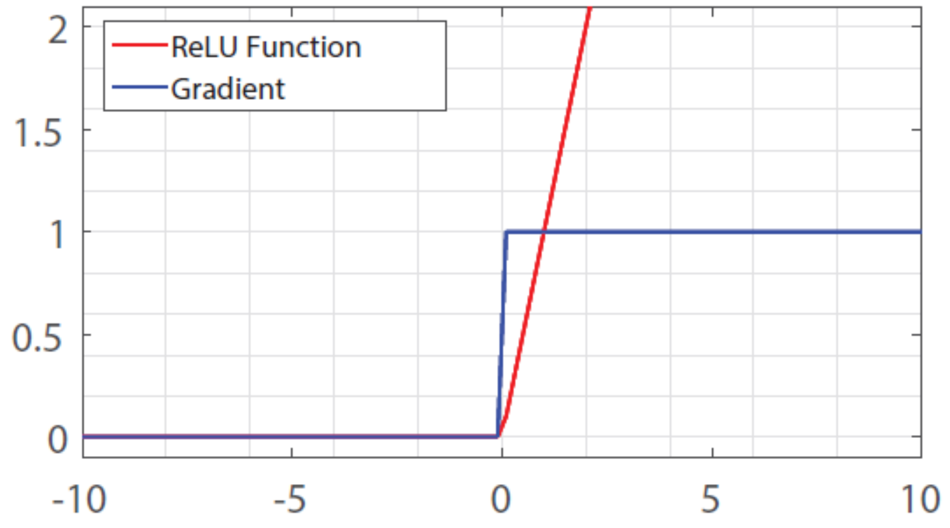
$$g'(z) = 1 - g(z)^2$$

Zero-centered

Issues:

- Vanishing gradient problem
- Exp function can be expensive to compute

Recap: ReLU



G. Roffo *Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications*

$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Simple implementation

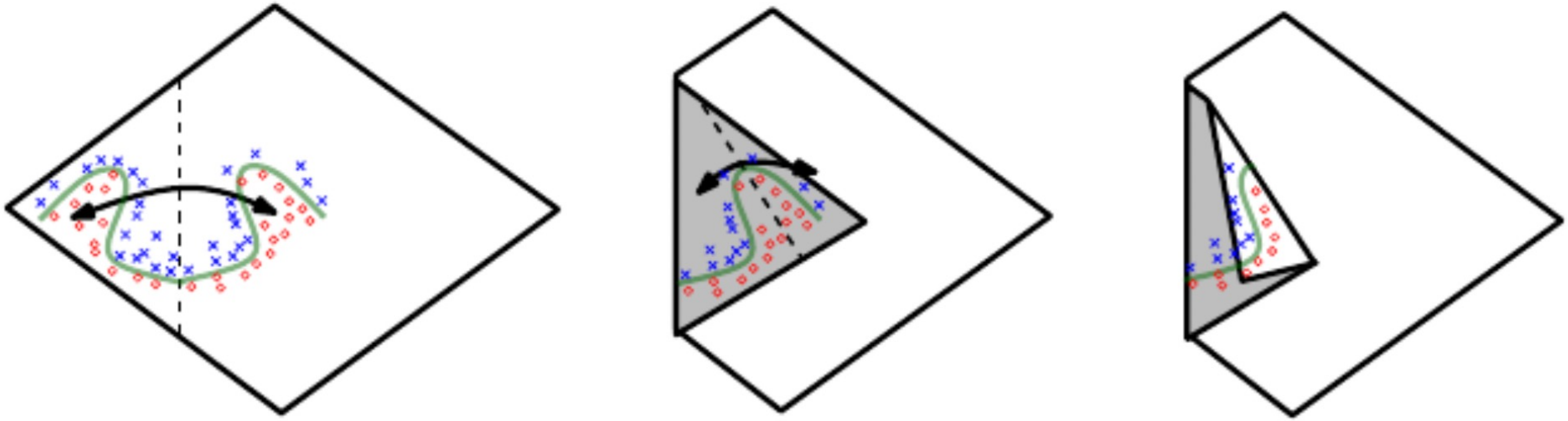
Does not saturate (in + region we have no vanishing gradient)

Faster convergence of SGD than sigmoid/tanh

Issues:

- Not zero-centered
- Gradient is zero for negative values: dead ReLU (as much as 40% of the network never activate if the learning rate is too high)

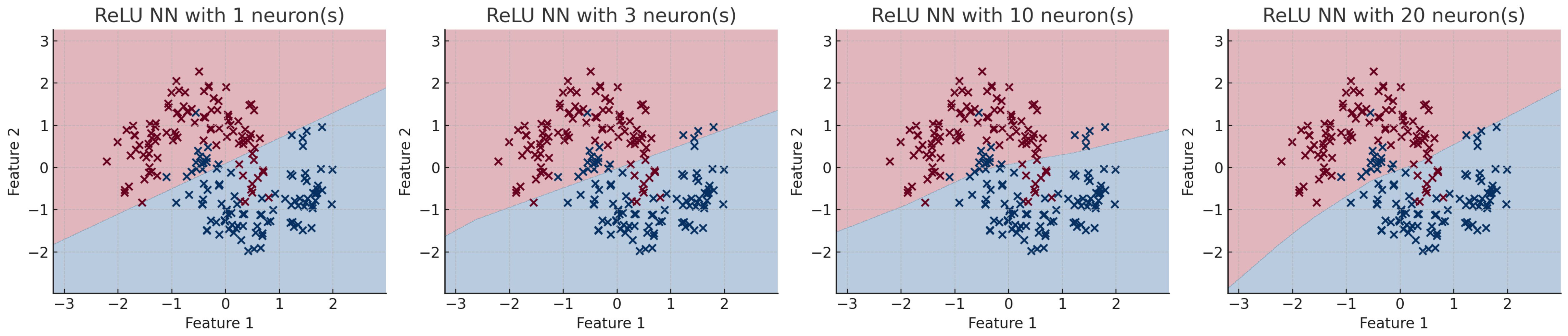
Recap: ReLU – Intuition



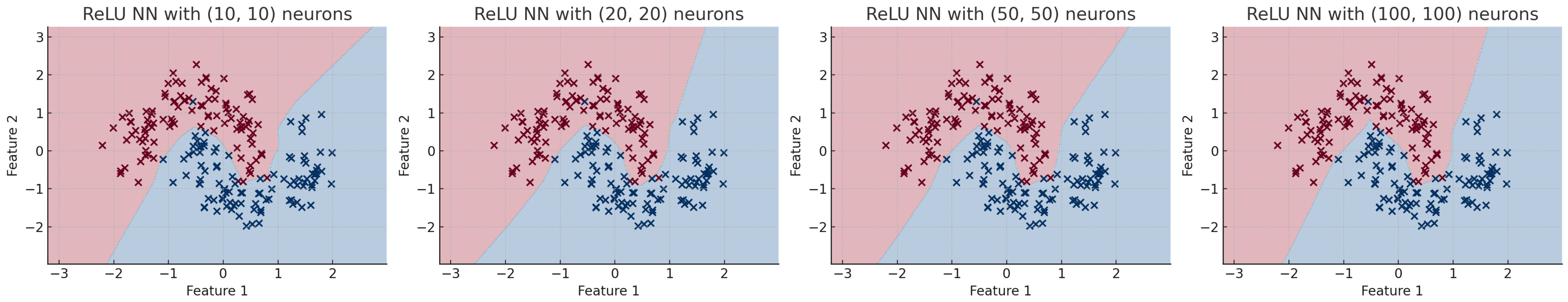
ReLU gives neural networks the power to ‘fold’ the input space.

This lets them transform complex, tangled data into something that even a straight line can separate.

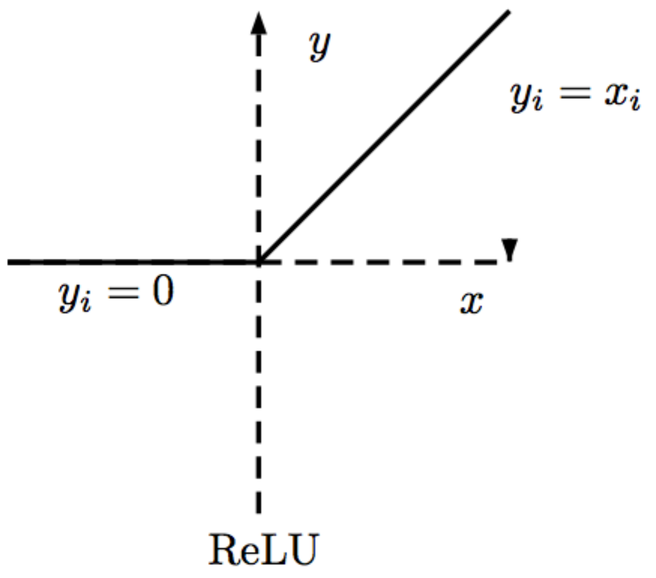
1 hidden layer

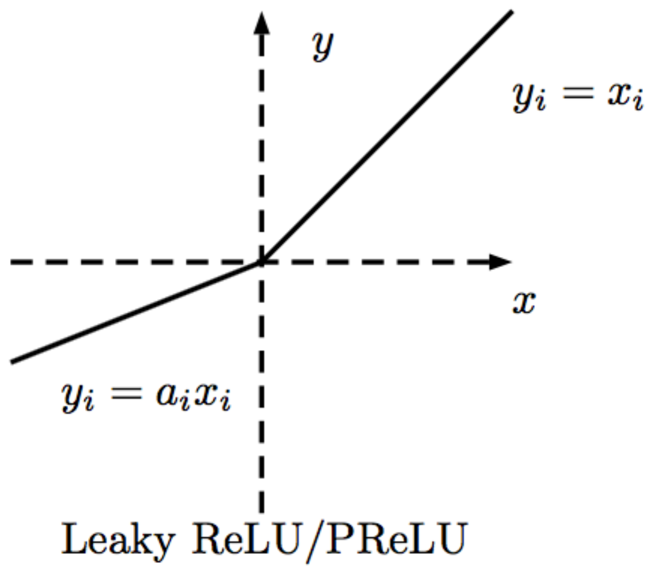


2 hidden layers



Leaky ReLU, Parametric ReLU, Randomized Leaky ReLU





$$g(z) = 0.01z, (z < 0)$$
$$g(z) = z, (z \geq 0)$$

$$g(z) = \alpha z, (z < 0)$$
$$g(z) = z, (z \geq 0)$$

Very easy to compute

Does not saturate (in + region)

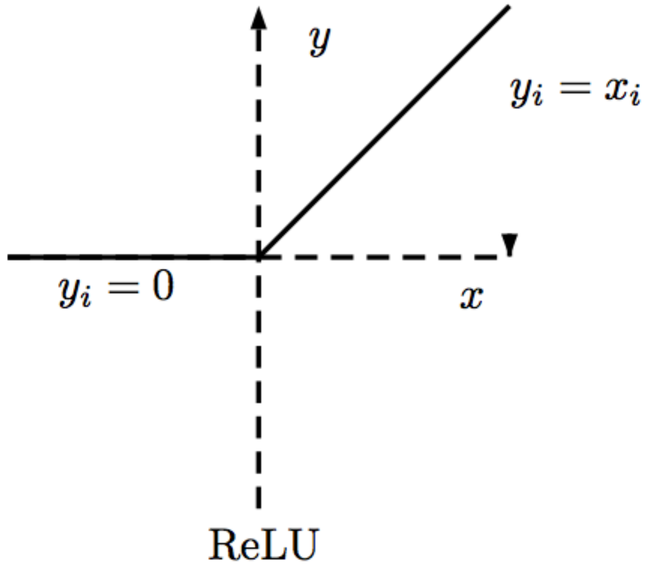
Faster convergence than sigmoid/tanh

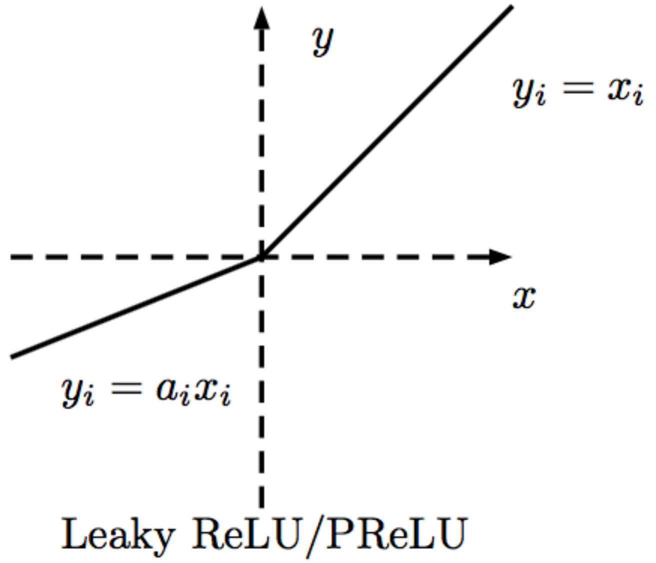
Does not die!

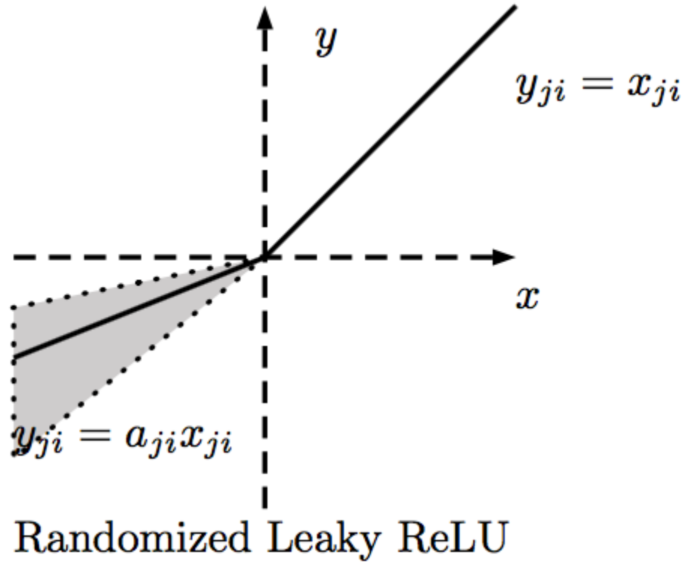
In the Parametric ReLU (PReLU) the parameter alpha is learned along with the other network parameters

https://isaacchanghau.github.io/post/activation_functions/

Leaky ReLU, Parametric ReLU, Randomized Leaky ReLU







$$g(z) = 0.01z, (z < 0)$$
$$g(z) = z, (z \geq 0)$$

$$g(z) = \alpha z, (z < 0)$$
$$g(z) = z, (z \geq 0)$$

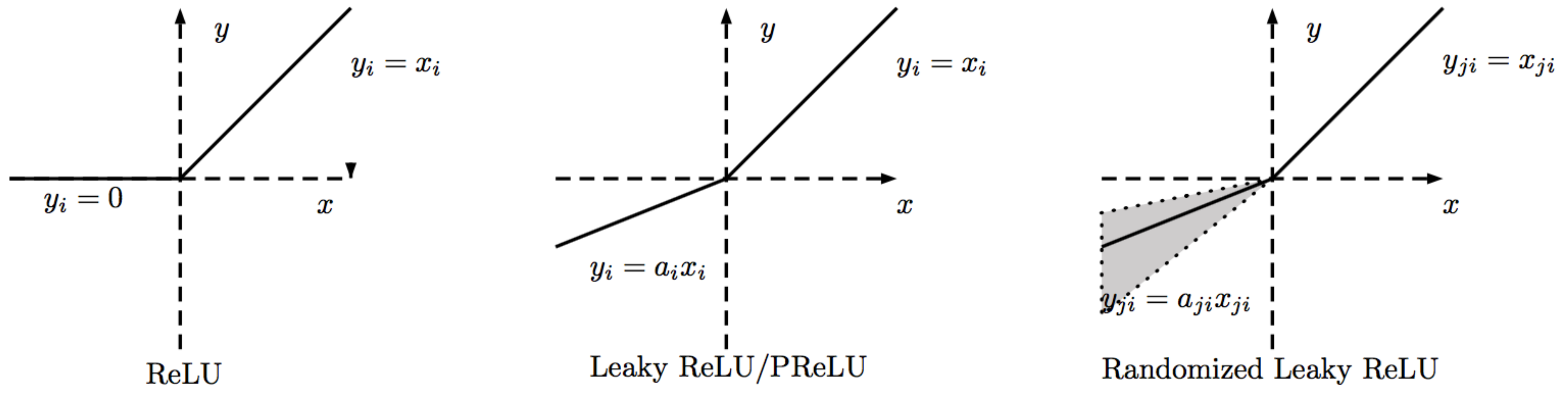
Alpha is chosen at random in a given range in training and it is then fixed in test

https://isaacchanghau.github.io/post/activation_functions/

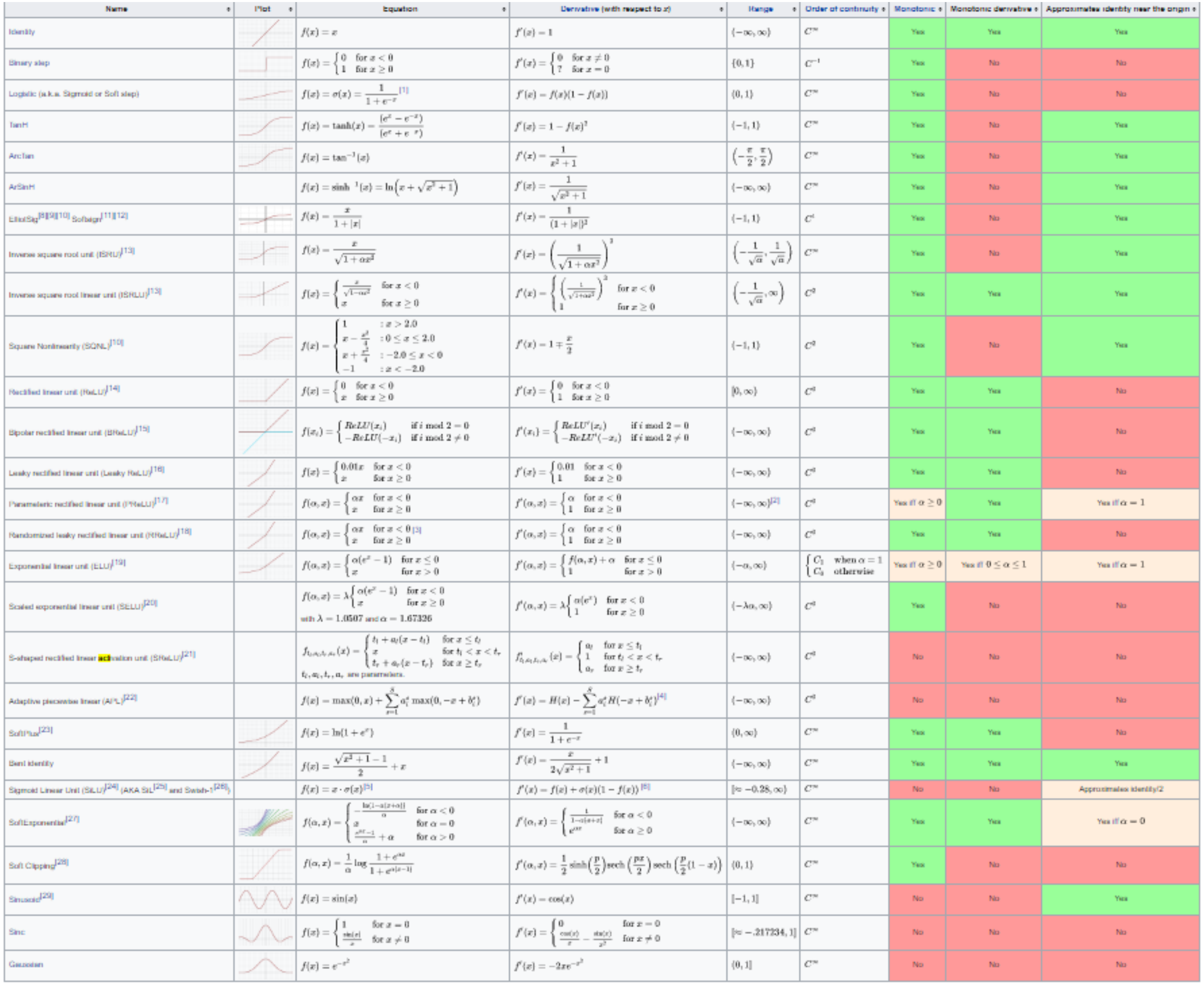
TLDR: Activations

Main issues with activation functions:

- Vanishing gradients
- Non-centered on zero outputs
- Costly computations



https://isaacchanghau.github.io/post/activation_functions/



Name	Plot	Equation	Derivative (with respect to x)	Range	Order of continuity	Monotonic	Monotonic derivative	Approximates identity near the origin
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞	Yes	Yes	Yes
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes	No	No
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$[1]	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	C^∞	Yes	No	No
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞	Yes	No	Yes
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes	No	Yes
ArSinH		$f(x) = \sinh^{-1}(x) = \ln\left(x + \sqrt{x^2+1}\right)$	$f'(x) = \frac{1}{\sqrt{x^2+1}}$	$(-\infty, \infty)$	C^∞	Yes	No	Yes
ElliotSig[8][9][10] Softsign[11][12]		$f(x) = \frac{x}{1+\lvert x \rvert}$	$f'(x) = \frac{1}{(1+\lvert x \rvert)^2}$	$(-1, 1)$	C^1	Yes	No	Yes
Inverse square root unit (ISRU)[13]		$f(x) = \frac{x}{\sqrt{1+\alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}}\right)$	C^∞	Yes	No	Yes
Inverse square root linear unit (ISRLU)[13]		$f(x) = \begin{cases} \frac{x}{\sqrt{1+\alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$\left(-\frac{1}{\sqrt{\alpha}}, \infty\right)$	C^2	Yes	Yes	Yes
Square Nonlinearity (SQNL)[10]		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$	$f'(x) = 1 \mp \frac{x}{2}$	$(-1, 1)$	C^2	Yes	No	Yes
Rectified linear unit (ReLU)[14]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$	C^0	Yes	Yes	No
Bipolar rectified linear unit (BReLU)[15]		$f(x_i) = \begin{cases} ReLU(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$f'(x_i) = \begin{cases} ReLU'(x_i) & \text{if } i \bmod 2 = 0 \\ -ReLU'(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Leaky rectified linear unit (Leaky ReLU)[16]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Parameteric rectified linear unit (PReLU)[17]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$[2]	C^0	Yes iff $\alpha \geq 0$	Yes	Yes iff $\alpha = 1$
Randomized leaky rectified linear unit (RReLU)[18]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$[3]	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes	Yes	No
Exponential linear unit (ELU)[19]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C_1 & \text{when } \alpha = 1 \\ C_0 & \text{otherwise} \end{cases}$	Yes iff $\alpha \geq 0$	Yes iff $0 \leq \alpha \leq 1$	Yes iff $\alpha = 1$
Scaled exponential linear unit (SELU)[20]		$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$	$f'(\alpha, x) = \lambda \begin{cases} \alpha(e^x) & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0	Yes	No	No
S-shaped rectified linear activation unit (SReLU)[21]		$f_{t_l, a_l, t_r, a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ t_l, a_l, t_r, a_r are parameters.	$f'_{t_l, a_l, t_r, a_r}(x) = \begin{cases} a_l & \text{for } x \leq t_l \\ 1 & \text{for } t_l < x < t_r \\ a_r & \text{for } x \geq t_r \end{cases}$	$(-\infty, \infty)$	C^0	No	No	No
Adaptive piecewise linear (APL)[22]		$f(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$	$f'(x) = H(x) - \sum_{s=1}^{S} a_i^s H(-x + b_i^s)$[4]	$(-\infty, \infty)$	C^0	No	No	No
SoftPlus[23]		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞	Yes	Yes	No
Bent identity		$f(x) = \frac{\sqrt{x^2+1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2+1}} + 1$	$(-\infty, \infty)$	C^∞	Yes	Yes	Yes
Sigmoid Linear Unit (SiLU)[24] (AKA SiL[25] and Swish-1[26])		$f(x) = x \cdot \sigma(x)$[5]	$f'(x) = f(x) + \sigma(x)(1 - f(x))$[6]	$[\approx -0.28, \infty)$	C^∞	No	No	Approximates identity/2
SoftExponential[27]		$f(\alpha, x) = \begin{cases} -\frac{\ln(1 - \alpha(x + \alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1 - \alpha(\alpha + x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$	$(-\infty, \infty)$	C^∞	Yes	Yes	Yes iff $\alpha = 0$
Soft Clipping[28]		$f(\alpha, x) = \frac{1}{\alpha} \log \frac{1 + e^{\alpha x}}{1 + e^{\alpha(x-1)}}$	$f'(\alpha, x) = \frac{1}{2} \sinh\left(\frac{p}{2}\right) \operatorname{sech}\left(\frac{px}{2}\right) \operatorname{sech}\left(\frac{p}{2}(1 - x)\right)$	$(0, 1)$	C^∞	Yes	No	No
Sinusoid[29]		$f(x) = \sin(x)$	$f'(x) = \cos(x)$	$[-1, 1]$	C^∞	No	No	Yes
Sinc		$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$	$[\approx -.217234, 1]$	C^∞	No	No	No
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$	$(0, 1]$	C^∞	No	No	No

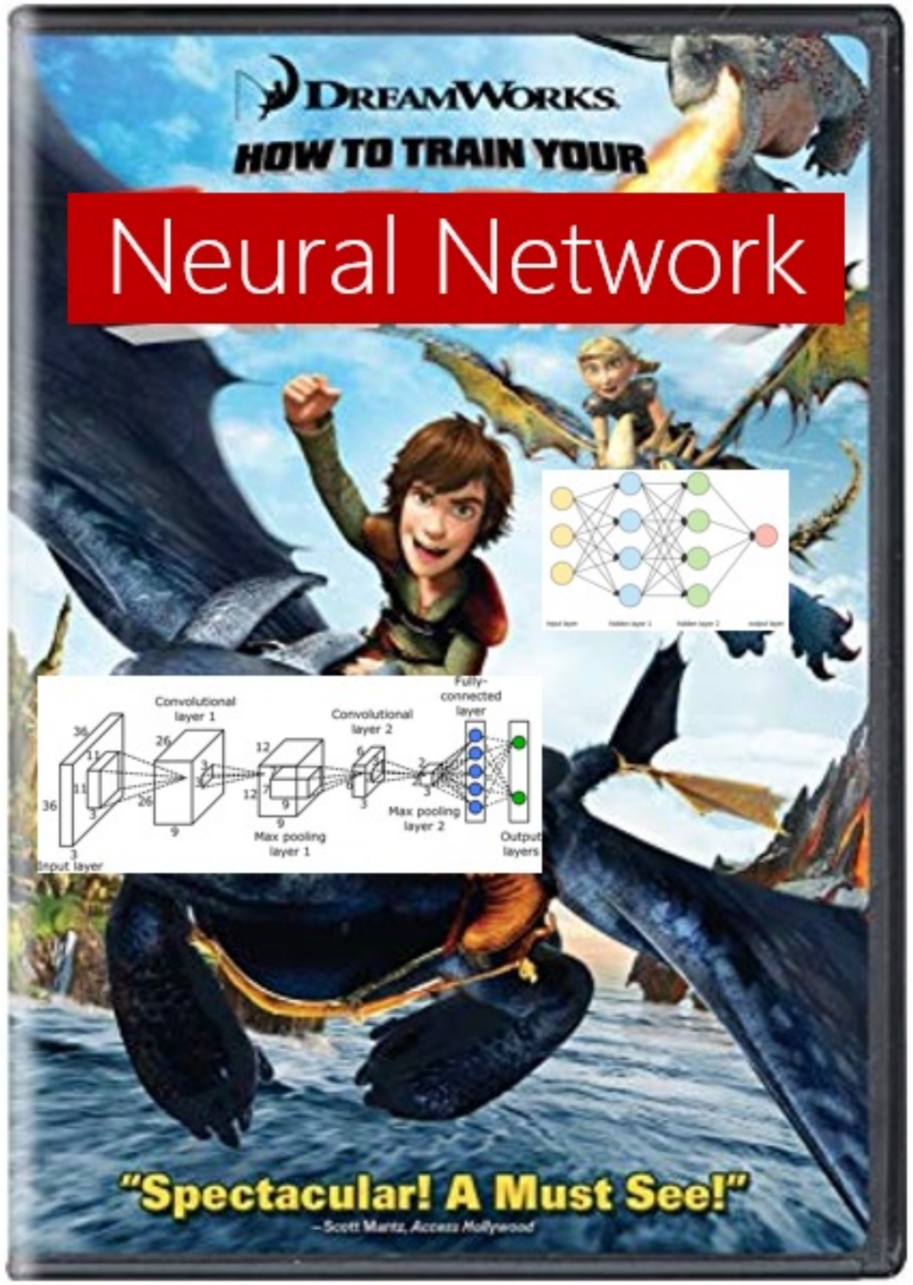
https://en.wikipedia.org/wiki/Activation_function

Which activation?

In practice:

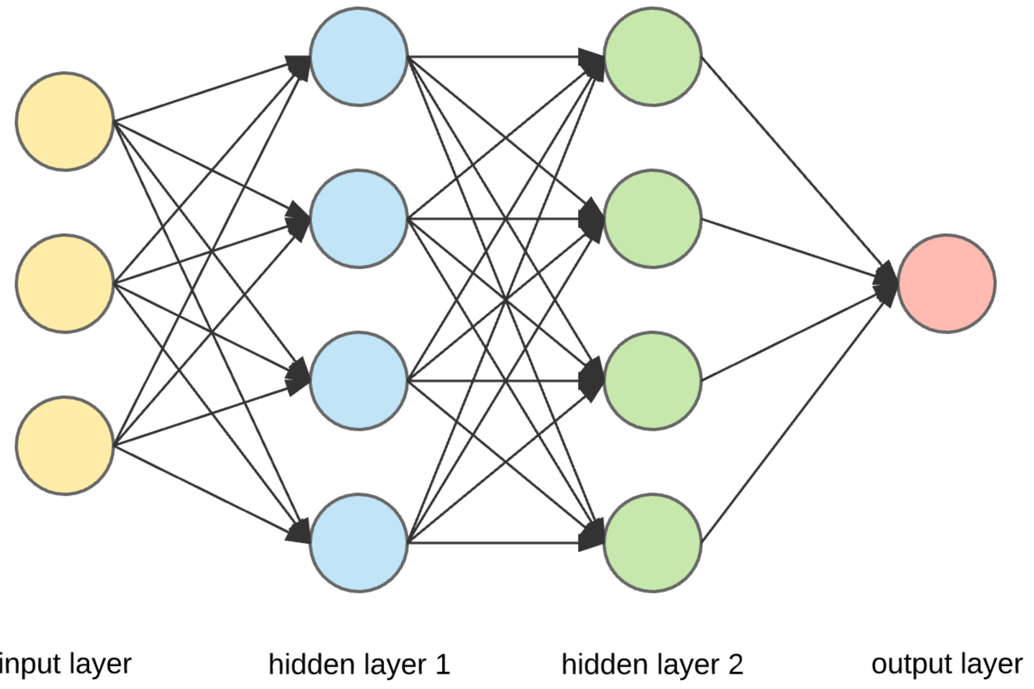
- prefer ReLU. Use slightly positive initial bias to avoid dead Relu issue.
- Try out Leaky ReLU/PRelu
- No sigmoid!

1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization



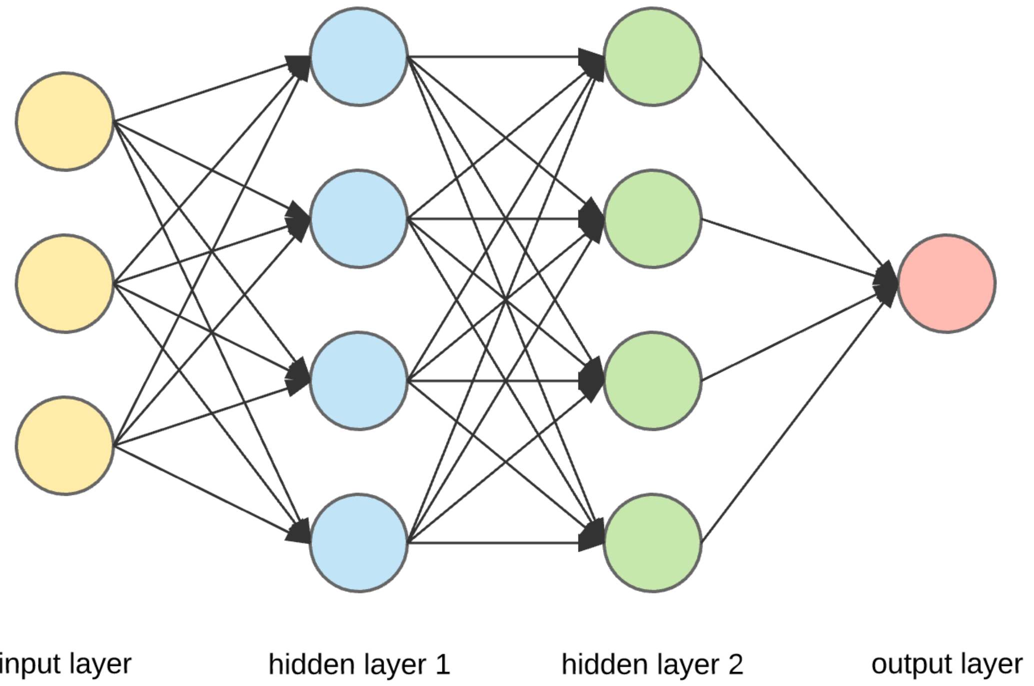
Initialization

First idea: all weights equals to 0



Initialization

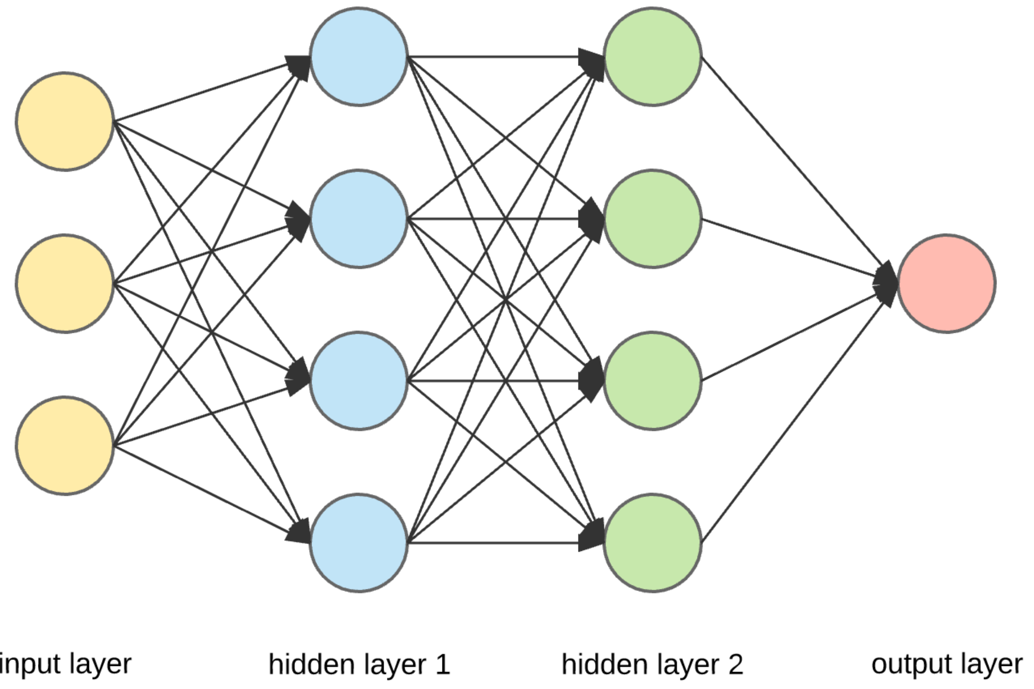
First idea: all weights equals to 0... not a great idea! All the neurons will output the same thing (we will get the same gradients), but we would like the neurons to learn different things!



Initialization

Second idea: all weights small random numbers

$\sim\mathcal{N}(0,\sigma^2)$ (with small theta)

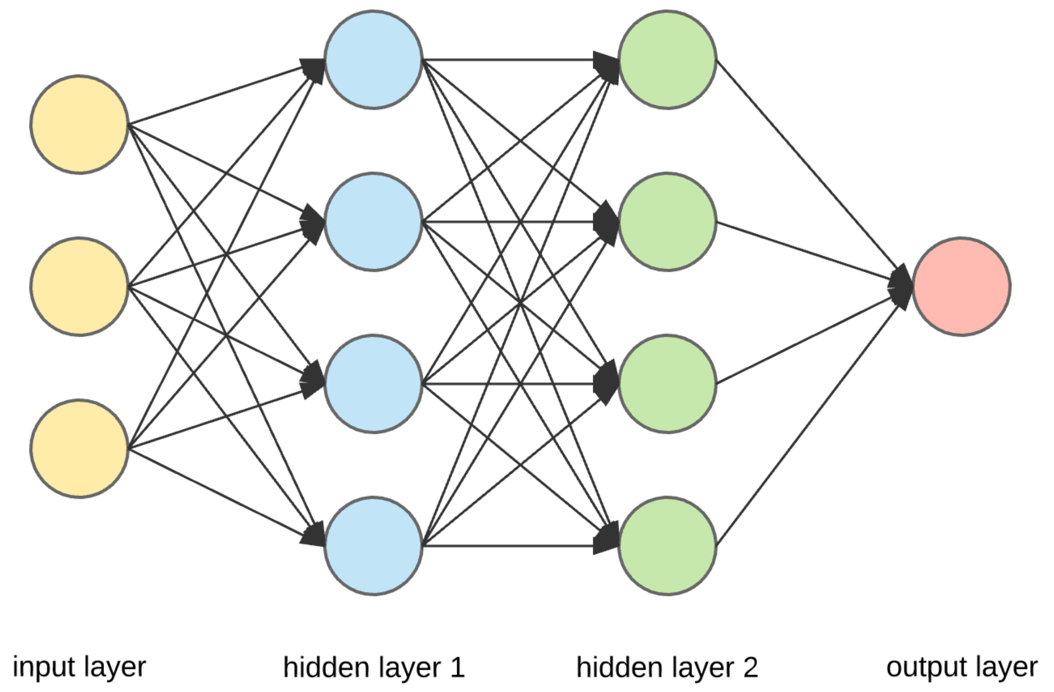


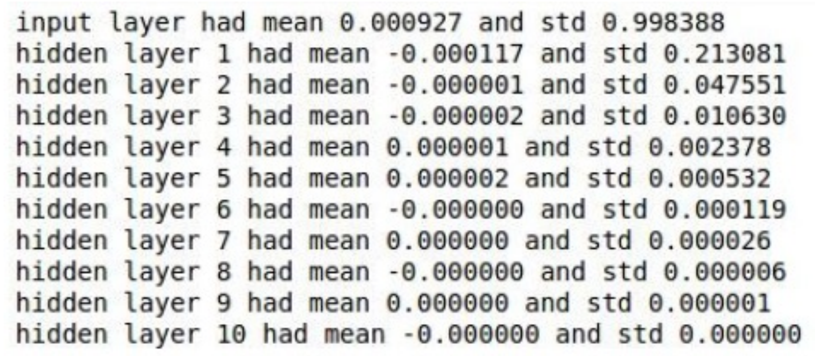
Initialization

Second idea: all weights small random numbers

(with small theta)

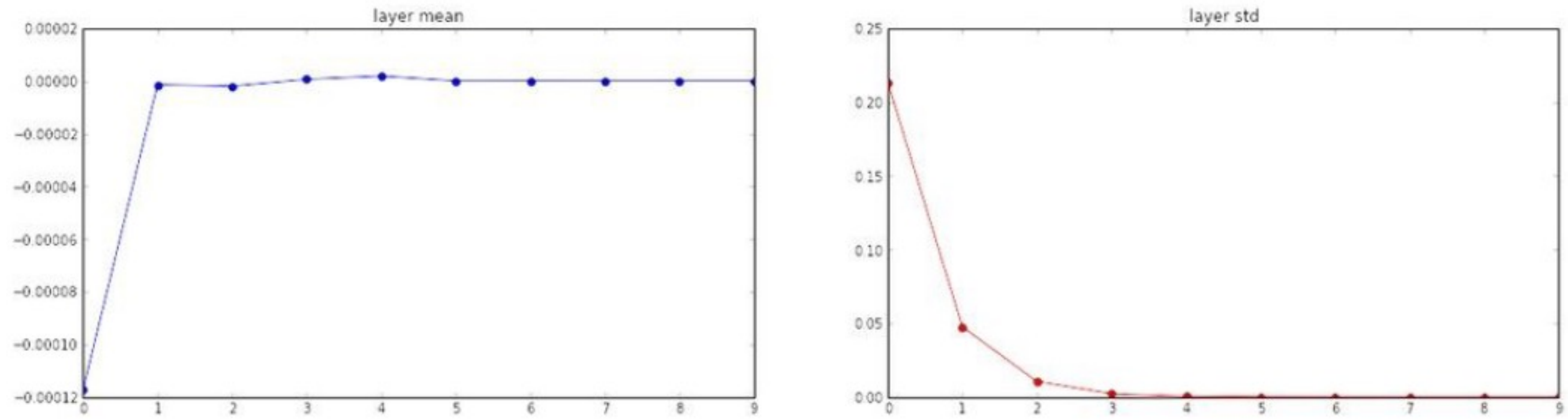
It just works with small networks...

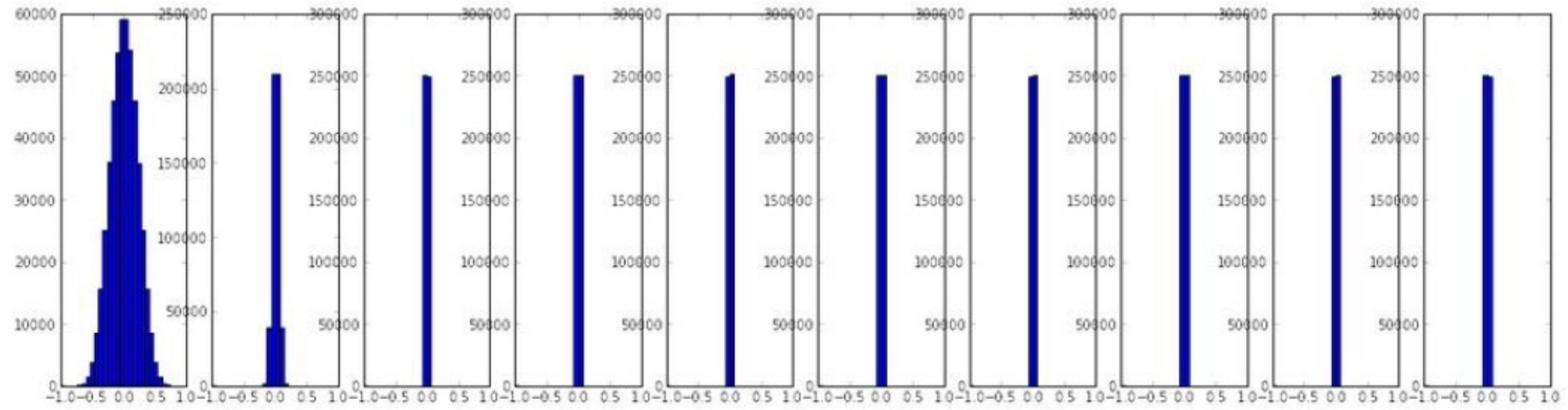




```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532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

A 10-layer NN with 500 neurons per layer: let us see the mean and std of the activations (output) for each layer





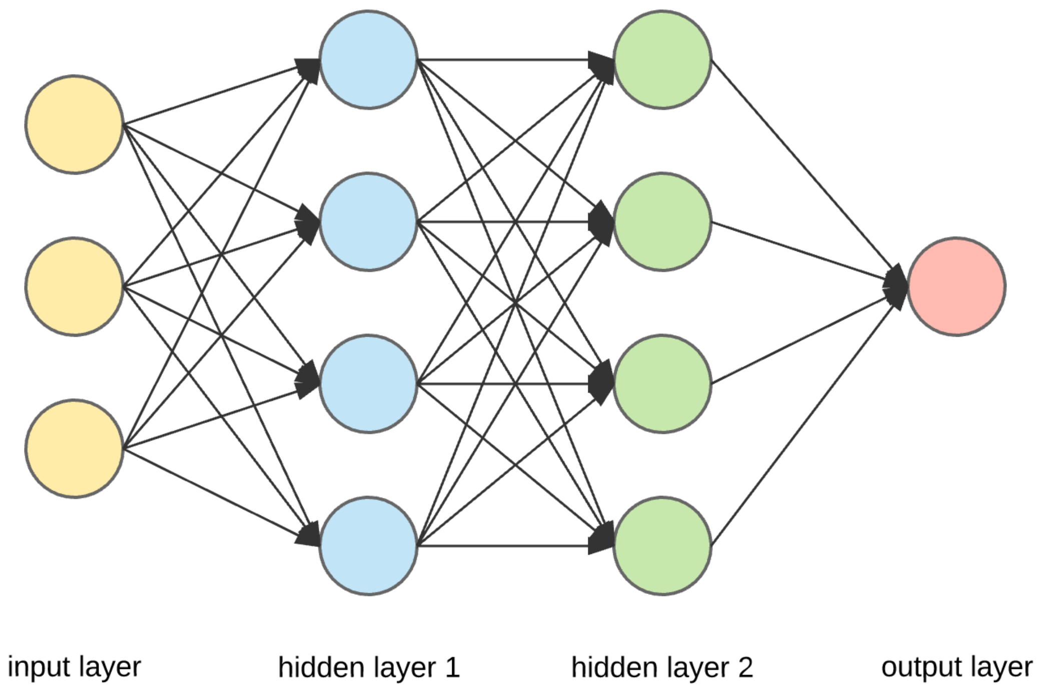
Std shrinks over time: vanishing gradient!

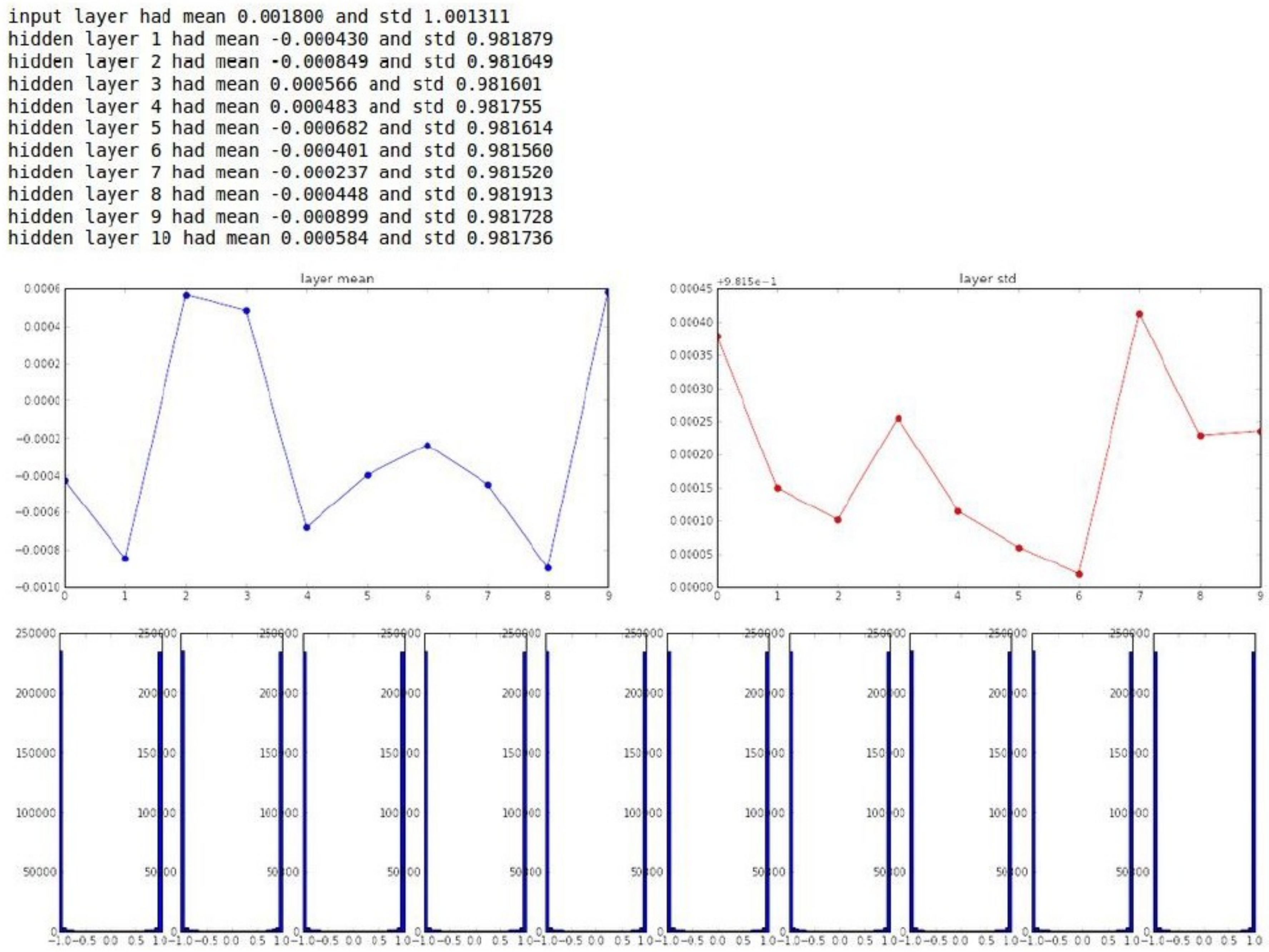
F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition*
http://cs231n.stanford.edu/

Initialization

Second idea: all weights small random numbers

$\sim\mathcal{N}(0, \sigma^2)$ (with big theta, ie. 1)

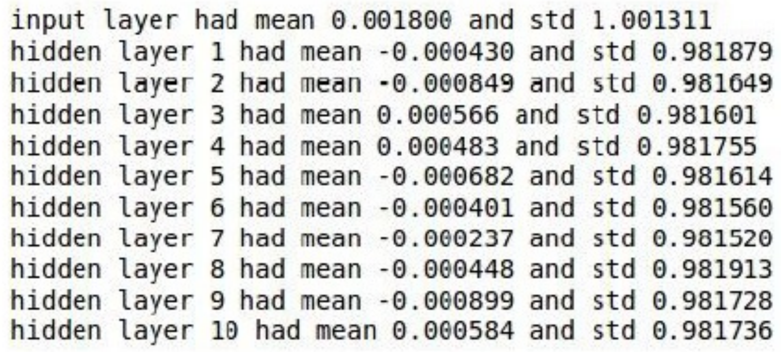




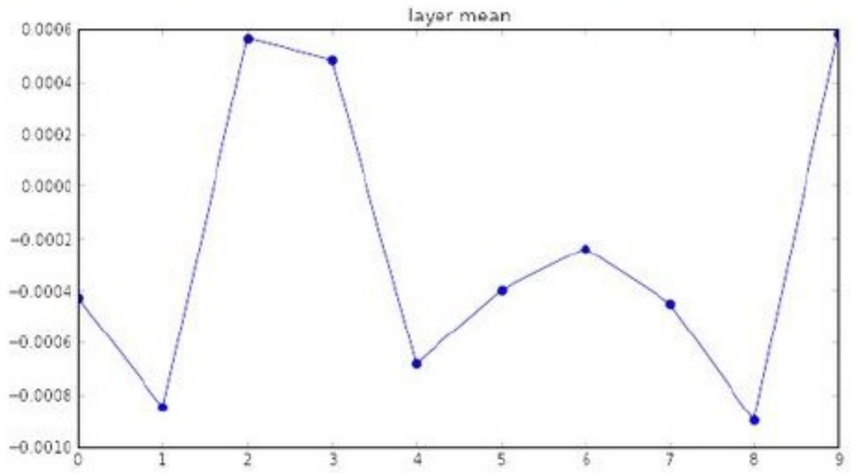
```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614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

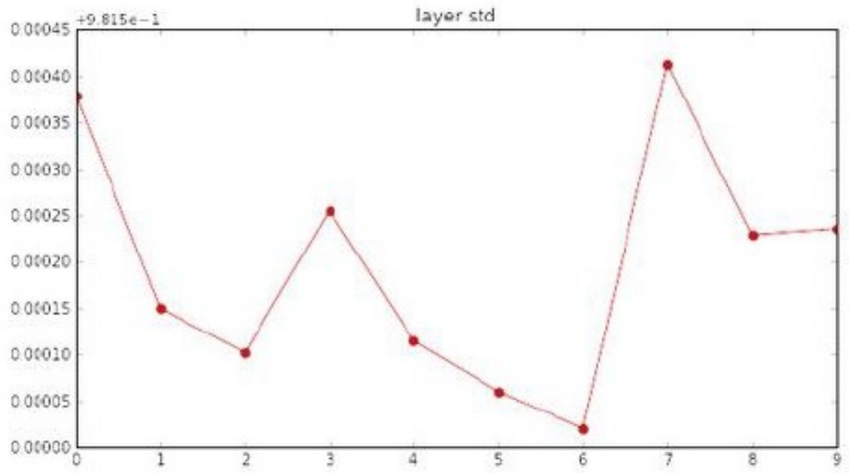
Weights are big and the neurons are always saturating (with tanh or sigmoid)

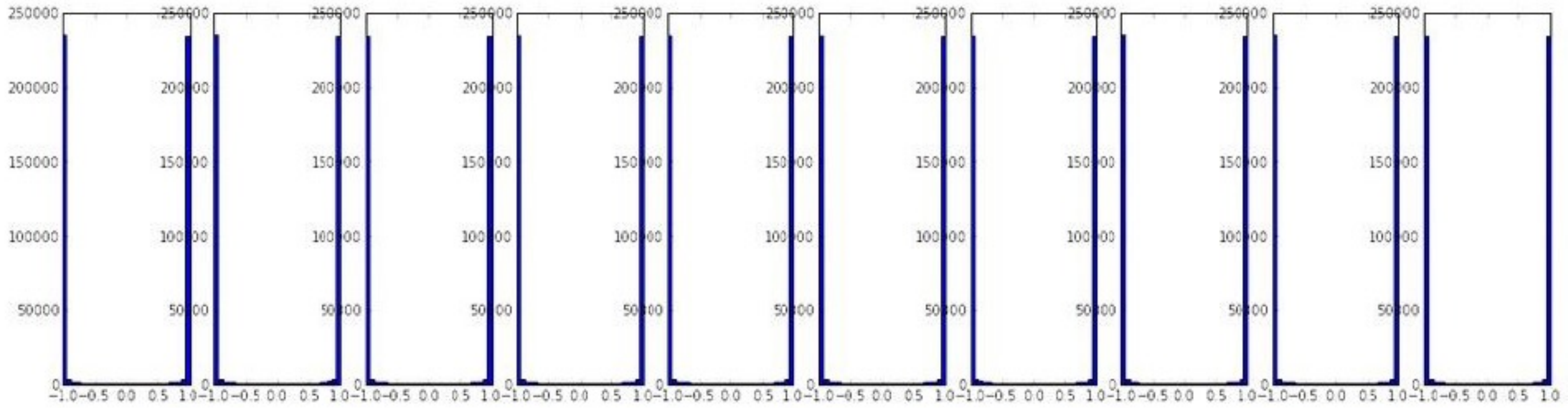
F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/



```
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
```







Weights are big and the neurons are always saturating (with tanh or sigmoid)

How to make it right?

Initialization

In principle: Glorot (a.k.a. Xavier) initialization (default in Keras, e.g.) can be a good starting point.

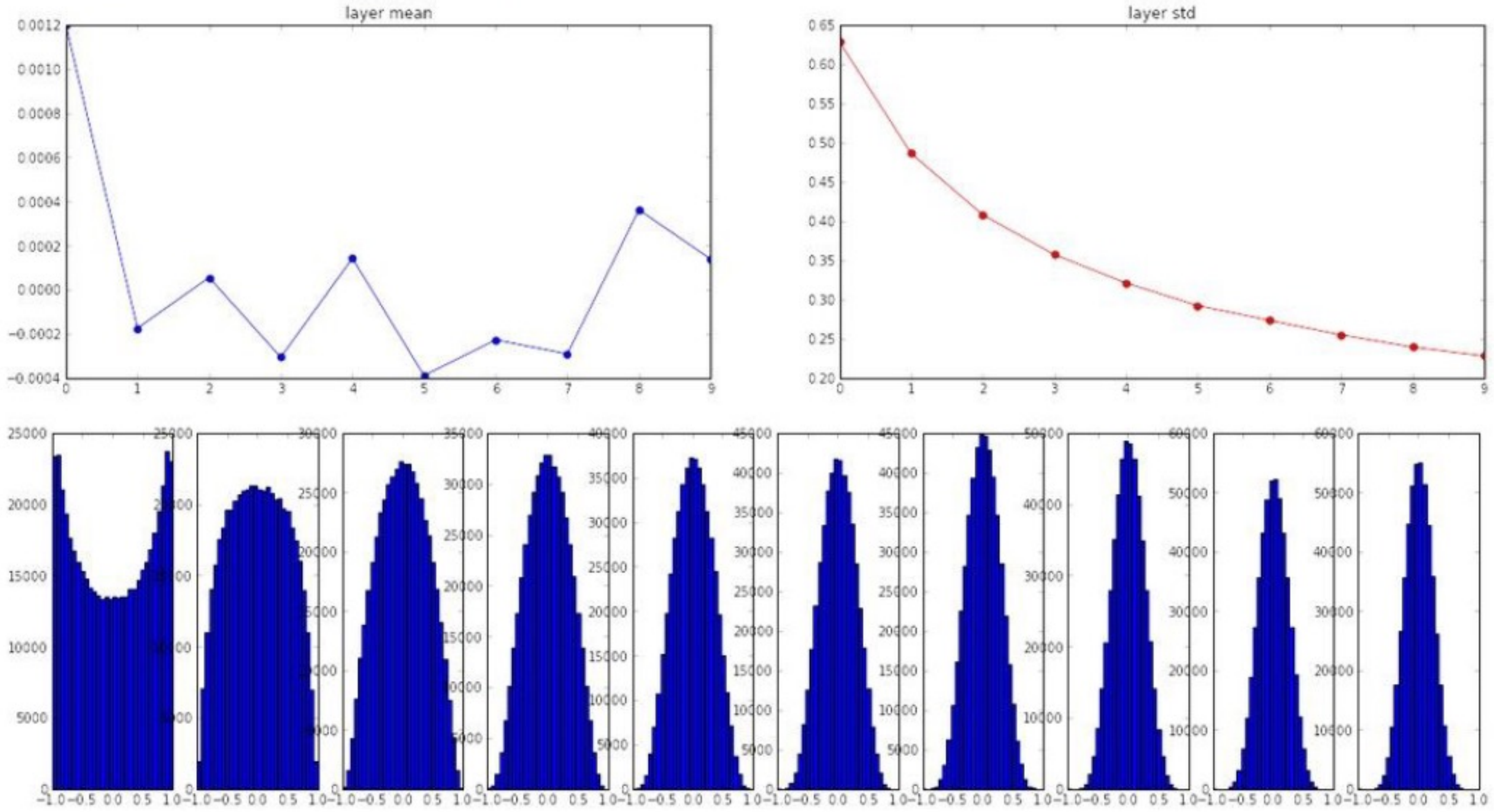
With Glorot initialization, you keep the weights in the right range (not too small, not too big).

Weights are chosen from a Gaussian distribution with zero mean and variance that is associated with the number of nodes at that layer (n_IN) and number of neurons the result is fed to (n_OUT)

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

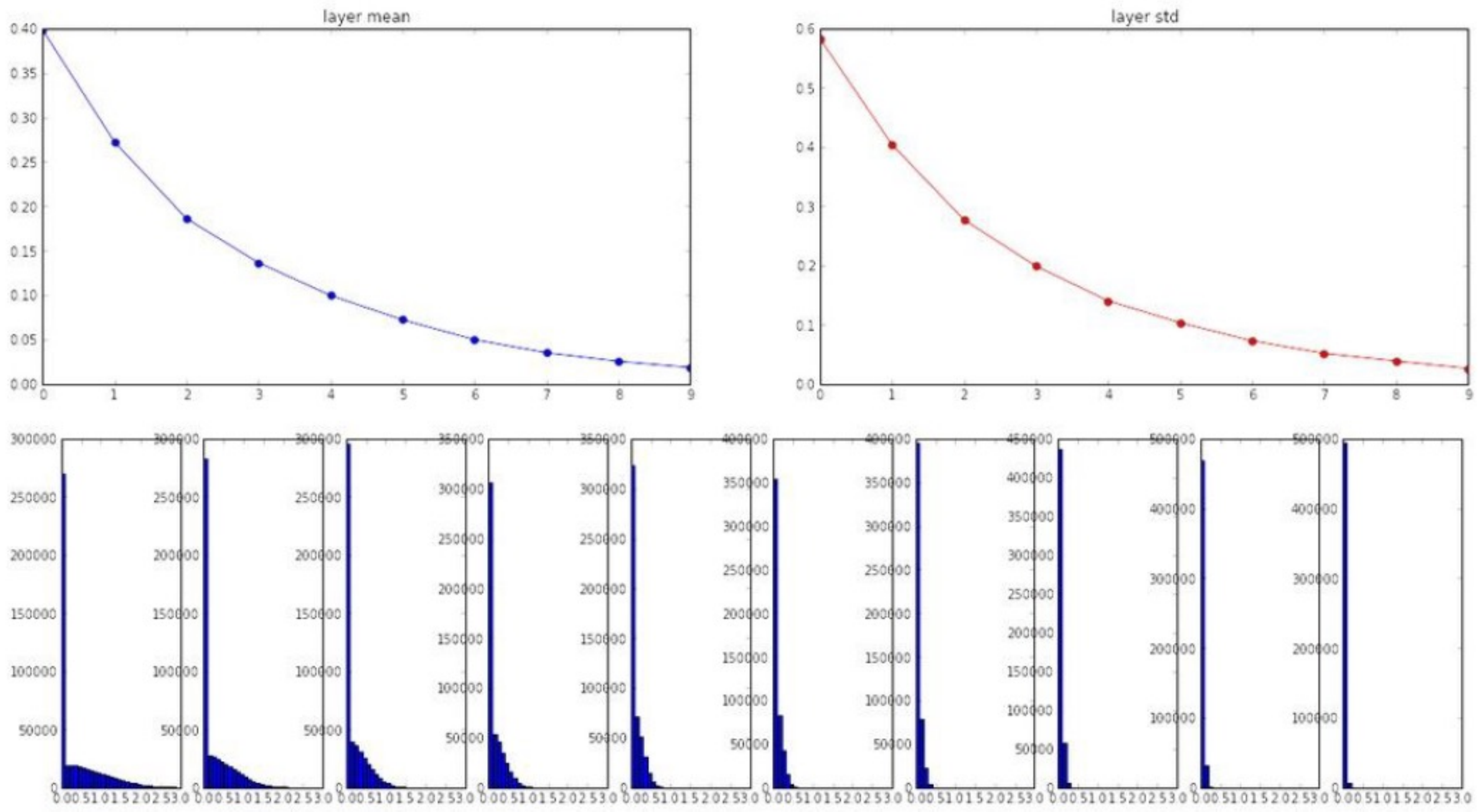
-X. Glorot, Y. Bengio *Understanding the difficulty of training deep feedforward neural networks*. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256), 2010

```
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
```



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

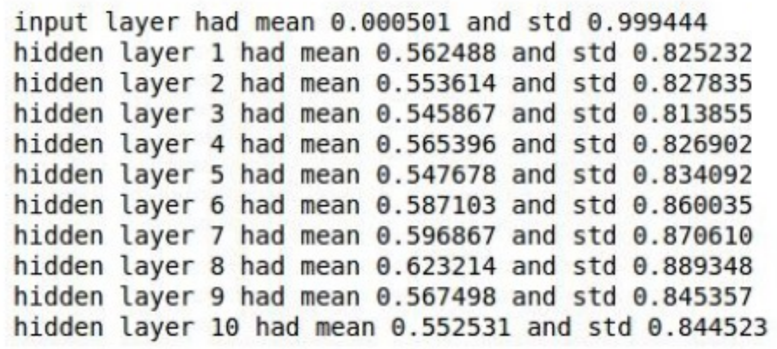
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
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
```



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Do not work well with ReLU...

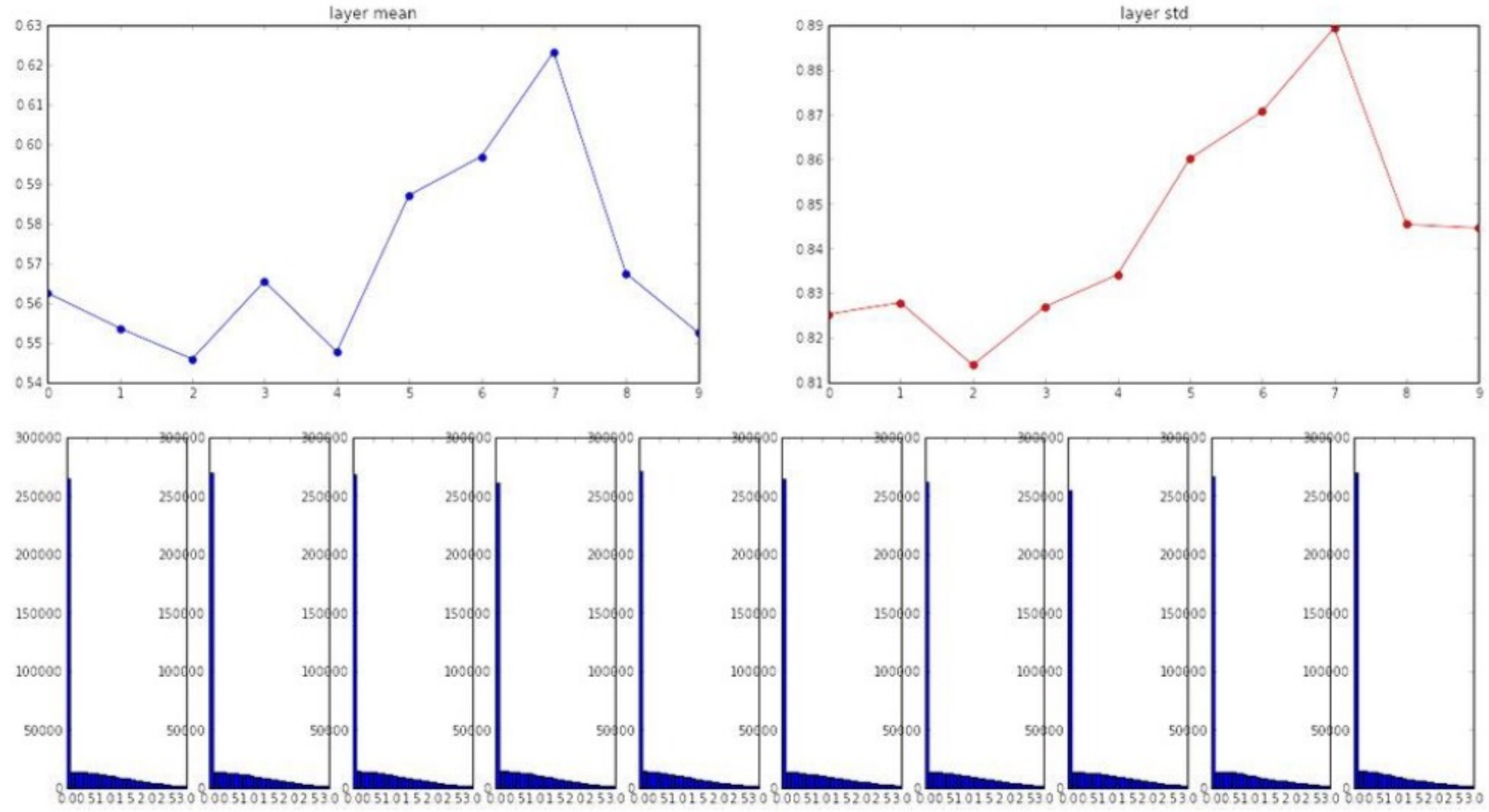
We try He initialization instead!



```
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
```

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}$$

K. He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification* https://arxiv.org/pdf/1502.01852.pdf



A ReLU is zero for half of its input, so you need to double the size of weight variance to keep the signal's variance constant...

F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

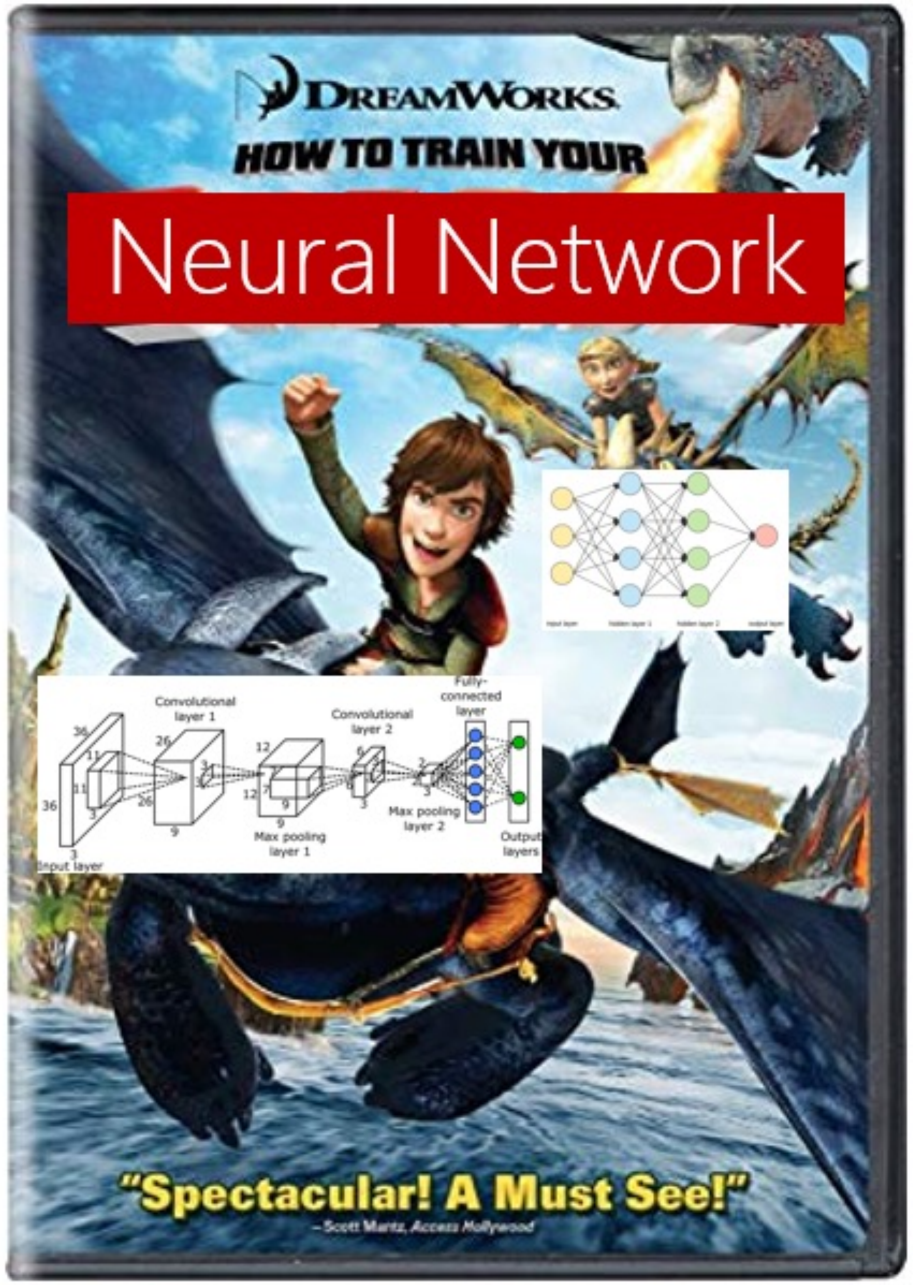
Initialization

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
- *Fixup Initialization: Residual Learning Without Normalization*, Zhang et al, 2019
- *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*, Frankle and Carbin, 2019

Keep it simple: start with ReLU and He initialization!

1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization



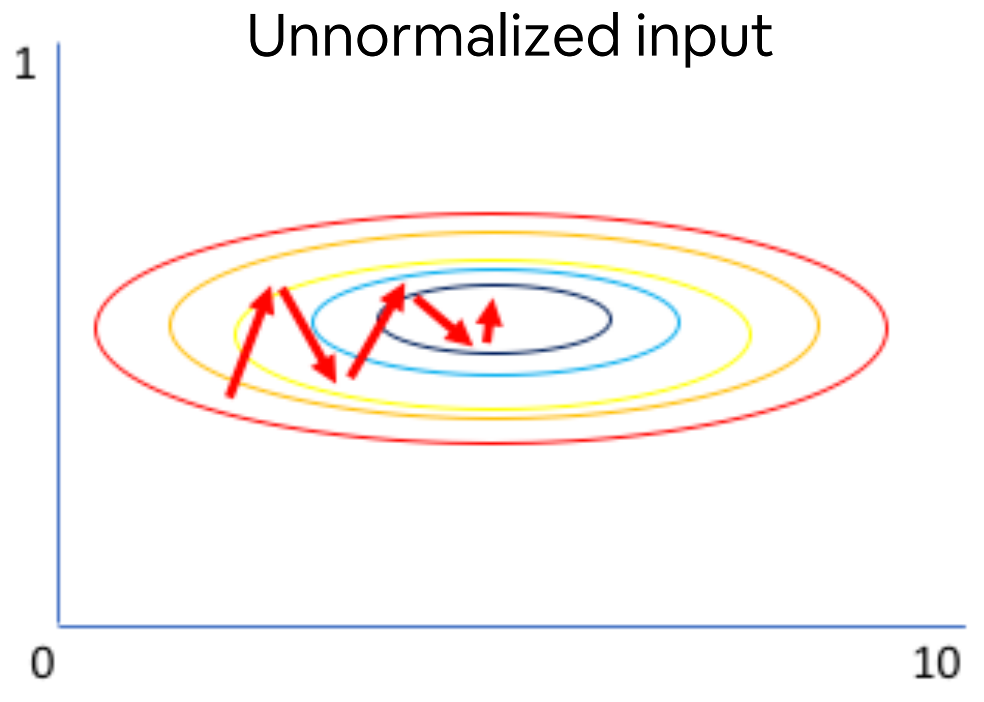
Data Normalization

One of the most important task in Machine Learning... but a forgotten one

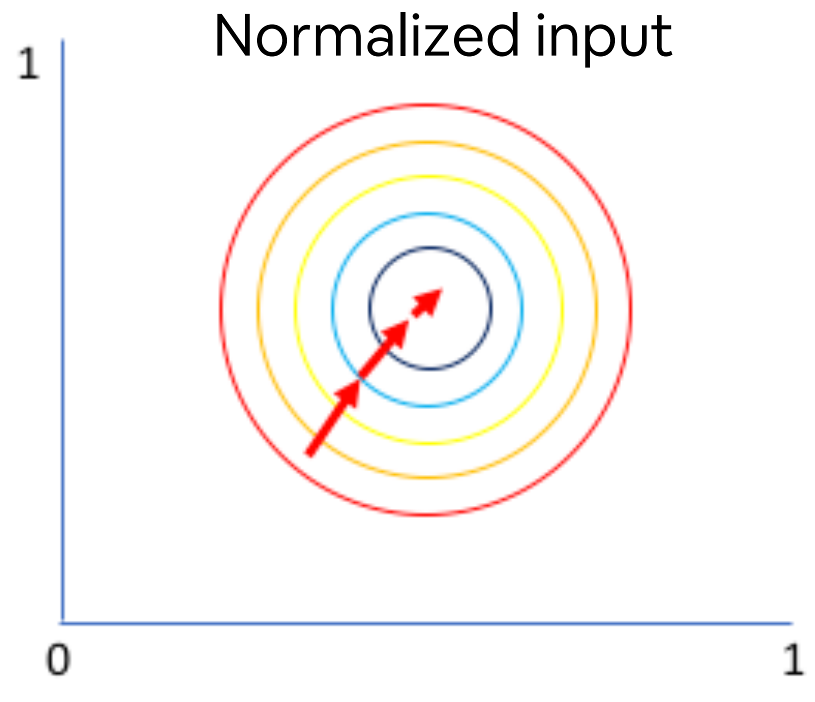
I think I forgot something.
If you forgot, then it wasn't important.
sankruth
Yeah, you're right.
data normalization
imgflip.com

Batch Normalization

Consider a simple neural network with two inputs. The first input value, x_1, varies from 0 to 1 while the second input value, x_2, varies from 0 to 0.01.
Since the network is tasked with learning how to combine these inputs through a series of linear combinations and nonlinear activations, the parameters associated with each input will also exist on different scales.



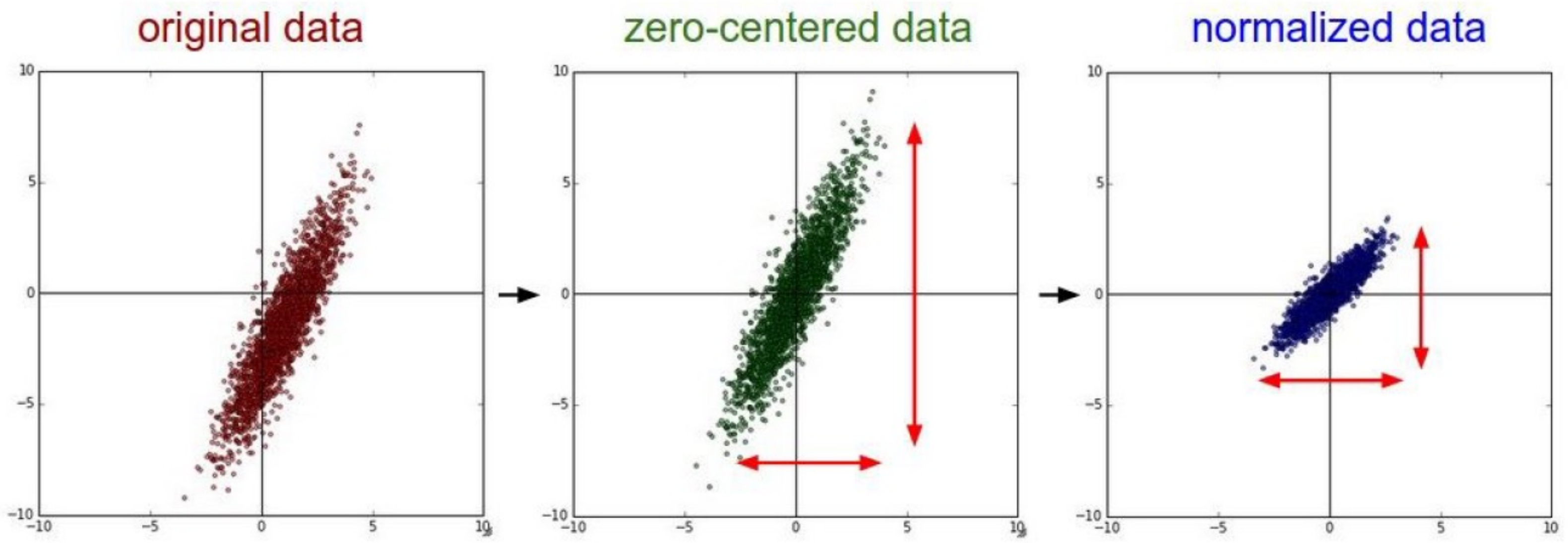
Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

J. Jordan https://www.jeremyjordan.me/batch-normalization/

Batch Normalization

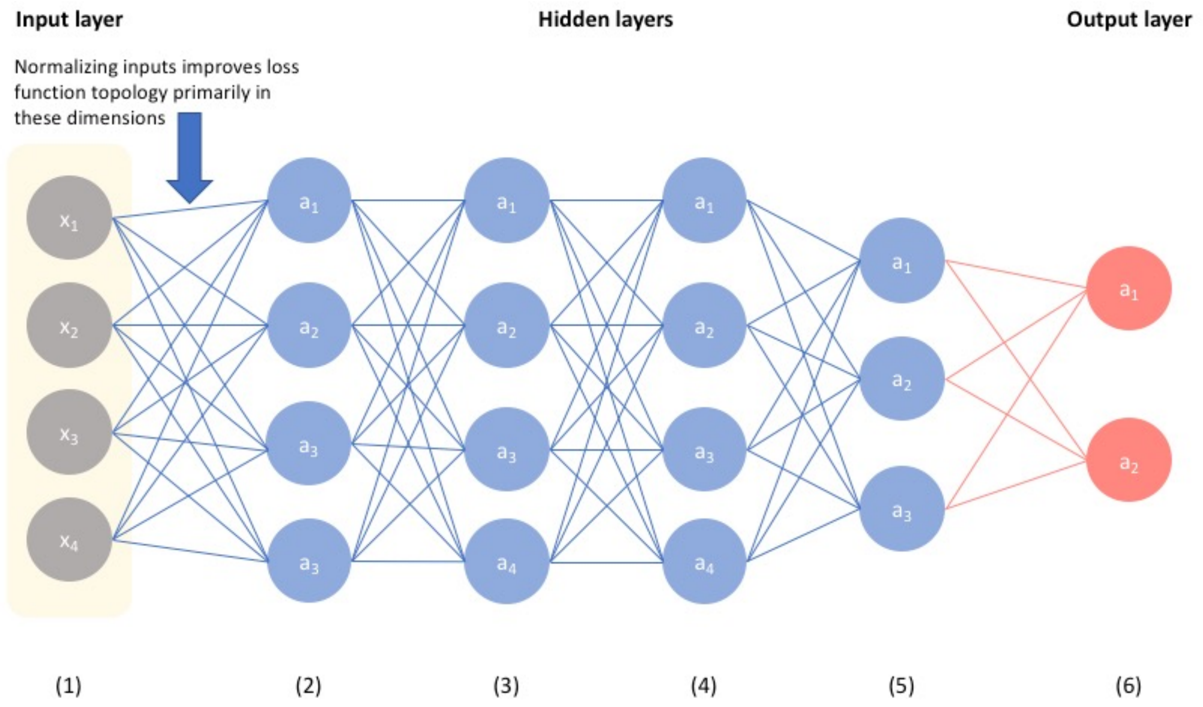


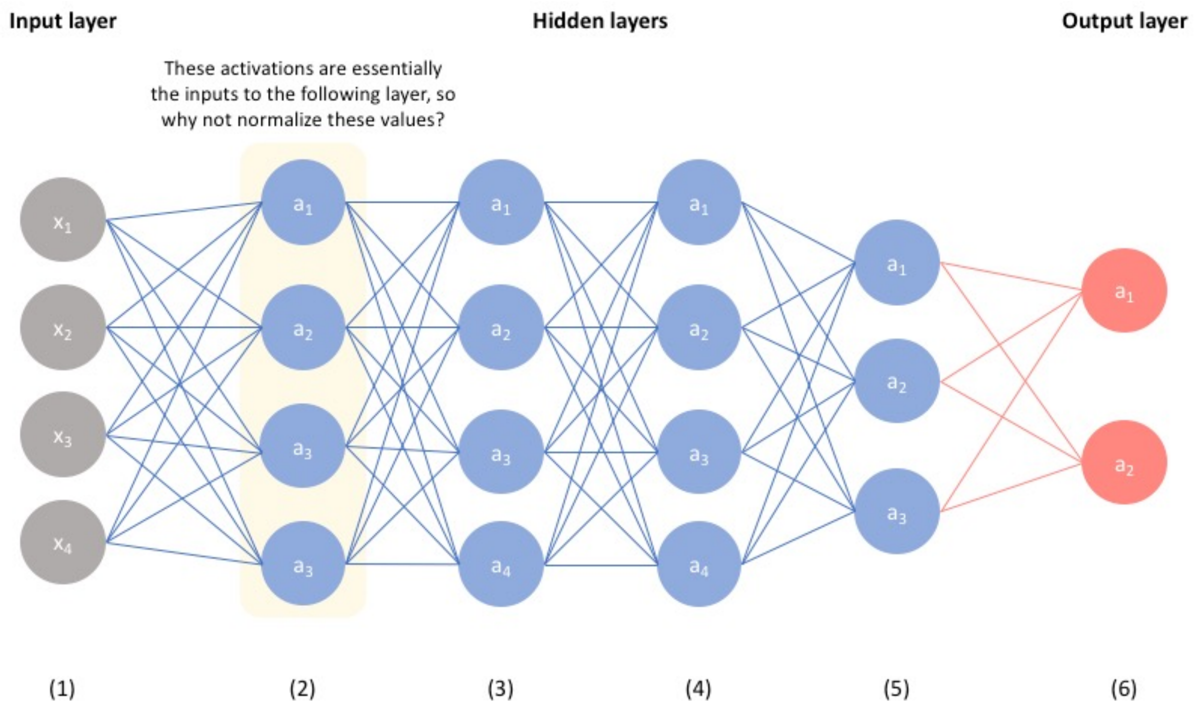
F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

1. Preprocessing: just helps the first layer

2. This is why we prefer zero-mean activations functions

Batch Normalization

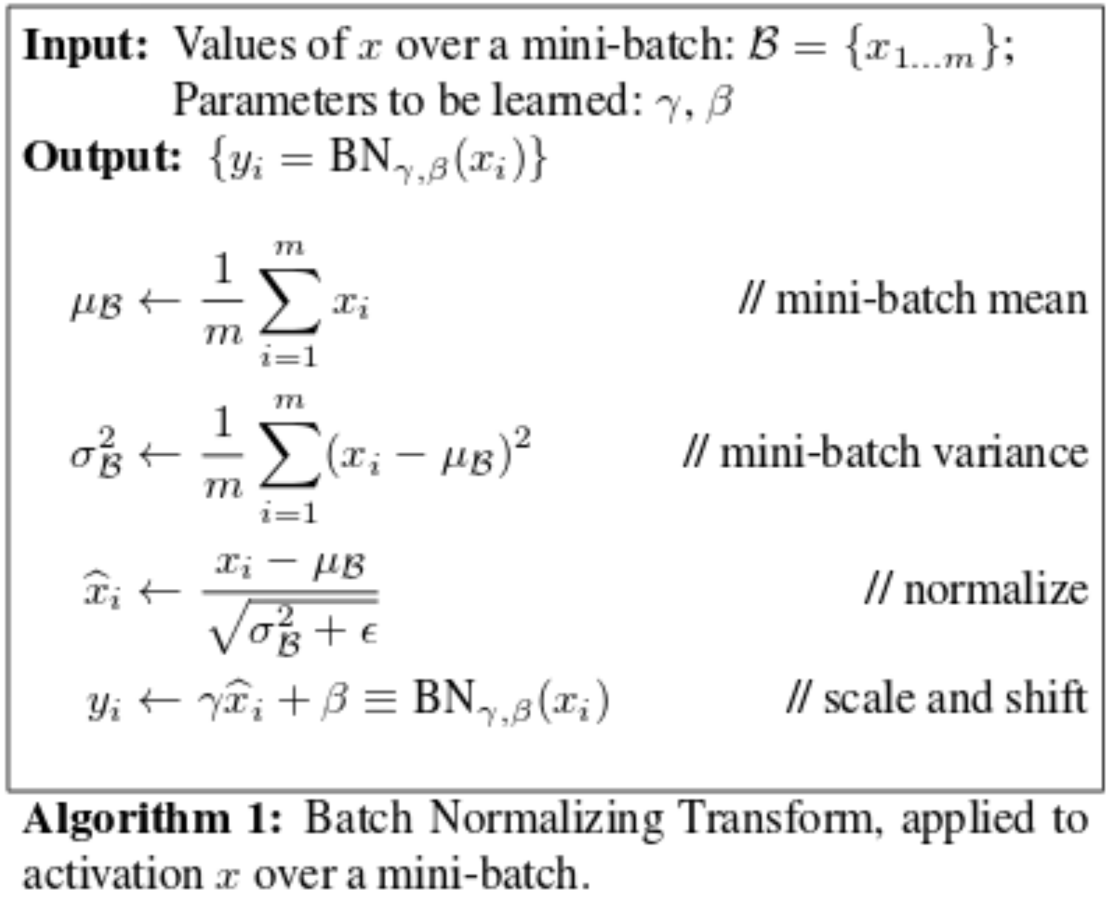




Input layer
Hidden layers
Output layer
These activations are essentially the inputs to the following layer, so why not normalize these values?
x_1
x_2
x_3
x_4
a_1
a_2
a_3
a_4
(1)
(2)
(3)
(4)
(5)
(6)

F. Doukkali https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

Batch Normalization



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

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

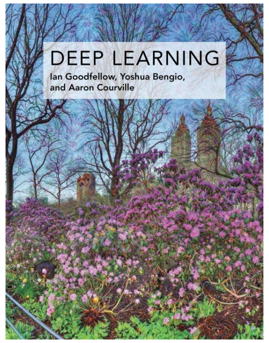
```
$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$    // mini-batch mean
$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$    // mini-batch variance
$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$    // normalize
$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$    // scale and shift
```

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

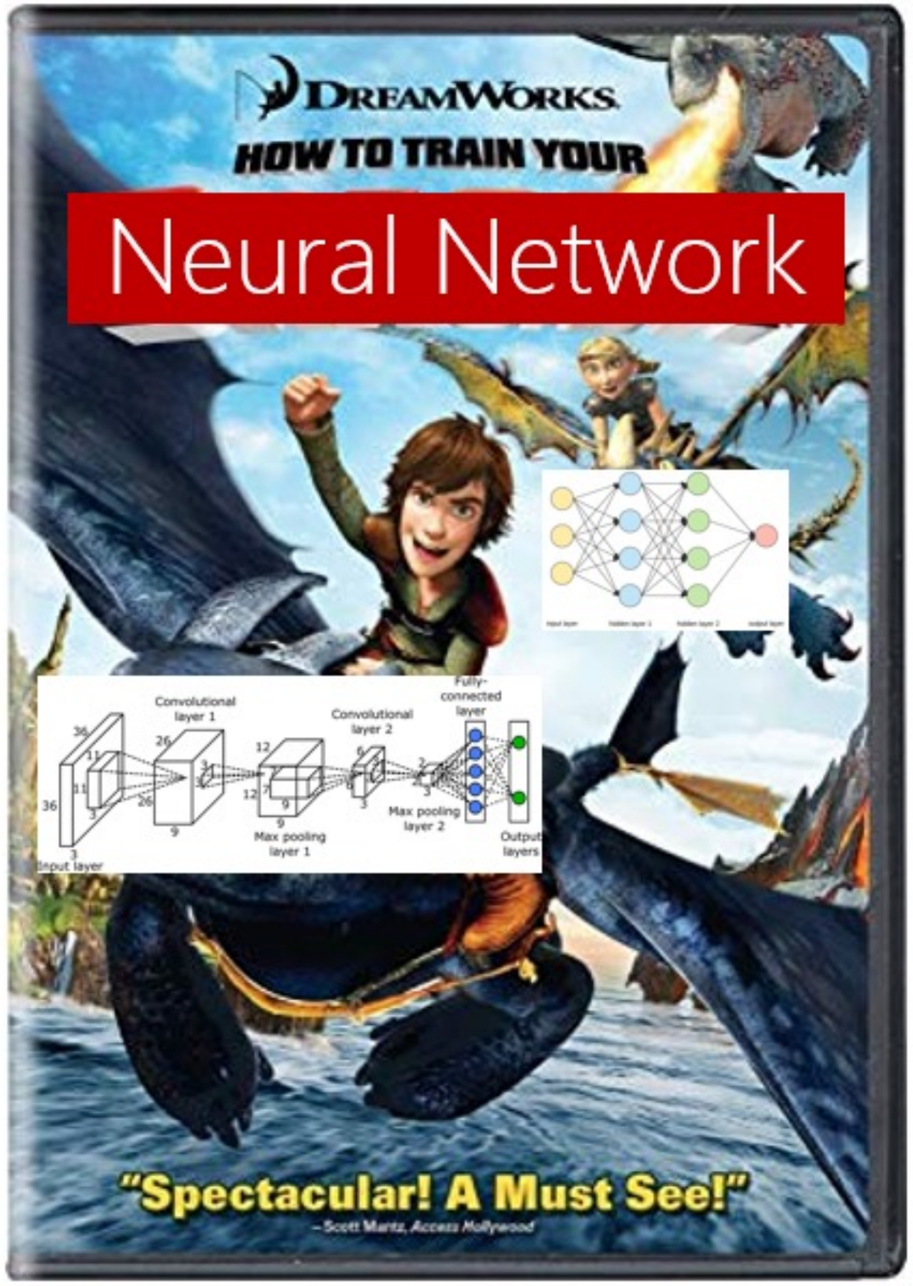
- μ_b and $\sigma_B{}^2$ are batch-specific;
- γ and β are learnt by the BN layer (shared across batches)
- Reduces the strong dependence on initialization
- Allows higher learning rates

S. Ioffe, C. Szegedy *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* https://arxiv.org/abs/1502.03167

1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization

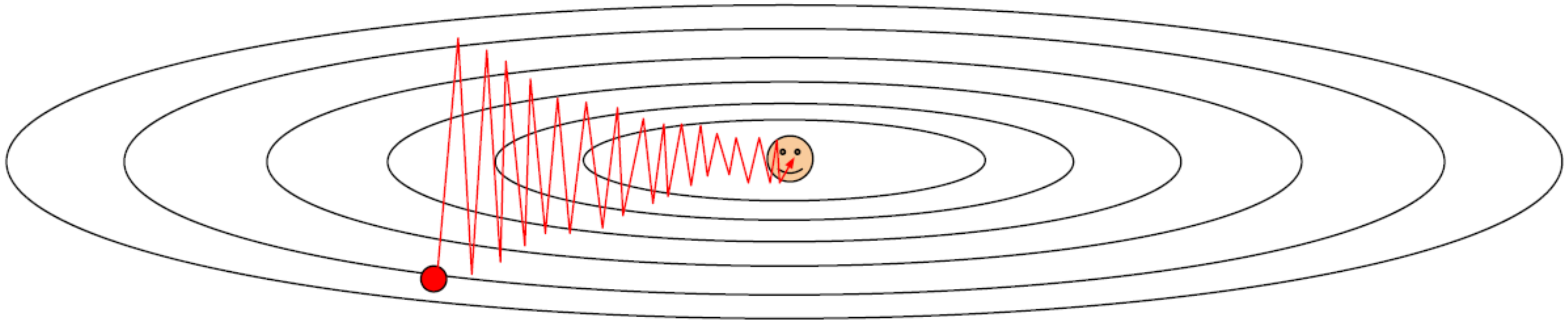


Chapter 8 Optimization for Training Deep Models



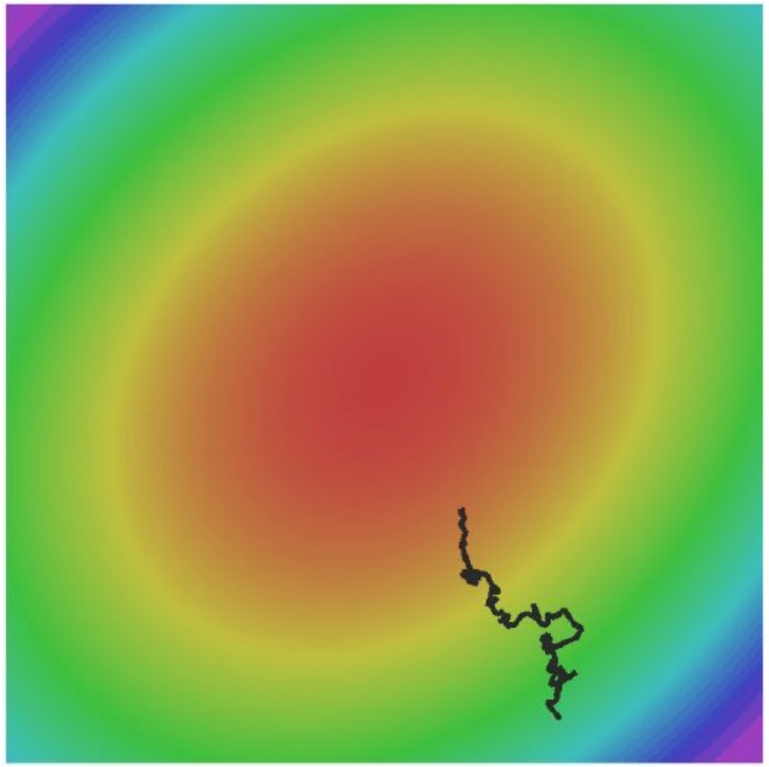
Optimization with SGD

If the loss changes quickly from one direction to another we get very slow progresses along shallow dimensions and jitters along steep directions



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization with SGD



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization with SGD: Momentum

velocity

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

SGD+Momentum

$$\boxed{v_{t+1}} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

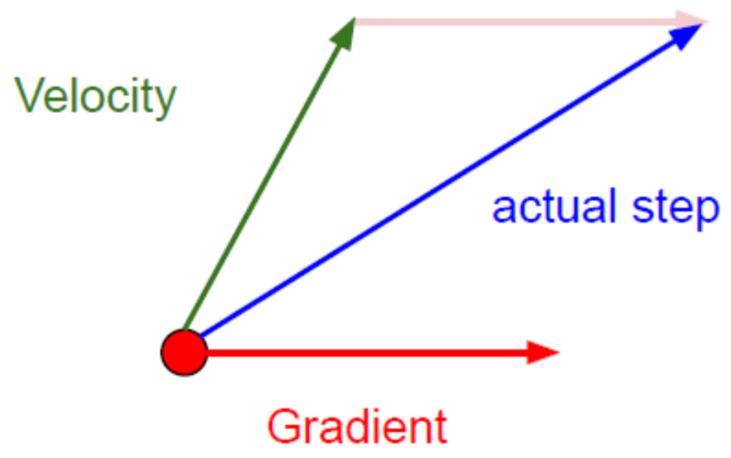
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

- Build up "velocity" as a running mean of gradients
- Rho gives "friction"; typically rho=0.9 or 0.99

F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/
Sutskever et al, *On the importance of initialization and momentum in deep learning*, ICML 2013

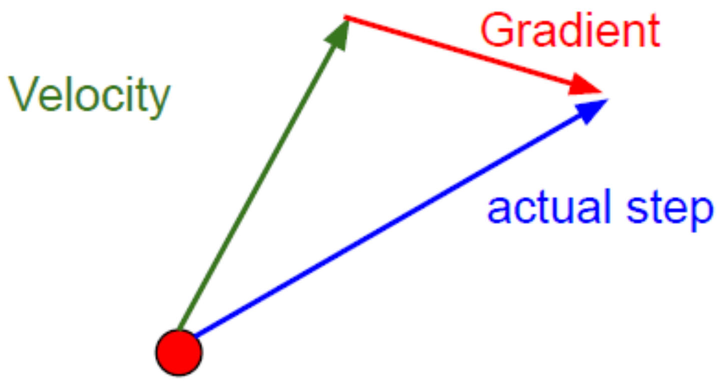
Optimization with SGD: Nesterov Momentum (Optional)

Momentum update:



Combine gradient at current point with velocity to get step used to update weights

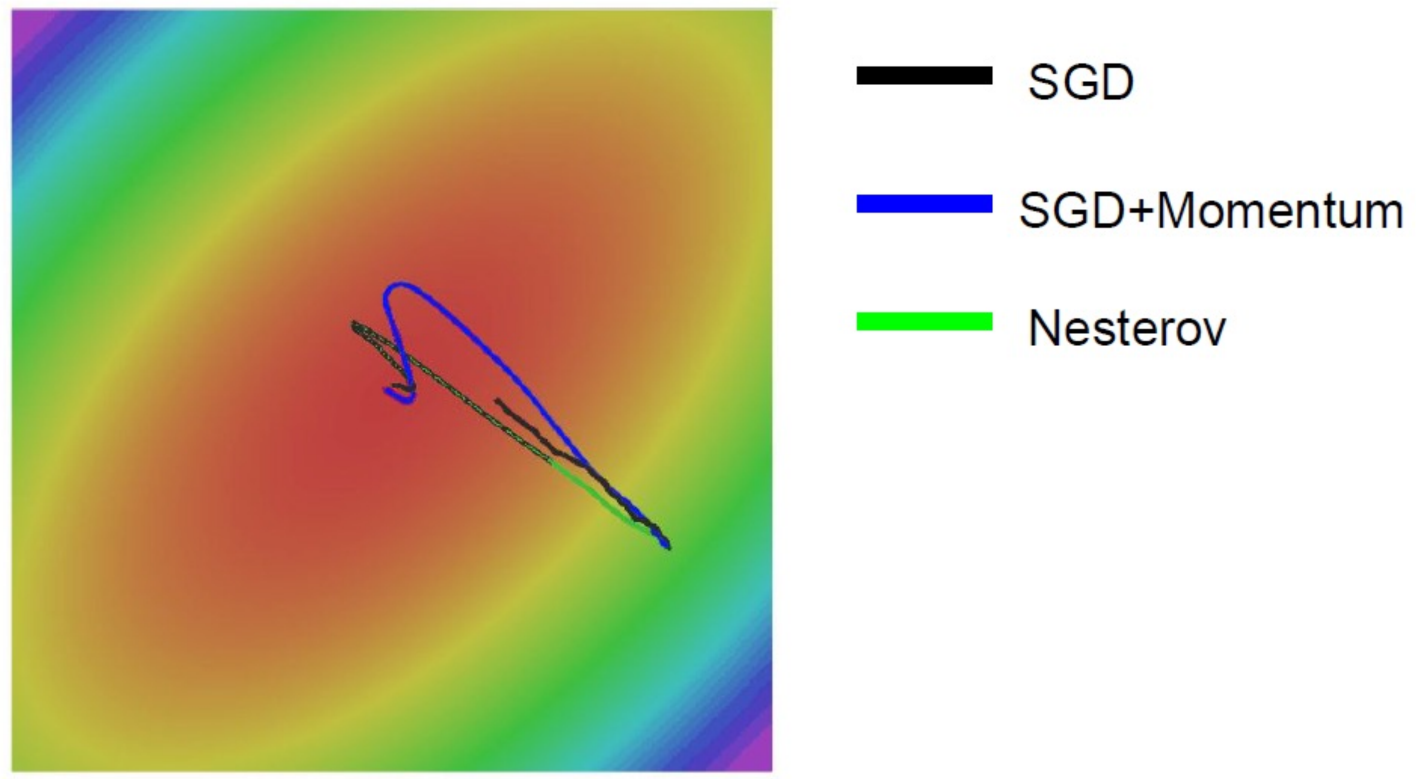
Nesterov Momentum



"Look ahead" to the point where updating using velocity would take us; compute gradient there and mix it with velocity to get actual update direction

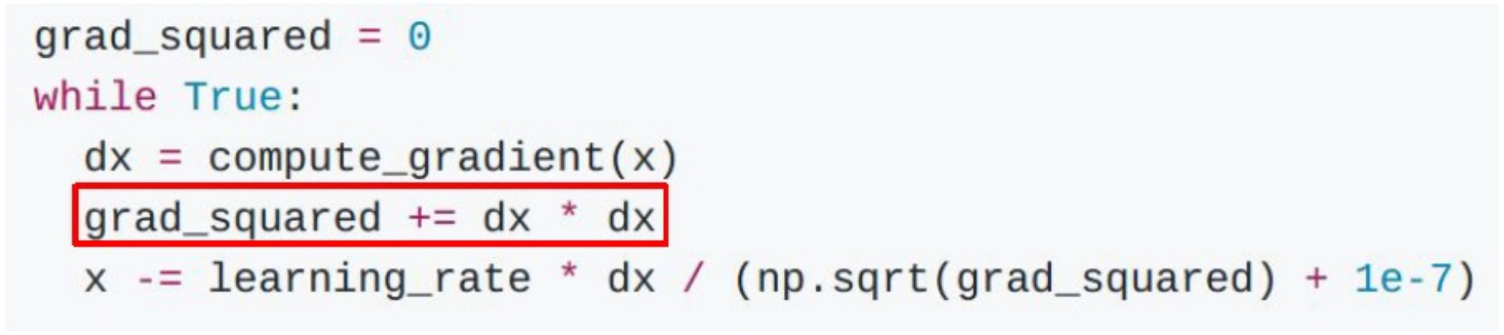
F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/
Nesterov, *A method of solving a convex programming problem with convergence rate O(1/k^2)*, 1983

Optimization with SGD: Nesterov Momentum (Optional)



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization: AdaGrad



```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

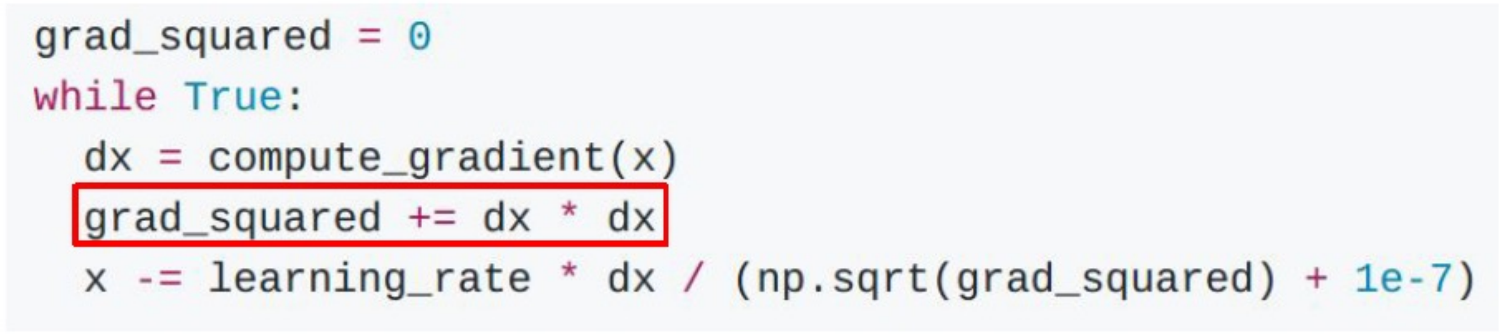
Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

This helps to accelerate movements with coordinates where the gradient is small and decelerate over the coordinates where the gradient is high

Duchi et al, *Adaptive subgradient methods for online learning and stochastic optimization*, JMLR 2011

F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization: AdaGrad



```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

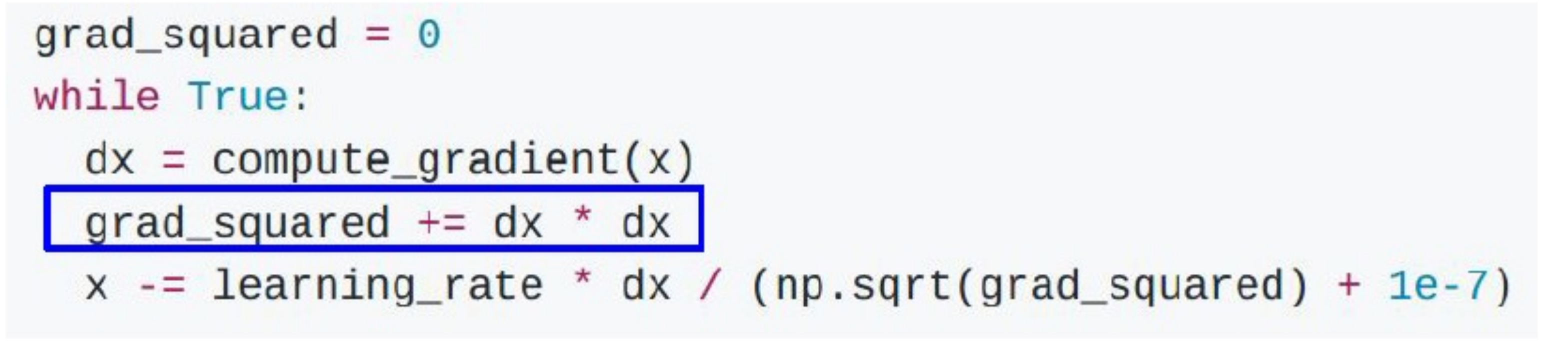
Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

The problem is that over time (during the training) the update gets smaller and smaller

Duchi et al, *Adaptive subgradient methods for online learning and stochastic optimization*, JMLR 2011

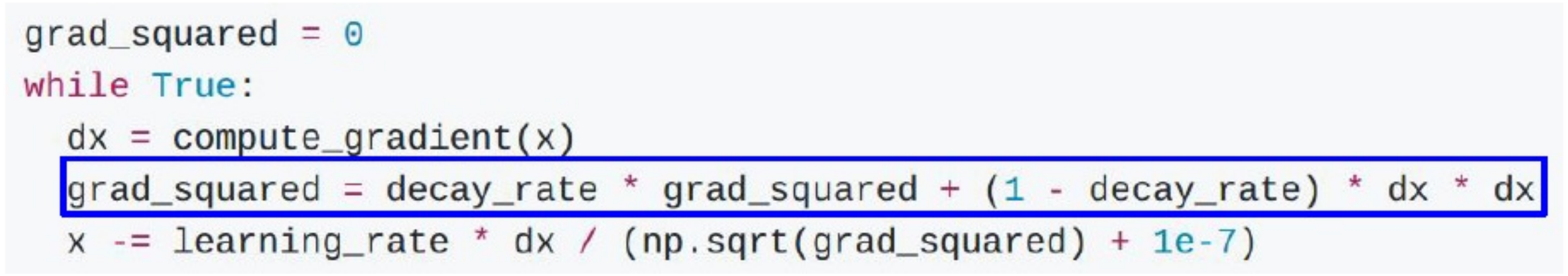
Optimization: AdaGrad & RMSProp

AdaGrad



```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

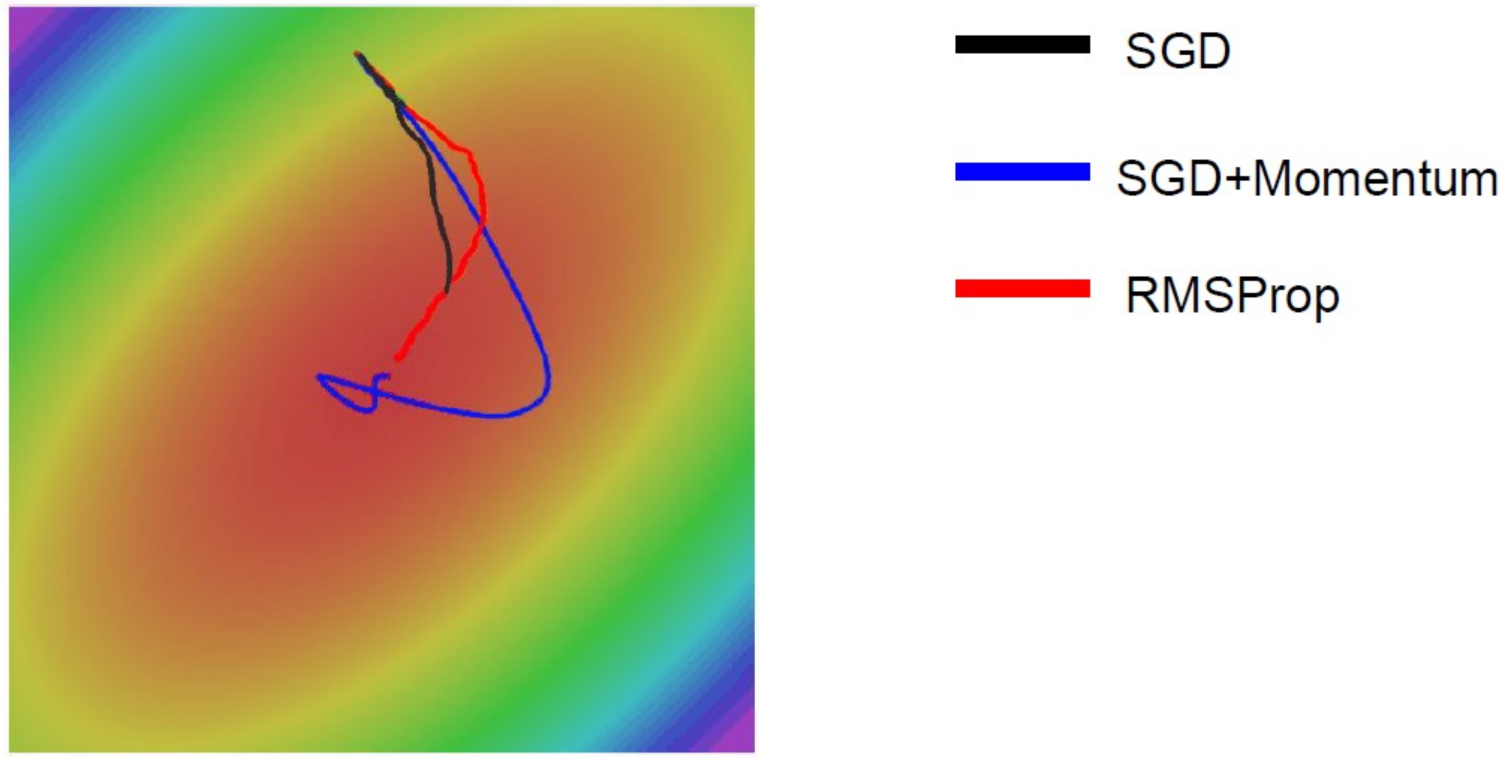


```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Basu, Amitabh, et al. "Convergence guarantees for rmsprop and adam in non-convex optimization and their comparison to nesterov acceleration on autoencoders." arXiv preprint arXiv:1807.06766 (2018).

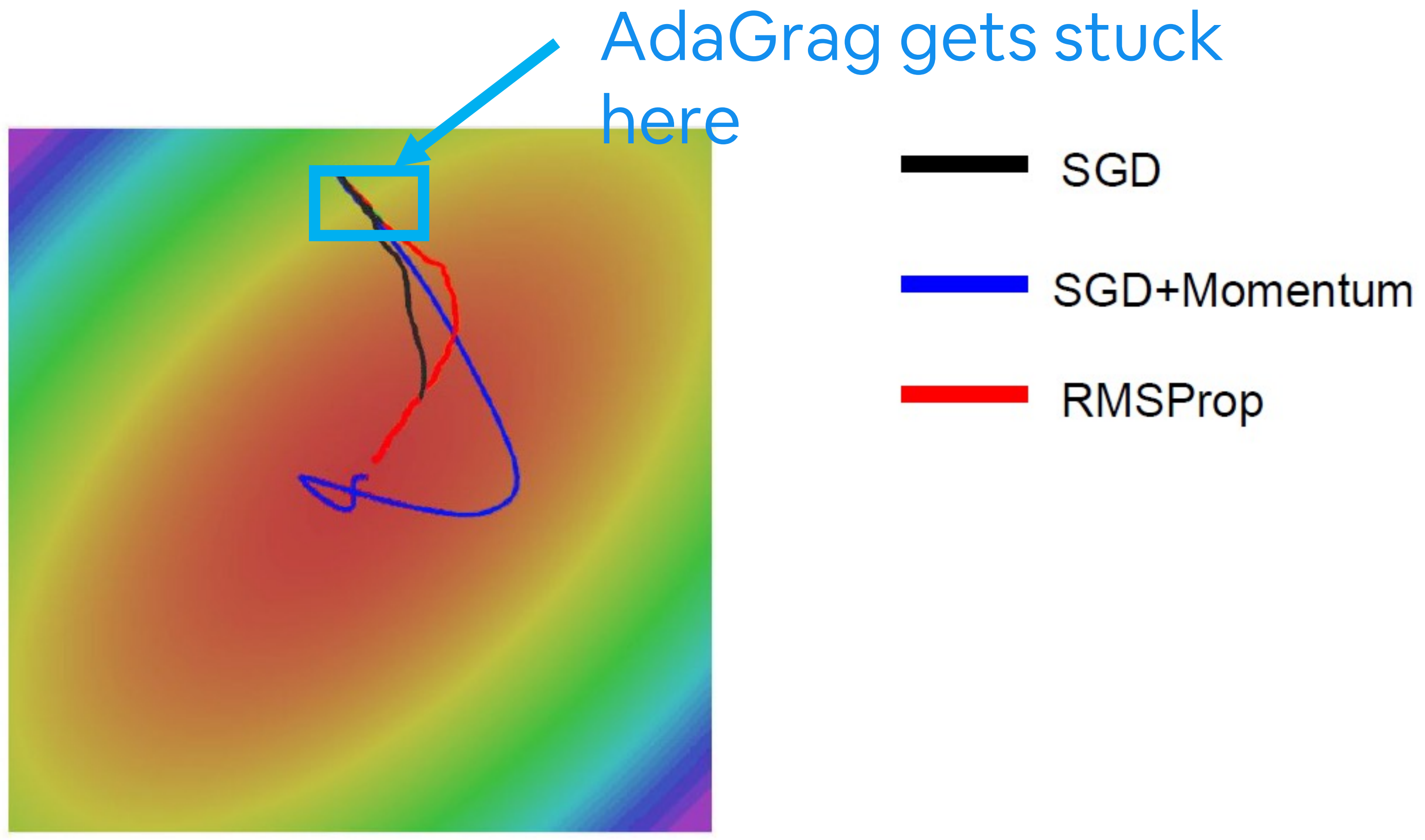
F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization with SGD: RMSProp

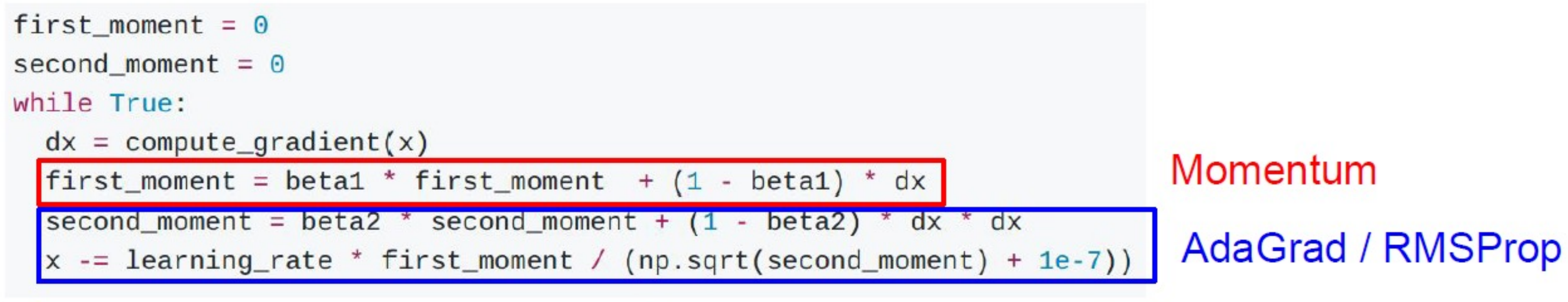


F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization with SGD: RMSProp



Optimization: Adam (almost)

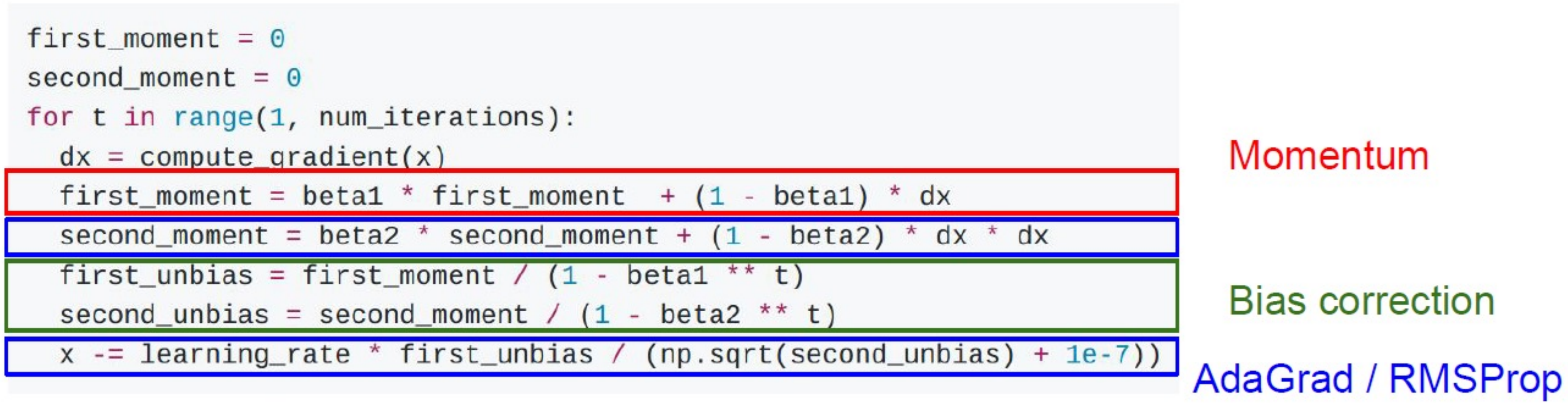


```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Kingma and Ba, *Adam: A method for stochastic optimization*, ICLR 2015

F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization: Adam (real)



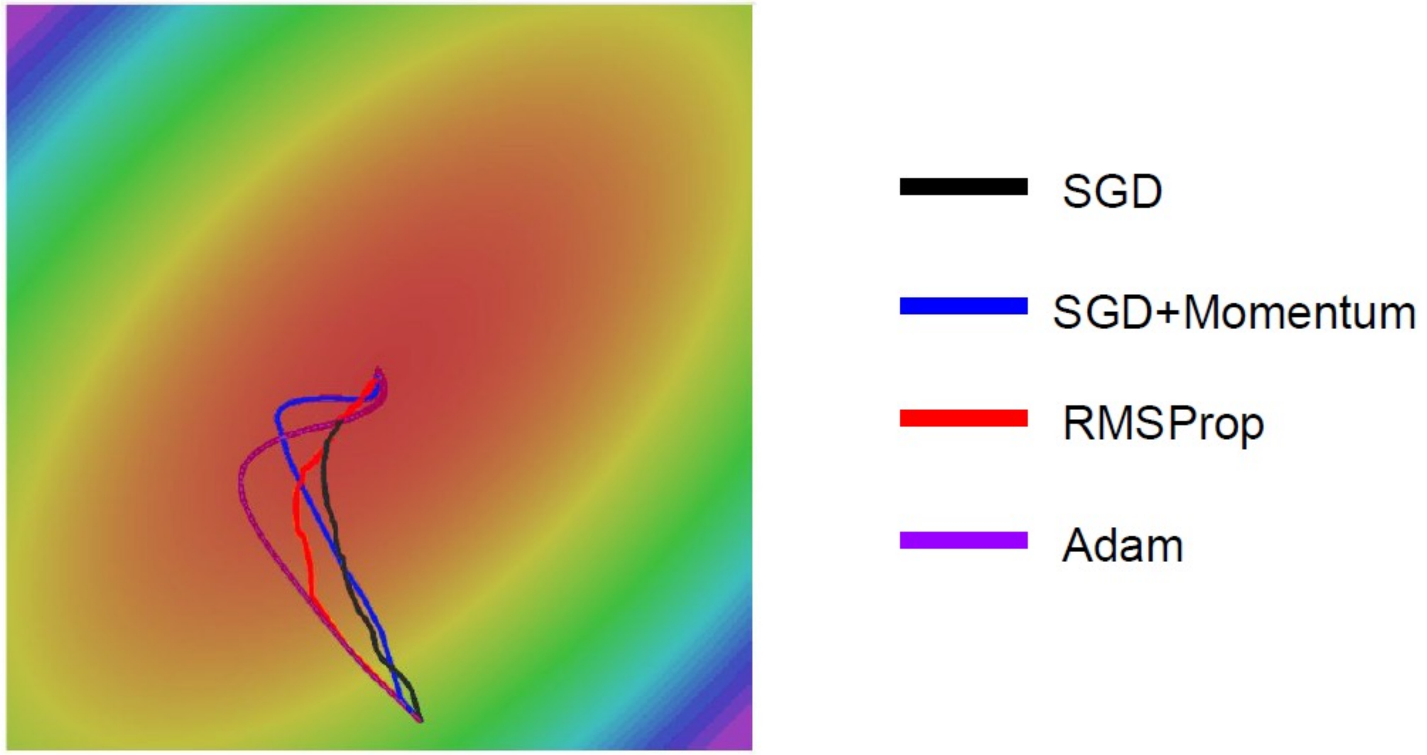
```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

- Kingma and Ba, *Adam: A method for stochastic optimization*, ICLR 2015
- http://ruder.io/optimizing-gradient-descent/index.html#gradientdescentoptimizationalgorithms
- F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

Optimization with SGD: Adam



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

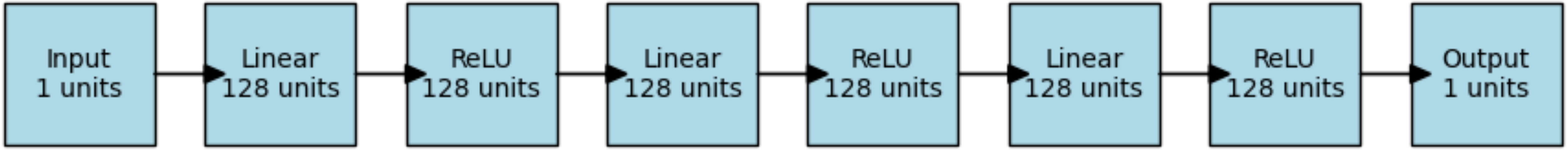
TLDR: Start with ADAM!

Another experiment

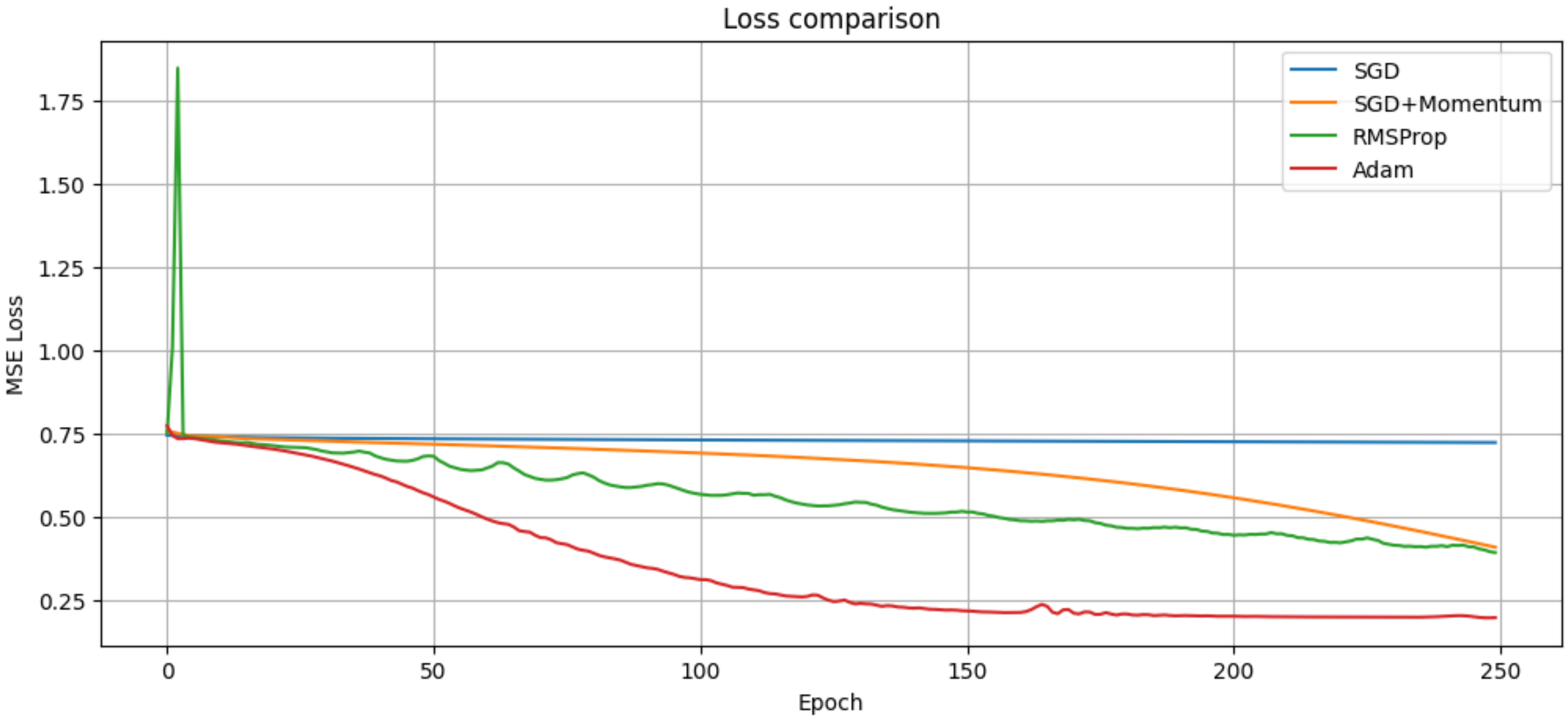
Data generated from:

$$y = \sin(5x) + 0.5\sin(20x) + \textbf{noise}$$

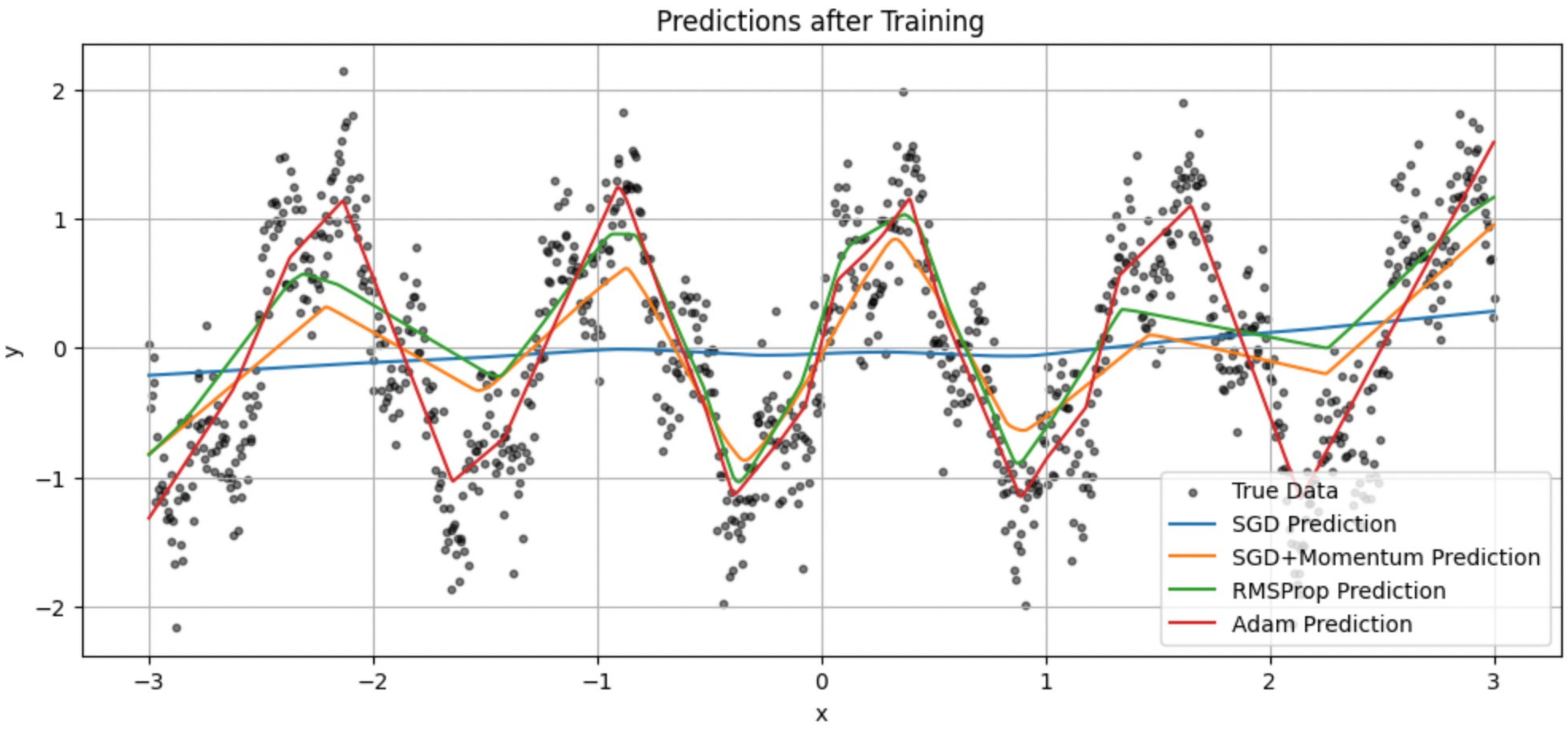
A NN with 3 hidden layers -> 128 units per layer



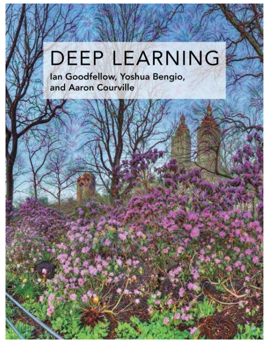
Another experiment



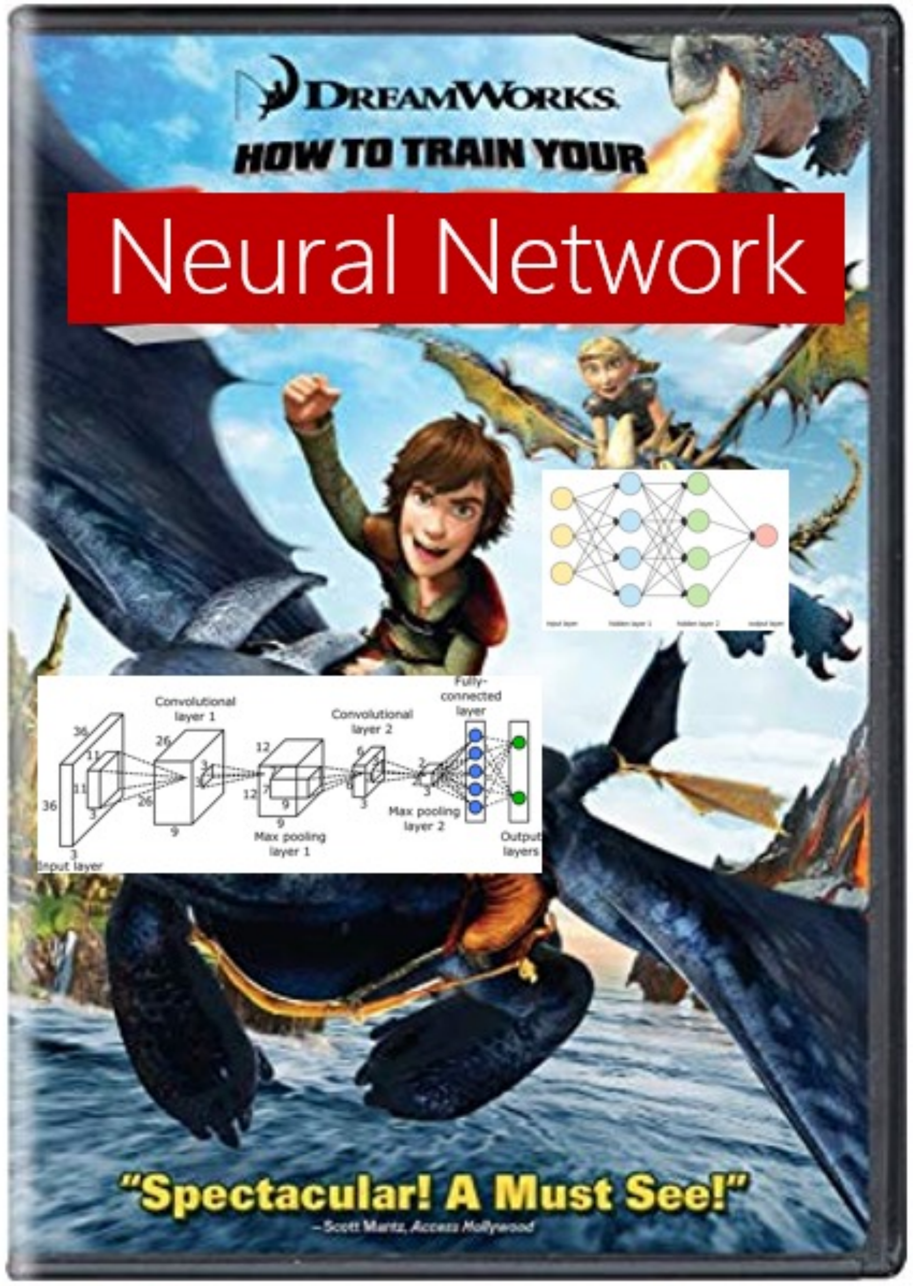
Another experiment



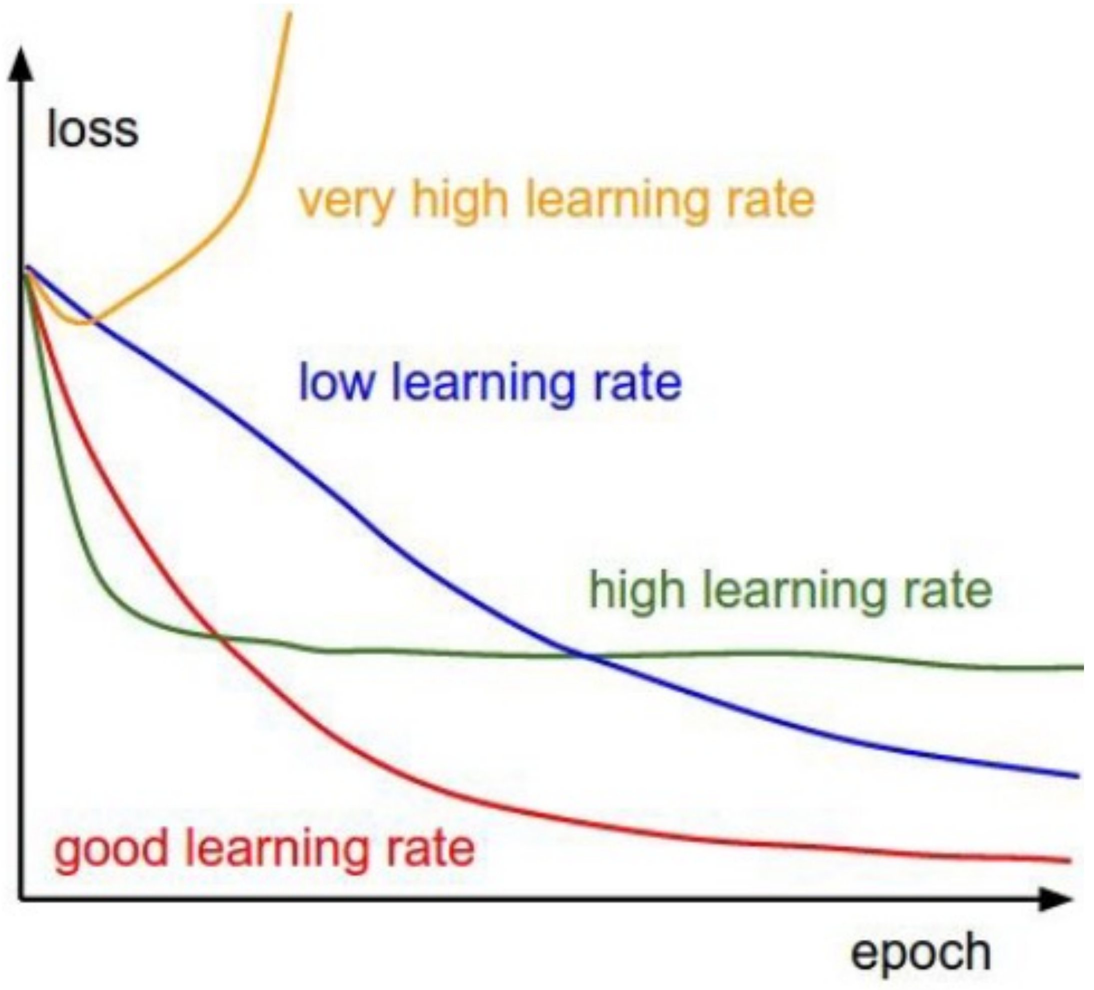
1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization



Chapter 4.3 Gradient-Based Optimization



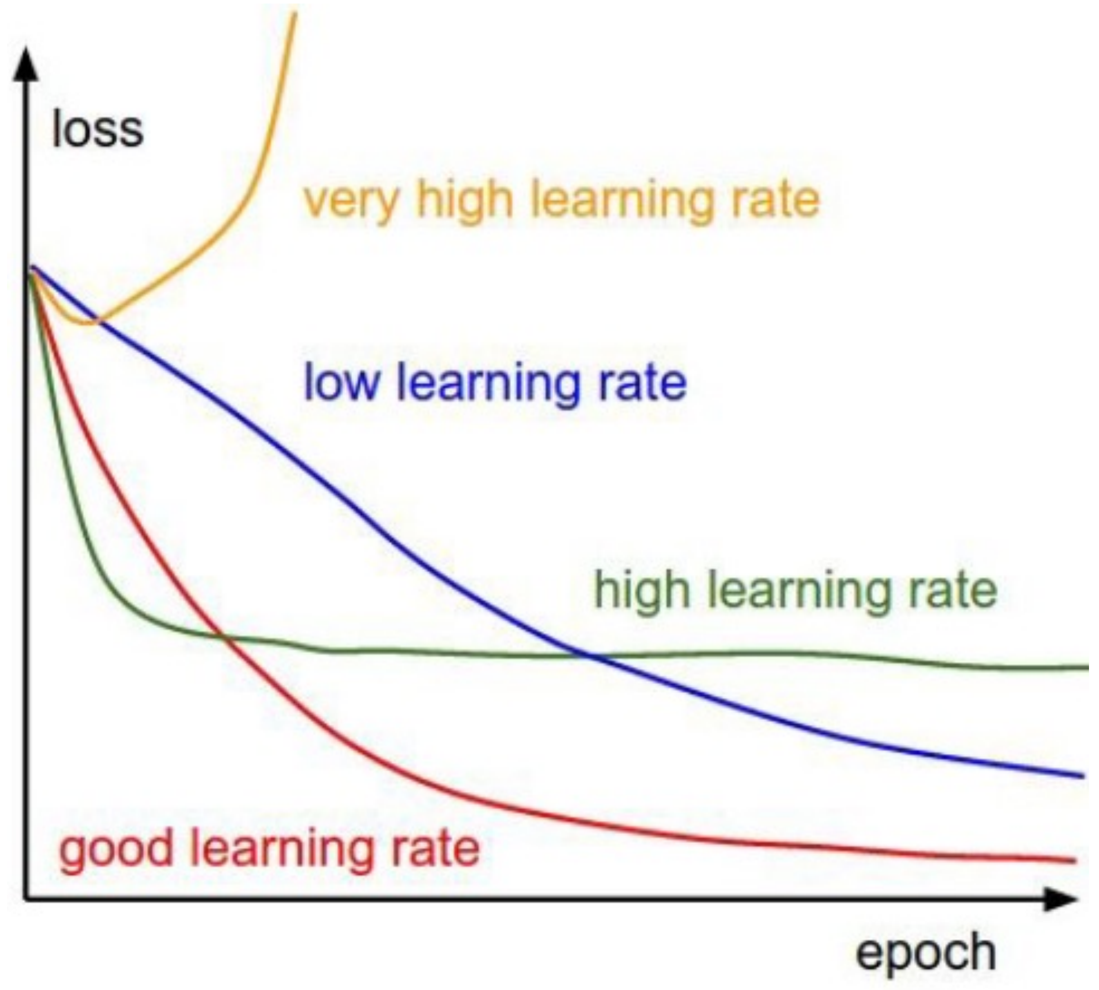
Learning rate



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

- SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have learning rate as a hyperparameter

Learning rate

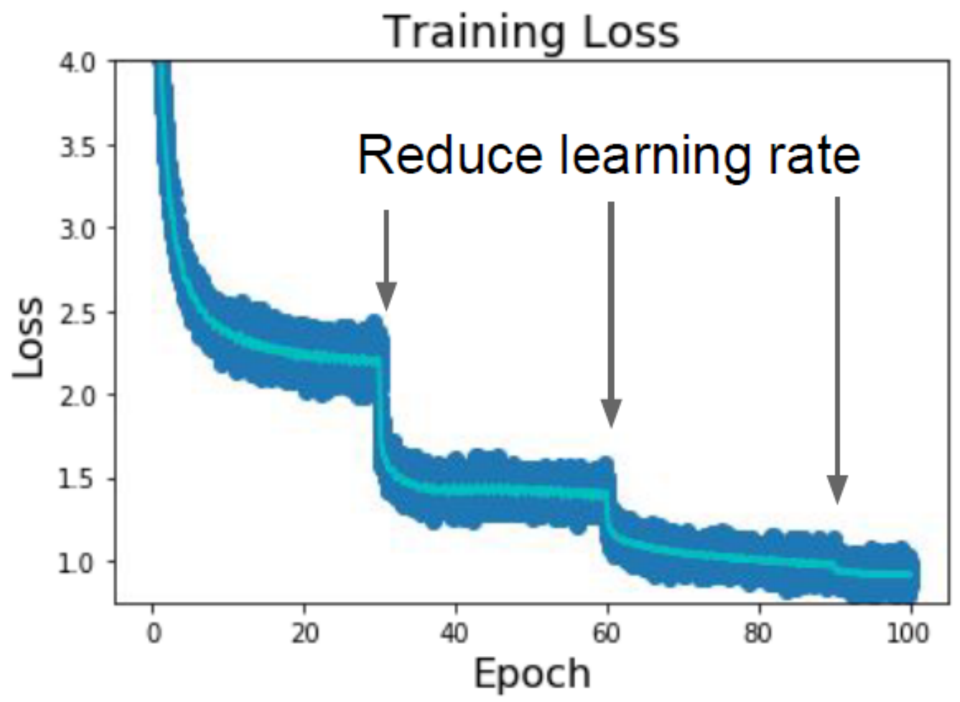


F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

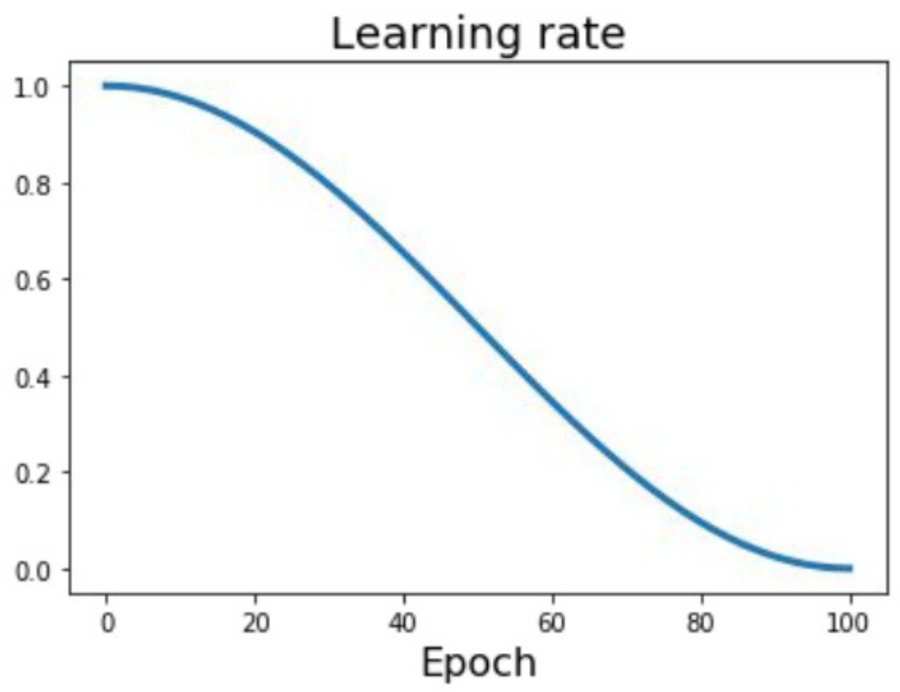
- SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have learning rate as a hyperparameter

- An interesting and common strategy is to employ time-varying learning rate

Learning rate



- Reduce learning rate at a few fixed points.
- E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

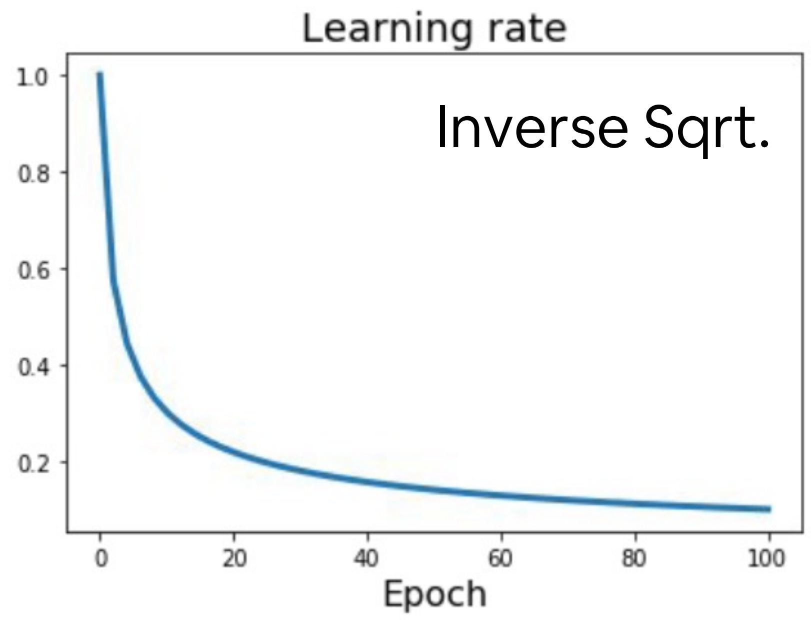


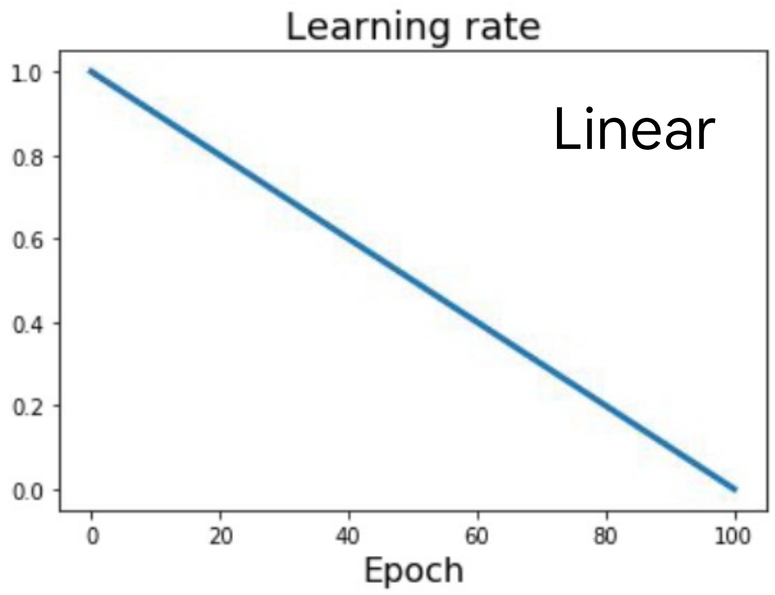
$$\alpha_t = \frac{1}{2}\alpha_0\left(1 + \cos(t\pi/T)\right)$$

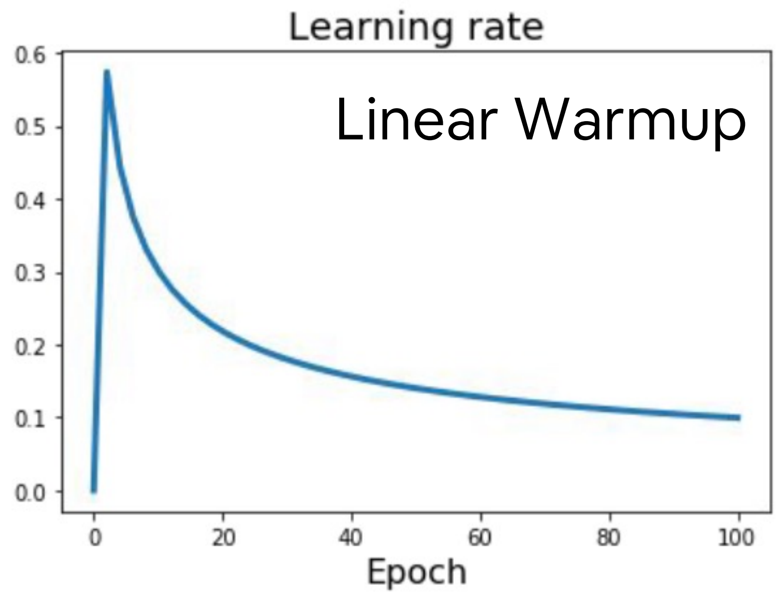
α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

- F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/
- Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017

Learning rate



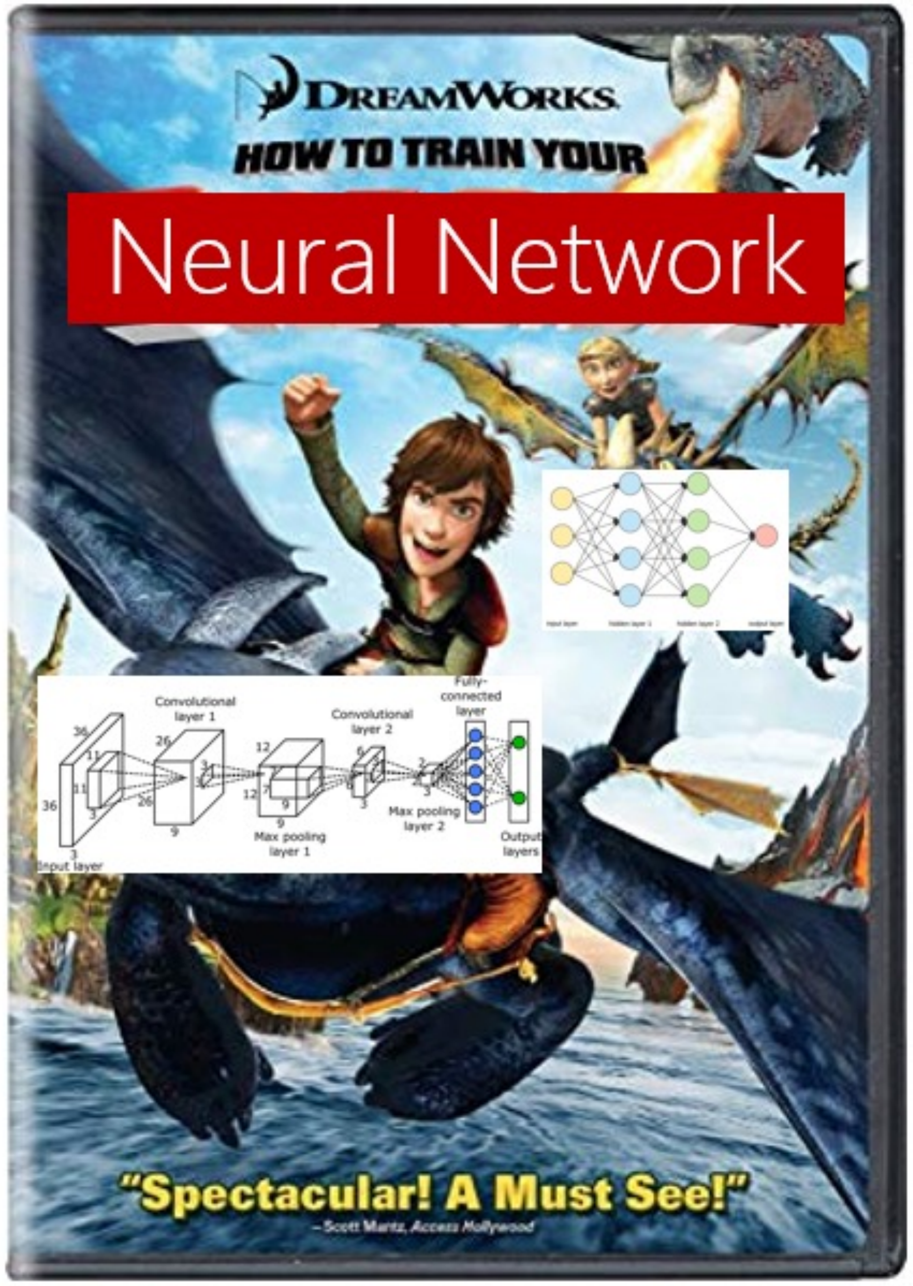


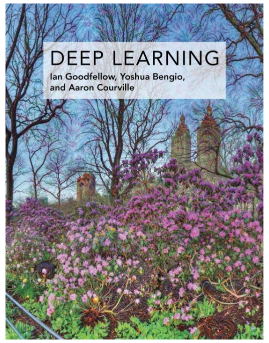


Goyal et al, *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*, arXiv 2017

F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

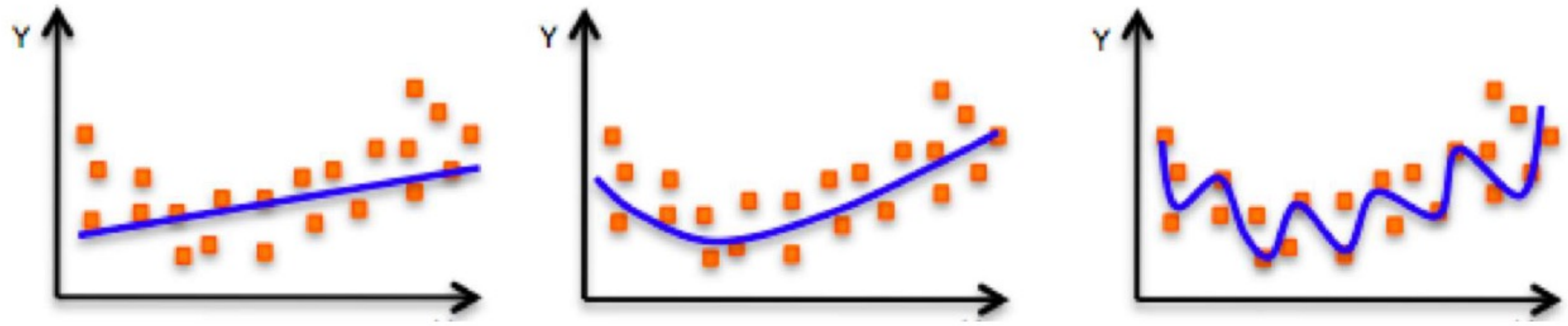
1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization





Chapter 7 Regularization for Deep Learning

The Problem of Overfitting and Regularization



Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

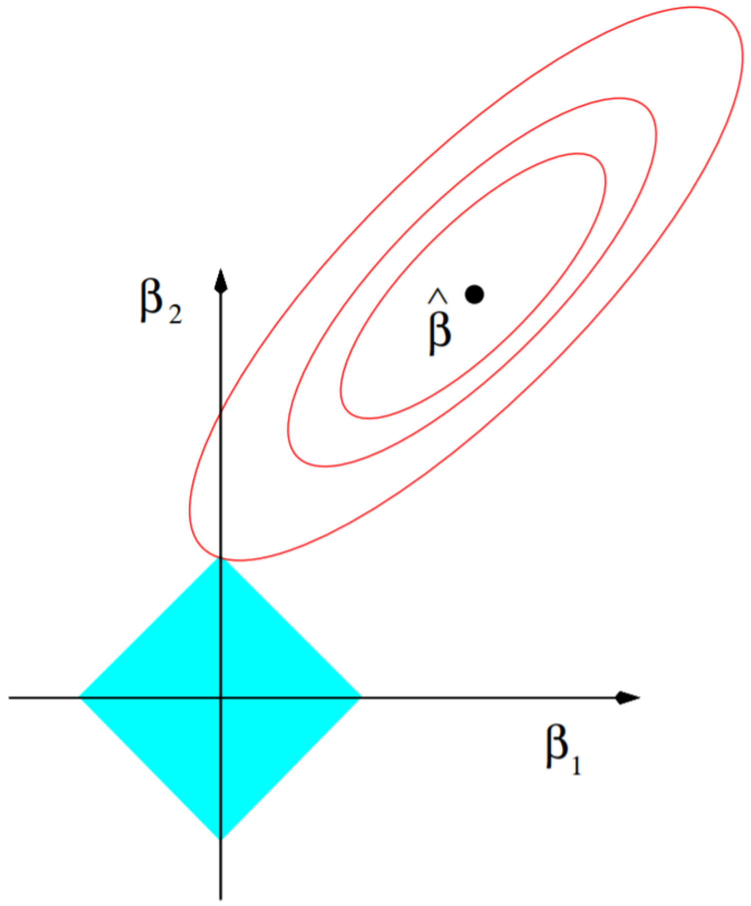
-MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Regularization #0: Lasso

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \underline{\frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2} + \lambda \underline{\sum_{j=1}^{p} |\beta_j|} \right\}$$

Regularization is a paradigm in Machine Learning that allow to have a good trade-off between accuracy on training data and model complexity enabling generalization

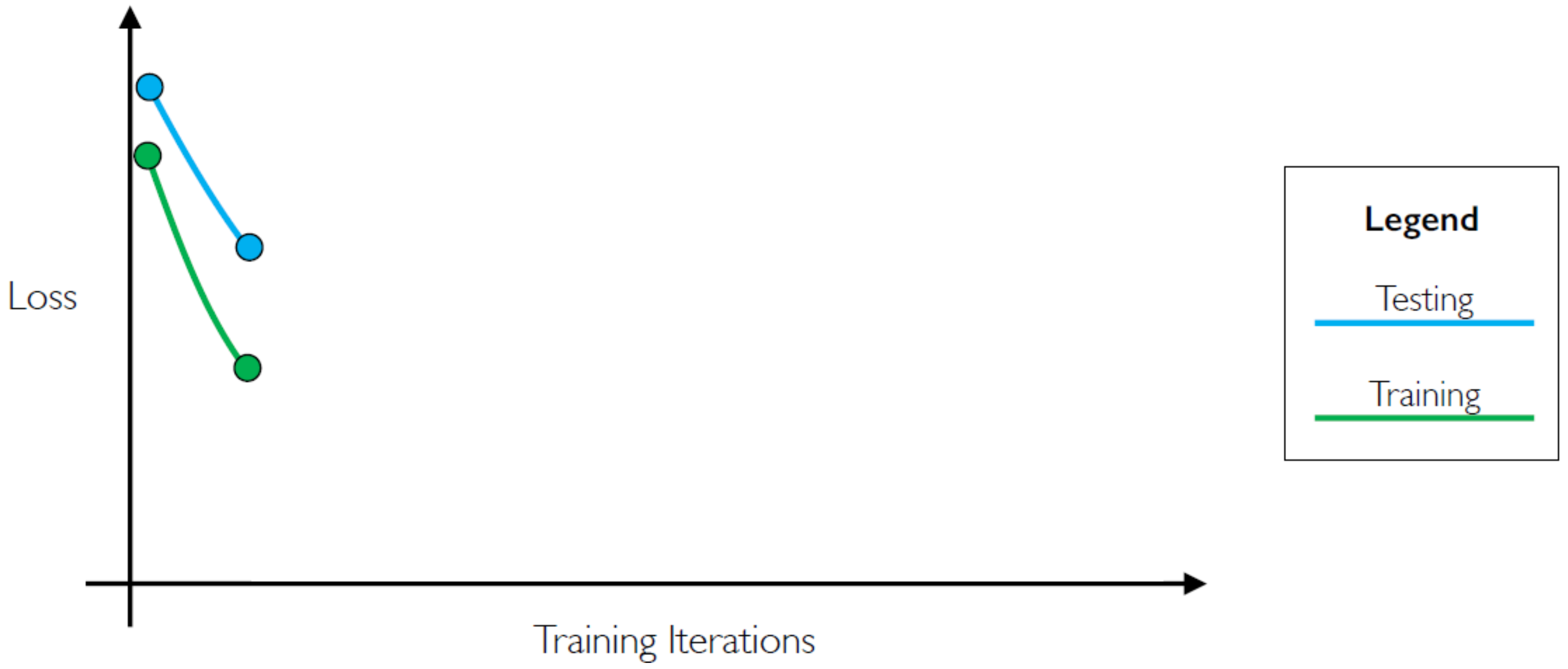
In the case of LASSO this is achieved by a dedicated loss fuλction and by tuning the hyperparameter



-T. Hastie, R. Tibshirani, J. Friedman *The Elements of Statistical Learning* https://web.stanford.edu/~hastie/Papers/ESLII.pdf

Regularization #1: Early stopping

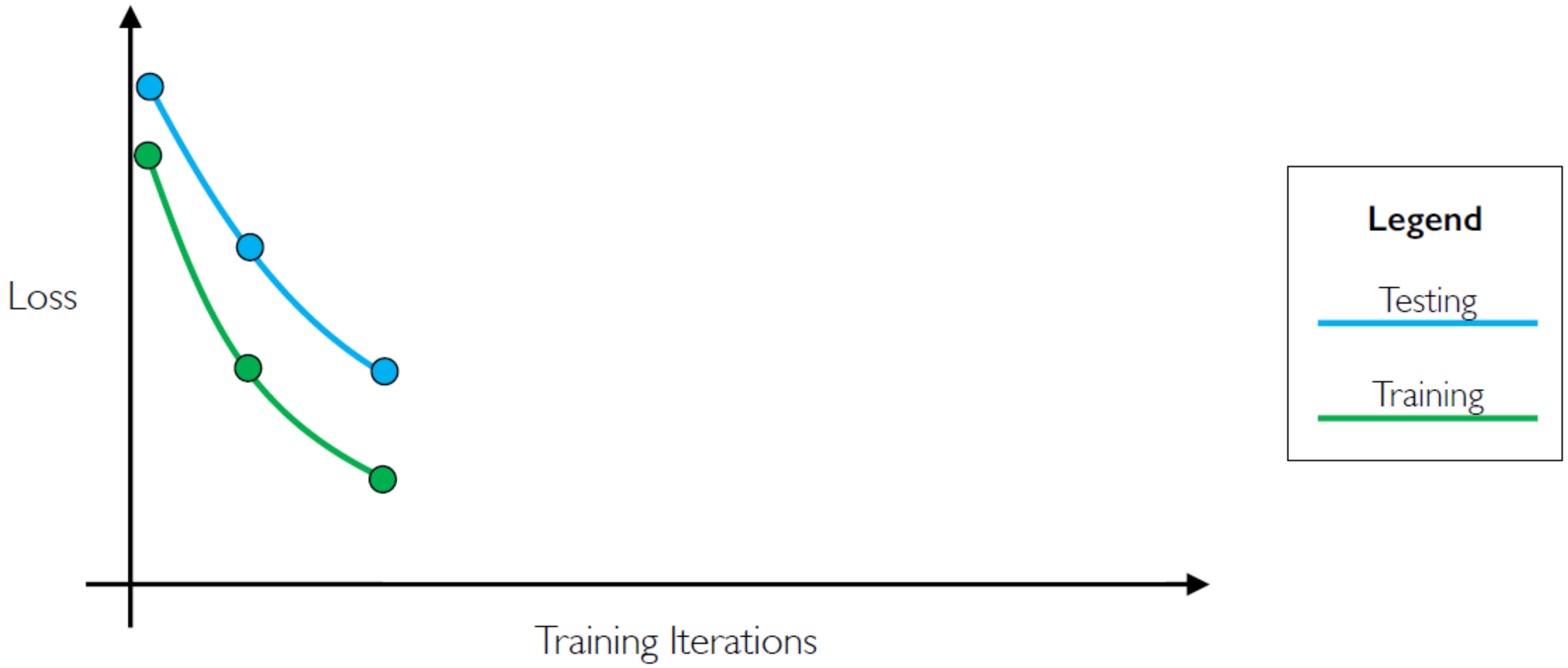
A simple approach is to take a look at a testing dataset and stop even if the loss on the training is still improving



-MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Regularization #1: Early stopping

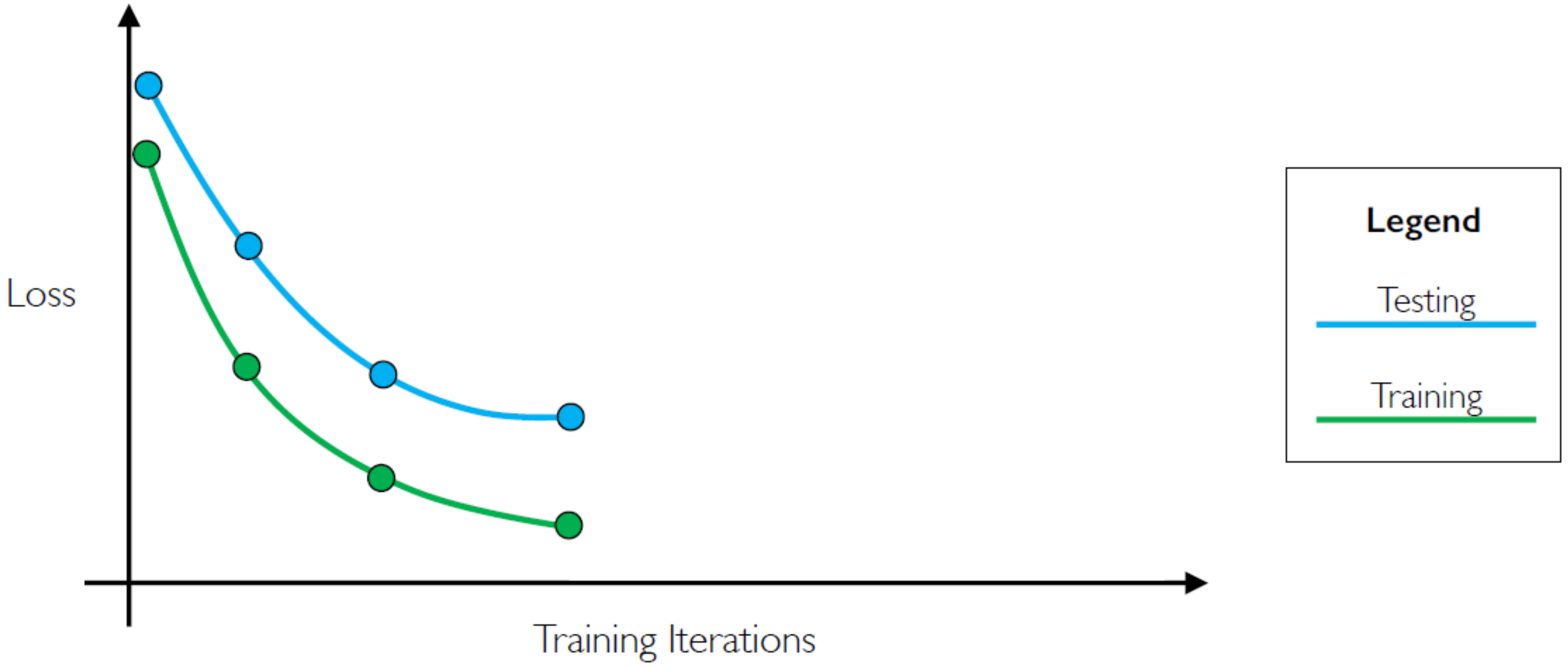
A simple approach is to take a look at a testing dataset and stop even if the loss on the training is still improving



-MIT *Introduction to Deep Learning* http://introtodeeplearning.com

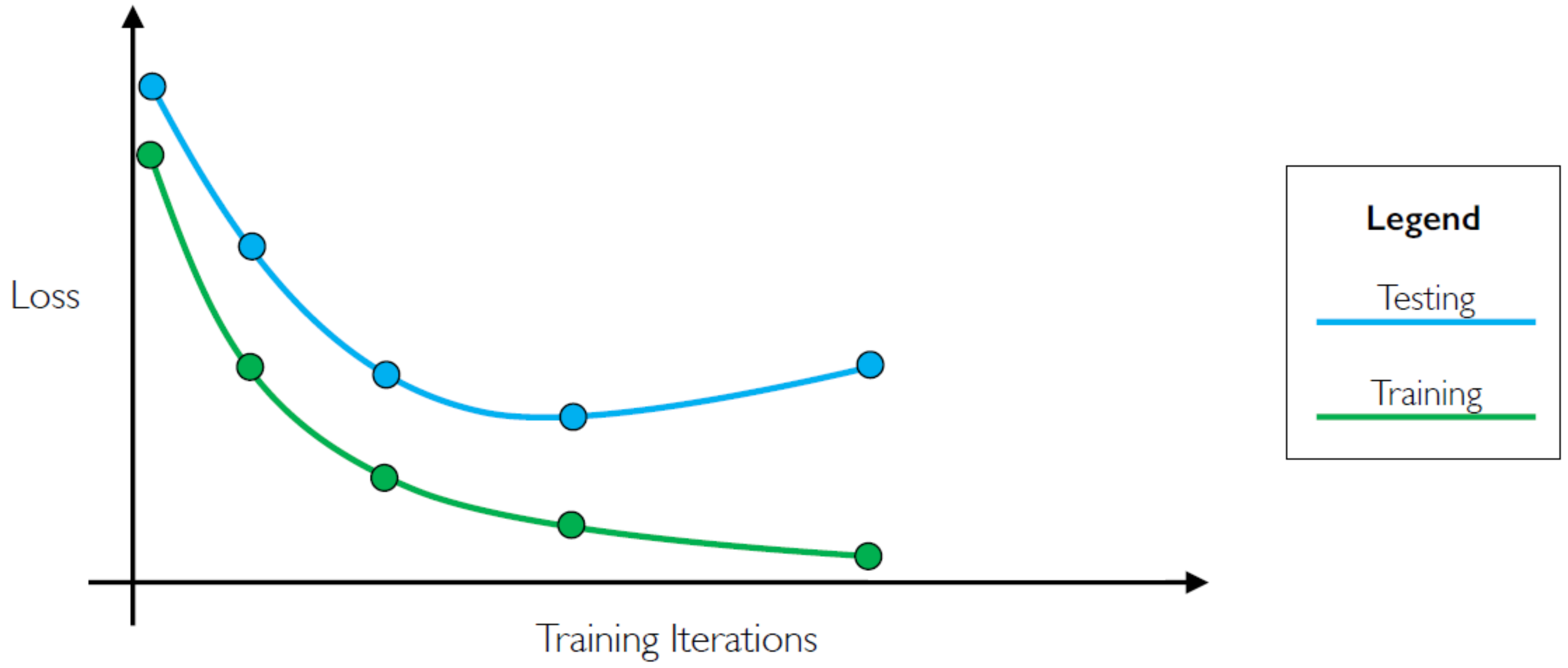
Regularization #1: Early stopping

A simple approach is to take a look at a testing dataset and stop even if the loss on the training is still improving



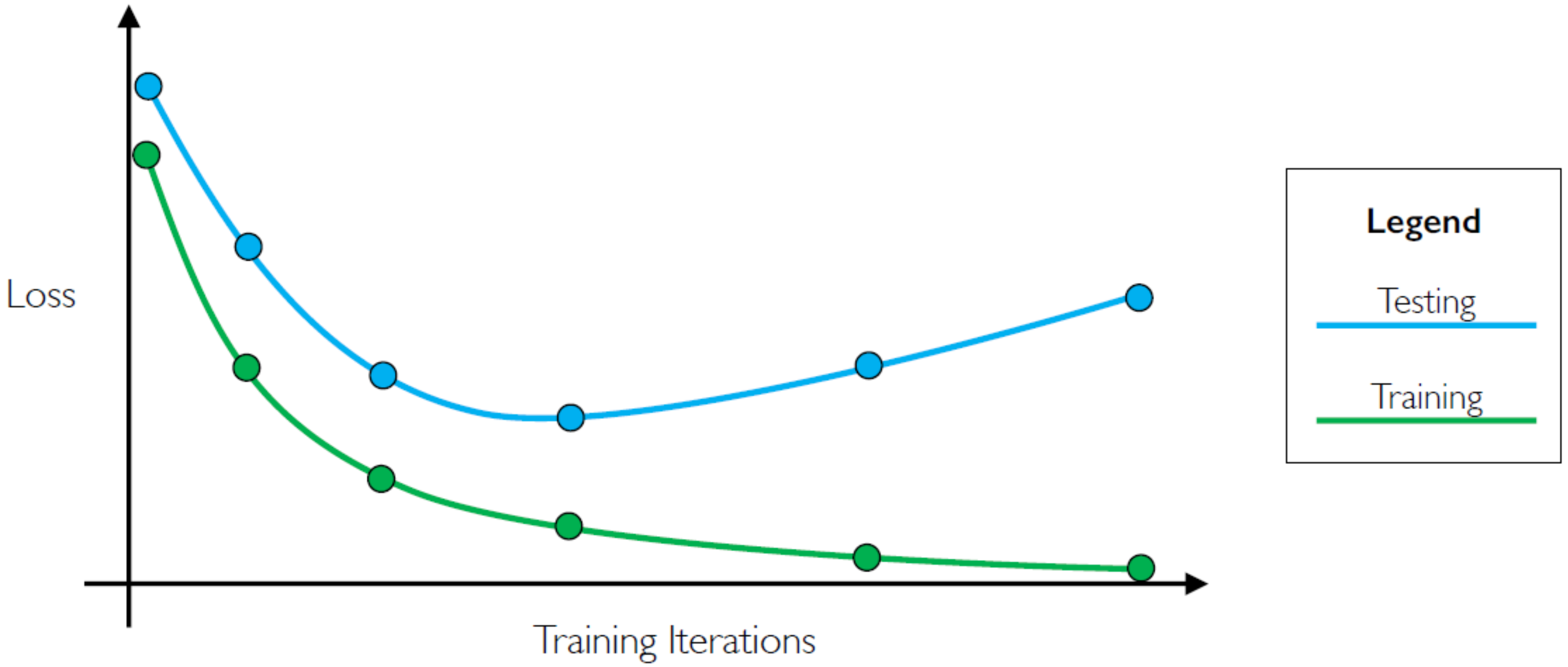
Regularization #1: Early stopping

A simple approach is to take a look at a testing dataset and stop even if the loss on the training is still improving



Regularization #1: Early stopping

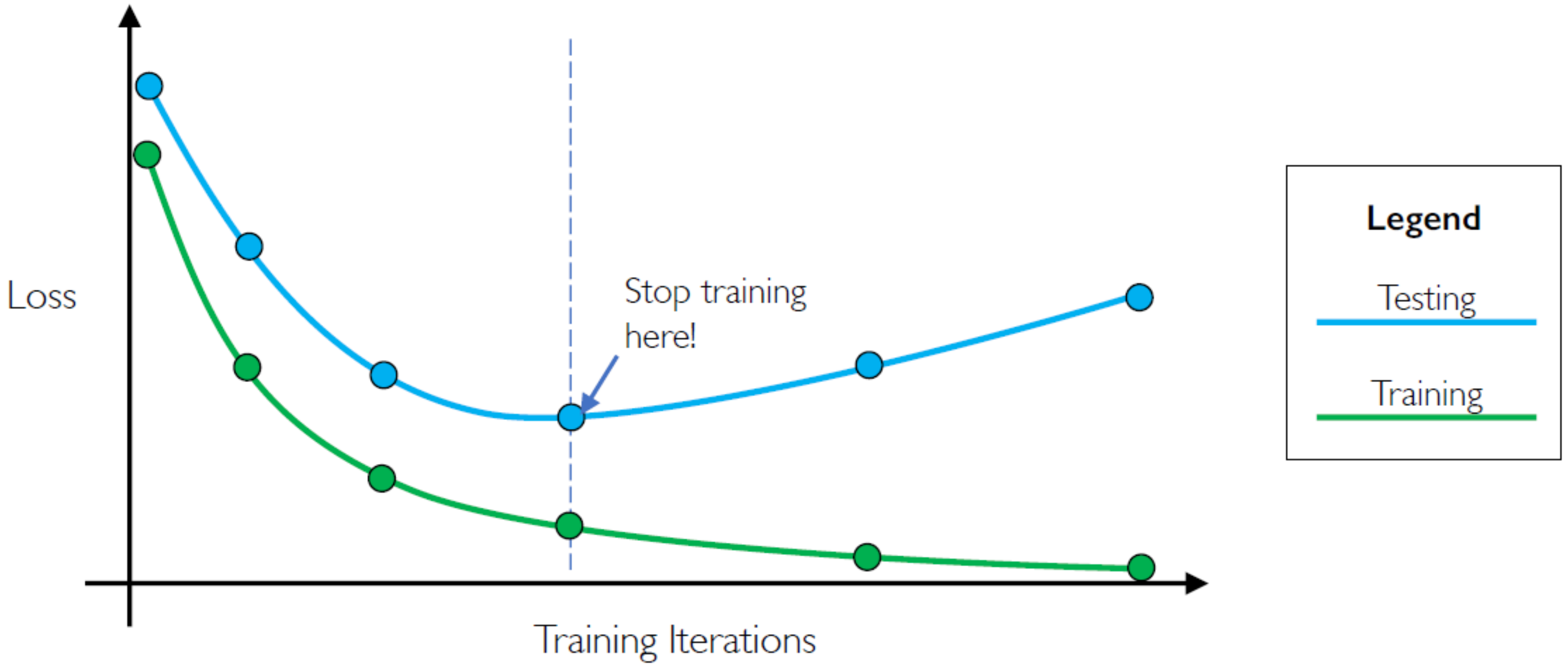
A simple approach is to take a look at a testing dataset and stop even if the loss on the training is still improving



-MIT *Introduction to Deep Learning* http://introtodeeplearning.com

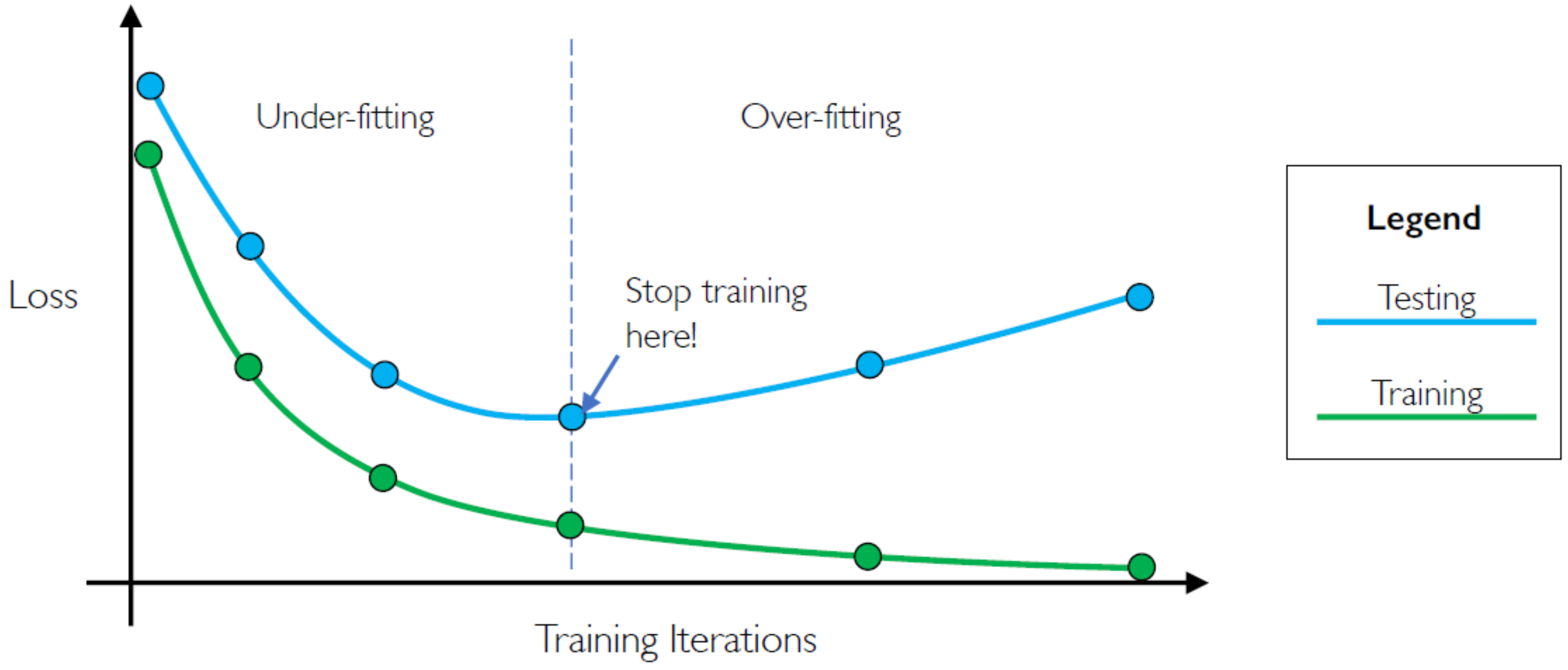
Regularization #1: Early stopping

A simple approach is to take a look at a testing dataset and stop even if the loss on the training is still improving



Regularization #1: Early stopping

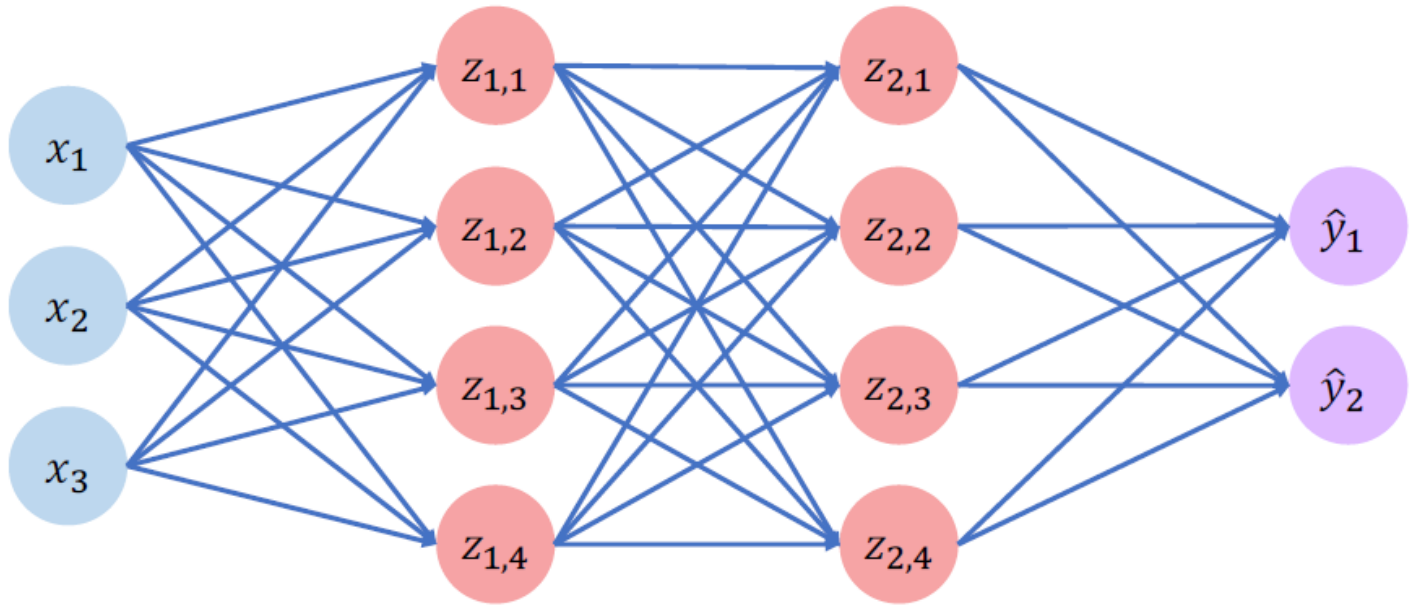
A simple approach is to take a look at a testing dataset and stop even if the loss on the training is still improving



-MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Regularization #2: Drop Out

A popular approach is to set-up some activations equals to 0 during training

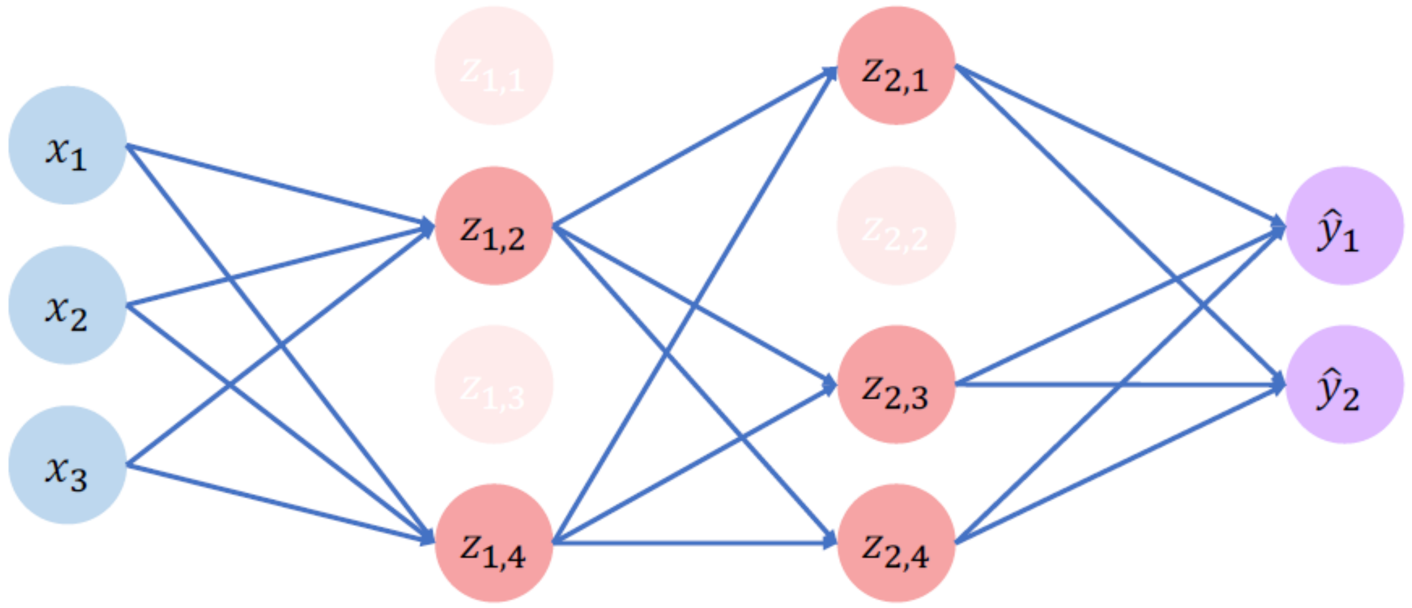


-MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Regularization #2: Drop Out

A popular approach is to set-up some activations equals to 0 during training

Typically, 50% of activations are dropped



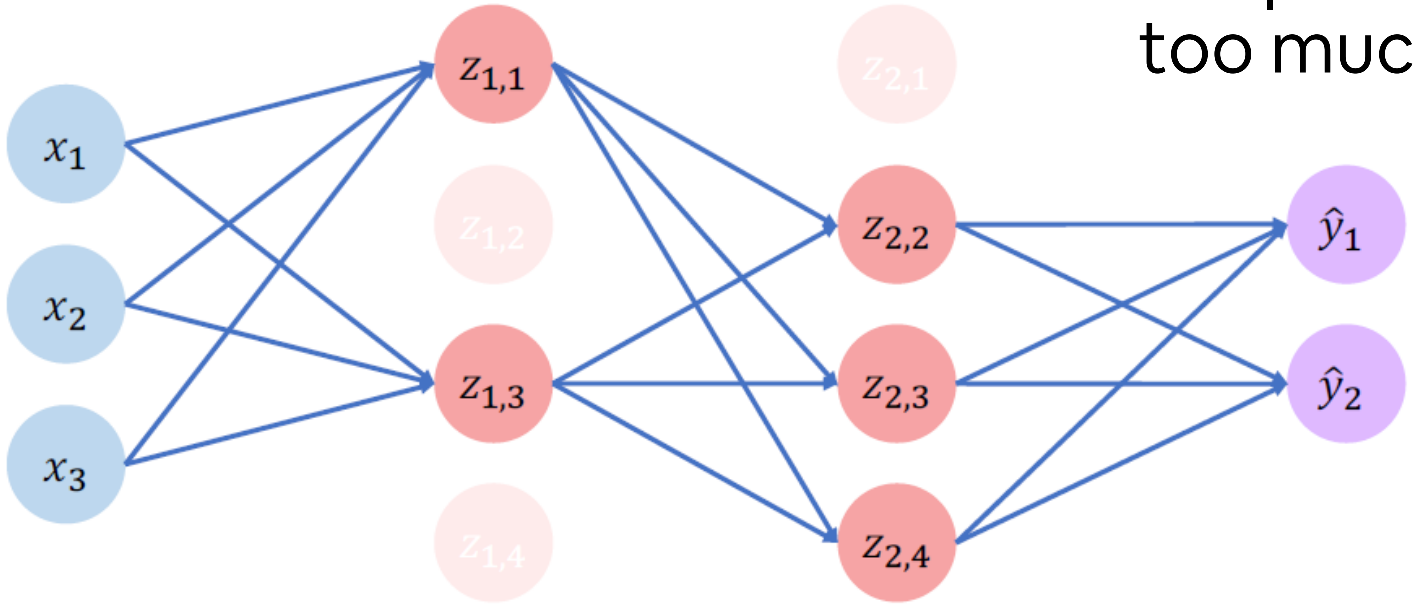
-MIT *Introduction to Deep Learning* http://introtodeeplearning.com

Regularization #2: Drop Out

A popular approach is to set-up some activations equals to 0 during training

Typically, 50% of activations are dropped

Dropout make the model not rely too much on a single node



Regularization #2: Drop Out

A popular approach is to set-up some activations equals to 0 during training

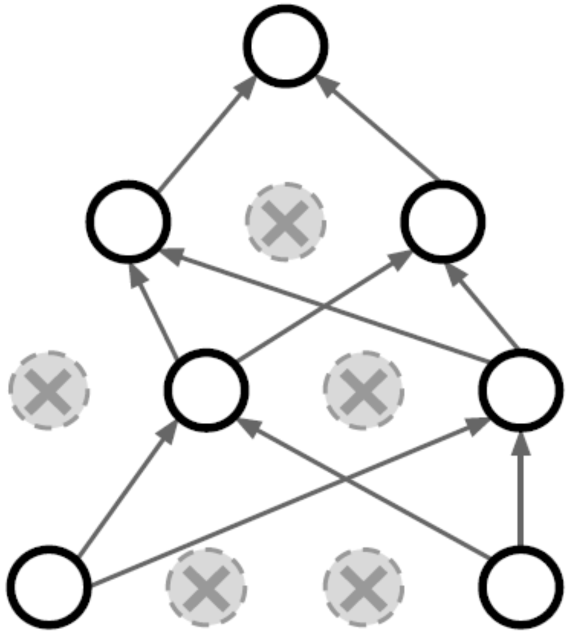
Typically, 50% of activations are dropped

Dropout make the model not rely too much on a single node

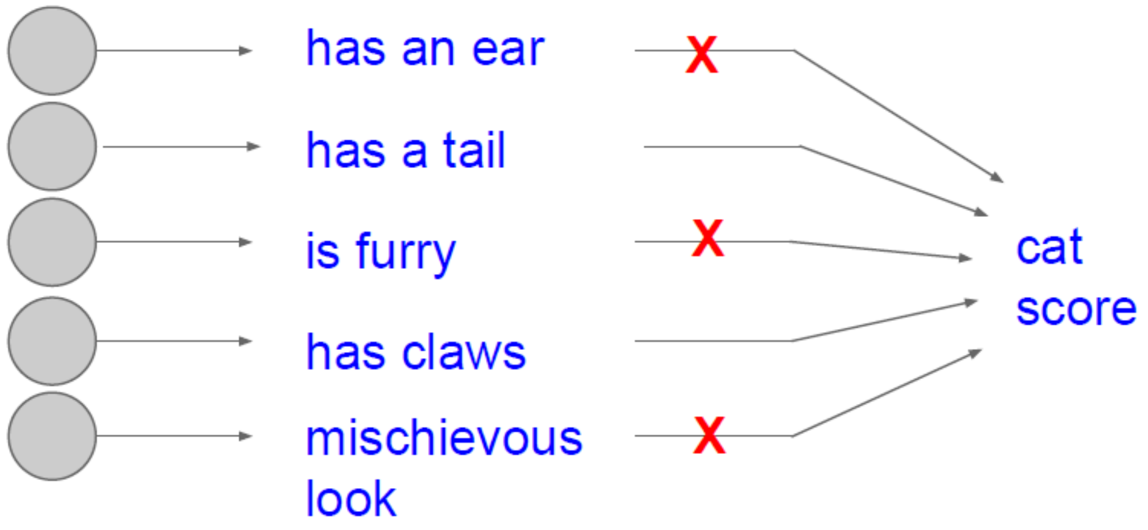
One way to remember it....

Regularization #2: Drop Out

How is that a good idea?

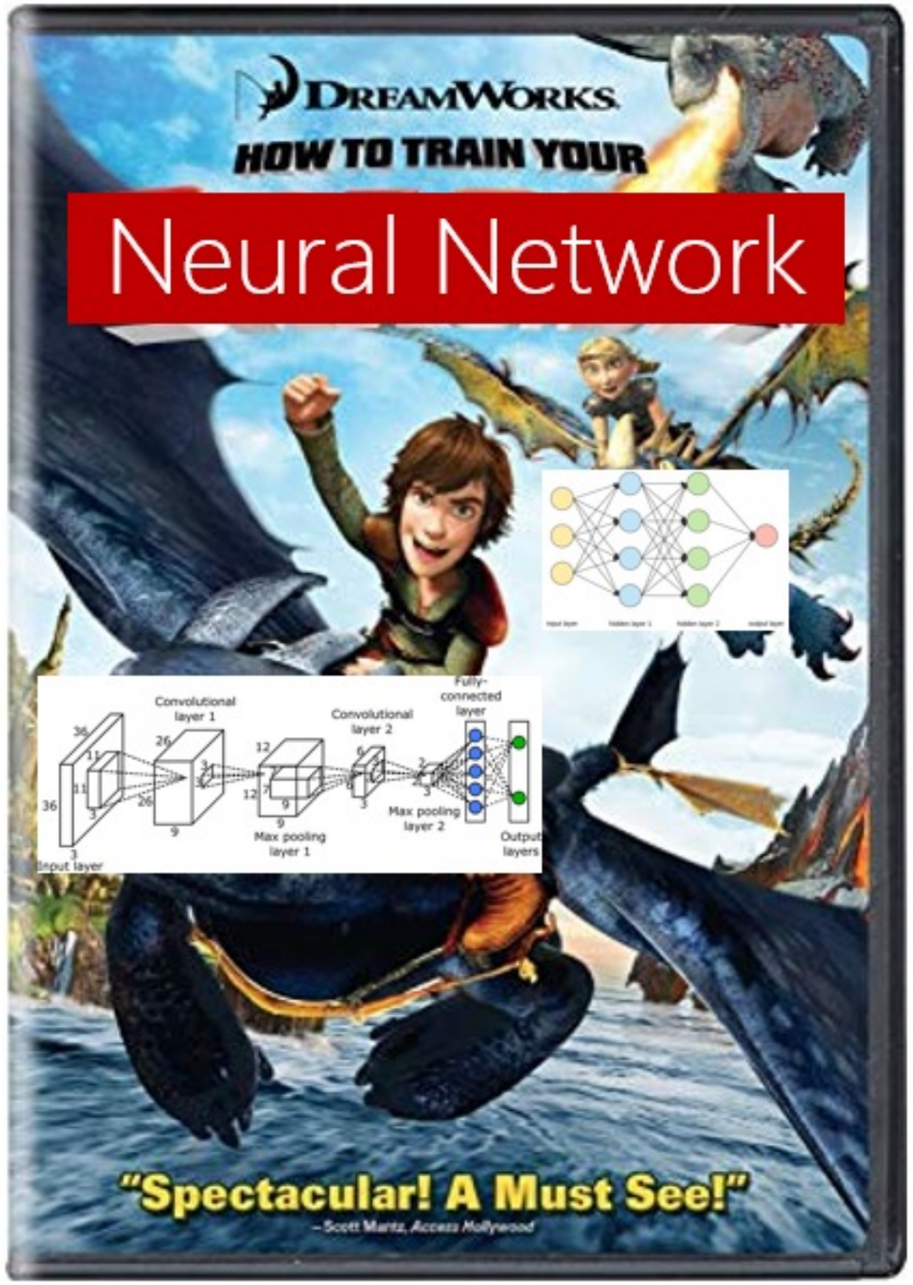


Forces the network to have a redundant representation;
Prevents co-adaptation of features



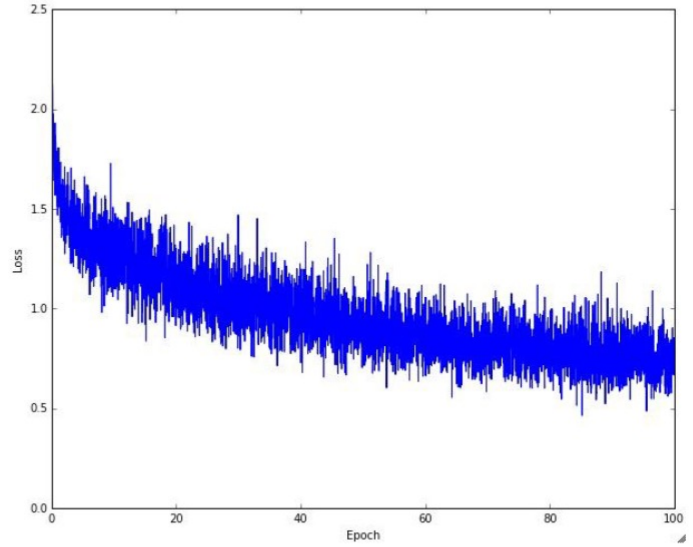
1. Activation functions
2. Weight Initialization
3. Batch Normalization
4. Optimization
5. Learning Rate
6. Regularization

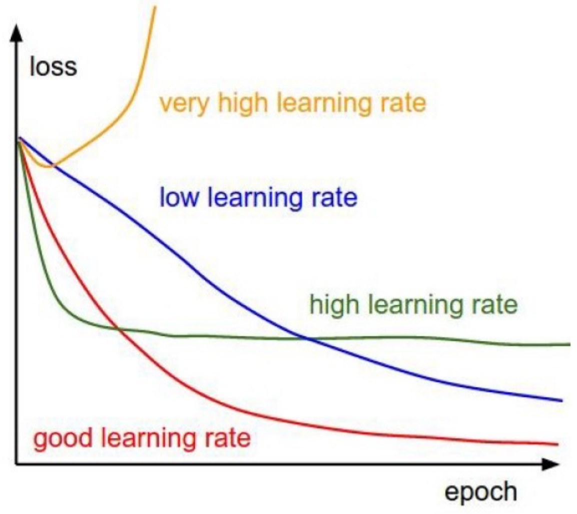
=> Summarizing...



How to train a NN: some tips for your first try

- Scale your input data (preprocessing)
- Use ReLU
- Use He Inizialization
- Use Batch Normalization
- Use ADAM
- Babysit the learning process

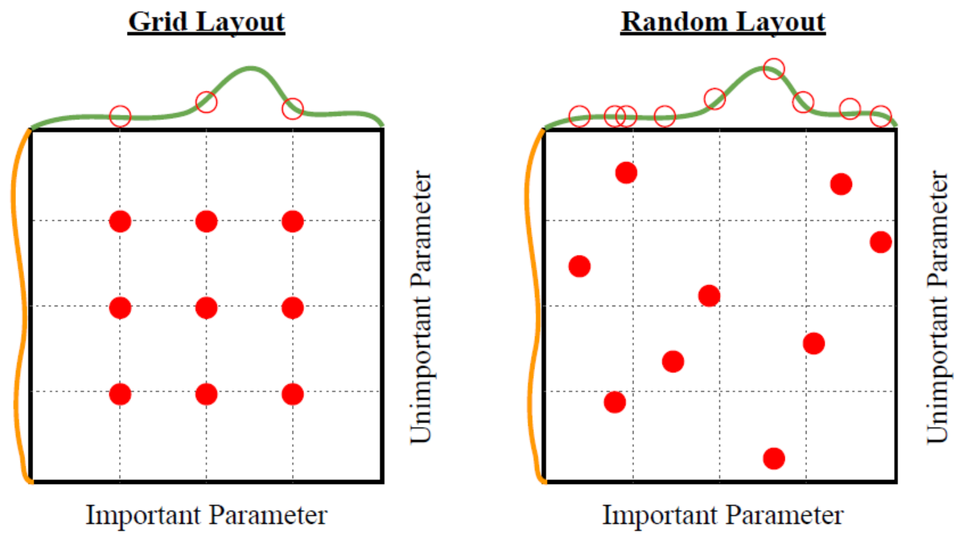




F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/

How to train a NN: some tips for your first try

- Scale your input data (preprocessing)
- Use ReLU
- Use He Inizialization
- Use Batch Normalization
- Use ADAM
- Babysit the learning process
- Hyperparameter Optimization (network architecture, learning rate and decay schedule, presence of regularizathion, more sophisticated activation functions, ...)



F.-F. Li et A. *Convolutional Neural Networks for Visual Recognition* http://cs231n.stanford.edu/
Random Search for Hyper-Parameter Optimization Bergstra and Bengio, 2012

Lot of things not covered today... what about a summary?

New optimization approaches:

- Variants and Enhancements of Adam (AdamW, AMSGrad, AdaBelief)
- Novel Optimizers (Adafactor, MADA)
- Adaptive Learning Rate Methods (AdaMax, Nadam)
- Memory-Efficient Techniques (Adafactor)

New initialization approaches:

- ZerO, Sylvester Solvers, AutoInit, Linear Product Structure (LPS), ...

...

Credits

Reference Material (used for this presentation):

- F.-F. Li et A. Convolutional Neural Networks for Visual Recognition http://cs231n.stanford.edu/
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
- MIT Introduction to Deep Learning http://introtodeeplearning.com
- I. Changhau Activation Functions in Neural Networks https://isaacchanghau.github.io/post/activation_functions/
- H. Li et al. Visualizing the Loss Landscape of Neural Nets NIPS 2018
- X. Glorot, Y. Bengio Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256), 2010
- F. Doukkali https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c
- K. He et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification https://arxiv.org/pdf/1502.01852.pdf
- J. Jordan https://www.jeremyjordan.me/batch-normalization/
- S. Ioffe, C. Szegedy Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift https://arxiv.org/abs/1502.03167
- Sutskever et al, On the importance of initialization and momentum in deep learning, ICML 2013
- Nesterov, A method of solving a convex programming problem with convergence rate O(1/k^2), 1983
- Duchi et al, Adaptive subgradient methods for online learning and stochastic optimization, JMLR 2011
- Basu, Amitabh, et al. Convergence guarantees for rmsprop and adam in non-convex optimization and their comparison to nesterov acceleration on autoencoders. arXiv preprint arXiv:1807.06766 (2018)
- Kingma and Ba, Adam: A method for stochastic optimization, ICLR 2015
- S. Ruder http://ruder.io/optimizing-gradient-descent/index.html#gradientdescentoptimizationalgorithms
- Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
- Goyal et al, Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, arXiv 2017
- Random Search for Hyper-Parameter Optimization Bergstra and Bengio, 2012

UNIVERSITÀ DEGLI STUDI DI PADOVA

Machine Learning 2024/2025

AMCO ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Thank you!

Gian Antonio Susto