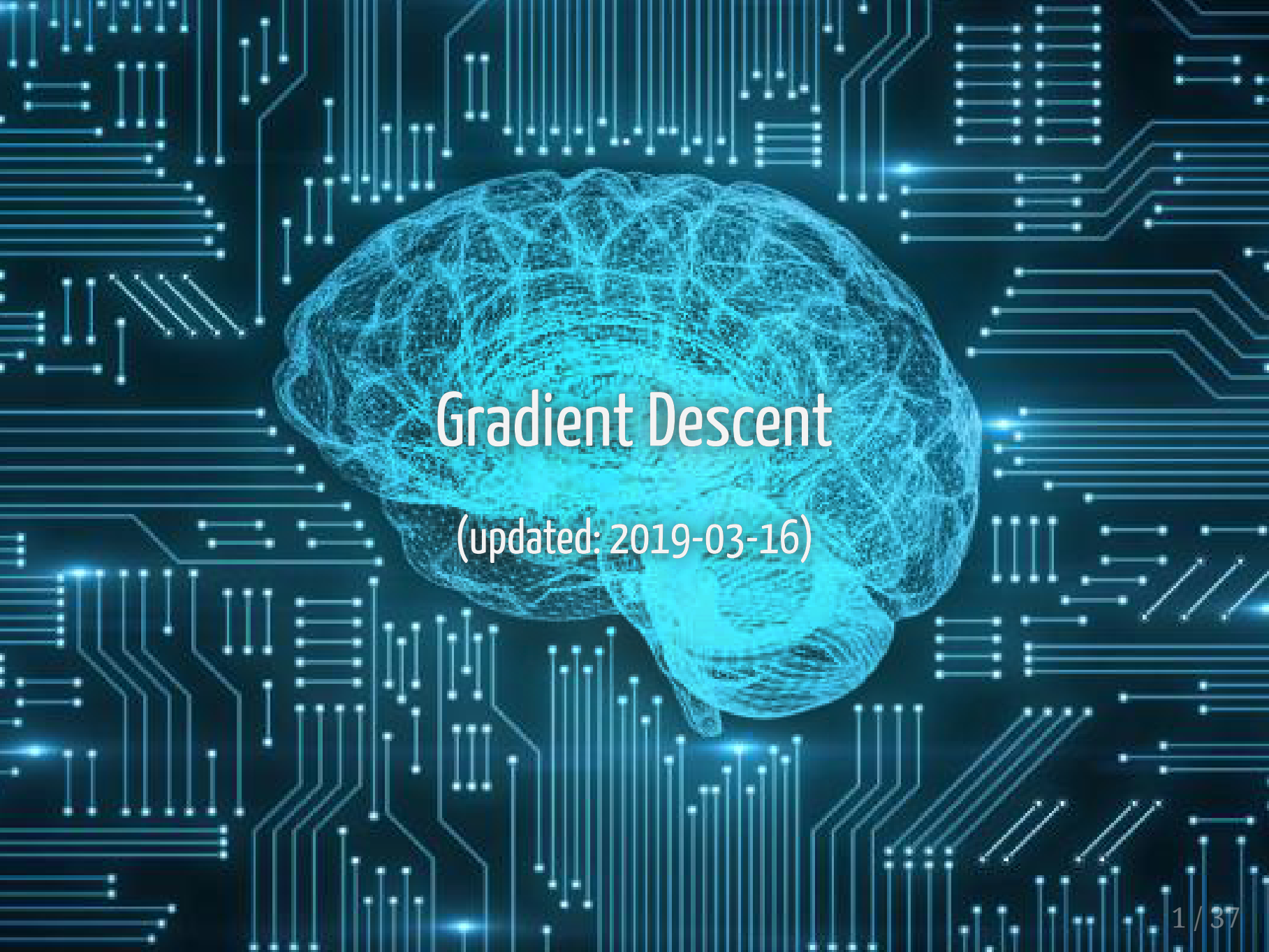


Gradient Descent

(updated: 2019-03-16)

What We Know

- Neural network will have 1 or more hidden layers
- Know how to match cost function with output layer activation based on task
- Will use gradient descent to incrementally improve randomly initialized adjustable parameters
- Know how to use forward propagation and backward propagation to compute cost function gradient.

Adding 3 more pieces:

- Parameter update algorithms
  - Exponential moving averages
  - Momentum, Nesterov momentum, adaptive methods
- Hidden node activation functions
  - ReLU, swish
- Learning rate schedules
  - Manual, table, algorithmic

Gradient-Descent Optimization

Computing the gradient on all data samples is computationally expensive. (Used this method for 1-layer networks.)

Definitions

- Batch GD: Compute gradient using all available data. For this method sample order does not matter.
- Mini-batch GD: Approximate the gradient using m selected (shuffled) training samples.
- Stochastic GD: Approximate the gradient using a single training data point.

Mini-batch GD is almost always used

- Noisy gradients help solution move away from saddle points and poor local minima.
- Using small batches has a positive 'regularization' effect.

Gradient Descent Optimization

Epoch

An epoch is one complete pass through the training data.

Step

A step is one update of the adjustable parameters $\boldsymbol{\theta}$

- Up to now we've had one step/epoch - ie. we used all of the data in each gradient calculation.

Using mini-batch updates means multiple update steps/epoch.

Given training data

$$\mathcal{D} = \left\{x^{(i)}, \cdots, x^{(m)}\right\}, \left\{t^{(i)}, \cdots, t^{(m)}\right\}$$

If mini-batch size is m_b, then there will be approximately m/m_b update steps per epoch.

Baby Sitter algorithm

Manually adjust learning rate

Initialize parameters
$\mathbf{b}^{[\ell]} \leftarrow 0,\ W^{[\ell]} \leftarrow \mathcal{N}(0, .1), \quad \ell = 1, 2, \ldots, L$

Choose

- Batch size m
- Initial learning rate α_0

$n \leftarrow 0$
$\alpha \leftarrow \alpha_0$
For each batch:
$\mathbf{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla C(\mathbf{x}^{(i)}, t^{(i)}; \boldsymbol{\theta})$
$C = C(\boldsymbol{\theta})$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \mathbf{g}$
$n \leftarrow n + 1\ if\ C(\boldsymbol{\theta}) \leq C$
end

if $n < \#\ of\ batches\ \alpha \leftarrow \alpha/2$ run again

Gradient Descent Optimization

Gradient Descent with Learning Rate Schedule

Initialize parameters $\boldsymbol{\theta}$
$\boldsymbol{\theta} = \left\{\mathbf{b}^{[1]}, W^{[1]}, \cdots, \mathbf{b}^{[L]}, W^{[L]}\right\}$

Initialize learning rate α
for epoch in $\{1, \cdots, \text{max\_epoch}\}$
  $\mathbf{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\theta} C(\mathbf{x}^{(i)}, t^{(i)}; \boldsymbol{\theta})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \mathbf{g}$
  $\alpha \leftarrow f(\alpha, epoch)$
end

If gradient is computed on the entire data set, the method is called gradient descent or batch gradient descent.

Gradient Descent Optimization

Rate Decay Mini-batch Gradient Descent

Initialize parameters $\boldsymbol{\theta}$

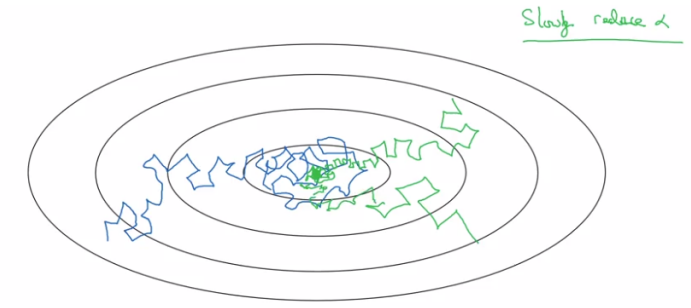
Choose

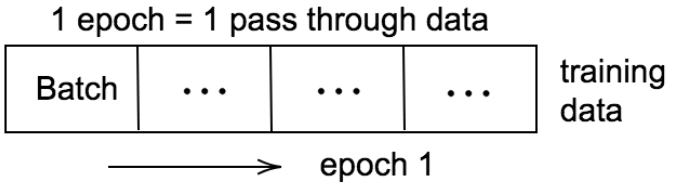
- Batch size m
- Initial learning rate α_0
- Final learning rate α_f
- Number of training epochs N_e

Linear Rate Decay Set $\alpha = \alpha_0$
For epoch in $1, \cdots, N_e$
  For batch in $1, \cdots, N_b$
    $\mathbf{g} = \frac{1}{m} \sum_{i=1}^{m^b} \nabla C(\mathbf{x}^{(i)}, t^{(i)}; \theta)$
    $\theta \leftarrow \theta - \alpha \mathbf{g}$
  end
  $\alpha \leftarrow (1 - \frac{epoch}{N_e})\alpha_0 + \frac{epoch}{N_e}\alpha_f$
end

Note: There are many learning rate decay rules

Learning Rate Decay





Method 1

1 epoch = 1 pass through data. For decay rate δ

$$\alpha = \frac{1}{1 + \delta \cdot N_e} \alpha_0$$

Epoch	C_α	α
1	1/2	.1
2	1/3	.2/3
3	1/4	.05
4	1/5	.2/5

$$\alpha_0 = .2, \delta = 1$$

Learning Rate Decay (Schedule)

Method 2

$$\alpha = .95^{N_e} \cdot \alpha_0$$

Method 3

$$\alpha = \frac{k}{\sqrt{N_e}} \cdot \alpha_0$$

Method 4

$$\frac{k}{\sqrt{t}} \cdot \alpha_0$$

Method 5

Set α using a table

Method 6

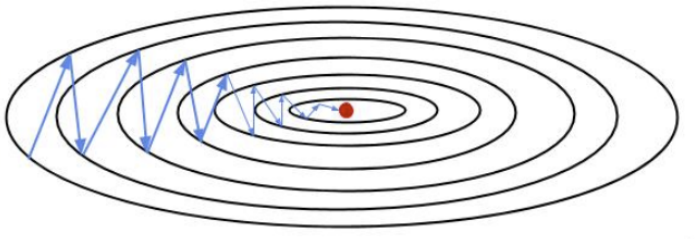
Manual decay (Babysitter Algorithm)

Optimization Algorithms

Descent algorithms are an active area of current research. Some recent publications claim simple batch descent with momentum result in better generalization than sophisticated adaptive algorithms

Vanilla Descent

- **Plus:** easy to understand.
- **Plus:** has just one tunable parameter (learning rate)
- **Con:** can oscillate when trapped in a narrow valley
  - This behavior is common



Optimization Algorithms

Counteracting oscillations in parameter updates requires understanding momentum and exponentially weighted moving averages (EMA).

Assume that θ_t is a measurement and the goal is to derive a smoothed value for θ

Moving window

$$v = \frac{1}{n}\sum_{i=1}^{n}\theta_i$$

- Requires remembering a history of n θ_i values

Exponentially Moving Average (EMA)

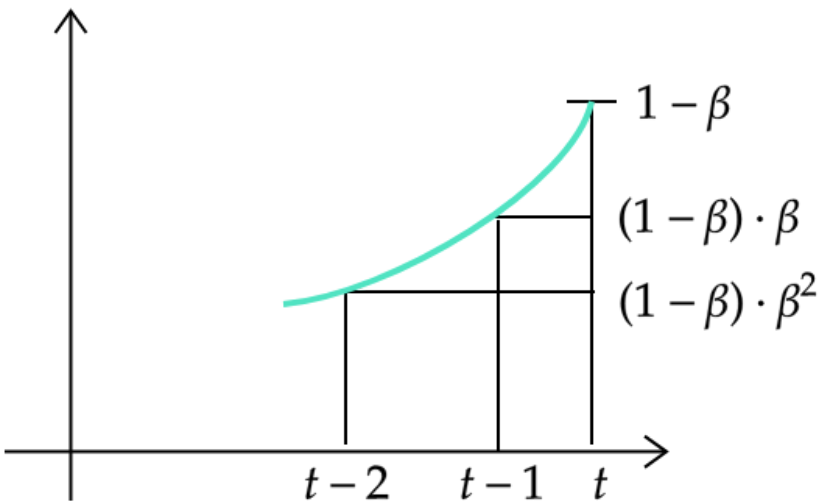
$$\begin{aligned} v_0 &= 0 \\ v_t &= \beta v_{t-1} + (1-\beta)\theta_t \end{aligned}, \quad t = 1, 2, \ldots$$

- Only need to remember the last averaged value
- Weights most recent measurements the most.

Optimization Algorithms

$$
\begin{aligned}
v_t &= (1-\beta)\theta_t + \beta v_{t-1} \\
&= (1-\beta)\theta_t + \beta\Big((1-\beta)\theta_{t-1} + \beta v_{t-2})\Big) \\
&= (1-\beta)\theta_t + \beta\Big((1-\beta)\theta_{t-1} + \beta\Big((1-\beta)\theta_{t-2} + \beta v_{t-3})\Big)\Big) \\
&= (1-\beta)\theta_t + (1-\beta)\beta\theta_{t-1} + (1-\beta)\beta^2\theta_{t-2} + \cdots
\end{aligned}
$$

Coefficients in average exponential decay



Optimization Algorithms

If $\beta = .9$

$$v_t = .1\left[.9^0\theta_t + .9^1\theta_{t-1} + (.9)^2\theta_{t-2} + \cdots\right]$$

Compute number of terms until θ coefficient drops to e^-1

$$e^{-1} = \lim_{n\to\infty}\left(1 - \frac{1}{n}\right)^n$$

$$(1 - \frac{1}{n}) = .9 \Rightarrow n = 10$$

So coefficient weight drops to $1/e$ of weight of θ_t when $n = 1/(1-\beta)$

Check

$(1 - \frac{1}{10})^{10} = .9^{10} \approx .348$
$\frac{1}{e} \approx .367$

For this reason call $1/(1-\beta)$ the averaging window

Bias Correction

$$v_t = c_0\theta_t + c_1\theta_{t-1} + c_2\theta_{t-2} + \cdots$$
$$v_0 = 0$$
$$v_1 = (1-\beta)\theta_1$$
$$\sum_{i=1}^{1} c_i = 1-\beta$$
$$v_2 = \beta(1-\beta)\theta_1 + (1-\beta)\theta_2$$
$$\sum_{i=1}^{2} c_i = \beta(1-\beta) + 1 - \beta = 1-\beta^2$$
$$v_3 = \beta\left[\beta(1-\beta)\theta_1 + (1-\beta)\theta_2\right] + (1-\beta)\theta_3$$
$$\sum_{i=1}^{3} c_i = (1-\beta)(1+\beta+\beta^2) = 1-\beta^3$$

In general

$$\sum_{i=1}^{t} c_i = 1-\beta^t$$

Initially, the values of v_t are too small. This can be corrected by scaling the coefficients so they sum to one.

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

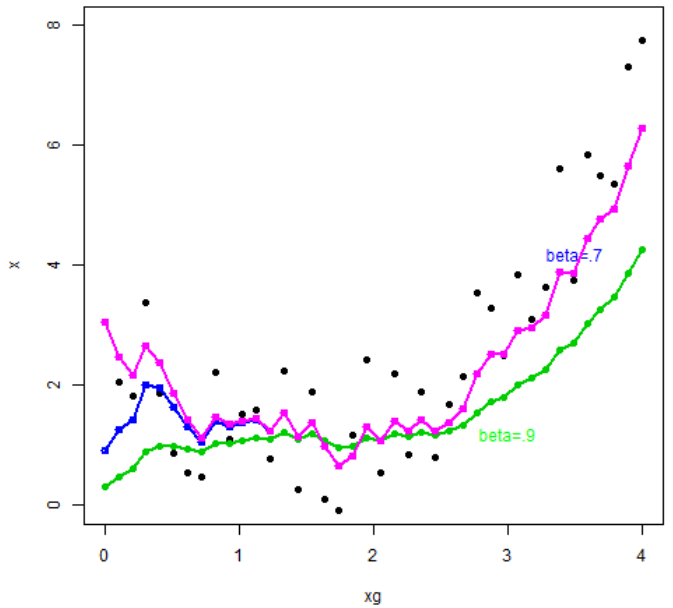
$$\hat{v} = \frac{v_t}{1-\beta^t}$$

$\hat{v}$ is called the bias corrected exponential moving average of θ

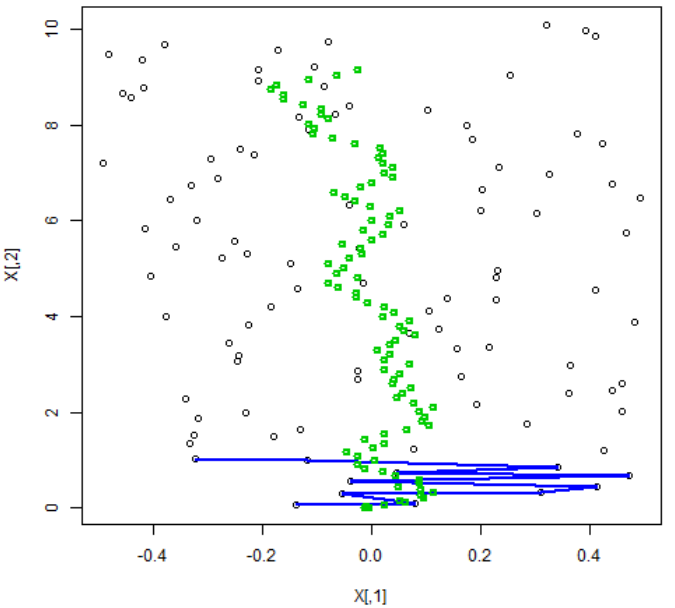
Note: As t increases $\beta^t \to 0, \hat{v} \approx v_t$

Exponential Moving Average

1-d EMA

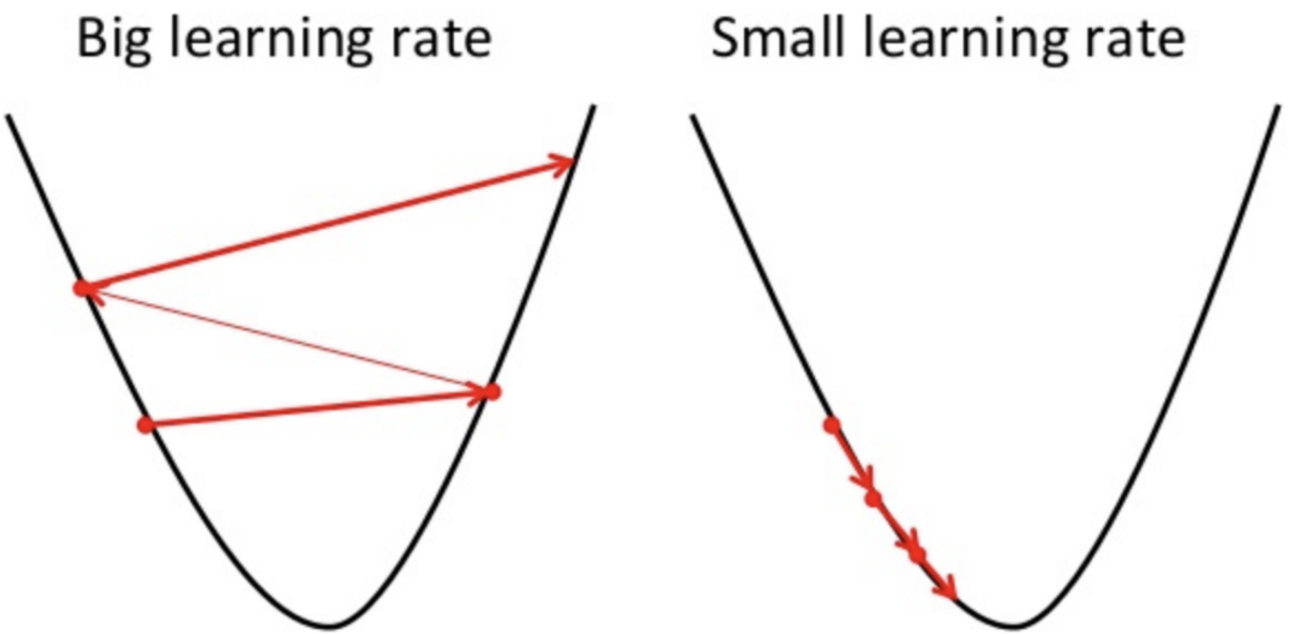


2-d EMA



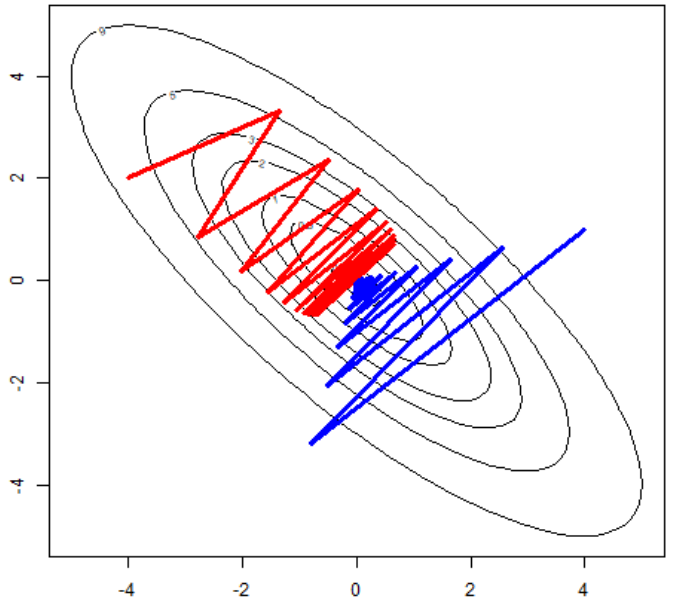
Gradient Descent - Learning Rate

Convexity is not enough to save you from large learning rate. Adaptive methods help avoid divergence by reducing learning in directions where gradients are large



Gradient Descent

Learning rates were chosen to demonstrate oscillations. Smaller rates would converge

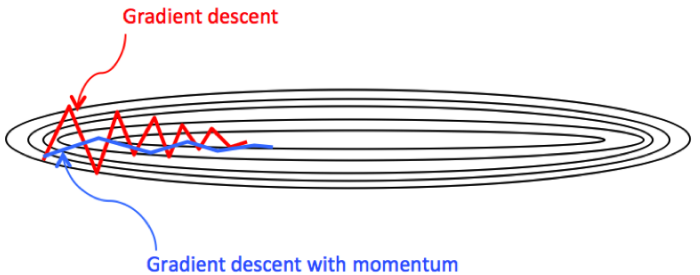


Gradient Descent With Momentum

Apply efficient averaging to the sequence of gradient vectors to cancel oscillations while accelerating non-oscillating components.

$$
\begin{aligned}
v_W &= \beta\, v_W + (1-\beta)\nabla_W C \\
v_b &= \beta\, v_b + (1-\beta)\nabla_b C \\
W &= W - \alpha\, v_W \\
b &= b - \alpha\, v_b
\end{aligned}
$$

GD with momentum has 2 hyper-parameters: learning rate α and the momentum averaging parameter β



Gradient Descent With Momentum

Warning!

- The momentum equations here (should) match the equations used by Ng.
- This form keeps the learning rate and momentum parameters separate.
  - That will be important when tuning.

Many authors use the following form:

$$
\begin{aligned}
v_w &= \beta\, v_w + \alpha\, \nabla_W C \\
v_b &= \beta v_b + \alpha\, \nabla_b C \\
W &= W - v_w \\
b &= b - v_b
\end{aligned}
$$

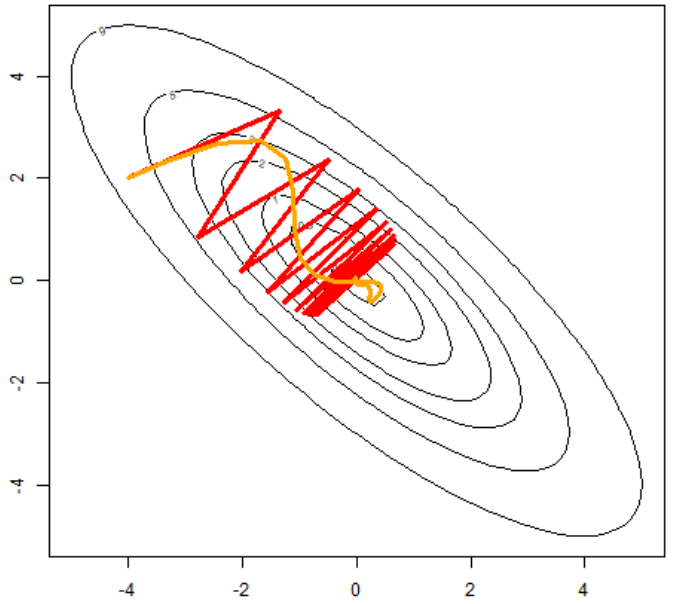
This form confounds the parameters α and β

Preferred form:

$$
\begin{aligned}
v_w &= \beta\, v_w + (1-\beta) \nabla_W C \\
v_b &= \beta\, v_b + (1-\beta) \nabla_b C \\
W &= W - \alpha\, v_W \\
b &= b - \alpha\, v_b
\end{aligned}
$$

Gradient Descent with Momentum

Red: $\alpha = .55$
Orange: $\alpha = .55, \beta = .75$



Nesterov Momentum

Traditional momentum methods smooth the gradient computed from the current values of the adjustable parameters.

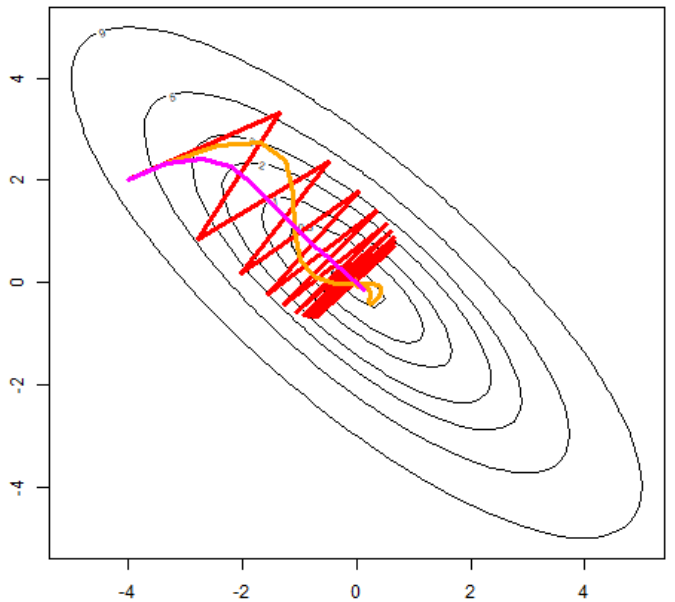
Nesterov momentum does a "look-ahead" gradient calculation

$$
\begin{aligned}
\mathbf{b}^* &= \mathbf{b} - \alpha v_b \\
W^* &= W - \alpha v_w \\
v_w &= \beta\, v_w + (1-\beta)\nabla_W C(\text{batch}; \mathbf{b}^*, W^*) \\
v_b &= \beta\, v_b + (1-\beta)\nabla_b C(\text{batch}; \mathbf{b}^*, W^*) \\
W &= W - \alpha\, v_W \\
b &= b - \alpha\, v_b
\end{aligned}
$$

Nesterov momentum has been shown to speed up training

Gradient Descent with Nesterov Momentum

Red: $\alpha = .55$
Orange: $\alpha = .55, \beta = .75$
Purple: $\alpha = .55, \beta = .75$, Nesterov momentum



Adaptive Gradient Methods

RMSProp

Root mean square propagation is an optimization algorithm which adapts the learning rate for each adjustable parameter.

- Setting the learning rate is challenging for neural networks.
- RMSprop normalize gradients using EMA values of the gradient squared:

$$
\begin{aligned}
S_w &= \beta S_w + (1-\beta)(\nabla_W C \odot \nabla_W C) \\
S_b &= \beta S_b + (1-\beta)(\nabla_b C \odot \nabla_b C) \\
W &= W - \alpha \frac{\nabla_W C}{\sqrt{S_w + \epsilon}} \quad \text{elementwise operations} \\
b &= b - \alpha \frac{\nabla_b C}{\sqrt{S_b + \epsilon}} \quad \text{elementwise operations}
\end{aligned}
$$

Adaptability means a larger α can be used.

If a component of ∇C is large, then the effective learning rate for that component is reduced.

Adaptive Gradient Methods

Adam

- **Adaptive Moment Estimation** combines momentum with RMSprop.
- This algorithm uses β_1 for the momentum EMA and β_2 for the gradient normalization EMA.

$$
\begin{aligned}
v_w &= \beta_1 v_w + (1-\beta_1)\nabla_W C \\
v_b &= \beta_1 v_b + (1-\beta_1)\nabla_b C \\
S_w &= \beta_2 S_w + (1-\beta_2)(\nabla_W C \odot \nabla_W C) \\
S_b &= \beta_2 S_b + (1-\beta_2)(\nabla_b C \odot \nabla_b C) \\
\hat{v}_w &= \frac{v_w}{1-\beta_1^t}, \hat{v}_b = \frac{v_b}{1-\beta_1^t} \\
\hat{S}_w &= \frac{S_w}{1-\beta_2^t}, \hat{S}_b = \frac{S_b}{1-\beta_2^t}
\end{aligned}
$$

$$
\begin{aligned}
W &= W - \alpha \frac{\hat{v}_w}{\sqrt{\hat{S}_W + \epsilon}} \\
b &= b - \alpha \frac{\hat{v}_b}{\sqrt{\hat{S}_b + \epsilon}}
\end{aligned}
$$

Parameters:

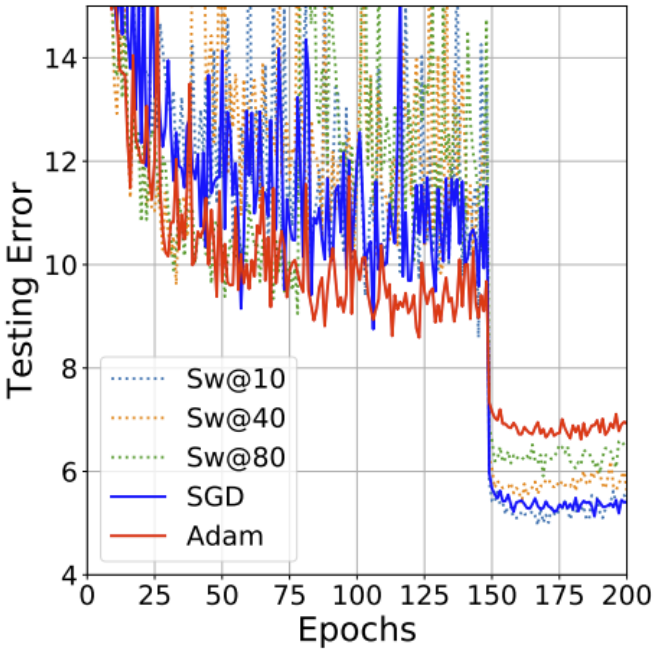
$$
\begin{aligned}
\alpha &= && \text{(learning rate)} \\
\beta_1 &= .9 && \text{(momentum)} \\
\beta_2 &= .999 && \text{(gradient normalization)} \\
\epsilon &= 10^{-8}
\end{aligned}
$$

Adaptive Gradient Methods

There could be more to this story:

"Improving Generalization Performance by Switching from Adam to SGD" --- Kreskar and Socher, December 2017

- Adaptive optimization methods have been found to generalize poorly compare to SGD.
- These methods often perform well initially but are outperformed by SGD in later stages of training.
- SWATs: switches from Adam to SGD when a condition is met.



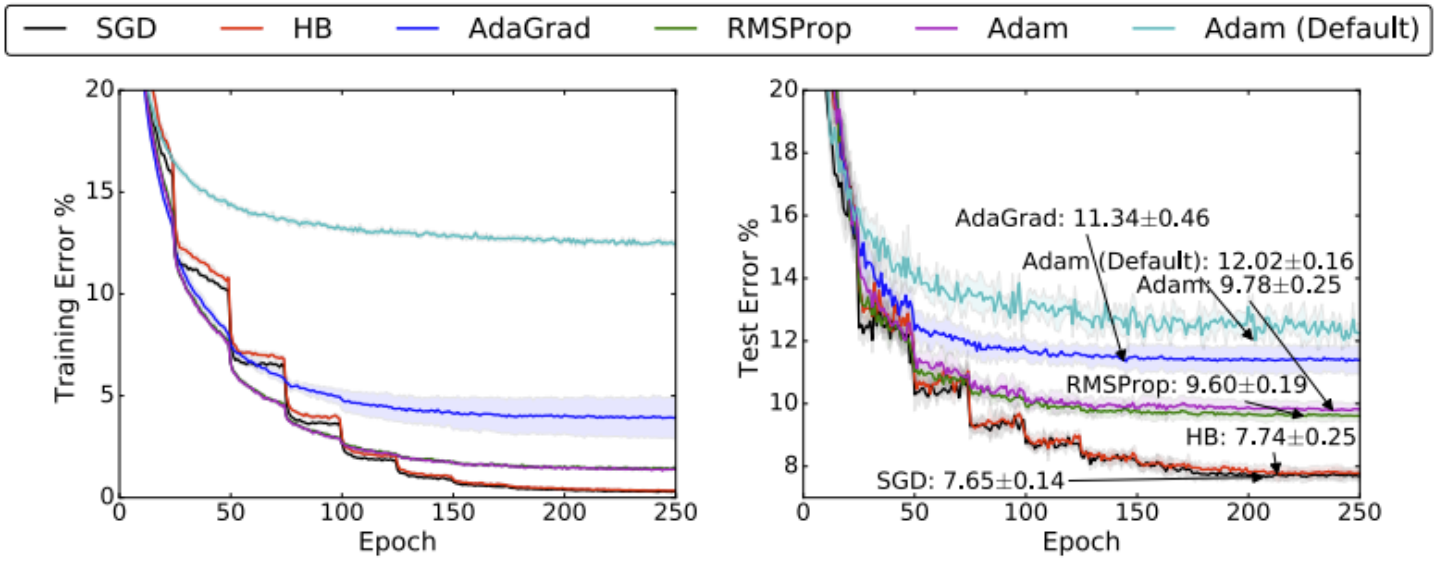
Adaptive Gradient Methods

"The Marginal Value of Adaptive Gradient Methods in Machine Learning"---
Wilson, Roelofs, Stern, Srebro and Recht, May 2018

- We show that for simple overparameterized problems, adaptive methods often find drastically different solutions than GD or SGD.
- Constructed a binary classification problem where the data is linearly separable, GD and SGD achieve zero test errors and AdaGrad, Adam and RMSprop attain test errors arbitrarily close to half.
- We observe that the solutions found by adaptive methods generalize worse (often significantly worse) than SGD, even when these solutions have better training performance.

Adaptive Gradient Methods

HB = Heavy Ball = Momentum

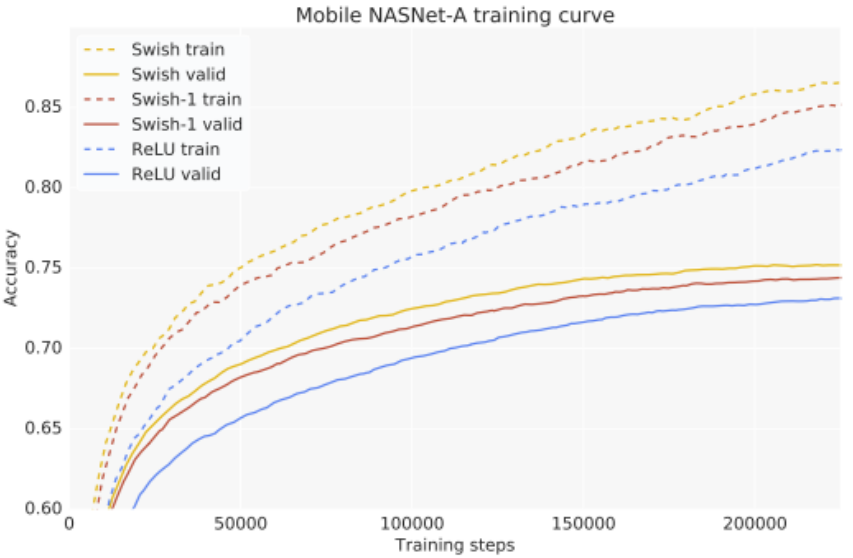


(a) CIFAR-10 (Train)

(b) CIFAR-10 (Test)

Searching for Activation Functions

- Activation functions affect training dynamics.
- ReLu most common.
- Discovered Swish through automatic search techniques.
- Switching ReLu to Swish improved classification models from .6% to .9%



Activation Functions

- Previously showed that non-linear activation functions are needed to solve the XOR problem
- Activation functions applied to net-values compute the output values of hidden layer nodes.
- Neural networks with ≥ 1 hidden layers and nonlinear activation functions are universal approximators and universal classifiers.
- Early neural networks relied on sigmoid activation:

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

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

- Activation functions are still an area of research. Google recently introduced the "swish" activation

Sigmoid (Logistic) Function

Because σ' tends toward 0 for $\mathbf{x}$ away from zero, backpropagation with sigmoid activations can have vanishing gradients that cause training to stall

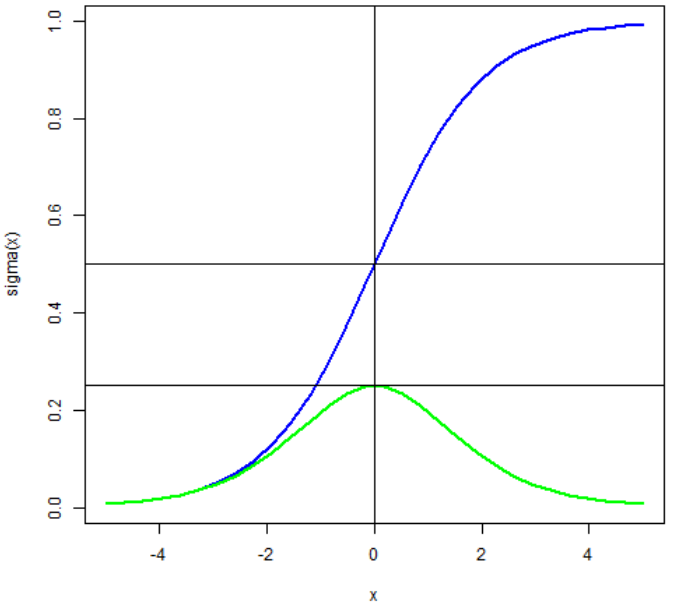
Standard Sigmoid Function

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma(0) = \frac{1}{2}$$

$$\sigma' = \sigma(1-\sigma), \quad \sigma'(0) = \frac{1}{4}$$

$$\lim_{x\uparrow+\infty} \sigma(x) = 1, \lim_{x\downarrow-\infty} \sigma(x) = 0$$

$$\lim_{x\uparrow+\infty} \sigma'(x) = 0, \lim_{x\downarrow-\infty} \sigma'(x) = 0$$



The sigmoid function is the CDF of the logistic distribution

Hyperbolic Tangent Function

The symmetric range of $\tanh$ is an improvement of σ, but $\tanh$ also suffers from vanishing derivatives for $\mathbf{x}$ away from 0

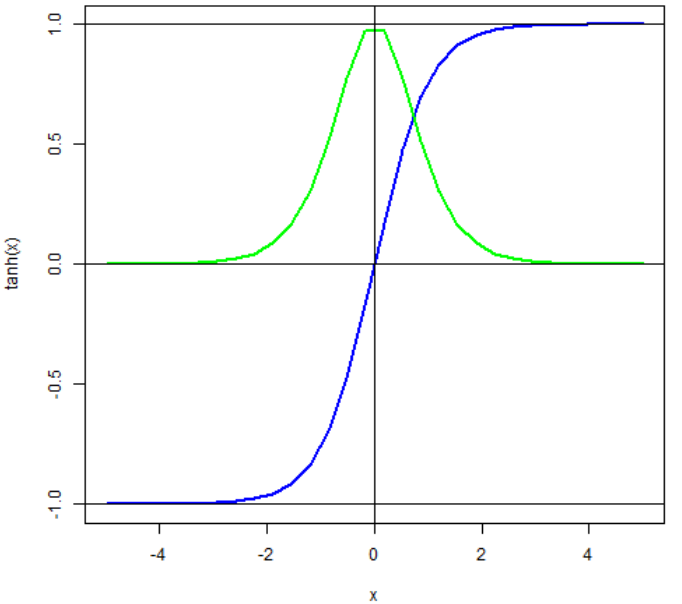
Hyperbolic Tangent Function

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \tanh(0) = 0$$

$$\tanh' = (1 - \tanh^2), \quad \tanh'(0) = 1$$

$$\lim_{x \uparrow +\infty} \tanh(x) = 1, \lim_{x \downarrow -\infty} \tanh(x) = -1$$

$$\lim_{x \uparrow +\infty} \tanh'(x) = 0, \lim_{x \downarrow -\infty} \tanh'(x) = 0$$



ReLu - Rectified Linear Unit

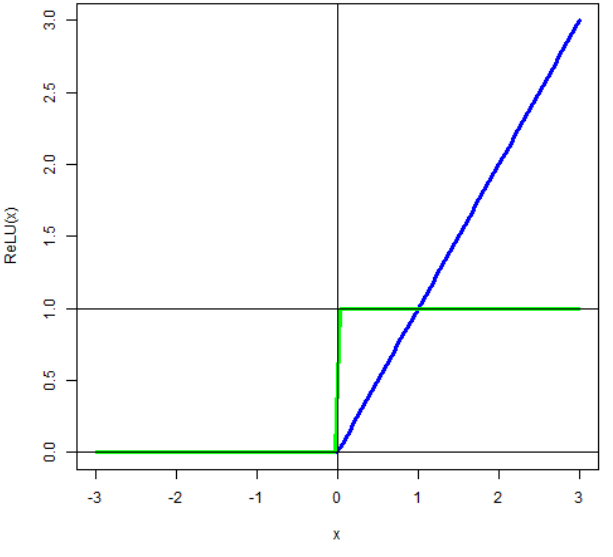
ReLu is the most common activation function used for deep nets.

- Computationally efficient
- Doesn't squash positive inputs, can lead to more efficient training
- Gradient for $x > 0$ does not vanish

ReLU Function

$$f(x) = \max(0, x)$$

$$f' = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{o.w.} \end{cases}$$



Leaky ReLu/Parametric Leaky ReLu

Leaky ReLU Function

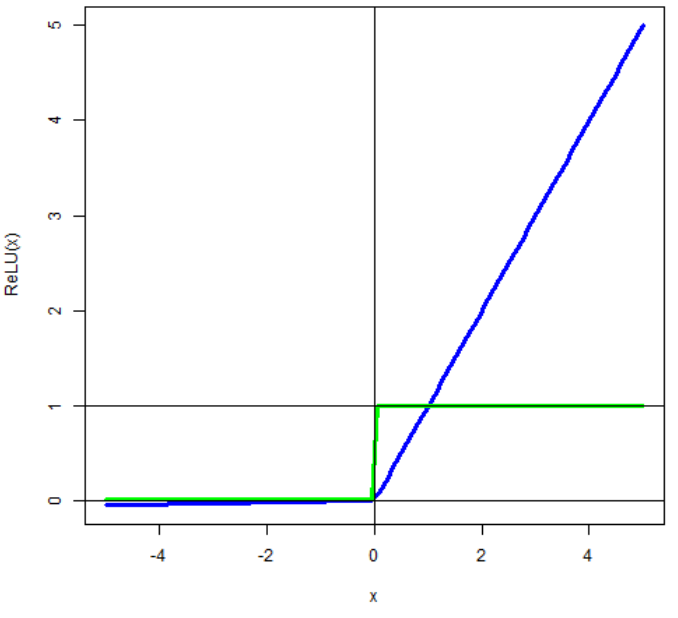
$$f = \begin{cases} .01x & \text{for } x < 0 \\ x & \text{o.w.} \end{cases}$$

$$f' = \begin{cases} .01 & \text{for } x < 0 \\ 1 & \text{o.w.} \end{cases}$$

Parametric Leaky ReLU Function

$$f = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{o.w.} \end{cases}$$

$$f' = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{o.w.} \end{cases}$$



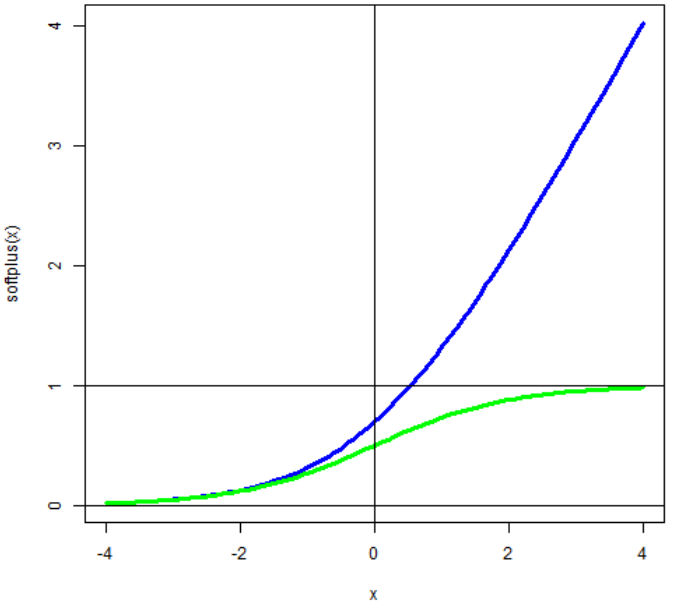
Softplus

Differentiable version of ReLU

Softplus

$$f(x) = \ln(1 + e^x), \quad f(0) = \ln(2)$$

$$f^{'} = \frac{1}{1 + e^{-x}} = \sigma(x), \quad f'(0) = 1/2$$

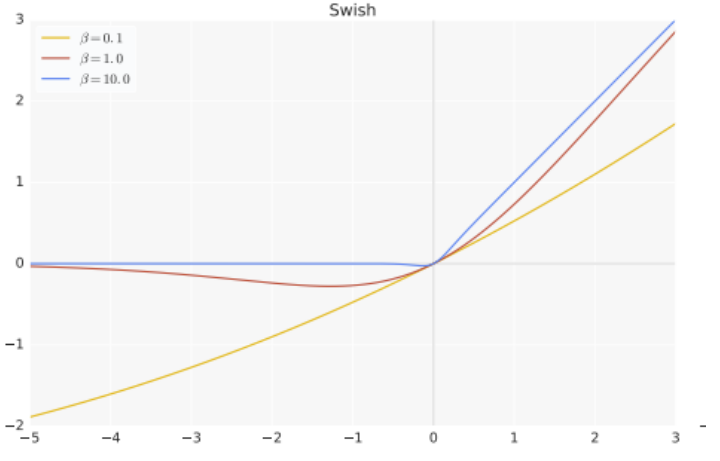


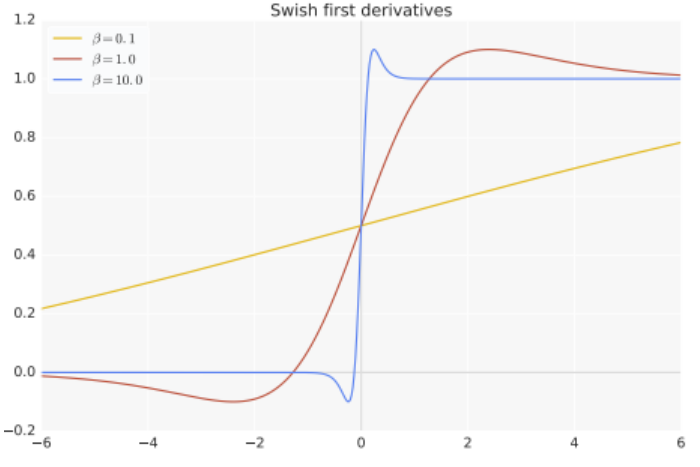
Swish

"Searching for Activation Functions" --- Ramachandran, Zoph and Le, Oct 2017

Ran extensive search for new activation functions. Search found:

$$f(x) = x\sigma(\beta x)$$
$$f^{'} = \beta f(x) + \sigma(\beta x)(1 - \beta f(x))$$





Swish

- Swish consistently outperformed ReLU and other activations on a wide range of tests
- Experiments used models and hyper-parameters that were designed for ReLU. Further improvements expected with models tuned for swish

Model	Top-1 Acc. (%)			Top-5 Acc. (%)		
LReLU	73.8	73.9	74.2	91.6	91.9	91.9
PReLU	74.6	74.7	74.7	92.4	92.3	92.3
Softplus	74.0	74.2	74.2	91.6	91.8	91.9
ELU	74.1	74.2	74.2	91.8	91.8	91.8
SELU	73.6	73.7	73.7	91.6	91.7	91.7
GELU	74.6	-	-	92.0	-	-
ReLU	73.5	73.6	73.8	91.4	91.5	91.6
Swish-1	74.6	74.7	74.7	92.1	92.0	92.0
Swish	74.9	74.9	75.2	92.3	92.4	92.4

Metrics

- **Cost**
  - Cost is a proxy, can move counter to accuracy
  - Should be able to make cost small with complex enough model
- **Accuracy**
  - Training error rate should go to zero with complex enough model
  - Want accuracy on both training and validation data sets
- **Magnitude of Gradient Vector**
  - Should be small at any minimum
- **Magnitude of Weights Vector**
  - Large weights might indicate overfitting