NPFL114, Lecture 02

Training Neural Networks

Milan Straka

Machine Learning Basics

We usually have a , which is assumed to consist of examples generated independently from a .

The goal of optimization is to match the training set as good as possible.

Machine Learning Basics

We usually have a , which is assumed to consist of examples generated independently from a .

The goal of optimization is to match the training set as good as possible.

However, the main goal of machine learning is to perform well on data, so called or . We typically estimate the generalization error using a of examples independent of the training set.

Machine Learning Basics

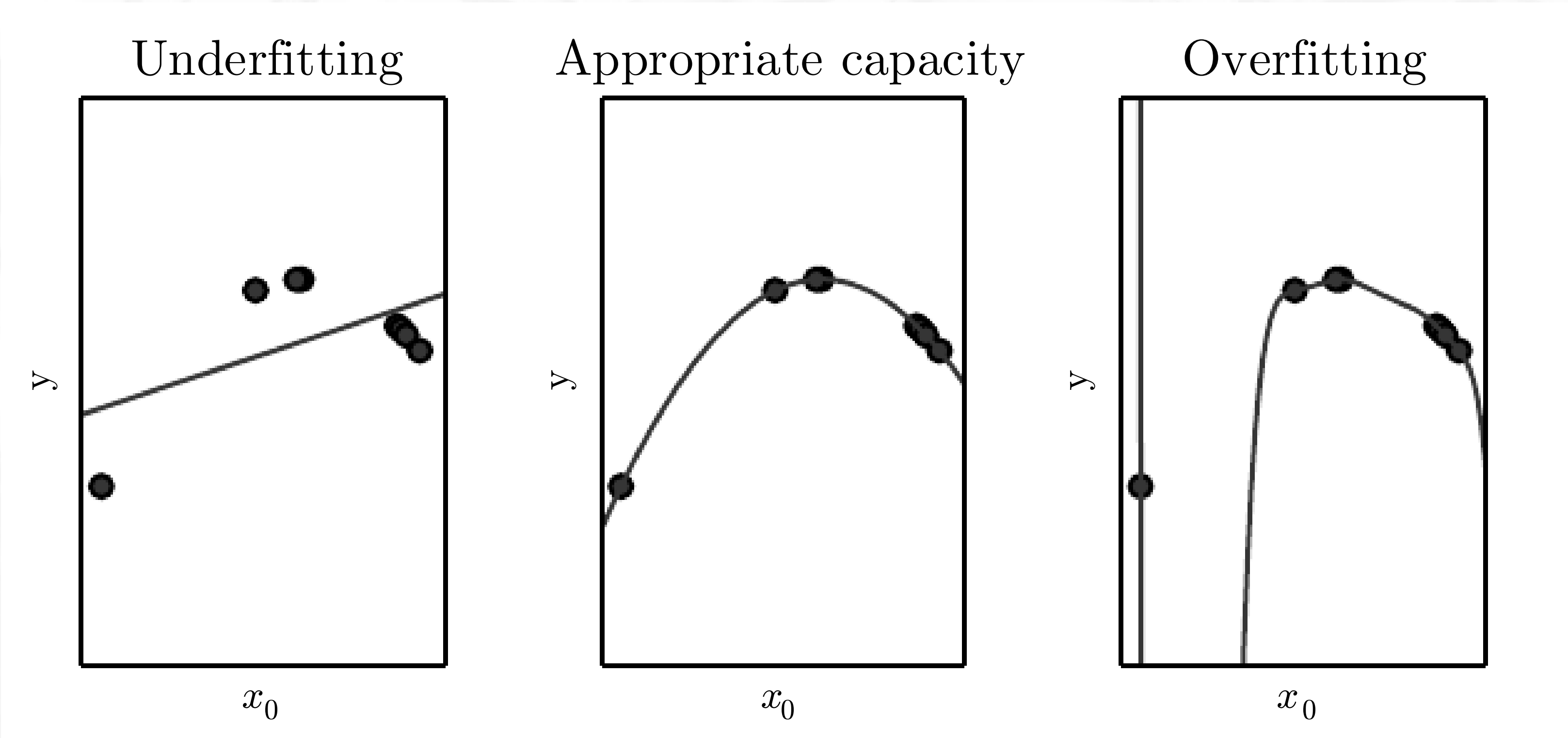
Challenges in machine learning:

-
-

Machine Learning Basics

Challenges in machine learning:

-
-

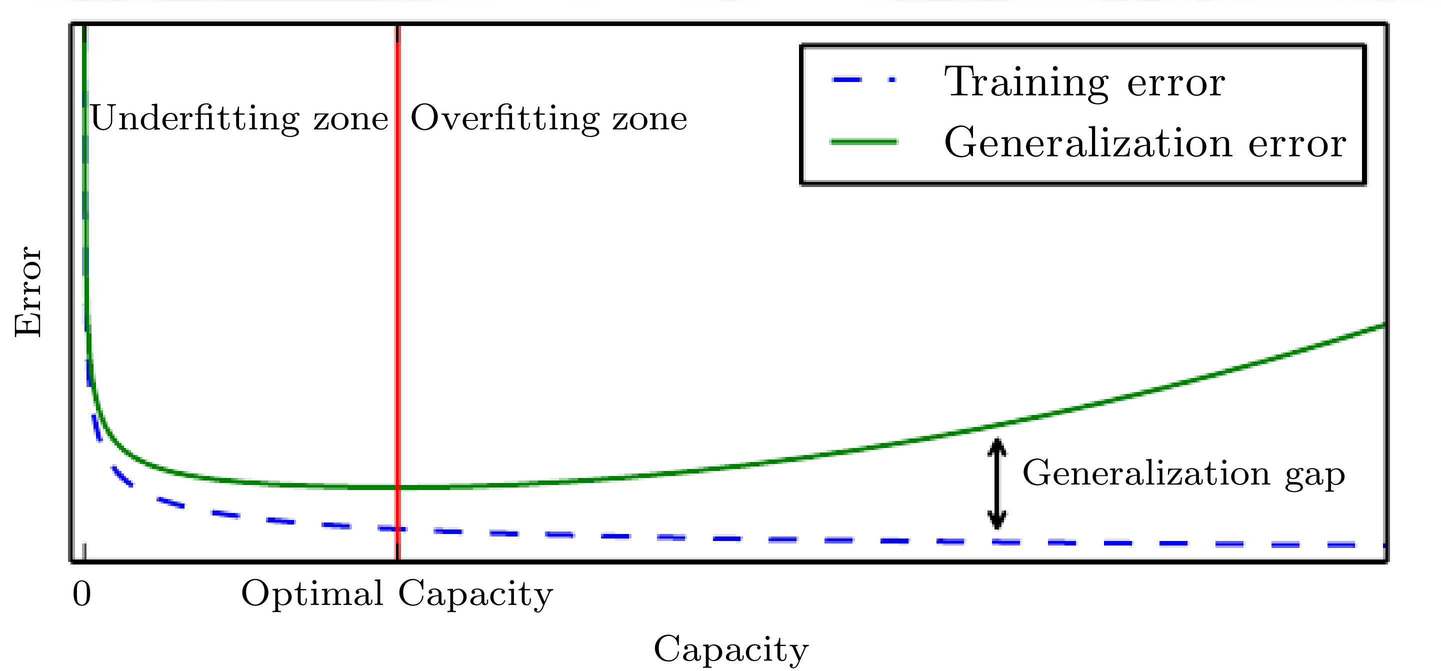


Machine Learning Basics

We control whether a model underfits or overfits by modifying its .
(representational capacity, effective capacity).

Machine Learning Basics

We control whether a model underfits or overfits by modifying its . (representational capacity, effective capacity).



The (Wolpert, 1996) states that averaging over all possible data distributions, every classification algorithm achieves same error when processing previously unseen examples. In a sense, no machine learning algorithm is universally better than any other.

Machine Learning Basics

Any change in the machine learning algorithm that is designed to reduce generalization error but not necessarily its training error is called

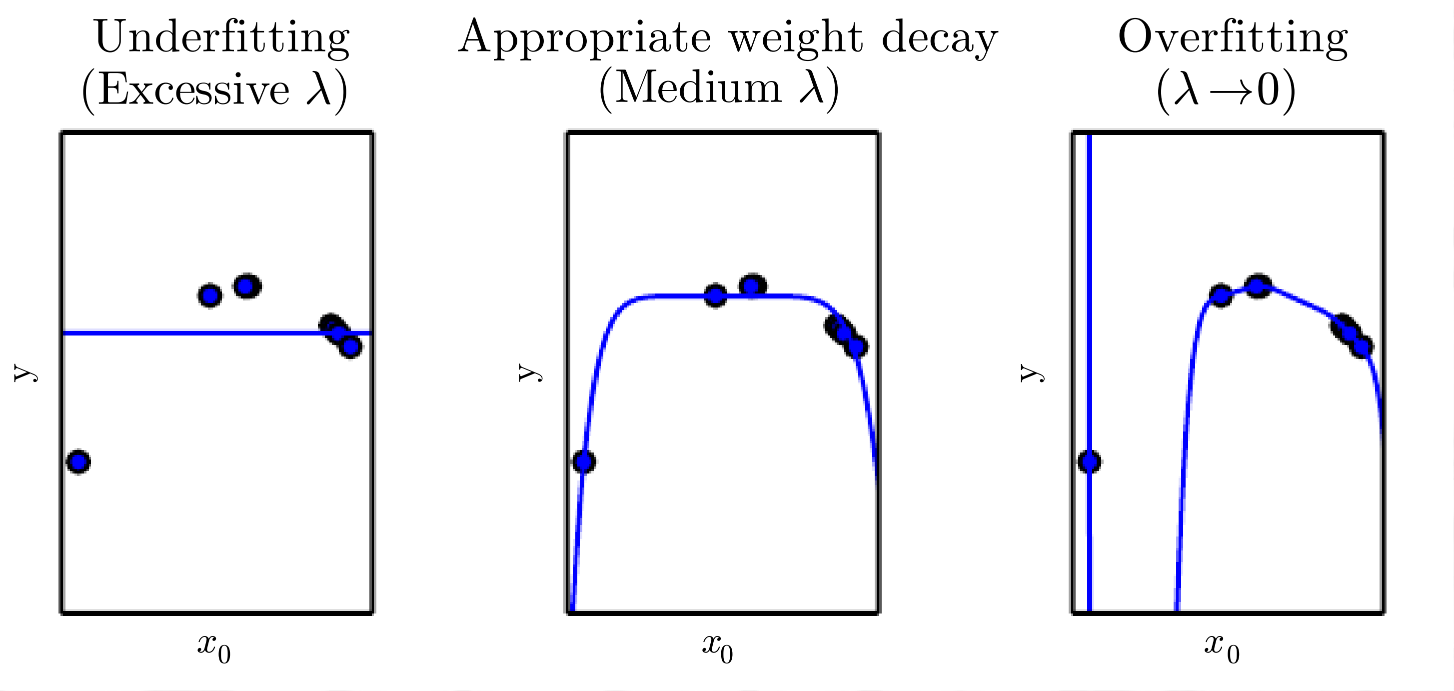
.

Machine Learning Basics

Any change in the machine learning algorithm that is designed to reduce generalization error but not necessarily its training error is called

.

L_2 regularization (also called weighted decay) penalizes models with large weights (i.e., penalty of $||\boldsymbol\theta||^2$).



Machine Learning Basics

are not adapted by learning algorithm itself.

Usually a or is used to estimate generalization error, allowing to update hyperparameters accordingly.

Loss Function

A model is usually trained in order to minimize a on the training data.

Loss Function

A model is usually trained in order to minimize a on the training data.

Assuming that a model computes $f(\boldsymbol{x}; \boldsymbol{\theta})$ using parameters $\boldsymbol{\theta}$,

is computed as

$$\sum_i \left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - y^{(i)} \right)^2.$$

Loss Function

A model is usually trained in order to minimize a   on the training data.

Assuming that a model computes $f(\boldsymbol{x}; \boldsymbol{\theta})$ using parameters $\boldsymbol{\theta}$,
          is computed as

$$\sum_i \left(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - y^{(i)} \right)^2 .$$

A common principle used to design loss functions is

.

Maximum Likelihood Estimation

Let $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ be training data drawn independently from data generating distribution p_{data}. We denote the empirical data distribution as $\hat{p}_{\text{data}}$.

Let $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ be a family of distributions. The

of parameters $\boldsymbol{\theta}$ is:

$$
\begin{aligned}
\boldsymbol{\theta}_{\text{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{X}; \boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \prod\nolimits_{i=1}^{m} p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \sum\nolimits_{i=1}^{m} -\log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [-\log p_{\text{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})] \\
&= \arg\min_{\boldsymbol{\theta}} H(\hat{p}_{\text{data}}, p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})) \\
&= \arg\min_{\boldsymbol{\theta}} D_{\text{KL}}(\hat{p}_{\text{data}} \| p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})) + H(\hat{p}_{\text{data}})
\end{aligned}
$$

Maximum Likelihood Estimation

Easily generalized to situations where our goal is predict y given $\boldsymbol{x}$.

$$\begin{aligned}
\boldsymbol{\theta}_{\mathrm{ML}} &= \arg\max_{\boldsymbol{\theta}} p_{\text{model}}(\mathbb{Y}|\mathbb{X};\boldsymbol{\theta}) \\
&= \arg\max_{\boldsymbol{\theta}} \prod\nolimits_{i=1}^{m} p_{\text{model}}(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \sum\nolimits_{i=1}^{m} -\log p_{\text{model}}(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta})
\end{aligned}$$

The resulting called , or or

.

Mean Square Error as MLE

Assume our goal is to perform a regression, i.e., to predict $p(y|\boldsymbol{x})$ for $y \in \mathbb{R}$.

Let $\hat{y}(\boldsymbol{x};\boldsymbol{\theta})$ gives the prediction of mean of y.

We define $p(y|\boldsymbol{x})$ as $\mathcal{N}(y; \hat{y}(\boldsymbol{x};\boldsymbol{\theta}), \sigma^2)$ for a given fixed σ. Then:

$$\begin{aligned}
\arg\max_{\boldsymbol{\theta}} p(y|\boldsymbol{x};\boldsymbol{\theta}) &= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^m -\log p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^m -\log \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(y^{(i)}-\hat{y}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}))^2}{2\sigma^2}} \\
&= -m \log \sigma - \frac{m}{2} \log 2\pi \\
&\quad - \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^m -\frac{||y - \hat{y}(\boldsymbol{x};\boldsymbol{\theta})||^2}{2\sigma^2} \\
&= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^m \frac{||y - \hat{y}(\boldsymbol{x};\boldsymbol{\theta})||^2}{2\sigma^2}
\end{aligned}$$

Gradient Descent

Let a model compute $f(\boldsymbol{x}; \boldsymbol{\theta})$ using parameters $\boldsymbol{\theta}$. In order to compute $J(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{(\boldsymbol{x}, y)} L(f(\boldsymbol{x}; \boldsymbol{\theta}), y)$, we may use :

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Gradient Descent

We use all training data to compute the $J(\boldsymbol{\theta})$.

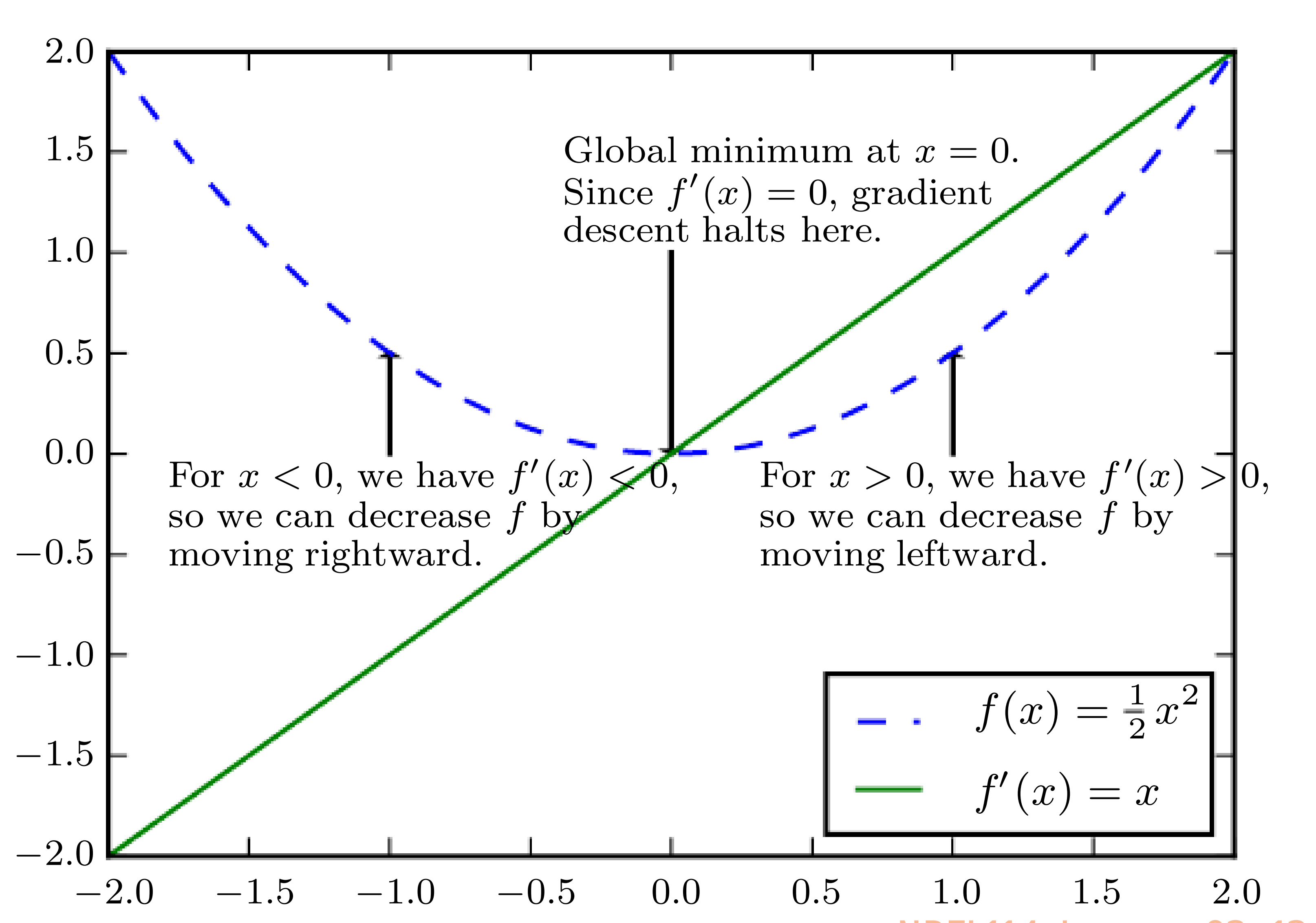
Online (or Stochastic) Gradient Descent

We estimate the expectation in $J(\boldsymbol{\theta})$ using a single randomly sampled example from the training data. Such estimate is unbiased, but very noisy.

Minibatch SGD

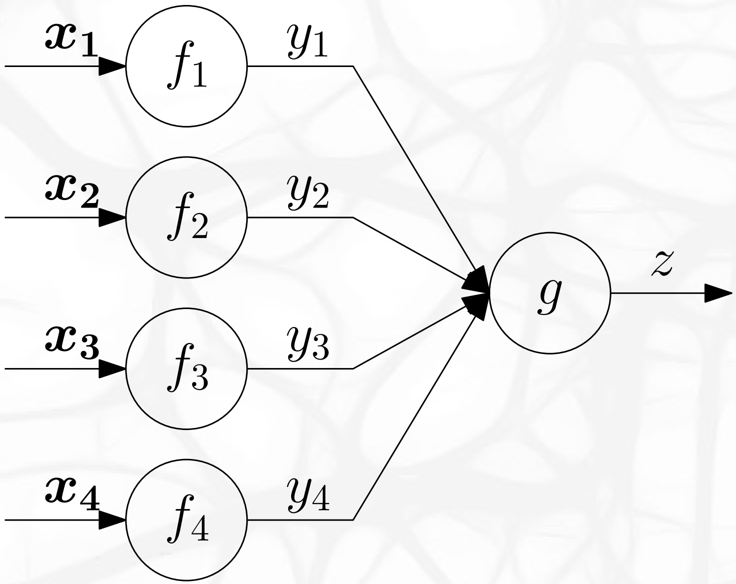
The minibatch SGD is a trade-off between gradient descent and SGD – the expectation in $J(\boldsymbol{\theta})$ is estimated using m random independent examples from the training data.

Gradient Descent



Backpropagation

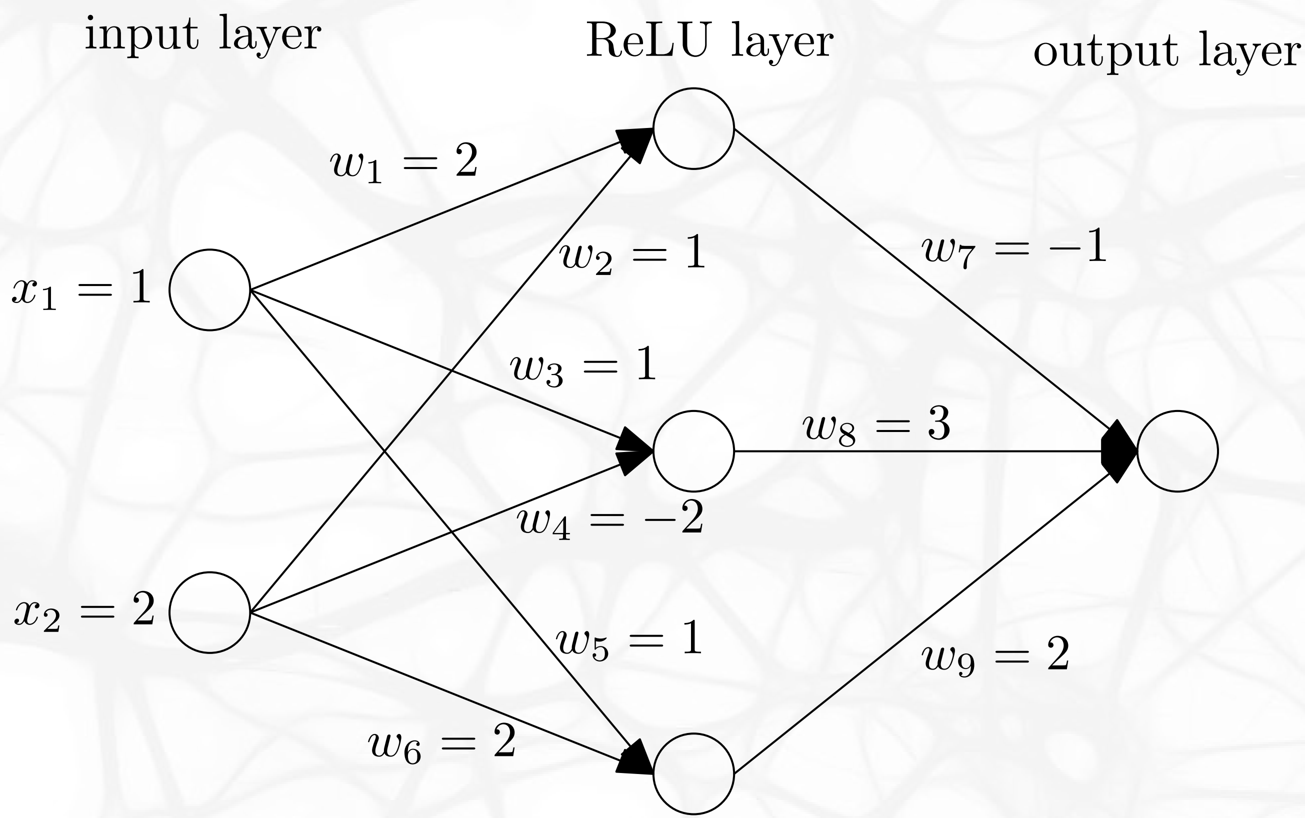
Assume we want to compute partial derivatives of a given loss function J and let $\frac{\partial J}{\partial z}$ be known.



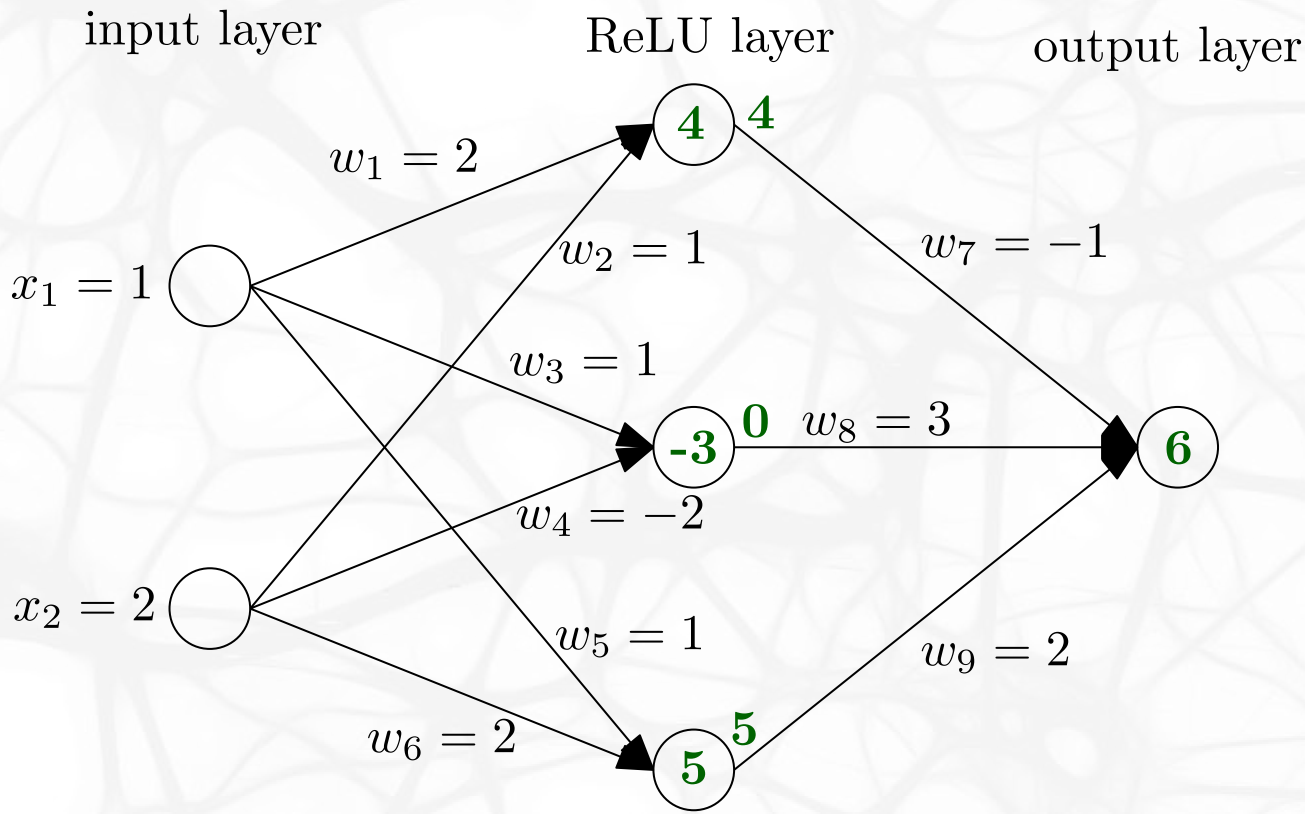
$$\frac{\partial J}{\partial y_i} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial y_i} = \frac{\partial J}{\partial z}\frac{\partial g(\boldsymbol{y})}{\partial y_i}$$

$$\frac{\partial J}{\partial \boldsymbol{x}_i} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial \boldsymbol{x}_i} = \frac{\partial J}{\partial z}\frac{\partial g(\boldsymbol{y})}{\partial y_i}\frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i}$$

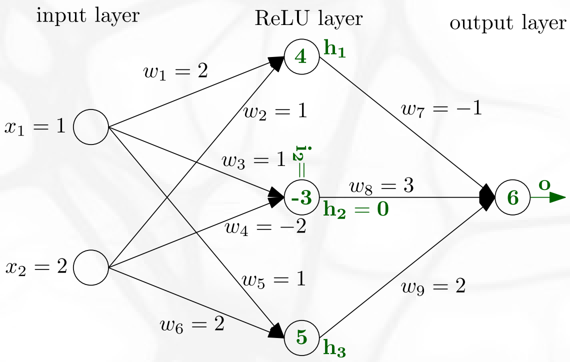
Backpropagation Example



Backpropagation Example



Backpropagation Example



$$\frac{\partial L}{\partial o} = 2(output - gold) = 6$$

$$\frac{\partial L}{\partial w_7} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial w_7} = \frac{\partial L}{\partial o} h_1 = 24$$

$$\frac{\partial L}{\partial w_8} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial w_8} = \frac{\partial L}{\partial o} h_2 = 0$$

$$\frac{\partial L}{\partial w_9} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial w_9} = \frac{\partial L}{\partial o} h_3 = 30$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial i_1}\frac{\partial i_1}{\partial w_1} = \frac{\partial L}{\partial i_1} x_1 = -6$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial i_1}\frac{\partial i_1}{\partial w_2} = \frac{\partial L}{\partial i_1} x_2 = -12$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial i_2}\frac{\partial i_2}{\partial w_3} = \frac{\partial L}{\partial i_2} x_1 = 0$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial i_2}\frac{\partial i_2}{\partial w_4} = \frac{\partial L}{\partial i_2} x_2 = 0$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial i_3}\frac{\partial i_3}{\partial w_5} = \frac{\partial L}{\partial i_3} x_1 = 12$$

$$\frac{\partial L}{\partial w_6} = \frac{\partial L}{\partial i_3}\frac{\partial i_3}{\partial w_6} = \frac{\partial L}{\partial i_3} x_2 = 24$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_1} = \frac{\partial L}{\partial o} w_7 = -6$$

$$\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_2} = \frac{\partial L}{\partial o} w_8 = 18$$

$$\frac{\partial L}{\partial h_3} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial h_3} = \frac{\partial L}{\partial o} w_9 = 12$$

$$\frac{\partial L}{\partial i_1} = \frac{\partial L}{\partial h_1}\frac{\partial h_1}{\partial i_1} = \frac{\partial L}{\partial h_1} 1 = -6$$

$$\frac{\partial L}{\partial i_2} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial i_2} = \frac{\partial L}{\partial h_2} 0 = 0$$

$$\frac{\partial L}{\partial i_3} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial i_3} = \frac{\partial L}{\partial h_3} 1 = 12$$

$$\frac{\partial L}{\partial x_1} = \sum_j \frac{\partial L}{\partial i_j}\frac{\partial i_j}{\partial x_1} = 0$$

$$\frac{\partial L}{\partial x_2} = \sum_j \frac{\partial L}{\partial i_j}\frac{\partial i_j}{\partial x_2} = 18$$

Backpropagation Algorithm

Forward Propagation

: Network with nodes $u^{(1)}, u^{(2)}, \ldots, u^{(n)}$ numbered in topological order, each node being computed as $u^{(i)} = f^{(i)}(\mathbb{A}^{(i)})$ for $\mathbb{A}^{(i)}$ composed of values of the predecessors $P(u^{(i)})$ of $u^{(i)}$.

: Value of $u^{(n)}$.

- For $i = 1, \ldots, n$:
  - $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} | j \in P(u^{(i)})\}$
  - $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$
- Return $u^{(n)}$

Backpropagation Algorithm

Simple Variant of Backpropagation

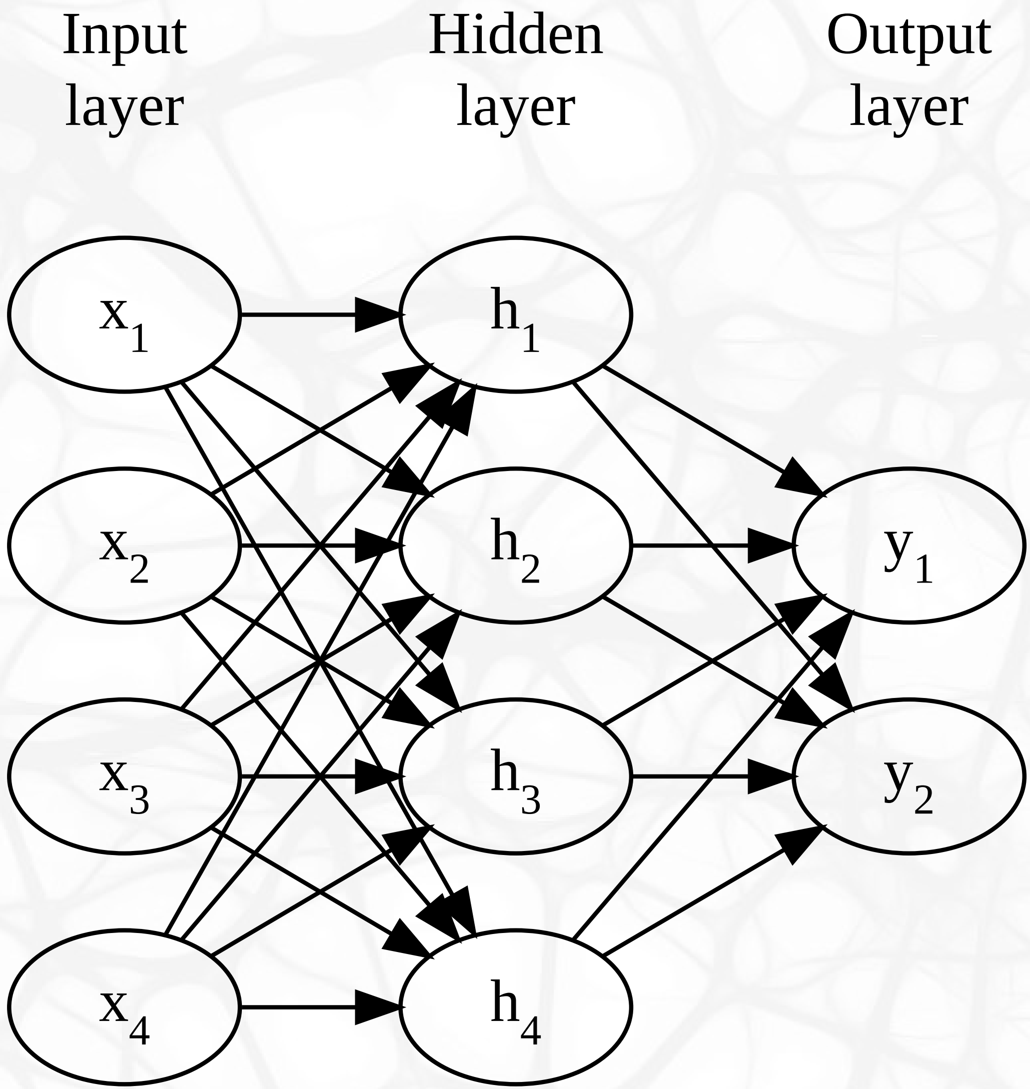
: The network as in the Forward propagation algorithm.

: Partial derivatives $g^{(i)} = \frac{\partial u^{(n)}}{\partial u^{(i)}}$ of $u^{(n)}$ with respect to all $u^{(i)}$.

- Run forward propagation to compute all $u^{(i)}$
- $g^{(n)} = 1$
- For $i = n-1, \ldots, 1$:
  - $g^{(i)} \leftarrow \sum_{j: i \in P(u^{(j)})} g^{(j)} \frac{\partial u^{(j)}}{\partial u^{(i)}}$
- Return $\boldsymbol{g}$

In practice, we do not usually represent network as a collection of scalar nodes; instead we represent it as a collection of tensor functions – most usually functions $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$. Then $\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is a Jacobian. However, the backpropagation algorithm is analogous.

Neural Network Architecture à la '80s



Neural Network Activation Functions

Hidden Layers Derivatives

- σ:

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$

- tanh:

$$\frac{d\tanh(x)}{dx} = 1 - \tanh(x)^2$$

- ReLU:

$$\frac{d\,\mathrm{ReLU}(x)}{dx} = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) Algorithm

: NN computing function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

: Learning rate α.

: Updated parameters $\boldsymbol{\theta}$.

- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{g}$

SGD With Momentum

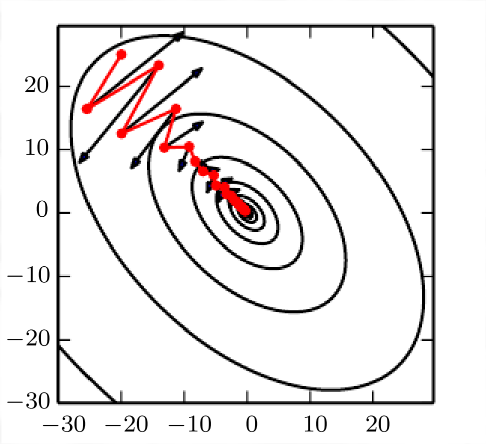
SGD With Momentum

: NN computing function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

: Learning rate α, momentum β.

: Updated parameters $\boldsymbol{\theta}$.

- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  - $\boldsymbol{v} \leftarrow \beta\boldsymbol{v} - \alpha\boldsymbol{g}$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$



SGD With Nesterov Momentum

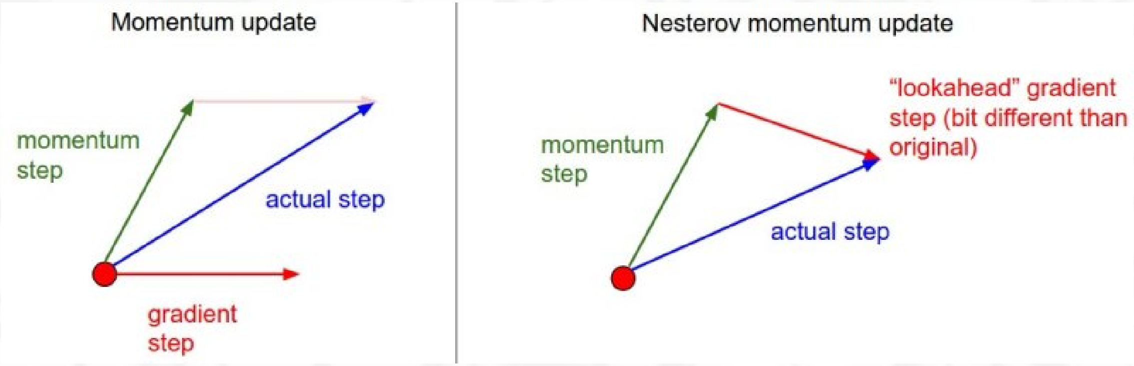
SGD With Nesterov Momentum

: NN computing function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

: Learning rate α, momentum β.

: Updated parameters $\boldsymbol{\theta}$.

- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \boldsymbol{v}$
  - $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  - $\boldsymbol{v} \leftarrow \beta \boldsymbol{v} - \alpha \boldsymbol{g}$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{g}$



AdaGrad (2011)

: NN computing function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

: Learning rate α, constant ε (usually 10^{-8}).

: Updated parameters $\boldsymbol{\theta}$.

- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  - $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g}^2$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\boldsymbol{r}+\varepsilon}} \boldsymbol{g}$

Algorithms with Adaptive Learning Rates

RMSProp (2012)

: NN computing function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

: Learning rate α, momentum β, constant ε (usually 10^{-8}).

: Updated parameters $\boldsymbol{\theta}$.

- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  - $\boldsymbol{r} \leftarrow \beta\boldsymbol{r} + (1-\beta)\boldsymbol{g}^2$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\boldsymbol{r}+\varepsilon}}\boldsymbol{g}$

Algorithms with Adaptive Learning Rates

Adam (2014)

: NN computing function $f(\boldsymbol{x}; \boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

: Learning rate α (default 0.001), constant ε (usually 10^{-8}).

: Momentum β_1 (default 0.9), momentum β_2 (default 0.999).

: Updated parameters $\boldsymbol{\theta}$.

- $\boldsymbol{s} \leftarrow 0, \boldsymbol{r} \leftarrow 0, t \leftarrow 0$
- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  - $t \leftarrow t + 1$
  - $\boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\boldsymbol{g}$
  - $\boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\boldsymbol{g}^2$
  - $\hat{\boldsymbol{s}} \leftarrow \boldsymbol{s}/(1 - \beta_1^t)$
  - $\hat{\boldsymbol{r}} \leftarrow \boldsymbol{r}/(1 - \beta_2^t)$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{\sqrt{\hat{\boldsymbol{r}} + \varepsilon}} \hat{\boldsymbol{s}}$

Algorithms with Adaptive Learning Rates

Adam (2014)

: NN computing function $f(\boldsymbol{x};\boldsymbol{\theta})$ with initial value of parameters $\boldsymbol{\theta}$.

: Learning rate α (default 0.001), constant ε (usually 10^{-8}).

: Momentum β_1 (default 0.9), momentum β_2 (default 0.999).

: Updated parameters $\boldsymbol{\theta}$.

- $\boldsymbol{s} \leftarrow 0, \boldsymbol{r} \leftarrow 0, t \leftarrow 0$
- Repeat until stopping criterion is met:
  - Sample a minibatch of m training examples $(\boldsymbol{x}^{(i)}, y^{(i)})$
  - $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$
  - $t \leftarrow t + 1$
  - $\boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\boldsymbol{g}$
  - $\boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\boldsymbol{g}^2$
  - $\alpha_t \leftarrow \alpha\sqrt{1 - \beta_2^t}/(1 - \beta_1^t)$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha_t}{\sqrt{\hat{\boldsymbol{r}}+\varepsilon}}\hat{\boldsymbol{s}}$

Adam Bias Correction

After t steps, we have

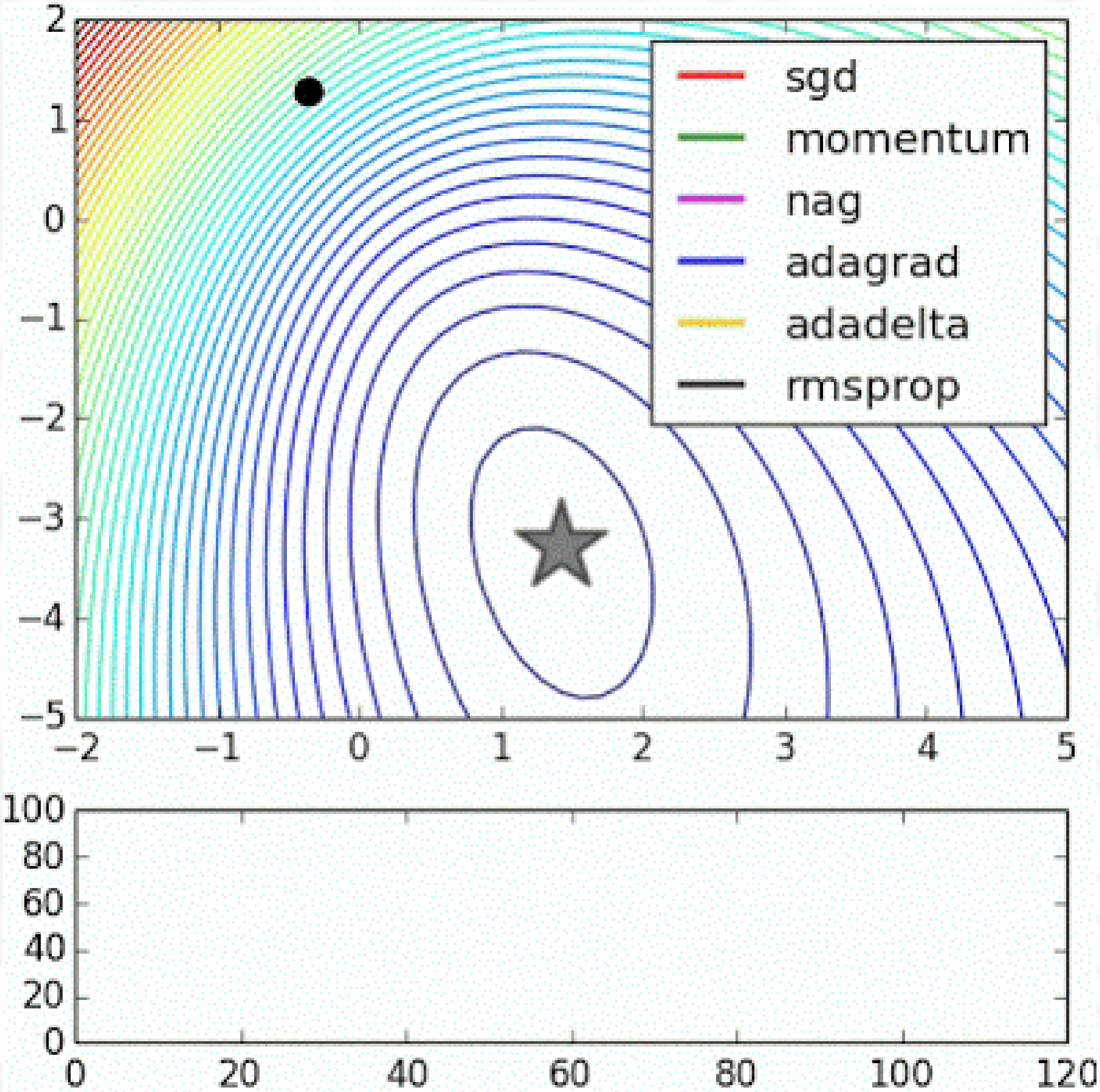
$$\boldsymbol{r}_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{g}_i^2 .$$

Assuming that the second moment $\mathbb{E}[\boldsymbol{g}_i^2]$ is stationary, we have

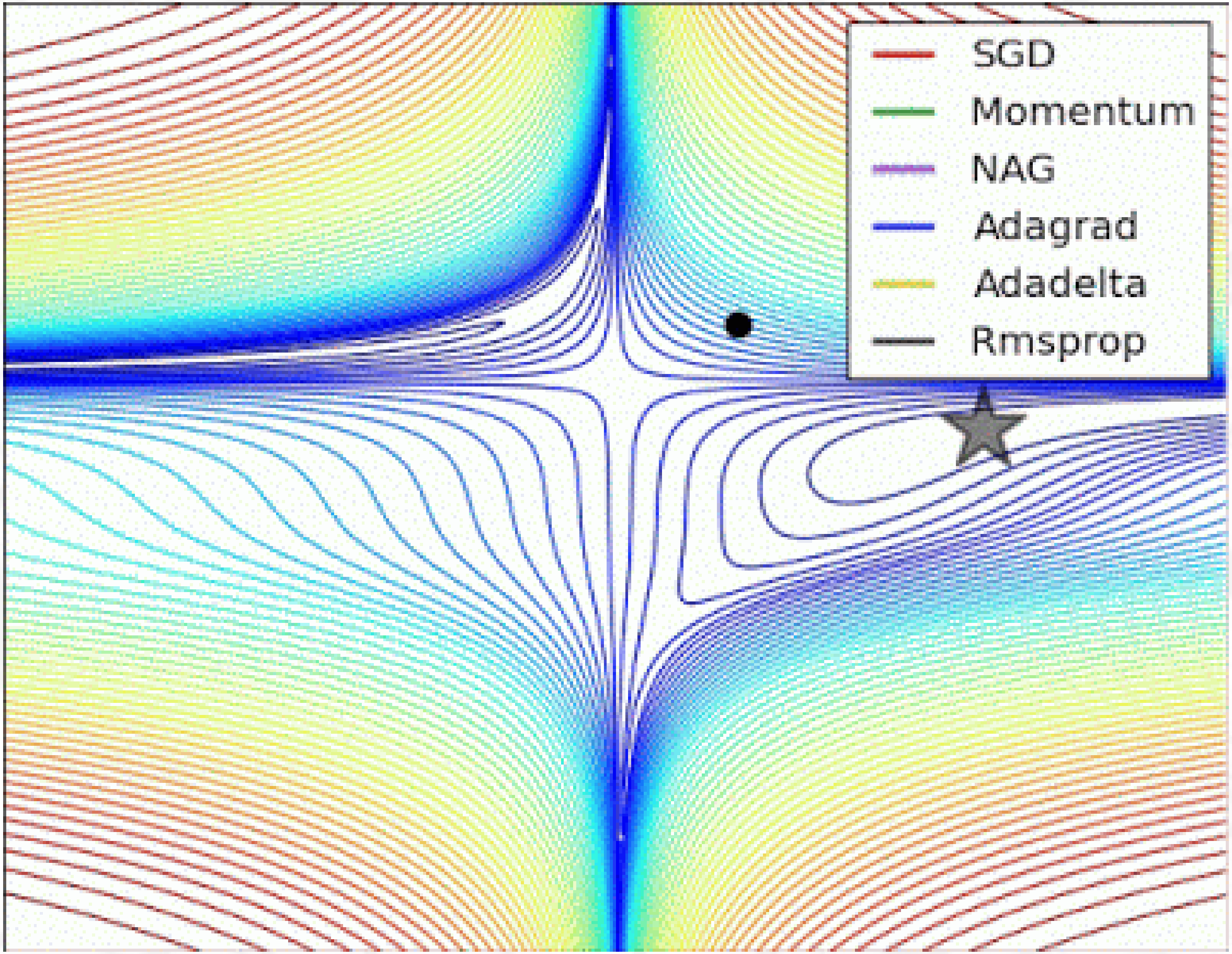
$$\begin{aligned}
\mathbb{E}[\boldsymbol{r}_t] &= \mathbb{E}\left[(1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{g}_i^2\right] \\
&= \mathbb{E}[\boldsymbol{g}_t^2] \cdot (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \\
&= \mathbb{E}[\boldsymbol{g}_t^2] \cdot (1 - \beta_2^t)
\end{aligned}$$

and analogously for correction of $\boldsymbol{s}$.

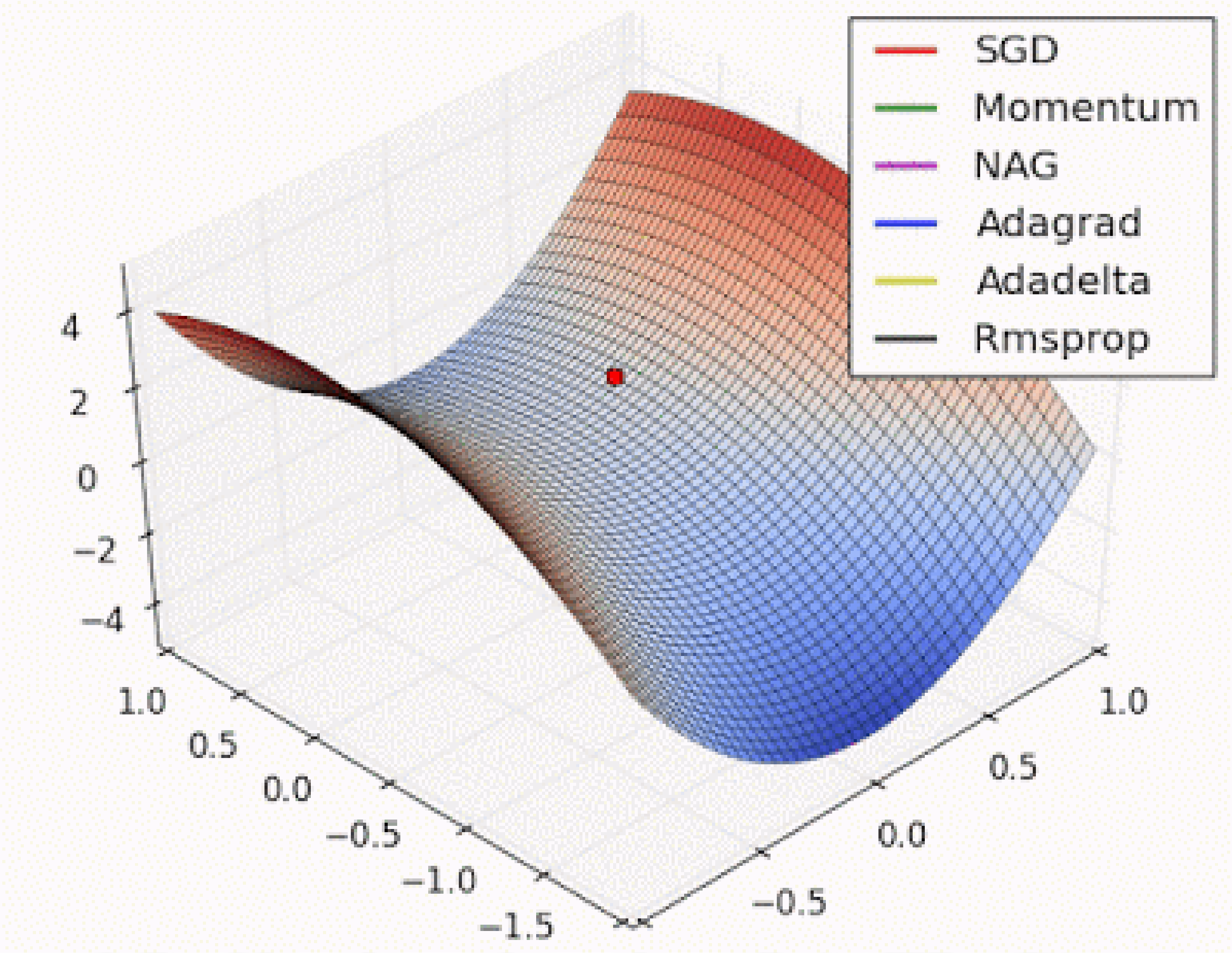
Adaptive Optimizers Animations



Adaptive Optimizers Animations



Adaptive Optimizers Animations



Adaptive Optimizers Animations