به نام یگانه معبود بخشنده مهربان

مبانی یادگیری ماشین

Machine Learning Foundations

گروه هوش مصنوعی، دانشکده مهندسی کامپیوتر، دانشگاه اصفهان

ترم اول سال تحصیلی ۰۲ - ۰۳

ارائه دهنده : پیمان ادیبی

شبکه های عصبی جلوسو

Feedforward Neural Networks

دسته بند شبکه عصبی

Neural Network for classification

Vector function with tunable parameters θ

$$\mathbf{f}(\cdot;\theta) : \mathbb{R}^N \rightarrow (0,1)^K$$

Sample s in dataset S:

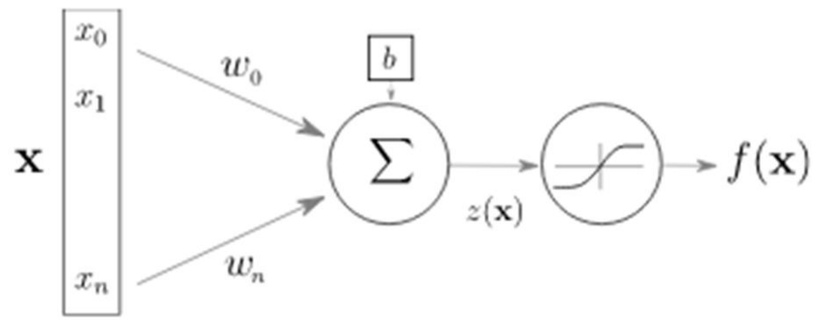
- input: $\mathbf{x}^s \in \mathbb{R}^N$
- expected output: $y^s \in [0, K-1]$

Output is a conditional probability distribution:

$$\mathbf{f}(\mathbf{x}^s;\theta)_c = P(Y = c|X = \mathbf{x}^s)$$

نورون مصنوعی

Artificial Neuron



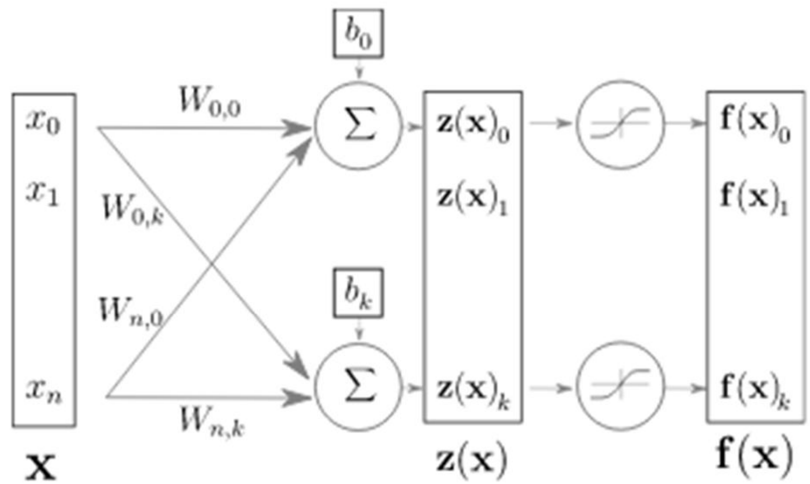
$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

- $\mathbf{x}, f(\mathbf{x})$ input and output
- $z(\mathbf{x})$ pre-activation
- $\mathbf{w}, b$ weights and bias
- g activation function

لایه ای از نورون های مصنوعی

Layer of Neurons

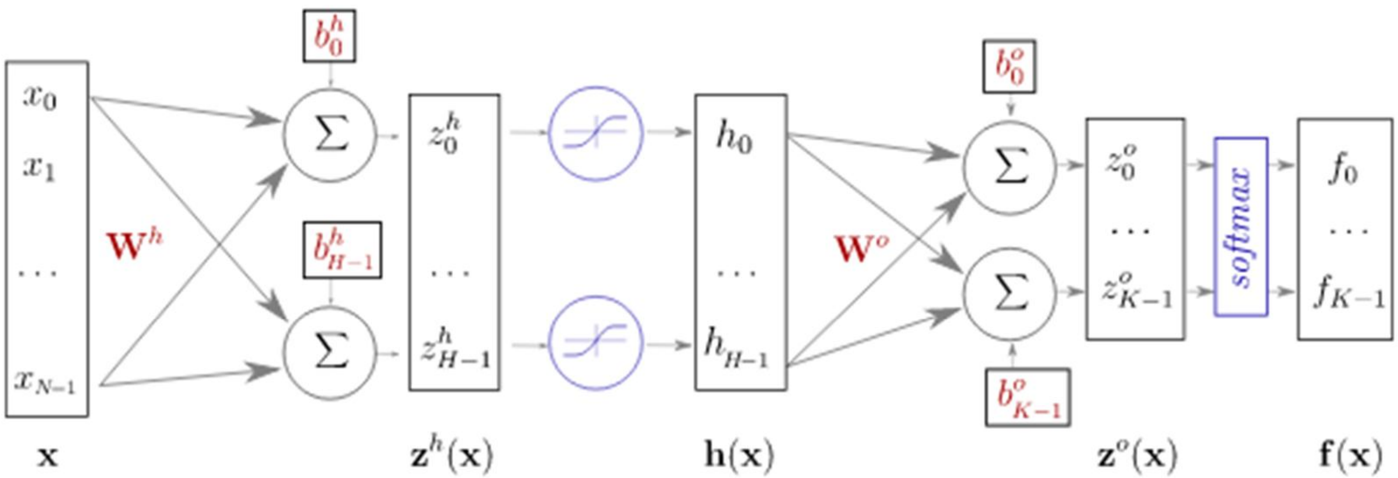


$$\mathbf{f}(\mathbf{x}) = g(\mathbf{z}(\mathbf{x})) = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- $\mathbf{W}, \mathbf{b}$ now matrix and vector

شبکه ای با یک لایه مخفی

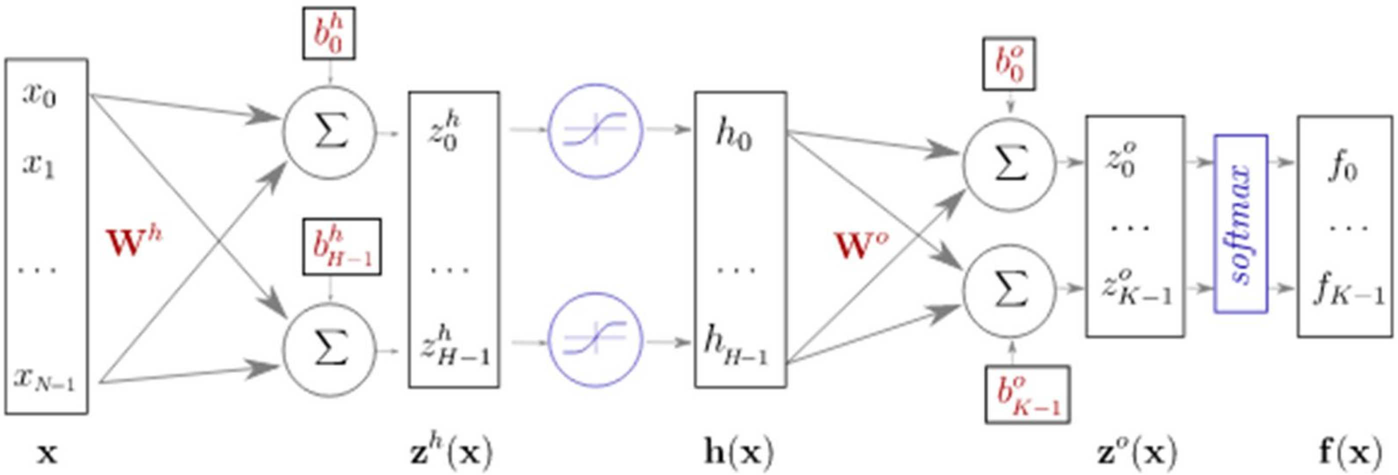
One Hidden Layer Network



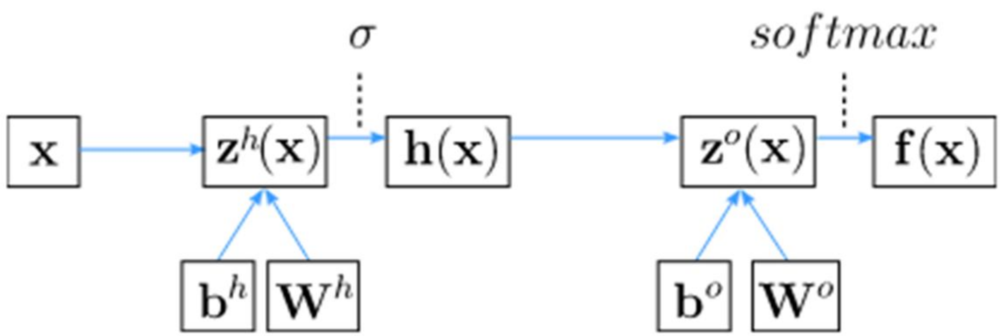
- $\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$
- $\mathbf{h}(\mathbf{x}) = g(\mathbf{z}^h(\mathbf{x})) = g(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h)$
- $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$
- $\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$

شبکه ای با یک لایه مخفی

One Hidden Layer Network

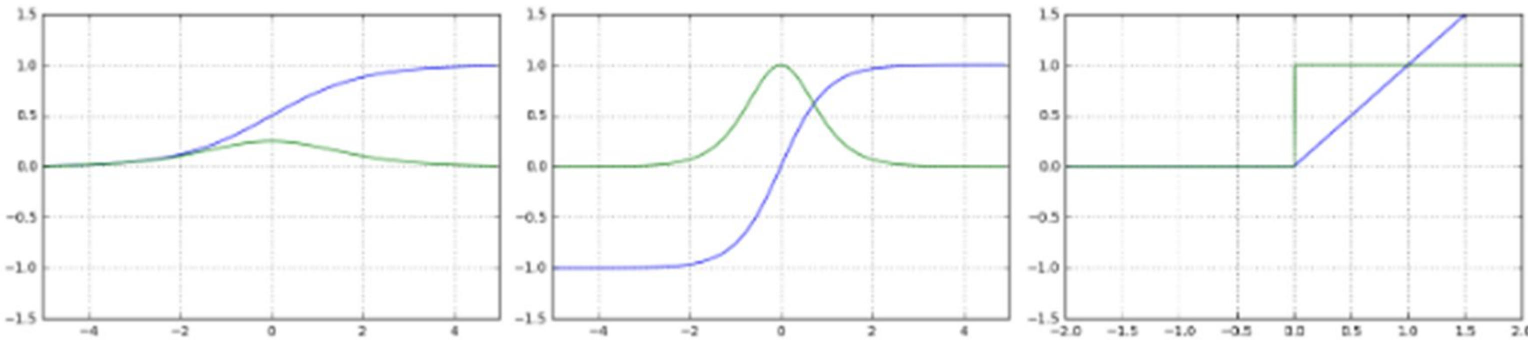


Alternate representation



توابع فعالیت متداول

Element-wise activation functions



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

$$\mathrm{sigm}'(x) = \mathrm{sigm}(x)(1-\mathrm{sigm}(x))$$

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\tanh'(x) = 1 - \tanh(x)^2$$

$$\mathrm{relu}(x) = \max(0, x)$$

$$\mathrm{relu}'(x) = 1_{x>0}$$

- blue: activation function
- green: derivative

تابع بیشینه نرم

Softmax function

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

$$\frac{\partial softmax(\mathbf{x})_i}{\partial x_j} = \begin{cases} softmax(\mathbf{x})_i \cdot (1 - softmax(\mathbf{x})_i) & i = j \\ -softmax(\mathbf{x})_i \cdot softmax(\mathbf{x})_j & i \neq j \end{cases}$$

- vector of values in (0, 1) that add up to 1
- $p(Y = c | X = \mathbf{x}) = \mathrm{softmax}(\mathbf{z}(\mathbf{x}))_c$
- the pre-activation vector $\mathbf{z}(\mathbf{x})$ is often called "the logits"

✓414

آموزش شبکه

Training the network

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s;\theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s;\theta)_{y^s}$$

example $\quad y^s = 3$

$$l(\mathbf{f}(\mathbf{x}^s;\theta), y^s) = l\left(\begin{bmatrix} f_0 \\ \cdots \\ f_3 \\ \cdots \\ f_{K-1} \end{bmatrix}, \begin{bmatrix} 0 \\ \cdots \\ 1 \\ \cdots \\ 0 \end{bmatrix} \right) = -\log\ f_3$$

آموزش شبکه

Training the network

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the **negative log likelihood** (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\mathbf{x}^s, y^s; \theta) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

The cost function is the negative likelihood of the model computed on the full training set (for i.i.d. samples):

$$L_S(\theta) = -\frac{1}{|S|} \sum_{s \in S} \log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s} + \lambda \Omega(\theta)$$

$\lambda \Omega(\theta) = \lambda(||W^h||^2 + ||W^o||^2)$ is an optional regularization term.

نزول در امتداد گرادیان تصادفی

Stochastic Gradient Descent

Initialize θ randomly

For E epochs perform:

- Randomly select a small batch of samples $(B \subset S)$
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)

Stop when reaching criterion:

- nll stops decreasing when computed on validation set

نزول در امتداد گرادیان تصادفی

Computing Gradients

Output Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}), y)}{\partial W^o_{i,j}}$

Output bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}), y)}{\partial b^o_i}$

Hidden Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}), y)}{\partial W^h_{i,j}}$

Hidden bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}), y)}{\partial b^h_i}$

- The network is a composition of differentiable modules
- We can apply the "chain rule"

نزول در امتداد گرادیان تصادفی

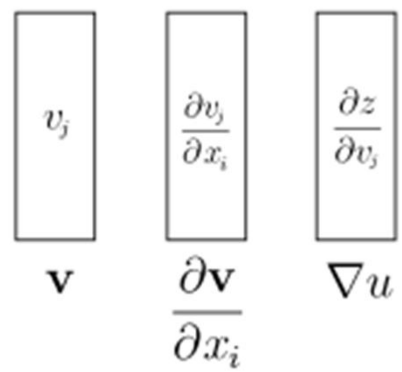
Chain rule

$$\boxed{\mathbf{x}} \longrightarrow \boxed{\mathbf{v}} \longrightarrow \boxed{z}$$

$$z = u(\mathbf{v}(\mathbf{x}))$$

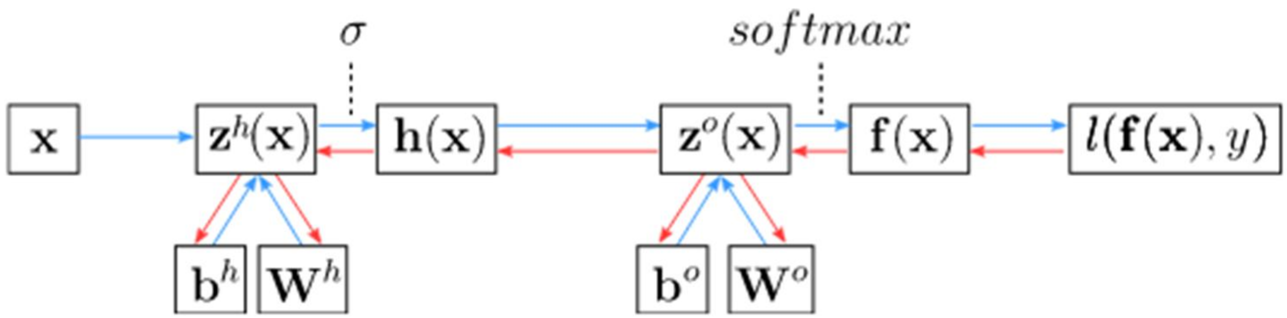
chain-rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial v_j}\frac{\partial v_j}{\partial x_i} = \nabla u . \frac{\partial \mathbf{v}}{\partial x_i}$$



الگوریتم پس انتشار خطا

Backpropagation



Compute partial derivatives of the loss

- $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{\partial - \log \mathbf{f}(\mathbf{x})_y}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y} = \frac{\partial \boldsymbol{l}}{\partial \mathbf{f}(\mathbf{x})_i}$
- $\frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_i} = ?$

الگوریتم پس انتشار خطا

$$\frac{\partial l}{\partial \mathbf{z}^o(\mathbf{x})_i} = \sum_j \frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_j} \frac{\partial \mathbf{f}(\mathbf{x})_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= \sum_j \frac{-1_{y=j}}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_j}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= -\frac{1}{\mathbf{f}(\mathbf{x})_y} \frac{\partial softmax(\mathbf{z}^o(\mathbf{x}))_y}{\partial \mathbf{z}^o(\mathbf{x})_i}$$

$$= \begin{cases} -\frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y (1 - softmax(\mathbf{z}^o(\mathbf{x}))_y) & \textbf{if } i = y \\ \frac{1}{\mathbf{f}(\mathbf{x})_y} softmax(\mathbf{z}^o(\mathbf{x}))_y softmax(\mathbf{z}^o(\mathbf{x}))_i & \textbf{if } i \neq y \end{cases}$$

$$= \begin{cases} -1 + \mathbf{f}(\mathbf{x})_y & \textbf{if } i = y \\ \mathbf{f}(\mathbf{x})_i & \textbf{if } i \neq y \end{cases}$$

Chain rule!

$$\frac{\partial l}{\partial \mathbf{f}(\mathbf{x})_i} = \frac{-1_{y=i}}{\mathbf{f}(\mathbf{x})_y}$$

$$\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o(\mathbf{x}))$$

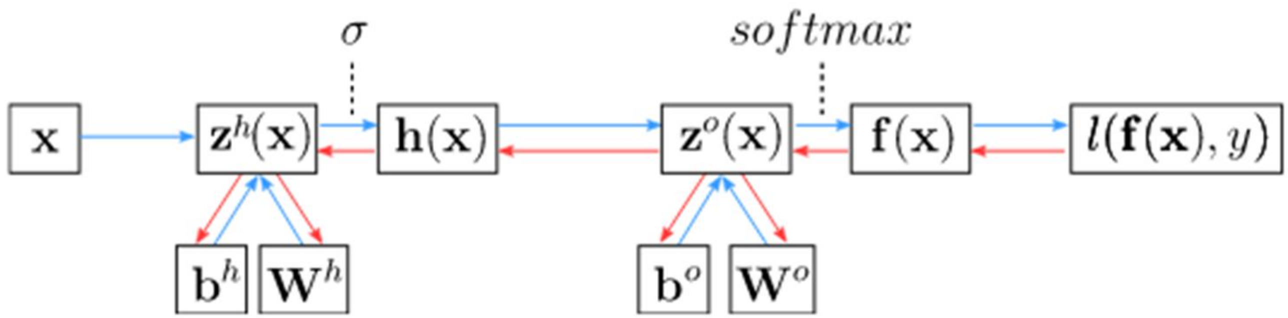
$$\nabla_{\mathbf{z}^o(\mathbf{x})} l(\mathbf{f}(\mathbf{x}), y) = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$$

$\mathbf{e}(y)$: one-hot encoding of y

$$\boxed{\mathbf{z}^o(\mathbf{x})} \xrightarrow{softmax} \boxed{\mathbf{f}(\mathbf{x})} \rightarrow \boxed{l(\mathbf{f}(\mathbf{x}), y)}$$

الگوریتم پس انتشار خطا

Backpropagation



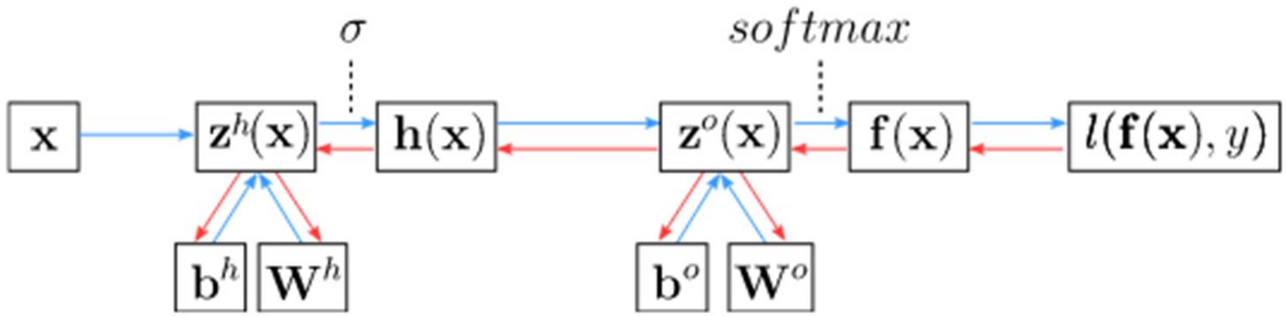
Gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$
- $\nabla_{\mathbf{b}^o} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

because $\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o \mathbf{h}(\mathbf{x}) + \mathbf{b}^o$ and then $\frac{\partial \mathbf{z}^o(\mathbf{x})_i}{\partial \mathbf{b}^o_j} = 1_{i=j}$

الگوریتم پس انتشار خطا

Backpropagation



Partial derivatives related to $\mathbf{W}^o$

- $\frac{\partial \boldsymbol{l}}{\partial W_{i,j}^o} = \sum_k \frac{\partial \boldsymbol{l}}{\partial \mathbf{z}^o(\mathbf{x})_k} \frac{\partial \mathbf{z}^o(\mathbf{x})_k}{\partial W_{i,j}^o}$
- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = (\mathbf{f}(\mathbf{x}) - \mathbf{e}(y)).\, \mathbf{h}(\mathbf{x})^\top$

الگوریتم پس انتشار خطا

Backprop gradients

Compute activation gradients

- $\nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} = \mathbf{f}(\mathbf{x}) - \mathbf{e}(y)$

Compute layer params gradients

- $\nabla_{\mathbf{W}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l} \cdot \mathbf{h}(\mathbf{x})^\top$
- $\nabla_{\mathbf{b}^o} \boldsymbol{l} = \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$

Compute prev layer activation gradients

- $\nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} = \mathbf{W}^{o\top} \nabla_{\mathbf{z}^o(\mathbf{x})} \boldsymbol{l}$
- $\nabla_{\mathbf{z}^h(\mathbf{x})} \boldsymbol{l} = \nabla_{\mathbf{h}(\mathbf{x})} \boldsymbol{l} \odot \sigma'(\mathbf{z}^{\mathbf{h}}(\mathbf{x}))$

✓450

شبکه های عصبی:

توابع اتلاف، مقداردهی اولیه، ترفندهای یادگیری

توابع اتلاف

Discrete output (classification)

- Binary classification: $y \in [0, 1]$
  - $Y|X = \mathbf{x} \sim Bernoulli(b = f(\mathbf{x}; \theta))$
  - output function: $logistic(x) = \frac{1}{1+e^{-x}}$
  - loss function: binary cross-entropy
- Multiclass classification: $y \in [0, K-1]$
  - $Y|X = \mathbf{x} \sim Multinoulli(\mathbf{p} = \mathbf{f}(\mathbf{x}; \theta))$
  - output function: $softmax$
  - loss function: categorical cross-entropy

توابع اتلاف

Continuous output (regression)

- Continuous output: $\mathbf{y} \in \mathbb{R}^n$
  - $Y|X = \mathbf{x} \sim \mathcal{N}(\mu = \mathbf{f}(\mathbf{x};\theta), \sigma^2\mathbf{I})$
  - output function: Identity
  - loss function: square loss
- Heteroschedastic if $\mathbf{f}(\mathbf{x};\theta)$ predicts both μ and σ^2
- Mixture Density Network (multimodal output)
  - $Y|X = \mathbf{x} \sim GMM_{\mathbf{x}}$
  - $\mathbf{f}(\mathbf{x};\theta)$ predicts all the parameters: the means, covariance matrices and mixture weights

مقداردهی اولیه، نرمال سازی

Initialization and normalization

- Input data should be normalized to have approx. same range:
  - standardization or quantile normalization
- Initializing W^h and W^o:
  - Zero is a saddle point: no gradient, no learning
  - Constant init: hidden units collapse by symmetry
  - Solution: random init, ex: $w \sim \mathcal{N}(0, 0.01)$
  - Better inits: Xavier Glorot and Kaming He & orthogonal
- Biases can (should) be initialized to zero

✓450

نرخ یادگیری

SGD learning rate

- Very sensitive:
  - Too high $\rightarrow$ early plateau or even divergence
  - Too low $\rightarrow$ slow convergence
  - Try a large value first: $\eta = 0.1$ or even $\eta = 1$
  - Divide by 10 and retry in case of divergence
- Large constant LR prevents final convergence
  - multiply η_t by $\beta < 1$ after each update
  - or monitor validation loss and divide η_t by 2 or 10 when no progress
  - See ReduceLROnPlateau in Keras

گشتاور و شتاب دهنده SGD

Momentum

Accumulate gradients across successive updates:

$$m_t = \gamma m_{t-1} + \eta \nabla_\theta L_{B_t}(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$

γ is typically set to 0.9

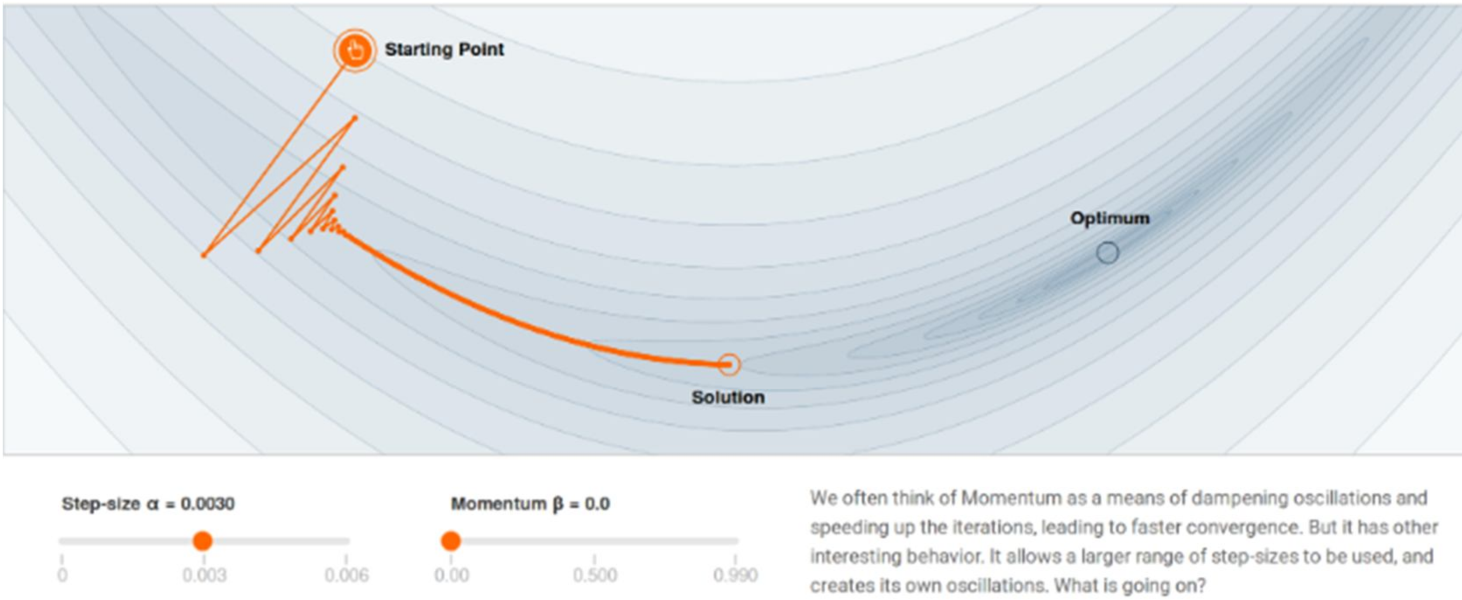
Larger updates in directions where the gradient sign is constant to accelerate in low curvature areas

Nesterov accelerated gradient

$$m_t = \gamma m_{t-1} + \eta \nabla_\theta L_{B_t}(\theta_{t-1} - \gamma m_{t-1})$$
$$\theta_t = \theta_{t-1} - m_t$$

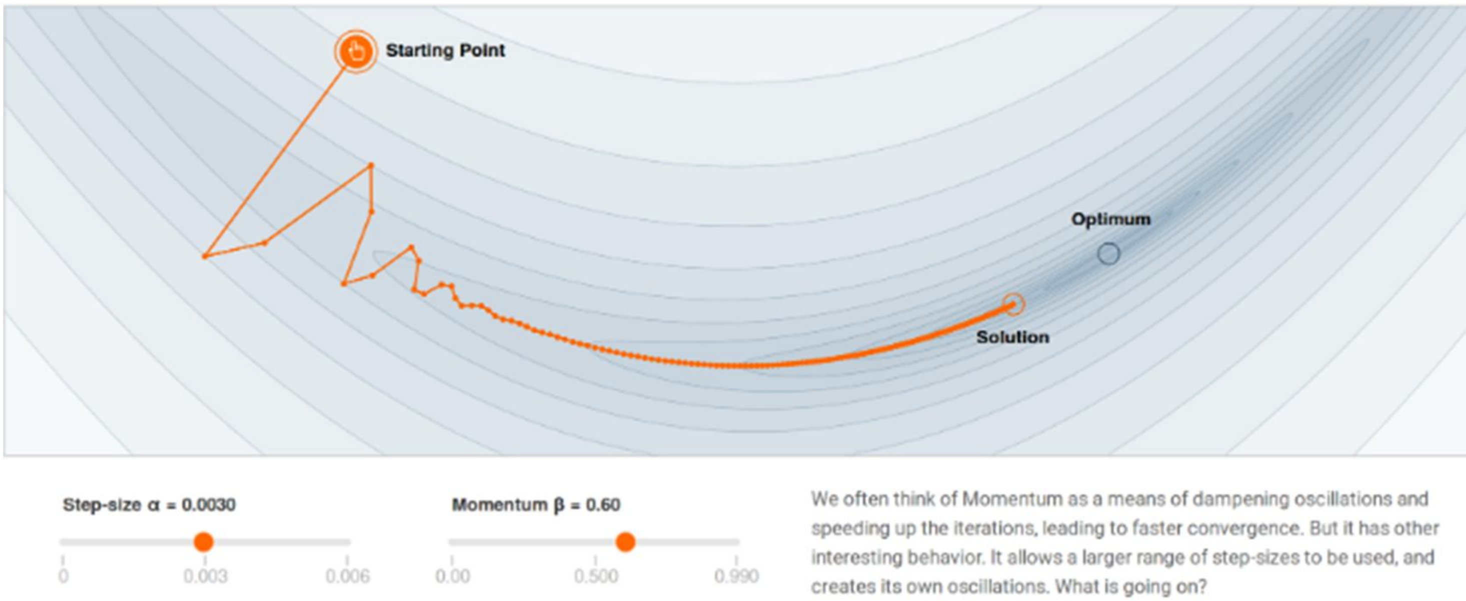
Better at handling changes in gradient direction.

گشتاور - مثال



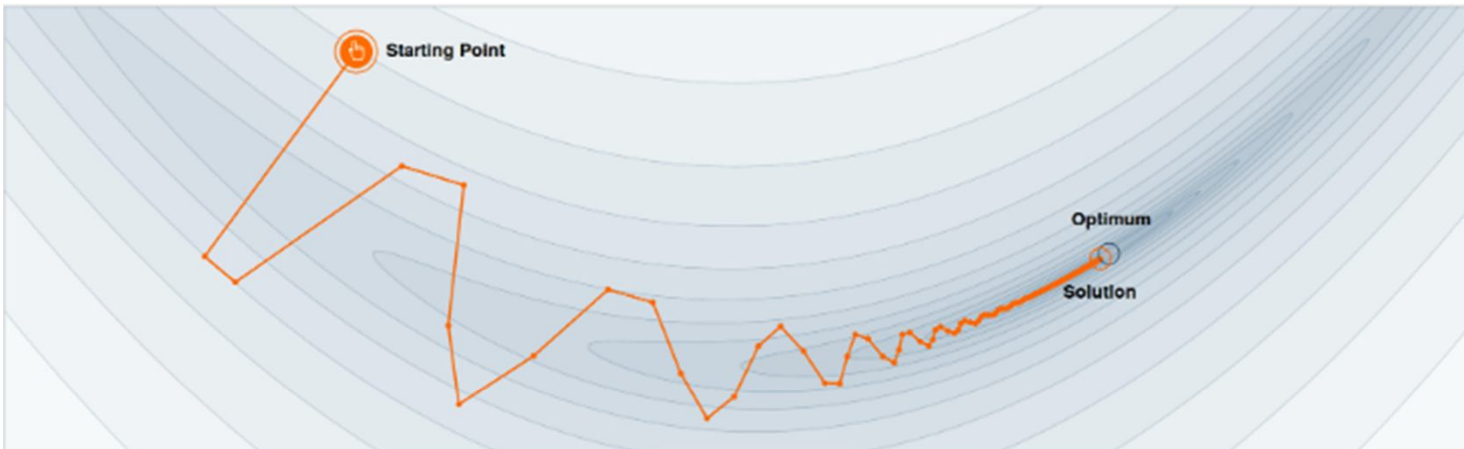
[Why Momentum Really Works]

گشتاور - مثال



[Why Momentum Really Works]

گشتاور - مثال



Step-size α = 0.0030

0 0.003 0.006

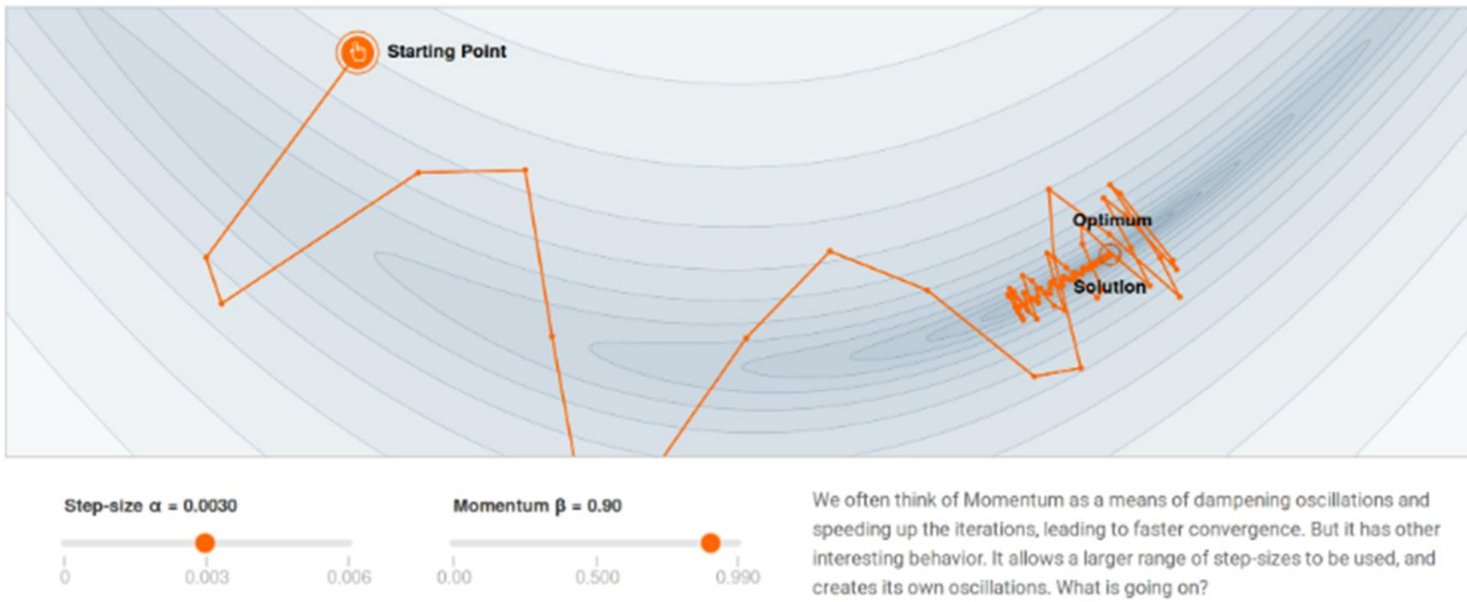
Momentum β = 0.80

0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

[Why Momentum Really Works]

گشتاور - مثال



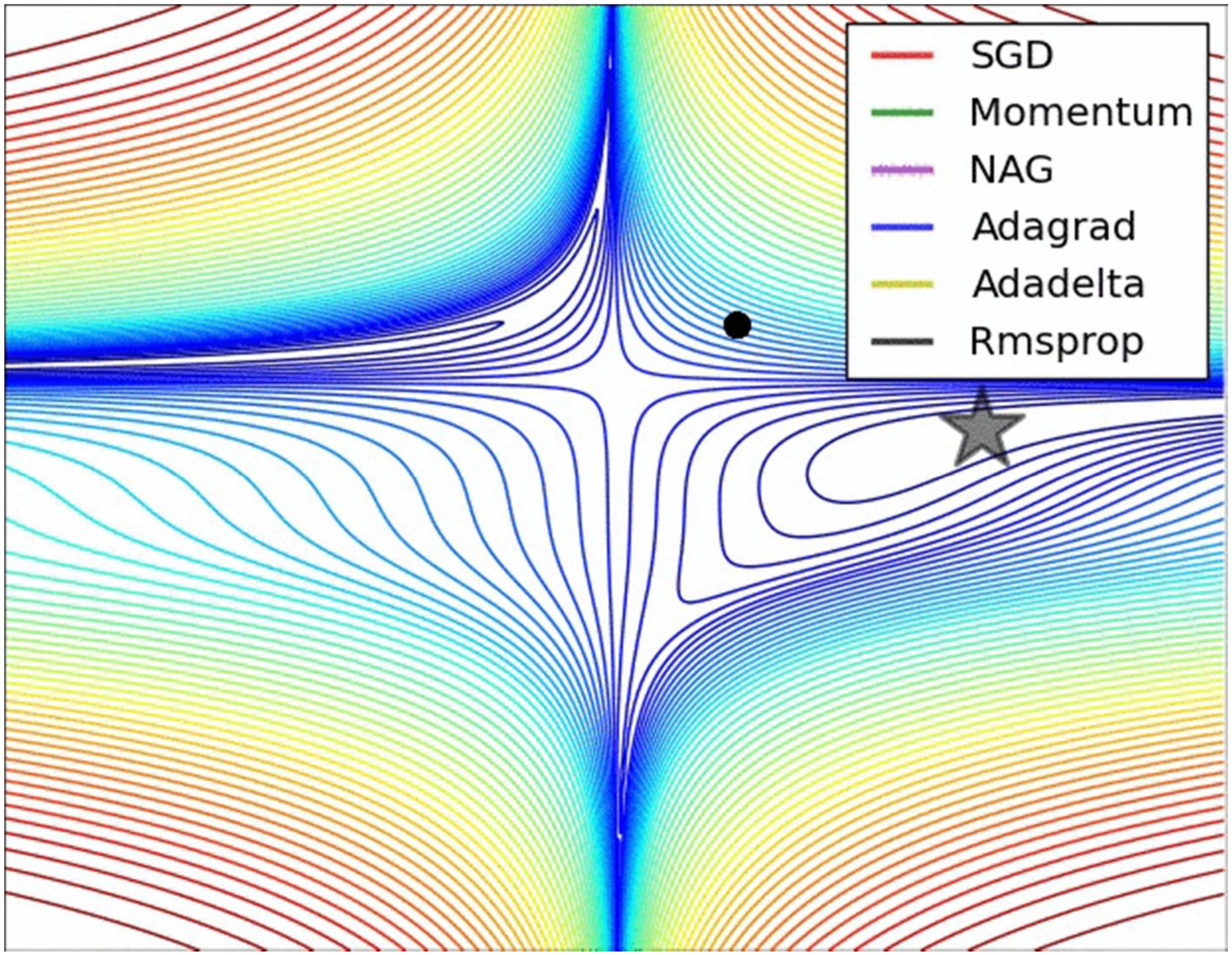
[Why Momentum Really Works]

انواع بهینه سازها

Alternative optimizers

- SGD (with Nesterov momentum)
  - Simple to implement
  - Very sensitive to initial value of η
  - Need learning rate scheduling
- Adam: adaptive learning rate scale for each param
  - Global η set to 3e-4 often works well enough
  - Good default choice of optimizer (often)
- But well-tuned SGD with LR scheduling can generalize better than Adam (with naive l2 reg)...
- Promising stochastic second order methods: K-FAC and Shampoo can be used to accelerate training of very large models.

بهینه سازها در اطراف یک نقطه زین اسبی



✓380