EECS 498/598: Deep Learning
Lecture 2. Deep Neural Network, Backpropagation, Optimization and Regularization

Honglak Lee

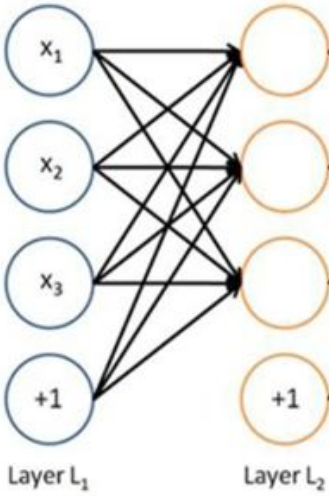
1/18/2019

Outline

- Backpropagation
- Optimization
- Regularization

Neural network

- Neural network: similar to running several logistic regressions (or other nonlinear mappings) for individual hidden units (neurons)
- If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs

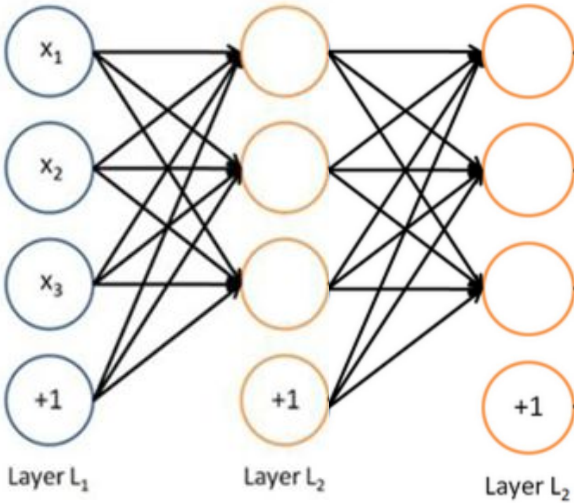


But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

Slide Credit: Yoshua Bengio

Neural network

- And we can do this compositionally over multiple layers...

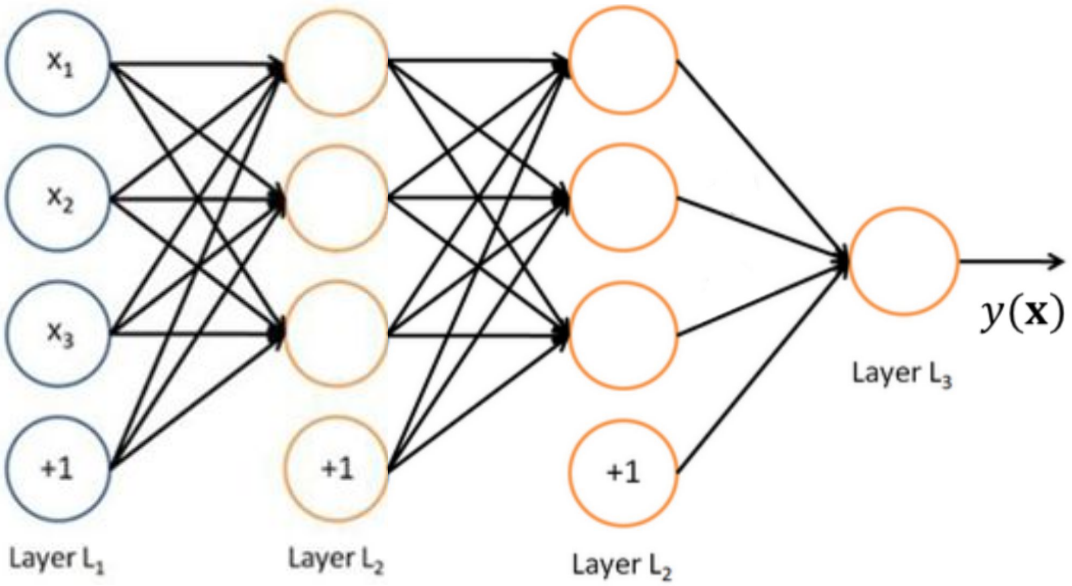


But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

Slide Credit: Yoshua Bengio

Neural network

- … which we can feed into another logistic regression function (or nonlinear function) as the final output.

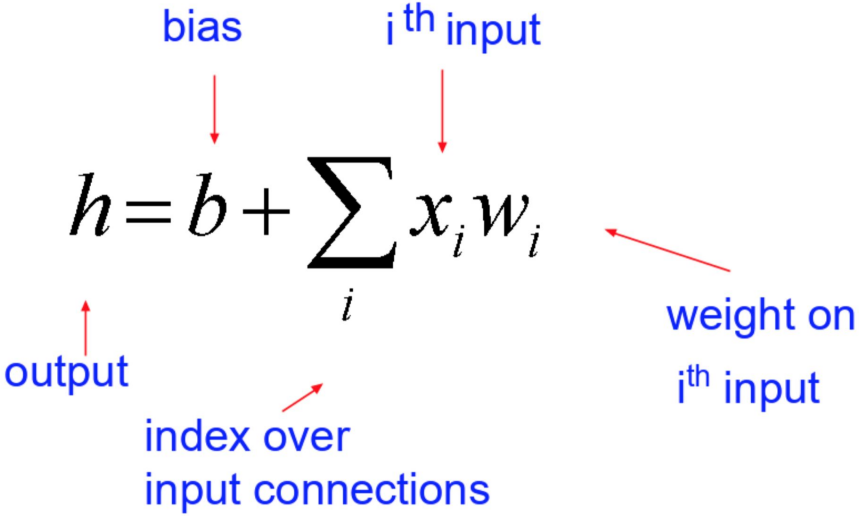


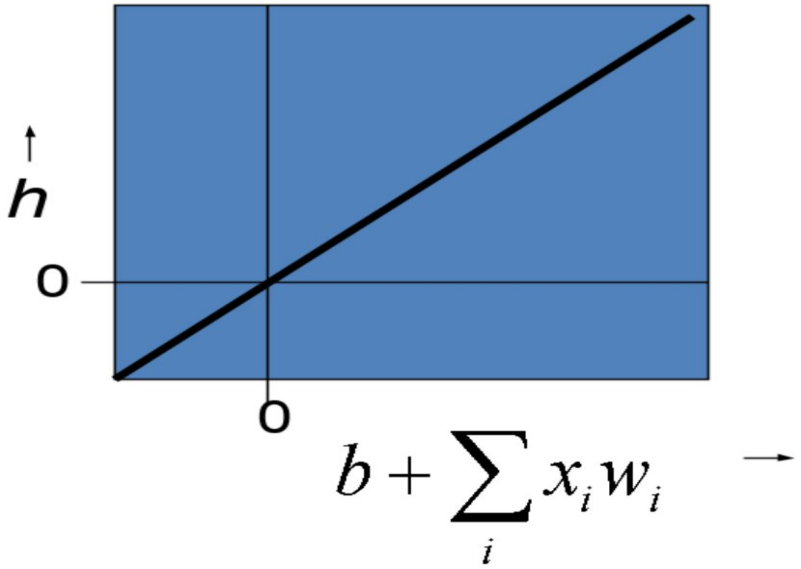
and it is the training criterion that will decide what those intermediate target variables should be, so as to make a good job of predicting the targets for the next layer, etc.

Slide Credit: Yoshua Bengio

Types of Neurons: Linear Neurons

- These are simple but computationally limited
  - If we can make them learn we may get insight into more complicated neurons.





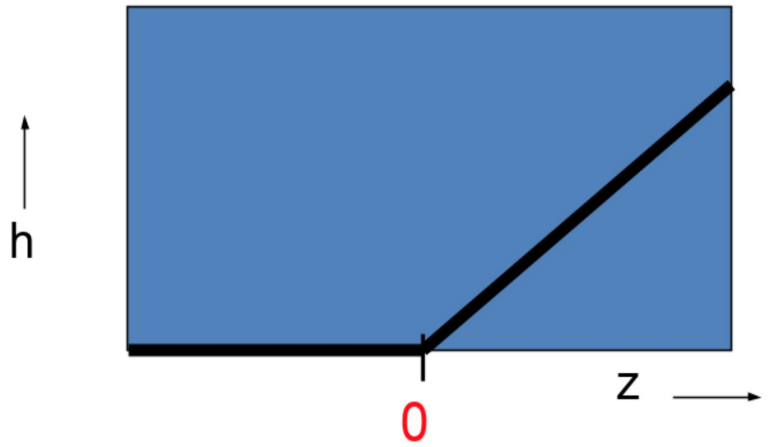
Slide Credit: Geoff Hinton

Rectified Linear (linear threshold) Neurons

- They compute a linear weighted sum of their inputs.
- The output is a non-linear function of the total input.

$$z = b + \sum_i x_i w_i$$

$$h = \begin{cases} z \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$$



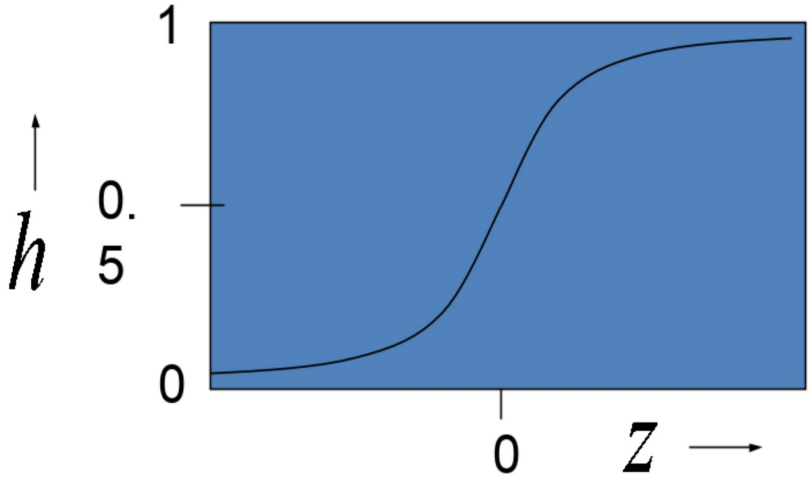
Slide Credit: Geoff Hinton

Sigmoid (logistic) neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - They have nice derivatives which make learning easy

$$z = b + \sum_i x_i w_i$$

$$h = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$



Slide Credit: Geoff Hinton

Softmax neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - The outputs sum up to 1 (useful for classification problems)
  - They have nice derivatives which make learning easy

$$z_k = b_k + \sum_i x_i w_i^k$$

$$h_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

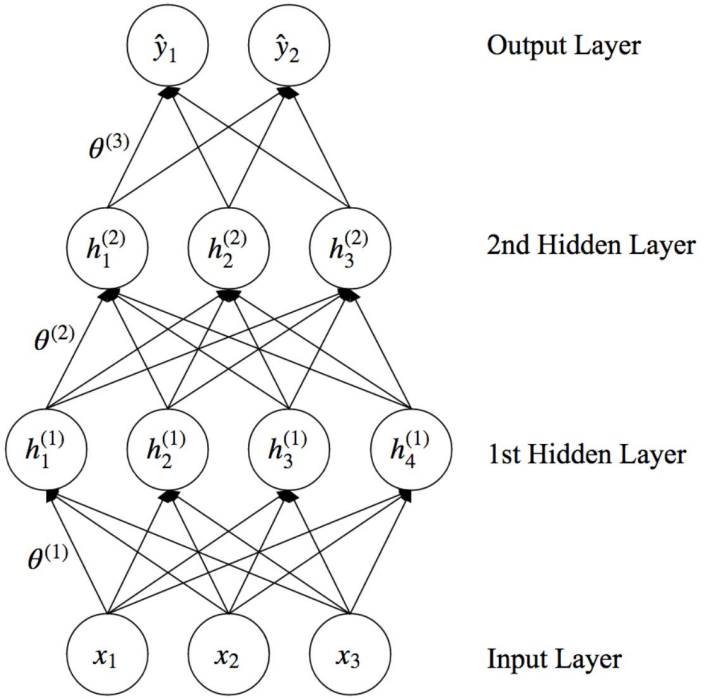
$\mathbf{w}^k$ is the weight vector for the k-th output
b^k is the bias for the k-th output

Recall:
This softmax function is a generalization of logistic (sigmoid) function.

Slide Credit: Geoff Hinton

Overview of Neural Networks

- Input Layer: provides input
- Hidden Layer: features extracted from input
- Output Layer: output of the network
- Parameters (or weights) for each layer



Slide credit: Junhyuk Oh, Jacob Abernethy

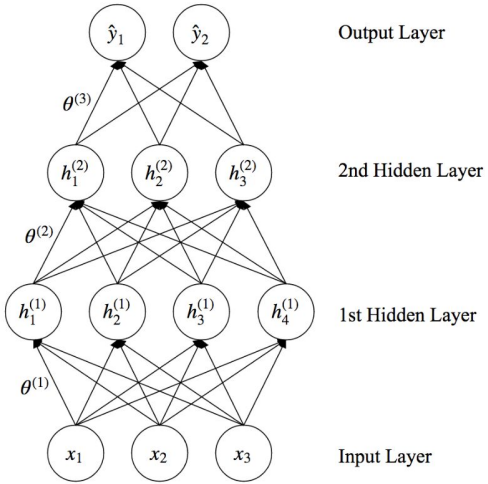
Overview of Neural Networks

- A **loss function** is defined over the *output units* and *desired outputs* (i.e., labels)

$$\mathcal{L}\left(\mathbf{y}, \hat{\mathbf{y}}\right) \text{ where } \hat{\mathbf{y}} = f(\mathbf{x}; \theta)$$

- The parameter of the network is trained to minimize the loss function based on gradient descent methods

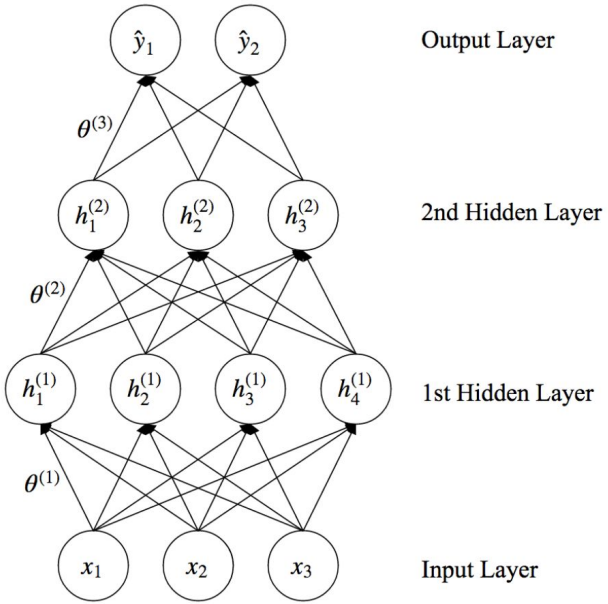
$$\min_{\theta} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim \text{data}} \left[\mathcal{L}\left(\mathbf{y}, \hat{\mathbf{y}}\right)\right] \text{ where } \hat{\mathbf{y}} = f(\mathbf{x}; \theta)$$



Slide credit: Junhyuk Oh, Jacob Abernethy

Overview of Neural Networks

- Forward Propagation (inference): Compute $\hat{\mathbf{y}} = f(\mathbf{x}; \theta)$ (output given input)
- Backward Propagation (learning): Compute $\nabla_\theta \mathcal{L}$ (gradient of loss w.r.t. parameters)



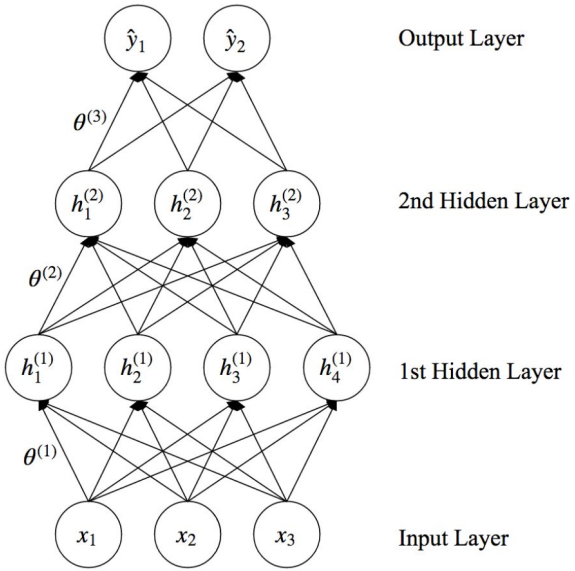
Slide credit: Junhyuk Oh, Jacob Abernethy

Forward Propagation

- The activation of each unit is computed based on **the previous layer** and **parameters (or weights)** associated with edges

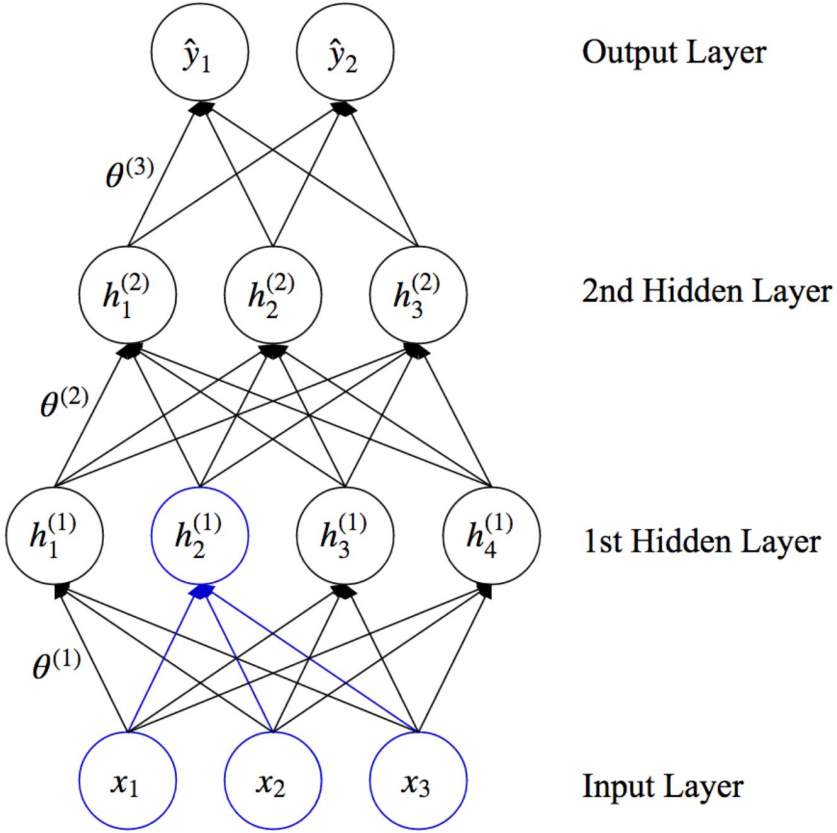
$$\underbrace{\mathbf{h}^{(l)}}_{l\text{-th layer}} = f^{(l)}\Big(\underbrace{\mathbf{h}^{(l-1)}}_{(l-1)\text{-th layer}} ; \underbrace{\theta^{(l)}}_{\text{weights}} \Big) \text{ where } \mathbf{h}^{(0)} \equiv \mathbf{x}, \mathbf{h}^{(L)} \equiv \hat{\mathbf{y}}$$

$$\hat{\mathbf{y}} = f(\mathbf{x}; \theta) = f^{(L)} \circ f^{(L-1)} \cdots f^{(2)} \circ f^{(1)} \left(\mathbf{x}; \theta^{(1)} \right)$$



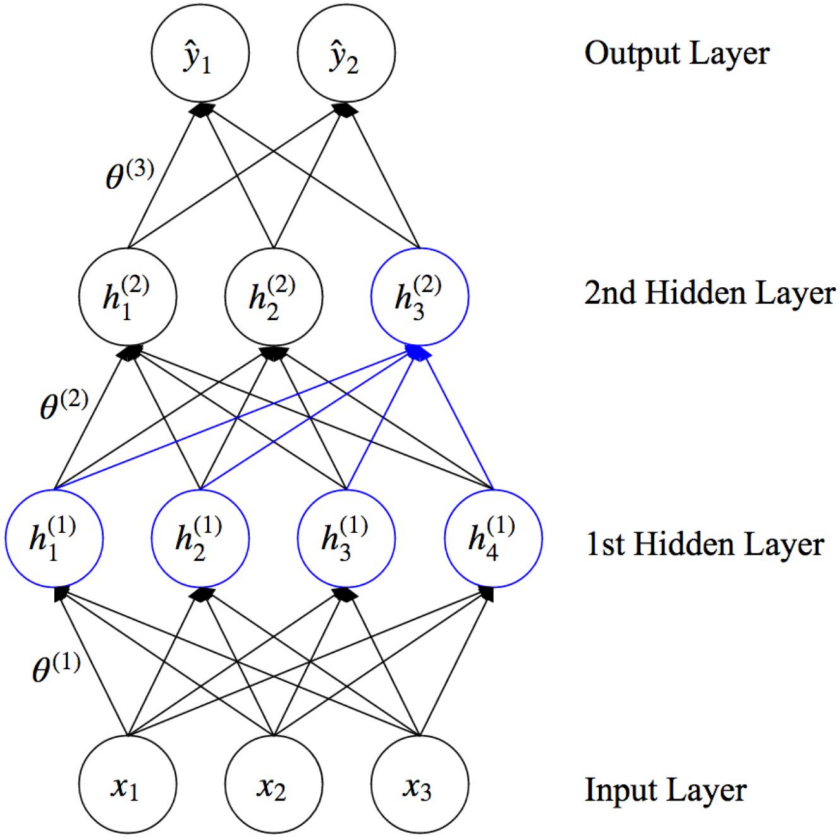
Slide credit: Junhyuk Oh, Jacob Abernethy

Forward Propagation



Slide credit: Junhyuk Oh, Jacob Abernethy

Forward Propagation



Slide credit: Junhyuk Oh, Jacob Abernethy

Training Neural Networks

- Repeat until convergence
  - $(\mathbf{x}, \mathbf{y}) \leftarrow$ Sample an example (or a mini-batch) from data
  - $\hat{\mathbf{y}} \leftarrow f(\mathbf{x}; \theta)$ Forward propagation
  - Compute $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$
  - $\nabla_\theta \mathcal{L} \leftarrow$ Backward propagation
  - Update weights using (stochastic) gradient descent
    - $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$

Slide credit: Junhyuk Oh, Jacob Abernethy

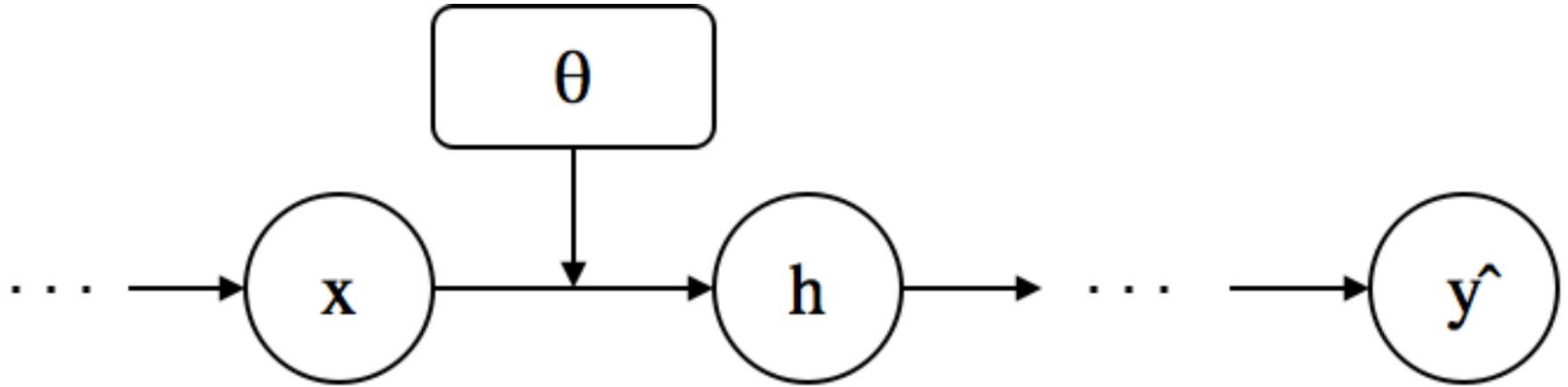
Overview of backpropagation

- Recall: Deep Neural Network represent a complex function with nested, composite functions (represented by layer-wise operation and nonlinearity, etc.)

- Q. How can we compute the gradient of the complex function?
- A. Backpropagation:
  - Computing gradient via **chain rule** for compositional function
  - The chain rule can be expressed as a **local computation**
  - Think about it as a **computational graph**

Backpropagation

- Denote x, h, θ is the input, output, and parameter of a layer.
- It is non-trivial to derive the gradient of loss w.r.t. parameters in intermediate layers



Slide credit: Junhyuk Oh, Jacob Abernethy

Idea behind backpropagation

- Assuming that $\frac{\partial \mathrm{L}}{\partial h}$ is given, use the **chain rule** to compute the gradients

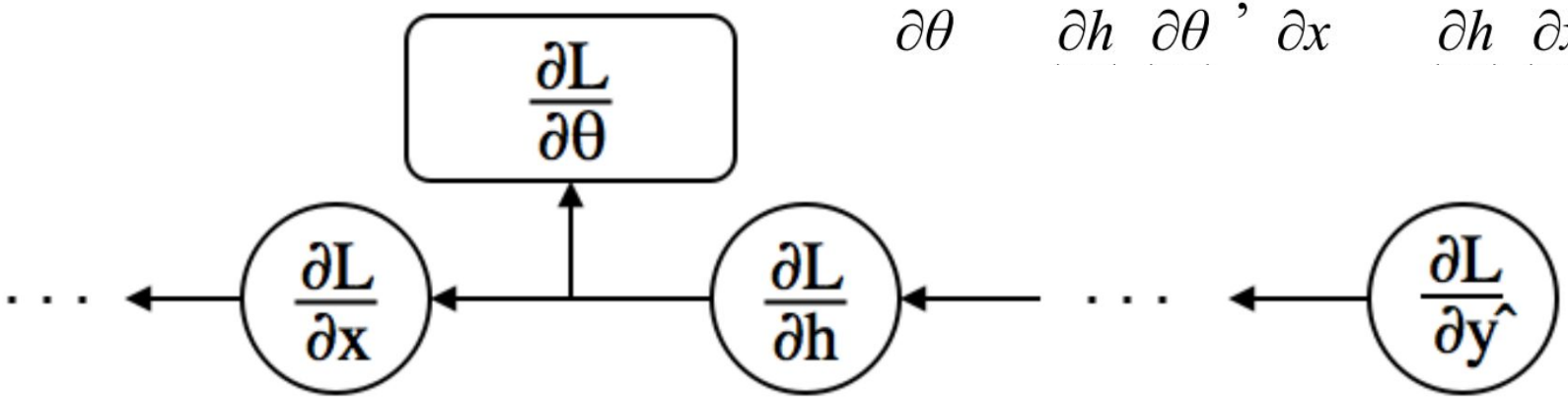
$$\frac{\partial \mathrm{L}}{\partial \theta} = \underbrace{\frac{\partial \mathrm{L}}{\partial h}}_{\text{giveneasy}} \underbrace{\frac{\partial h}{\partial \theta}}, \quad \frac{\partial \mathrm{L}}{\partial x} = \underbrace{\frac{\partial \mathrm{L}}{\partial h}}_{\text{giveneasy}} \underbrace{\frac{\partial h}{\partial x}}$$

Slide credit: Junhyuk Oh, Jacob Abernethy

Idea behind backpropagation

- We need only $\frac{\partial L}{\partial \theta}$ for gradient descent. Why compute $\frac{\partial L}{\partial x}$?
  - The previous layer needs it because x is the output of the previous layer.

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \theta}, \frac{\partial L}{\partial x} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial x}$$



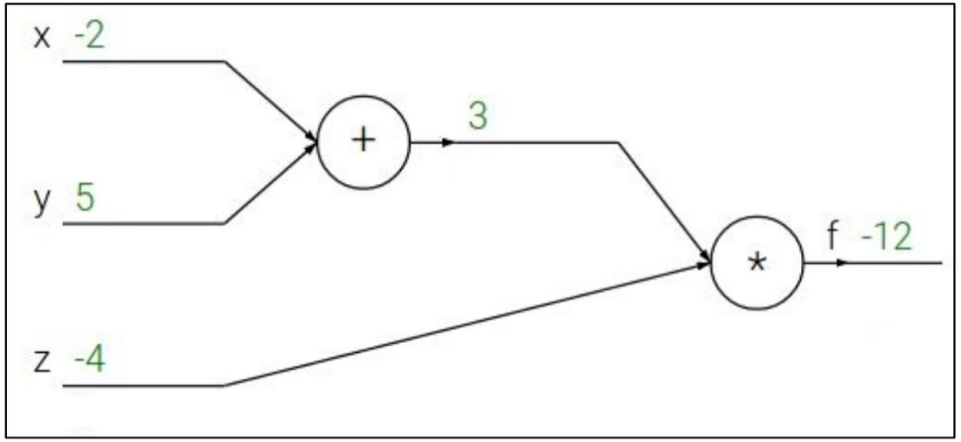
Slide credit: Junhyuk Oh, Jacob Abernethy

Backpropagation: toy examples (simple computational graphs)

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Backpropagation: a simple example

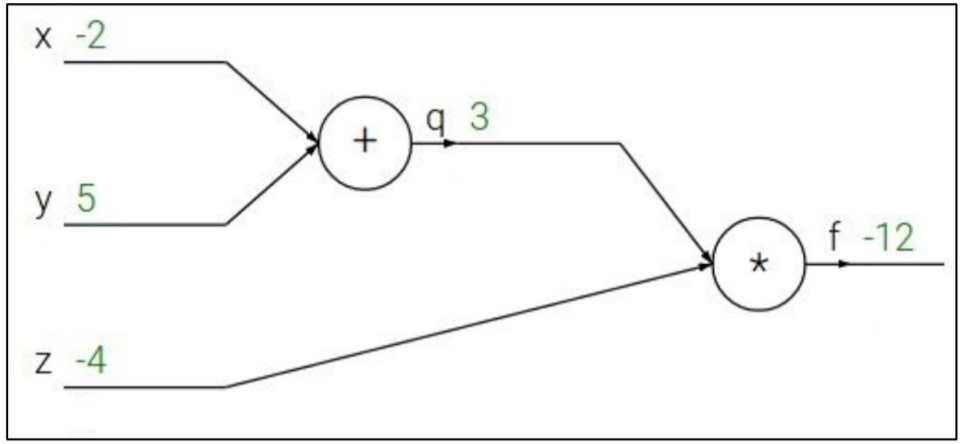
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Backpropagation: a simple example

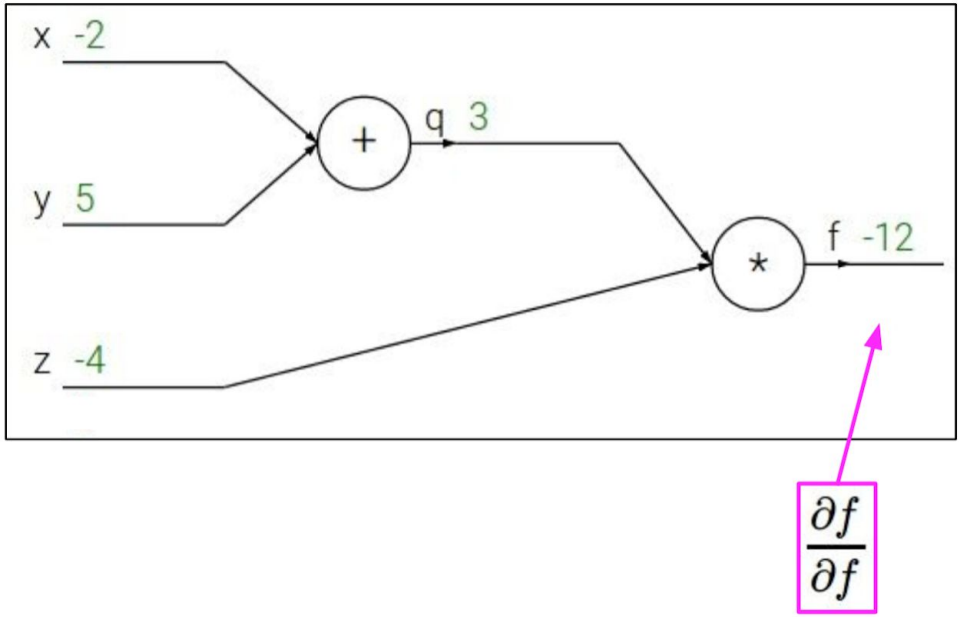
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Backpropagation: a simple example

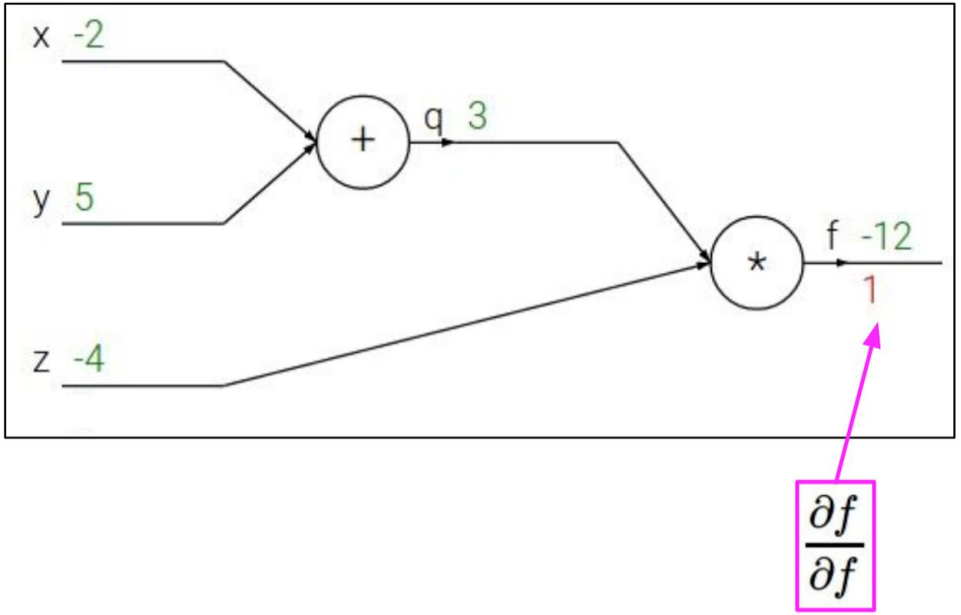
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

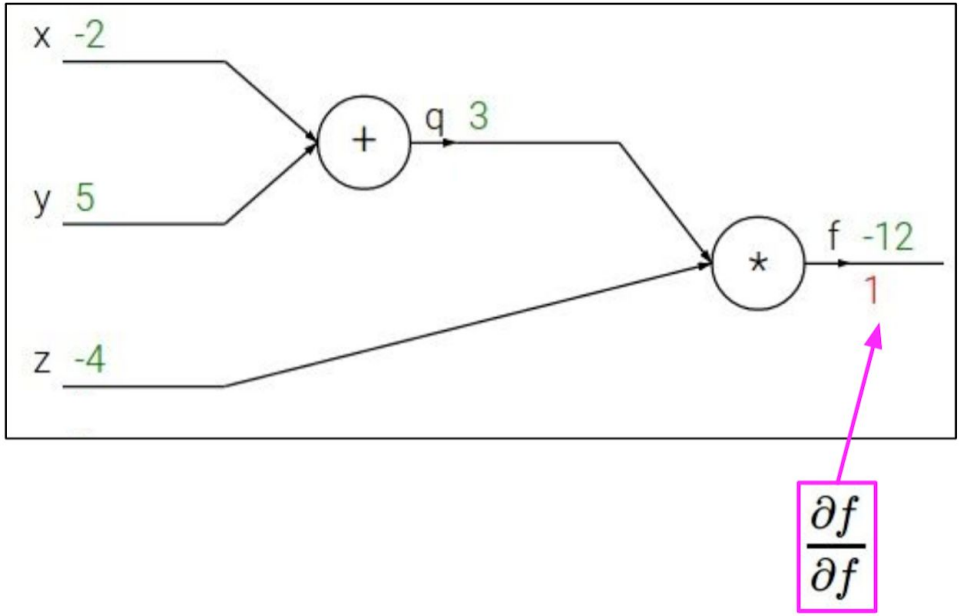
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

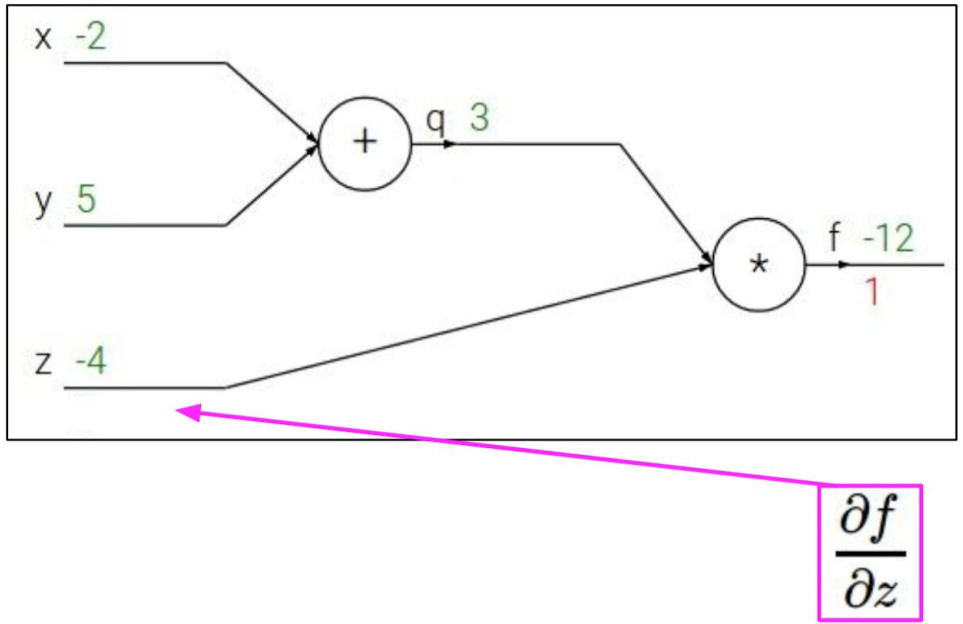
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

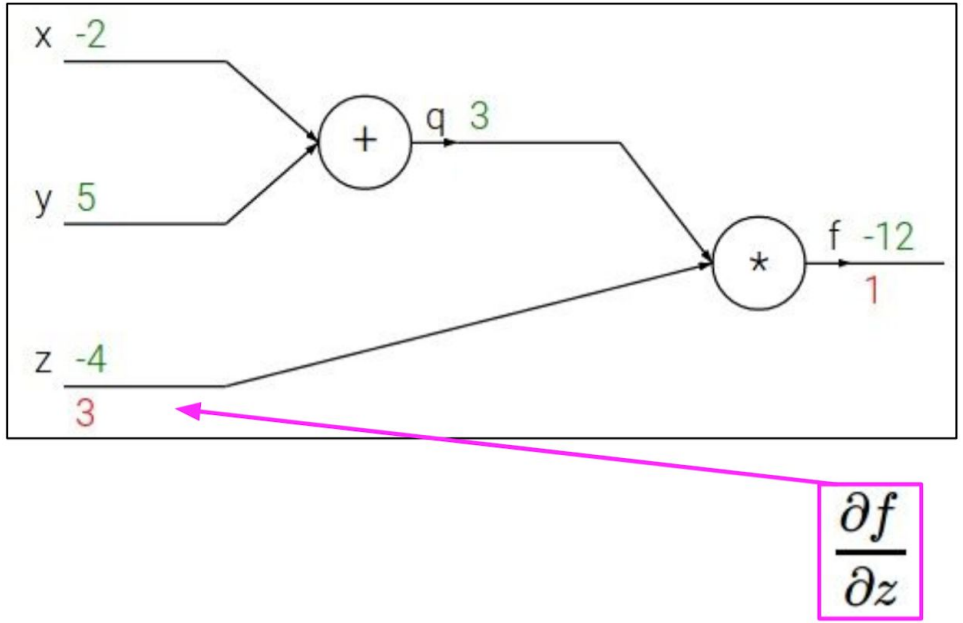
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

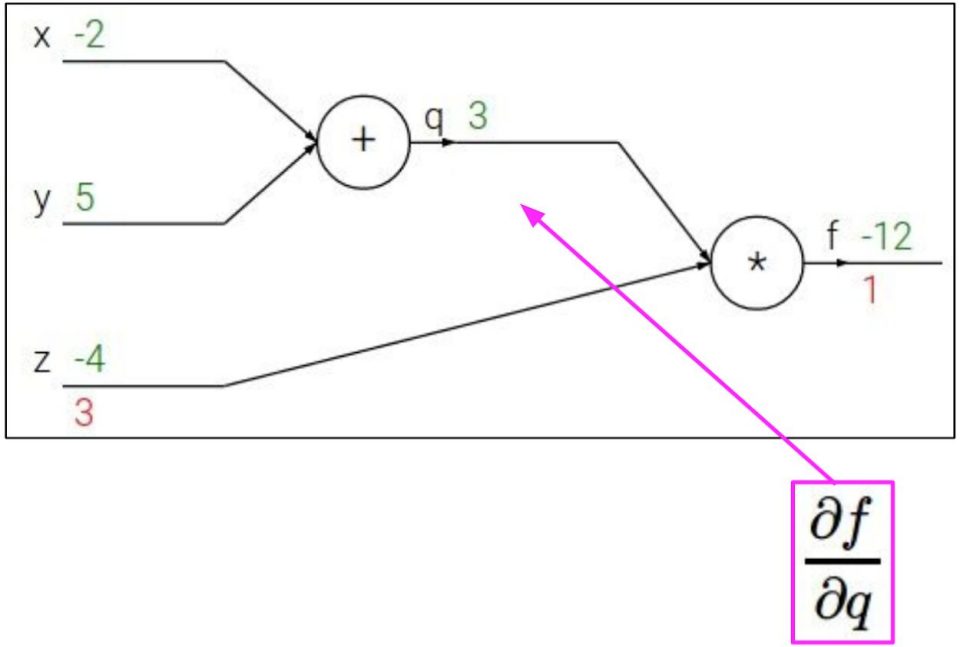
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

Backpropagation: a simple example

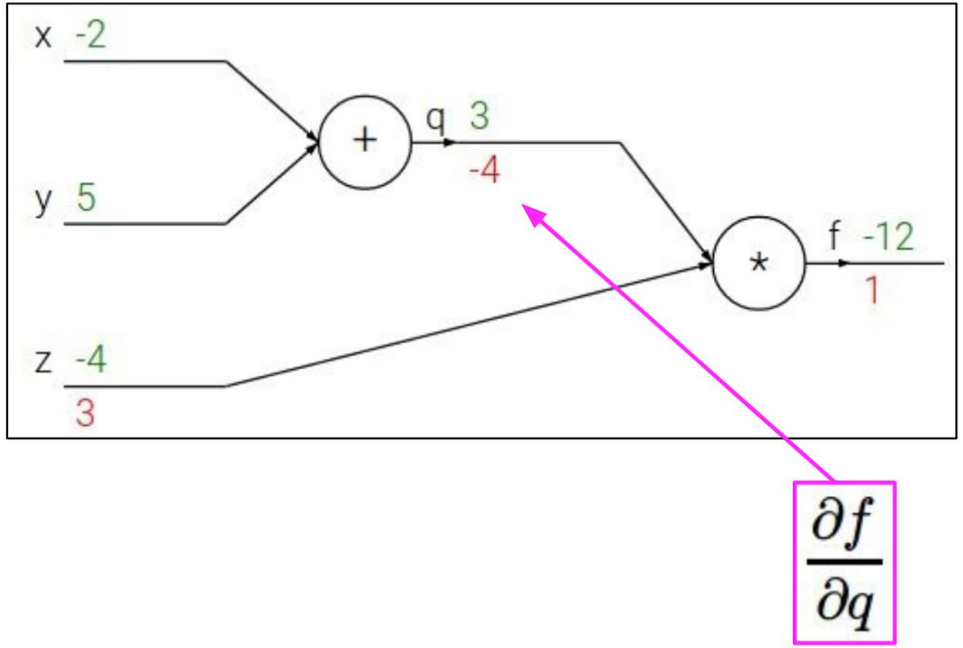
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

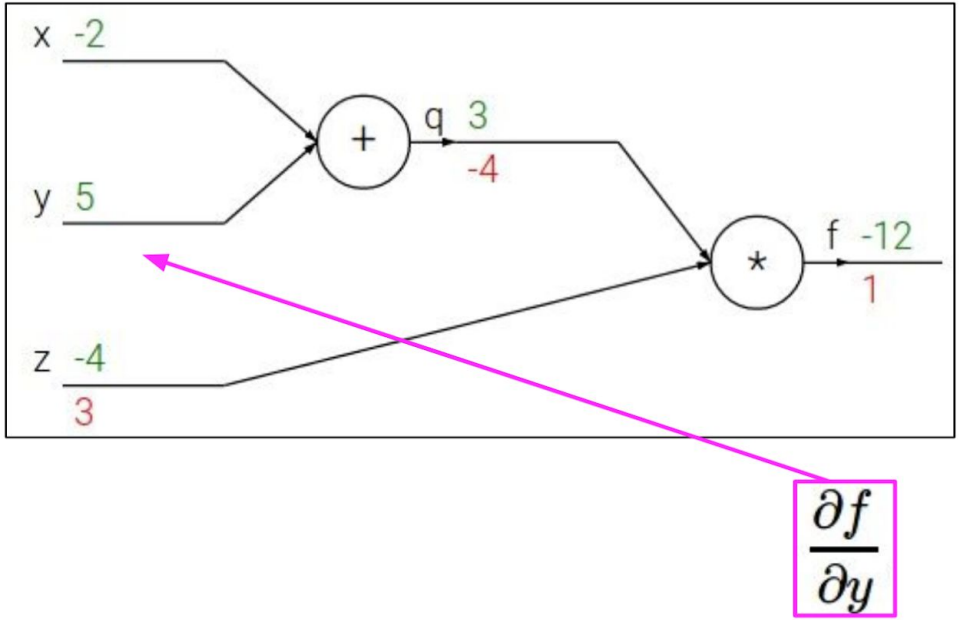
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

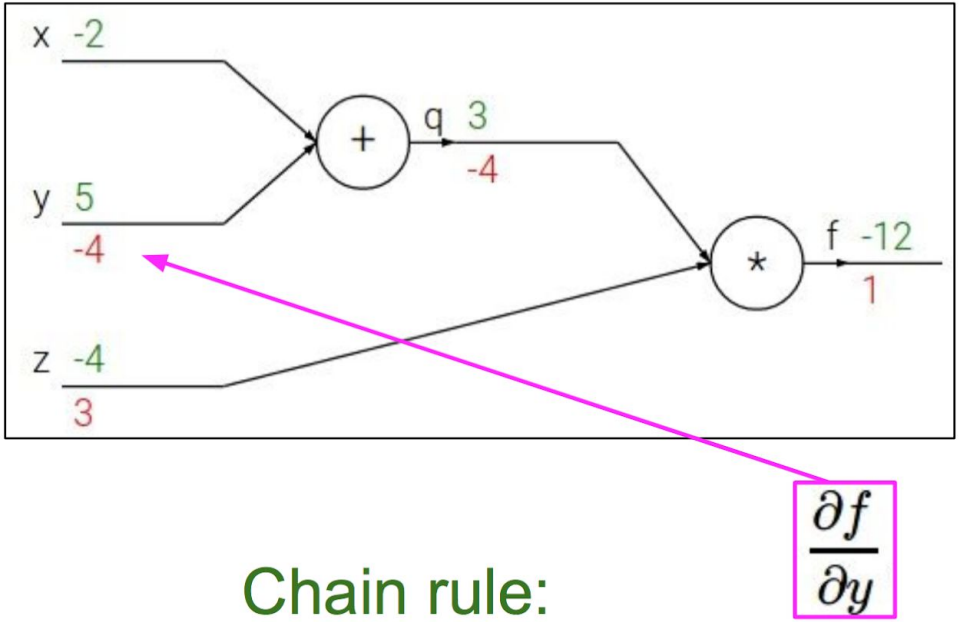
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Backpropagation: a simple example

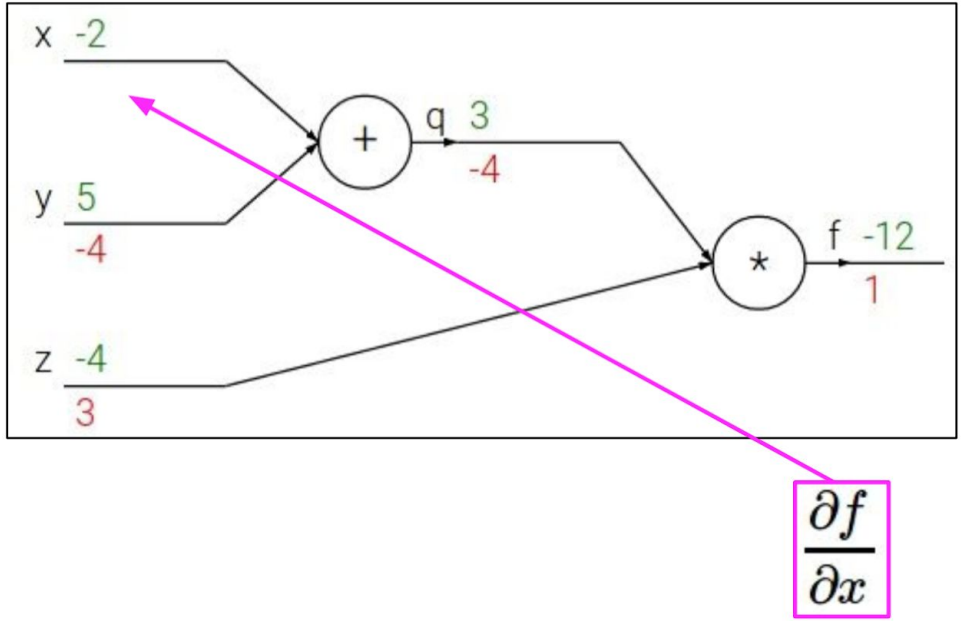
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

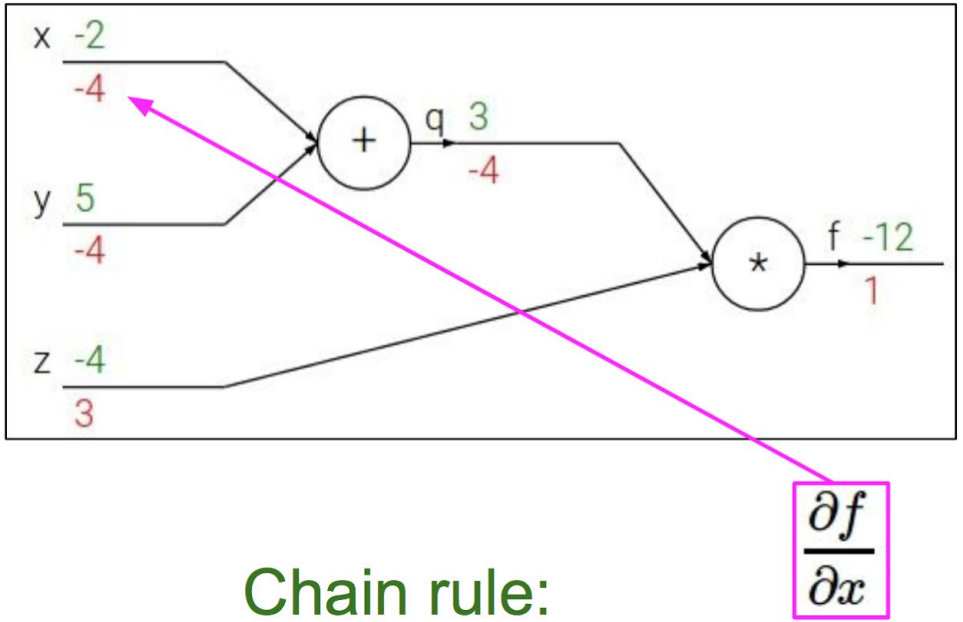
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

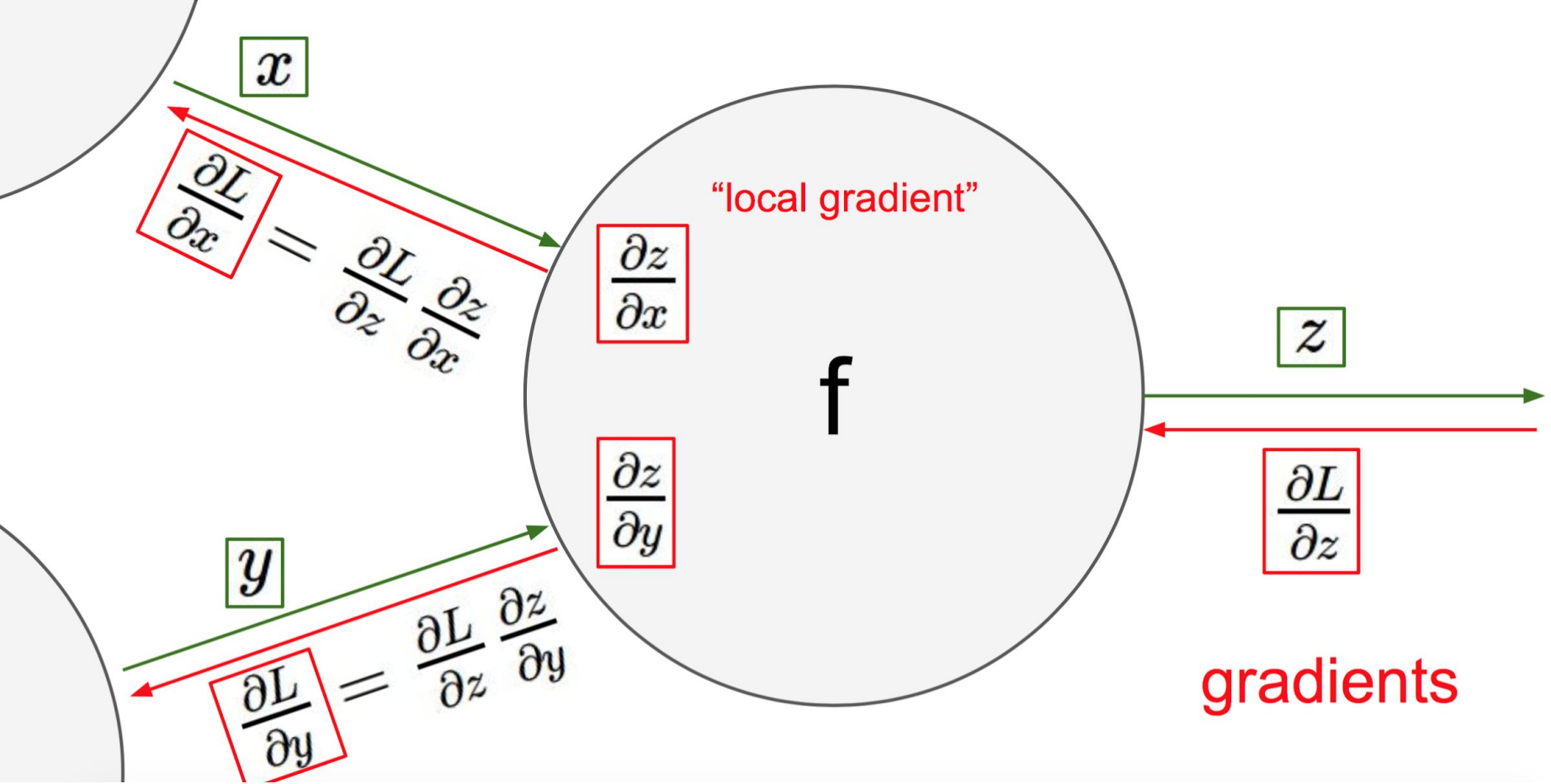
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung



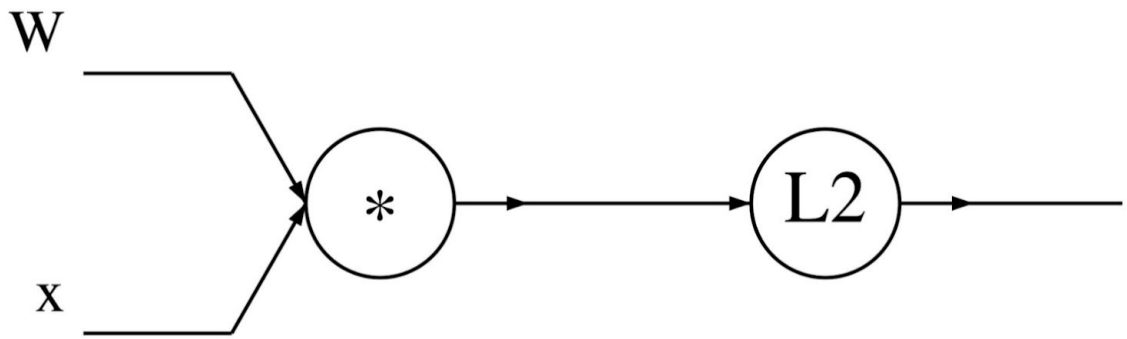
Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

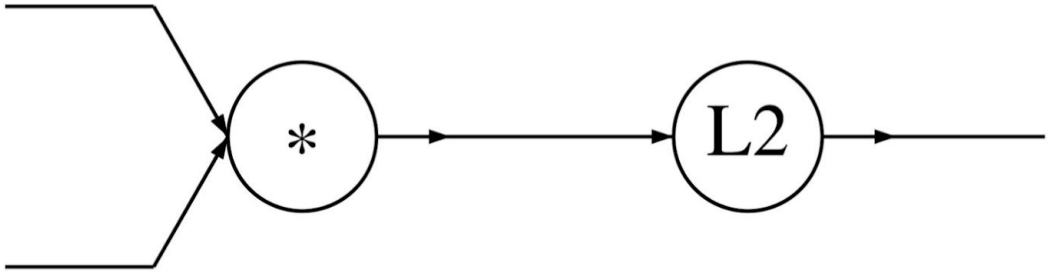


Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}_{\mathrm{W}}$$

$$\begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}_{\mathrm{x}}$$

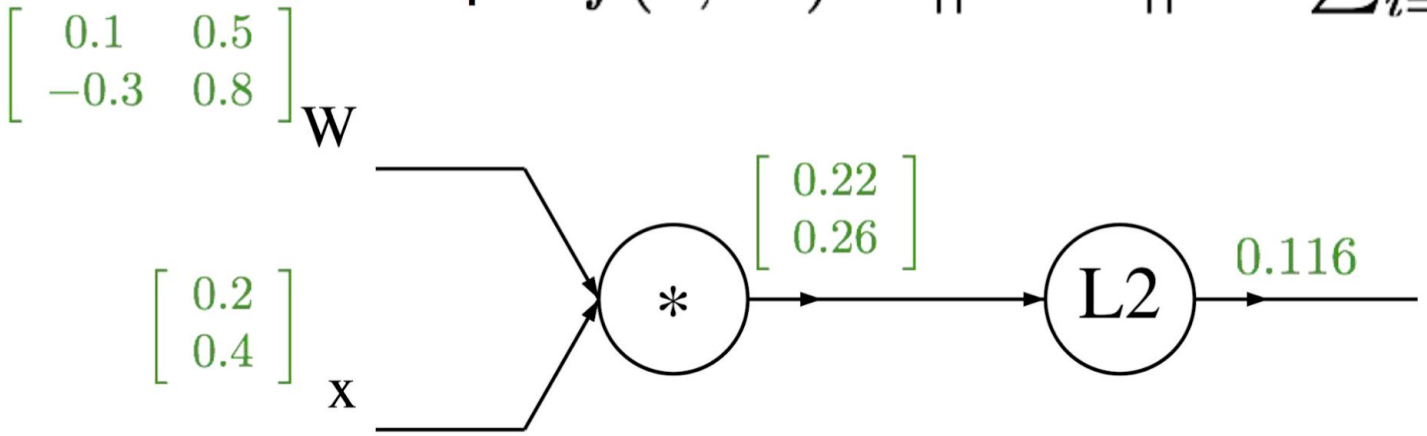


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

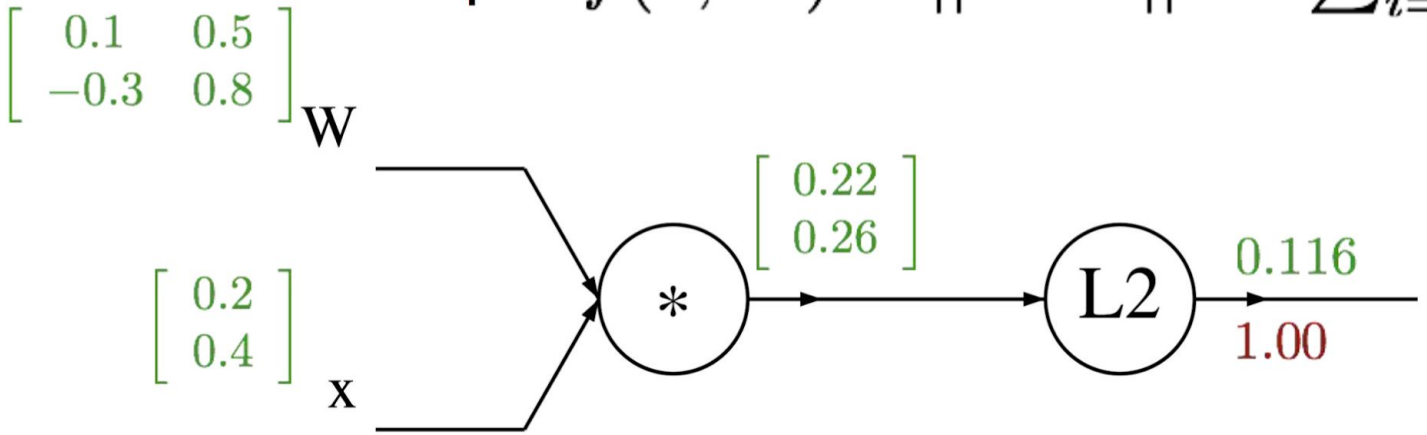


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

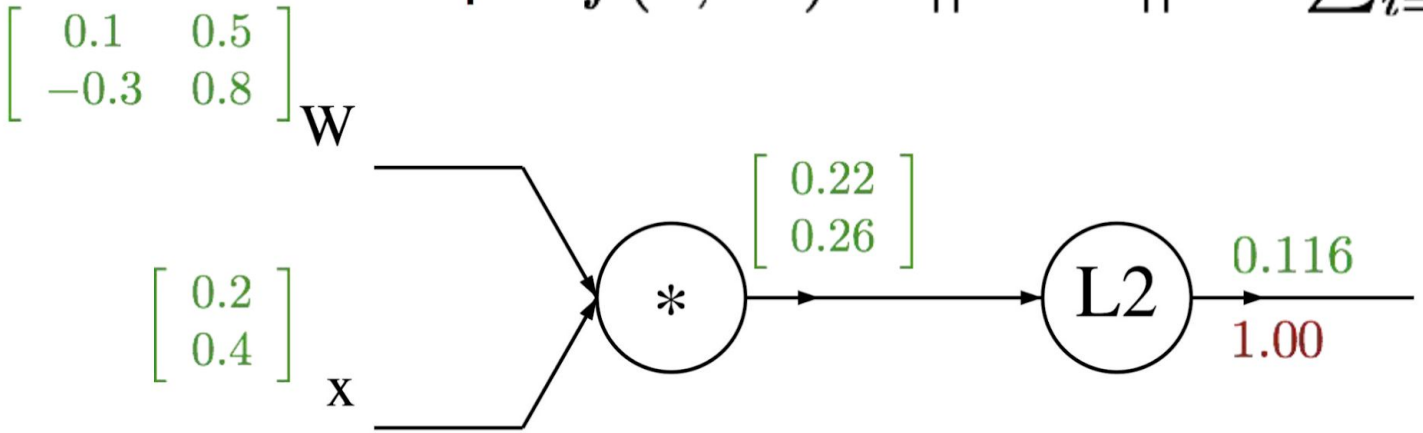


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

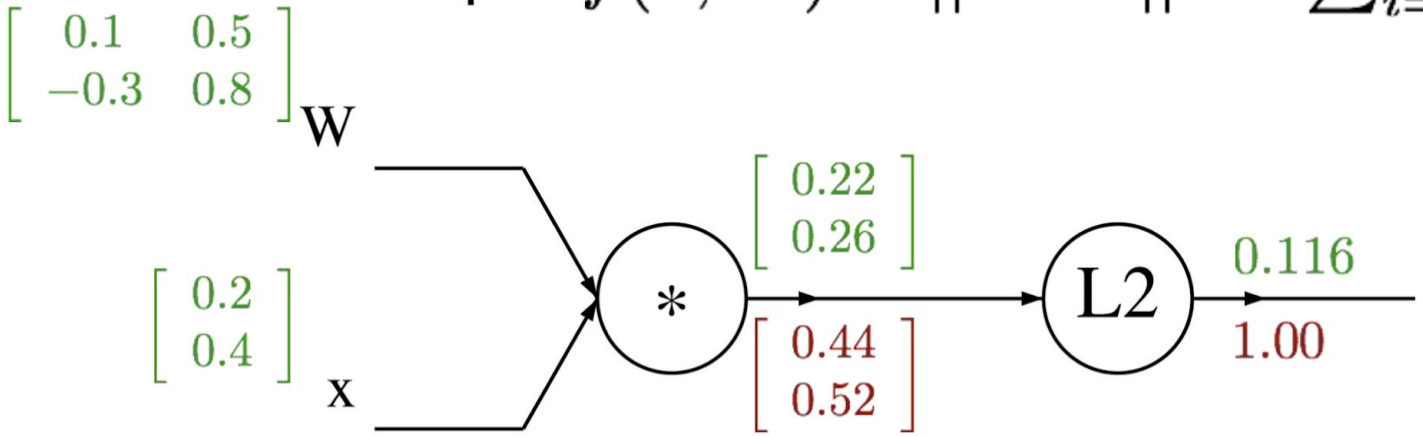
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\nabla_q f = 2q$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

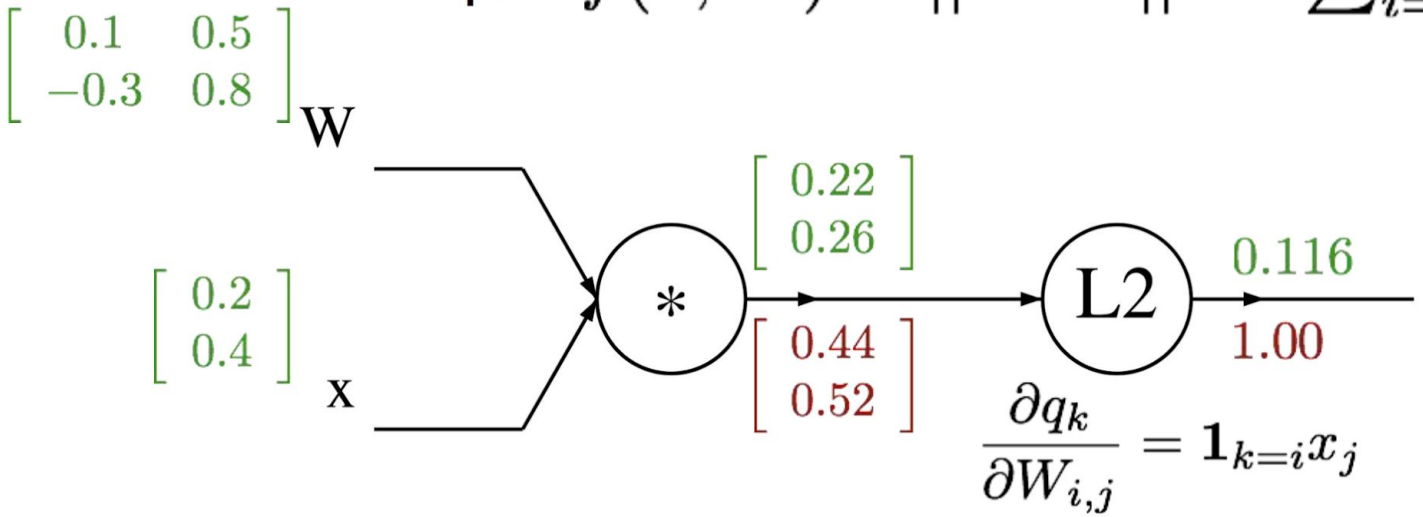
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\nabla_q f = 2q$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

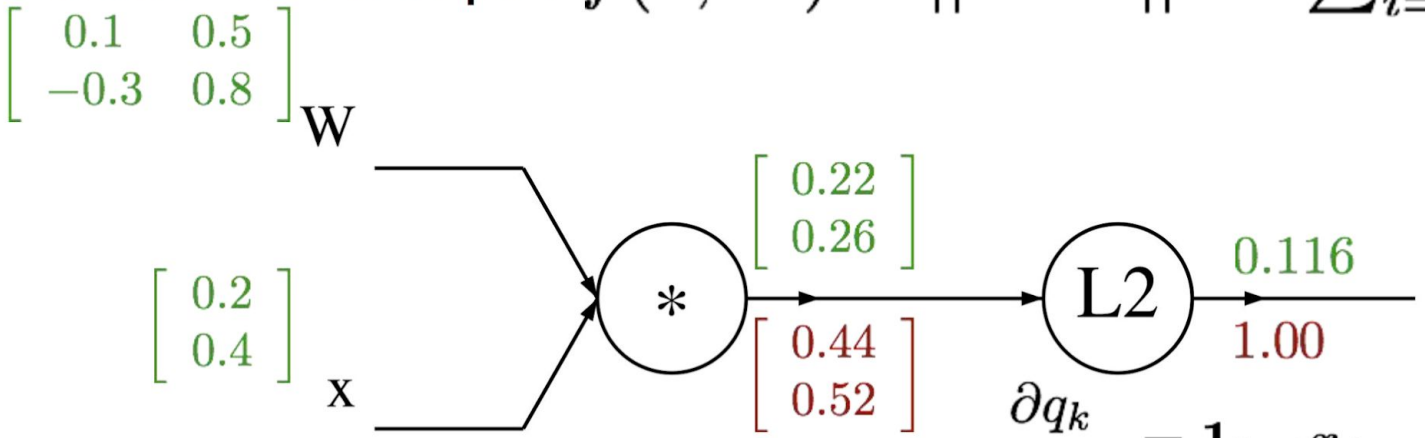


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

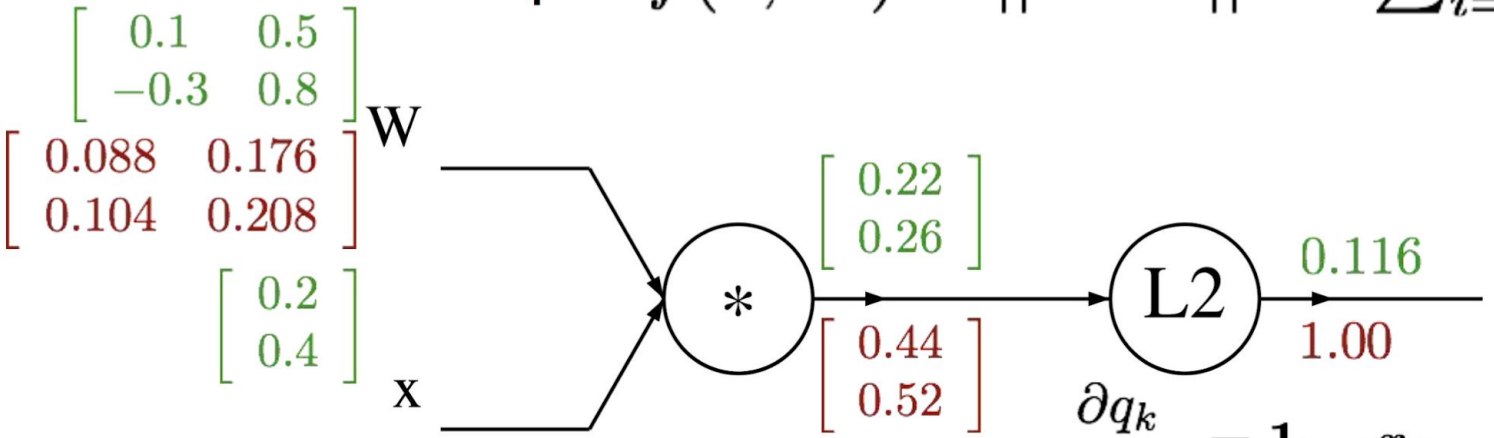
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j \end{aligned}$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

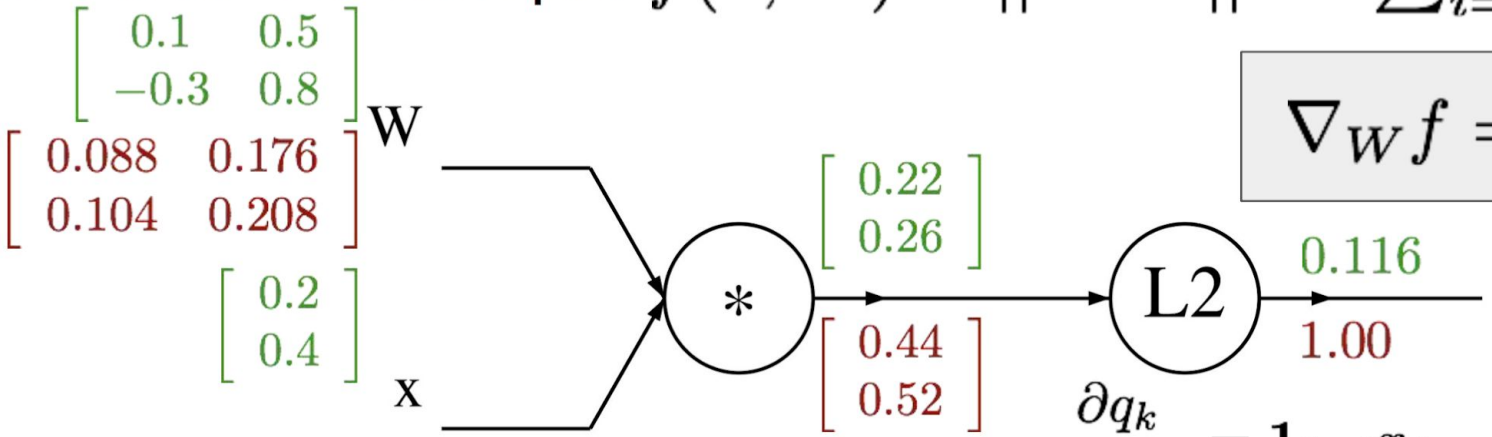
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j \end{aligned}$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$\nabla_W f = 2q \cdot x^T$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j \end{aligned}$$

Slide credit: Fei-Fei Li, Justin Johnson, Serena Yeung

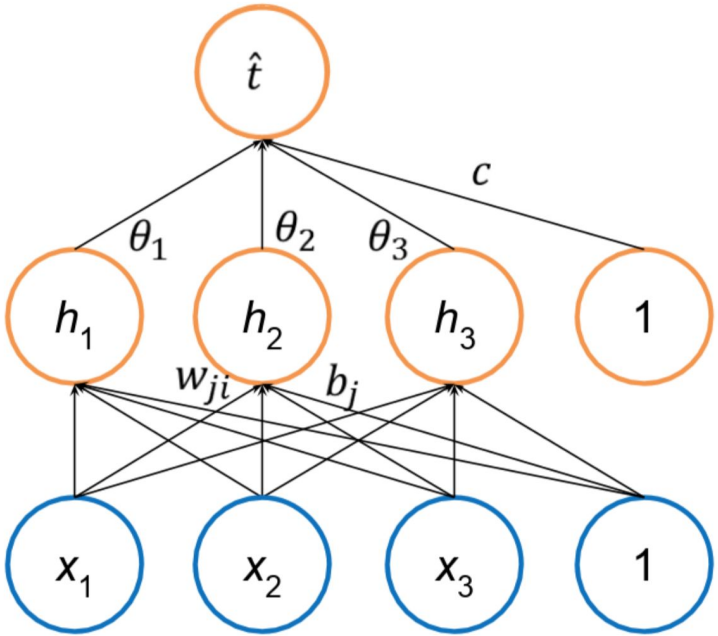
Backpropagation: examples
(NN with 1-hidden layer for regression)

Forward Propagation

- Example: a network with 1 hidden layer
  - Input: $\boldsymbol{x}$
  - Output: $\hat{t}$
  - Target: t
  - Loss function: squared error $(\hat{t} - t)^2$

Hidden layer $$h_j = f(\sum_i w_{ji} x_i + b_j)$$

Output $$\hat{t} = \sum_j \theta_j h_j + c$$



Slide Credit: Yoshua Bengio

Forward Propagation

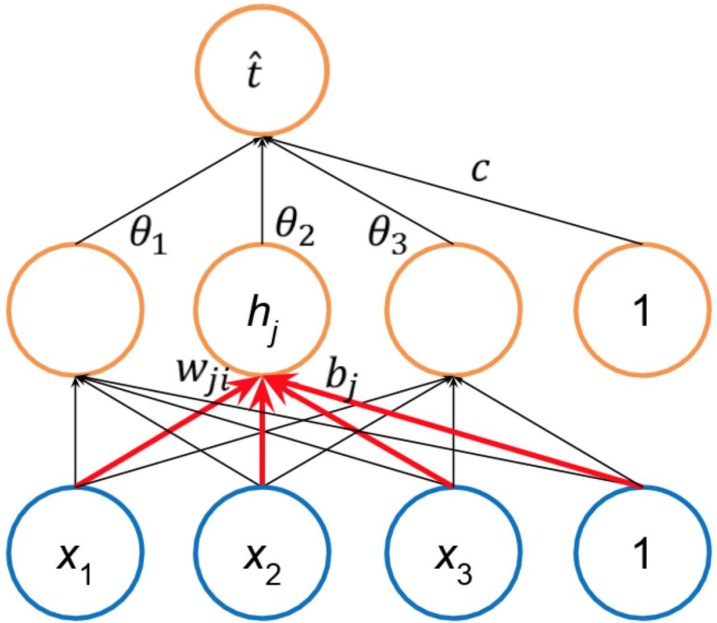
- Example: a network with 1 hidden layer
  - Input: $\boldsymbol{x}$
  - Output: $\hat{t}$
  - Target: t
  - Loss function: squared error $(\hat{t} - t)^2$

Hidden layer

$$h_j = f(\sum_i w_{ji} x_i + b_j)$$

Output

$$\hat{t} = \sum_j \theta_j h_j + c$$



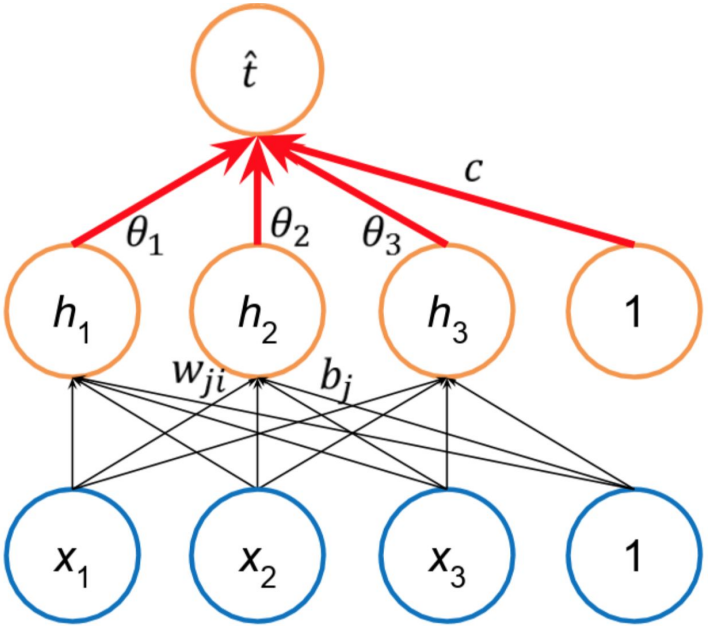
Slide Credit: Yoshua Bengio

Forward Propagation

- Example: a network with 1 hidden layer
  - Input: $\boldsymbol{x}$
  - Output: $\hat{t}$
  - Target: t
  - Loss function: squared error $(\hat{t} - t)^2$

Hidden layer $$h_j = f(\sum_i w_{ji} x_i + b_j)$$

Output $$\hat{t} = \sum_j \theta_j h_j + c$$



Slide Credit: Yoshua Bengio

Backpropagation

- Goal: compute gradient w.r.t parameters $\{W, b, \theta, c\}$
- Main idea: apply a chain rule recursively

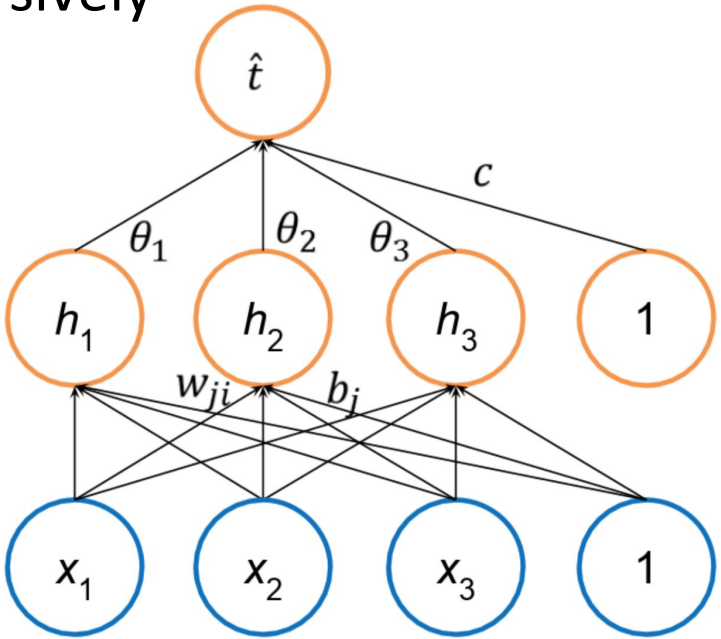
$$\frac{\partial L}{\partial \hat{t}} = 2(\hat{t} - t)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{t}}{\partial \theta_j}\frac{\partial L}{\partial \hat{t}} = h_j \frac{\partial L}{\partial \hat{t}}$$

$$\frac{\partial L}{\partial h_j} = \frac{\partial \hat{t}}{\partial h_j}\frac{\partial L}{\partial \hat{t}} = \theta_j \frac{\partial L}{\partial \hat{t}}$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}}\frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

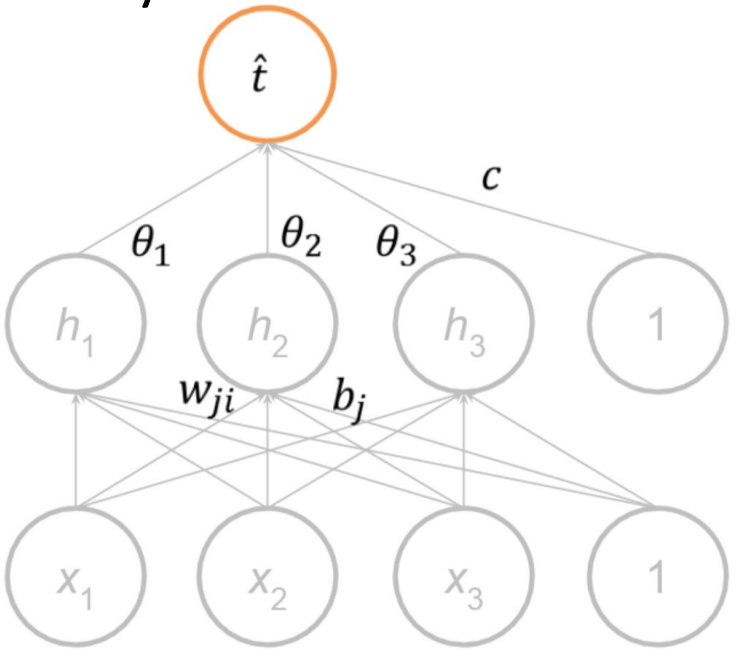
$$f' = f'(\sum_i w_{ji} x_i + b_j)$$



Backpropagation

- Goal: compute gradient w.r.t parameters $\{W, b, \theta, c\}$
- Main idea: apply a chain rule recursively

$$\frac{\partial L}{\partial \hat{t}} = 2(\hat{t} - t)$$



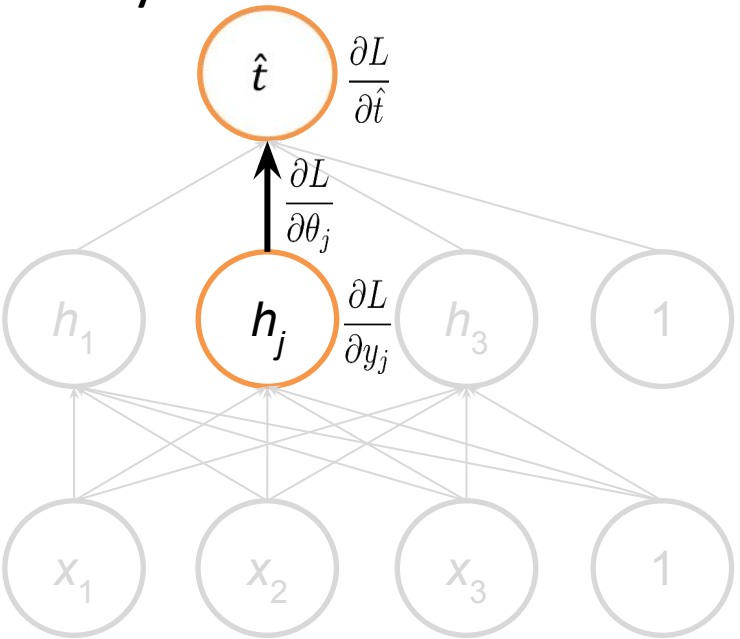
Backpropagation

- Goal: compute gradient w.r.t parameters $\{W, b, \theta, c\}$
- Main idea: apply a chain rule recursively

$$\frac{\partial L}{\partial \hat{t}} = 2(\hat{t} - t)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{t}}{\partial \theta_j}\frac{\partial L}{\partial \hat{t}} = h_j \frac{\partial L}{\partial \hat{t}}$$

$$\frac{\partial L}{\partial h_j} = \frac{\partial \hat{t}}{\partial h_j}\frac{\partial L}{\partial \hat{t}} = \theta_j \frac{\partial L}{\partial \hat{t}}$$



Backpropagation

- Goal: compute gradient w.r.t parameters $\{W, b, \theta, c\}$
- Main idea: apply a chain rule recursively

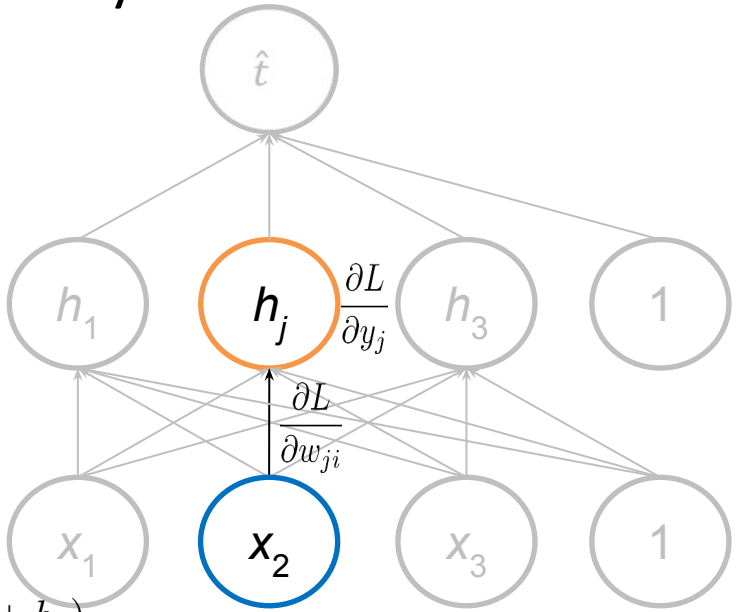
$$\frac{\partial L}{\partial \hat{t}} = 2(\hat{t} - t)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{t}}{\partial \theta_j}\frac{\partial L}{\partial \hat{t}} = h_j \frac{\partial L}{\partial \hat{t}}$$

$$\frac{\partial L}{\partial h_j} = \frac{\partial \hat{t}}{\partial h_j}\frac{\partial L}{\partial \hat{t}} = \theta_j \frac{\partial L}{\partial \hat{t}}$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}}\frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

where $f' = f'(\sum_i w_{ji} x_i + b_j)$



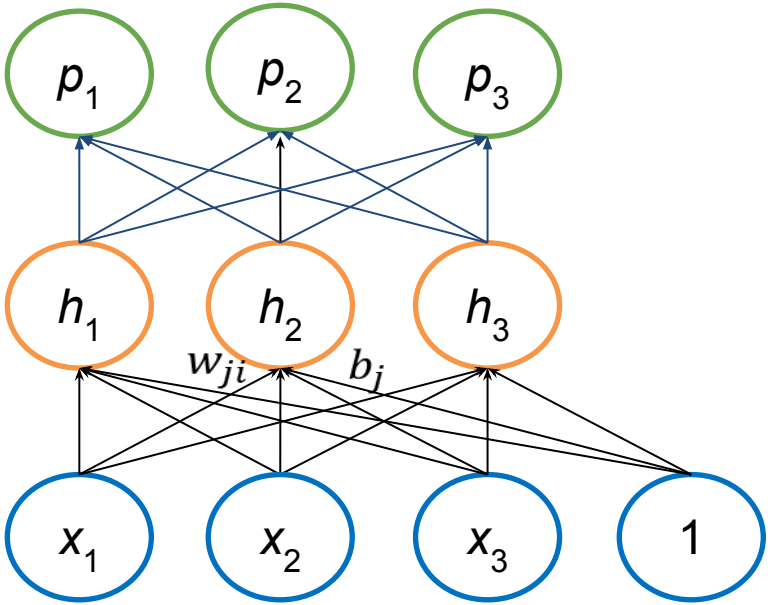
Backpropagation: examples (NN with 1-hidden layer for classification)

Forward Propagation

- Example: a network with 1 hidden layer
  - Input: $\mathbf{x}$
  - Output: $\mathbf{p}$
  - Target: $\mathbf{t}$
  - Loss function: cross entropy $-\sum_i t_i \log p_i$

Hidden layer $$h_j = f(\sum_i w_{ji} x_i + b_j)$$

Output $$p_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$$

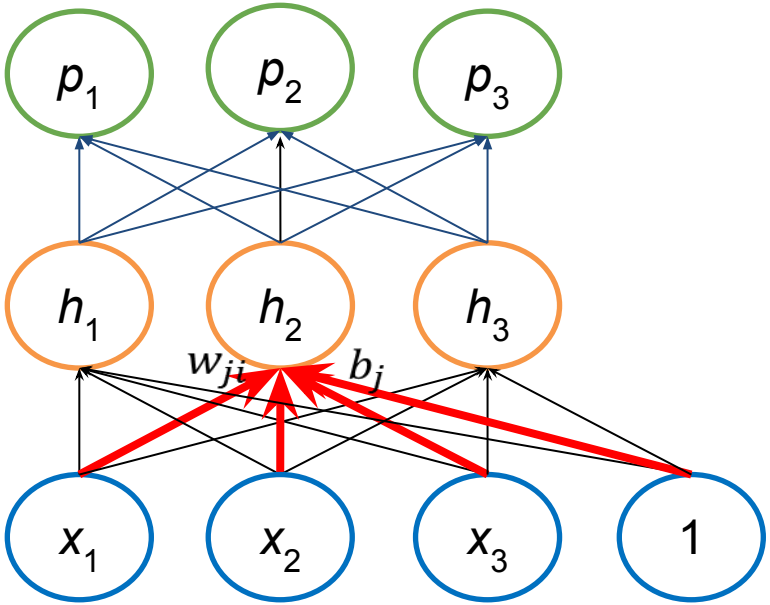


Forward Propagation

- Example: a network with 1 hidden layer
  - Input: $\mathbf{x}$
  - Output: $\mathbf{p}$
  - Target: $\mathbf{t}$
  - Loss function: cross entropy $-\sum_i t_i \log p_i$

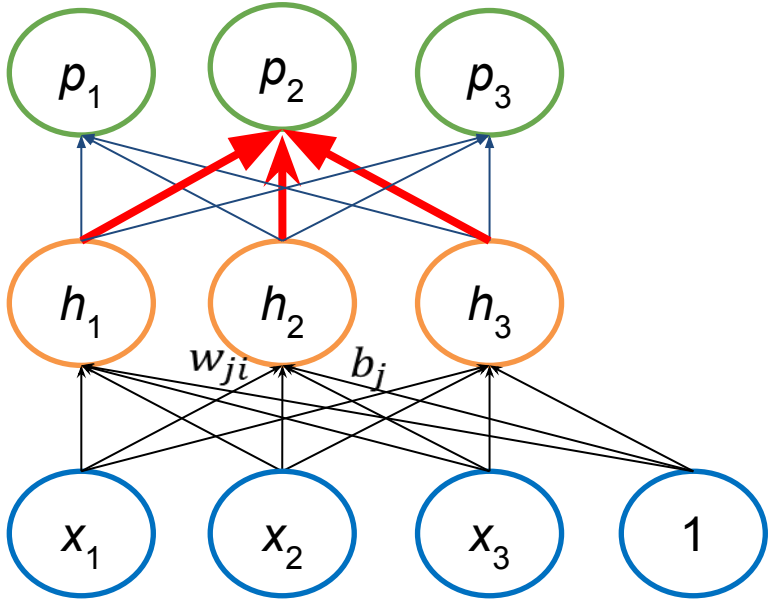
Hidden layer $$h_j = f(\sum_i w_{ji} x_i + b_j)$$

Output $$p_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$$



Forward Propagation

- Example: a network with 1 hidden layer
  - Input: $\mathbf{x}$
  - Output: $\mathbf{p}$
  - Target: $\mathbf{t}$
  - Loss function: cross entropy $-\sum_i t_i \log p_i$



Hidden layer $h_j = f(\sum_i w_{ji} x_i + b_j)$

Output $p_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$

Backpropagation

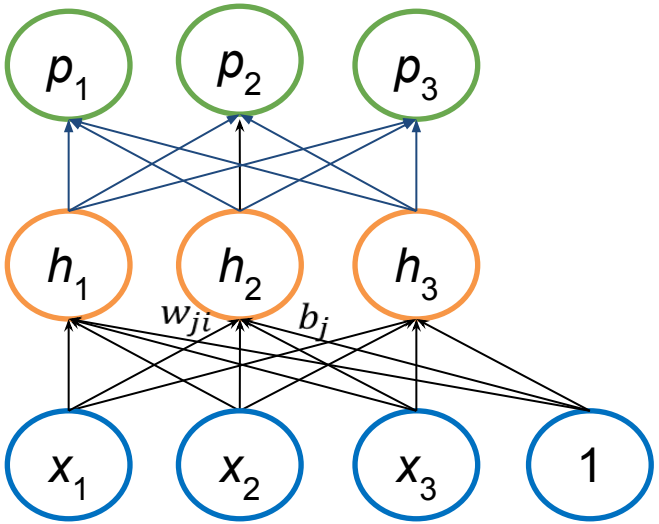
- Goal: compute gradient w.r.t parameters $\{\mathbf{W}, \mathbf{b}\}$
- Main idea: apply a chain rule recursively

$$L = -\sum_j t_j \log p_j = -\sum_j t_j \log \frac{e^{h_j}}{\sum_k e^{h_k}}$$
$$= -\sum_j t_j h_j + (\sum_j t_j) \log(\sum_k e^{h_k})$$
$$= \log(\sum_j e^{h_j}) - \sum_j t_j h_j$$

$$\frac{\partial L}{\partial h_j} = \frac{e^{h_j}}{\sum_k e^{h_k}} - t_j = p_j - t_j$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}} \frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

$$f' = f'(\sum_i w_{ji} x_i + b_j)$$

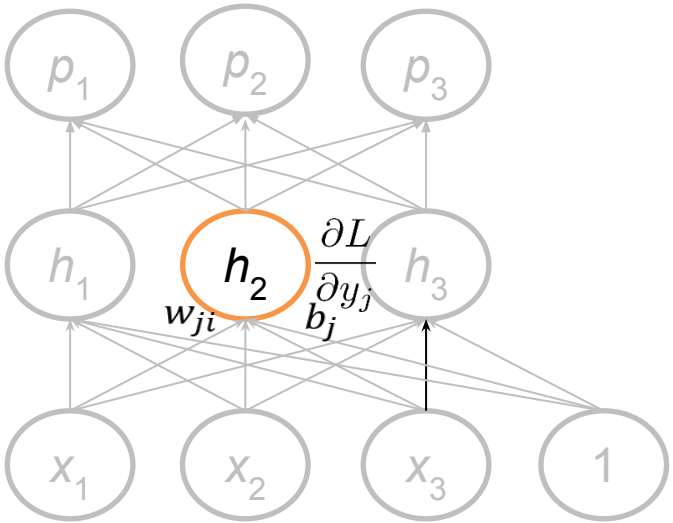


Backpropagation

- Goal: compute gradient w.r.t parameters $\{\mathbf{W}, \mathbf{b}\}$
- Main idea: apply a chain rule recursively

$$
\begin{aligned}
L &= -\sum_j t_j \log p_j = -\sum_j t_j \log \frac{e^{h_j}}{\sum_k e^{h_k}} \\
&= -\sum_j t_j h_j + (\sum_j t_j) \log(\sum_k e^{h_k}) \\
&= \log(\sum_j e^{h_j}) - \sum_j t_j h_j
\end{aligned}
$$

$$
\boxed{\frac{\partial L}{\partial h_j} = \frac{e^{h_j}}{\sum_k e^{h_k}} - t_j = p_j - t_j}
$$



Backpropagation

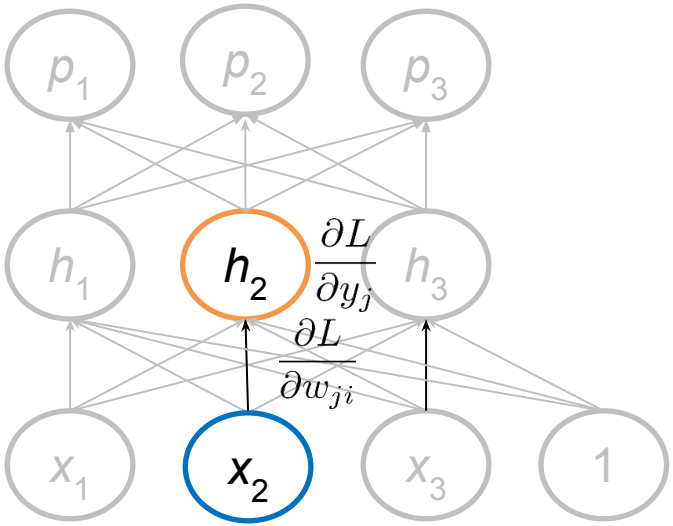
- Goal: compute gradient w.r.t parameters $\{\mathbf{W}, \mathbf{b}\}$
- Main idea: apply a chain rule recursively

$$L = -\sum_j t_j \log p_j = -\sum_j t_j \log \frac{e^{h_j}}{\sum_k e^{h_k}}$$
$$= -\sum_j t_j h_j + (\sum_j t_j) \log(\sum_k e^{h_k})$$
$$= \log(\sum_j e^{h_j}) - \sum_j t_j h_j$$

$$\frac{\partial L}{\partial h_j} = \frac{e^{h_j}}{\sum_k e^{h_k}} - t_j = p_j - t_j$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}} \frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

where $f' = f'(\sum_i w_{ji} x_i + b_j)$

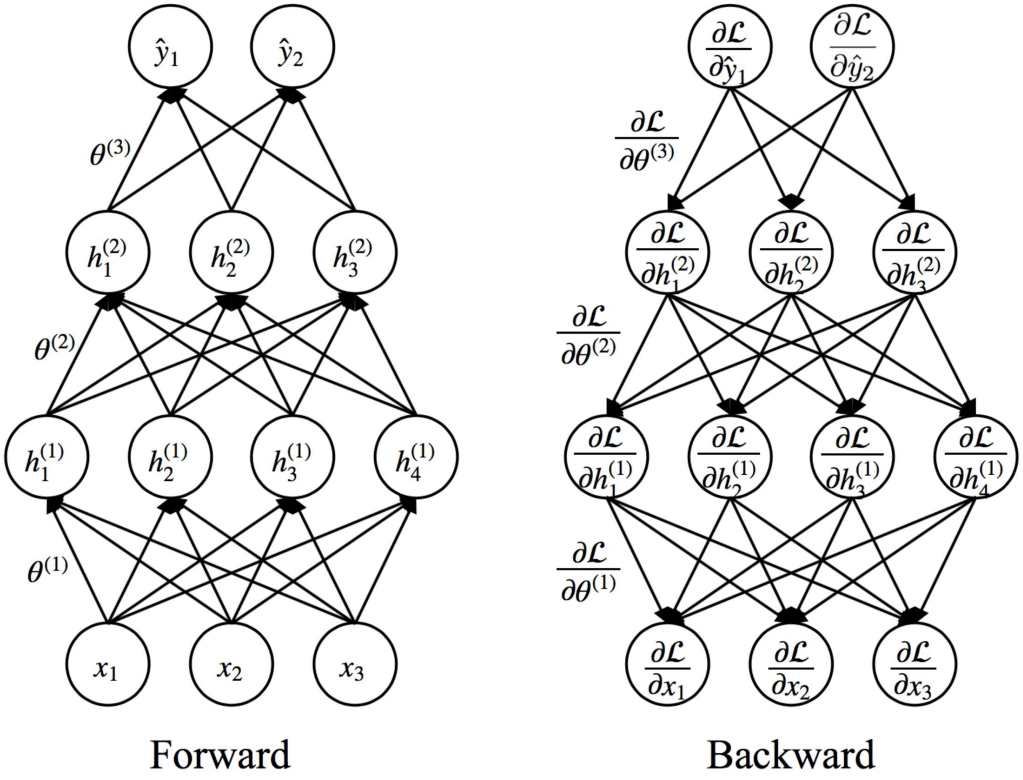


Deep Neural Networks

- Simple to construct
- Computing gradient is straightforward (via back propagation)
  - For general formula, see Goodfellow et al.'s book

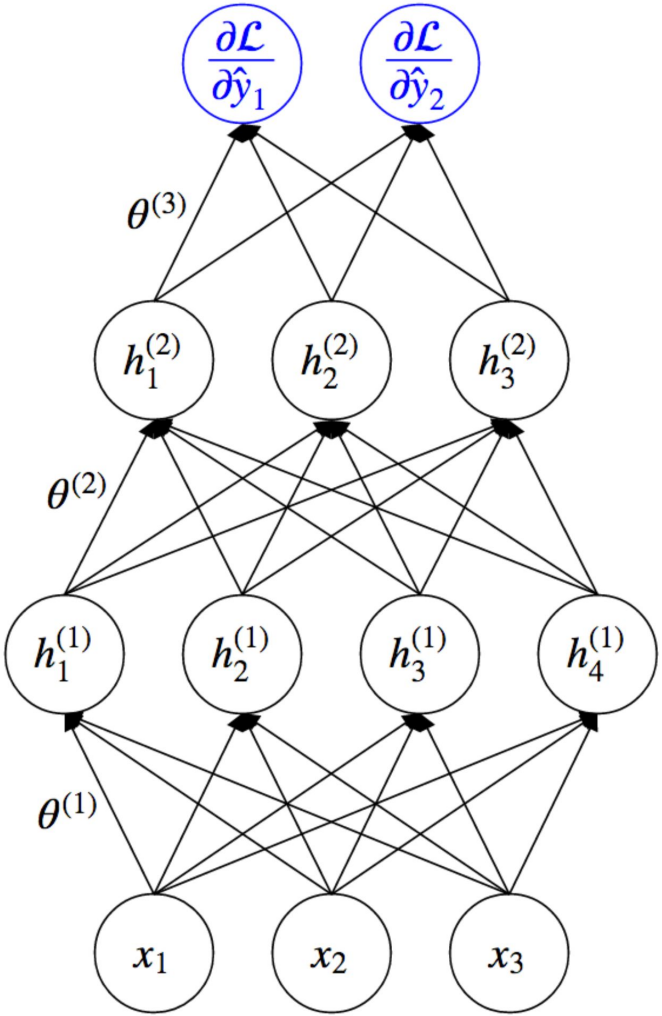
Backpropagation for deep neural nets

- Computing gradient via **chain rule** for compositional function
- The chain rule can be expressed as a **local computation**
- Think about it as a **computational graph**



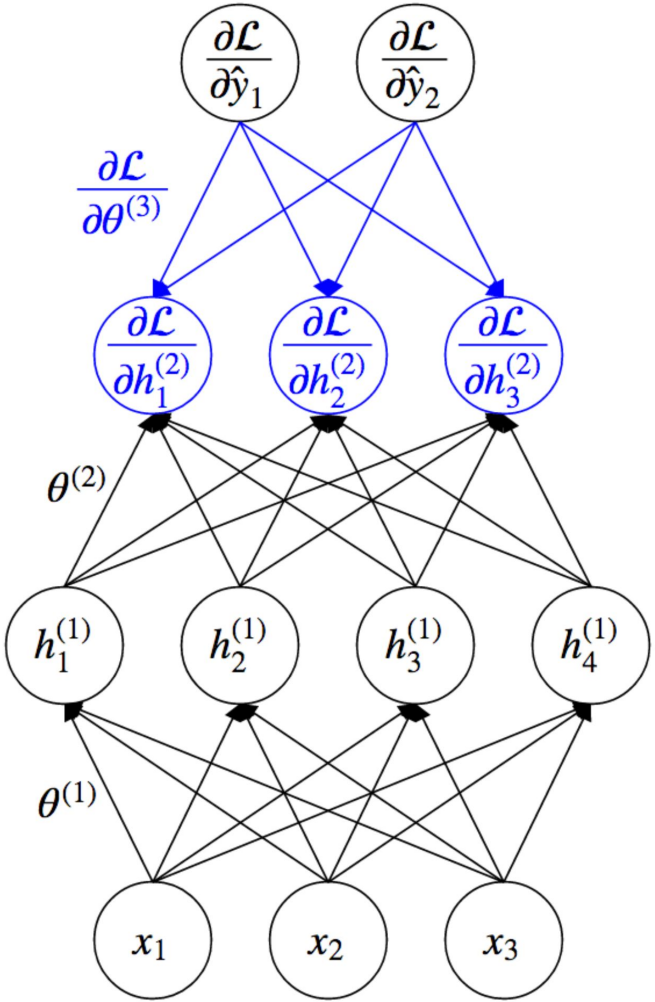
Slide credit: Junhyuk Oh, Jacob Abernethy

Backpropagation (for deep neural nets)



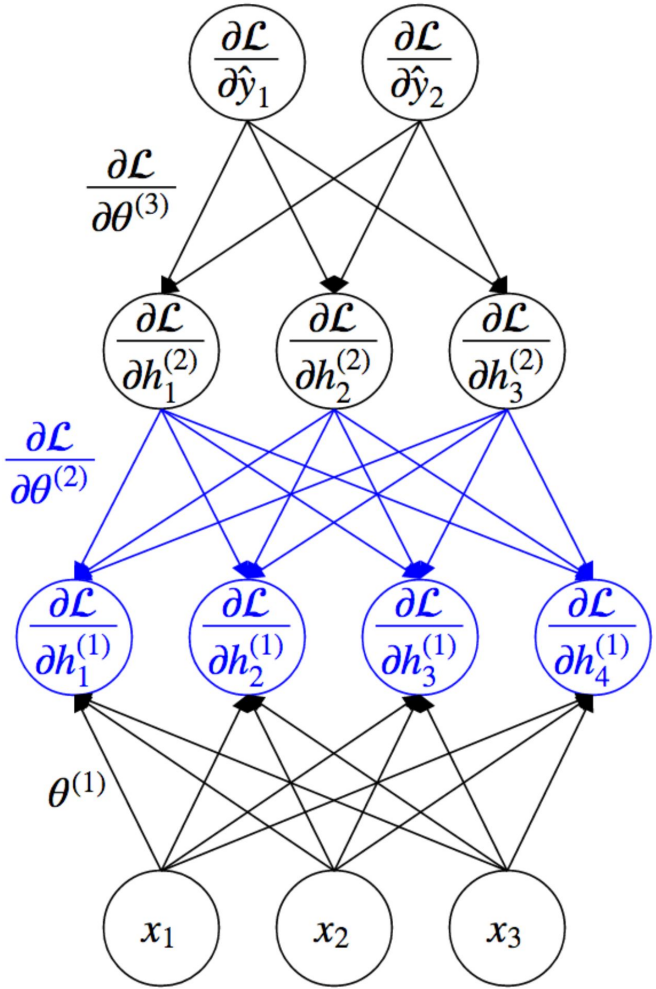
Slide credit: Junhyuk Oh, Jacob Abernethy

Backpropagation (for deep neural nets)



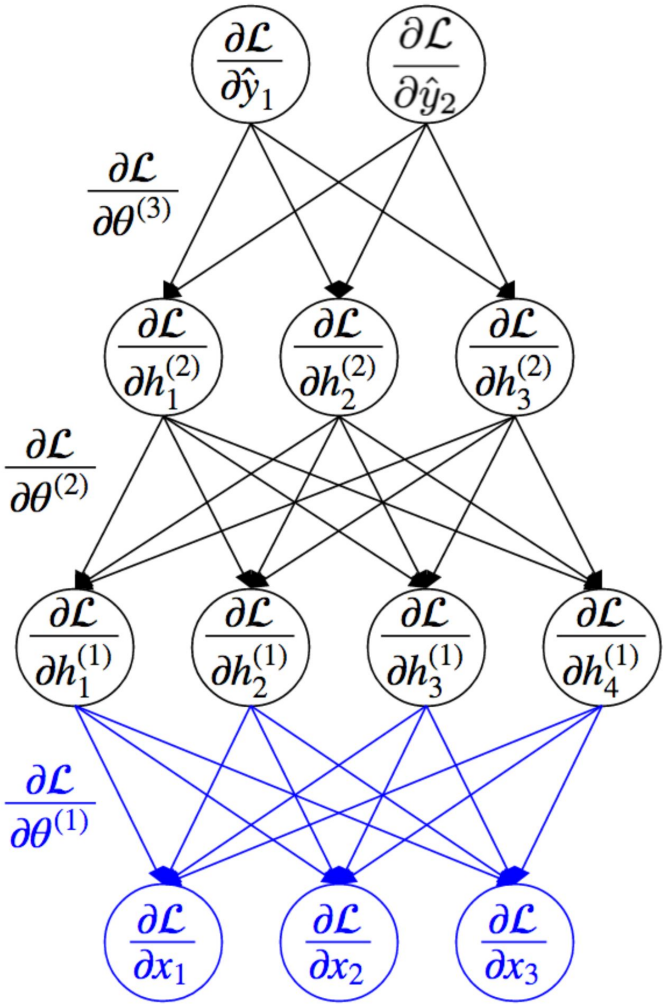
Slide credit: Junhyuk Oh, Jacob Abernethy

Backpropagation (for deep neural nets)



Slide credit: Junhyuk Oh, Jacob Abernethy

Backpropagation (for deep neural nets)



Slide credit: Junhyuk Oh, Jacob Abernethy

Back-Propagation Algorithm

- Compute $\nabla_{\mathbf{y}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial y_1}, \ldots, \frac{\partial \mathcal{L}}{\partial y_n} \right]$ directly from the loss function.
- For each layer (from top to bottom) with output $\mathbf{h}$, input $\mathbf{x}$, and weights $\mathbf{W}$,
  - Assuming that $\nabla_{\mathbf{h}} \mathcal{L}$ is given, compute gradients using the **chain rule** as follows:

$$\nabla_{\mathbf{W}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L} \nabla_{\mathbf{W}} \mathbf{h}$$
$$\nabla_{\mathbf{x}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L} \nabla_{\mathbf{x}} \mathbf{h}$$

Slide credit: Junhyuk Oh, Jacob Abernethy

Practice: Linear

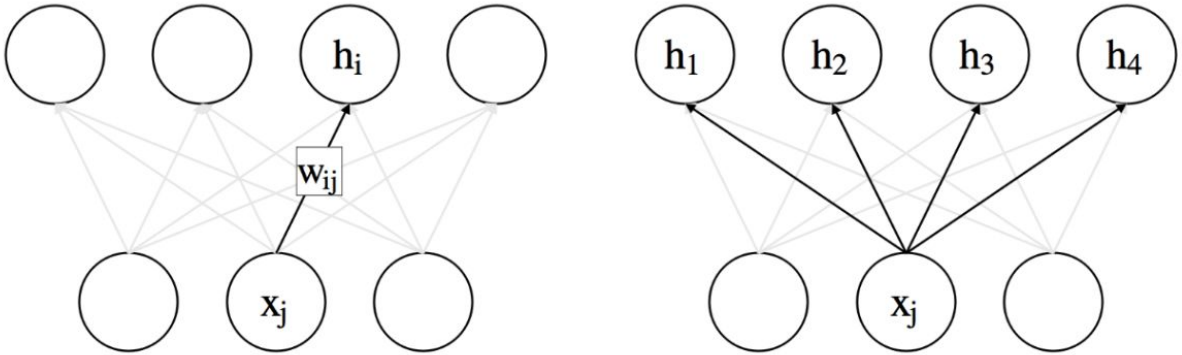
- Forward: $h_i = \sum_j w_{ij} x_j + b_i \iff \mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$
- Gradient w.r.t. parameters

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial h_i}\frac{\partial h_i}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial h_i} x_j \iff \nabla_{\mathbf{W}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L} \mathbf{x}^\top$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L}$$

- Gradient w.r.t. inputs

$$\frac{\partial \mathcal{L}}{\partial x_j} = \sum_i \frac{\partial \mathcal{L}}{\partial h_i}\frac{\partial h_i}{\partial x_j} = \sum_i \frac{\partial \mathcal{L}}{\partial h_i} w_{ij} \iff \nabla_{\mathbf{x}} \mathcal{L} = \mathbf{W}^\top \nabla_{\mathbf{h}} \mathcal{L}$$



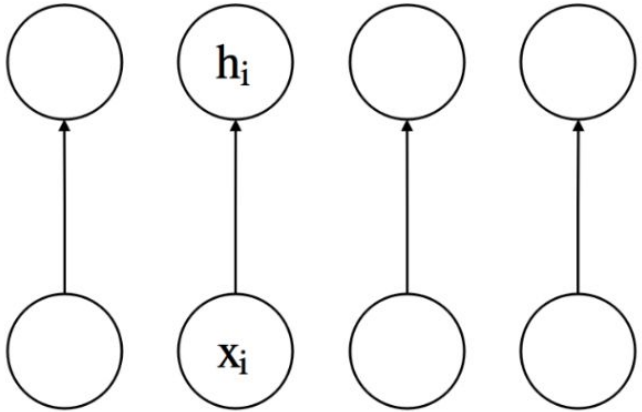
Slide credit: Junhyuk Oh, Jacob Abernethy

Practice: Non-Linear Activation (Sigmoid)

- Forward: $h_i = \sigma(x_i) = \frac{1}{1+\exp(-x_i)} \iff \mathbf{h} = \sigma(\mathbf{x})$
- Gradient w.r.t. inputs

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial h_i}\frac{\partial h_i}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial h_i}\sigma(x_i)(1-\sigma(x_i)) = \frac{\partial \mathcal{L}}{\partial h_i}h_i(1-h_i)$$

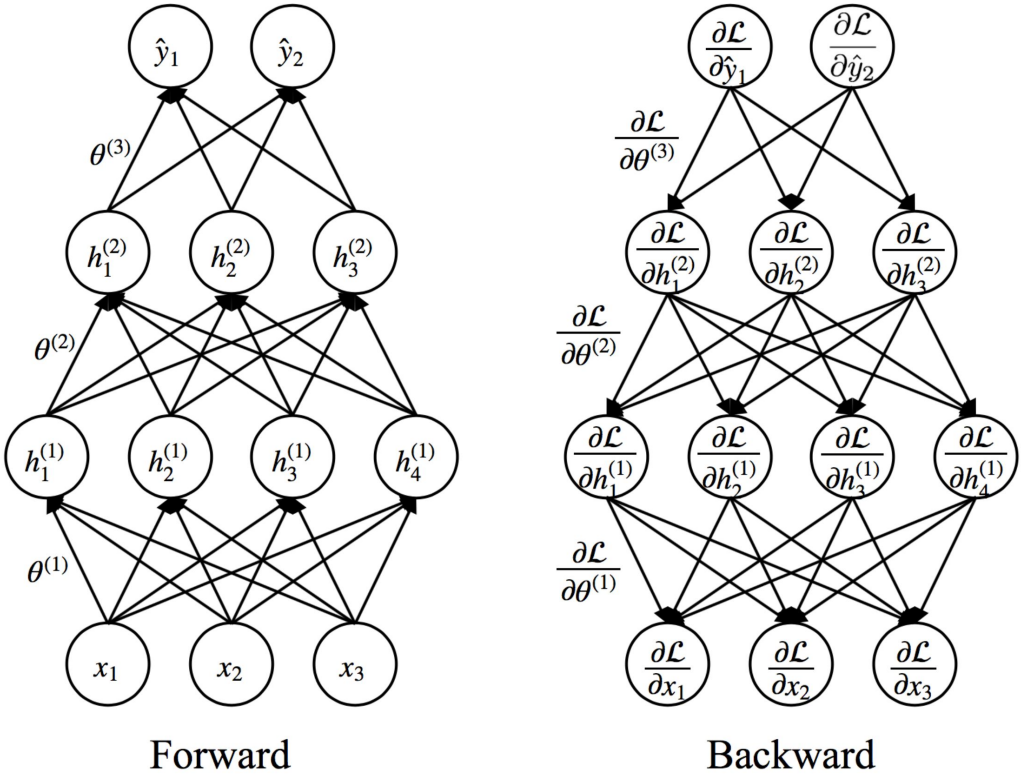
$$\nabla_{\mathbf{x}}\mathcal{L} = \nabla_{\mathbf{h}}\mathcal{L} \odot \mathbf{h} \odot (\mathbf{1}-\mathbf{h})$$



Slide credit: Junhyuk Oh, Jacob Abernethy

Backpropagation for deep neural nets

- Computing gradient via **chain rule** for compositional function
- The chain rule can be expressed as a **local computation**
- Think about it as a **computational graph**



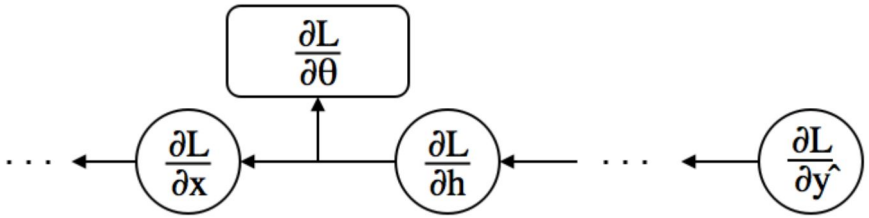
Slide credit: Junhyuk Oh, Jacob Abernethy

Idea of Back-Propagation

- Assuming that $\frac{\partial \mathcal{L}}{\partial h}$ is given, use the **chain rule** to compute the gradients

$$\frac{\partial \mathcal{L}}{\partial \theta} = \underbrace{\frac{\partial \mathcal{L}}{\partial h}}_{\text{given}} \underbrace{\frac{\partial h}{\partial \theta}}_{\text{easy}}, \frac{\partial \mathcal{L}}{\partial x} = \underbrace{\frac{\partial \mathcal{L}}{\partial h}}_{\text{given}} \underbrace{\frac{\partial h}{\partial x}}_{\text{easy}}$$

- We need only $\frac{\partial \mathcal{L}}{\partial \theta}$ for gradient descent. Why compute $\frac{\partial \mathcal{L}}{\partial x}$?
  - The previous layer needs it because x is the output of the previous layer.



Outline

- Backpropagation
- Optimization
- Regularization

Optimization

- Stochastic Gradient Descent
- Momentum Method
- Adaptive Learning Method(AdaGrad, RMSProp, Adam)
- Batch Normalization

Batch Gradient Descent

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1: **while** stopping criteria not met **do**

2: Compute gradient estimate over N examples:

3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$

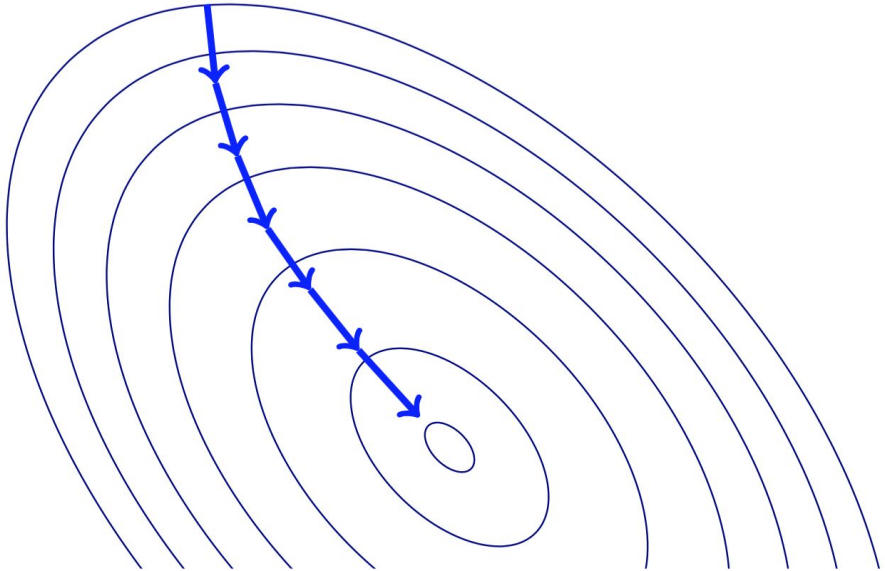
4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$

5: **end while**

- Positive: Gradient estimates are stable
- Negative: Need to compute gradients over the entire training for one update

Slide credit: Shubhendu Trivedi, Risi Kondor

Gradient Descent



Slide credit: Shubhendu Trivedi, Risi Kondor

Stochastic Gradient Descent

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1: **while** stopping criteria not met **do**

2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set

3: Compute gradient estimate:

4: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

5: Apply Update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

6: **end while**

- ϵ_k is learning rate at step k
- Sufficient condition to guarantee convergence:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

Slide credit: Shubhendu Trivedi, Risi Kondor

Learning Rate Schedule

- In practice the learning rate is decayed linearly till iteration τ

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \text{ with } \alpha = \frac{k}{\tau}$$

- τ is usually set to the number of iterations needed for a large number of passes through the data
- ϵ_τ is roughly set to be 1% of ϵ_0

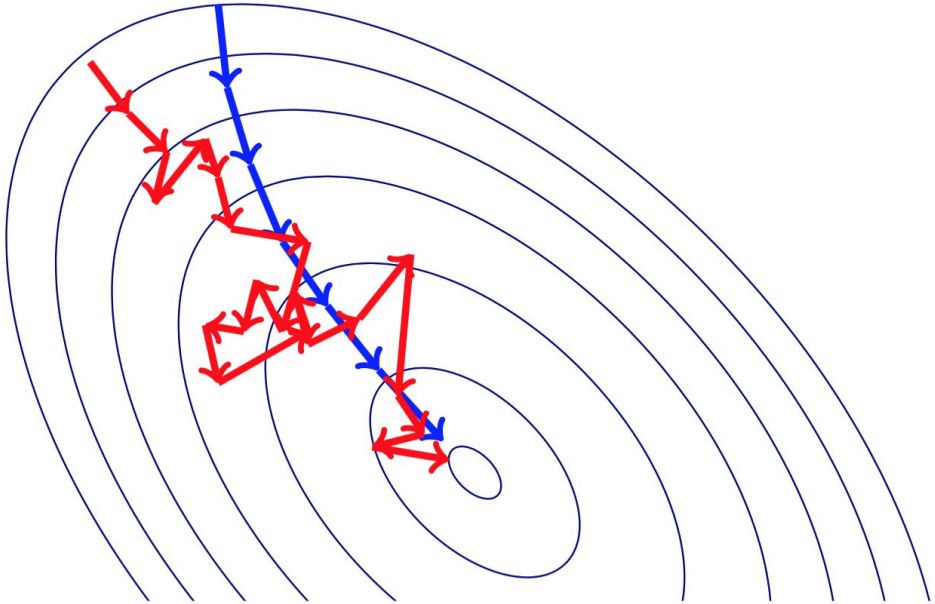
Slide credit: Shubhendu Trivedi, Risi Kondor

Minibatching

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches
- Advantage: Computation time per update does not depend on number of training examples N
- This allows convergence on extremely large datasets
  - See: Large Scale Learning with Stochastic Gradient Descent by Leon Bottou

Slide credit: Shubhendu Trivedi, Risi Kondor

Stochastic Gradient Descent

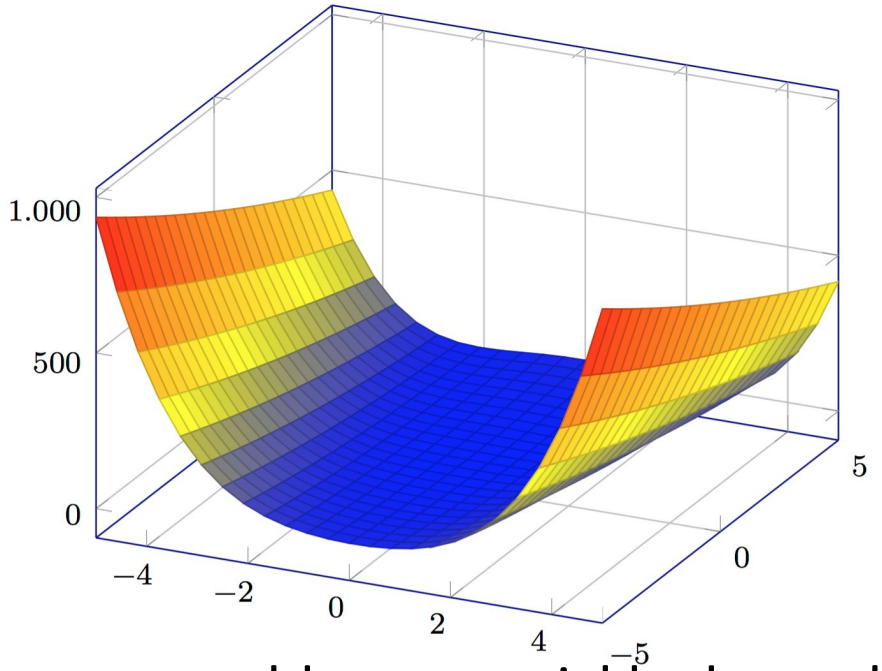


Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
  - Error surface has high curvature
  - We get small but consistent gradients
  - The gradients are very noisy

Momentum



- Gradient descent would move quickly down the walls, but very slowly through the valley floor

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

- How do we try and solve this problem?
- Introduce a new variable v, the velocity
- We think of v as the direction and speed by which the parameters move as the learning dynamics progresses
- The velocity is an exponentially decaying moving average of the negative gradients

$$\alpha \in [0, 1) \qquad \mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$

- Update rule: $\theta \leftarrow \theta + \mathbf{v}$

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

- Let's look at the velocity term

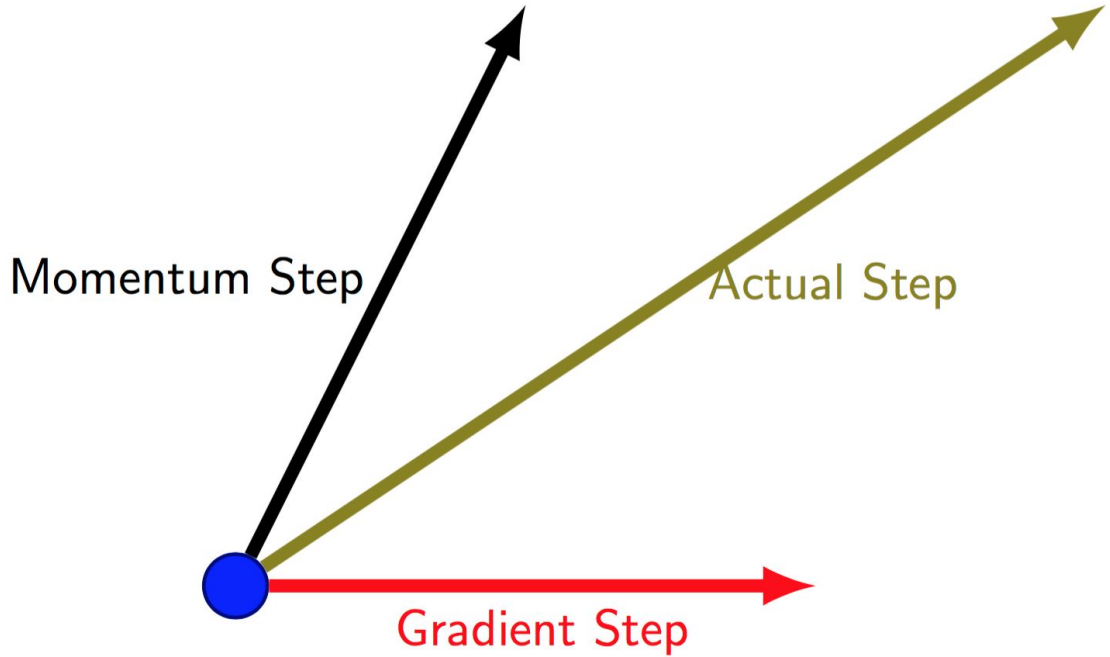
$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$

- The velocity accumulates the previous gradients What is the role of α?
- If α is larger than ε the current update is more affected by the previous gradients
- Usually values for α are set high ≈ 0.8, 0.9

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$



Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum: Step Sizes

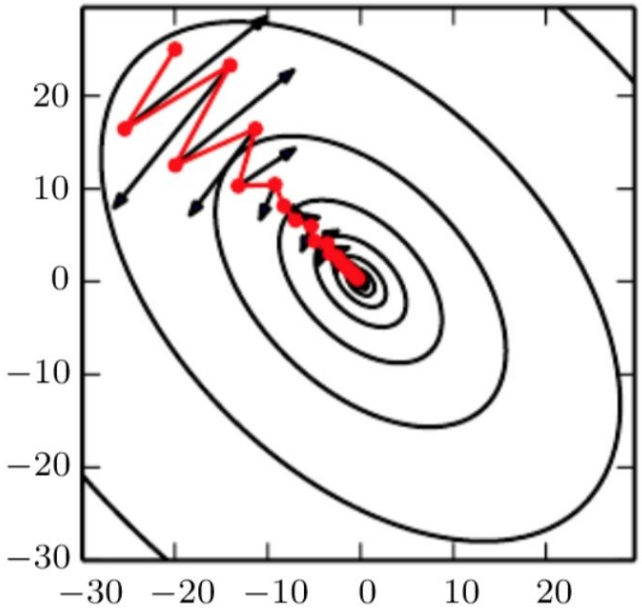
- In SGD, the step size was the norm of the gradient scaled by the learning rate $\epsilon\|\mathbf{g}\|$. Why?
- While using momentum, the step size will also depend on the norm and alignment of a sequence of gradients
- For example, if at each step we observed $\mathbf{g}$, the step size would be (exercise!):

$$\epsilon \frac{\|\mathbf{g}\|}{1-\alpha}$$

- If $\alpha = 0.9 \implies$ multiply the maximum speed by 10 relative to the current gradient direction

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum



- Illustration of how momentum traverses such an error surface better compared to gradient descent
- Visualization of SGD with momentum

Slide credit: Shubhendu Trivedi, Risi Kondor

Momentum

Algorithm 2 Stochastic Gradient Descent with Momentum

Require: Learning rate ϵ_k
Require: Momentum Parameter α
Require: Initial Parameter θ
Require: Initial Velocity $\mathbf{v}$

while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
  Compute gradient estimate:
  $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
  Compute the velocity update:
  $\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\hat{\mathbf{g}}$
  Apply Update: $\theta \leftarrow \theta + \mathbf{v}$
end while

Note: minibatch SGD version with momentum is a popular choice.

Slide credit: Shubhendu Trivedi, Risi Kondor

Adaptive Learning Methods: Motivation

- Till now we assign the same learning rate to all features
- If the features vary in importance and frequency, why is this a good idea?
- It's probably not!

Slide credit: Shubhendu Trivedi, Risi Kondor

AdaGrad

- Idea: Downscale a model parameter by square-root of sum of squares of all its historical values
- Parameters that have large partial derivative of the loss – learning rates for them are rapidly declined
- Some interesting theoretical properties

Slide credit: Shubhendu Trivedi, Risi Kondor

AdaGrad

Algorithm 4 AdaGrad

Require: Global Learning rate ϵ, Initial Parameter θ, δ

Initialize $\mathbf{r} = 0$

1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
3: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
4: Accumulate: $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
5: Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
6: Apply Update: $\theta \leftarrow \theta + \Delta\theta$
7: **end while**

Slide credit: Shubhendu Trivedi, Risi Kondor

RMSProp

- AdaGrad is good when the objective is convex.
- AdaGrad might shrink the learning rate too aggressively, we want to keep the history in mind
- We can adapt it to perform better in non-convex settings by accumulating an exponentially decaying average of the gradient
- This is an idea that we use again and again in Neural Networks. Currently has about 500 citations on scholar, but was proposed in a slide in Geoffrey Hinton's coursera course.

Slide credit: Shubhendu Trivedi, Risi Kondor

RMSProp

Algorithm 5 RMSProp

Require: Global Learning rate ϵ, decay parameter ρ, δ

Initialize $\mathbf{r} = 0$

1: **while** stopping criteria not met **do**

2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set

3: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

4: Accumulate: $\mathbf{r} \leftarrow \rho\mathbf{r} + (1-\rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

5: Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$

6: Apply Update: $\theta \leftarrow \theta + \Delta\theta$

7: **end while**

Slide credit: Shubhendu Trivedi, Risi Kondor

Adam

- We could have used RMSProp with momentum
- Use of Momentum with rescaling is not well motivated
- Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments

Adam

Algorithm 7 ADAptive Moments

Require: ϵ (set to 0.0001), decay rates ρ_1 (set to 0.9), ρ_2 (set to 0.9), θ, δ

Initialize moments variables $\mathbf{s} = 0$ and $\mathbf{r} = 0$, time step t $= 0$

1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
3: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
4: $t \leftarrow t + 1$
5: Update: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$
6: Update: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
7: Correct Biases: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$
8: Compute Update: $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\delta}$
9: Apply Update: $\theta \leftarrow \theta + \Delta\theta$
10: **end while**

Slide credit: Shubhendu Trivedi, Risi Kondor

Optimization

SGD: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \mathbf{v}$

AdaGrad: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ then $\theta \leftarrow \theta + \Delta\theta$

RMSProp: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1-\rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

Adam:

$$\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1-\rho_1)\hat{\mathbf{g}}$$
$$\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1-\rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$$
$$\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t} \text{ then } \Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta} \text{ then } \theta \leftarrow \theta + \Delta\theta$$

- Visualization:
  - Visualization of optimizers
  - Distill: why momentum really works

Slide credit: Shubhendu Trivedi, Risi Kondor

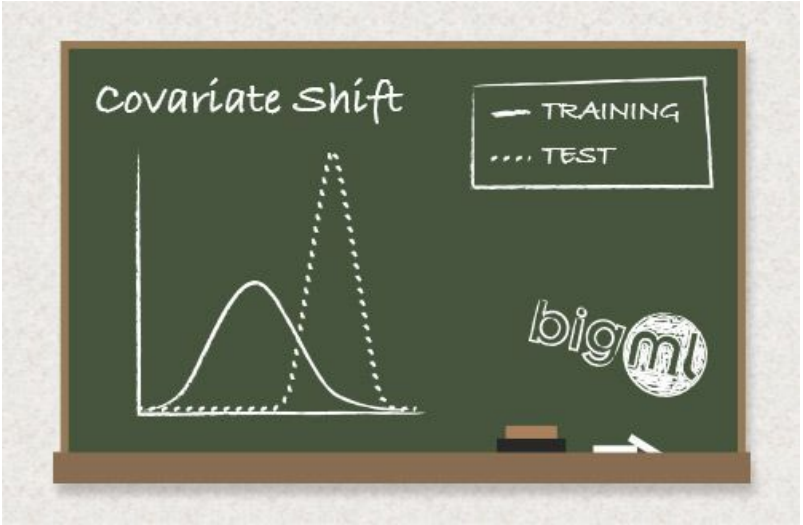
Batch Normalization: motivation

- We have a recipe to compute gradients (Backpropagation), and update every parameter (we saw several methods)
- Implicit assumption: Other layers don't change i.e. other functions are fixed
- In practice: We update all layers simultaneously. This can give rise to unexpected difficulties

Slide credit: Shubhendu Trivedi, Risi Kondor

Batch Normalization Motivation

- Covariate shift:
  - Training and test input follow different distributions But functional relation remains the same
- Problem: internal covariance shift
  - Change of distribution in activation across layers
  - Sensitive to saturations
  - Change in optimal learning rate => need really small steps



Slide credit: Nan Tian

Batch Normalization: forward prop

- Normalize distribution of each input feature in each layer across each minibatch to N(0, 1)
- Learn the scale and shift

- After training, at test time: Use running averages of μ and σ collected during training, use these for evaluating new input x

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

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

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Batch Normalization: Backpropagation

- It's differentiable via chain rule

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Batch Normalization

- Method to reparameterize a deep network to reduce co-ordination of update across layers
- Can be applied to input layer, or any hidden layer
- Let H be a design matrix having activations in any layer for m examples in the mini-batch

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1k} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{m1} & h_{m2} & h_{m3} & \dots & h_{mk} \end{bmatrix}$$

Slide credit: Shubhendu Trivedi, Risi Kondor

Batch Normalization

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1k} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \\ h_{m1} & h_{m2} & h_{m3} & \dots & h_{mk} \end{bmatrix}$$

- Each row represents all the activations in layer for one example
- Idea: Replace H by H′ such that: $H' = \frac{H - \mu}{\sigma}$
- μ is mean of each unit and σ the standard deviation

Slide credit: Shubhendu Trivedi, Risi Kondor

Batch Normalization

- μ is a vector with μ_j the j-th column mean
- σ is a vector with σ_j the j-th column standard deviation

$$\mu = \frac{1}{m}\sum_j H_{:,j} \qquad \sigma = \sqrt{\delta + \frac{1}{m}\sum_j (H-\mu)_j^2}$$

- Replace H by H′ such that: $H' = \frac{H-\mu}{\sigma}$

- We then operate on H′ as before ==> we backpropagate through the normalized activations

Slide credit: Shubhendu Trivedi, Risi Kondor

Batch Normalization

- The update will never act to only increase the mean and standard deviation of any activation
- Previous approaches added penalties to cost or per layer to encourage units to have standardized outputs
- Batch normalization makes the reparameterization easier
- At test time: Use running averages of μ and σ collected during training, use these for evaluating new input x

Slide credit: Shubhendu Trivedi, Risi Kondor

Problems with Back Propagation

- Typically requires lots of labeled data
  - When there is a large amount of labeled data, backpropagation works very well.

- Given limited amounts of labeled data, training via backpropagation does not work well (if randomly initialized)
  - Deep networks trained with backpropagation (without unsupervised pretraining) sometimes perform worse than shallow networks – Overfitting

Problems with Back Propagation

- Gradient is progressively getting more dilute
  - Below top few layers, correction signal from supervision may get smaller
- May get stuck in local minima
  - Especially since they start out far from 'good' regions (i.e., random initialization)
- **Given all these said, DNN via stochastic gradient descent with sufficient amounts of labeled data.**

Outline

- Backpropagation
- Optimization
- Regularization

Regularization

- In general: any method to prevent overfitting or help the optimization
- Specifically: additional terms in the training optimization objective to prevent overfitting or help the optimization

Slide credit: Yingyu Liang

Overfitting

- Empirical loss and expected loss are different
- Smaller the data set, larger the difference between the two
- Larger the hypothesis class, easier to find a hypothesis that fits the difference between the two
  - Thus has small training error but large test error (overfitting)

Slide credit: Yingyu Liang

Preventing Overfitting: overview

- Larger data size set always helps
- Reducing the model capacity (e.g., # of hidden layers, # of hidden units) helps
- Classical regularization (e.g., L2 regularization): some principal ways to constrain hypotheses
- Other types of regularization: data augmentation, early stopping, etc.

Slide credit: Yingyu Liang

L2 Regularization

- perhaps the most common form of regularization
- penalize the squared magnitude of all parameters directly in the objective. $\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \frac{\alpha}{2}||\theta||_2^2$
- intuitive interpretation: heavily penalizing peaky weight vectors and preferring diffuse weight vectors.
- due to multiplicative interactions between weights and inputs this has the appealing property of encouraging the network to use all of its inputs a little rather than some of its inputs a lot.

l_2 regularization

$$\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \frac{\alpha}{2}||\theta||_2^2$$

- Effect on (stochastic) gradient descent
- Effect on the optimal solution

Slide credit: Yingyu Liang

Effect on gradient descent

- Gradient of regularized objective

$$\nabla \hat{L}_R(\theta) = \nabla \hat{L}(\theta) + \alpha\theta$$

- Gradient descent update

$$\theta \leftarrow \theta - \eta \nabla \hat{L}_R(\theta) = \theta - \eta\, \nabla \hat{L}(\theta) - \eta\alpha\theta = (1 - \eta\alpha)\theta - \eta\, \nabla \hat{L}(\theta)$$

- Terminology: weight decay

Slide credit: Yingyu Liang

Effect on the optimal solution

- Consider a quadratic approximation around θ^*

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + (\theta - \theta^*)^T \nabla \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

- Since θ^* is optimal, $\nabla \hat{L}(\theta^*) = 0$

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

$$\nabla \hat{L}(\theta) \approx H(\theta - \theta^*)$$

Slide credit: Yingyu Liang

Effect on the optimal solution

- Gradient of regularized objective

$$\nabla \hat{L}_R(\theta) \approx H(\theta - \theta^*) + \alpha\theta$$

- On the optimal θ_R^*

$$0 = \nabla \hat{L}_R(\theta_R^*) \approx H(\theta_R^* - \theta^*) + \alpha\theta_R^*$$

$$\theta_R^* \approx (H + \alpha I)^{-1} H\theta^*$$

Slide credit: Yingyu Liang

Effect on the optimal solution

- The optimal

$$\theta_R^* \approx (H + \alpha I)^{-1} H \theta^*$$

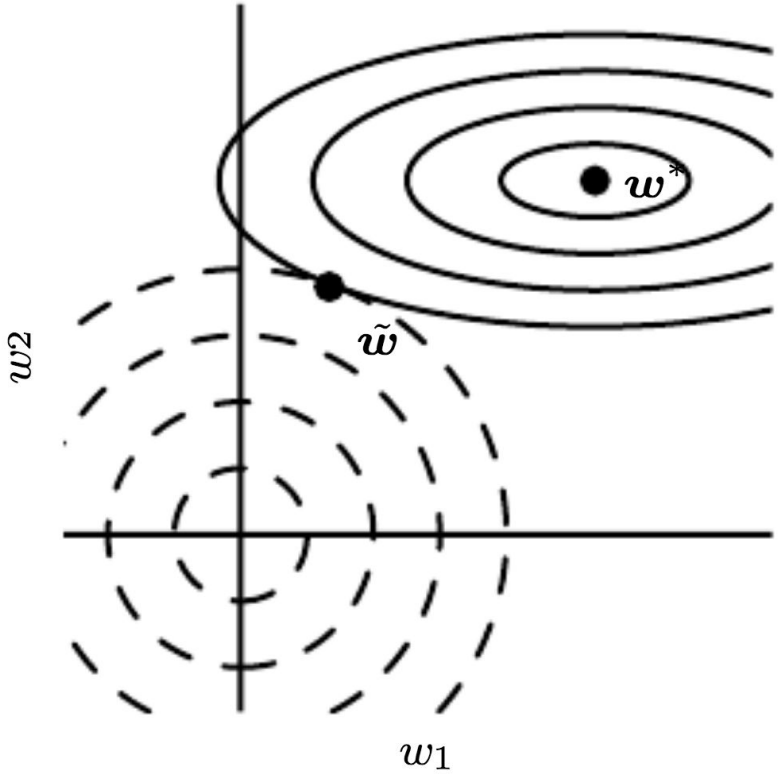
- Suppose H has eigen-decomposition $H = Q \Lambda Q^T$

$$\theta_R^* \approx (H + \alpha I)^{-1} H \theta^* = Q(\Lambda + \alpha I)^{-1} \Lambda Q^T \theta^*$$

- Effect: rescale along eigenvectors of H

Slide credit: Yingyu Liang

Effect on the optimal solution



$$\theta^* = w^*, \theta_R^* = \tilde{w}$$

Figure from *Deep Learning*, Goodfellow, Bengio and Courville

L1 Regularization

- another relatively common form of regularization
  $\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \alpha||\theta||_1$
- leads the weight vectors to become sparse during optimization (i.e. very close to exactly zero). In other words, neurons with L1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to the “noisy” inputs.
- In practice, if you are not concerned with explicit feature selection, L2 regularization can be expected to give superior performance over L1.

l_1 regularization

$$\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \alpha||\theta||_1$$

- Effect on (stochastic) gradient descent
- Effect on the optimal solution

Slide credit: Yingyu Liang

Effect on gradient descent

- Gradient of regularized objective

$$\nabla \hat{L}_R(\theta) = \nabla \hat{L}(\theta) + \alpha \operatorname{sign}(\theta)$$

  where sign applies to each element in θ

- Gradient descent update

$$\theta \leftarrow \theta - \eta \nabla \hat{L}_R(\theta) = \theta - \eta \nabla \hat{L}(\theta) - \eta\alpha \operatorname{sign}(\theta)$$

Slide credit: Yingyu Liang

Effect on the optimal solution

- Consider a quadratic approximation around θ^*

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + (\theta - \theta^*)^T \nabla \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

- Since θ^* is optimal, $\nabla \hat{L}(\theta^*) = 0$

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$$

Slide credit: Yingyu Liang

Effect on the optimal solution

- Further assume that H is diagonal and positive $(H_{ii} > 0, \forall i)$
  - not true in general but assume for getting some intuition
- The regularized objective is (ignoring constants)

$$\hat{L}_R(\theta) \approx \sum_i \frac{1}{2} H_{ii}(\theta_i - \theta_i^*)^2 + \alpha |\theta_i|$$

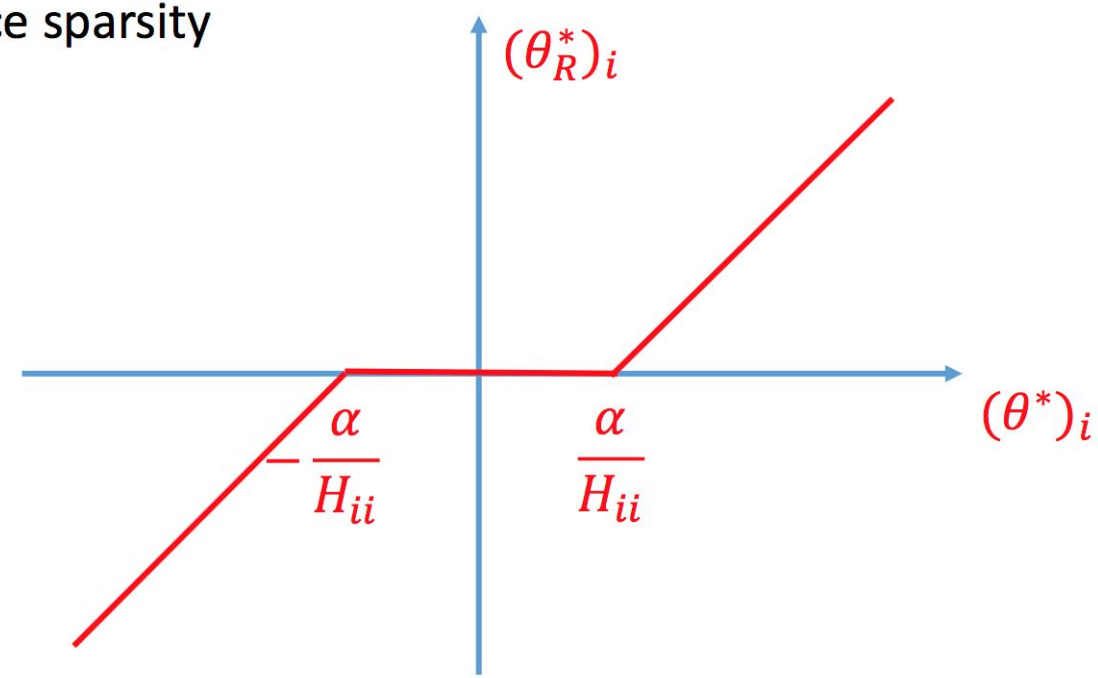
- The optimal θ_R^*

$$(\theta_R^*)_i \approx \begin{cases} \max\left\{\theta_i^* - \frac{\alpha}{H_{ii}}, 0\right\} & \text{if } \theta_i^* \geq 0 \\ \min\left\{\theta_i^* + \frac{\alpha}{H_{ii}}, 0\right\} & \text{if } \theta_i^* < 0 \end{cases}$$

Slide credit: Yingyu Liang

Effect on the optimal solution

- Effect: induce sparsity



Slide credit: Yingyu Liang

Early Stopping

- Idea: don’t train the network to too small training error
- Recall overfitting: Larger the hypothesis class, easier to find a hypothesis that fits the difference between the two
- Prevent overfitting: do not push the hypothesis too much; use validation error to decide when to stop

Slide credit: Yingyu Liang

Early Stopping

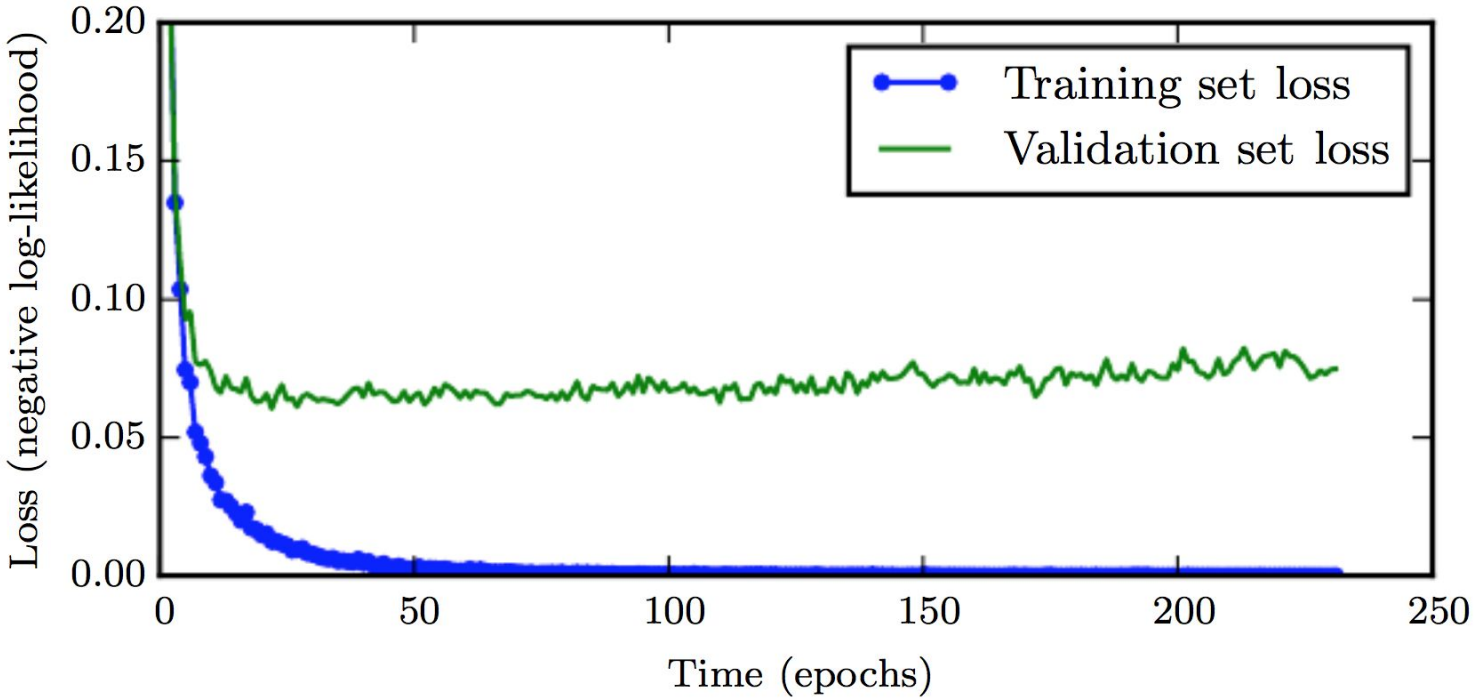


Figure from *Deep Learning*, Goodfellow, Bengio and Courville

Early Stopping

- When training, also output validation error
- Every time validation error improved, store a copy of the weights
- When validation error not improved for some time, stop
- Return the copy of the weights stored

Slide credit: Yingyu Liang

Early Stopping

- “Implicit” hyperparameter selection:
  - training step is the hyperparameter
- Advantage
  - Efficient:along with training; only store an extra copy of weights
  - Simple: no change to the model/algo
- Disadvantage: need validation data

Slide credit: Yingyu Liang

Early Stopping

- Strategy to get rid of the disadvantage
  - After early stopping of the first run, train a second run and reuse validation data
- How to reuse validation data
  - Start fresh, train with both training data and validation data up to the previous number of epochs
  - Start from the weights in the first run, train with both training data and validation data util the validation loss < the training loss at the early stopping point

Slide credit: Yingyu Liang

Dropout

- Randomly select weights to update
- More precisely, in each update step
  - Randomly sample a different binary mask to all the input and hidden units
  - Multiple the mask bits with the units and do the update as usual
- Typical dropout probability: 0.2 for input and 0.5 for hidden units

Slide credit: Yingyu Liang

Dropout

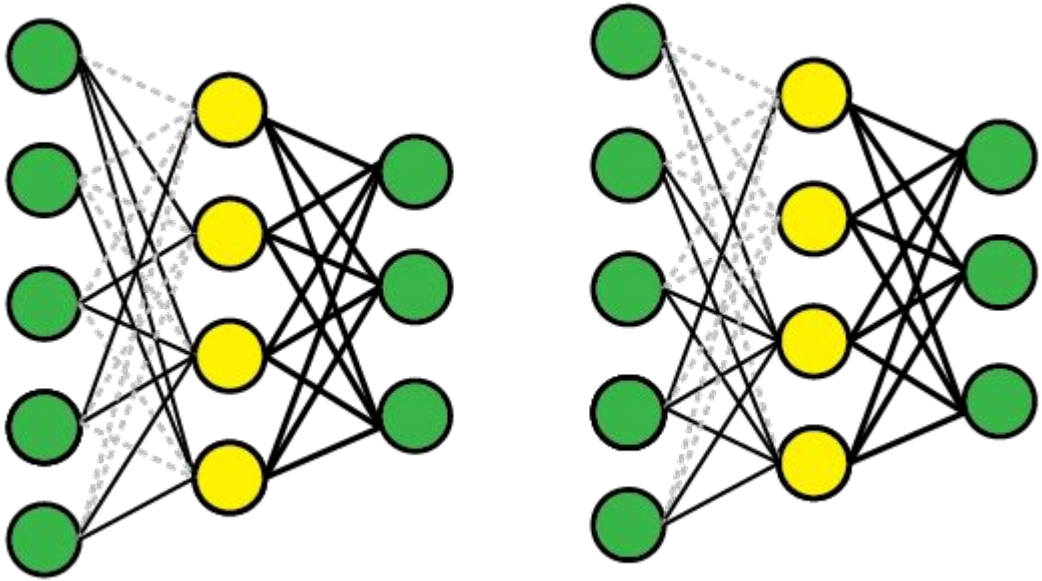


Figure from *Deep Learning*, Goodfellow, Bengio and Courville

Dropout

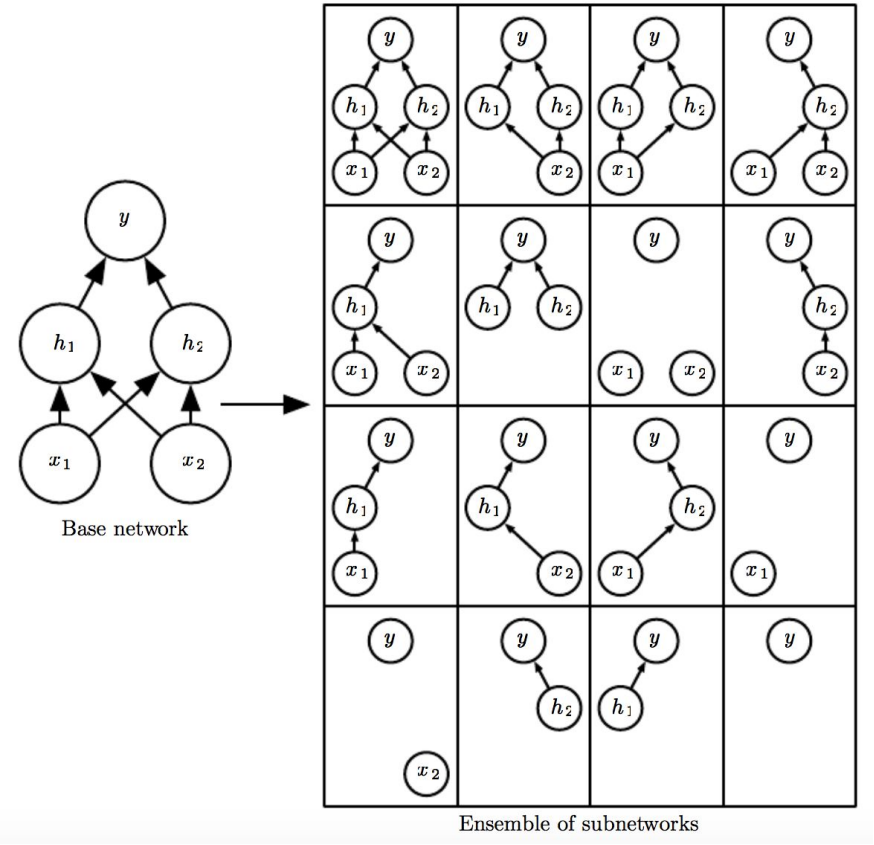


Figure from *Deep Learning*, Goodfellow, Bengio and Courville

Summary

- Backpropagation:
  - Computing gradient via chain rule for compositional function
  - Think about it as a computational graph
- Optimization:
  - SGD with minibatch
  - Adaptive learning rates are quite useful: AdaGrad, RMSprop, Adam, etc.
  - Batch normalization
- Avoiding overfitting and regularization:
  - Control the model capacity (# layers, #hidden units, etc.)
  - Weight regularization (L2 or L1)
  - Early stopping
  - Dropout

Next Lecture

Convolutional Neural Networks

Early Stopping as Regularizer

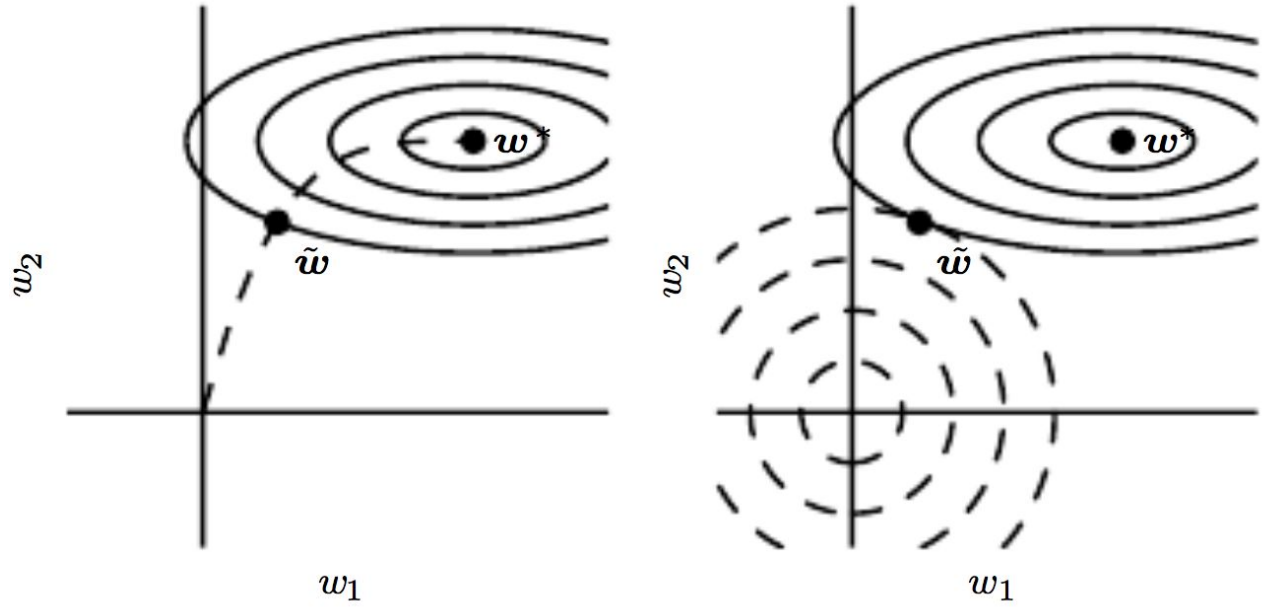


Figure from *Deep Learning*, Goodfellow, Bengio and Courville

Extra slides

Intuition

- Consider a second order approximation of our cost function (which is a function composition) around current point $\theta^{(0)}$:

$$J(\theta) \approx J(\theta^{(0)}) + (\theta - \theta^{(0)})^T \mathbf{g} + \frac{1}{2}(\theta - \theta^{(0)})^T H (\theta - \theta^{(0)})$$

- $\mathbf{g}$ is gradient and H the Hessian at $\theta^{(0)}$
- If ϵ is the learning rate, the new point

$$\theta = \theta^{(0)} - \epsilon \mathbf{g}$$

Slide credit: Shubhendu Trivedi, Risi Kondor

Intuition

- Plugging our new point, $\theta = \theta^{(0)} - \epsilon \mathbf{g}$ into the approximation:

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

- There are three terms here:
  - Value of function before update
  - Improvement using gradient (i.e. first order information)
  - Correction factor that accounts for the curvature of the function

Slide credit: Shubhendu Trivedi, Risi Kondor

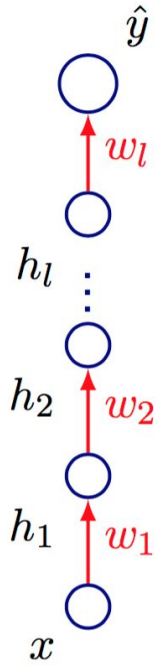
Intuition

$$J(\theta^{(0)} - \epsilon \mathbf{g}) = J(\theta^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2} \mathbf{g}^T H \mathbf{g}$$

- Observations:
  - $\mathbf{g}^T H \mathbf{g}$ too large: Gradient will start moving upwards
  - $\mathbf{g}^T H \mathbf{g} = 0$: J will decrease for even large ϵ
  - Optimal step size $\epsilon^* = \mathbf{g}^T \mathbf{g}$ for zero curvature, $\epsilon^* = \frac{\mathbf{g}^T \mathbf{g}}{\mathbf{g}^T H \mathbf{g}}$ to take into account curvature
- Conclusion: Just neglecting second order effects can cause problems (remedy: second order methods). What about higher order effects?

Slide credit: Shubhendu Trivedi, Risi Kondor

Higher Order Effects: Toy Model



- Just one node per layer, no non-linearity
- $\hat{y}$ is linear in x but non-linear in w_i

Slide credit: Shubhendu Trivedi, Risi Kondor

Higher Order Effects: Toy Model

- Suppose $\delta = 1$, so we want to decrease our output $\hat{y}$
- Usual strategy:
  - Using backprop find $\mathbf{g} = \nabla_{\mathbf{w}}(\hat{y} - y)^2$
  - Update weights $\mathbf{w} := \mathbf{w} - \epsilon\mathbf{g}$
- The first order Taylor approximation (in previous slide) says the cost will reduce by $\epsilon\mathbf{g}^T\mathbf{g}$
- If we need to reduce cost by 0.1, then learning rate should be $\frac{0.1}{\mathbf{g}^T\mathbf{g}}$

Slide credit: Shubhendu Trivedi, Risi Kondor

Higher Order Effects: Toy Model

- The new $\hat{y}$ will however be:

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)\dots(w_l - \epsilon g_l)$$

- Contains terms like $\epsilon^3 g_1 g_2 g_3 w_4 w_5 \dots w_l$
- If weights $w_4, w_5, \dots, w_l$ are small, the term is negligible. But if large, it would explode
- Conclusion: Higher order terms make it very hard to choose the right learning rate
- Second Order Methods are already expensive, nth order methods are hopeless. Solution?

Slide credit: Shubhendu Trivedi, Risi Kondor