ELEC 576: Neural Networks & Backpropagation Lecture 3

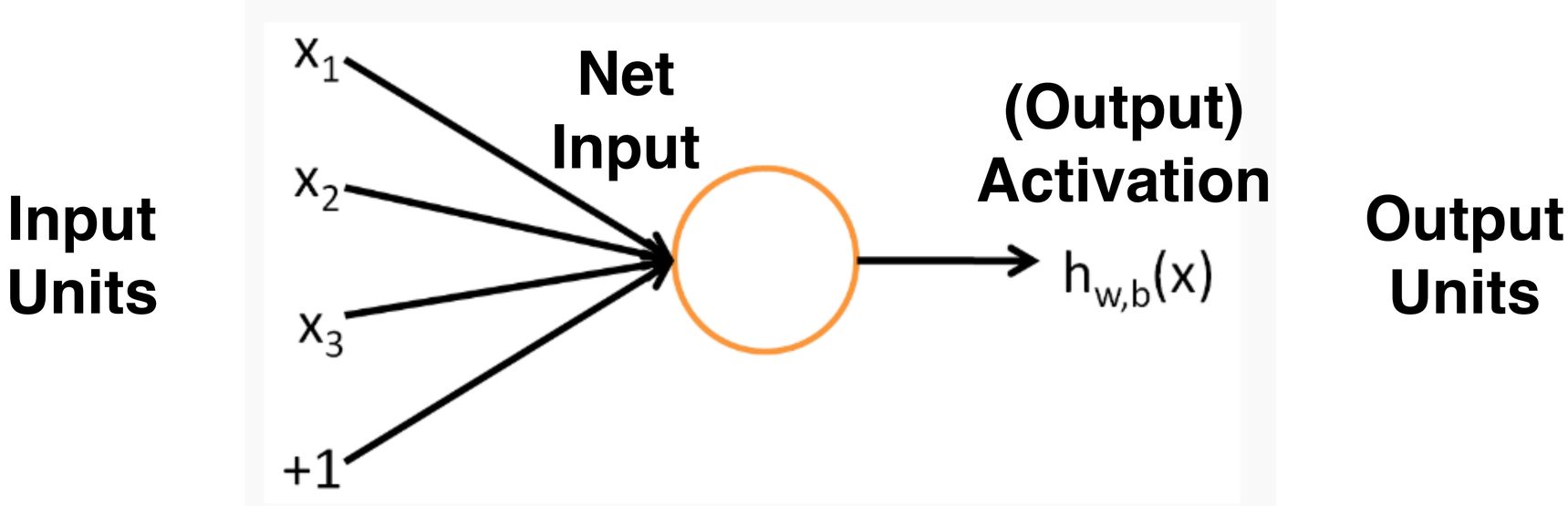
Ankit B. Patel
Baylor College of Medicine (Neuroscience Dept.)
Rice University (ECE Dept.)

Outline

- Neural Networks
  - Definition of NN and terminology
- Review of (Old) Theoretical Results about NNs
  - Intuition for why compositions of nonlinear functions are more expressive
  - Expressive power theorems [McC-Pitts, Rosenblatt, Cybenko]
- Backpropagation algorithm (Gradient Descent + Chain Rule)
  - History of backprop summary
  - Gradient descent (Review).
  - Chain Rule (Review).
  - Backprop
- Intro to Convnets
  - Convolutional Layer, ReLu, Max-Pooling

Neural Networks

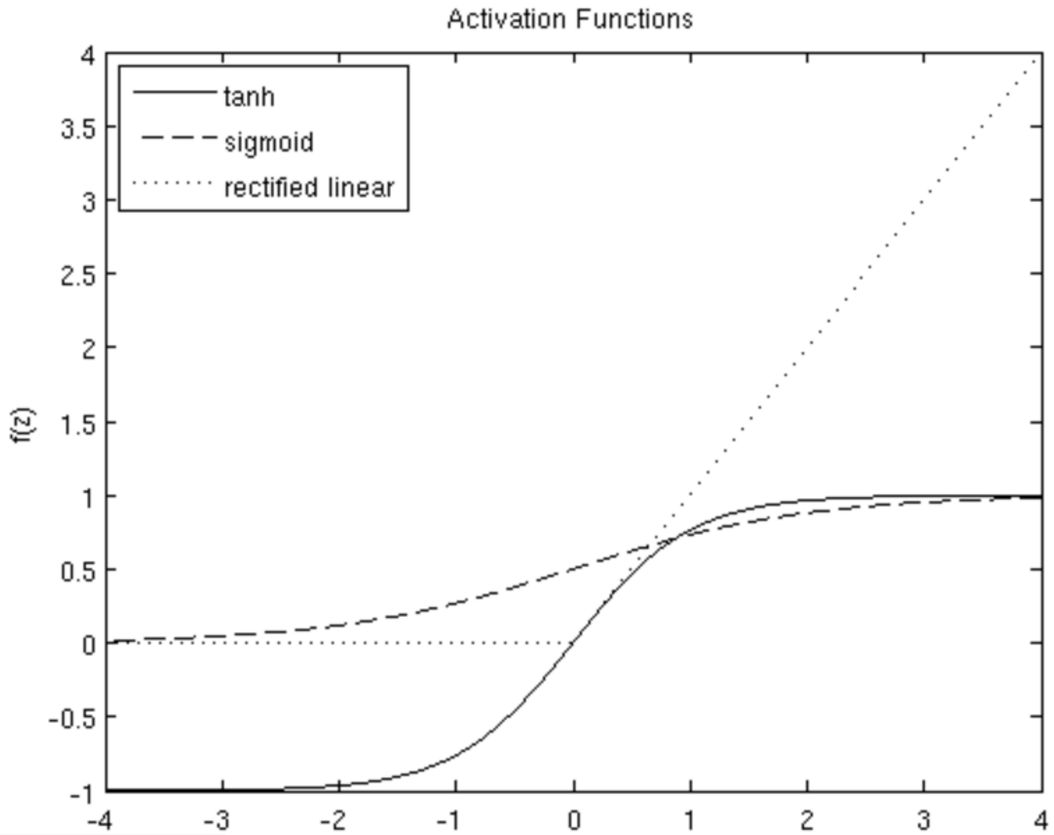
Neural Network: Definitions



$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Neural Networks: Activation Functions



$$f(z) = \frac{1}{1 + \exp(-z)}$$

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

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

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

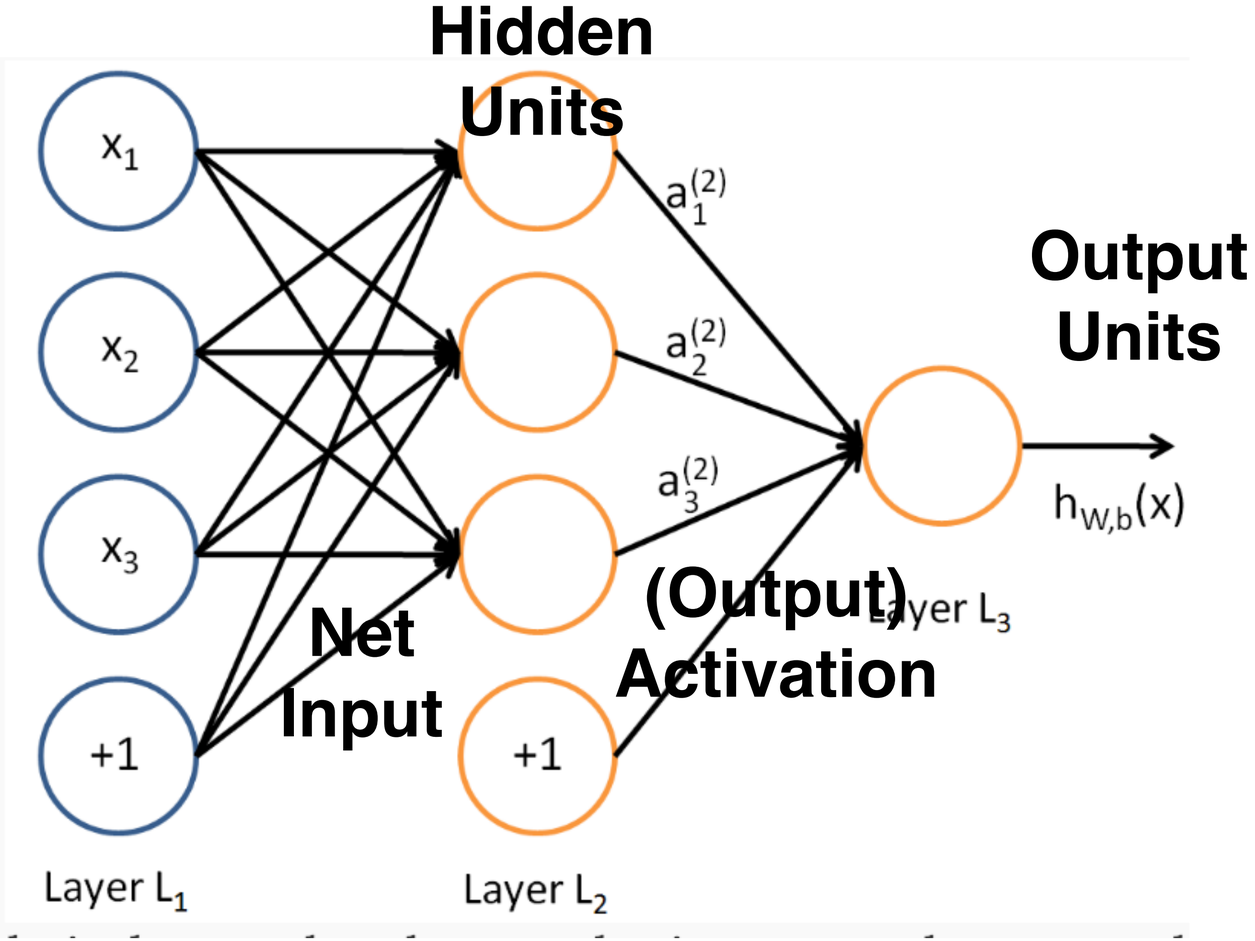
Neural Networks: Definitions

Feedforward Propagation: Scalar Form

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

Input Units

$$z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)}x_j + b_i^{(1)}$$



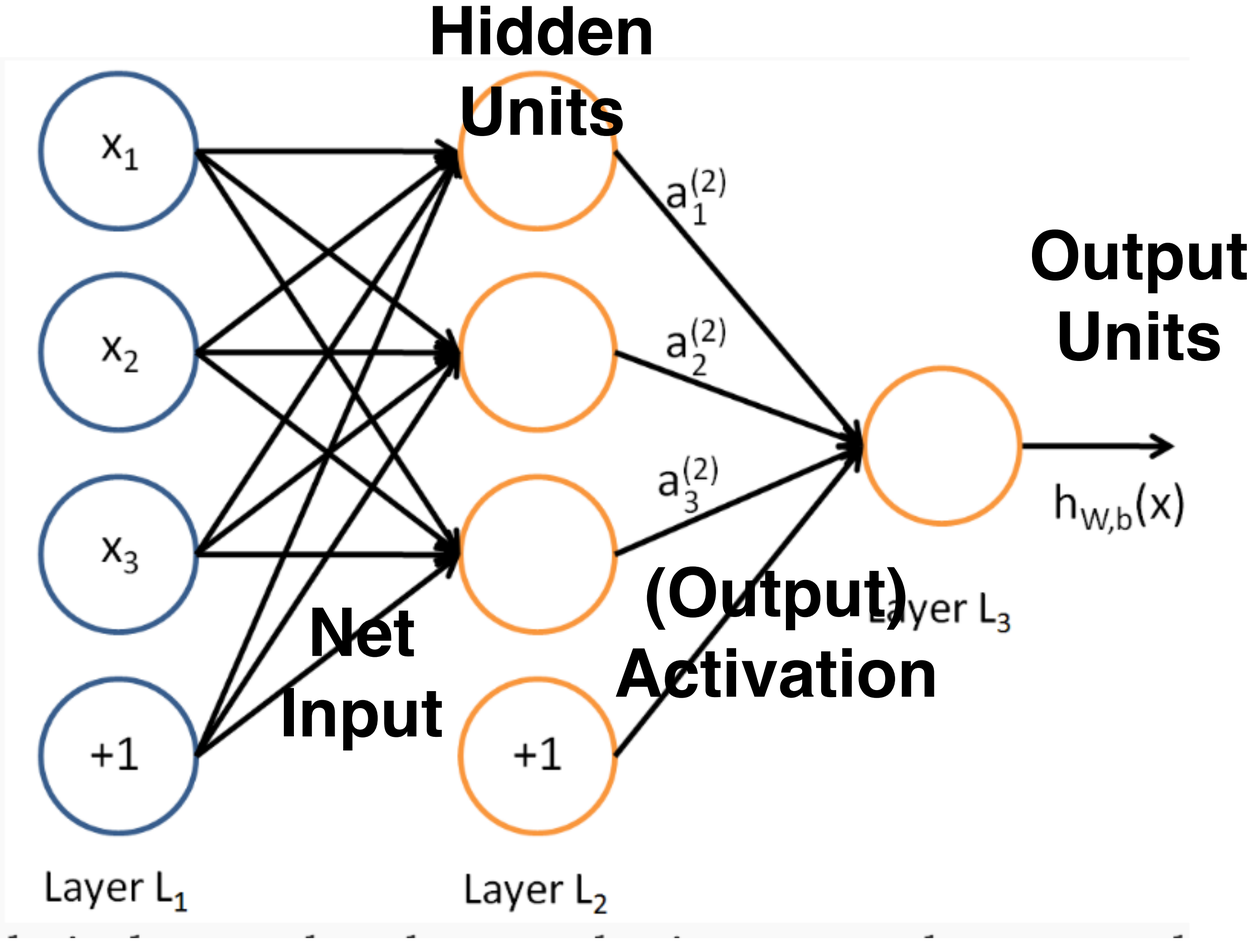
http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Neural Networks: Definitions

Feedforward Propagation: Vector Form

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{aligned}
$$

Input Units



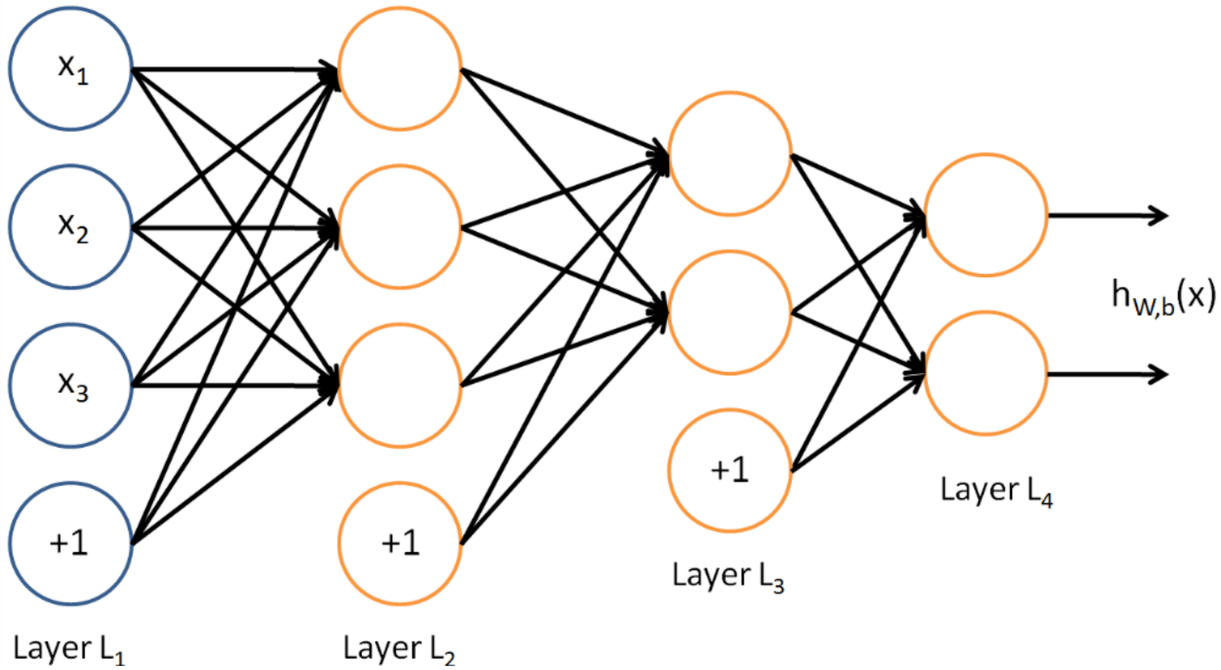
http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Neural Networks: Definitions

***Deep* Feedforward Propagation: Vector Form**

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

$$a^{(1)} = x$$



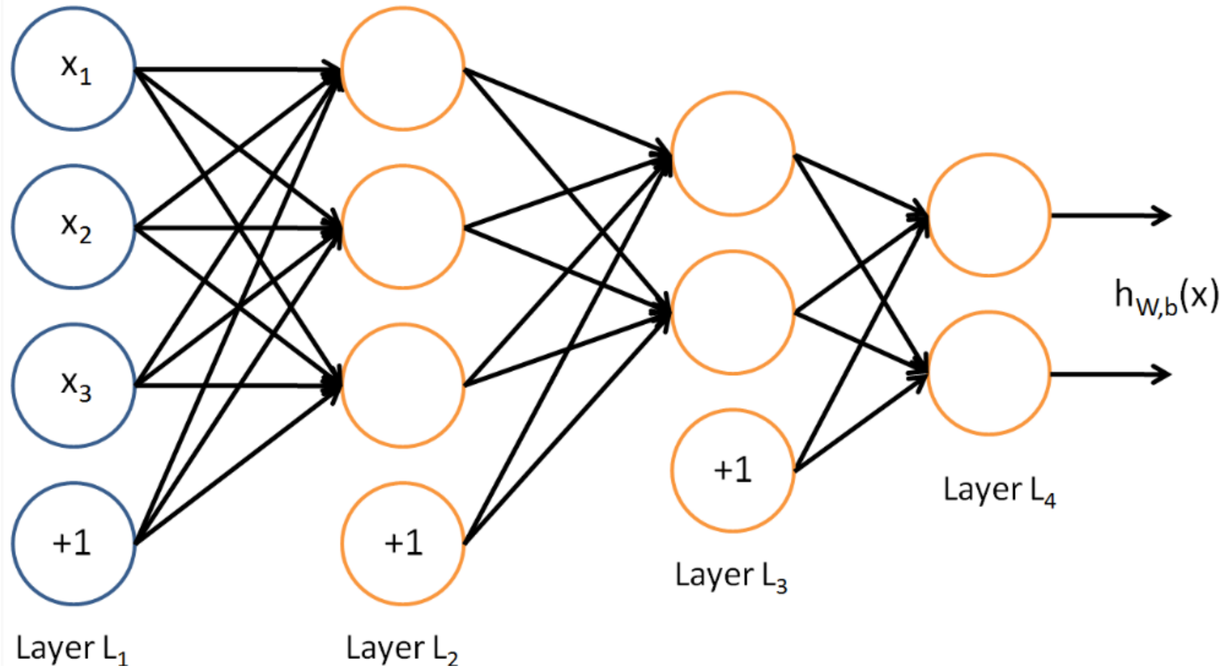
http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Neural Networks: Definitions

The Training Objective

$$J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right]$$

$$J(W,b;x,y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2$$

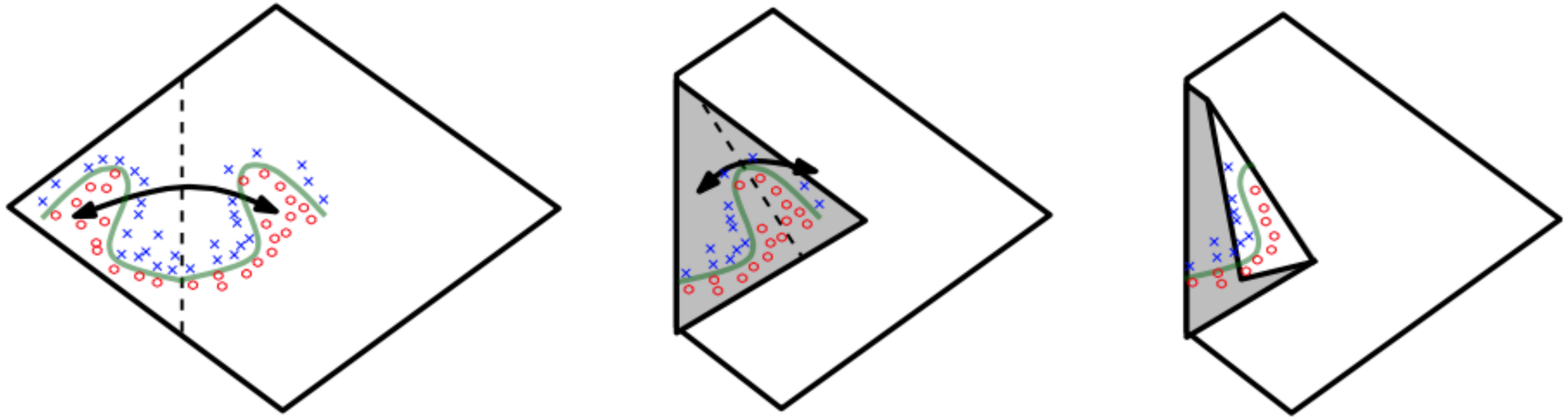


http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Expressive Power Theorems

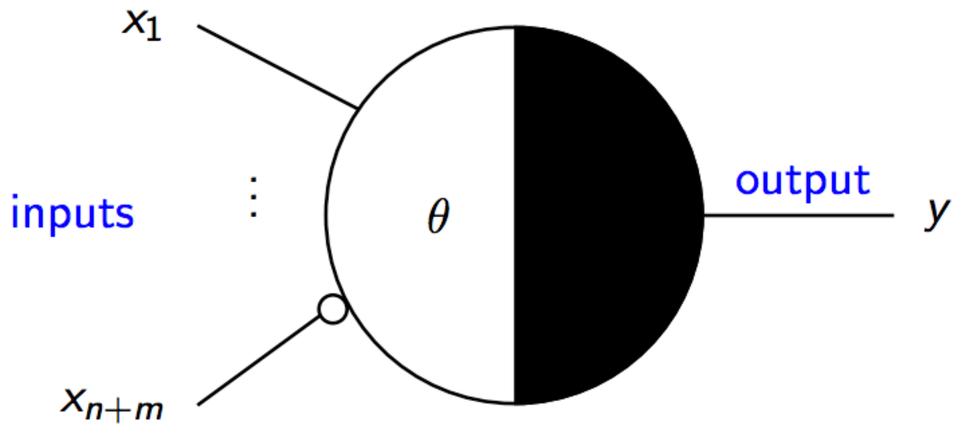
Compositions of Nonlinear Functions are more expressive

Deeper nets with rectifier/maxout units are exponentially more expressive than shallow ones (1 hidden layer) because they can split the input space in many more (not-independent) linear regions, with constraints, e.g., with abs units, each unit creates mirror responses, folding the input space:



[Yoshua Bengio]

McCulloch-Pitts Neurons



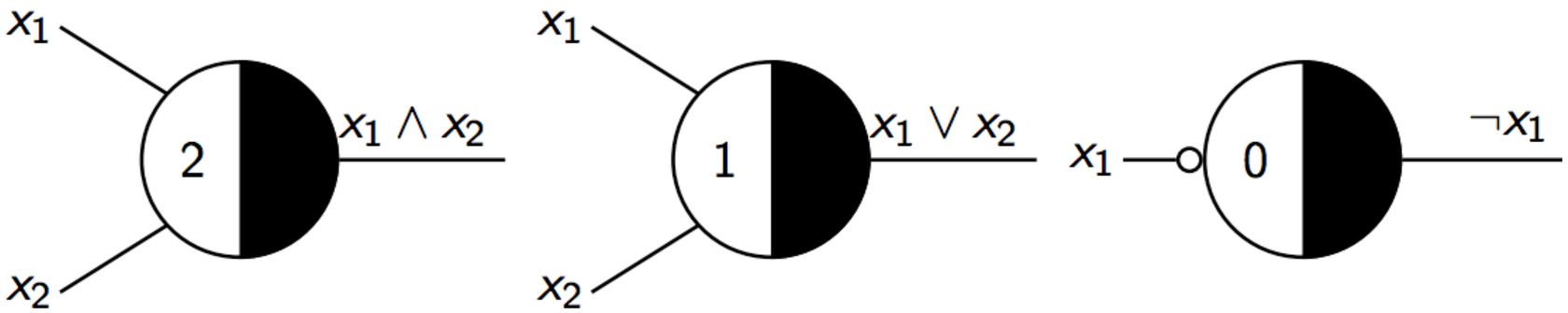
Stefan Droste

- all signals binary ($\in \{0, 1\}$)
- threshold $\theta \in \mathbb{N}_0$
- n excitatory inputs $x_1, \ldots, x_n$
- m inhibitory inputs $x_{n+1}, \ldots, x_{n+m}$
- one output y (with unrestricted fan-out)

$$y(x_1, \ldots, x_{n+m}, \theta) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geq \theta \text{ and } \sum_{i=n+1}^{m} x_i = 0 \\ 0 & \text{if } \sum_{i=1}^{n} x_i < \theta \text{ or } \sum_{i=n+1}^{m} x_i > 0 \end{cases}$$

Expressive Power of McCulloch-Pitts Nets

Feed-forward McCulloch-Pitts nets can compute any Boolean function $f : \{0,1\}^n \to \{0,1\}^m$.

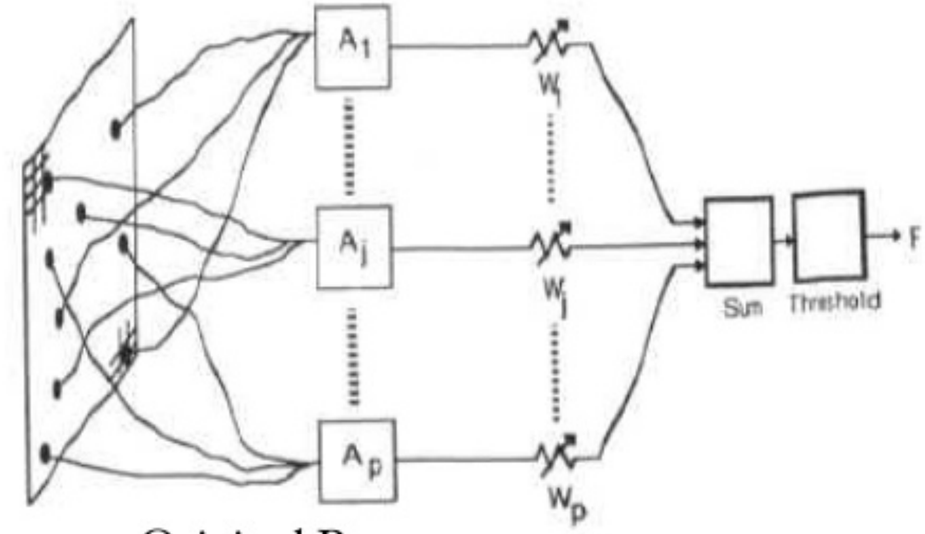


Stefan Droste

Recursive McCulloch-Pitts nets can simulate any deterministic finite automaton (DFA).

The Perceptron (Rosenblatt)

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

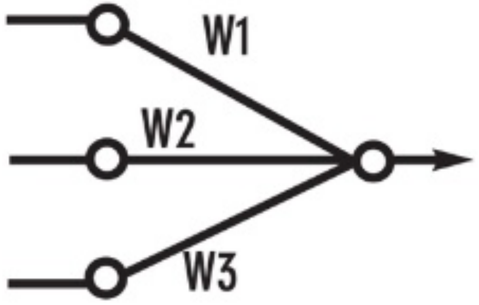


Original Perceptron

(From Perceptrons by M. L Minsky and S. Papert, 1969, Cambridge, MA: MIT Press. Copyright 1969 by MIT Press.

Frank Rosenblatt
(1928-1971)

Simplified model:



Limitations of Perceptron

- Rosenblatt was overly enthusiastic about the perceptron and made the ill-timed proclamation that:

- "Given an elementary α-perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always yield a solution to C(W) in finite time…" [4]

- In 1969, Marvin Minsky and Seymour Papert showed that the perceptron could only solve linearly separable functions. Of particular interest was the fact that the perceptron still could not solve the XOR and NXOR functions.

- Problem outlined by Minsky and Papert can be solved by **deep NNs**. However, many of the artificial neural networks in use today still stem from the early advances of the McCulloch-Pitts neuron and the Rosenblatt perceptron.

Universal Approximation Theorem [Cybenko 1989, Hornik 1991]

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the *m*-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any function $f \in C(I_m)$ and $\varepsilon > 0$, there exists an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

as an approximate realization of the function f where f is independent of φ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

This still holds when replacing I_m with any compact subset of $\mathbb{R}^m$.

- https://en.wikipedia.org/wiki/Universal_approximation_theorem

Universal Approximation Theorem

- https://en.wikipedia.org/wiki/Universal_approximation_theorem

- Shallow neural networks can *represent* a wide variety of interesting functions when given appropriate parameters; however, it does not touch upon the algorithmic learnability of those parameters.

- Proved by George Cybenko in 1989 for sigmoid activation functions.[2]

- Kurt Hornik showed in 1991[3] that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators.

Question (5 min):

Why is the theorem true? What is the intuition?

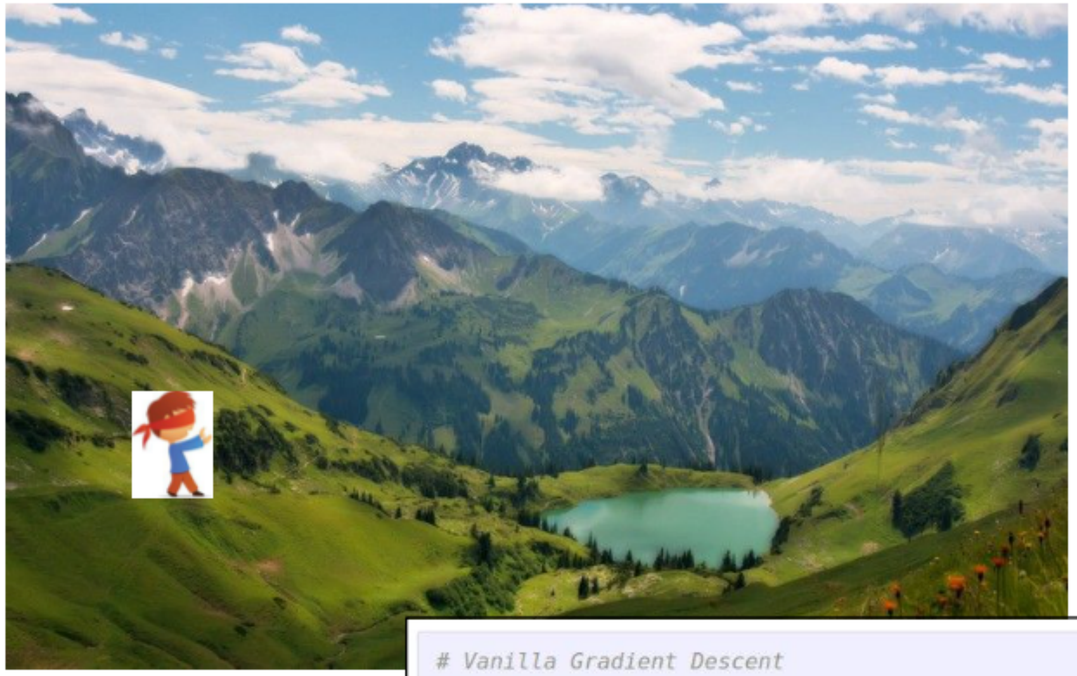
What happens when you go deep?

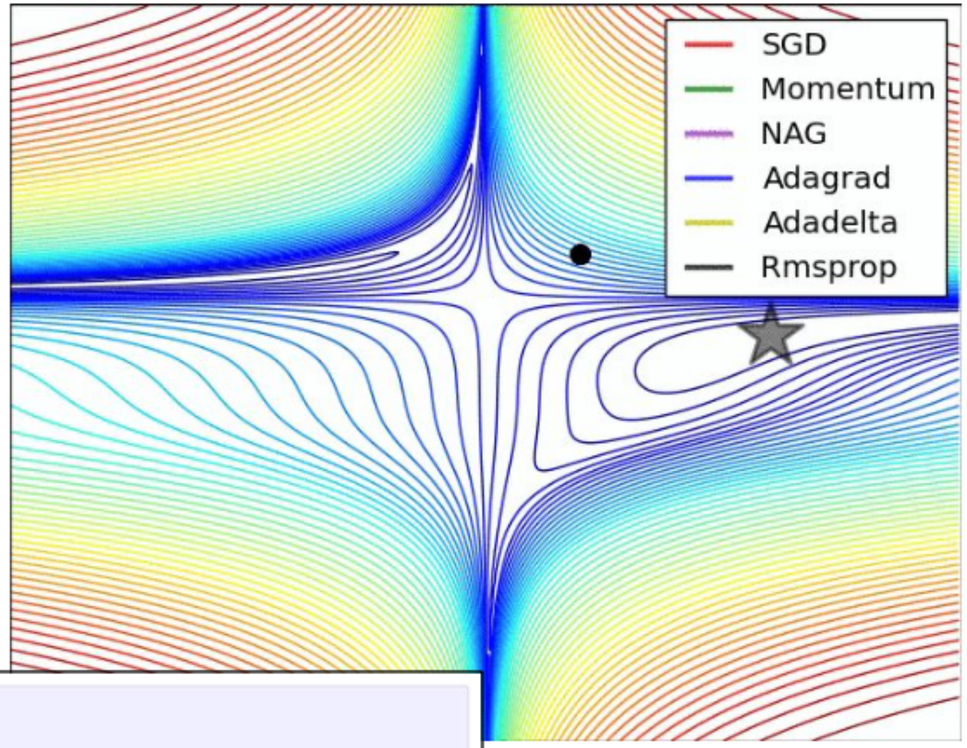
Try iterating $f(x) = x^2$ vs. $f(x) = ax+b$

Training Neural Networks Via Gradient Descent

Gradient Descent

Optimization





(image credits
to Alec Radford)

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Gradient Descent

- **Optimization problem**

$$\min_{\mathbf{w} \in \mathbb{R}^n} f(\mathbf{w})$$

- **Insight:** At the point $\mathbf{a}$, $f(\mathbf{w})$ decreases fastest in the direction of the **negative gradient** of f (assuming f is defined and differentiable at $\mathbf{a}$)

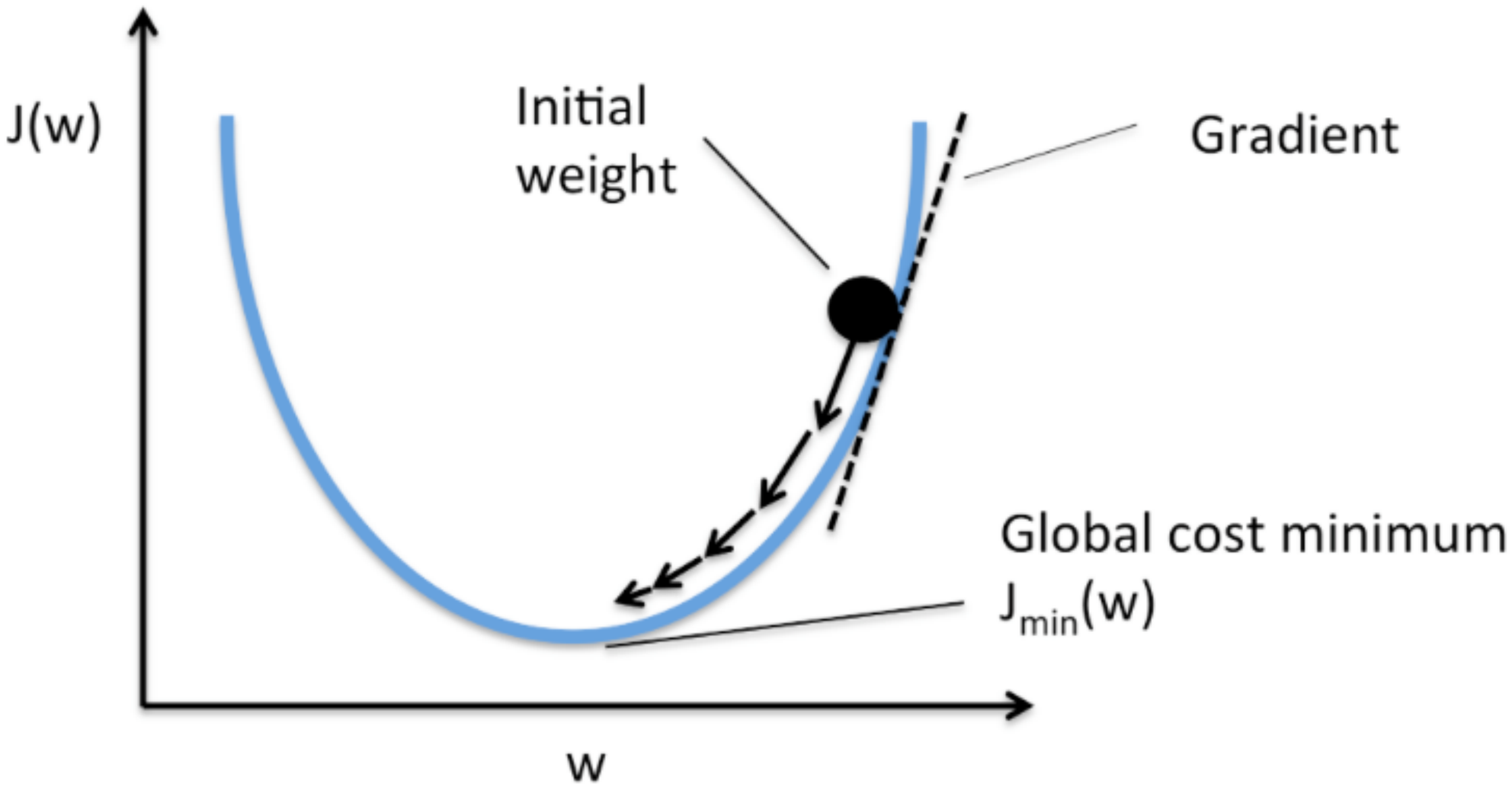
$$-\nabla f(\mathbf{a})$$

- Step in the direction of the negative gradient

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mu^t \nabla f(\mathbf{w}^t)$$

where μ^t is the **step size** at step t

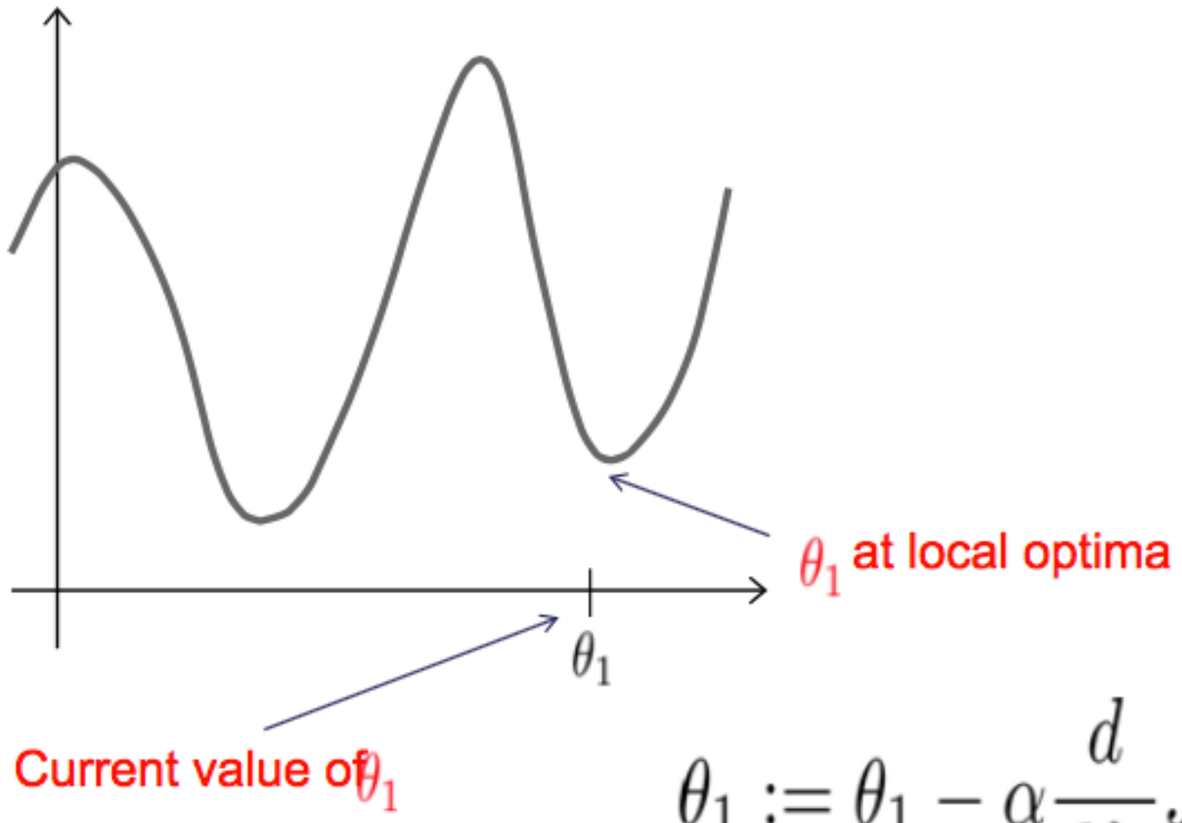
Gradient Descent



Question:

What kind of problems might you run into with Gradient Descent? (4 min)

Global Optima is not Guaranteed

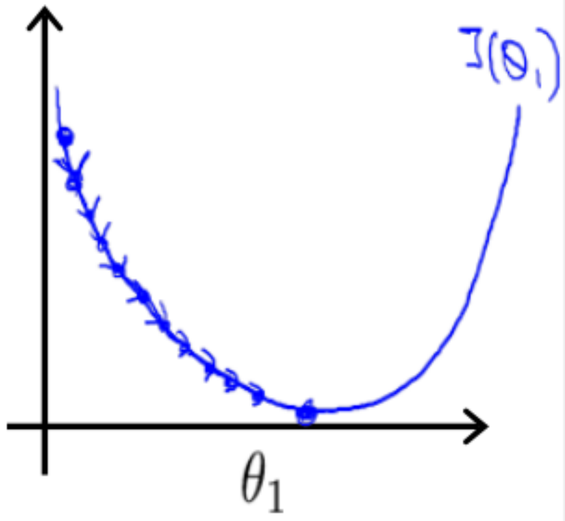


$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

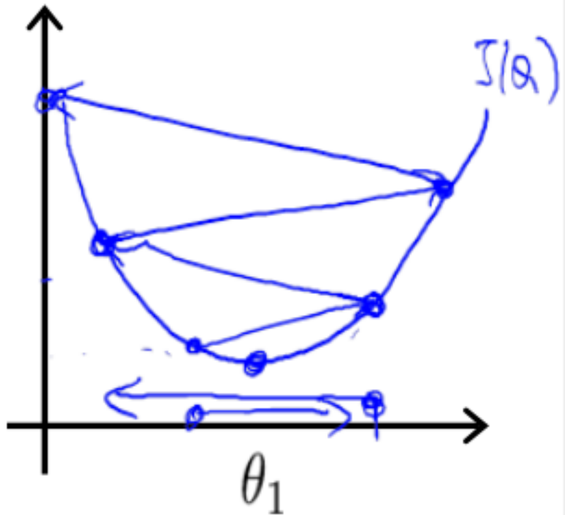
Learning Rate Needs to Be Carefully Chosen

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Training Neural Networks: Computing Gradients Efficiently with the Backpropagation Algorithm

Chain Rule

If f and g are both differentiable and F is the composite function defined by $F(x)=f(g(x))$, then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x)) \cdot g'(x)$$

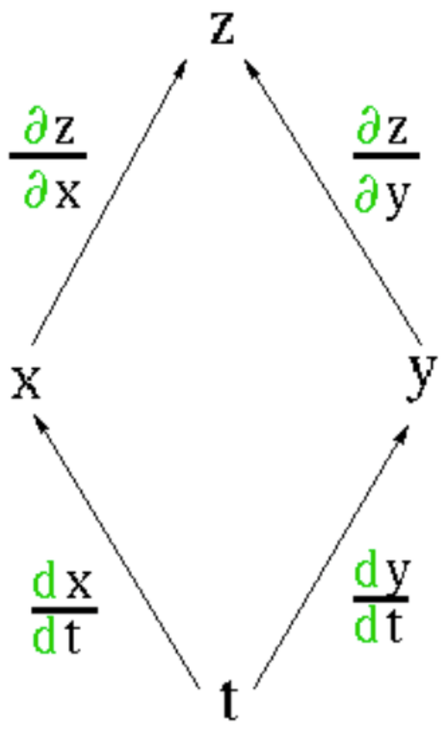
In Leibniz notation, if $y=f(u)$ and $u=g(x)$ are both differentiable functions, then

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Chain Rule

Let $x = x(t)$ and $y = y(t)$ be differentiable at t and suppose that $z = f(x, y)$ is differentiable at the point $(x(t), y(t))$. Then $z = f(x(t), y(t))$ is differentiable at t and

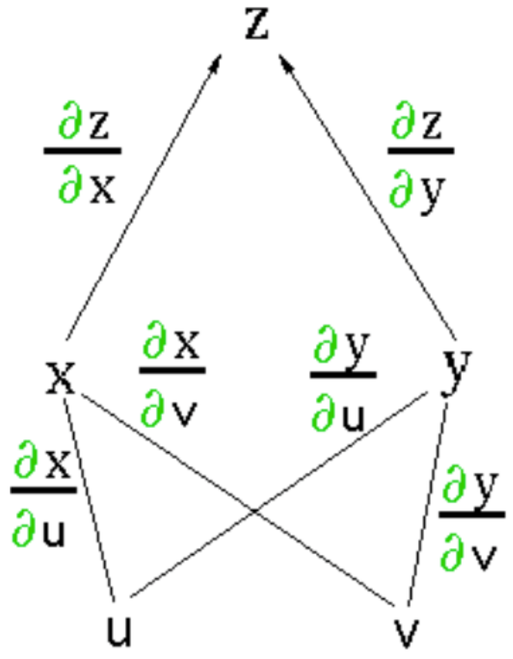
$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt}.$$



Chain Rule

Let $x = x(u,v)$ and $y = y(u,v)$ have first-order partial derivatives at the point (u,v) and suppose that $z = f(x,y)$ is differentiable at the point $(x(u,v), y(u,v))$. Then $f(x(u,v), y(u,v))$ has first-order partial derivatives at (u,v) given by

$$\begin{aligned}\frac{\partial z}{\partial u} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial v} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial v}.\end{aligned}$$



Exercise:

Do Chain Rule on
a nested function (2 min)

Backpropagation is an efficient way to compute gradients

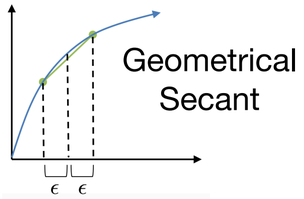
- a.k.a. **Reverse Mode Automatic Differentiation (AD)**, and based on a systematic application of the chain rule. It is fast for low-dimensional outputs. For one output (e.g. a scalar loss function), time to compute gradients with respect to **ALL** inputs is proportional to the time to compute the output. An explicit mathematical expression of the output is not required, only an algorithm to compute it.

- it is **NOT** the same as **symbolic differentiation** (e.g. mathematica).

- **Numerical/Finite Differences** are slow for high-dimensional inputs (e.g. model parameters) and outputs. For a single output, time to compute gradients scales as the number of inputs. May suffer from issues of floating point precision and requires a choice of a parameter increment.

Centered Finite Difference

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{(J(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{d}_i) - J(\boldsymbol{\theta} - \epsilon \cdot \boldsymbol{d}_i))}{2\epsilon}$$



https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture20-backprop.pdf

The ***Cheap Gradient Principle*** in Backpropagation: The time complexity scales up to the number of operations performed in the forward pass

$$\mathtt{OPS}\,\{\,\nabla \mathtt{f}(\mathbf{x})\} \leq \omega\ \mathtt{OPS}\,\{\mathtt{f}(\mathbf{x})\}$$

$\mathbf{X}$ Is a multidimensional input, $\omega \sim 5$

$\omega = 3$ for polynomial operations and OPS counting the number of multiplications

More generally, for an m-dimensional output $\mathbf{F}(\mathbf{x})$

$$\mathtt{OPS}\left\{\mathbf{F}'(\mathbf{x})\right\} \leq \mathtt{m}\ \omega\ \mathtt{OPS}\left\{\mathbf{F}(\mathbf{x})\right\}$$

There is **no** equivalent **Cheap Jacobian Principle** or **Cheap Hessian Principle**

https://www.math.uni-bielefeld.de/documenta/vol-ismp/52_griewank-andreas-b.pdf

The Spatial Complexity of Backpropagation scales with the number of operations performed in the forward pass

$$\texttt{MEM}\left\{\mathbf{F}'(\mathbf{x})\right\} \sim \texttt{OPS}\left\{\mathbf{F}(\mathbf{x})\right\} \gtrsim \texttt{MEM}\left\{\mathbf{F}(\mathbf{x})\right\}$$

https://www.math.uni-bielefeld.de/documenta/vol-ismp/52_griewank-andreas-b.pdf

Temporal Complexity in Automatic Differentiation

$\mathbf{x}$ Is a n-dimensional input, $\mathbf{F}(\mathbf{x})$ Is an m-dimensional output

Reverse Mode: $\mathtt{OPS}\left\{\mathbf{F}'(\mathbf{x})\right\} \leq \mathtt{m}\ \omega\ \mathtt{OPS}\left\{\mathbf{F}(\mathbf{x})\right\}$

Forward Mode: $\mathtt{OPS}\left\{\mathbf{F}'(\mathbf{x})\right\} \leq \mathtt{n}\ \omega\ \mathtt{OPS}\left\{\mathbf{F}(\mathbf{x})\right\}$

$$\omega < 6$$

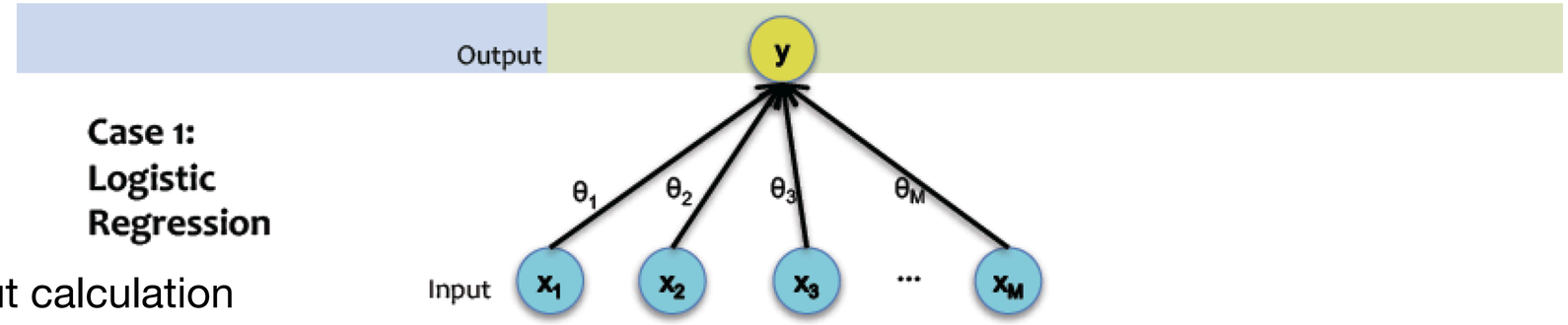
There is **no Cheap Jacobian Principle** or **Cheap Hessian Principle** but a Jacobian-vector product can be computed as efficiently as the gradient, and a Hessian-vector product can be computed efficiently in O(n) instead of O(nxn)

https://arxiv.org/pdf/1502.05767.pdf
https://www.math.uni-bielefeld.de/documenta/vol-ismp/52_griewank-andreas-b.pdf

How to Learn NNs? History of the Backpropagation Algorithm (1960-86)

- Introduced by Henrey J. Kelley (1960) and Arthur Bryson (1961) in control theory, using Dynamic Programming
- Simpler derivation using **Chain Rule** by Stephen Dreyfus (1962)
- General method for Automatic Differentiation by Seppo Linnainamaa (1970)
- Using backdrop for parameters of controllers minimizing error by Stuart Dreyfus (1973)
- Backprop brought into NN world by Paul Werbos (1974)
- Used it to learn representations in hidden layers of NNs by Rumelhart, Hinton & Williams (1986)

Backpropagation Example (5 min)



Output calculation

Forward Pass

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward Pass

$$\frac{dJ}{dy} =$$

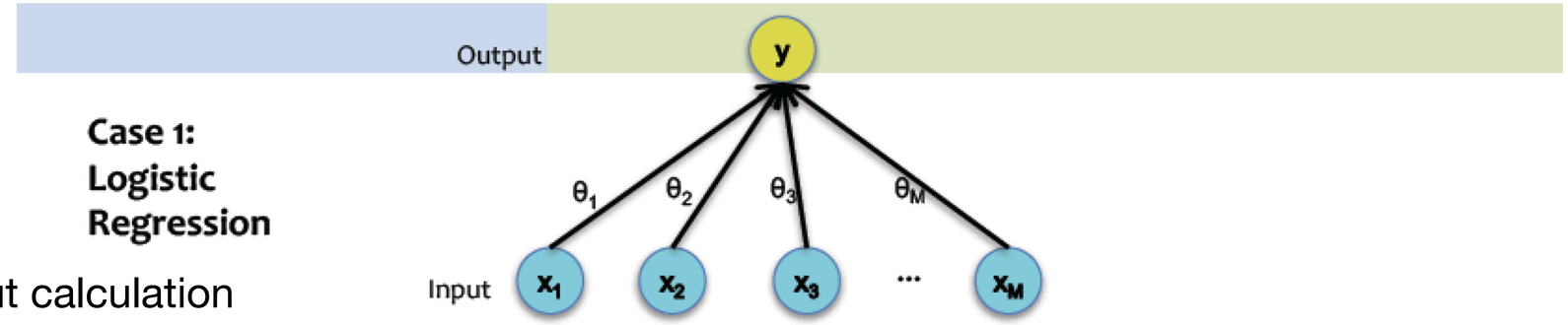
$$\frac{dJ}{da} =$$

$$\frac{dJ}{d\theta_j} =$$

$$\frac{dJ}{dx_j} =$$

Modified from https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture20-backprop.pdf

Backpropagation Example



Output calculation

Forward Pass

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward Pass

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Gradient calculation

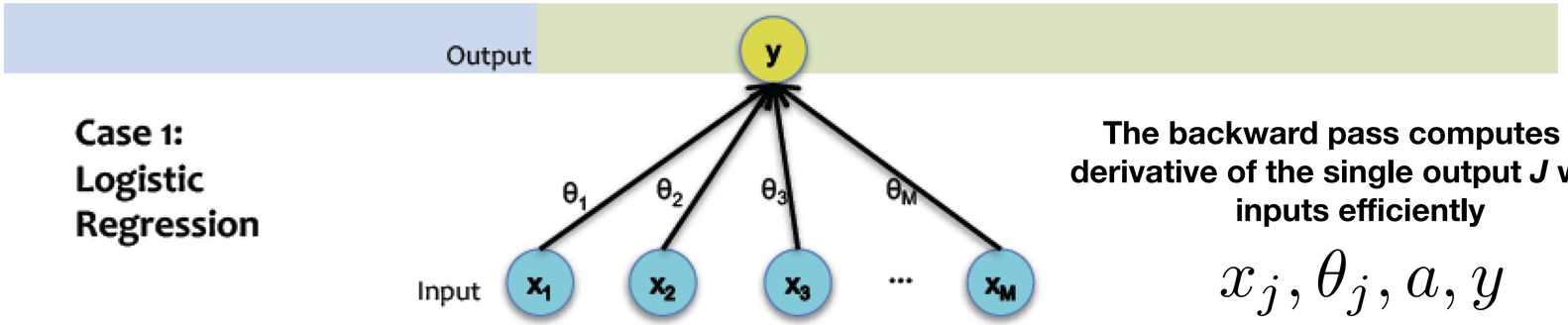
Linked by the chain rule

84

Modified from https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture20-backprop.pdf

Backpropagation Example

Case 1: Logistic Regression



The backward pass computes the derivative of the single output *J* wrt all inputs efficiently

$$x_j, \theta_j, a, y$$

Forward Pass

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward Pass

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

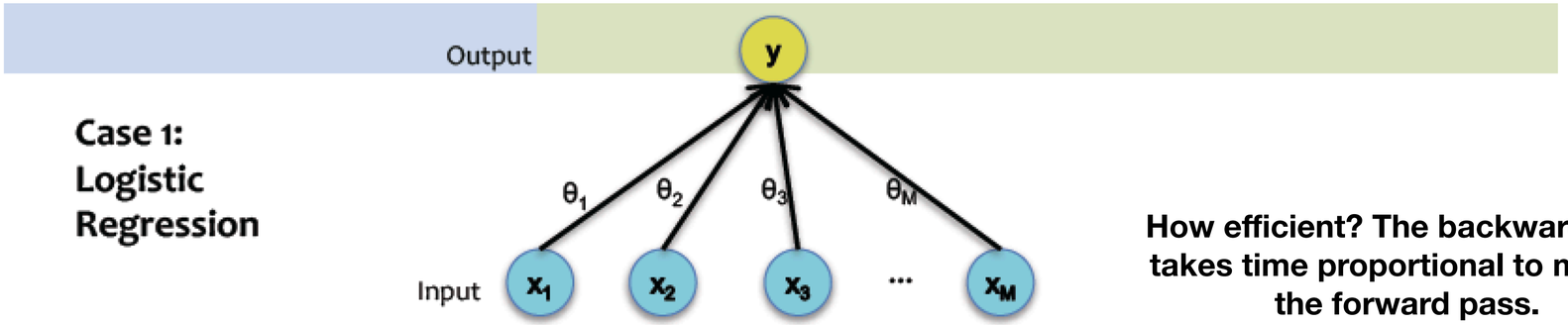
$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Modified from https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture20-backprop.pdf

Backpropagation Example

Case 1: Logistic Regression



How efficient? The backward pass takes time proportional to making the forward pass.

Forward Pass

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward Pass

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

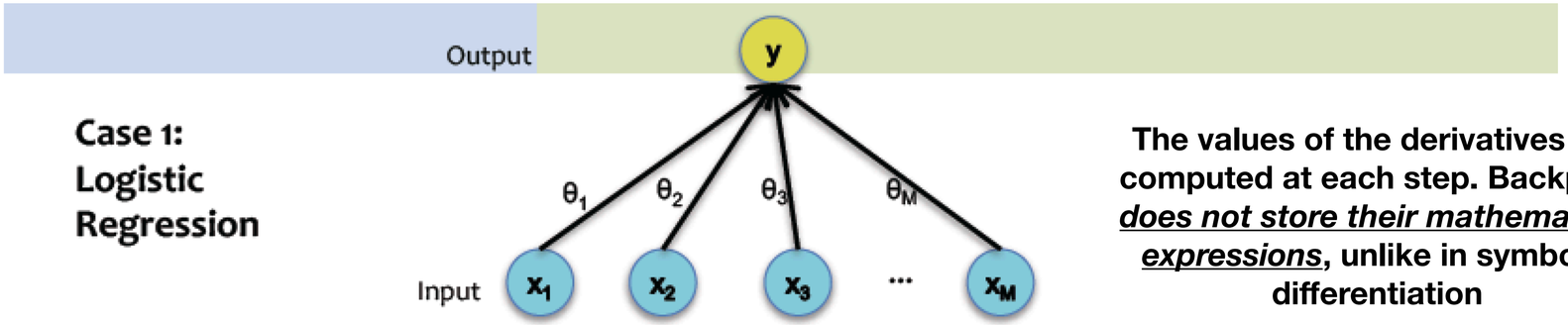
$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Modified from https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture20-backprop.pdf

Backpropagation Example

Case 1: Logistic Regression



The values of the derivatives are computed at each step. Backprop *does not store their mathematical expressions*, unlike in symbolic differentiation

Forward Pass

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward Pass

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Modified from https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture20-backprop.pdf

When is Backpropagation efficient?

Efficient?	High-dimensional inputs $\frac{\partial J(\{\theta_i\},\{x_j\})}{\partial \theta_1}, \frac{\partial J(\{\theta_i\},\{x_j\})}{\partial \theta_2}, \ldots$	High-dimensional outputs $\frac{\partial J_1(\{\theta_i\},\{x_j\})}{\partial \theta}, \frac{\partial J_2(\{\theta_i\},\{x_j\})}{\partial \theta}, \ldots$
Backprop (Reverse Mode AD)	**YES** ***Cheap Gradient Principle*** **time cost ~ one forward pass**	**May not be** **Unless common subexpressions are leveraged**
Forward Mode AD	**NO**	**YES**
FD	**NO** **time cost is multiple forward passes; 2 PER input**	**NO**
Symbolic Differentiation	**May not be** **Formula for *J* can grow exponentially in size, aka *Expression Swell* (https://arxiv.org/pdf/1502.05767.pdf)**	**May not be**

Pseudo-Code for Backprop: Scalar Form

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2}\|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Pseudo-Code for Backprop: Matrix-Vector Form

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, up to the output layer L_{n_l}, using the equations defining the forward propagation steps

2. For the output layer (layer n_l), set

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$, set

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

4. Compute the desired partial derivatives:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$
$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Gradient Descent for Neural Networks

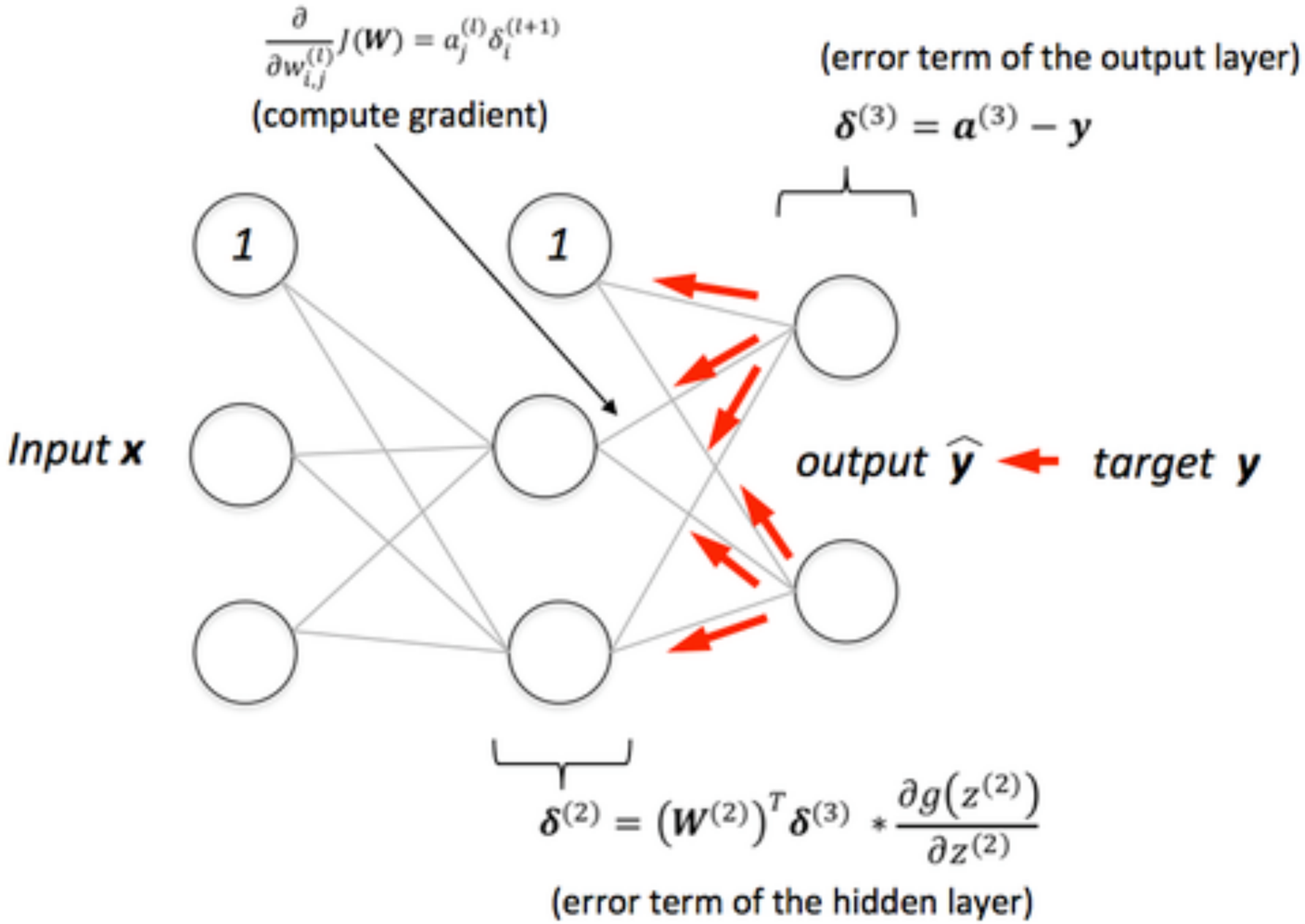
1. Set $\Delta W^{(l)} := 0$, $\Delta b^{(l)} := 0$ (matrix/vector of zeros) for all l.
2. For $i = 1$ to m,
    1. Use backpropagation to compute $\nabla_{W^{(l)}} J(W, b; x, y)$ and $\nabla_{b^{(l)}} J(W, b; x, y)$.
    2. Set $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$.
    3. Set $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$.
3. Update the parameters:

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

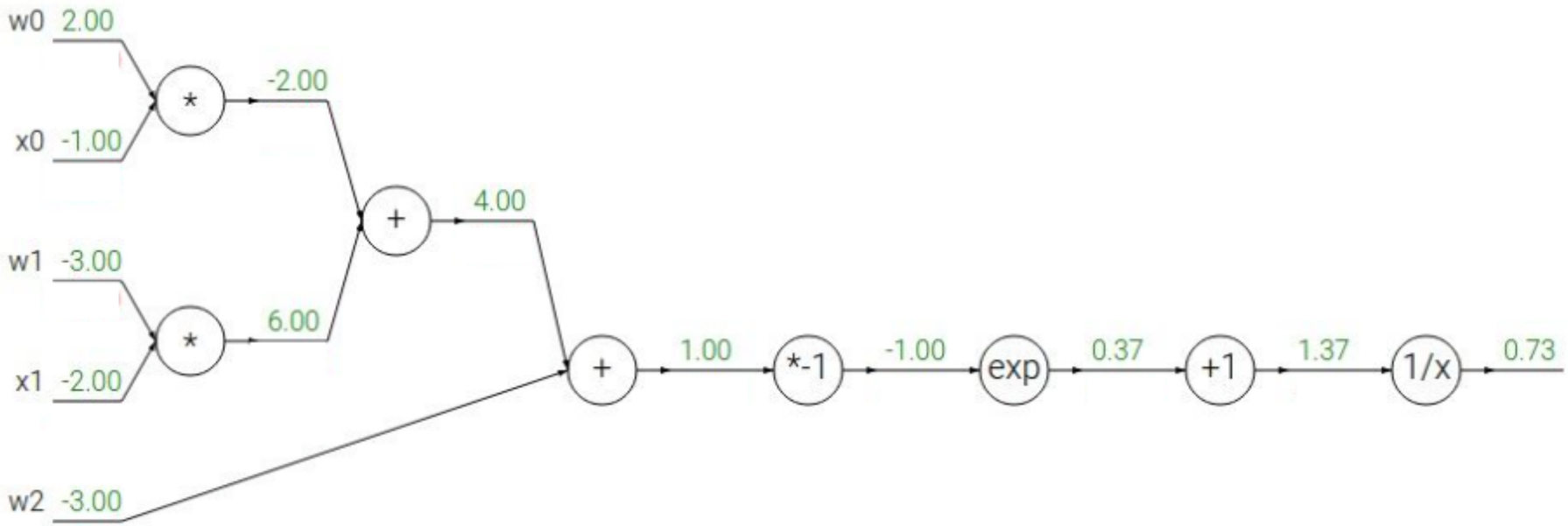
Backpropagation: Network View



Another *Deeper* Example (for practice)

Example

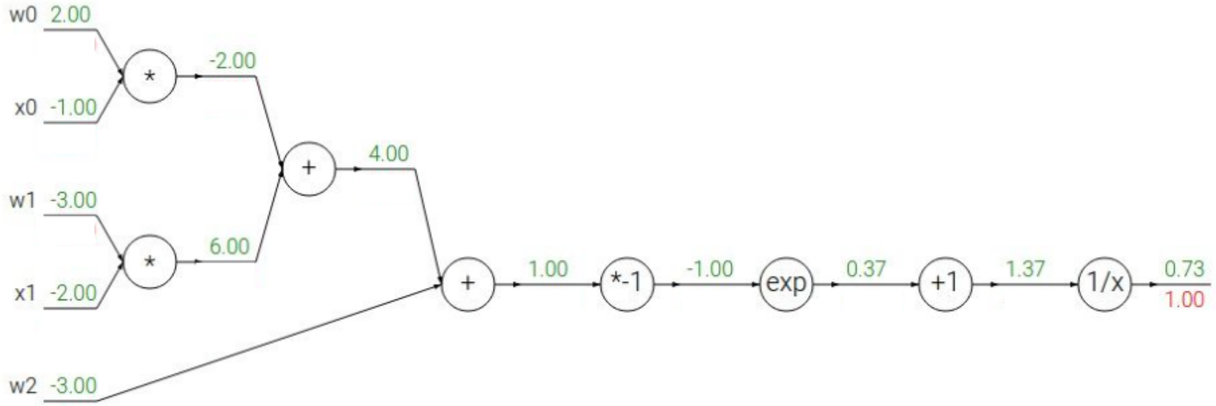
Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

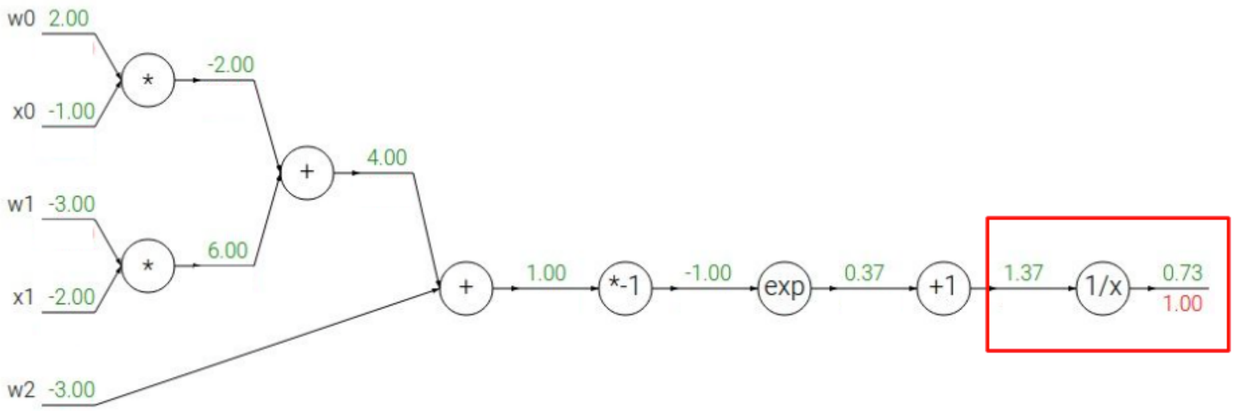
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

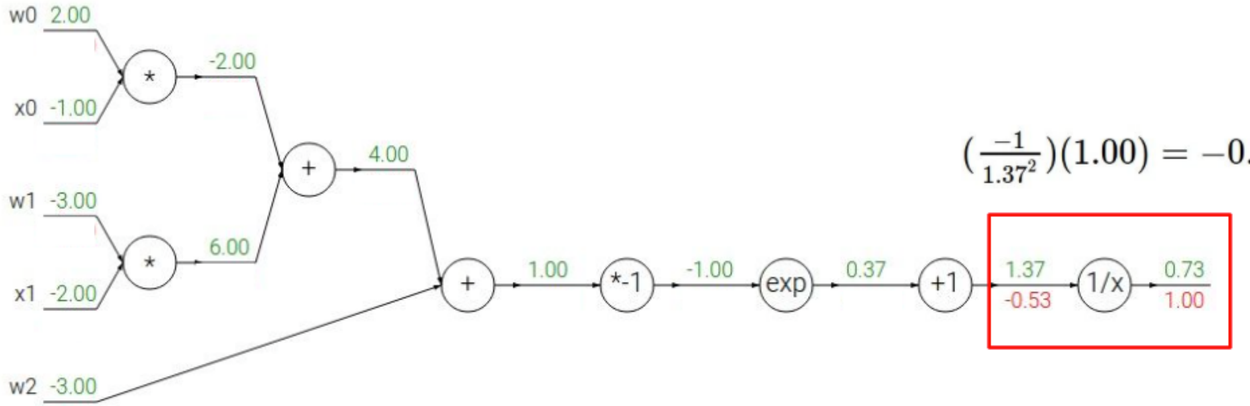
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$\left(\frac{-1}{1.37^2}\right)(1.00) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

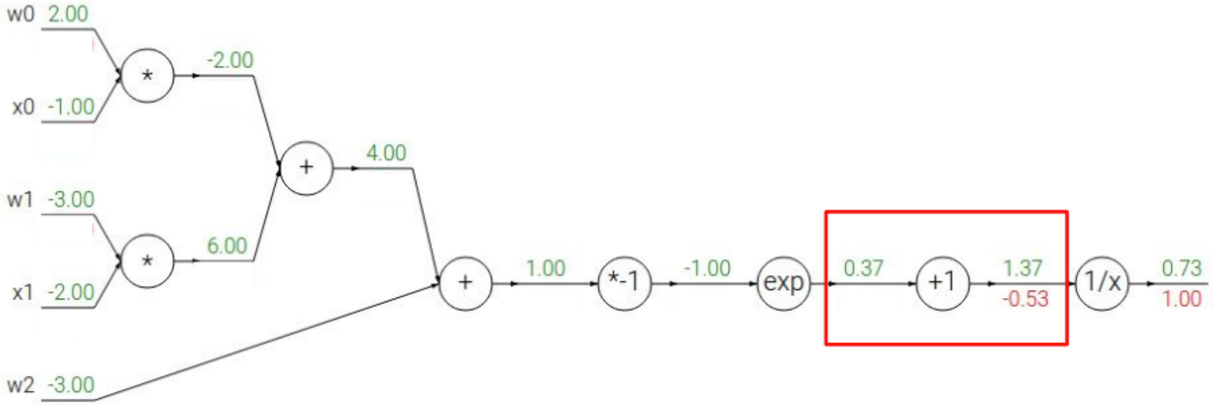
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

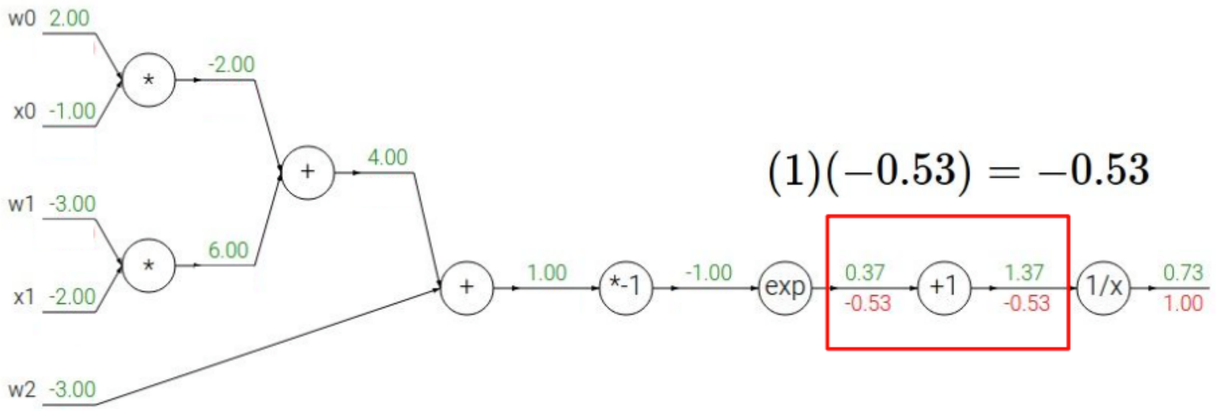
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$\boxed{f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1}$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(1)(-0.53) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

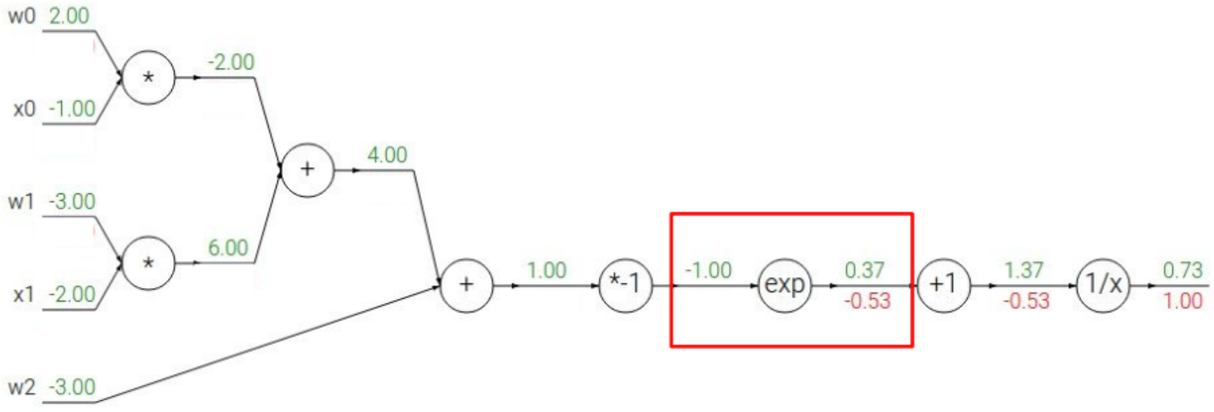
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$\boxed{f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1}$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

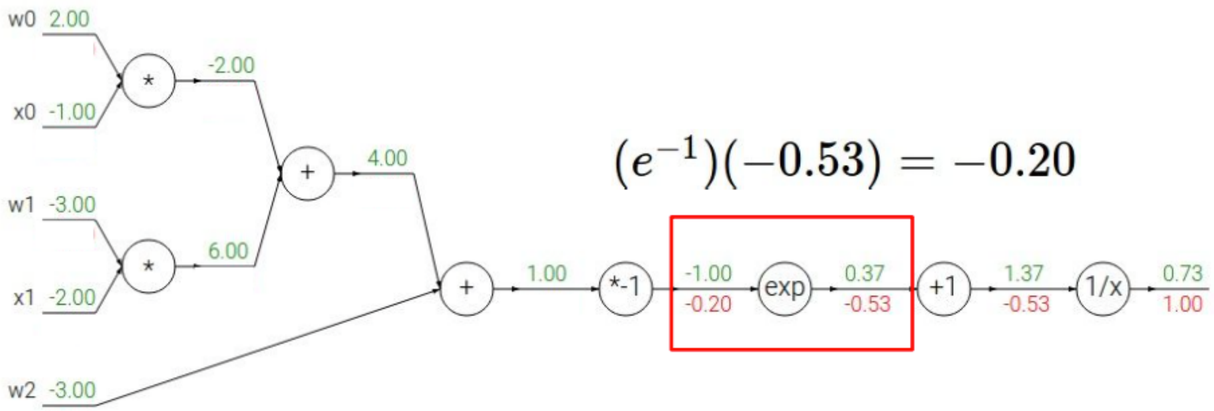
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(e^{-1})(-0.53) = -0.20$$

$$\boxed{f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x}$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

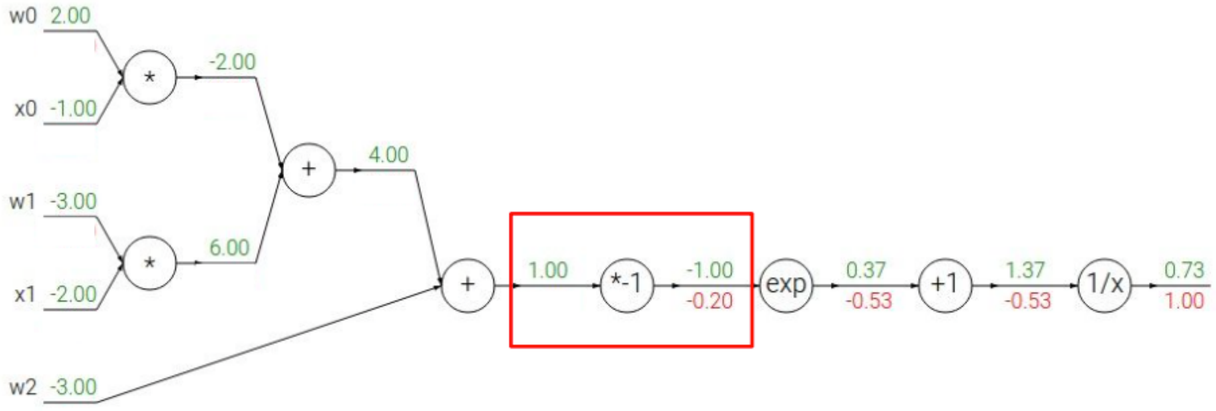
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

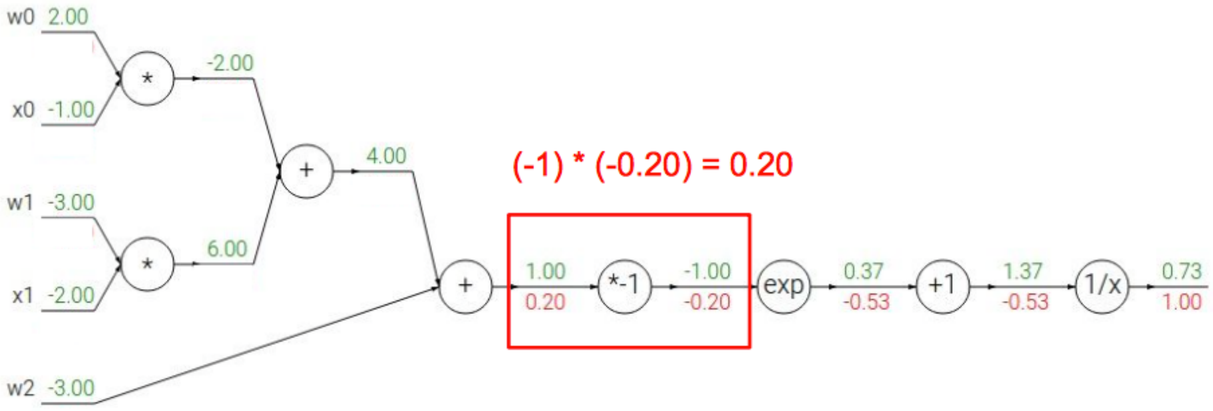
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

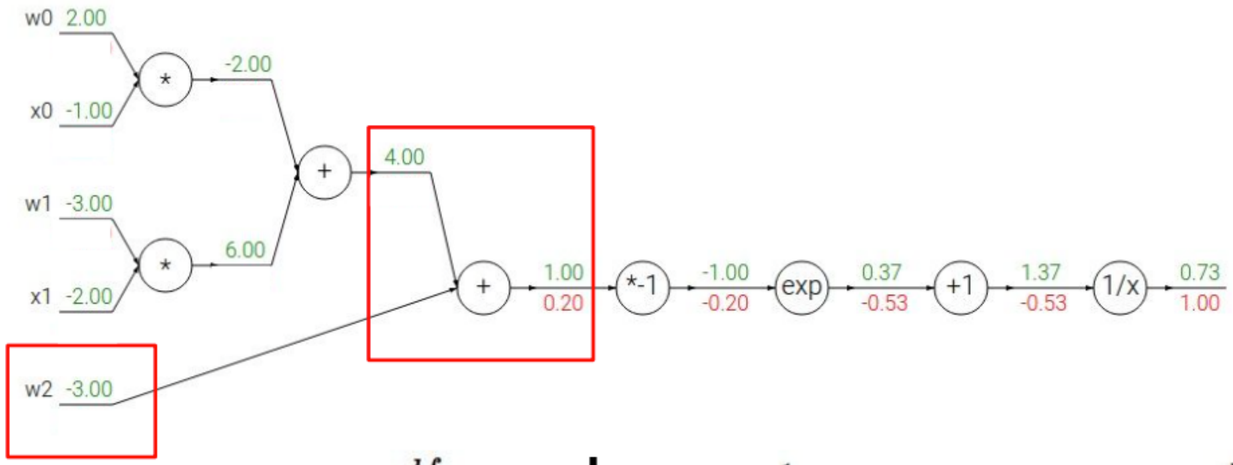
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

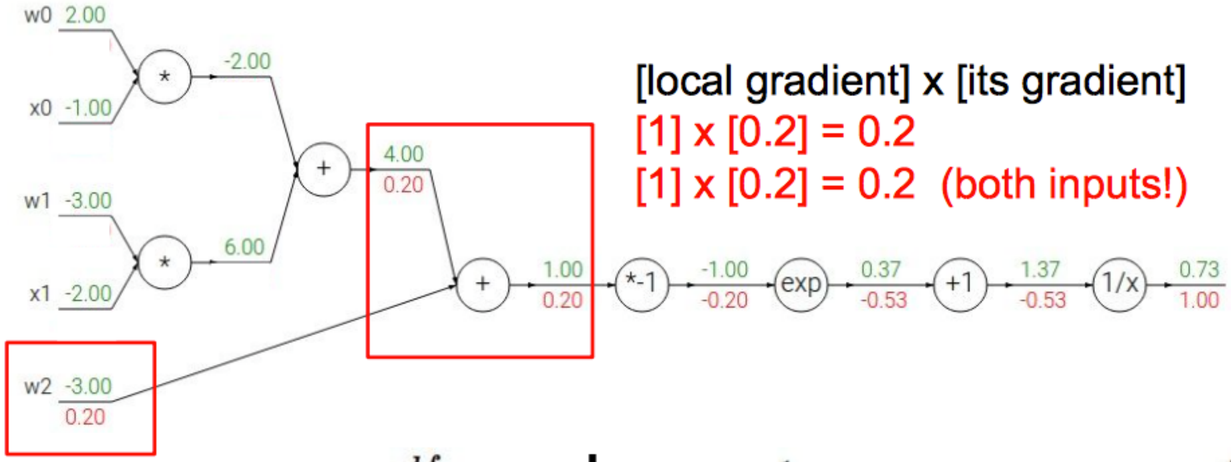
$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[local gradient] x [its gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs!)

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

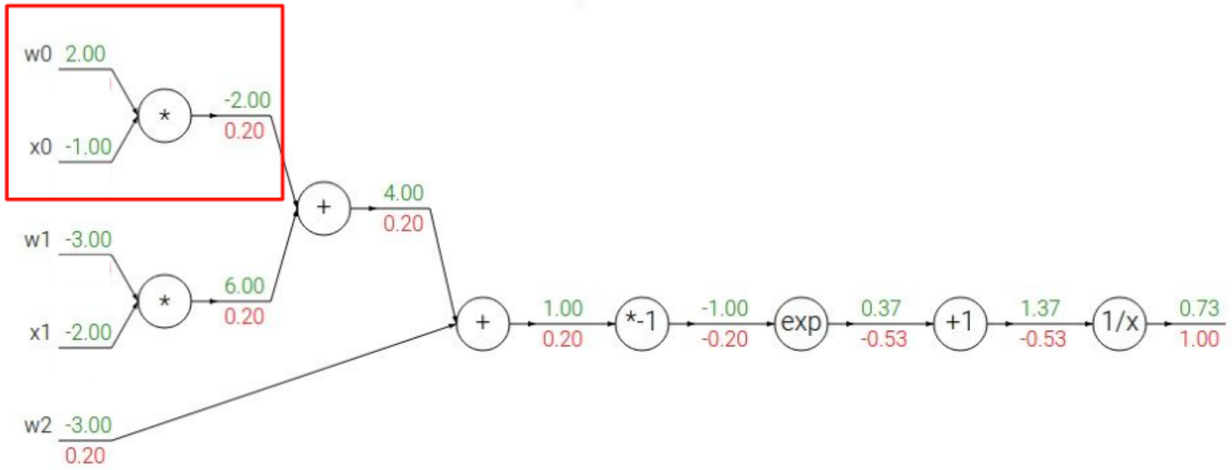
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

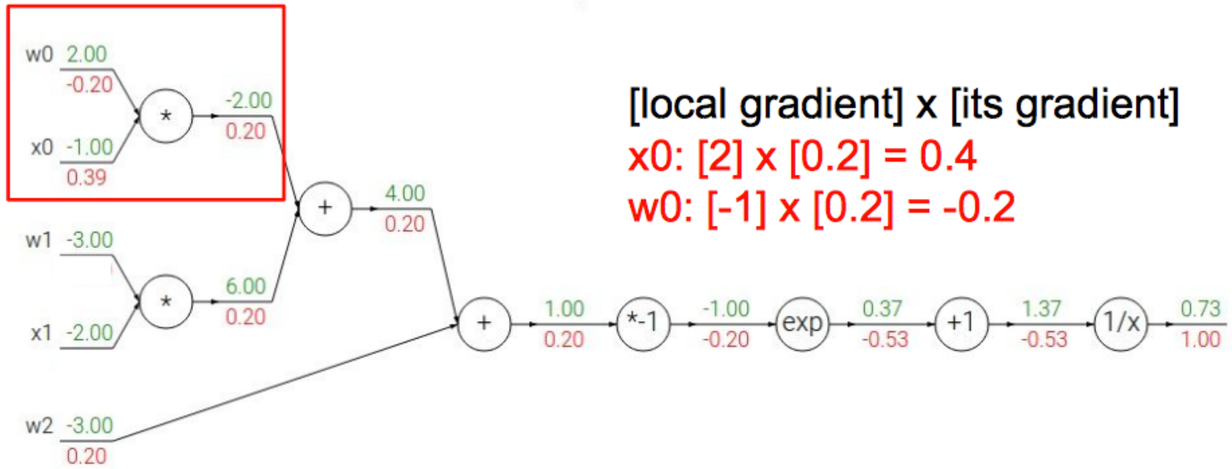
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]

x0: [2] x [0.2] = 0.4

w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

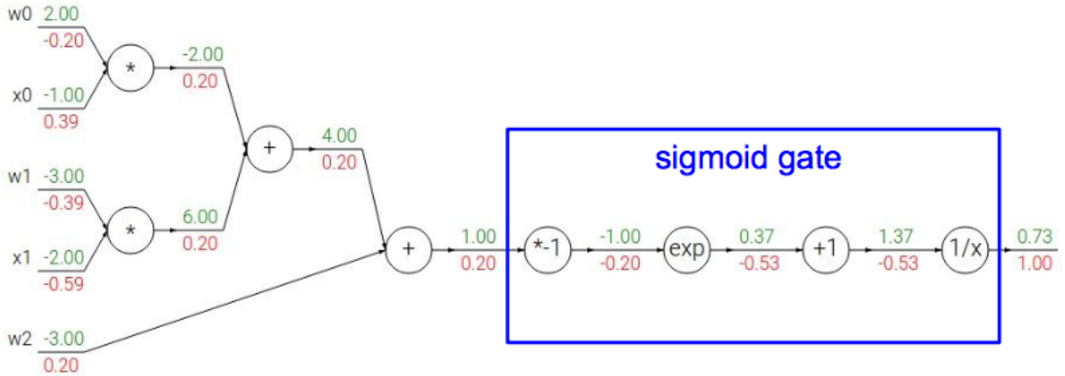
[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



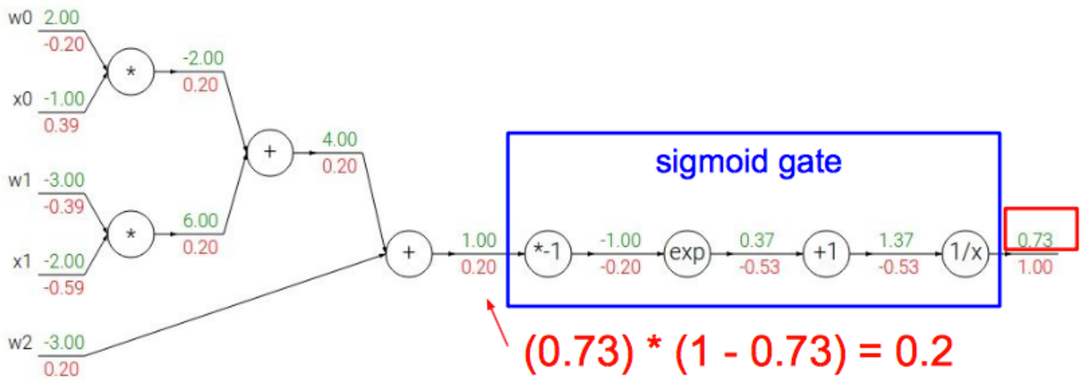
[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

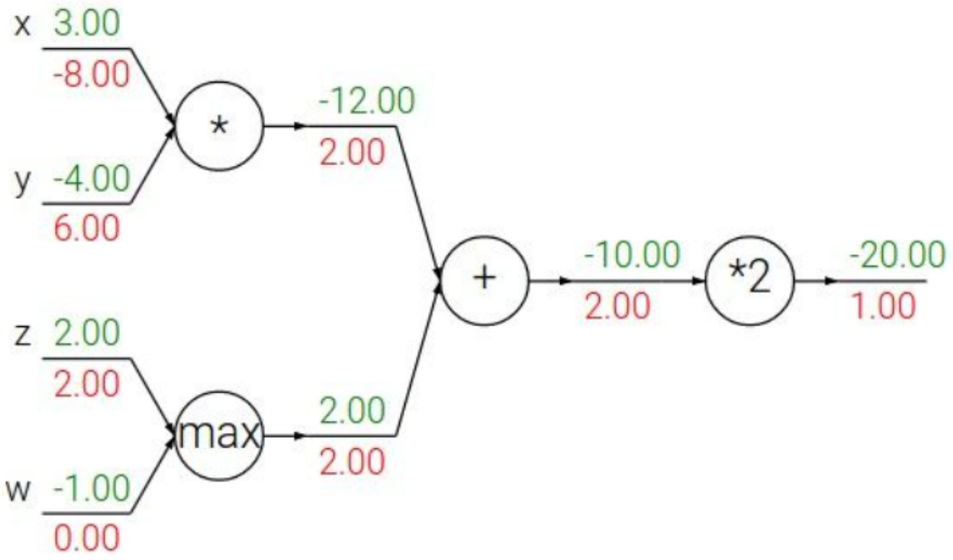


[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Example

Patterns in backward flow

add gate: gradient distributor
max gate: gradient router
mul gate: gradient… “switcher”?



[Fei-Fei Li, Andrej Karpathy, Justin Johnson]

Question:

What problems might you encounter with deeply nested functions? (3 min)

Visualizing Backprop during Training: Classification with 2-Layer Neural Network

- http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html
- Try playing around with this app to build intuition:
  - change datapoints to see how decision boundaries change
  - change network layer types, widths, activation functions, etc.
  - try shallower vs deeper