

Backpropagation

Most slides are from Matt Gormley

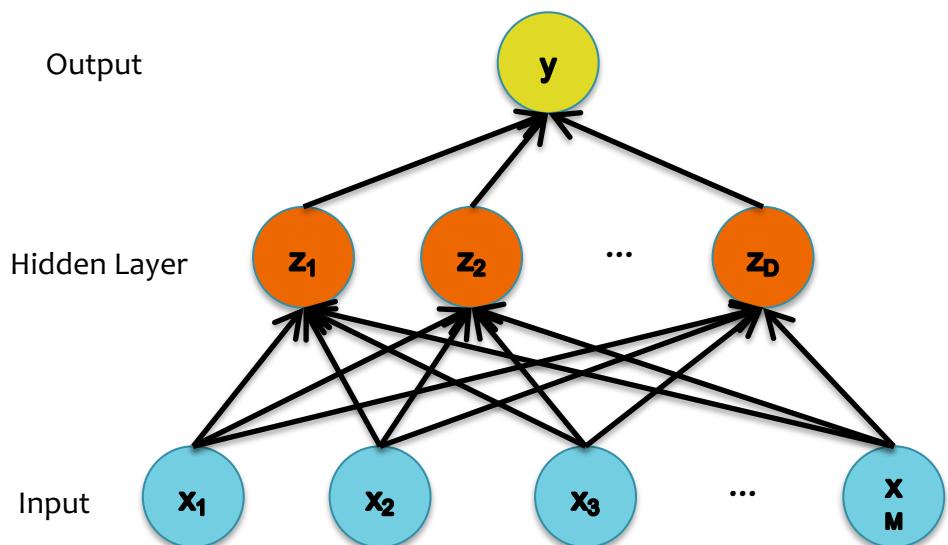
Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function

Activation Functions

Neural Network with sigmoid activation functions



(F) Loss

$$J = \frac{1}{2}(y - y^*)^2$$

(E) Output (sigmoid)

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

(D) Output (linear)

$$b = \sum_{j=0}^D \beta_j z_j$$

(C) Hidden (sigmoid)

$$z_j = \frac{1}{1+\exp(-a_j)}, \forall j$$

(B) Hidden (linear)

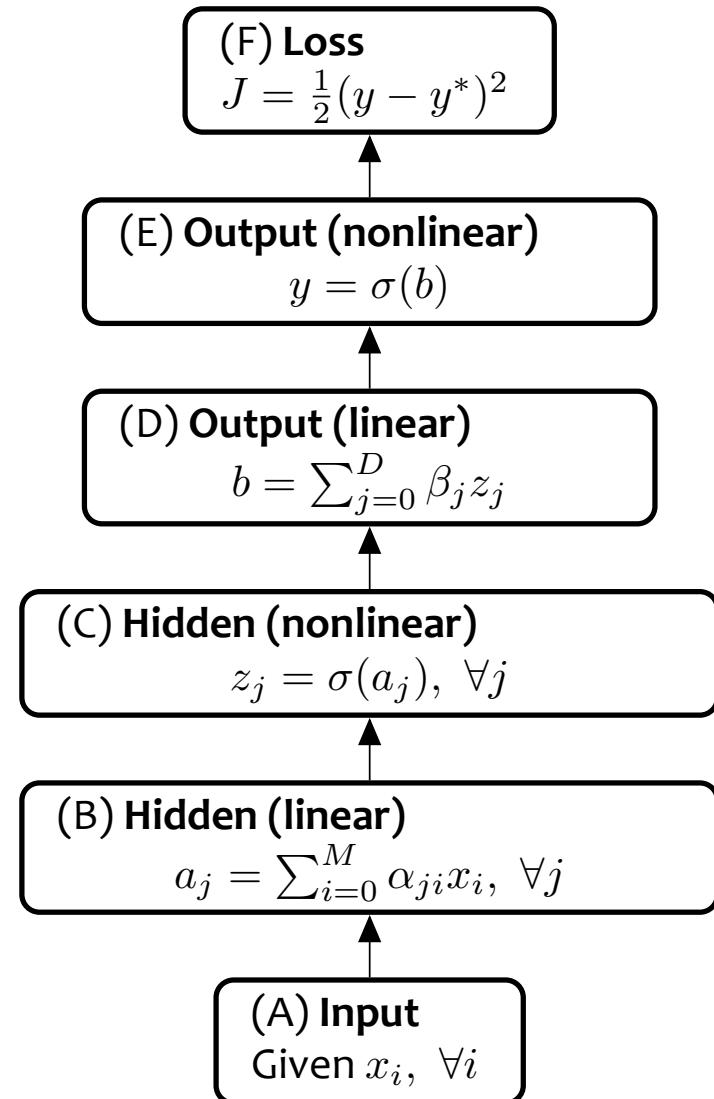
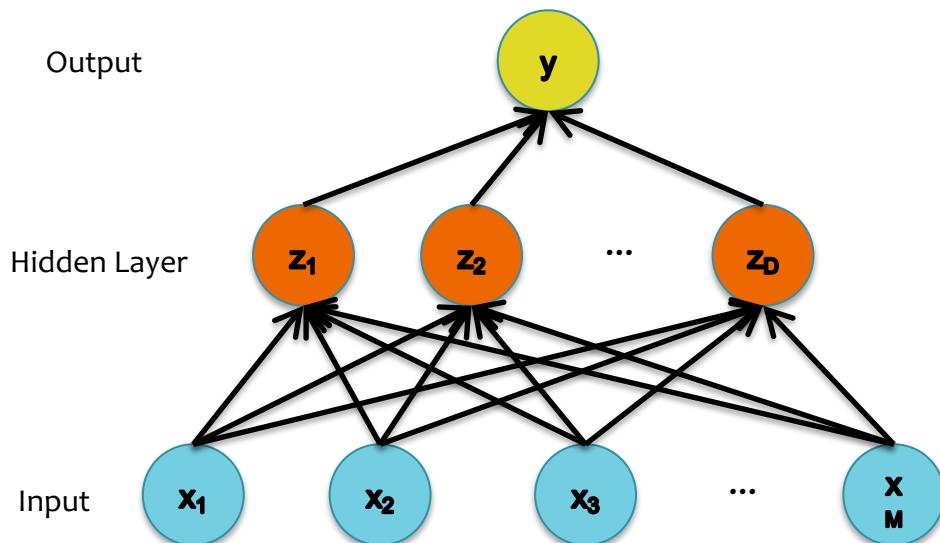
$$a_j = \sum_{i=0}^M \alpha_{ji} x_i, \forall j$$

(A) Input

Given $x_i, \forall i$

Activation Functions

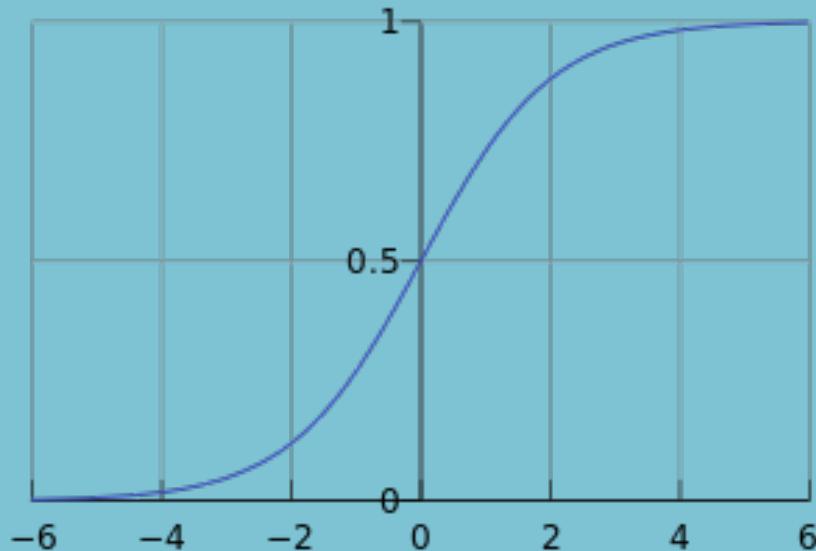
Neural Network with arbitrary nonlinear activation functions



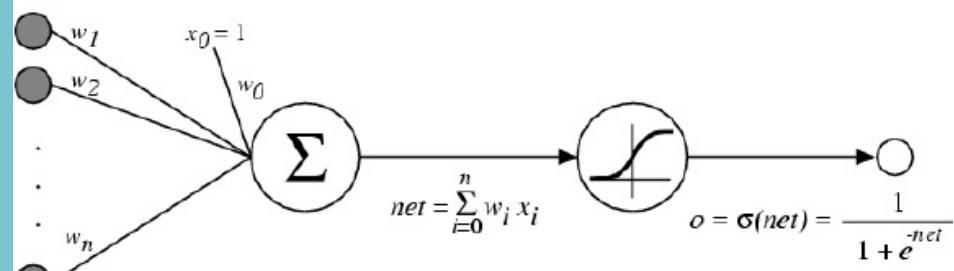
Activation Functions

Sigmoid / Logistic Function

$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

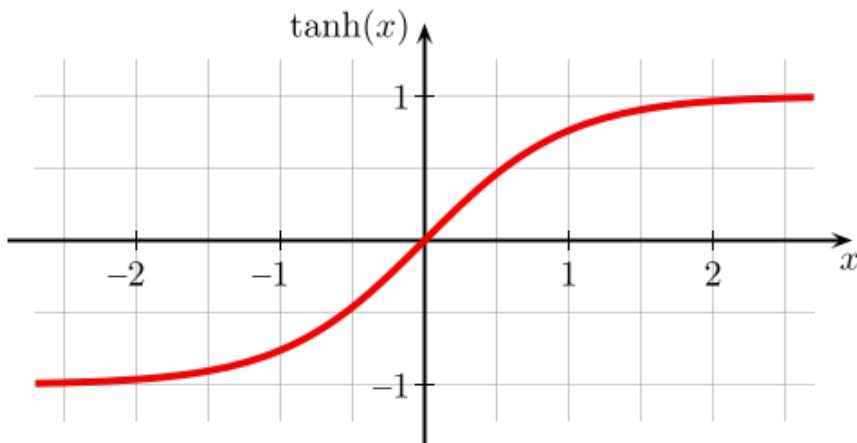


So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function...



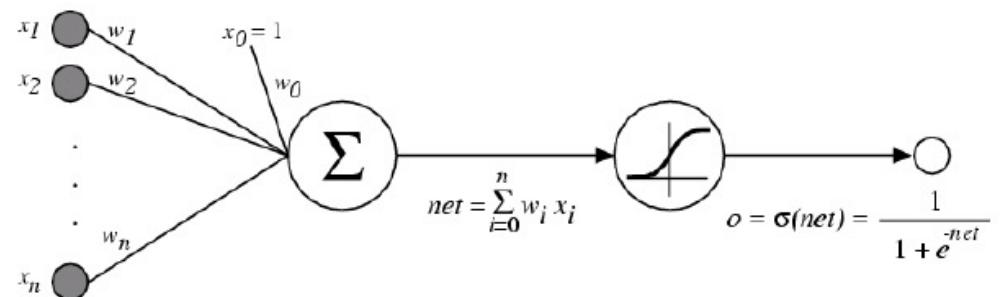
Activation Functions

- A new change: modifying the nonlinearity
 - The logistic is not widely used in modern ANNs



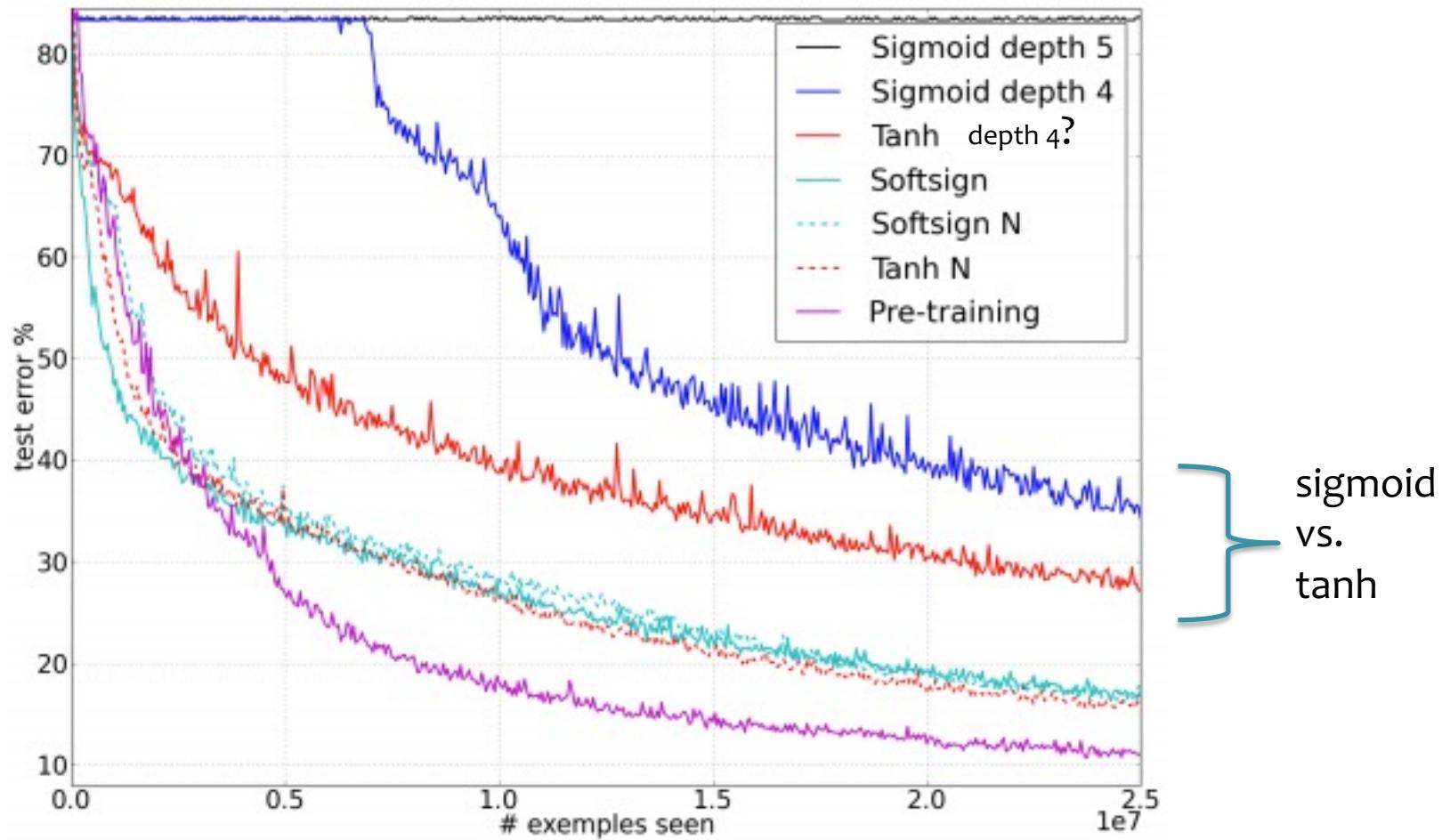
Alternate 1:
 \tanh

Like logistic function but
shifted to range $[-1, +1]$



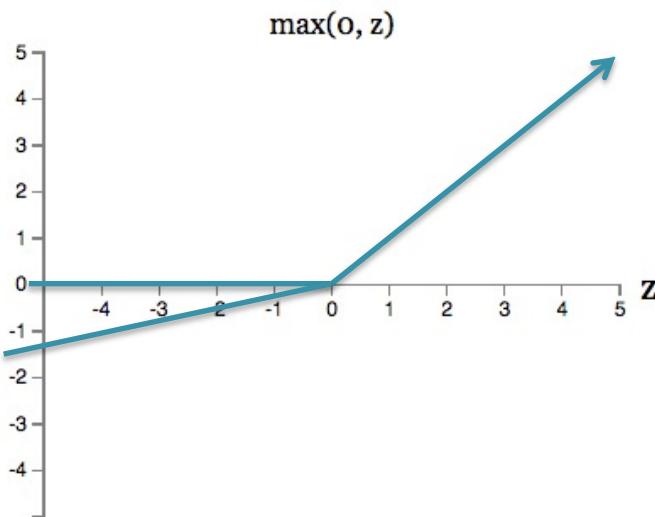
Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010



Activation Functions

- A new change: modifying the nonlinearity
 - reLU often used in vision tasks

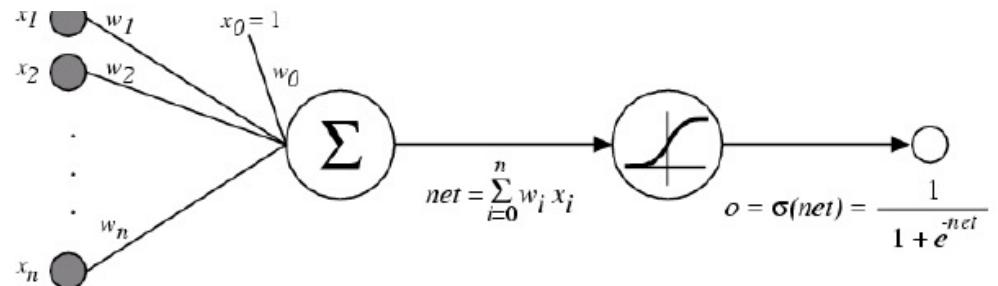


Alternate 2: rectified linear unit

Linear with a cutoff at zero

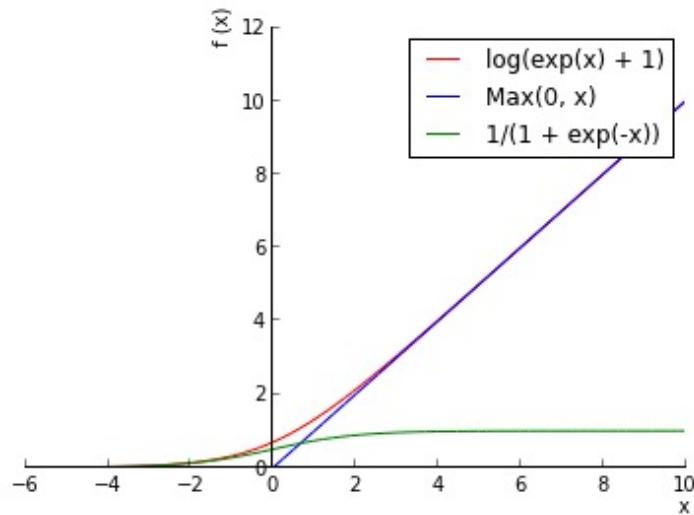
(Implementation: clip the gradient when you pass zero)

$$\max(0, w \cdot x + b).$$



Activation Functions

- A new change: modifying the nonlinearity
 - reLU often used in vision tasks



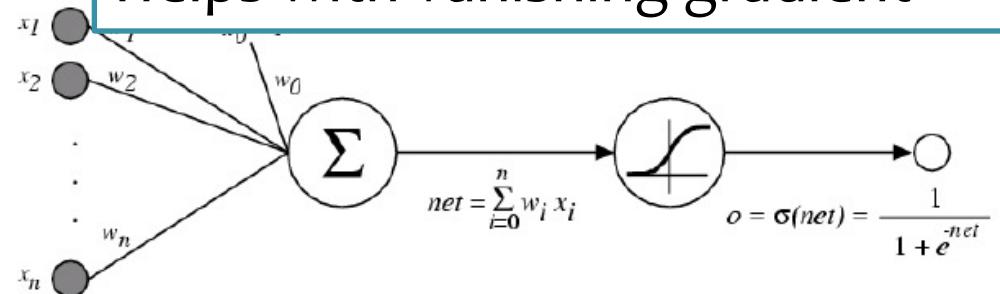
Alternate 2: rectified linear unit

Soft version: $\log(\exp(x)+1)$

Doesn't saturate (at one end)

Sparsifies outputs

Helps with vanishing gradient



Objective Functions for NNs

- Regression:
 - Use the same objective as Linear Regression
 - Quadratic loss (i.e. mean squared error)
- Classification:
 - Use the same objective as Logistic Regression
 - Cross-entropy (i.e. negative log likelihood)
 - This requires probabilities, so we add an additional “softmax” layer at the end of our network

Forward

Quadratic $J = \frac{1}{2}(y - y^*)^2$

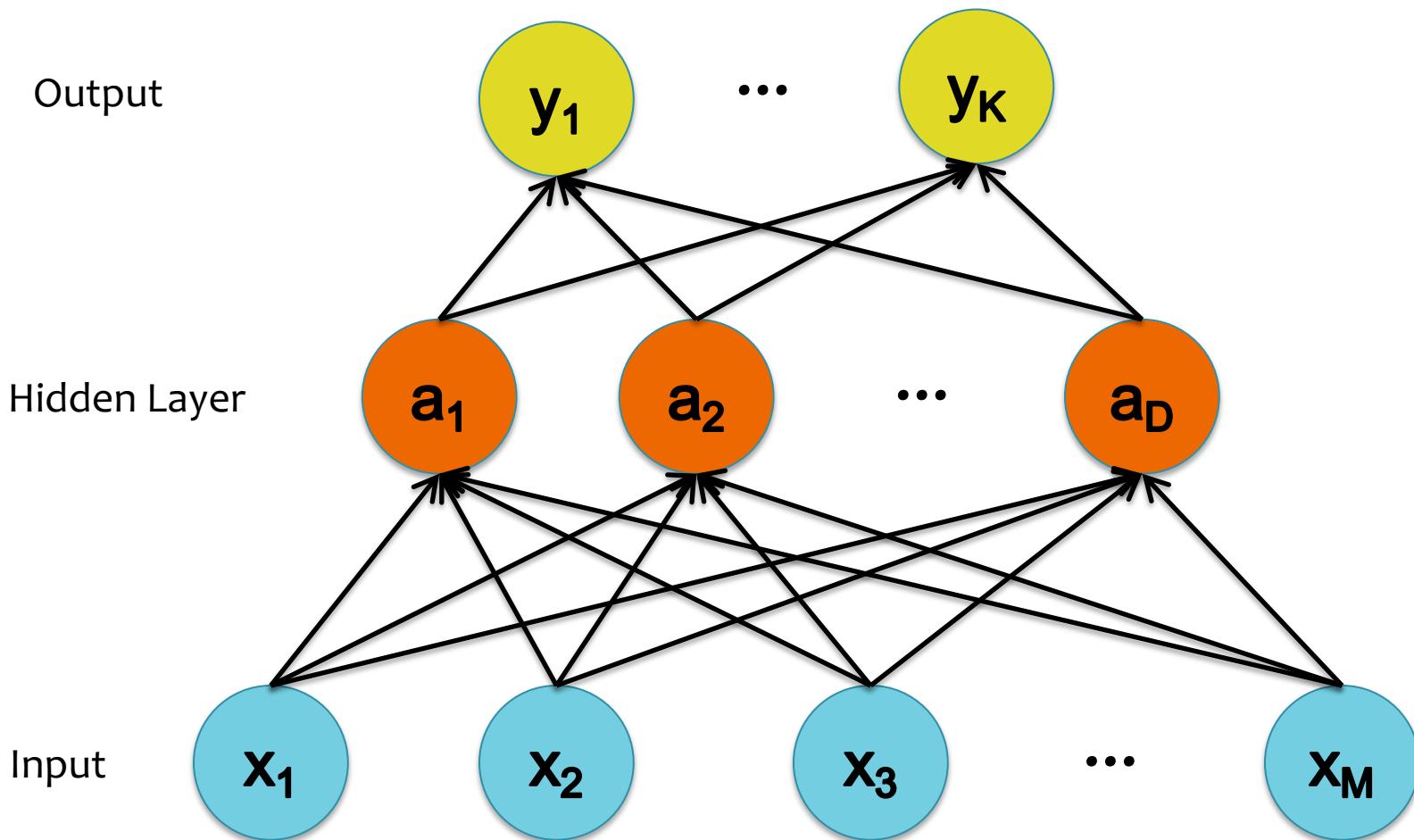
Cross Entropy $J = y^* \log(y) + (1 - y^*) \log(1 - y)$

Backward

$$\frac{dJ}{dy} = y - y^*$$

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

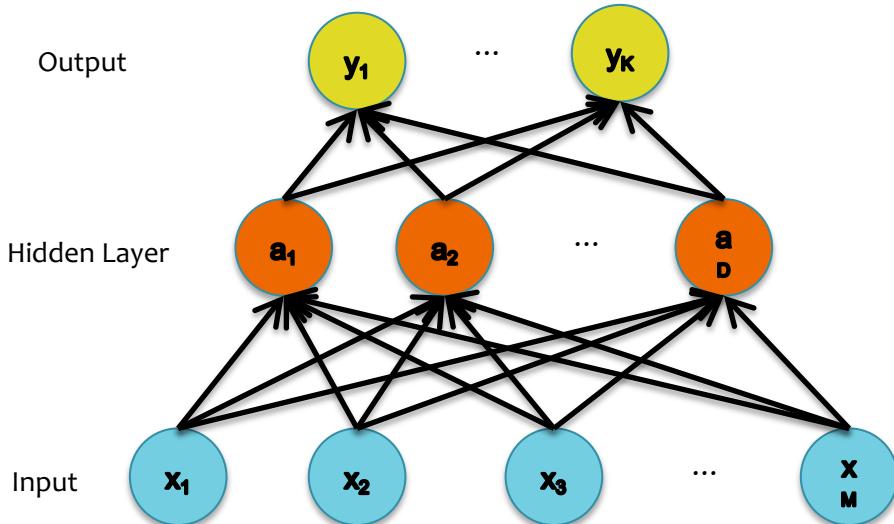
Multi-Class Output



Multi-Class Output

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^K \exp(b_l)}$$



(F) Loss

$$J = \sum_{k=1}^K y_k^* \log(y_k)$$

(E) Output (softmax)

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^K \exp(b_l)}$$

(D) Output (linear)

$$b_k = \sum_{j=0}^D \beta_{kj} z_j \quad \forall k$$

(C) Hidden (nonlinear)

$$z_j = \sigma(a_j), \quad \forall j$$

(B) Hidden (linear)

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i, \quad \forall j$$

(A) Input

Given $x_i, \forall i$

Cross-entropy vs. Quadratic loss

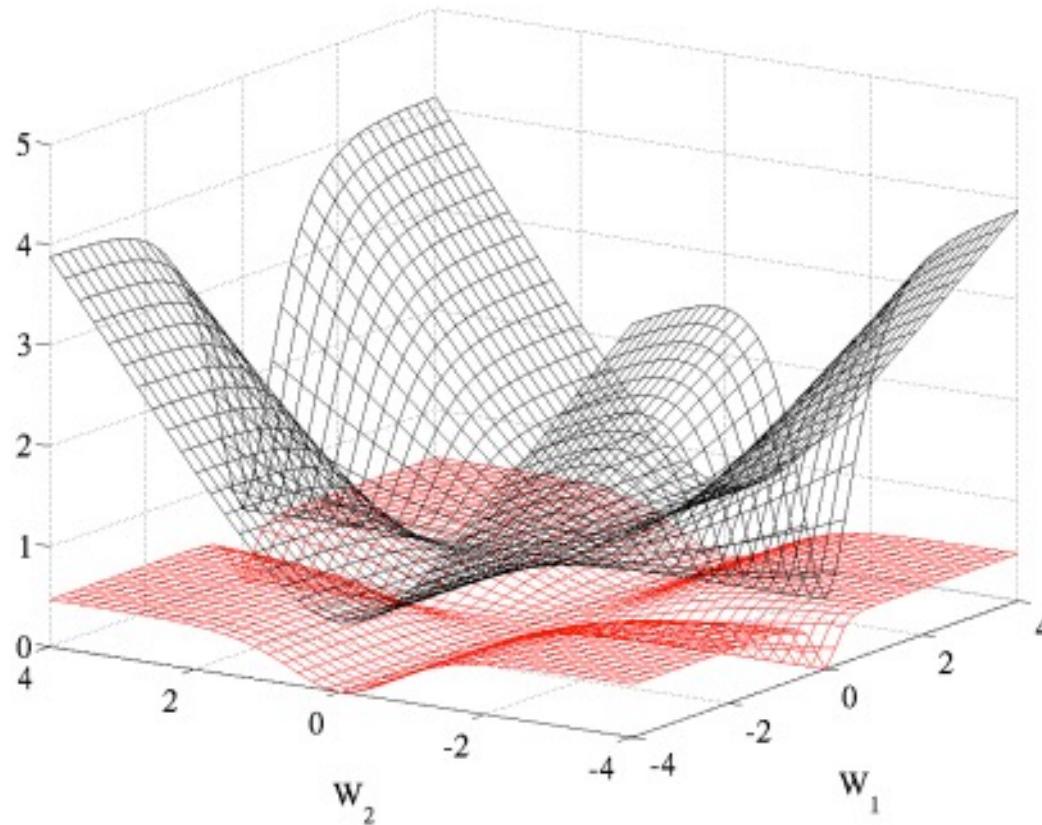


Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

- Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with SGD:

(take small steps
opposite the gradient)

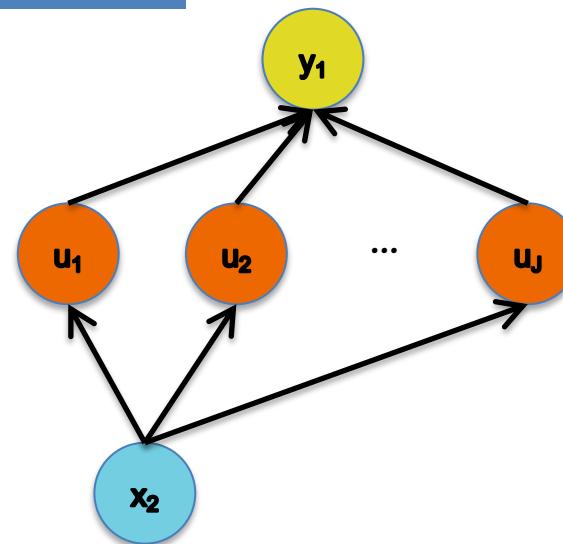
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

- **Question 1:**
When can we compute the gradients of the parameters of an arbitrary neural network?
- **Question 2:**
When can we make the gradient computation efficient?

Given: $y = g(\mathbf{u})$ and $\mathbf{u} = h(\mathbf{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^J \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Training

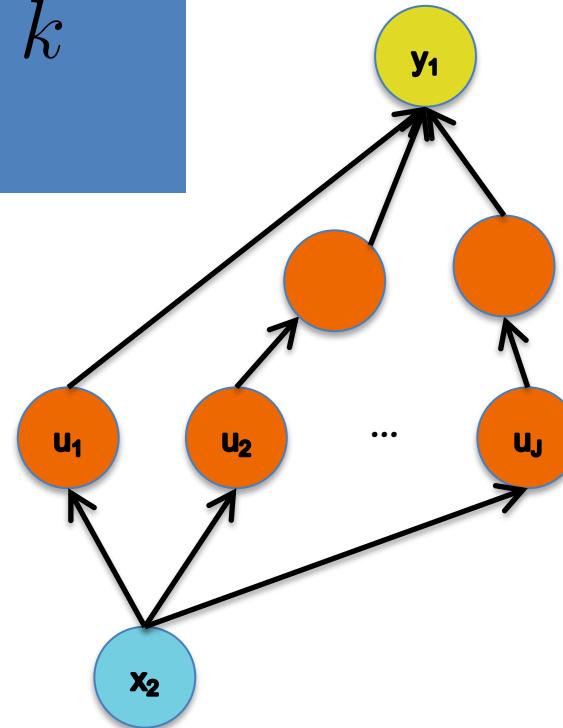
Chain Rule

Given: $y = g(\mathbf{u})$ and $\mathbf{u} = h(\mathbf{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^J \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$

Backpropagation
is just repeated
application of the
chain rule from
Calculus 101.



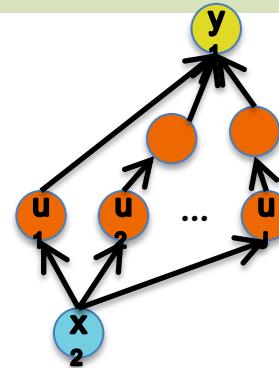
Training

Chain Rule

Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^J \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Backpropagation:

1. Instantiate the computation as a directed acyclic graph, where each intermediate quantity is a node
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node's intermediate quantity.
3. Initialize all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents

This algorithm is also called **automatic differentiation in the reverse-mode**

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Training

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Backward

$$\frac{dJ}{du} += -\sin(u)$$

Training

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Backward

$$\frac{dJ}{du} += -\sin(u)$$

$$\frac{dJ}{du_1} += \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1$$

$$\frac{dJ}{du_2} += \frac{dJ}{du} \frac{du}{du_2}, \quad \frac{du}{du_2} = 1$$

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Backward

$$\frac{dJ}{du} += -\sin(u)$$

$$\frac{dJ}{du_1} += \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1$$

$$\frac{dJ}{dt} += \frac{dJ}{du_1} \frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$$

$$\frac{dJ}{du_2} += \frac{dJ}{du} \frac{du}{du_2}, \quad \frac{du}{du_2} = 1$$

Training

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Backward

$$\frac{dJ}{du} += -\sin(u)$$

$$\frac{dJ}{du_1} += \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1$$

$$\frac{dJ}{dt} += \frac{dJ}{du_1} \frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$$

$$\frac{dJ}{dt} += \frac{dJ}{du_2} \frac{du_2}{dt}, \quad \frac{du_2}{dt} = 3$$

$$\frac{dJ}{du_2} += \frac{dJ}{du} \frac{du}{du_2}, \quad \frac{du}{du_2} = 1$$

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Backward

$$\frac{dJ}{du} += -\sin(u)$$

$$\frac{dJ}{du_1} += \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1$$

$$\frac{dJ}{dt} += \frac{dJ}{du_1} \frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$$

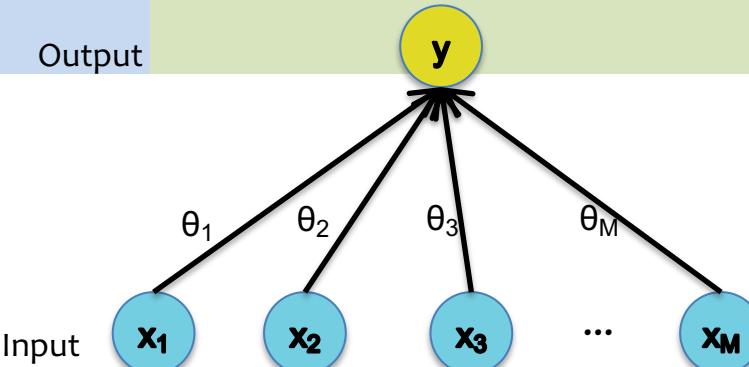
$$\frac{dJ}{dt} += \frac{dJ}{du_2} \frac{du_2}{dt}, \quad \frac{du_2}{dt} = 3$$

$$\frac{dJ}{dx} += \frac{dJ}{dt} \frac{dt}{dx}, \quad \frac{dt}{dx} = 2x$$

Training

Backpropagation

Case 1:
**Logistic
Regression**



Forward

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

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

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

Backward

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

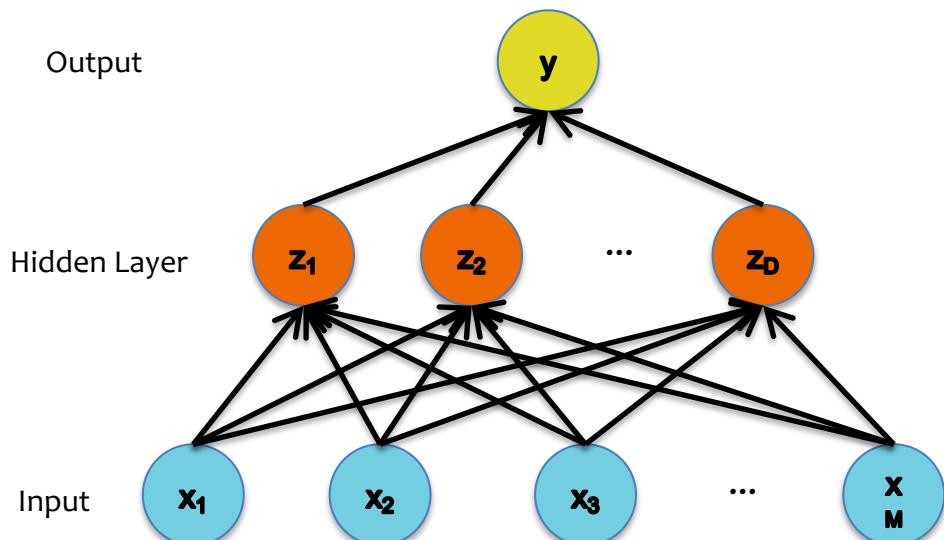
$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

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

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

Training

Backpropagation



(E) Output (sigmoid)

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

(D) Output (linear)

$$b = \sum_{j=0}^D \beta_j z_j$$

(C) Hidden (sigmoid)

$$z_j = \frac{1}{1+\exp(-a_j)}, \forall j$$

(B) Hidden (linear)

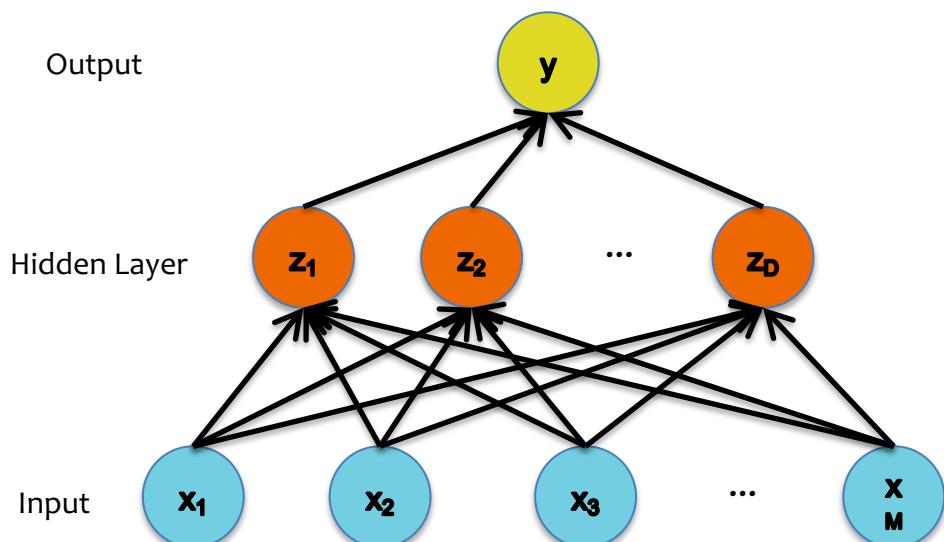
$$a_j = \sum_{i=0}^M \alpha_{ji} x_i, \forall j$$

(A) Input

Given $x_i, \forall i$

Training

Backpropagation



(F) Loss

$$J = \frac{1}{2}(y - y^*)^2$$

(E) Output (sigmoid)

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

(D) Output (linear)

$$b = \sum_{j=0}^D \beta_j z_j$$

(C) Hidden (sigmoid)

$$z_j = \frac{1}{1+\exp(-a_j)}, \forall j$$

(B) Hidden (linear)

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i, \forall j$$

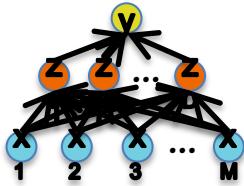
(A) Input

Given $x_i, \forall i$

Training

Backpropagation

Case 2:
**Neural
Network**



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

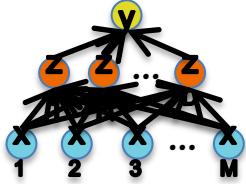
Backward

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

Training

Backpropagation

Case 2:
**Neural
Network**



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

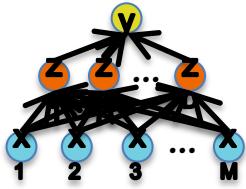
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

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

Training

Backpropagation

Case 2:
**Neural
Network**



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

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

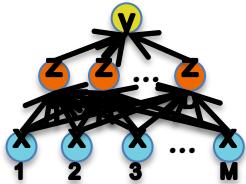
$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

Training

Backpropagation

Case 2:
**Neural
Network**



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

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

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

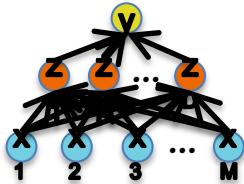
$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

Training

Backpropagation

Case 2:
**Neural
Network**



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

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

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

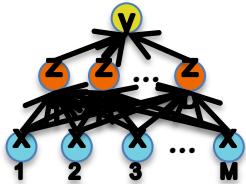
$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

Training

Backpropagation

Case 2:
**Neural
Network**



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

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

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

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

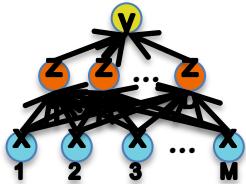
$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

Training

Backpropagation

Case 2:
**Neural
Network**



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

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

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

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

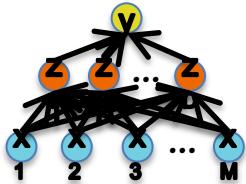
$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

Training

Case 2: Neural Network



Forward

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

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

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backpropagation

Backward

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

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

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^D \alpha_{ji}$$

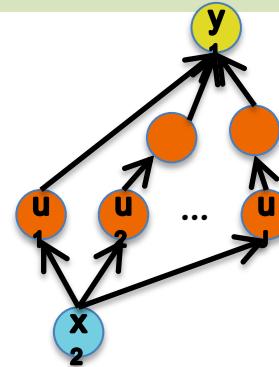
Training

Chain Rule

Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^J \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Backpropagation:

1. Instantiate the computation as a directed acyclic graph, where each intermediate quantity is a node
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node's intermediate quantity.
3. Initialize all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents

This algorithm is also called **automatic differentiation in the reverse-mode**

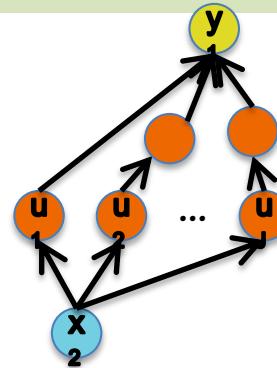
Training

Chain Rule

Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^J \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Backpropagation:

1. Instantiate the computation as a directed acyclic graph, where each node represents a Tensor.
2. At each node, store (a) the quantity computed in the forward pass and (b) the partial derivatives of the goal with respect to that node's Tensor.
3. Initialize all partial derivatives to 0.
4. Visit each node in reverse topological order. At each node, add its contribution to the partial derivatives of its parents

This algorithm is also called automatic differentiation in the reverse-mode

Training

Backpropagation

Case 2:

Module 5

Forward

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

Backward

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

Module 4

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

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

Module 3

$$b = \sum_{j=0}^D \beta_j z_j$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

Module 2

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

Module 1

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^D \alpha_{ji}$$

Background

1. Given training data

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of the

- Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

A Recipe for Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

(opposite the gradient)

$$\theta^{(t)} \xrightarrow{(t)} -\eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Summary

1. Neural Networks...

- provide a way of learning features
- are highly nonlinear prediction functions
- (can be) a highly parallel network of logistic regression classifiers
- discover useful hidden representations of the input

2. Backpropagation...

- provides an efficient way to compute gradients
- is a special case of reverse-mode automatic differentiation