CS60010: Deep Learning

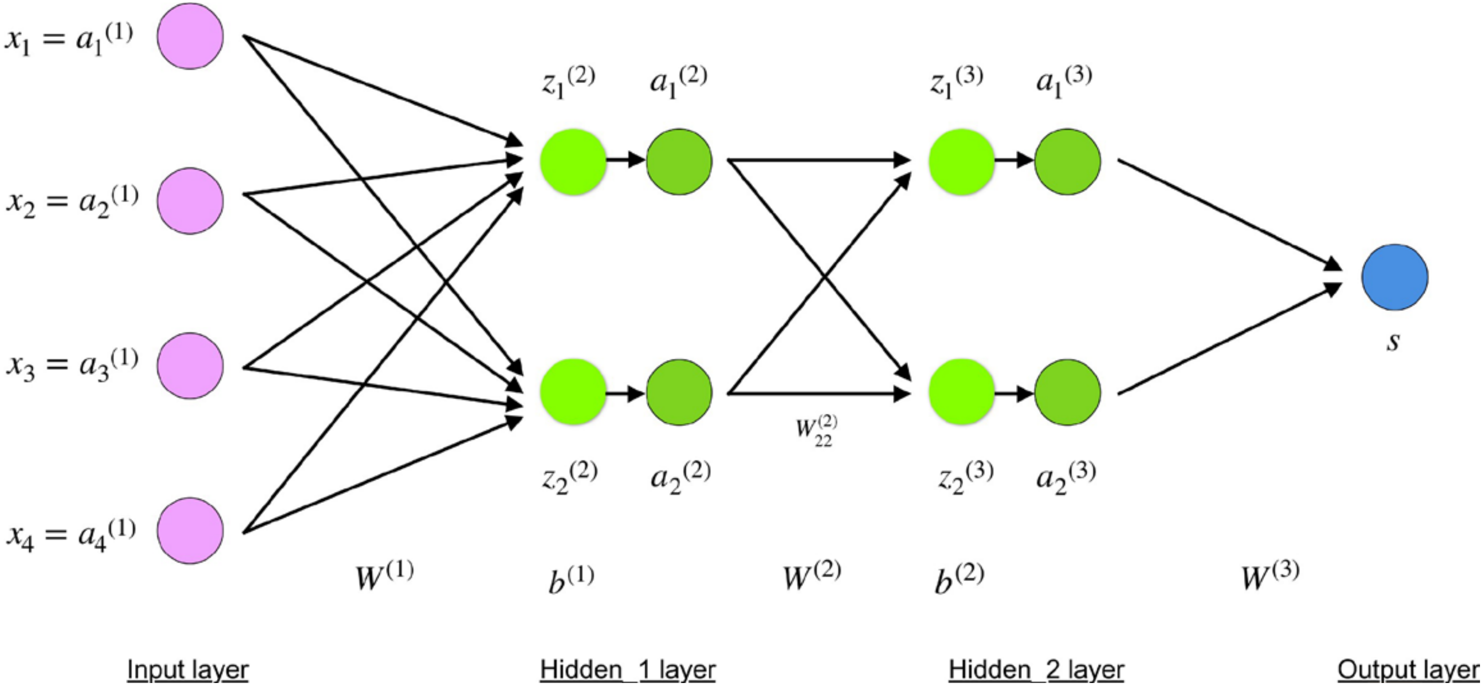
Spring 2023

Sudeshna Sarkar

Multilayer Perceptron – Part 2

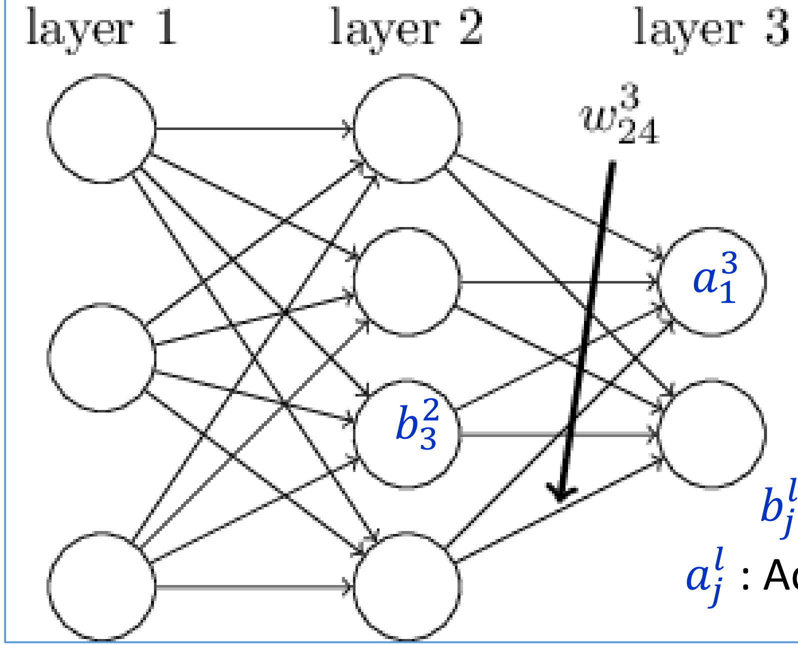
25 Jan 2023

Multilayer Neural Network



https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

Notations



w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

b_j^l : Bias for jth neuron in lth layer.

a_j^l : Activation of the jth neuron in the lth layer.

$$a_j^l = g\left(\sum_k w_{jk}^l \, a_k^{l-1} + b_j^l\right)$$

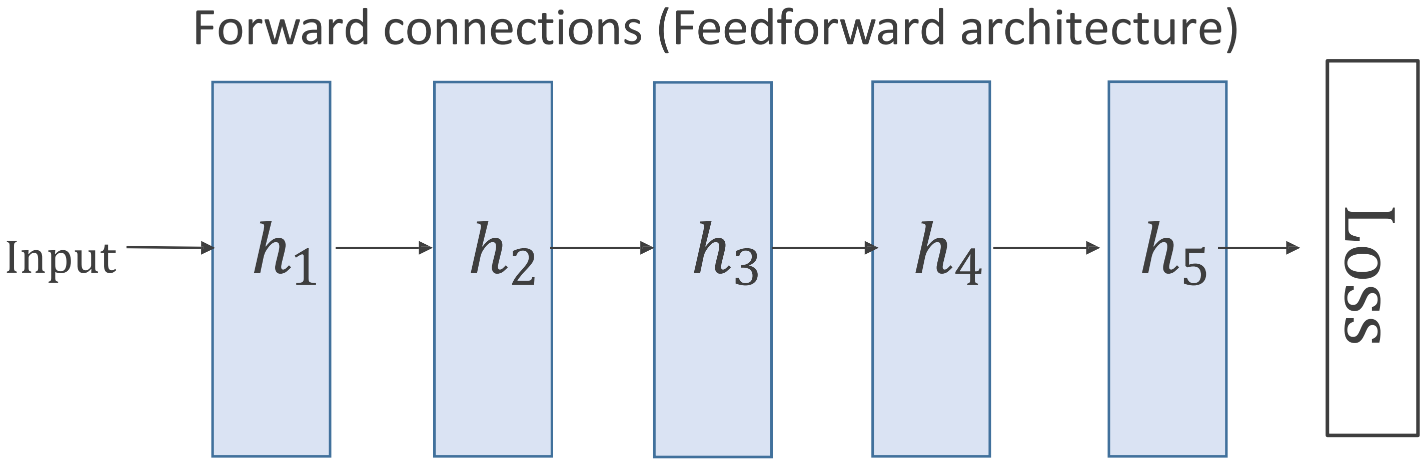
Vectorized form: $a^l = g\left(w^l a^{l-1} + b^l\right)$

$$z^l = w^l a^{l-1} + b^l$$

$$a^l = g\left(z^l\right)$$

Neural networks in blocks

- We can visualize $a_L = h_L \circ h_{L-1} \circ \cdots \circ h_1(x)$ as a cascade of blocks.



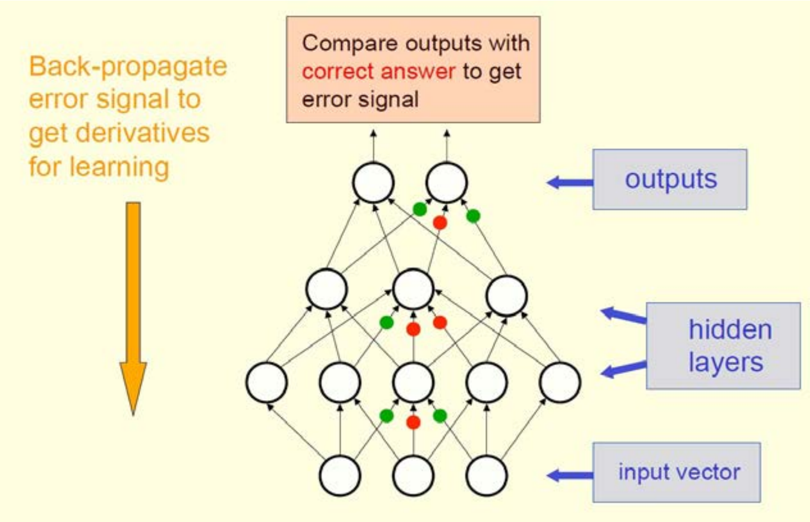
The activation functions must be **1st-order differentiable (almost) everywhere**

Backpropagation

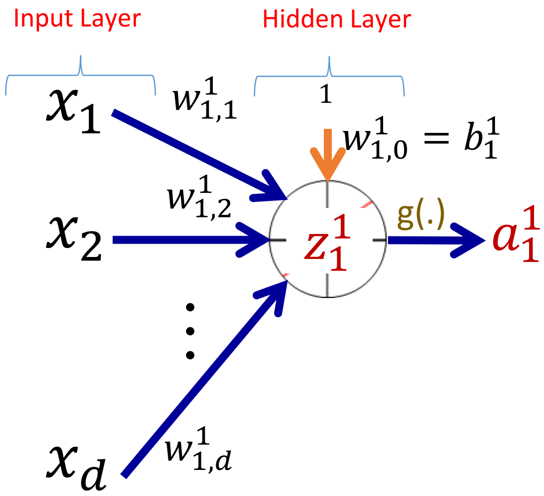
- Feedforward Propagation: Accept input $x^{(i)}$, pass through intermediate stages and obtain output $\hat{y}^{(i)}$
- During Training: Compute scalar cost J(θ)

$$J(\theta) = \sum_i L\left(NN\left(x^{(i)}; \theta\right), y^{(i)}\right)$$

- Backpropagation allows information to flow backwards from cost to compute the gradient



Multilayer Neural Network

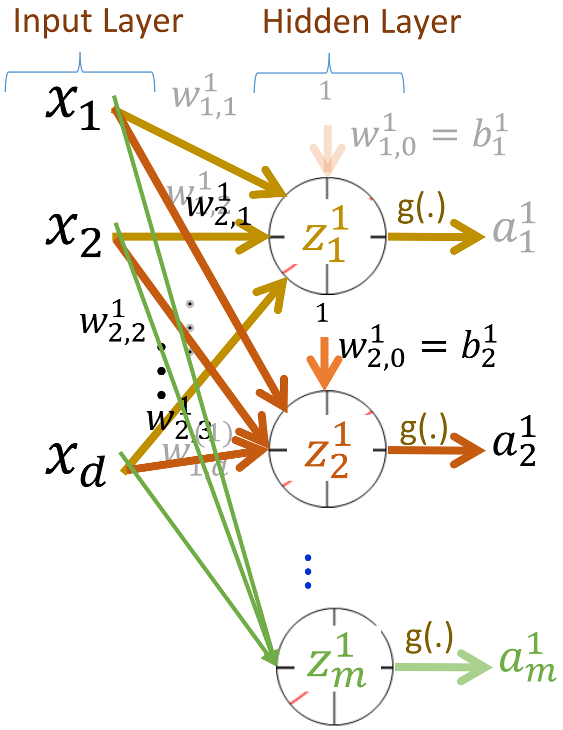


$$z_1^1 = b_1^1 + \sum_{i=1}^{d} w_{1,i}^1\, x_i = [\boldsymbol{w}_1^1 b_1^1]\begin{bmatrix}\boxed{\boldsymbol{x}}\\ 1\end{bmatrix}$$

$$[x_1\ x_2 \ldots x_d]^T$$

$$a_1^1 = g(z_1^1)$$

Multilayer Neural Network

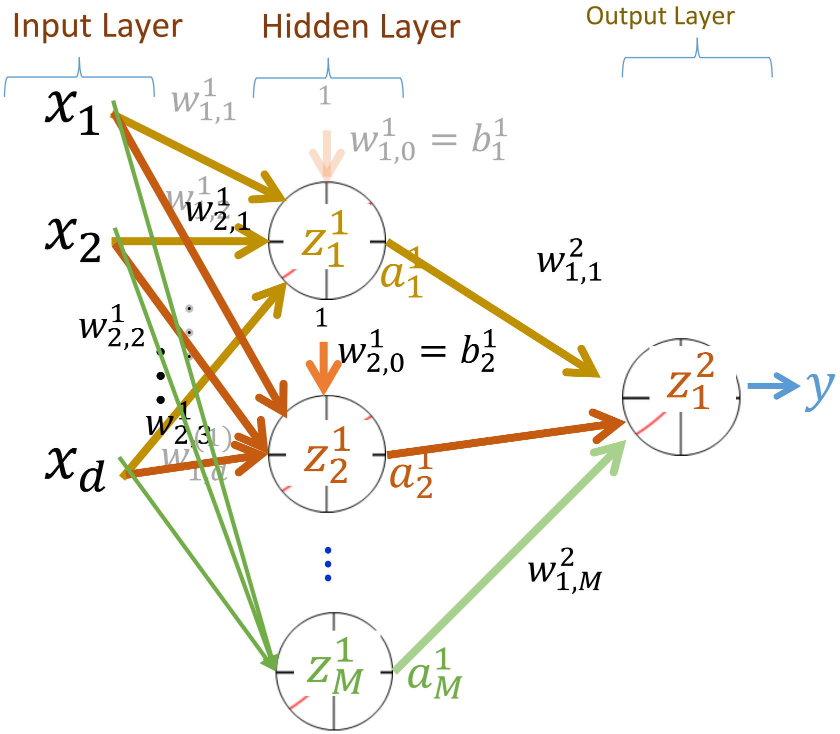


$$
\left.
\begin{aligned}
z_1^1 &= [\boldsymbol{w}_1^1 b_1^1]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}\\
z_2^1 &= [\boldsymbol{w}_2^1 b_2^1]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}\\
&\vdots\\
z_M^1 &= [\boldsymbol{w}_m^1 b_2^1]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}
\end{aligned}
\right\}
\underbrace{\begin{bmatrix}z_1^1\\z_2^1\\\vdots\\z_m^1\end{bmatrix} = \begin{bmatrix}\boldsymbol{w}_1^1 & b_1^1\\\boldsymbol{w}_2^1 & b_2^1\\\vdots & \vdots\\\boldsymbol{w}_m^1 & b_m^1\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}}_{\boldsymbol{z}^1 = [\boldsymbol{W}^1\boldsymbol{b}^1]\begin{bmatrix}\boldsymbol{x}\\1\end{bmatrix}}
$$

$$
\left.\begin{bmatrix}a_1^1\\a_2^1\\\vdots\\a_m^1\end{bmatrix} = \begin{bmatrix}g(z_1^1)\\g(z_2^1)\\\vdots\\g(z_m^1)\end{bmatrix}\right\}\quad \boldsymbol{a}^1 = \boldsymbol{g}\left(\boldsymbol{z}^{(1)}\right)
$$

$$
\boxed{\begin{aligned}
a^{(0)} &= x\\
z^{(1)} &= \boldsymbol{w}^{(1)}\boldsymbol{a}^{(0)}\\
a^{(1)} &= g\left(\boldsymbol{z}^{(1)}\right)
\end{aligned}}
$$

Multilayer Neural Network



Output Layer Pre-activation

$$z_1^2 = [\mathbf{w}_1^2 \; b_1^2]\begin{bmatrix}\mathbf{a}^1\\ 1\end{bmatrix}$$

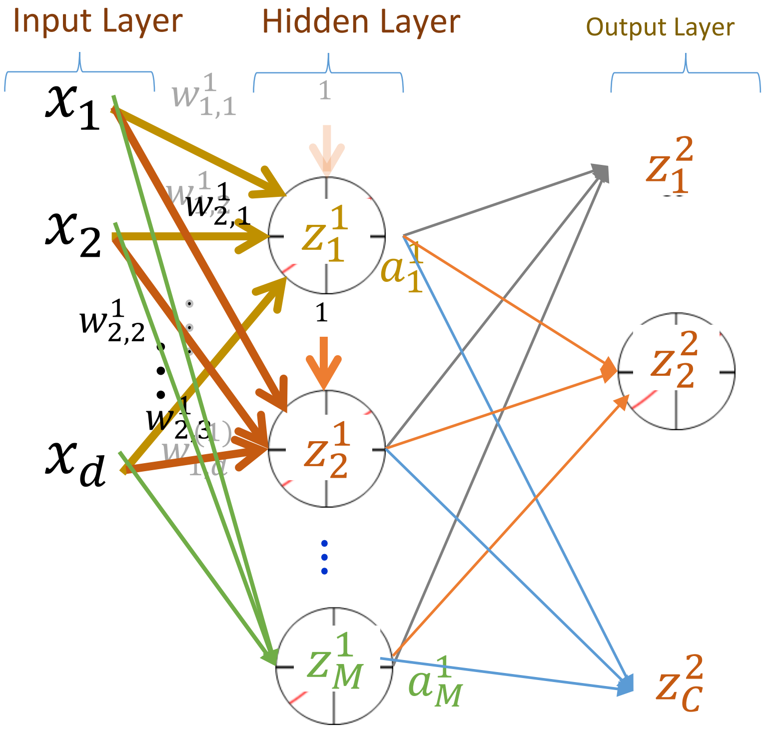
Output Layer Activation

$$y_1 = o\left(\mathbf{z}_1^2\right)$$

output

- Sigmoid for 2-class classification
- Softmax for multi-class classification
- Linear for regression

Multilayer Neural Network

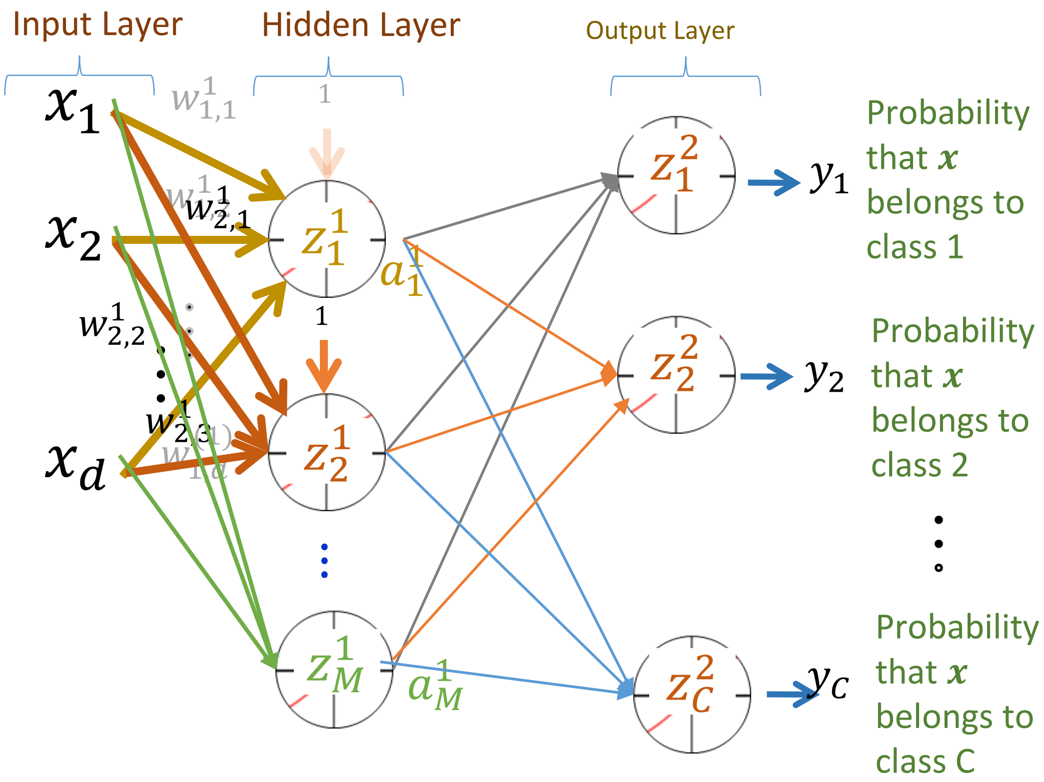


$$y_1 = o_1(z^2_1) = \frac{\exp(z^2_1)}{\sum_c \exp(z^2_c)}$$

$$y_2 = o_2(z^2_2) = \frac{\exp(z^2_2)}{\sum_c \exp(z^2_c)}$$

$$y_C = o_C(z^2_C) = \frac{\exp(z^2_C)}{\sum_c \exp(z^2_c)}$$

Training a Neural Network – Loss Function



Aim to maximize the probability corresponding to the correct class for any example $\boldsymbol{x}$

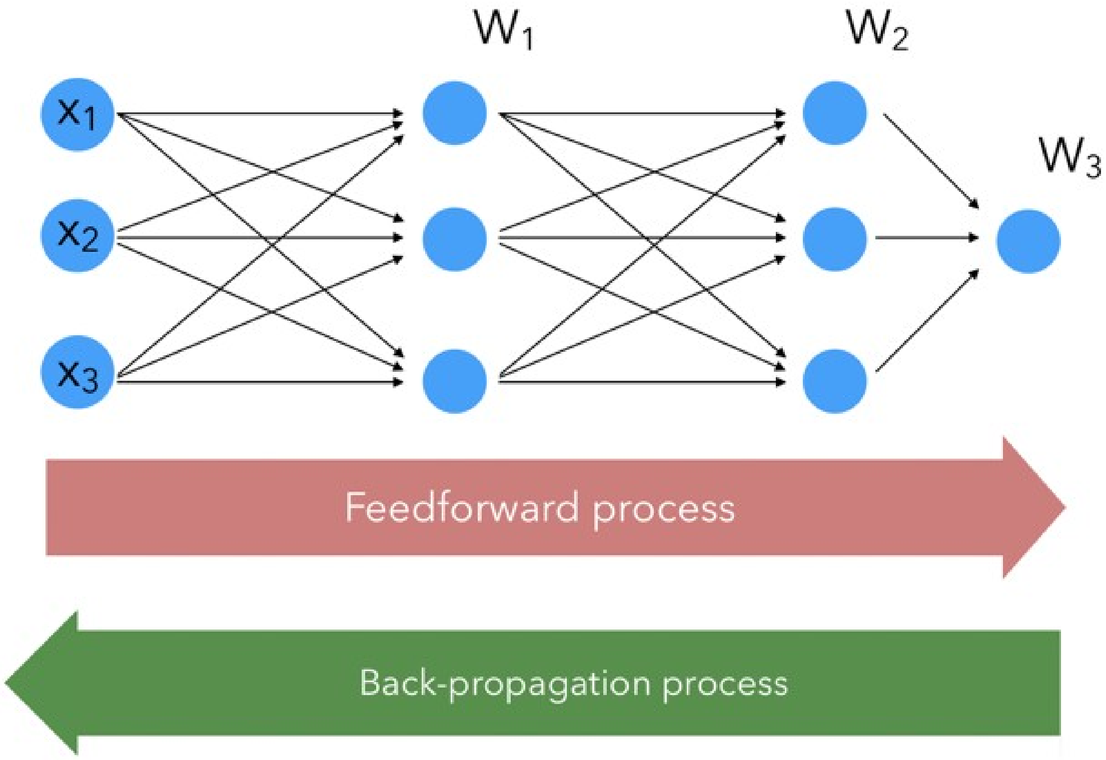
$$\max \boldsymbol{y}_c$$
$$\equiv \max\,(log\ \boldsymbol{y}_c)$$
$$\equiv \min\,(-log\ \boldsymbol{y}_c)$$

Can be equivalently expressed as

$$-\sum_i \prod_{i=c} \log(y_i)$$

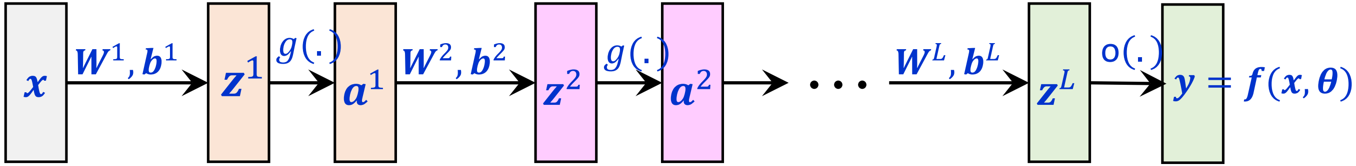
known as <u>cross-entropy loss</u>

Forward-Backward Passes



Forward Pass

$\boldsymbol{\theta}$ is the collection of all learnable parameters i.e., all $\boldsymbol{W}$ and $\boldsymbol{b}$

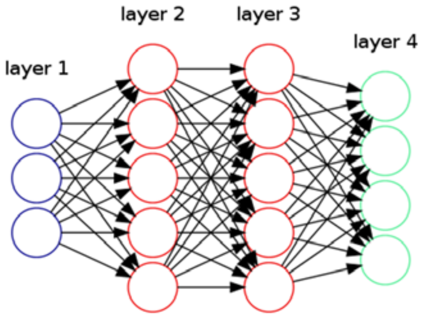


Hidden layer pre-activation:

For $l = 1, \ldots, L$; $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$

Hidden layer activation:

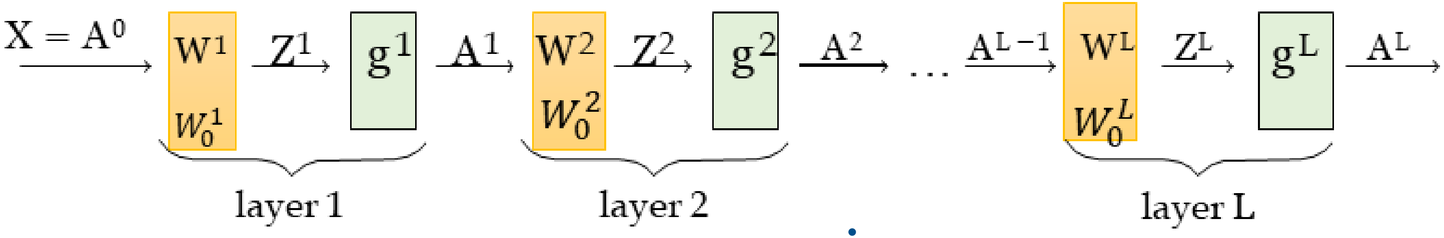
For $l = 1, \ldots, L-1$; $\boldsymbol{a}^{(l)} = g(\boldsymbol{z}^{(l)})$



Output layer activation:

For $l = L$; $\boldsymbol{y} = \boldsymbol{a}^{(L)} = o(\boldsymbol{z}^{(L)}) = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\theta})$

Error back-propagation



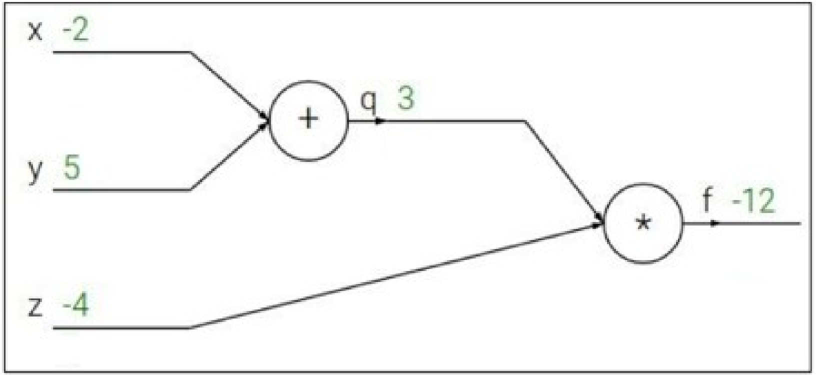
1. Compute Loss
2. Compute the derivative of the L w.r.t. the final output of the network A^L
3. Compute the derivative of L w.r.t. the pre-activation Z^L
4. Compute the derivative of L wrt W^L
5. Then compute the derivative of the L w.r.t. the final output of the network A^{L-1}
6. Then compute the derivative of L w.r.t. the pre-activation Z^{L-1}
7. Compute the derivative of L wrt W^{L-1}

…. ….

Consider following computational graph:

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

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

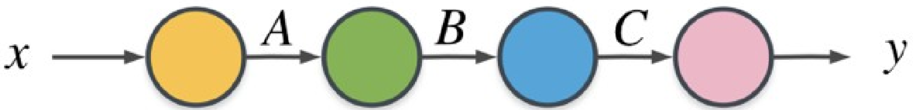


$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

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

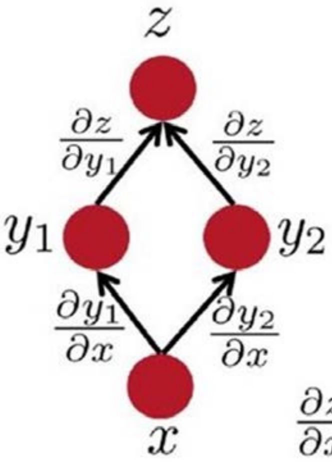
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial C} \times \frac{\partial C}{\partial B} \times \frac{\partial B}{\partial A} \times \frac{\partial A}{\partial x}$$

Multiple Path



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Backpropagation is just repeated application of the chain rule.

Computation Graph and Backpropagation

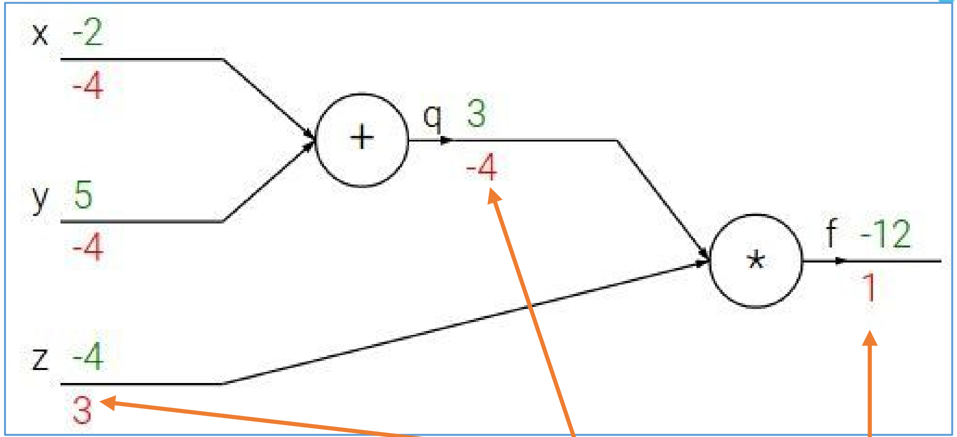
$$f(x,y,z) = (x+y)z$$

x = -2, y = 5, z = -4

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

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

Find $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\frac{\partial f}{\partial q}$ $\frac{\partial f}{\partial z}$ $\frac{\partial f}{\partial f}$

Computation Graph and Backpropagation

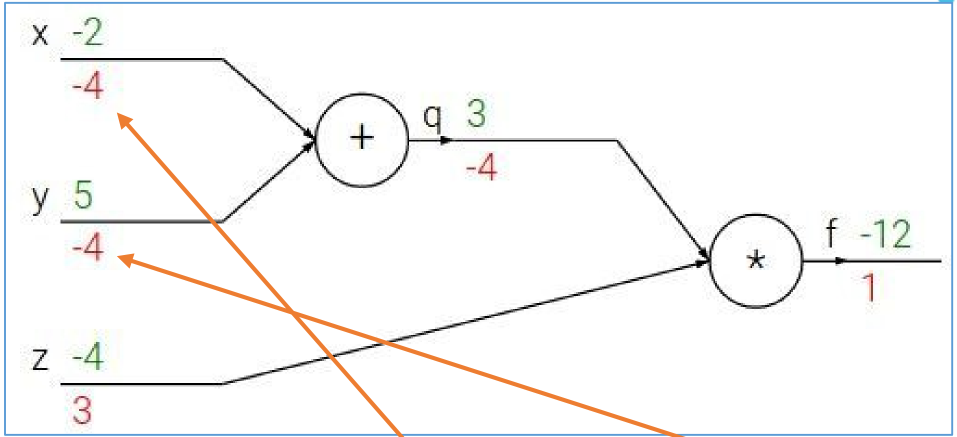
$$f(x, y, z) = (x + y)z$$

x = -2, y = 5, z = -4

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

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

Find $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



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

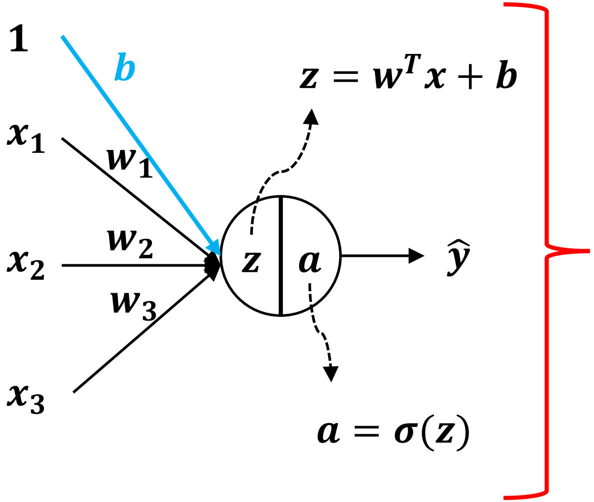
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

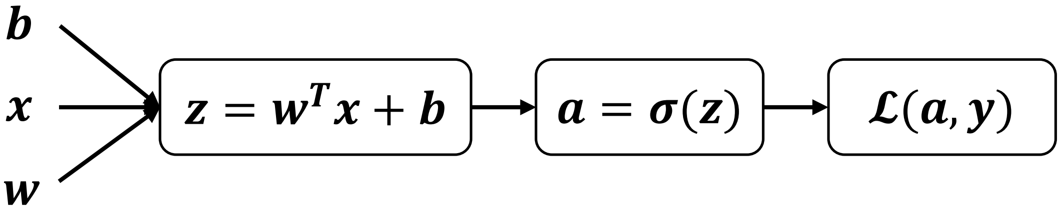
Upstream gradient

Local gradient

The Computation Graph of Logistic Regression

Let us translate logistic regression into a computation graph

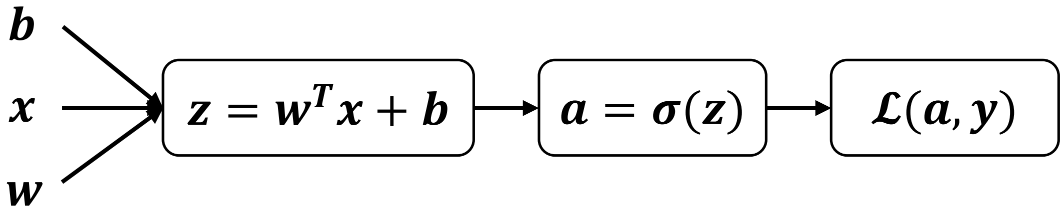




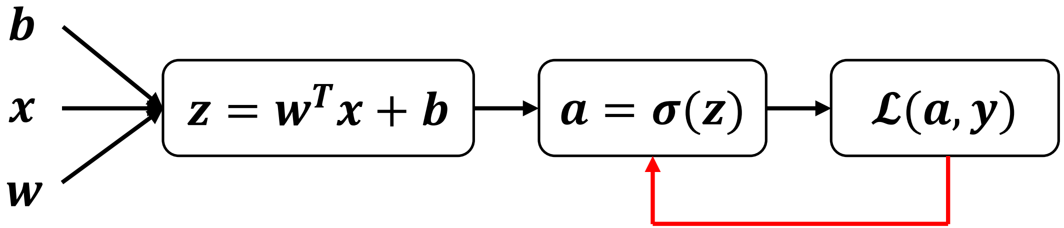
Where $b = 1$, $w = [w_1, w_2, w_3]$, $x = [x_1, x_2, x_3]$, and $\mathcal{L}(a, y)$ is the cost (or ***loss***) function

Forward Propagation

- The loss function can be computed by moving from left to right



Backward Propagation

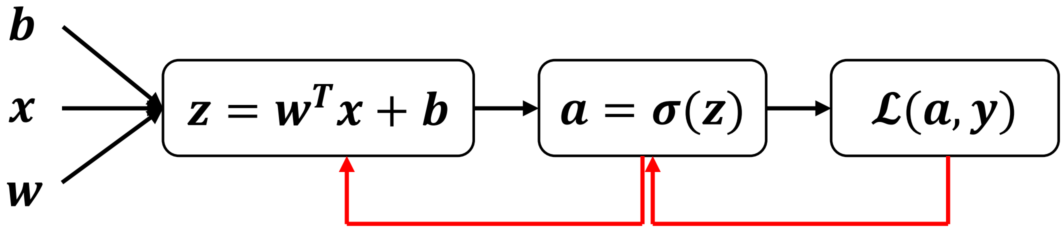


Partial derivative of $\mathcal{L}$ with respect to a

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial}{\partial a}\big(-y\log(a) - (1-y)\log(1-a)\big)$$

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

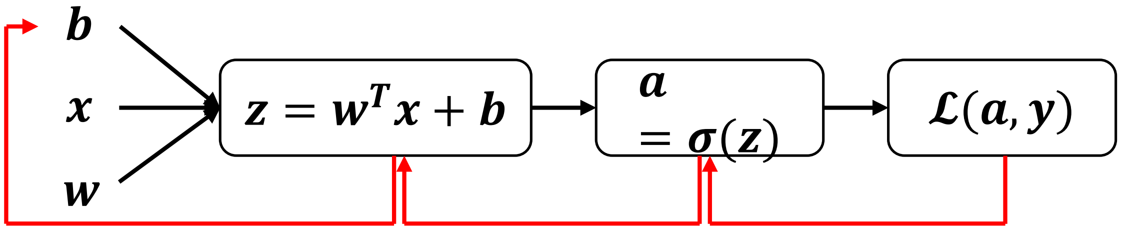
Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{(1-y)}{(1-a)} \right) \times \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{(1-y)}{(1-a)} \right) \times \; a(1-a)$$

$$= a - y$$

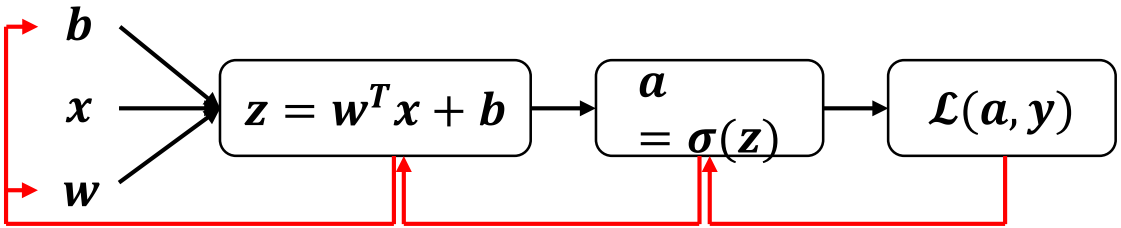
Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial b} = \text{Partial derivative of } \mathcal{L} \text{ with respect to } b$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial b} = (a - y) \times \frac{\partial z}{\partial b} = (a - y) \times 1 = (a - y)$$

Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w} = (a - y) \times \frac{\partial z}{\partial w} = (a - y)x$$

Backward Propagation: Summary

- Here is the summary of the gradients in logistic regression:

$$\boldsymbol{dz} = \frac{\partial \mathcal{L}}{\partial z} = a - y$$

$$\boldsymbol{db} = \frac{\partial \mathcal{L}}{\partial b} = a - y$$

$$\boldsymbol{dw} = \frac{\partial \mathcal{L}}{\partial w} = (a - y)x$$

Gradient Descent For Logistic Regression

Outline:

- Have a loss function $\boldsymbol{\mathcal{L}(w, b)}$
- Start off with some guesses for $\boldsymbol{w_1}, \dots, \boldsymbol{w_m}$

- Repeat until convergence{

$$w_j = w_j - \eta \frac{\partial \boldsymbol{\mathcal{L}(w, b)}}{\partial\, w_j}$$

$$b = b - \eta \frac{\partial \boldsymbol{\mathcal{L}(w, b)}}{\partial b}$$

}

Gradient Descent For Logistic Regression

- **Outline**:
  - Repeat until convergence{

Assuming n examples

$$for\ i = 1\ to\ n:$$

Forward propagation

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$

Backward propagation

$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw = dw + dz^{(i)} x^{(i)}$$
$$db = db + dz^{(i)}$$

Outside the loop

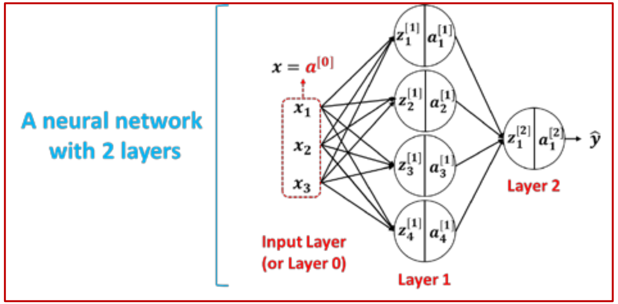
$$dw = dw/n$$
$$db = db/n$$
$$w = w - \eta dw$$
$$b = b - \eta db$$

}

Vectorized version

$$Z = w^T X + b$$
$$A = \sigma(Z)$$
$$dZ = A - Y$$
$$dw = \frac{1}{n} X dZ^T$$
$$db = \frac{1}{n} \sum_{i=1}^{n} dz^{(i)}$$
$$w = w - \eta dw$$
$$b = b - \eta db$$

Computation Graph of a 2-layer Neural Network

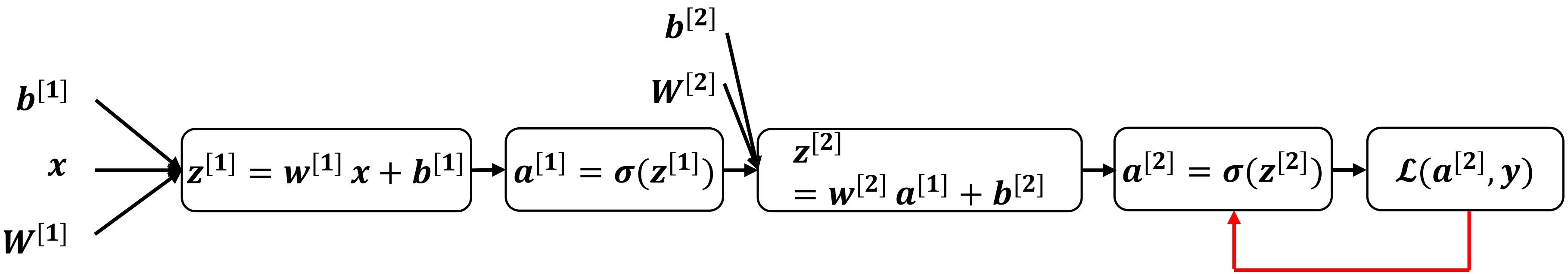


$$\boldsymbol{b}^{[1]},\ \boldsymbol{x},\ \boldsymbol{W}^{[1]} \rightarrow \boxed{\boldsymbol{z}^{[1]} = \boldsymbol{w}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}} \rightarrow \boxed{\boldsymbol{a}^{[1]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[1]})} \xrightarrow{\boldsymbol{W}^{[2]},\ \boldsymbol{b}^{[2]}} \boxed{\boldsymbol{z}^{[2]} = \boldsymbol{w}^{[2]}\boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}} \rightarrow \boxed{\boldsymbol{a}^{[2]} = \boldsymbol{\sigma}(\boldsymbol{z}^{[2]})} \rightarrow \boxed{\mathcal{L}(\boldsymbol{a}^{[2]}, \boldsymbol{y})}$$

The loss function can be computed by moving from left to right

Backward Propagation

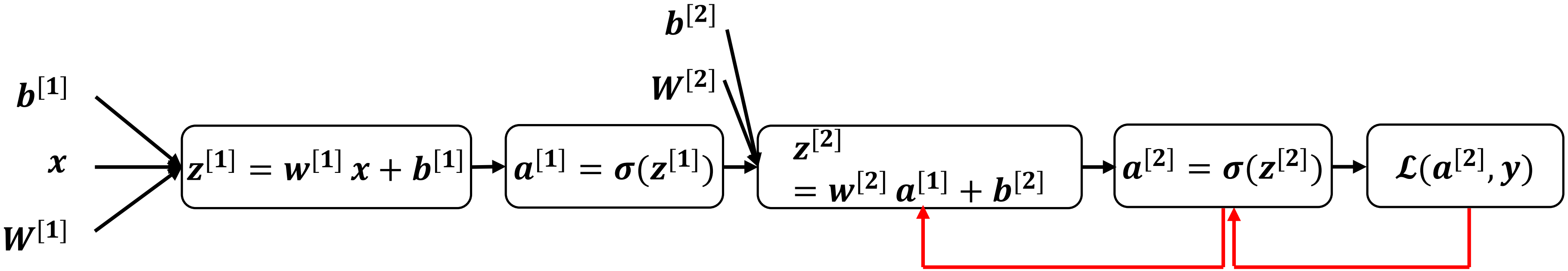
The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial a^{[2]}} = \frac{-y}{a^{[2]}} + \frac{(1-y)}{(1-a^{[2]})}$$

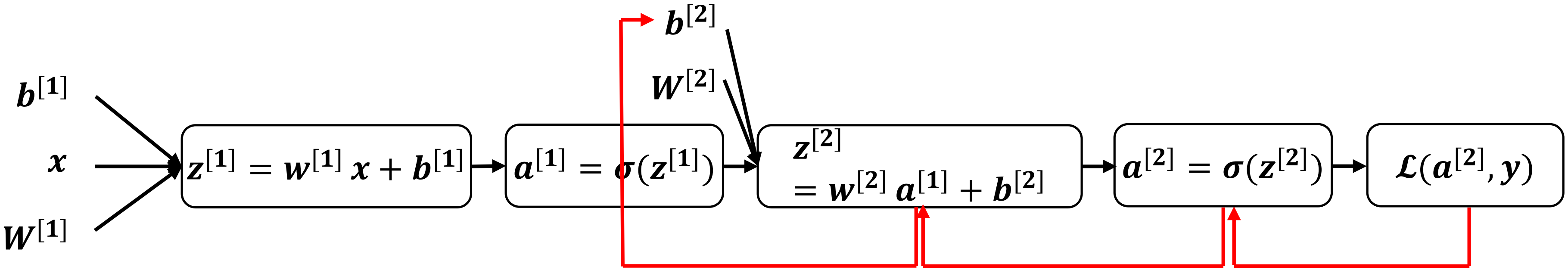
Backward Propagation

- The derivatives can be computed by moving from right to left



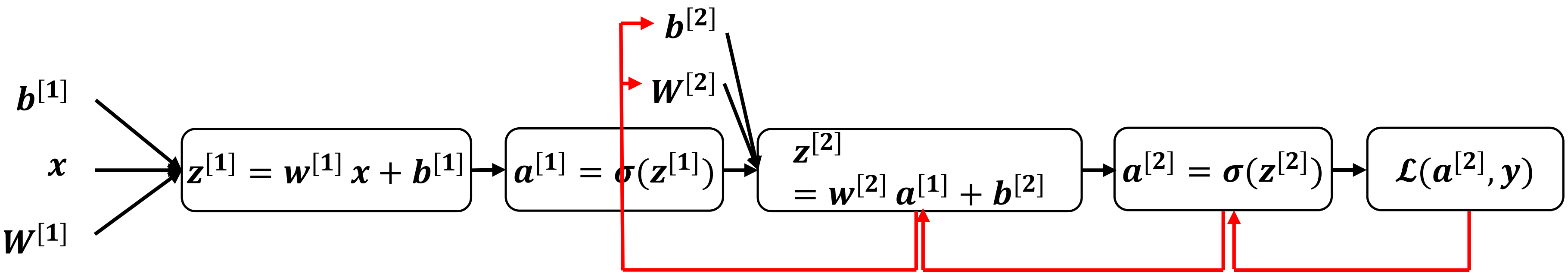
$$\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} = a^{[2]} - y$$

Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial b^{[2]}} = a^{[2]} - y$$

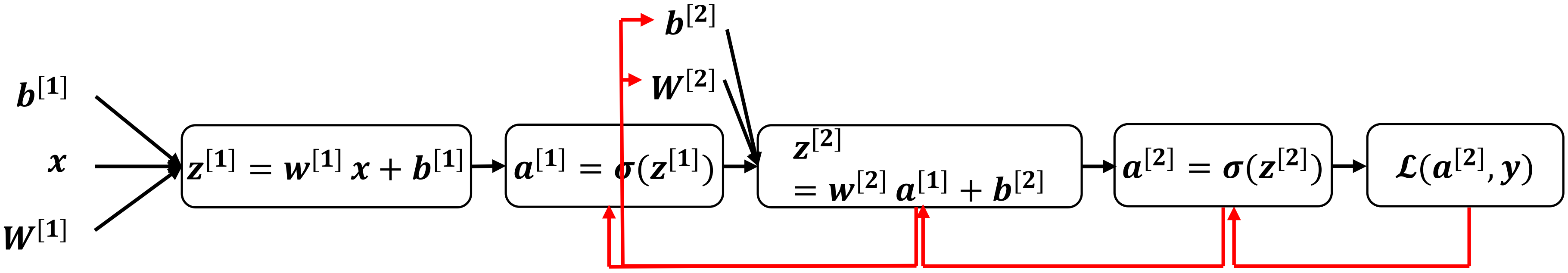
Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T}$$

Backward Propagation

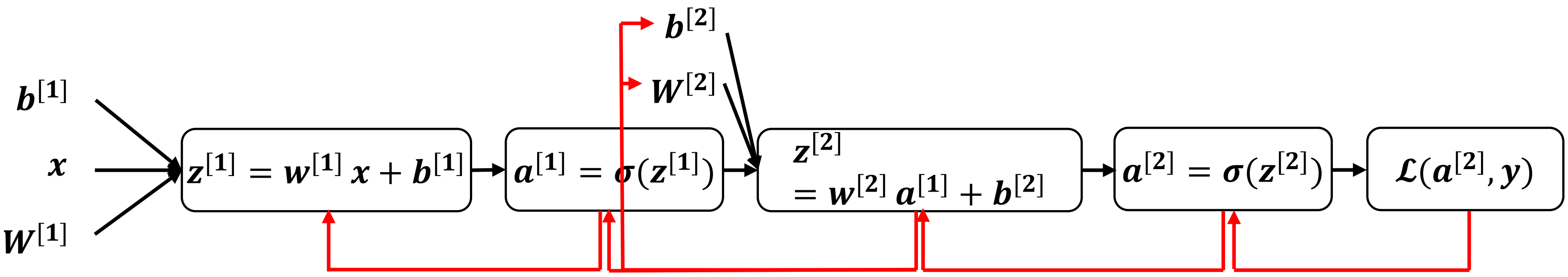
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial a^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} = (a^{[2]} - y)w^{[2]T}$$

Backward Propagation

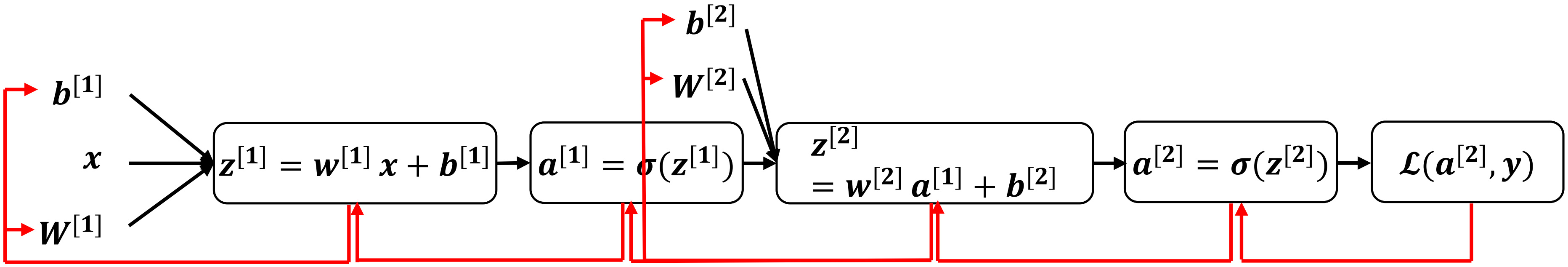
- The derivatives can be computed by moving from right to left



$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} = \left(a^{[2]} - y\right) w^{[2]T} * a^{[1]}\left(1 - a^{[1]}\right)$$

Element-wise product

Backward Propagation



$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \left((a^{[2]} - y) w^{[2]T} * a^{[1]} (1 - a^{[1]}) \right) x^T$$

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \times \frac{\partial a^{[2]}}{\partial z^{[2]}} \times \frac{\partial z^{[2]}}{\partial a^{[1]}} \times \frac{\partial a^{[1]}}{\partial z^{[1]}} \times \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = (a^{[2]} - y) w^{[2]T} * a^{[1]} (1 - a^{[1]})$$

Backward Propagation: Summary

$$dz^{[2]} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} = a^{[2]} - y$$

$$db^{[2]} = \frac{\partial \mathcal{L}}{\partial b^{[2]}} = a^{[2]} - y$$

$$dW^{[2]} = \frac{\partial \mathcal{L}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T}$$

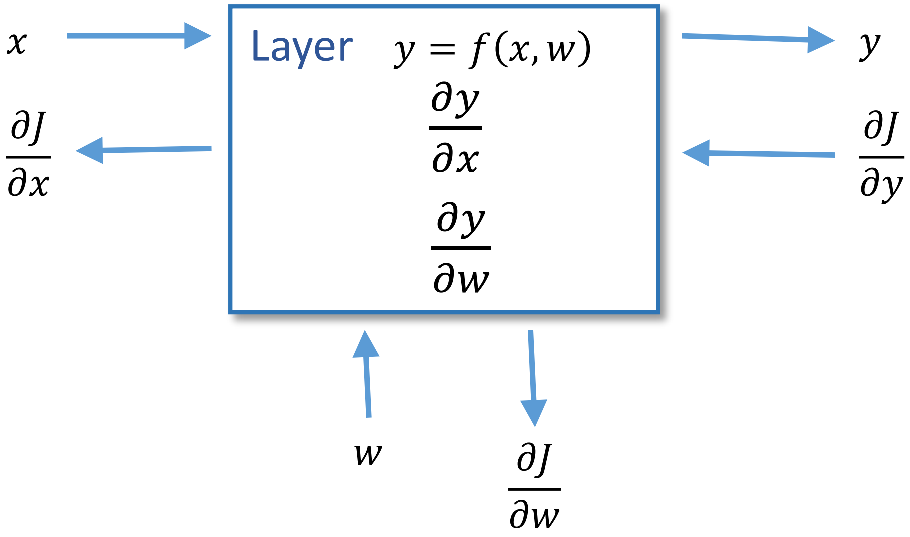
$$dz^{[1]} = \frac{\partial \mathcal{L}}{\partial z^{[1]}} = dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})$$

$$db^{[1]} = \frac{\partial \mathcal{L}}{\partial b^{[1]}} = dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})$$

$$dW^{[1]} = \frac{\partial \mathcal{L}}{\partial W^{[1]}} = \left(dz^{[2]}w^{[2]T} * a^{[1]}(1 - a^{[1]})\right) x^T$$

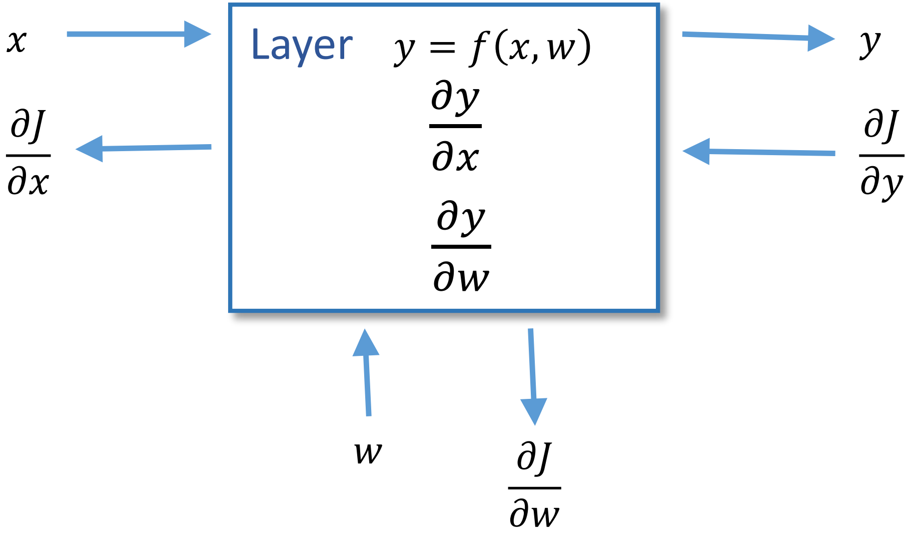
Backpropagation

- Compute derivatives per layer, utilizing previous derivatives
- Objective: $J(\boldsymbol{w})$
- Arbitrary layer: $y = f(x, w)$
- Need:
  - $\frac{\partial J}{\partial x} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x}$
  - $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w}$

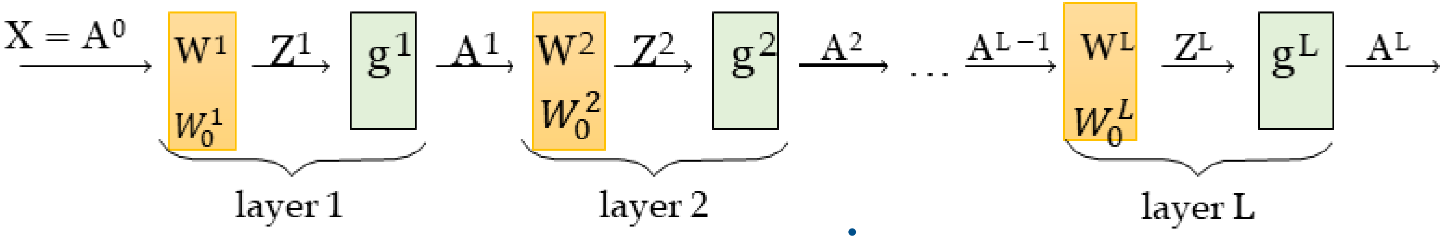


Backpropagation (layerwise)

- Compute derivatives per layer, utilizing previous derivatives
- Objective: $J(\boldsymbol{w})$
- Arbitrary layer: $y = f(x, w)$
- Init:
  - $\frac{\partial J}{\partial x} = 0$
  - $\frac{\partial J}{\partial w} = 0$
- Compute:
  - $\frac{\partial J}{\partial x} \mathrel{+}= \frac{\partial J}{\partial y}\frac{\partial y}{\partial x}$
  - $\frac{\partial J}{\partial w} \mathrel{+}= \frac{\partial J}{\partial y}\frac{\partial y}{\partial w}$



Error back-propagation

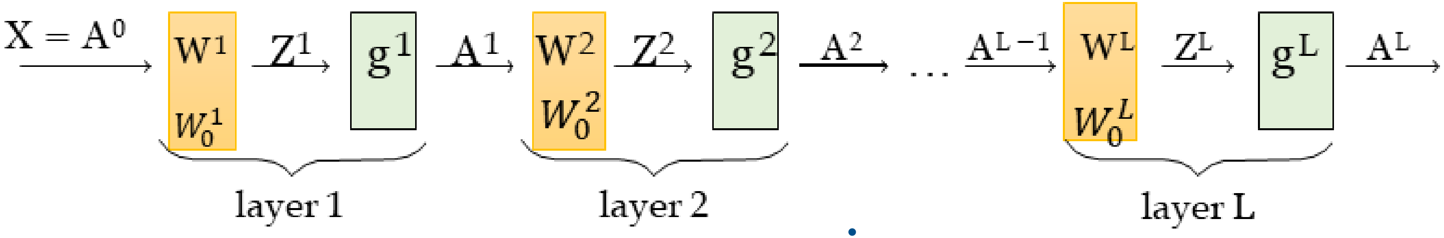


To do SGD for a training example (x, y), we need to compute

$$\nabla_W Loss(NN(x; W), y)$$

$$\frac{\partial L}{\partial W^L} = \frac{\partial L}{\partial A^L} \cdot \frac{\partial A^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial W^L}$$

Error back-propagation



$$\frac{\partial Loss}{\partial W^L} = \frac{\partial Loss}{\partial A^L} \cdot \frac{\partial A^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial W^L}$$

$$\underset{m^l \times n^l}{\frac{\partial L}{\partial W^l}} = \underset{m^l \times 1}{A^{l-1}} \underset{1 \times n^l}{\left(\frac{\partial L}{\partial Z^l}\right)^T}$$

So, in order to find the gradient of the loss with respect to the weights in the other layers of the network,

we just need to be able to find $\frac{\partial Loss}{\partial Z^l}$

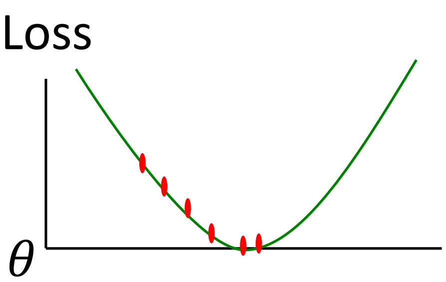
Gradient descent

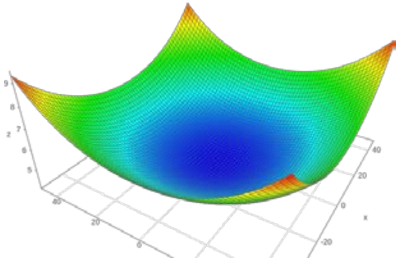
Partial derivatives give us the slope (i.e. direction to move) in that dimension

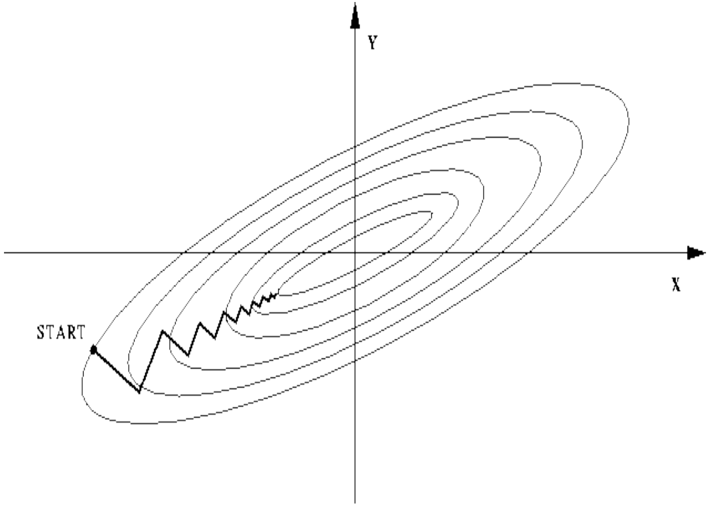
Approach:

- pick a starting point (θ)
- repeat:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

$$\theta_j = \theta_j - \eta \frac{d}{d\theta_j} \text{Loss}(\theta)$$





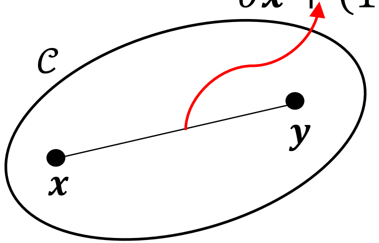


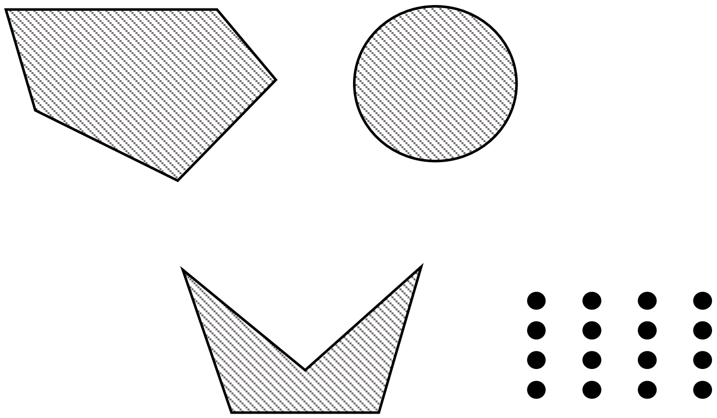
Aside - Convex Sets and Functions

- **Convex Set**: A set $\mathcal{C} \subseteq \mathbb{R}^n$ is a convex set if for all $x, y \in \mathcal{C}$, the line segment connecting x and y is in $\mathcal{C}$, i.e.,

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}, \theta \in [0,1], \theta\boldsymbol{x} + (1-\theta)\boldsymbol{y} \in \mathcal{C}$$

$\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}, \qquad \theta \in [0,1]$

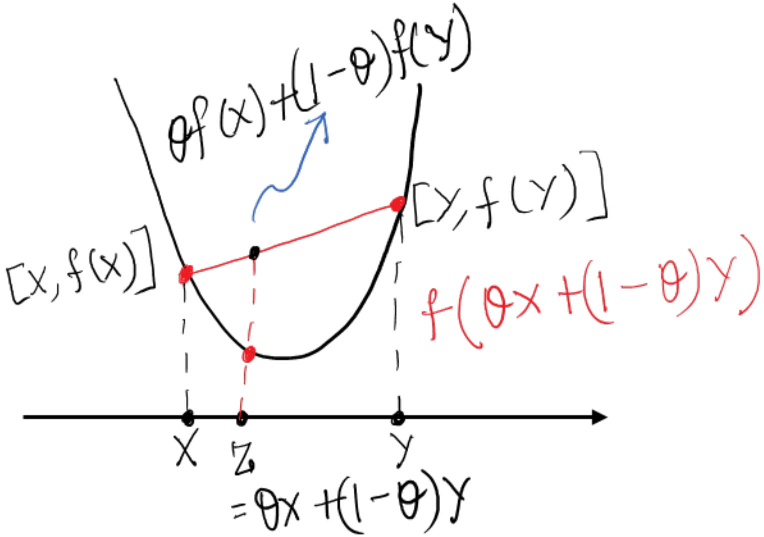




Convex Sets and Functions

- **Convex Function**: A function $f:\ \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex function if its domain $dom(f)$ is a convex set and for all $x, y \in dom(f)$, and $\theta \in [0,1]$, we have

$$f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y})$$



Batch, Stochastic and Minibatch

- Optimization algorithms that use the entire training set to compute the gradient are called batch or deterministic gradient methods. Ones that use a single training example for that task are called stochastic or online gradient methods
- Most of the algorithms we use for deep learning fall somewhere in between!
- These are called minibatch or minibatch stochastic methods

Batch, Stochastic and Mini-batch Stochastic Gradient Descent

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ
while stopping criteria not met **do**
  Compute gradient estimate over N examples:
  $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
  Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
end while

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k
Require: Initial Parameter θ
while stopping criteria not met **do**
  Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
  Compute gradient estimate:
  $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
  Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
end while

Mini-batch

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
end while

Images courtesy: Shubhendu Trivedi et. Al, Goodfellow et. al..

Batch and Stochastic Gradient Descent