# 10-301/601: Introduction to Machine Learning Lecture 13 – Backpropagation

Henry Chai & Zack Lipton

10/11/23

# Front Matter

- Announcements
  - HW3 released 10/4, due 10/11 (today!) at 11:59 PM
  - HW4 released 10/11 (today!), due 10/25 (**after fall break**) at 11:59 PM
  - Project details will be released on 10/13 (Friday)
  - Midterm exam on 10/31 from 6:30 – 8:30 PM
    - If you have a conflict with this date/time fill out the conflict on Piazza ASAP
- Recommended Readings
  - Mitchell, Chapters 4.1 – 4.6

# Recall: Stochastic Gradient Descent for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta^{(0)}$
- Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
- While TERMINATION CRITERION is not satisfied
  - For $i \in \text{shuffle}(\{1, \ldots, N\})$
    - For $l = 1, \ldots, L$
      - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)} \left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)$
      - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
    - Increment $t$: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$

# Matrix Calculus

*Numerator*

| *Types of Derivatives* | **scalar** | **vector** | **matrix** |
|---|---|---|---|
| **scalar** | $\frac{\partial y}{\partial x}$ | $\frac{\partial \mathbf{y}}{\partial x}$ | $\frac{\partial \mathbf{Y}}{\partial x}$ |
| **vector** | $\frac{\partial y}{\partial \mathbf{x}}$ | $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ | $\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$ |
| **matrix** | $\frac{\partial y}{\partial \mathbf{X}}$ | $\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$ | $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ |

*Denominator*

Table courtesy of Matt Gormley

# Matrix Calculus: Denominator Layout

- Derivatives of a scalar always have the *same shape* as the entity that the derivative is being taken with respect to.

| *Types of Derivatives* | **scalar** |
|---|---|
| **scalar** | $\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$ |
| **vector** | $\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$ |
| **matrix** | $\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \frac{\partial y}{\partial X_{12}} & \cdots & \frac{\partial y}{\partial X_{1Q}} \\ \frac{\partial y}{\partial X_{21}} & \frac{\partial y}{\partial X_{22}} & \cdots & \frac{\partial y}{\partial X_{2Q}} \\ \vdots & & & \vdots \\ \frac{\partial y}{\partial X_{P1}} & \frac{\partial y}{\partial X_{P2}} & \cdots & \frac{\partial y}{\partial X_{PQ}} \end{bmatrix}$ |

Table courtesy of Matt Gormley

# Matrix Calculus: Denominator Layout

| *Types of Derivatives* | **scalar** | **vector** |
|---|---|---|
| **scalar** | $\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$ | $\frac{\partial \mathbf{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x} \quad \frac{\partial y_2}{\partial x} \quad \cdots \quad \frac{\partial y_N}{\partial x}\right]$ |
| **vector** | $\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$ | $\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \cdots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$ |

Table courtesy of Matt Gormley

# Computing Gradients

$$\nabla_{W^{(l)}} \ell^{(i)} \left( W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)} \right) = \begin{bmatrix} \frac{\partial \ell^{(i)}}{\partial w_{1,0}^{(l)}} & \frac{\partial \ell^{(i)}}{\partial w_{1,1}^{(l)}} & \cdots & \frac{\partial \ell^{(i)}}{\partial w_{1,d^{(l-1)}}^{(l)}} \\ \frac{\partial \ell^{(i)}}{\partial w_{2,0}^{(l)}} & \frac{\partial \ell^{(i)}}{\partial w_{2,1}^{(l)}} & \cdots & \frac{\partial \ell^{(i)}}{\partial w_{2,d^{(l-1)}}^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \ell^{(i)}}{\partial w_{d^{(l)},0}^{(l)}} & \frac{\partial \ell^{(i)}}{\partial w_{d^{(l)},1}^{(l)}} & \cdots & \frac{\partial \ell^{(i)}}{\partial w_{d^{(l)},d^{(l-1)}}^{(l)}} \end{bmatrix}$$

# Computing Gradients: Intuition

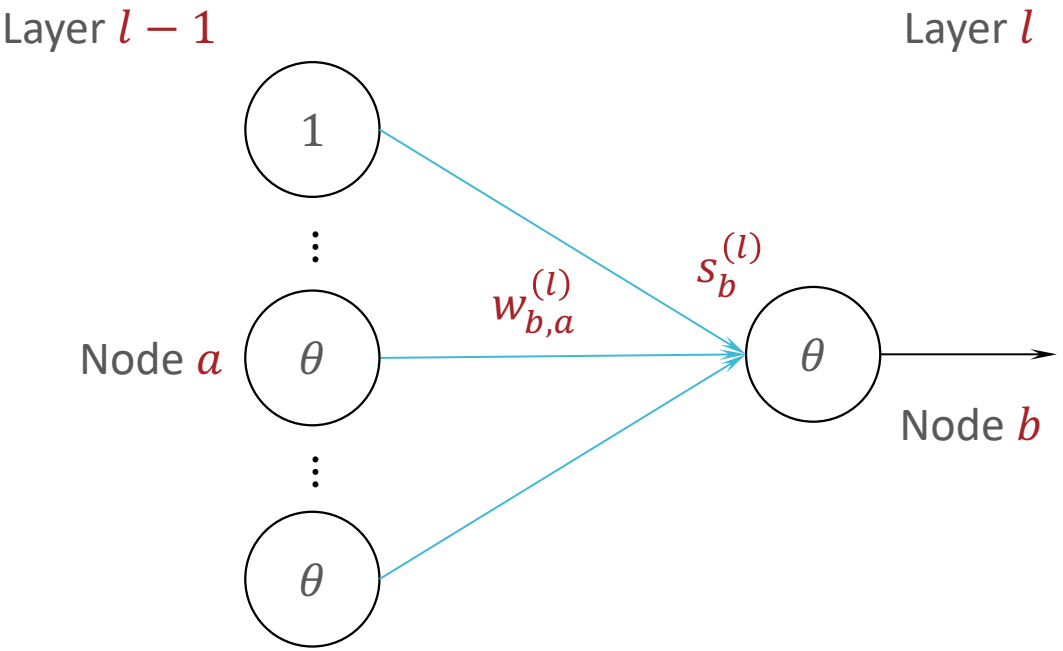
- A weight affects the prediction of the network (and therefore the error) through downstream signals/outputs
  - Use the chain rule!
- Any weight going into the same node will affect the prediction through the same downstream path
  - Compute derivatives starting from the last layer and move “backwards”
  - Derive a recursive definition for the relevant partial derivatives
  - Automatic differentiation: store intermediate values and reuse for efficiency (dynamic programming)

# Computing Partial Derivatives

Computing $\nabla_{W^{(l)}} \ell^{(i)} \left( W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)} \right)$ reduces to computing

$$\frac{\partial \ell^{(i)}}{\partial w_{b,a}^{(l)}}$$

Insight: $w_{b,a}^{(l)}$ *only* affects $\ell^{(i)}$ via $s_b^{(l)}$



# Computing Partial Derivatives

Computing $\nabla_{W^{(l)}} \ell^{(i)} \left( W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)} \right)$ reduces to computing

$$\frac{\partial \ell^{(i)}}{\partial w_{b,a}^{(l)}}$$

Insight: $w_{b,a}^{(l)}$ *only* affects $\ell^{(i)}$ via $s_b^{(l)}$

Chain rule: $\frac{\partial \ell^{(i)}}{\partial w_{b,a}^{(l)}} = \frac{\partial \ell^{(i)}}{\partial s_b^{(l)}} \left( \frac{\partial s_b^{(l)}}{\partial w_{b,a}^{(l)}} \right)$

$$s_b^{(l)} = \sum_{a=0}^{d^{(l-1)}} w_{b,a}^{(l)} o_a^{(l-1)} \rightarrow \frac{\partial s_b^{(l)}}{\partial w_{b,a}^{(l)}} = o_a^{(l-1)}$$

Compute outputs $\boldsymbol{o}^{(l)}$ $\forall\, l \in \{0, \ldots, L\}$ by forward propagation

# Computing Partial Derivatives

Computing $\nabla_{W^{(l)}} \ell^{(i)} \left( W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)} \right)$ reduces to computing

$$\frac{\partial \ell^{(i)}}{\partial w_{b,a}^{(l)}}$$

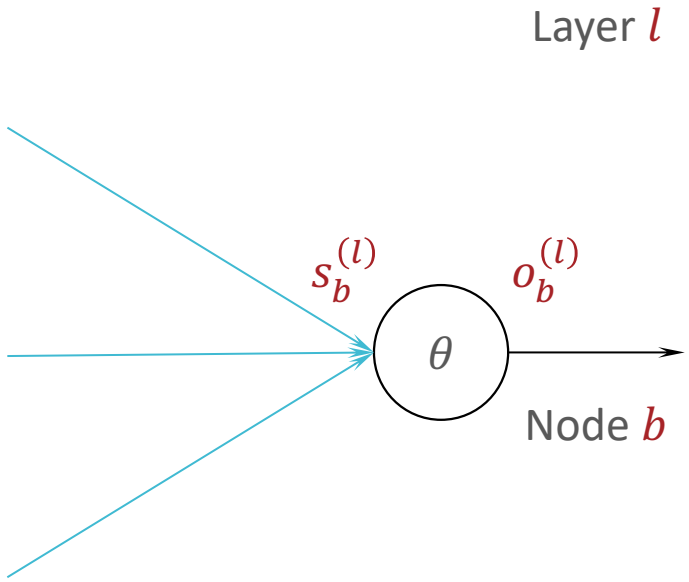
Insight: $w_{b,a}^{(l)}$ *only* affects $\ell^{(i)}$ via $s_b^{(l)}$

Chain rule: $$\frac{\partial \ell^{(i)}}{\partial w_{b,a}^{(l)}} = \frac{\partial \ell^{(i)}}{\partial s_b^{(l)}} \left( \frac{\partial s_b^{(l)}}{\partial w_{b,a}^{(l)}} \right)$$

$$\delta_b^{(l)} := \frac{\partial \ell^{(i)}}{\partial s_b^{(l)}}$$

# Computing Partial Derivatives

Insight: $s_b^{(l)}$ *only* affects $\ell^{(i)}$ via $o_b^{(l)}$



# Computing Partial Derivatives

Insight: $s_b^{(l)}$ *only* affects $\ell^{(i)}$ via $o_b^{(l)}$

Chain rule: $\delta_b^{(l)} = \frac{\partial \ell^{(i)}}{\partial o_b^{(l)}} \left( \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$

$$o_b^{(l)} = \theta\left(s_b^{(l)}\right) \rightarrow \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta\left(s_b^{(l)}\right)}{\partial s_b^{(l)}}$$

$$= 1 - \left(\tanh\left(s_b^{(l)}\right)\right)^2$$

when $\theta(\cdot) = \tanh(\cdot)$

# Aside: Vanishing Gradients

Insight: $s_b^{(l)}$ *only* affects $\ell^{(i)}$ via $o_b^{(l)}$

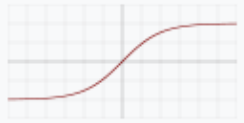
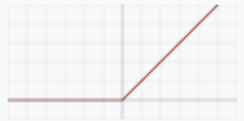
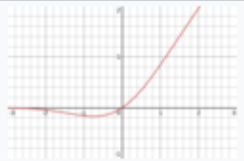
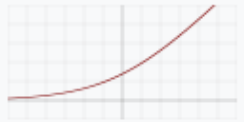
Chain rule: $\delta_b^{(l)} = \frac{\partial \ell^{(i)}}{\partial o_b^{(l)}} \left( \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$

$$o_b^{(l)} = \theta\left(s_b^{(l)}\right) \rightarrow \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} = \frac{\partial \theta\left(s_b^{(l)}\right)}{\partial s_b^{(l)}}$$

$$= 1 - \left(\tanh\left(s_b^{(l)}\right)\right)^2 \leq 1$$

when $\theta(\cdot) = \tanh(\cdot)$

# Recall: Other Activation Functions

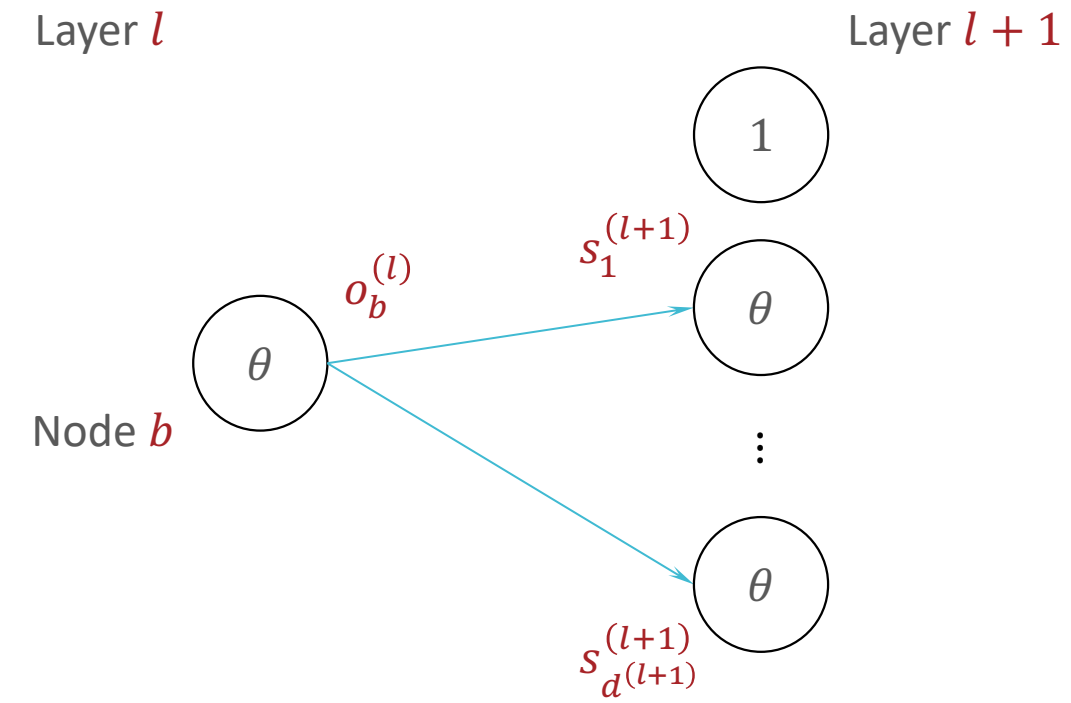
| | | |
|---|---|---|
| Logistic, sigmoid, or soft step |  | $\sigma(x) = \frac{1}{1+e^{-x}}$ |
| Hyperbolic tangent (tanh) |  | $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ |
| Rectified linear unit (ReLU)[7] |  | $\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$ |
| Gaussian Error Linear Unit (GELU)[4] |  | $\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$ |
| Softplus[8] |  | $\ln(1 + e^x)$ |
| Exponential linear unit (ELU)[9] |  | $\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter $\alpha$ |
| Leaky rectified linear unit (Leaky ReLU)[11] |  | $\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ |
| Parametric rectified linear unit (PReLU)[12] |  | $\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter $\alpha$ |



Source: https://en.wikipedia.org/wiki/Activation_function

# Computing Partial Derivatives

Insight: $o_b^{(l)}$ affects $\ell^{(i)}$ via $s_1^{(l+1)}, \ldots, s_{d^{(l+1)}}^{(l+1)}$



# Computing Partial Derivatives

Insight: $o_b^{(l)}$ affects $\ell^{(i)}$ via $s_1^{(l+1)}, \ldots, s_{d^{(l+1)}}^{(l+1)}$

Chain rule: $$\frac{\partial \ell^{(i)}}{\partial o_b^{(l)}} = \sum_{c=1}^{d^{(l+1)}} \frac{\partial \ell^{(i)}}{\partial s_c^{(l+1)}} \left( \frac{\partial s_c^{(l+1)}}{\partial o_b^{(l)}} \right)$$

$$s_c^{(l+1)} = \sum_{b=0}^{d^{(l)}} w_{c,b}^{(l+1)} o_b^{(l)} \rightarrow \frac{\partial s_c^{(l+1)}}{\partial o_b^{(l)}} = w_{c,b}^{(l+1)}$$

$$= \sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} \left( w_{c,b}^{(l+1)} \right)$$

# Computing Partial Derivatives

$$\delta_b^{(l)} = \frac{\partial \ell^{(i)}}{\partial o_b^{(l)}} \left( \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$$

$$= \left( \sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} \left( w_{c,b}^{(l+1)} \right) \right) \left( 1 - \left( o_b^{(l)} \right)^2 \right)$$

$$\boldsymbol{\delta}^{(l)} := \nabla_{\boldsymbol{s}^{(l)}} \ell^{(i)} \left( W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)} \right)$$

# Computing Partial Derivatives

$$\delta_b^{(l)} = \frac{\partial \ell^{(i)}}{\partial o_b^{(l)}} \left( \frac{\partial o_b^{(l)}}{\partial s_b^{(l)}} \right)$$

$$= \left( \sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} \left( w_{c,b}^{(l+1)} \right) \right) \left( 1 - \left( o_b^{(l)} \right)^2 \right)$$

$$\boldsymbol{\delta}^{(l)} = W^{(l+1)^T} \boldsymbol{\delta}^{(l+1)} \odot \left( 1 - \boldsymbol{o}^{(l)} \odot \boldsymbol{o}^{(l)} \right)$$

where $\odot$ is the element-wise product operation

Sanity check: $W^{(l+1)} \in \mathbb{R}^{d^{(l+1)} \times \left(d^{(l)}+1\right)}$ and

$\boldsymbol{\delta}^{(l+1)} \in \mathbb{R}^{d^{(l+1)} \times 1}$ so

$W^{(l+1)^T} \boldsymbol{\delta}^{(l+1)} \in \mathbb{R}^{\left(d^{(l)}+1\right) \times 1}$, the same size as $\boldsymbol{o}^{(l)}$!

# Computing Gradients

$$\frac{\partial \ell^{(i)}}{\partial w_{b,a}^{(l)}} = \delta_b^{(l)} \left( \frac{\partial s_b^{(l)}}{\partial w_{b,a}^{(l)}} \right) = \delta_b^{(l)} \left( o_a^{(l-1)} \right)$$

$$\nabla_{W^{(l)}} \ell^{(i)} \left( W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)} \right) = \boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T}$$

Sanity check: $\boldsymbol{o}^{(l-1)} \in \mathbb{R}^{\left(d^{(l-1)}+1\right) \times 1}$ and

$\boldsymbol{\delta}^{(l)} \in \mathbb{R}^{d^{(l)} \times 1}$ so

$\boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T} \in \mathbb{R}^{d^{(l)} \times \left(d^{(l-1)}+1\right)}$, the same size

# Computing Partial Derivatives

- Can recursively compute $\boldsymbol{\delta}^{(l)}$ using $\boldsymbol{\delta}^{(l+1)}$; need to compute the base case: $\boldsymbol{\delta}^{(L)}$
- Assume the output layer is a single node and the error function is the squared error: $\boldsymbol{\delta}^{(L)} = \delta_1^{(L)}$, $\boldsymbol{o}^{(L)} = o_1^{(L)}$ and $\ell^{(i)}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right) = \left(o_1^{(L)} - y^{(i)}\right)^2$

$$\delta_1^{(L)} = \frac{\partial \ell^{(i)}}{\partial s_1^{(L)}} = \frac{\partial}{\partial s_1^{(L)}}\left(o_1^{(L)} - y^{(i)}\right)^2$$

$$= 2\left(o_1^{(L)} - y^{(i)}\right)\frac{\partial o_1^{(L)}}{\partial s_1^{(L)}} = 2\left(o_1^{(L)} - y^{(i)}\right)\left(1 - \left(o_1^{(L)}\right)^2\right)$$

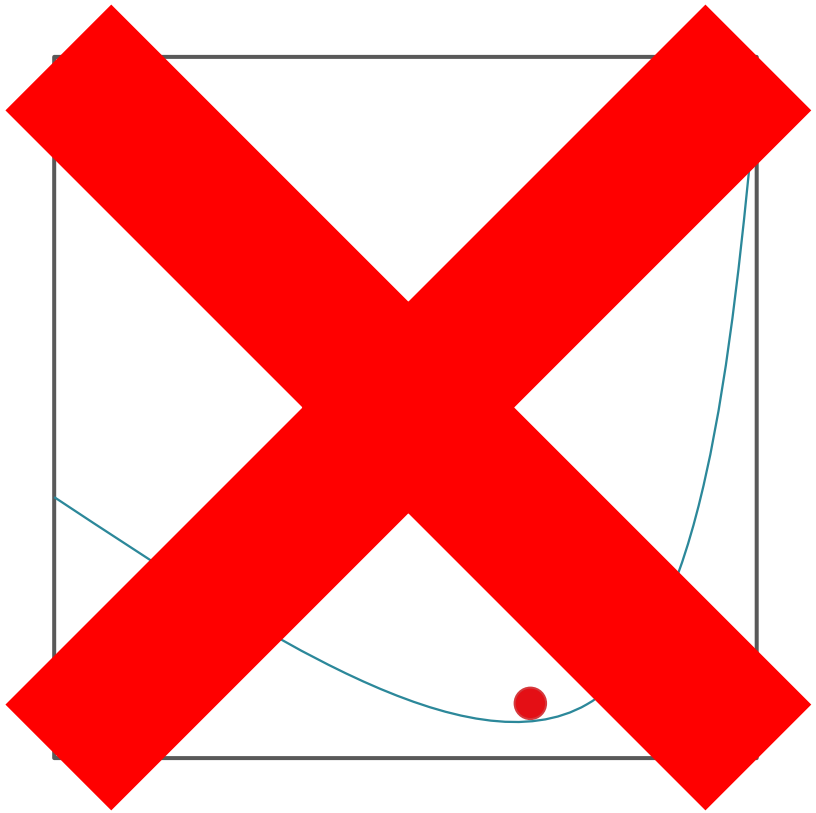
when $\theta(\cdot) = \tanh(\cdot)$

# Back-propagation

- Input: $W^{(1)}, \dots, W^{(L)}$ and $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$
- Run forward propagation with $\boldsymbol{x}^{(i)}$ to get $\boldsymbol{o}^{(1)}, \dots, \boldsymbol{o}^{(L)}$
- (Optional) Compute $\ell^{(i)} = \left(o^{(L)} - y^{(i)}\right)^2$
- Initialize: $\boldsymbol{\delta}^{(L)} = 2\left(o_1^{(L)} - y^{(i)}\right)\left(1 - \left(o_1^{(L)}\right)^2\right)$
- For $l = L-1, \dots, 1$
  - Compute $\boldsymbol{\delta}^{(l)} = {W^{(l+1)}}^T \boldsymbol{\delta}^{(l+1)} \odot \left(1 - \boldsymbol{o}^{(l)} \odot \boldsymbol{o}^{(l)}\right)$
  - Compute $G^{(l)} = \boldsymbol{\delta}^{(l)} {\boldsymbol{o}^{(l-1)}}^T$
- Output: $G^{(1)}, \dots, G^{(L)}$, the gradients of $\ell^{(i)}$ w.r.t $W^{(1)}, \dots, W^{(L)}$

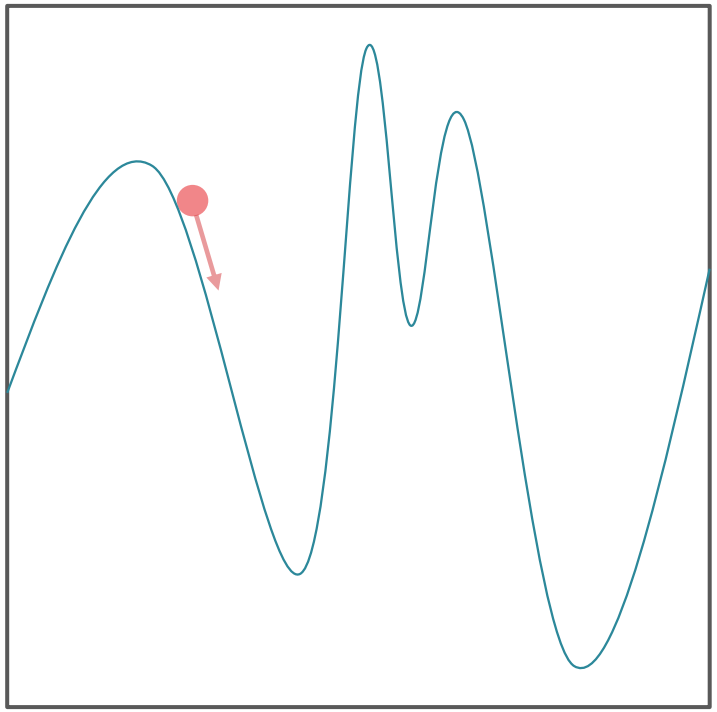
# Recall: Gradient Descent

- Iterative method for minimizing functions
- Requires the gradient to exist everywhere



# Non-convexity

- Gradient descent is not guaranteed to find a global minimum on non-convex surfaces



# Stochastic Gradient Descent for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta^{(0)}$
- Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
- While TERMINATION CRITERION is not satisfied
  - For $i \in \text{shuffle}(\{1, \ldots, N\})$
    - For $l = 1, \ldots, L$
      - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)} \left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)$
      - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
    - Increment $t$: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$

# Mini-batch Stochastic Gradient Descent for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$

1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample $B$ data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,

$$G^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right) \; \forall \, l$$

   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_{t}^{(l)} - \eta_{MB}^{(0)} G^{(l)} \; \forall \, l$
   d. Increment $t$: $t \leftarrow t + 1$

- Output: $W_{t}^{(1)}, \ldots, W_{t}^{(L)}$

# Mini-batch Stochastic Gradient Descent with Momentum for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B, \beta$

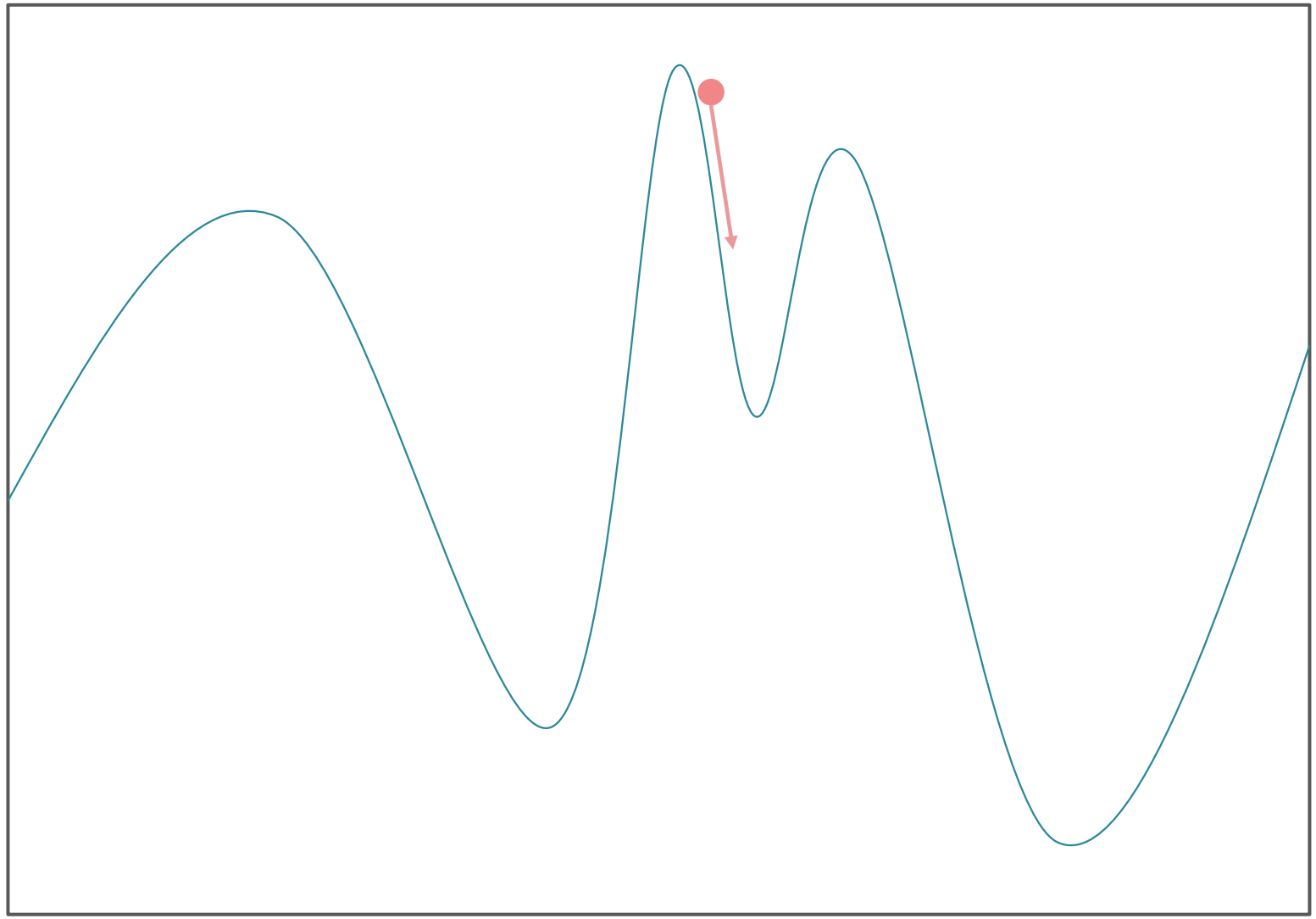
1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $G_{-1}^{(l)} = 0 \odot W^{(l)} \; \forall \, l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample $B$ data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,

$$G_t^{(l)} = \frac{1}{B} \sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)} \left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right) \forall \, l$$

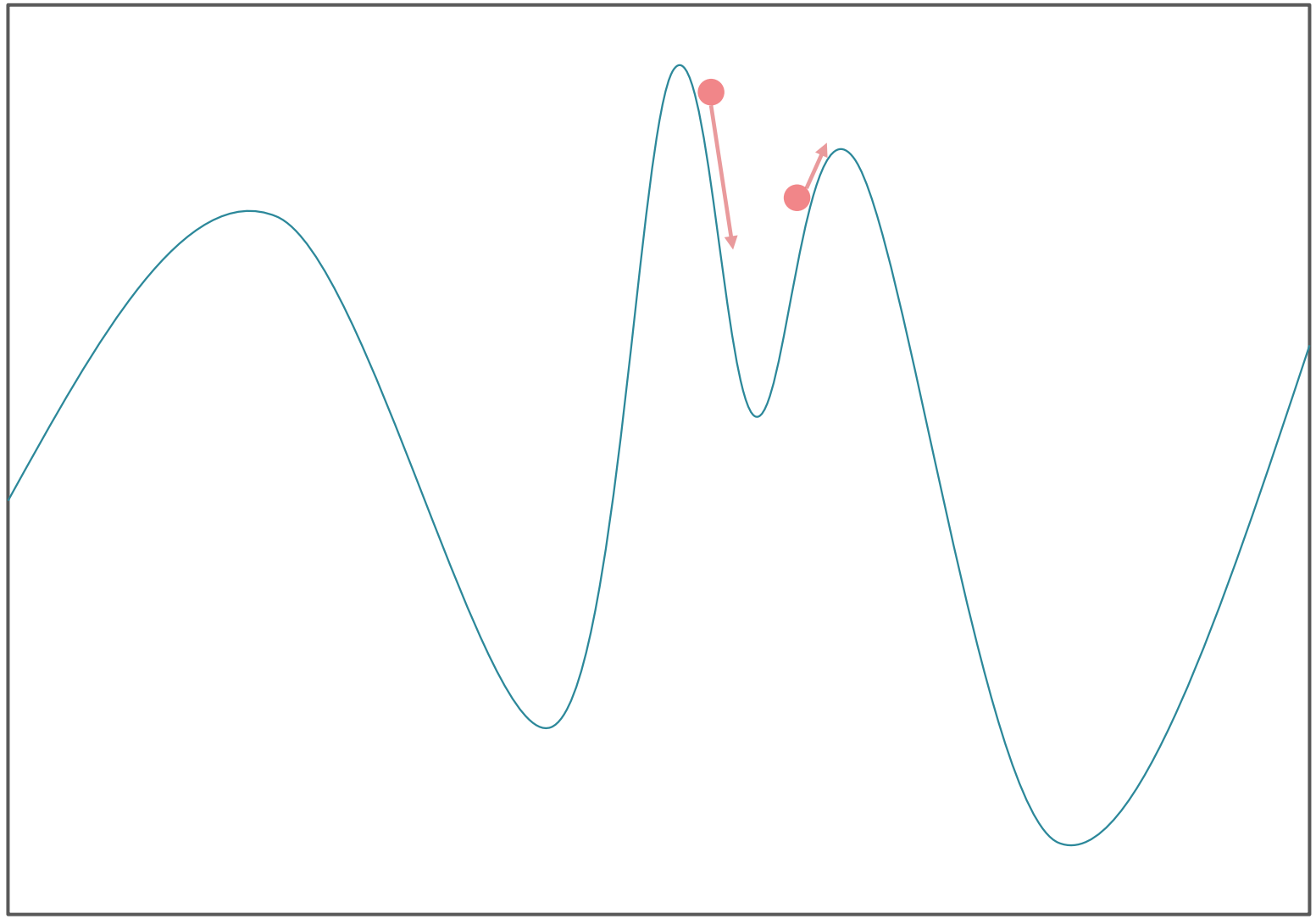
   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta_{MB}^{(0)} \left(\beta G_{t-1}^{(l)} + G_t^{(l)}\right) \forall \, l$
   d. Increment $t$: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \ldots, W_t^{(L)}$

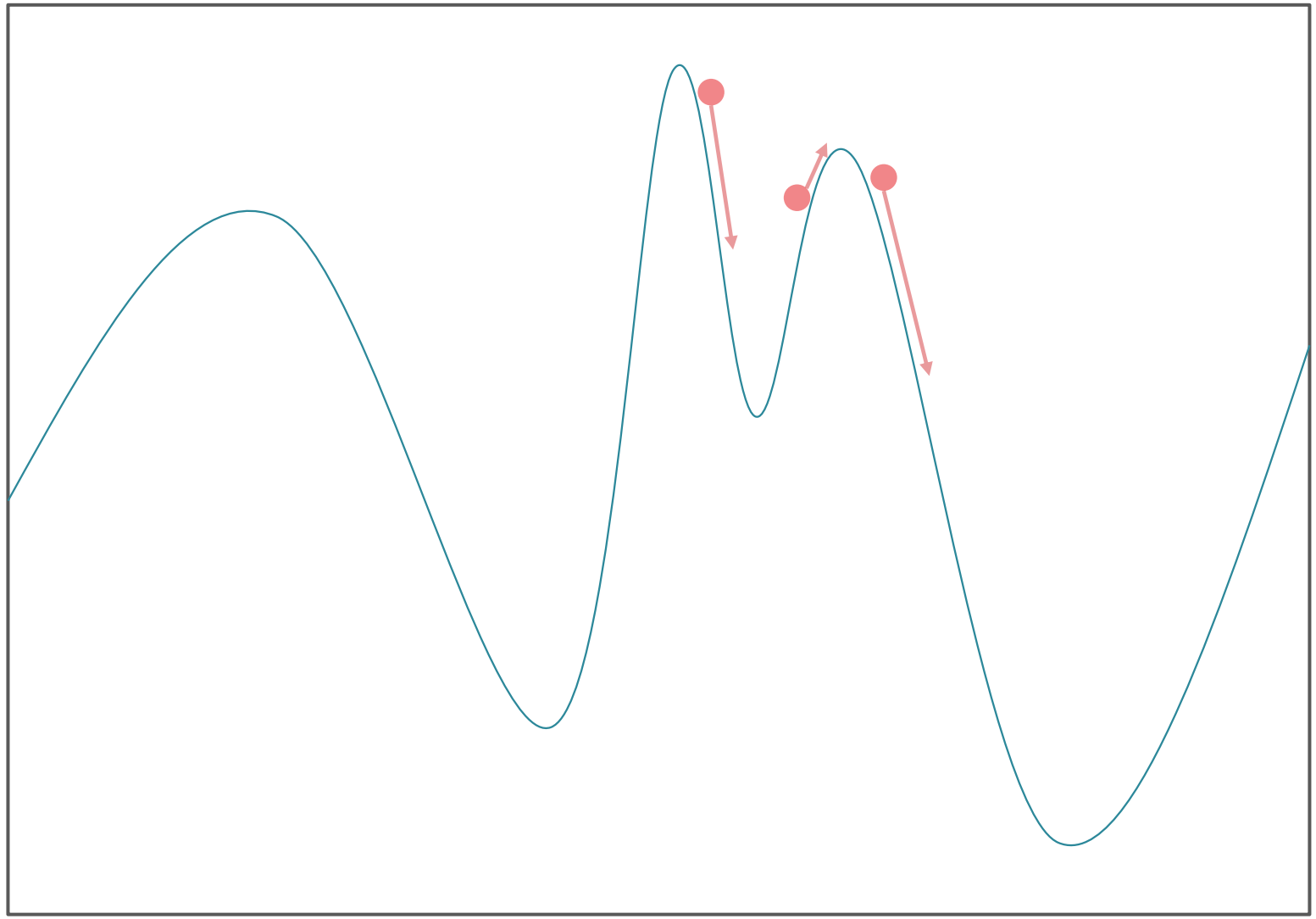
# Mini-batch Stochastic Gradient Descent with Momentum for Learning



# Mini-batch Stochastic Gradient Descent with Momentum for Learning



# Mini-batch Stochastic Gradient Descent with Momentum for Learning



# Mini-batch Stochastic Gradient Descent with Adaptive Gradients for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B, \epsilon$

1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $S_{-1}^{(l)} = 0 \odot W^{(l)} \; \forall \; l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample $B$ data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,

$$G_t^{(l)} = \frac{1}{B} \sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)} \left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right) \; \forall \; l$$

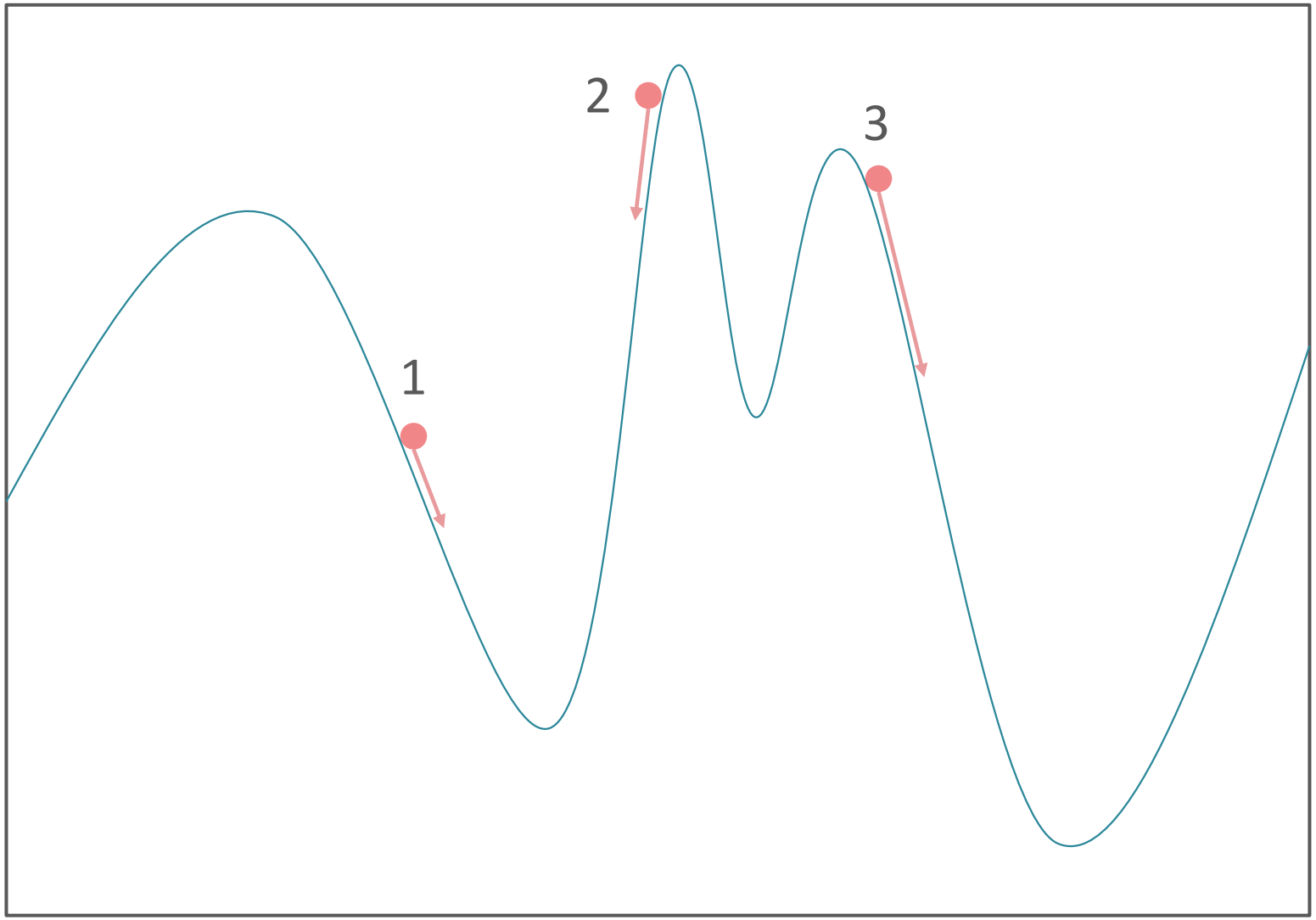
   c. Update $S^{(l)}$: $S_t^{(l)} = S_{t-1}^{(l)} + G_t^{(l)} \odot G_t^{(l)} \; \forall \; l$
   d. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \frac{\eta_{MB}^{(0)}}{\sqrt{S_t^{(l)} + \epsilon}} \odot G_t^{(l)} \; \forall \; l$
   e. Increment $t$: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \ldots, W_t^{(L)}$

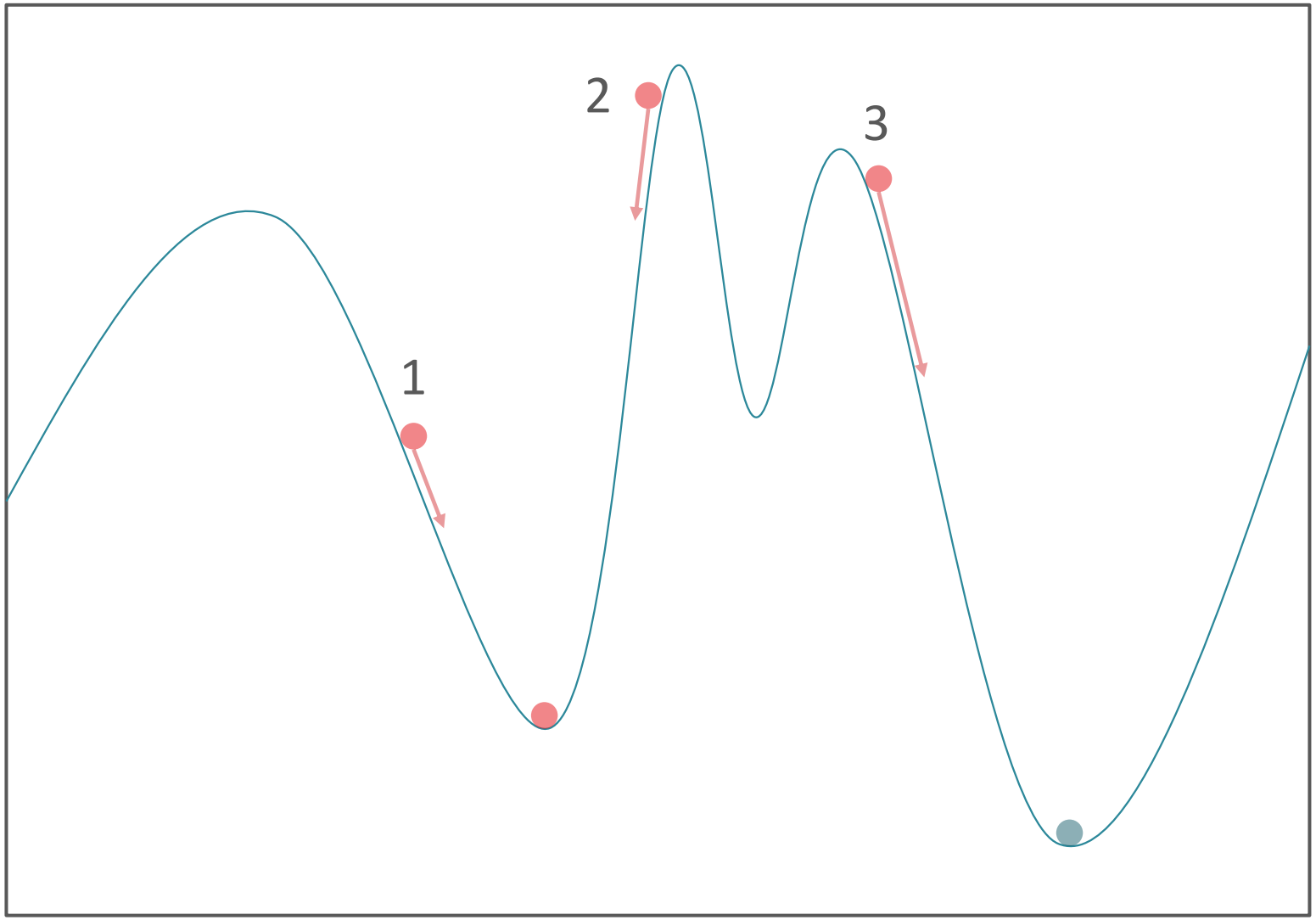
# Random Restarts

- Run mini-batch gradient descent (with momentum & adaptive gradients) multiple times, each time starting with a ***different***, ***random*** initialization for the weights.
- Compute the training error of each run at termination and return the set of weights that achieves the lowest training error.

# Random Restarts

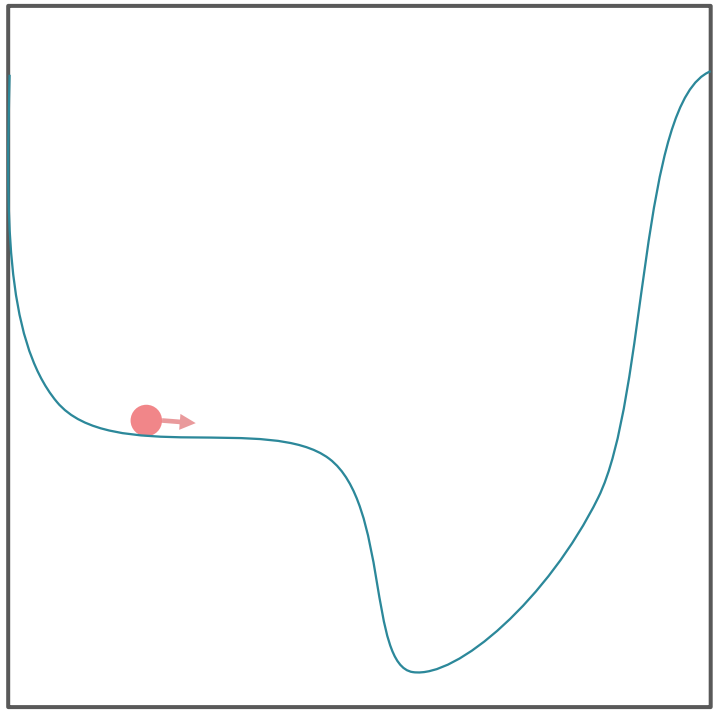


# Random Restarts



# Terminating Gradient Descent

- For non-convex surfaces, the gradient's magnitude is often not a good metric for proximity to a minimum



# Terminating Gradient Descent "Early"

- For non-convex surfaces, the gradient's magnitude is often not a good metric for proximity to a minimum
- Combine multiple termination criteria e.g. only stop if enough iterations have passed and the improvement in error is small
- Alternatively, terminate early by using a validation data set: if the validation error starts to increase, just stop!
  - Early stopping asks like regularization by **limiting how much of the hypothesis set** is explored

# Neural Networks and Regularization

- Minimize $\ell_{AUG}^{(i)}\left(W^{(1)}, \ldots, W^{(L)}, \lambda_C\right)$

$$= \ell^{(i)}\left(W^{(1)}, \ldots, W^{(L)}\right) + \lambda_C r\left(W^{(1)}, \ldots, W^{(L)}\right)$$

e.g. L2 regularization

$$r\left(W^{(1)}, \ldots, W^{(L)}\right) = \sum_{l=1}^{L} \sum_{i=0}^{d^{(l-1)}} \sum_{j=1}^{d^{(l)}} \left(w_{j,i}^{(l)}\right)^2$$

# Neural Networks and "Strange" Regularization (Bishop, 1995)

- Jitter
  - In each iteration of gradient descent, add some random noise or "jitter" to each training data point
    - Instead of computing the gradient w.r.t. $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$, use $\left(\boldsymbol{x}^{(i)} + \boldsymbol{\epsilon}, y^{(i)}\right)$ where

$$\boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2 I)$$

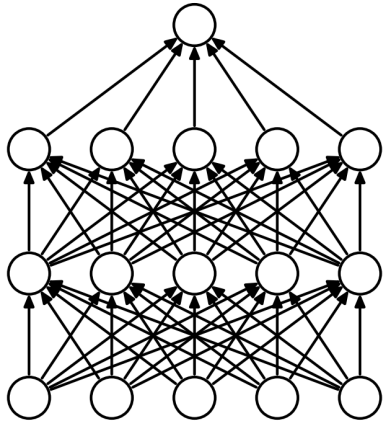
  - Makes neural networks resilient to input noise
  - Has been proven to be equivalent to using a certain kind of regularizer $r$ for some error metrics



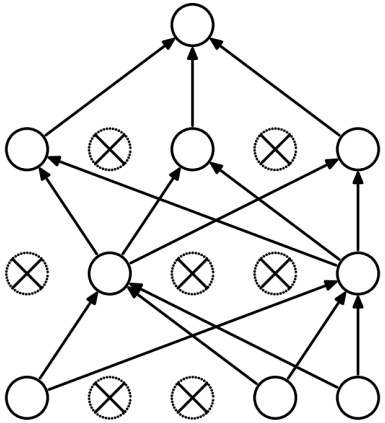
Source: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bishop-tikhonov-nc-95.pdf

# Neural Networks and "Strange" Regularization (Srivastava et al., 2014)

- Dropout
  - In each iteration of gradient descent, randomly remove some of the nodes in the network
  - Compute the gradient using only the remaining nodes
  - The weights on edges going into and out of "dropped out" nodes are not updated



(a) Standard Neural Net



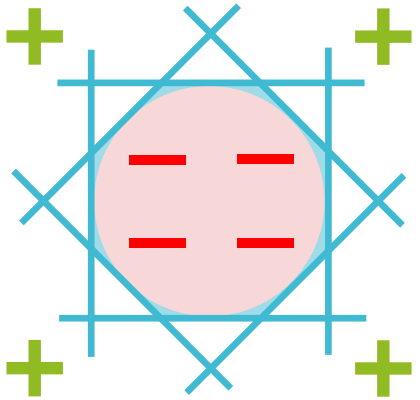
(b) After applying dropout.



Source: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

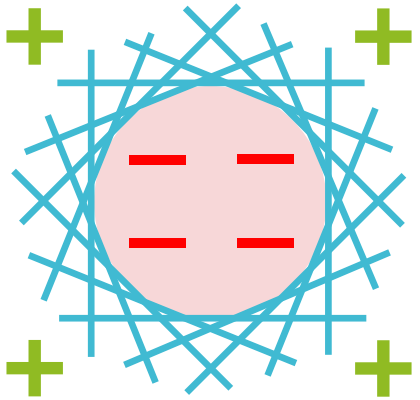
# MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



# MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



- Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 3-layer MLP

# NNs as Universal Approximators (Cybenko, 1989 & Hornik, 1991)

- Theorem: Any bounded, continuous function can be approximated to an arbitrary precision using a 2-layer (1 hidden layer) feed-forward NN if the activation function, $\theta$, is continuous, bounded and non-constant.

Source: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F21A09B7475DFB9487990020839A39D2?doi=10.1.1.441.7873&rep=rep1&type=pdf
https://doi.org/10.1016/0893-6080%2891%2990009-T

# NNs as Universal Approximators (Cybenko, 1988)

- Theorem: Any function can be approximated to an arbitrary precision using a 3-layer (2 hidden layers) feed-forward NN if the activation function, $\theta$, is continuous, bounded and non-constant.

Source: G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Dept. of Computer Science, Tufts University, Medford, MA, 1988.

# Deep Learning

- From Wikipedia's page on Deep Learning…

## Definition [ edit ]

Deep learning is a class of machine learning algorithms that[11](pp199–200) uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

- Deep learning = more than one layer



Source: https://en.wikipedia.org/wiki/Deep_learning

# Key Takeaways

- Backpropagation for efficient gradient computation
- Advanced optimization and regularization techniques for neural networks
  - Momentum can be used to break out of local minima
  - Adagrad helps when parameters behave differently w.r.t. step sizes
  - Random restarts
  - Jitter & dropout act like regularization for neural networks by preventing them fitting the training dataset perfectly
- MLPs and neural networks of sufficient depth are universal approximators