10-301/601: Introduction to Machine Learning Lecture 14 – Backpropagation

Henry Chai

5/22/25

Front Matter

- Announcements:
  - HW3 released on 5/20, due 5/23 (tomorrow) at 11:59 PM
  - Quiz 2 on 5/23 (tomorrow) at 11:00 AM in BH A36 (here)
    - Study guide solutions partially released 5/21 (yesterday)
    - The remaining solutions to be released after recitation on 5/22 (today!)
  - Midterm on 5/30 at 9:30 AM in BH A36
    - Lectures 1 – 14 are in-scope; **next week's lectures will not be tested on the midterm**

Midterm Logistics

- Time and place:
  - Friday, 5/30 from 9:30 AM to 12:00 PM in BH A36 (here)
- Closed book/notes
  - 1-page cheatsheet allowed, both back and front; can be typeset or handwritten
  - No electronic devices allowed, **including calculators**

Midterm Coverage

- Lectures: 1 – 14 (through this week's lectures)
  - Foundations: probability, linear algebra, calculus
  - Important concepts: inductive bias, overfitting, model selection/hyperparameter optimization, regularization
  - Models: decision trees, kNN, Perceptron, linear regression, logistic regression, neural networks
  - Methods: (stochastic) gradient descent, closed-form optimization, backpropagation, MLE/MAP

Midterm Preparation

- Review midterm practice problems, to be posted on 5/26 to the course website (under Schedule)
- Attend the exam review recitation on 5/29
- Review the homeworks and study guides
- Consider whether you understand the “Key Takeaways” for each lecture / section
- Write your cheat sheet

Recall: Loss Functions for Neural Networks

- Multi-class classification - cross-entropy loss
  - Express the label as a one-hot or one-of-C vector e.g.,

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$$

  - Assume the neural network output is also a vector of length C

$$P\left(y[k] = 1 \middle| \boldsymbol{x}, W^{(1)}, \dots, W^{(L)}\right) = h_{W^{(1)}, \dots, W^{(L)}}\left(\boldsymbol{x}^{(n)}\right)[k]$$

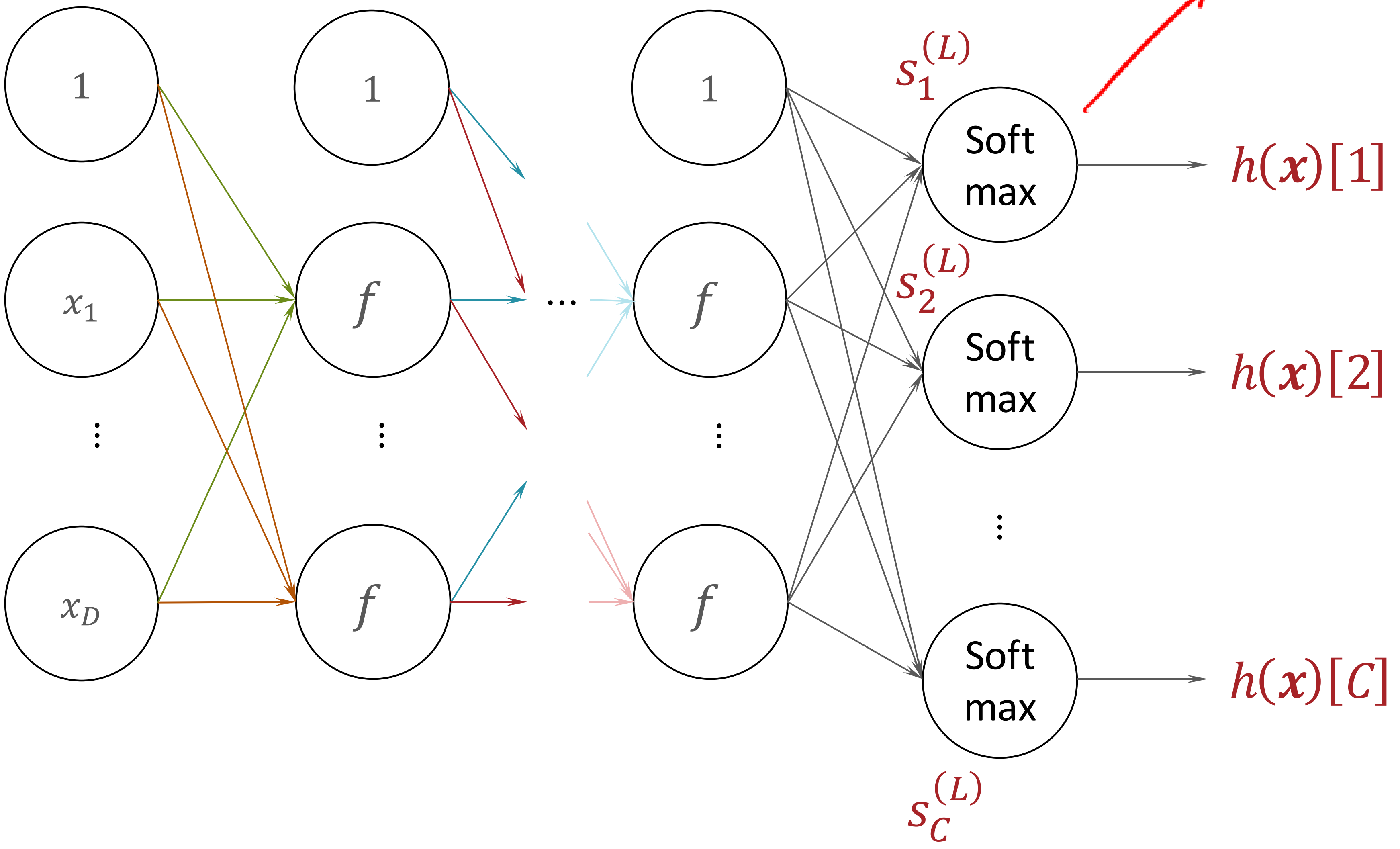
  - Then the cross-entropy loss is

$$\ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = -\sum_{n=1}^{N} \log P\left(y^{(n)} \middle| \boldsymbol{x}^{(n)}, W^{(1)}, \dots, W^{(L)}\right)$$

$$= -\sum_{n=1}^{N} \sum_{k=1}^{C} y[k] \log h_{W^{(1)}, \dots, W^{(L)}}\left(\boldsymbol{x}^{(n)}\right)[k]$$

Multi-dimensional Outputs

$$h(\vec{x})[k] = \frac{\exp(s_k^{(L)})}{\sum_{c=1}^{C} \exp(s_c^{(L)})}$$



Recall: 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 $l = 1, \ldots, L$
    - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)$ (???)
    - 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

Denominator

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	$\frac{\partial y}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

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

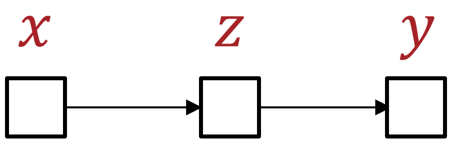
Types of Derivatives	**scalar**	**vector**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x} \right]$	$\frac{\partial \mathbf{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x} \quad \frac{\partial y_2}{\partial x} \quad \cdots \quad \frac{\partial y_N}{\partial x} \right]$
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

The Chain Rule of Calculus

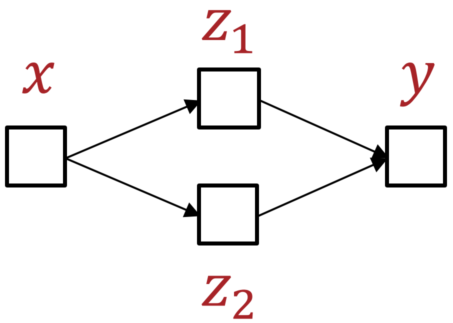
- If $y = f(z)$ and $z = g(x)$ then

computation graph is



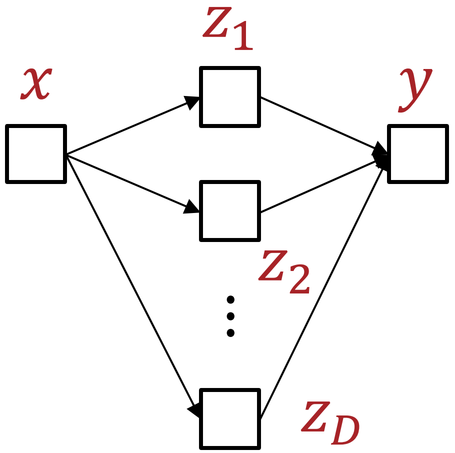
$$\Rightarrow \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x}$$

- If $y = f(z_1, z_2)$ and $z_1 = g_1(x), z_2 = g_2(x)$ then



$$\Rightarrow \frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x}$$

- If $y = f(\mathbf{z})$ and $\mathbf{z} = g(x)$ then



$$\Rightarrow \frac{\partial y}{\partial x} = \sum_{d=1}^{D} \frac{\partial y}{\partial z_d}\frac{\partial z_d}{\partial x}$$

Computing Gradients

$$\ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = \sum_{n=1}^{N} \ell^{(n)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$$

$$\nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$$

$$= \begin{bmatrix} \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,d^{(l-1)}}^{(l)}} \\ \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,d^{(l-1)}}^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},d^{(l-1)}}^{(l)}} \end{bmatrix}$$

$$\frac{\partial \ell_{\mathcal{D}}}{\partial w_{b,a}^{(l)}} = \sum_{n=1}^{N} \boxed{\frac{\partial \ell^{(n)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)}{\partial w_{b,a}^{(l)}}}$$

Computing Gradients: Intuition

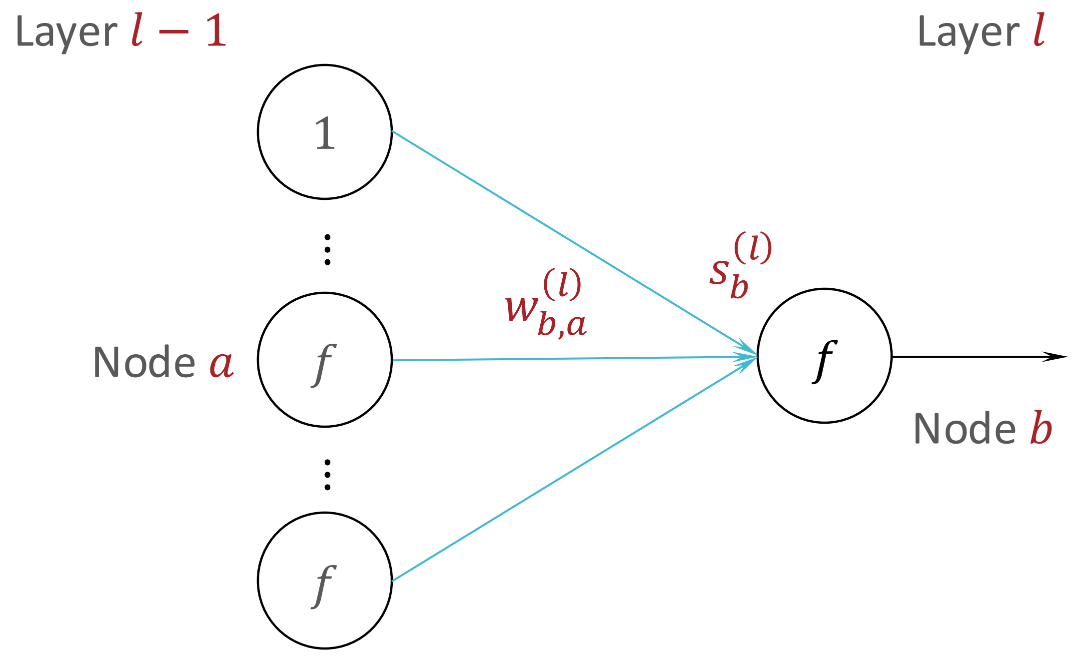
- A weight affects the prediction of the network (and therefore the error) through downstream signals/outputs
  - Use the chain rule!
- Any weight going into the same node will affect the prediction through the same downstream path
  - Compute derivatives starting from the last layer and move “backwards”
  - Store computed derivatives and reuse for efficiency (dynamic programming)

Computing Partial Derivatives

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

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

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



Computing Partial Derivatives

Computing $\nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$ reduces to computing

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

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

$$\frac{\partial \ell^{(n)}}{\partial w_{b,a}^{(\ell)}} = \frac{\partial \ell^{(n)}}{\partial s_b^{(\ell)}} \frac{\partial s_b^{(\ell)}}{\partial w_{b,a}^{(\ell)}}$$

$$\hookrightarrow s_b^{(\ell)} = \sum_{\alpha=0}^{d^{(\ell-1)}} o_\alpha^{(\ell-1)} w_{b,\alpha}^{(\ell)}$$

$$\Downarrow$$

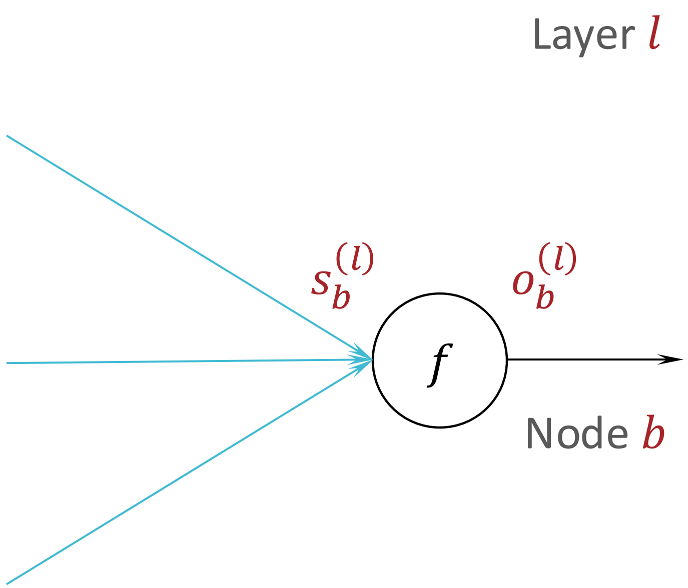
$$\frac{\partial s_b^{(\ell)}}{\partial w_{b,a}^{(\ell)}} = o_a^{(\ell-1)}$$

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

"sensitivity"

Computing Partial Derivatives

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



Computing Partial Derivatives

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

$$\delta_b^{(\ell)} := \frac{\partial \ell^{(n)}}{\partial s_b^{(\ell)}} = \frac{\partial \ell^{(n)}}{\partial o_b^{(\ell)}} \frac{\partial o_b^{(\ell)}}{\partial s_b^{(\ell)}}$$

harder...

$$o_b^{(\ell)} = f(s_b^{(\ell)})$$

$$\Rightarrow \frac{\partial o_b^{(\ell)}}{\partial s_b^{(\ell)}} = \frac{\partial f(s_b^{(\ell)})}{\partial s_b^{(\ell)}}$$

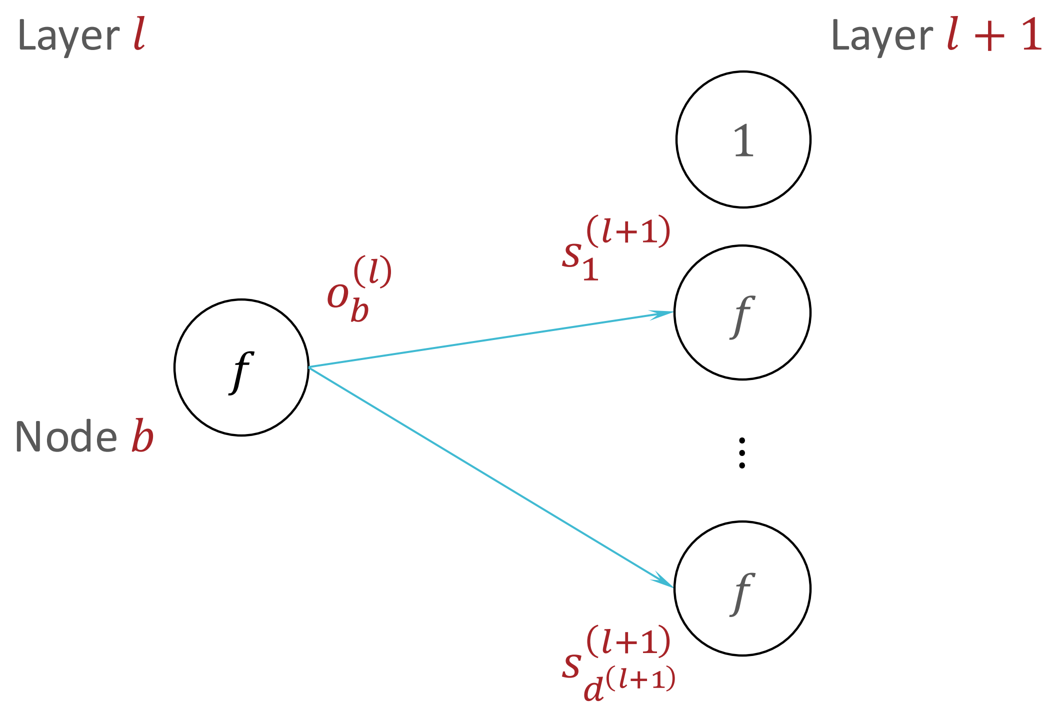
for example: $f(z) = \text{ReLU}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$

for example $f(z) = \tanh(z)$

$$\frac{df}{dz} = 1 - (\tanh(z))^2$$

Computing Partial Derivatives

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



Computing Partial Derivatives

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

$$\frac{\partial \ell^{(n)}}{\partial o_b^{(\ell)}} = \sum_{c=1}^{d^{(\ell+1)}} \frac{\partial \ell^{(n)}}{\partial s_c^{(\ell+1)}} \frac{\partial s_c^{(\ell+1)}}{\partial o_b^{(\ell)}}$$

$$\delta_c^{(\ell+1)}$$

$$s_c^{(\ell+1)} = \sum_{\beta=0}^{d^{(\ell)}} o_\beta^{(\ell)} w_{c,\beta}^{(\ell+1)}$$

$$\frac{\partial s_c^{(\ell+1)}}{\partial o_b^{(\ell)}} = w_{c,b}^{(\ell+1)}$$

$$\vec{s}^{(\ell)} = \begin{bmatrix} s_1^{(\ell)} \\ s_2^{(\ell)} \\ \vdots \\ s_{d^{(\ell)}}^{(\ell)} \end{bmatrix}$$

Computing Partial Derivatives

if $f(z) = \tanh(z)$

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

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

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

Based solely on their shape alone, which of the following could be an expression for $\delta^{(l)} = \nabla_{\mathbf{s}^{(l)}} e(\mathbf{o}^{(l)}, y^{(n)})$? Here $\odot$ is the element-wise product operation.

$W^{(l+1)^T} \delta^{(l+1)} \odot (1 - \mathbf{o}^{(l)})$

0%

$W^{(l+1)^T} \delta^{(l+1)} \odot (1 - \mathbf{o}^{(l)^T})$

0%

$\delta^{(l+1)^T} W^{(l+1)} \odot (1 - \mathbf{o}^{(l)})$

0%

$\delta^{(l+1)^T} W^{(l+1)} \odot (1 - \mathbf{o}^{(l)^T})$

0%

Computing Partial Derivatives

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

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

same size as

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

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

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

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

$\Rightarrow W^{(l+1)T} \delta^{(l+1)} \in \mathbb{R}^{(d^{(l)}+1) \times 1}$

$o^{(l)} \in \mathbb{R}^{(d^{(l)}+1) \times 1}$

Computing Gradients

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

$$\rightarrow \nabla_{W^{(l)}} \ell^{(n)} = \boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T}$$

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

$o^{(l-1)} \in \mathbb{R}^{(d^{(l-1)}+1) \times 1}$

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

$\Rightarrow \delta^{(l)} o^{(l-1)^T} \in \mathbb{R}^{d^{(l)} \times (d^{(l-1)}+1)}$

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 $\ell^{(n)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = \left(o_1^{(L)} - y^{(n)}\right)^2$

$$\delta_1^{(L)} = \frac{\partial e\left(o_1^{(L)}, y^{(n)}\right)}{\partial s_1^{(L)}} = \frac{\partial}{\partial s_1^{(L)}}\left(o_1^{(L)} - y^{(n)}\right)^2$$

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

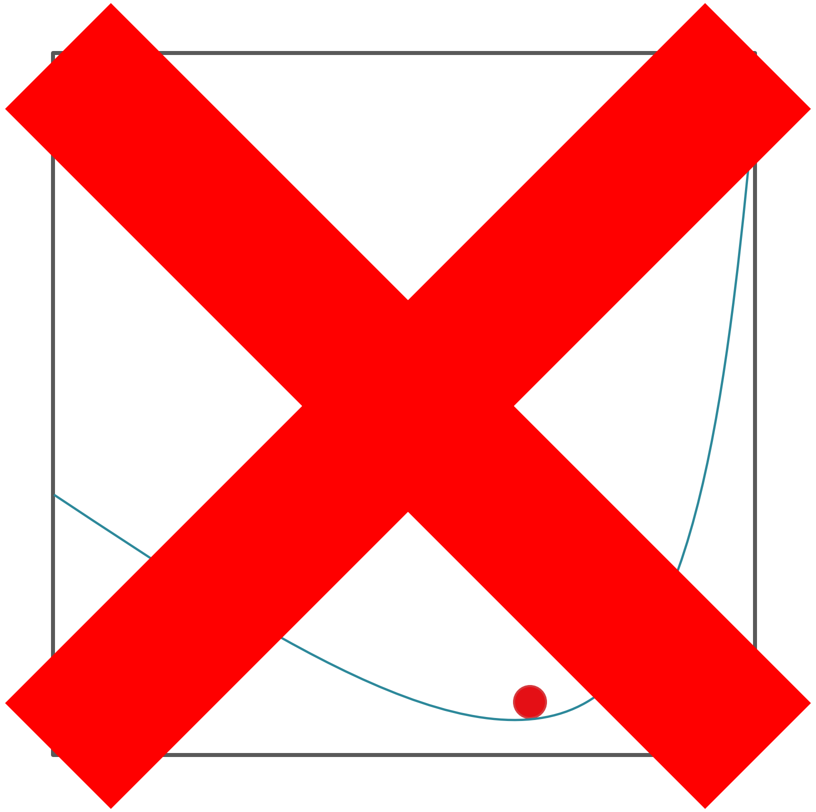
when $f(\cdot) = \tanh(\cdot)$

Back-propagation

- Input: $W^{(1)}, \dots, W^{(L)}$ and $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
- Initialize: $\ell_{\mathcal{D}} = 0$ and $G^{(l)} = 0 \odot W^{(l)}\ \forall\, l = 1, \dots, L$
- For $n = 1, \dots, N$
  - Run forward propagation with $\boldsymbol{x}^{(n)}$ to get $\boldsymbol{o}^{(1)}, \dots, \boldsymbol{o}^{(L)}$
  - (Optional) Increment $\ell_{\mathcal{D}}$: $\ell_{\mathcal{D}} = \ell_{\mathcal{D}} + \left(o^{(L)} - y^{(n)}\right)^2$
  - Initialize: $\boldsymbol{\delta}^{(L)} = 2\left(o_1^{(L)} - y^{(n)}\right)\left(1 - \left(o_1^{(L)}\right)^2\right)$
  - For $l = L - 1, \dots, 1$
    - Compute $\boldsymbol{\delta}^{(l)} = W^{(l+1)^T}\boldsymbol{\delta}^{(l+1)} \odot \left(1 - \boldsymbol{o}^{(l)} \odot \boldsymbol{o}^{(l)}\right)$
    - Increment $G^{(l)}$: $G^{(l)} = G^{(l)} + \boldsymbol{\delta}^{(l)}\boldsymbol{o}^{(l-1)^T}$
- Output: $G^{(1)}, \dots, G^{(L)}$, the gradients of $\ell_{\mathcal{D}}$ w.r.t $W^{(1)}, \dots, W^{(L)}$

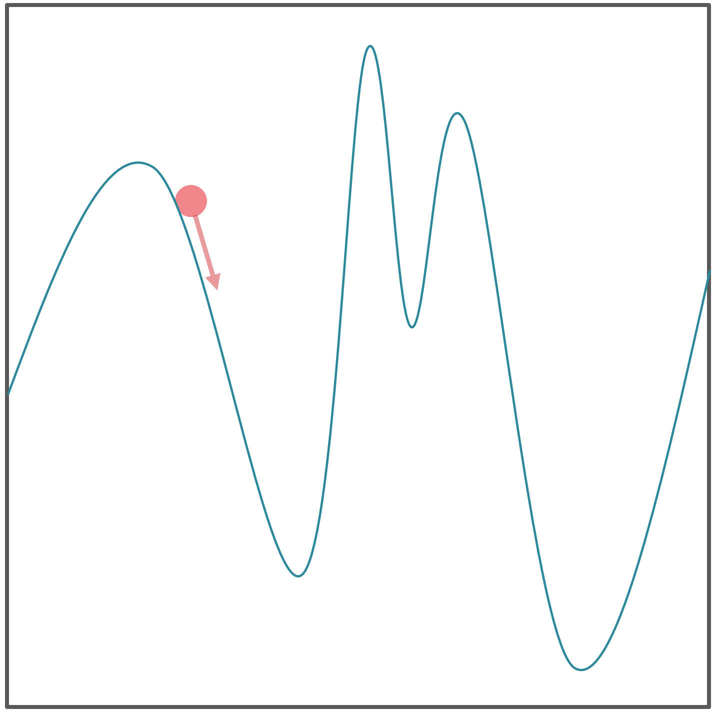
Recall: Gradient Descent

- 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 Neural Networks

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

1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample a data point from $\mathcal{D}$, $\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)$
   b. Compute the pointwise gradient using backpropagation

   $$G^{(l)} = \nabla_{W^{(l)}} \ell^{(n)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) \forall\, l$$

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

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

Mini-batch Stochastic Gradient Descent for Neural Networks

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

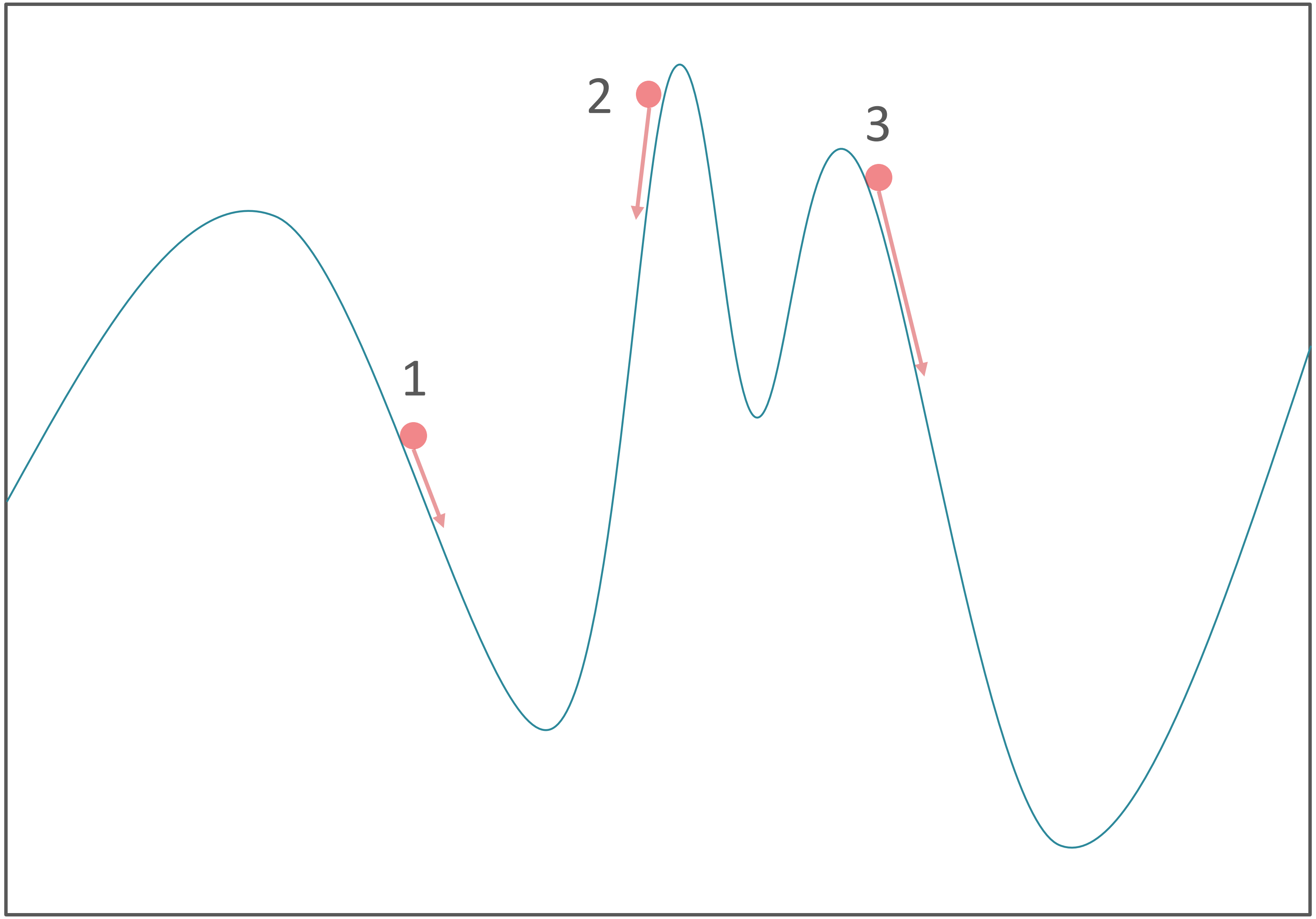
1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient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

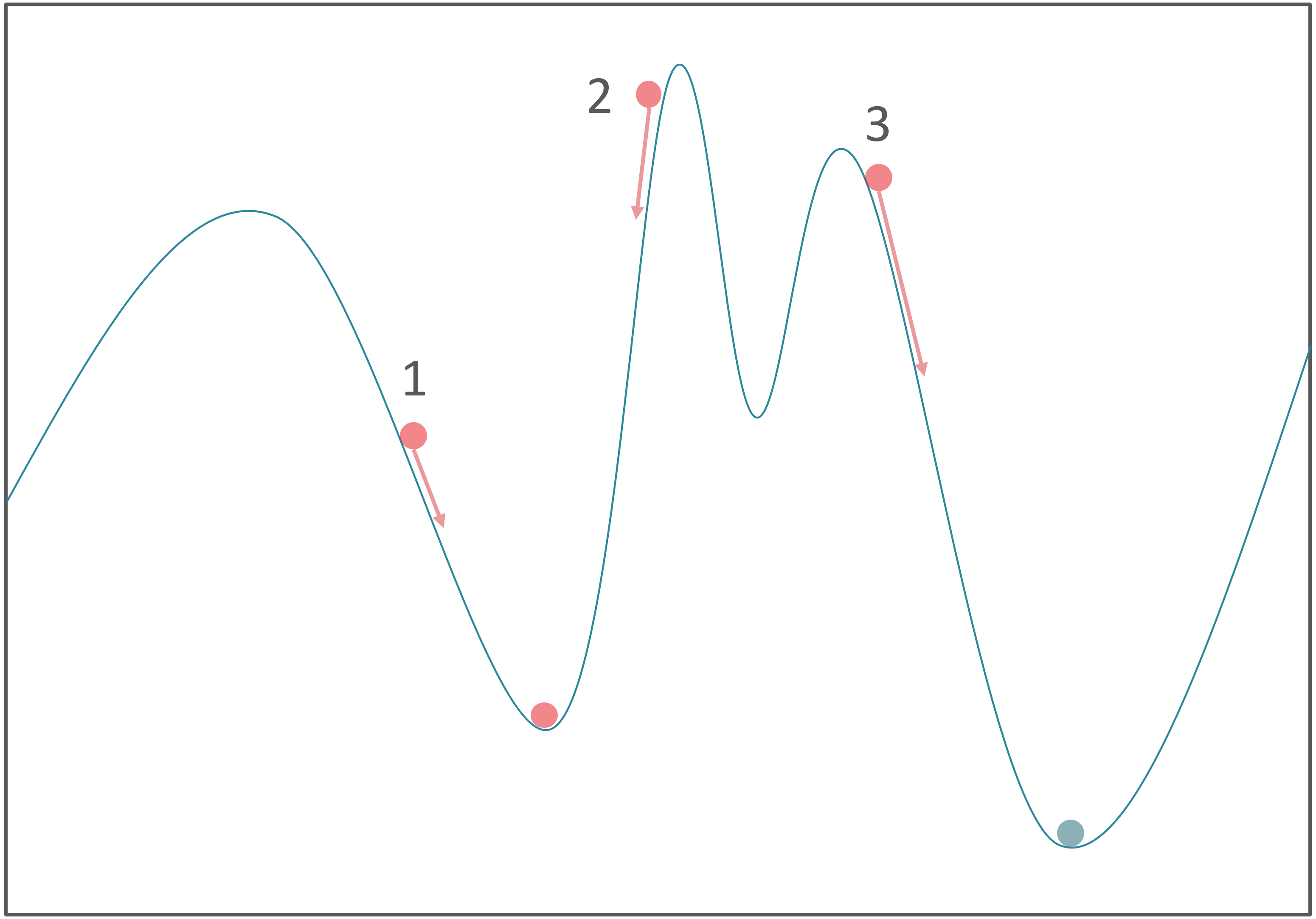
Random Restarts

- 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



Key Takeaways

- Backpropagation for efficient gradient computation
- Advanced optimization and regularization techniques for neural networks
    - SGD and Mini-batch gradient descent
    - Random restarts
    - Jitter & dropout act like regularization for neural networks by preventing them fitting the training dataset perfectly