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

Henry Chai

6/14/23

Front Matter

- Announcements
  - PA3 released 6/8, due 6/15 (tomorrow) at 11:59 PM
  - PA4 released 6/15 (tomorrow), due 7/13 (**4 weeks from tomorrow**) at 11:59 PM
    - We have scheduled this so that **you do not have to be working on PA4 during exam week or over break!**
  - Quiz 4: Neural Networks on 6/20 (next Tuesday)
  - No lecture **or OH** on 6/19 (next Monday) for Juneteenth
  - Midterm on 6/23, one week from Friday
    - Reminder: all of this week's material is in-scope
- Recommended Readings
  - Mitchell, Chapters 4.1 – 4.6

Computation Graph 10-301/601 Conventions

- The diagram represents *an algorithm*
- Nodes are rectangles with one node per intermediate variable in the algorithm
- Each node is labeled with the function that it computes (inside the box) and the variable name (outside the box)
- Edges are directed and do not have labels
- For neural networks:
  - Each weight, feature value, label and *bias term* appears as a node
  - We *can* include the loss function

Neural Network Diagram Conventions

- The diagram represents a *neural network*
- Nodes are circles with one node per hidden unit
- Each node is labeled with the variable corresponding to the hidden unit
- Edges are directed and each edge is labeled with its weight
- Following standard convention, the bias term is typically *not* shown as a node, but rather is assumed to be part of the activation function i.e., its weight does not appear in the picture anywhere.
- The diagram typically does *not* include any nodes related to the loss computation

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 $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$ (???)
- While TERMINATION CRITERION is not satisfied (???)
  - For $l = 1, \dots, L$
    - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$ (???)
    - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
  - Increment t: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}$

Computing Gradients

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

$W^{(\ell)} \in \mathbb{R}^{d^{(\ell)} \times (d^{(\ell-1)}+1)}$

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

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

$$\frac{\partial \ell_{\mathcal{D}}}{\partial w_{b,a}^{(l)}} = \sum_{n=1}^{N} \frac{\partial \ell_{(x^{(n)}, y^{(n)})}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)}{\partial w_{b,a}^{(l)}} := \sum_{n=1}^{N} \frac{\partial e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)}{\partial w_{b,a}^{(l)}}$$

going into node b → (b) — coming from node a (a)

Computing Gradients: Intuition

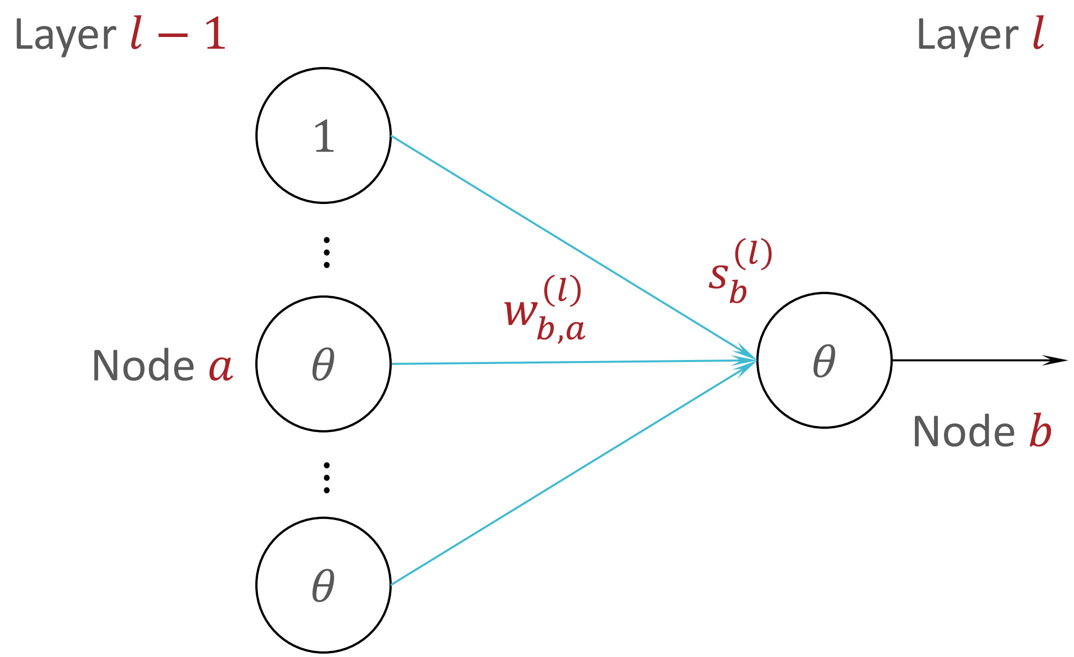
- A weight affects the prediction of the network (and therefore the error) through downstream signals/outputs
  - Use the chain rule!
- Any weight going into the same node will affect the prediction through the same downstream path
  - Compute derivatives starting from the last layer and move “backwards”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

Computing Partial Derivatives

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

$$\frac{\partial e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)}{\partial w_{b,a}^{(l)}}$$

Insight: $w_{b,a}^{(l)}$ *only* affects $e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)$ 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

$$\frac{\partial e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)}{\partial w_{b,a}^{(l)}}$$

Insight: $w_{b,a}^{(l)}$ *only* affects $e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)$ via $s_b^{(l)}$

$$\frac{\partial e\left(o^{(L)}, y^{(n)}\right)}{\partial w_{b,a}^{(\ell)}} = \frac{\partial e\left(o^{(L)}, y^{(n)}\right)}{\partial s_b^{(\ell)}} \frac{\partial s_b^{(\ell)}}{\partial w_{b,a}^{(\ell)}}$$

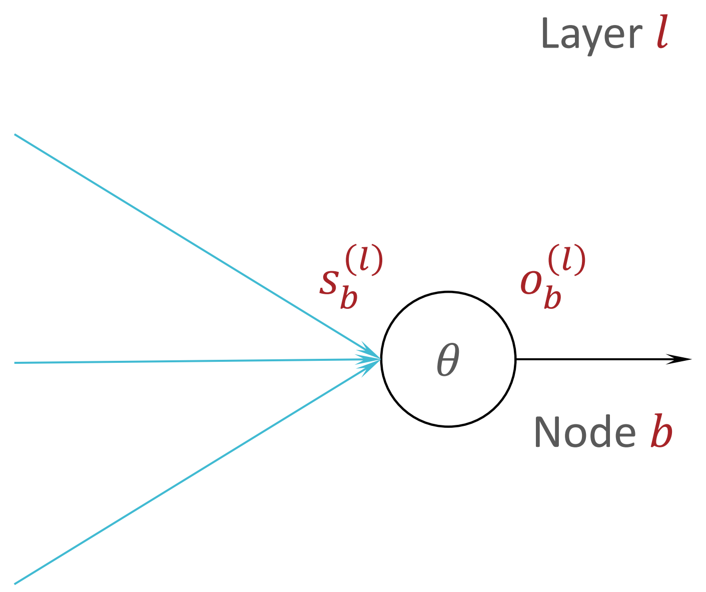
$$s_b^{(\ell)} = \sum_{A=0}^{d^{(\ell-1)}} w_{b,A}^{(\ell)} o_A^{(\ell-1)}$$

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

$$\delta_b^{(\ell)} := \frac{\partial e\left(o^{(L)}, y^{(n)}\right)}{\partial s_b^{(\ell)}}$$

Computing Partial Derivatives

Insight: $s_b^{(l)}$ *only* affects $e(\boldsymbol{o}^{(L)}, y^{(n)})$ via $o_b^{(l)}$



Computing Partial Derivatives

Insight: $s_b^{(l)}$ *only* affects $e(\boldsymbol{o}^{(L)}, y^{(n)})$ via $o_b^{(l)}$

$$\delta_b^{(\ell)} = \frac{\partial e(o^{(L)}, y^{(n)})}{\partial s_b^{(\ell)}} = \frac{\partial e(o^{(L)}, y^{(n)})}{\partial o_b^{(\ell)}} \frac{\partial o_b^{(\ell)}}{\partial s_b^{(\ell)}}$$

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

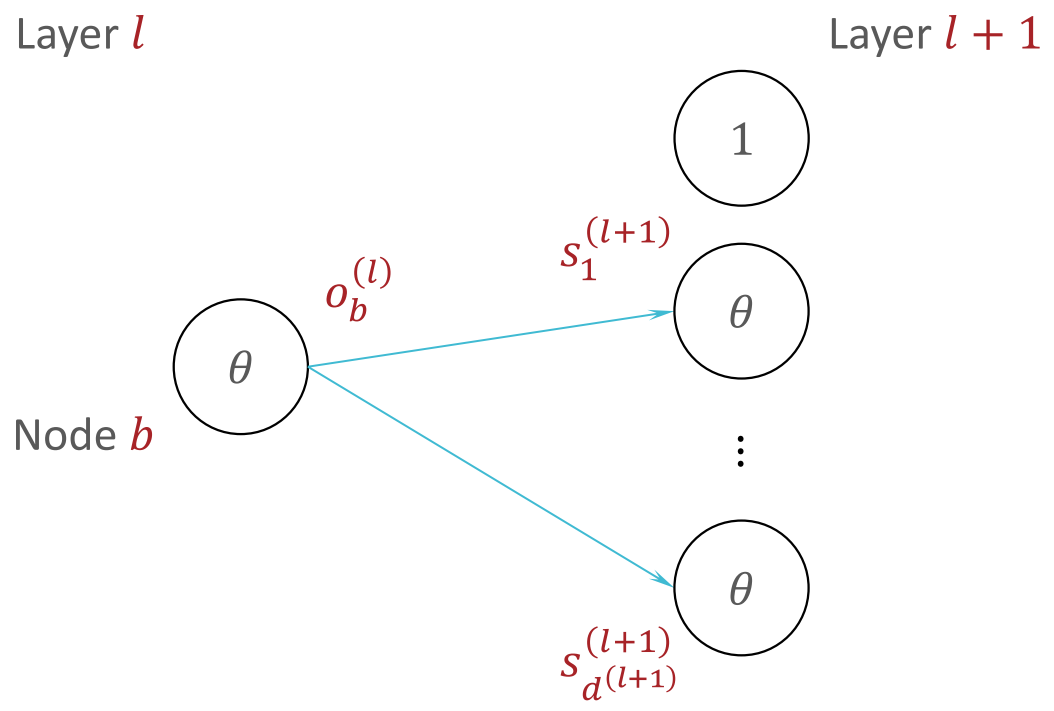
if $\theta(\cdot) = \tanh(\cdot)$

$$\frac{\partial o_b^{(\ell)}}{\partial s_b^{(\ell)}} = 1 - \left(\tanh(s_b^{(\ell)})\right)^2 = 1 - o_b^{(\ell)2}$$

(???)

Computing Partial Derivatives

Insight: $o_b^{(l)}$ affects $e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)$ via $s_1^{(l+1)}, \dots, s_{d^{(l+1)}}^{(l+1)}$



Computing Partial Derivatives

Insight: $o_b^{(l)}$ affects $e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)$ via $s_1^{(l+1)}, \ldots, s_{d^{(l+1)}}^{(l+1)}$

$$\frac{\partial e(o^{(L)}, y^{(n)})}{\partial o_b^{(l)}} = \sum_{c=1}^{d^{(l+1)}} \frac{\partial e(o^{(L)}, y^{(n)})}{\partial s_c^{(l+1)}} \frac{\partial s_c^{(l+1)}}{\partial o_b^{(l)}}$$

$$s_c^{(l+1)} = \sum_{B=0}^{d^{(l)}} w_{c,B}^{(l+1)} o_B^{(l)}$$

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

$$\delta_c^{(l+1)} := \frac{\partial e(o^{(L)}, y^{(n)})}{\partial s_c^{(l+1)}}$$

$$\frac{\partial e(o^{(L)}, y^{(n)})}{\partial o_b^{(l)}} = \sum_{c=1}^{d^{(l+1)}} \delta_c^{(l+1)} w_{c,b}^{(l+1)}$$

Computing Partial Derivatives

$$\delta_b^{(l)} = \frac{\partial e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)}{\partial o_b^{(l)}} \left(\frac{\partial o_b^{(l)}}{\partial s_b^{(l)}}\right)$$

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

(assuming hyperbolic tangent)

$$\vec{\delta}^{(\ell)} := \nabla_{\vec{s}^{(\ell)}} e\left(o^{(L)}, y^{(n)}\right)$$

Computing Gradients

$$\frac{\partial e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right)}{\partial w_{b,a}^{(l)}} = \delta_b^{(l)}\left(\frac{\partial s_b^{(l)}}{\partial w_{b,a}^{(l)}}\right) = \delta_b^{(l)}\left(o_a^{(l-1)}\right)$$

$$\nabla_{W^{(l)}} e\left(\boldsymbol{o}^{(L)}, y^{(n)}\right) = \boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T}$$

Sanity check: $\nabla_{W^{(l)}} e\left(o^{(L)}, y^{(n)}\right) \in \mathbb{R}^{d^{(l)} \times (d^{(l-1)}+1)}$

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

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

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

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:

$$\boldsymbol{\delta}^{(L)} = \delta_1^{(L)}, \boldsymbol{o}^{(L)} = o_1^{(L)} \text{ and } e\left(o_1^{(L)}, y^{(n)}\right) = \left(o_1^{(L)} - y^{(n)}\right)^2$$

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

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

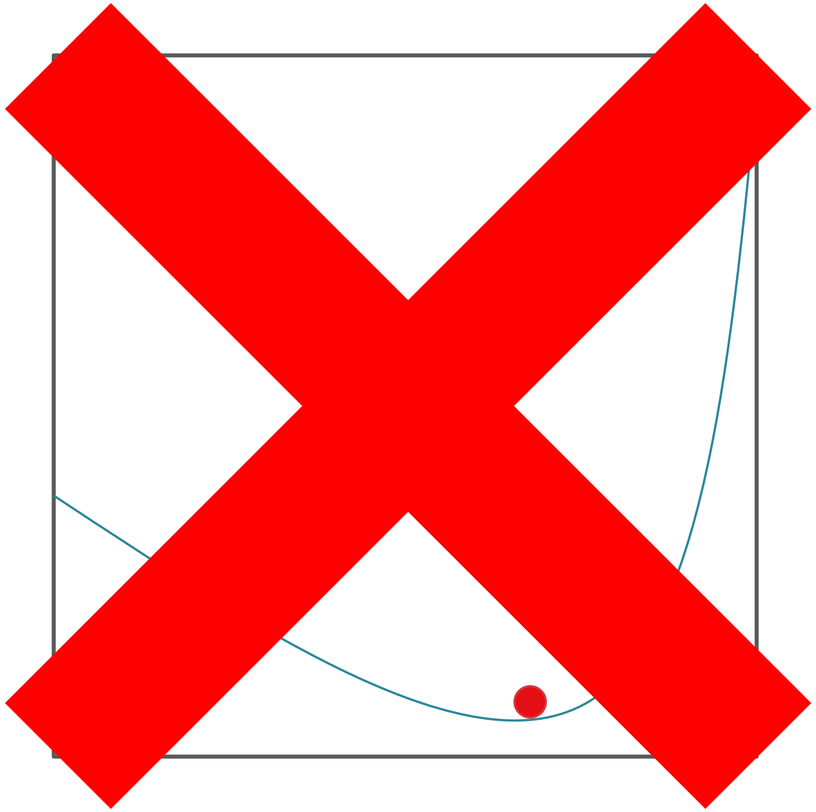
when $\theta(\cdot) = \tanh(\cdot)$

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 $W^{(1)}, \dots, W^{(L)}$

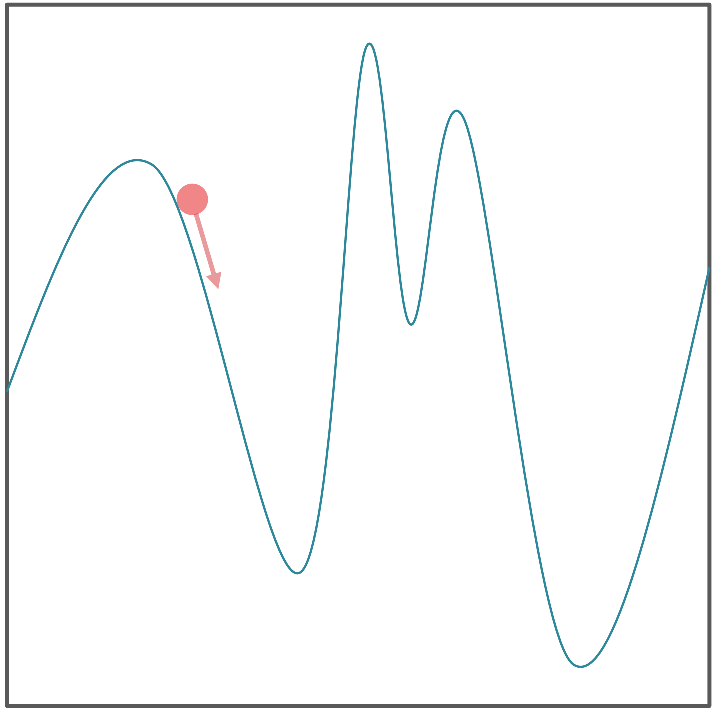
Recall: Gradient Descent

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

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 a data point from $\mathcal{D}$, $(\boldsymbol{x}^{(n)}, y^{(n)})$
   b. Compute the pointwise gradient,
   $$G^{(l)} = \nabla_{W^{(l)}} e(\boldsymbol{o}^{(L)}, y^{(n)}) \; \forall \, l$$
   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: $W_t^{(1)}, \ldots, W_t^{(L)}$

Mini-batch Stochastic Gradient Descent for Neural Networks

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

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 B data points from $\mathcal{D}$, $\{(\boldsymbol{x}^{(b)}, y^{(b)})\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,

$$G^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} e(\boldsymbol{o}^{(L)}, y^{(b)}) \ \forall\, l$$

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

typically some function of N e.g.

$B(N) = \frac{N}{10}$

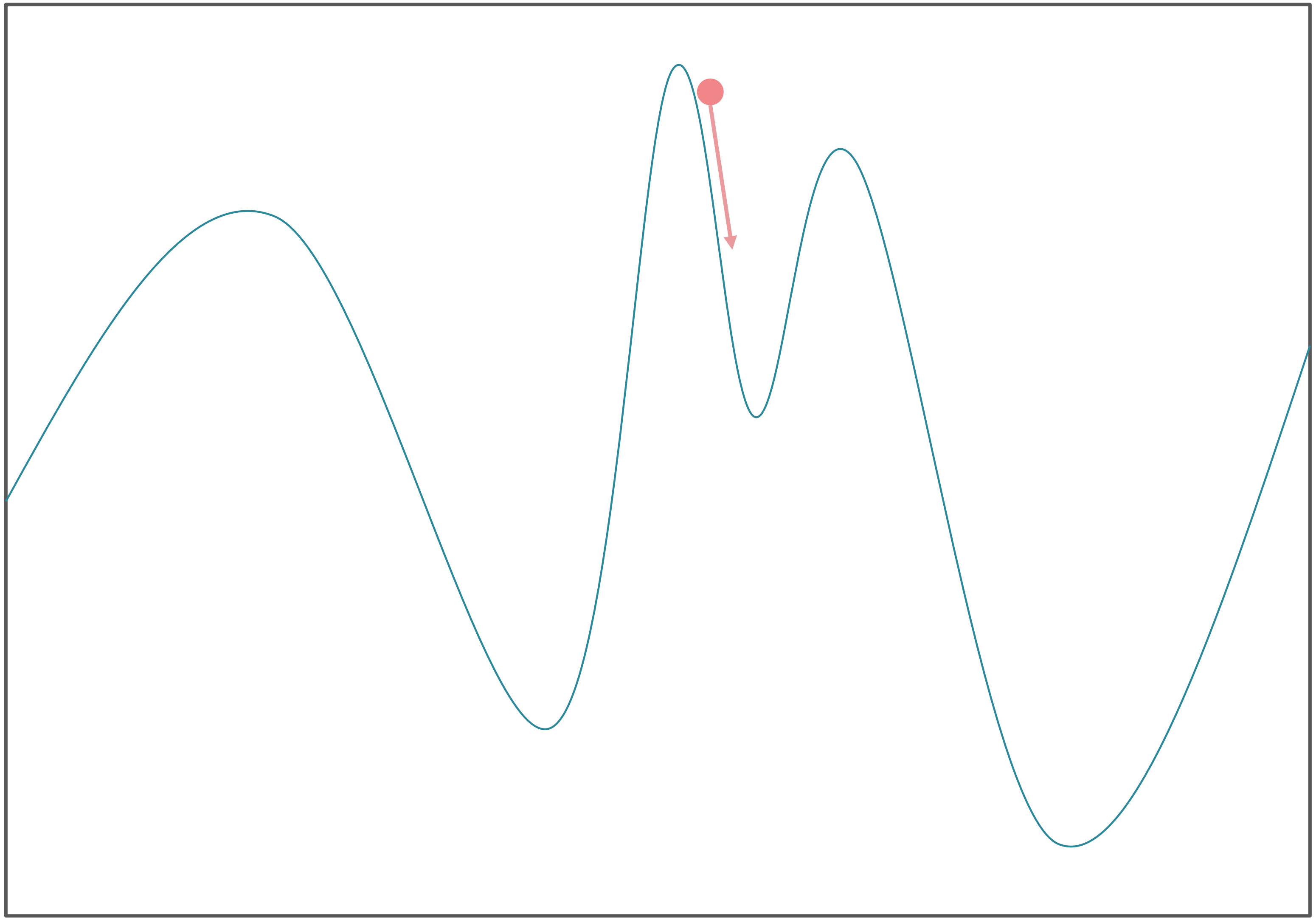
Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks

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

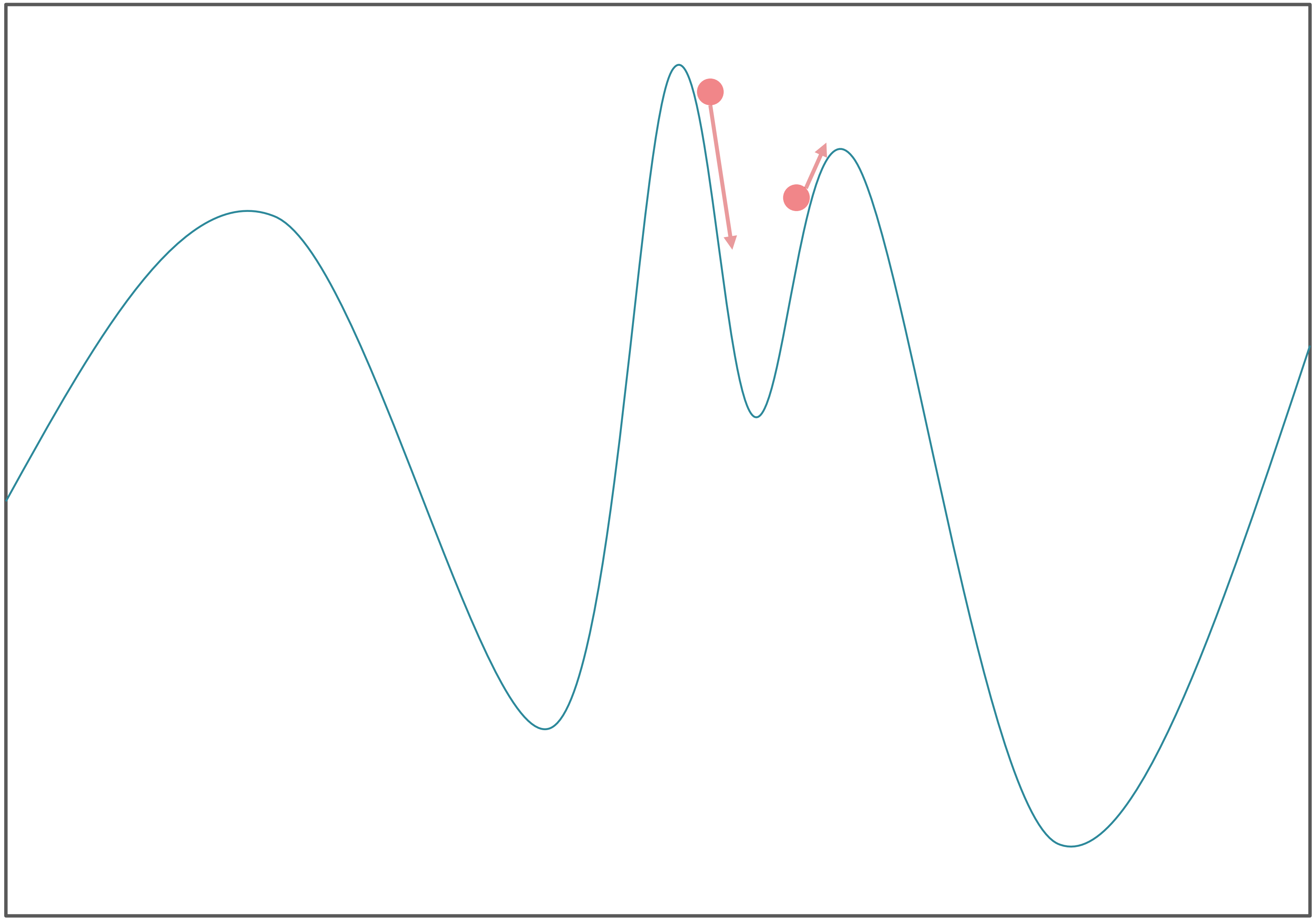
1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $G_{-1}^{(l)} = 0 \odot W^{(l)} \; \forall \, l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)},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

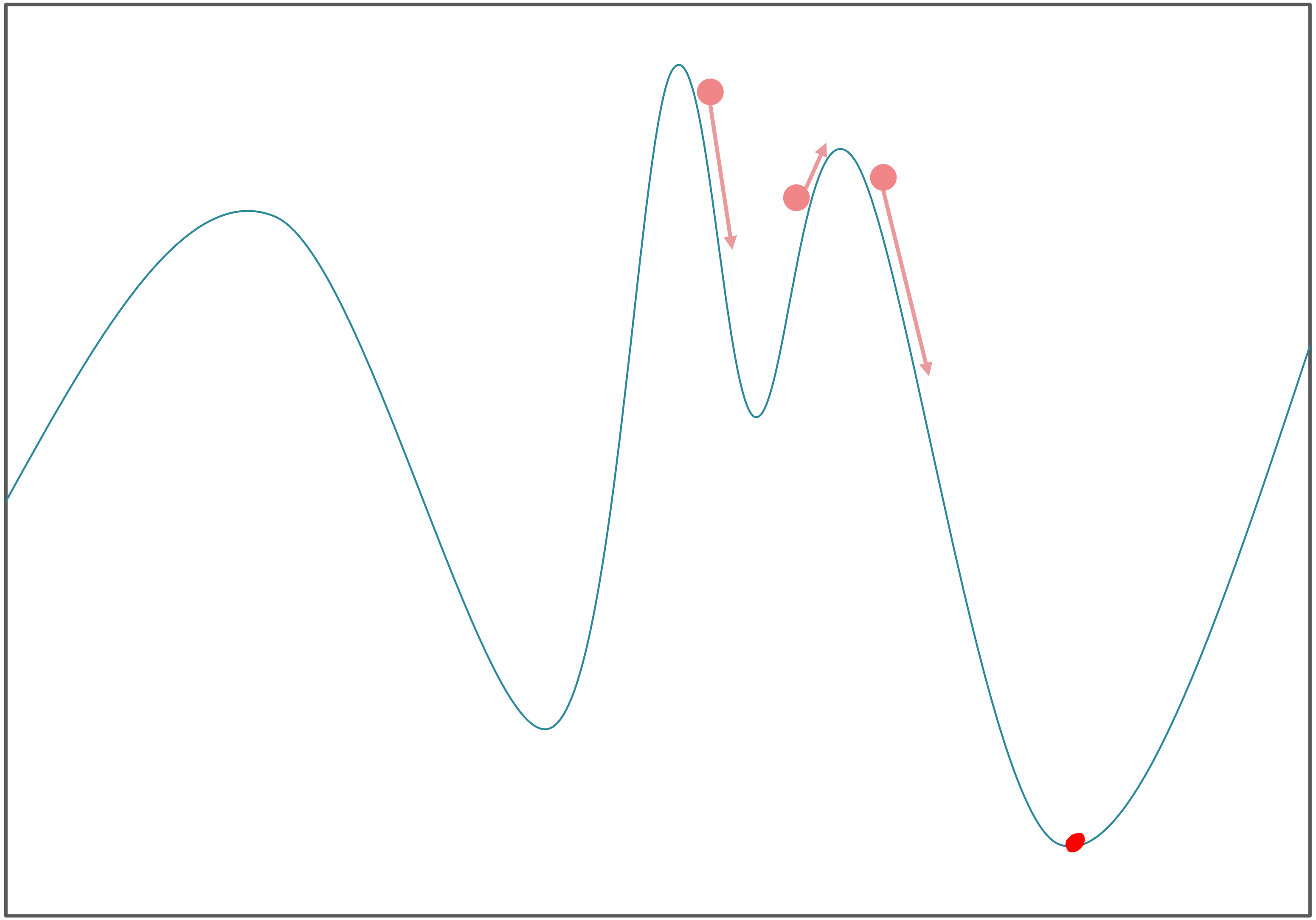
Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks



Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks



Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks



Mini-batch Stochastic Gradient Descent with Adaptive Gradients for Neural Networks

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

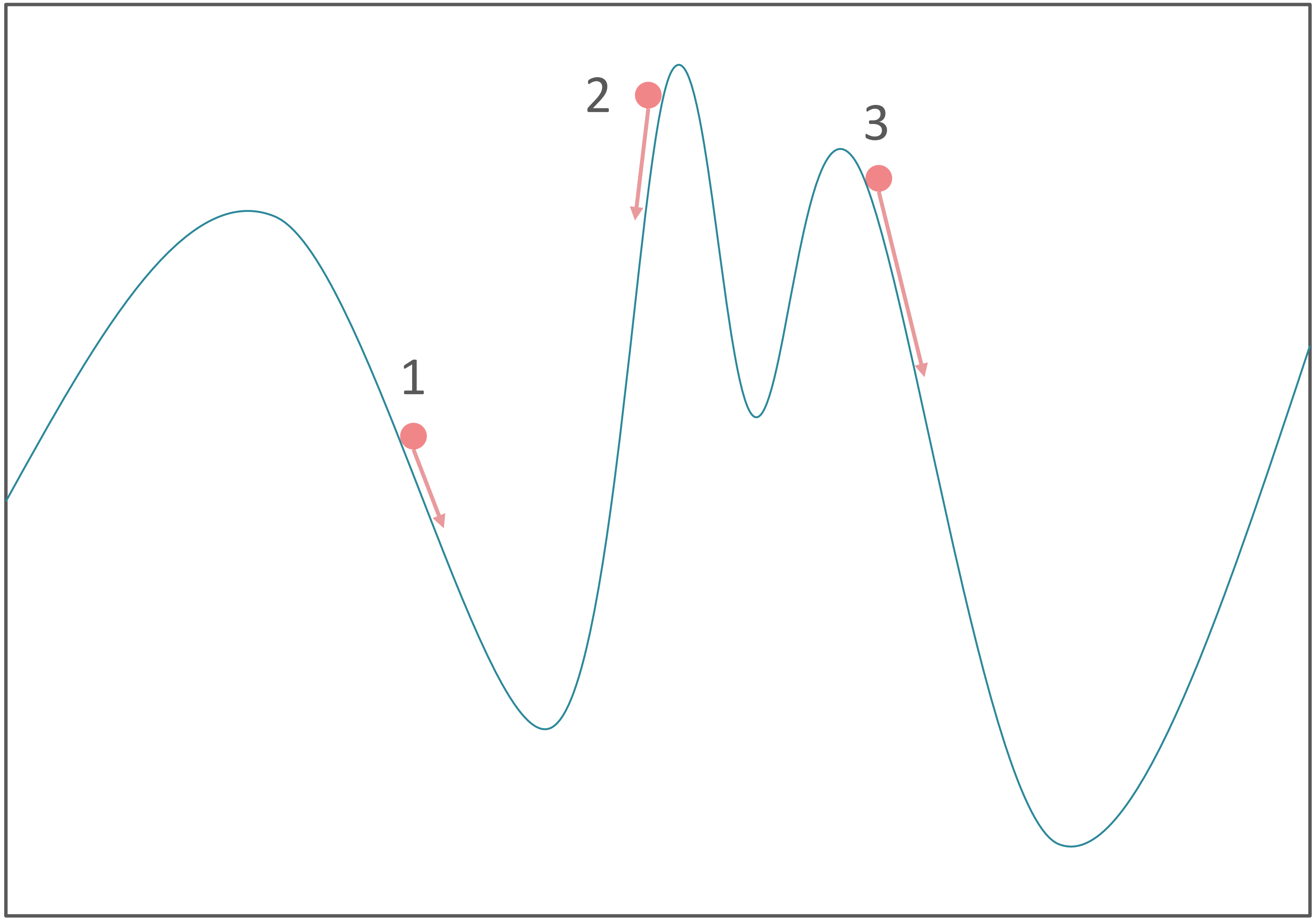
1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $S_{-1}^{(l)} = 0 \odot W^{(l)} \; \forall \, l = 1, \dots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\{(\boldsymbol{x}^{(b)}, y^{(b)})\}_{b=1}^{B}$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

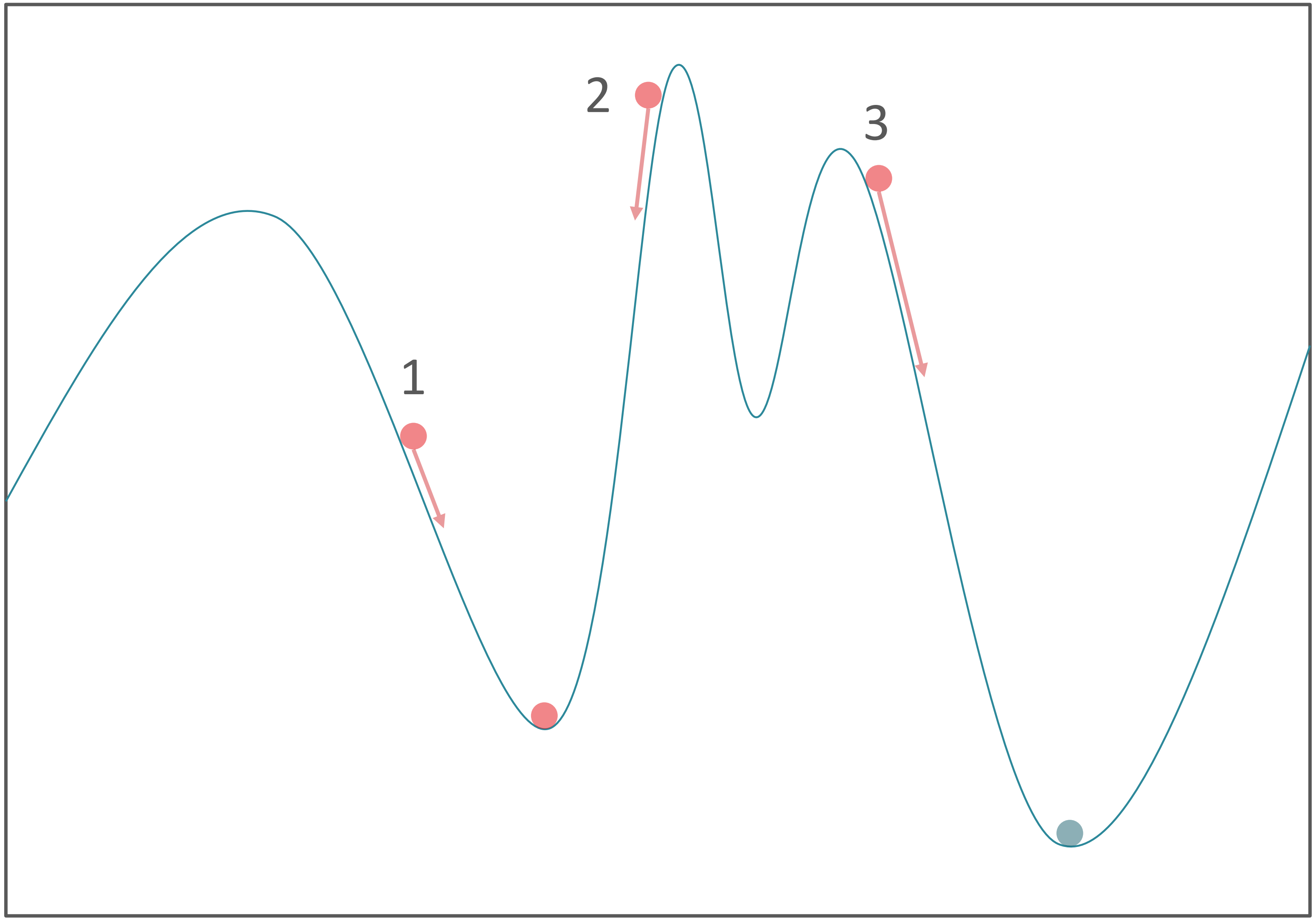
Random Restarts

- 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

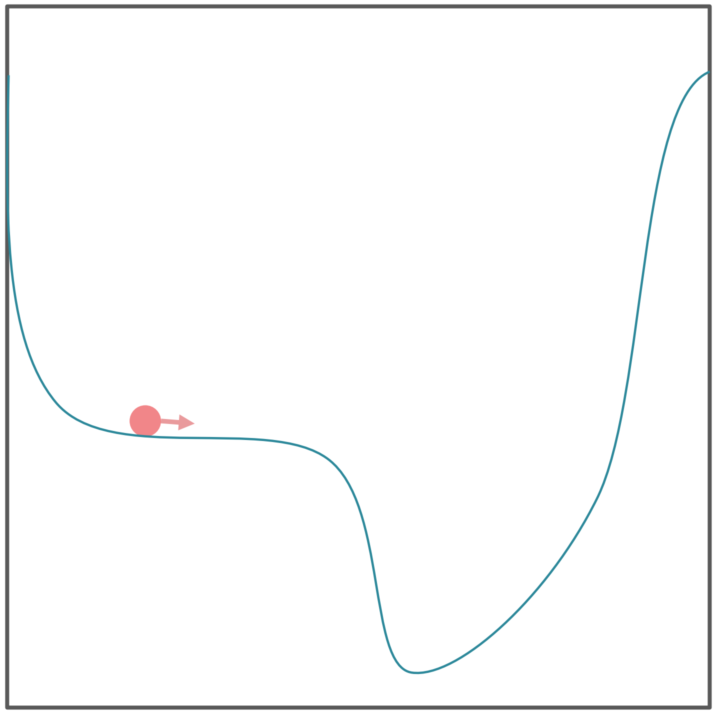


Random Restarts



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 $\ell_{\mathcal{D}}^{AUG}\left(W^{(1)}, \ldots, W^{(L)}, \lambda_C\right)$

$$= \ell_{\mathcal{D}}\left(W^{(1)}, \ldots, W^{(L)}\right) + \lambda_C \Omega\left(W^{(1)}, \ldots, W^{(L)}\right)$$

e.g. L2 regularization

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

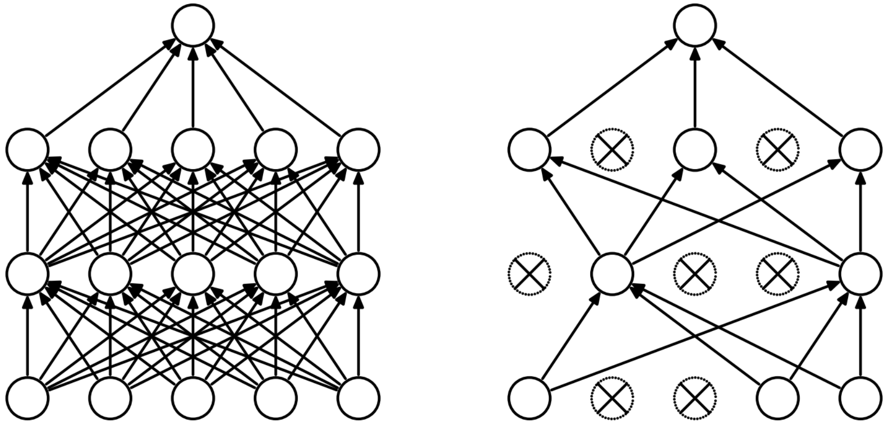
Neural Networks and "Strange" Regularization (Bishop, 1995)

- 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. $(\boldsymbol{x}^{(n)}, y^{(n)})$, use $(\boldsymbol{x}^{(n)} + \boldsymbol{\epsilon}, y^{(n)})$ 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 Ω 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

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