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

Henry Chai

6/12/24

Front Matter

- Announcements
  - HW5 released 6/11, due 6/18 at 11:59 PM
  - Midterm on Friday, 6/21 *tentatively* from 4 PM to 7 PM in SH 105
    - Reminder: all of this week's material is in-scope
- Recommended Readings
  - Mitchell, Chapters 4.1 – 4.6

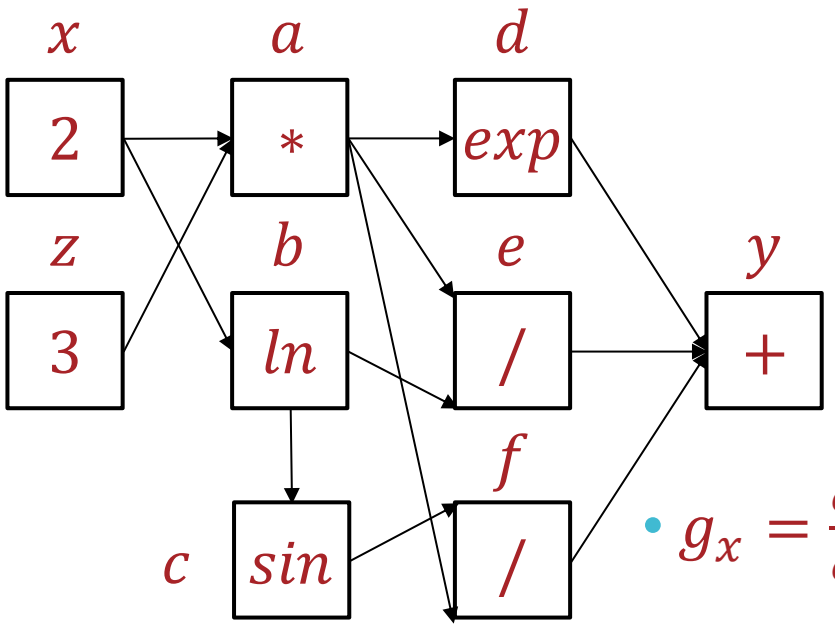
Recall: Automatic Differentiation (reverse mode)

- Given

$$y = f(x, z) = e^{xz} + \frac{xz}{\ln(x)} + \frac{\sin(\ln(x))}{xz}$$

what are $\partial y/\partial x$ and $\partial y/\partial z$ at $x = 2, z = 3$?

- Then compute partial derivatives, starting from y and working back



- $g_y = \frac{\partial y}{\partial y} = 1$
- $g_d = g_e = g_f = 1$
- $g_c = \frac{\partial y}{\partial c} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial c} = g_f\left(\frac{1}{a}\right)$
- $g_b = \frac{\partial y}{\partial b} = \frac{\partial y}{\partial e}\frac{\partial e}{\partial b} + \frac{\partial y}{\partial c}\frac{\partial c}{\partial b}$
  $= g_e\left(-\frac{a}{b^2}\right) + g_c(\cos(b))$
- $g_a = \frac{\partial y}{\partial a} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial a} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial a} + \frac{\partial y}{\partial d}\frac{\partial d}{\partial a}$
  $= g_f\left(\frac{-c}{a^2}\right) + g_e\left(\frac{1}{b}\right) + g_d(e^a)$
- $g_x = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial b}\frac{\partial b}{\partial x} + \frac{\partial y}{\partial a}\frac{\partial a}{\partial x} = g_b\left(\frac{1}{x}\right) + g_a(z)$
- $g_z = \frac{\partial y}{\partial z} = \frac{\partial y}{\partial a}\frac{\partial a}{\partial z} = g_a(x)$

Example courtesy of Matt Gormley

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
- 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 $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ 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: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$

Computing Gradients

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

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

all the weights going into node 2 in Layer l

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

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

Computing Gradients: Intuition

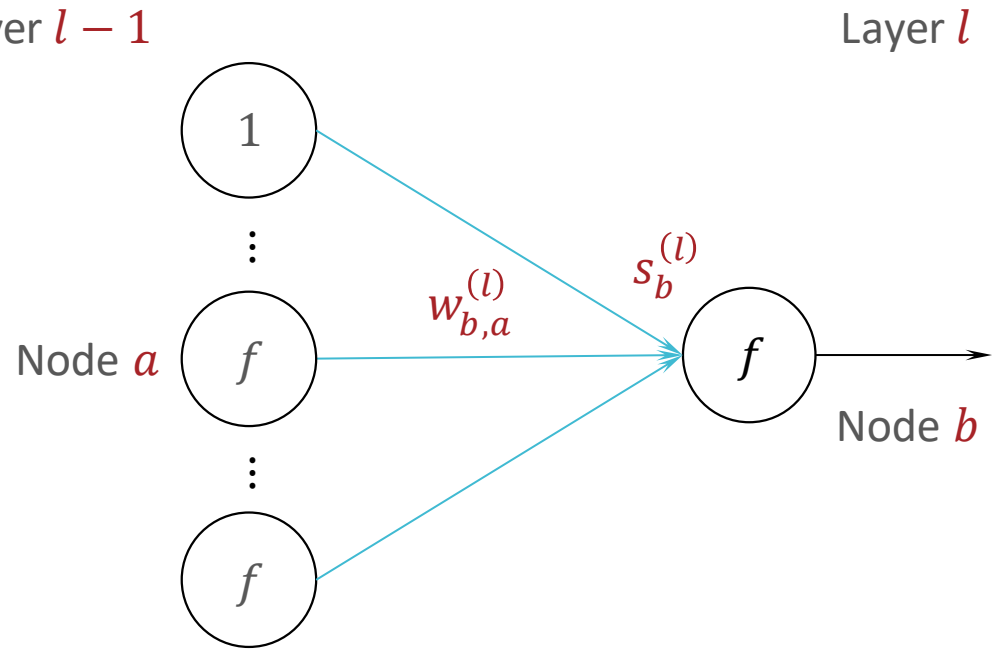
- A weight affects the prediction of the network (and therefore the error) through downstream signals/outputs
  - Use the chain rule!
- Any weight going into the same node will affect the prediction through the same downstream path
  - Compute derivatives starting from the last layer and move “backwards”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

Computing Partial Derivatives

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

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

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



Computing Partial Derivatives

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

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

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

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

$s_b^{(l)} = \sum_{a=0}^{d^{(l-1)}} w_{ba}^{(l)} o_a^{(l-1)}$

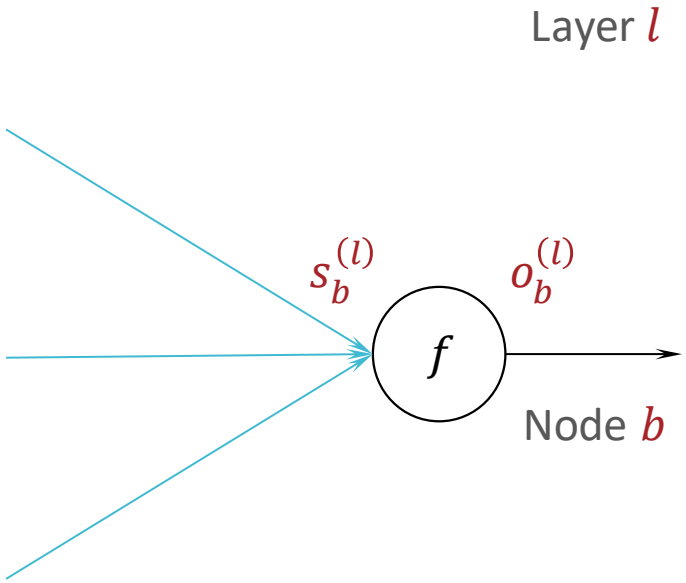
$\Downarrow$

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

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

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 $o_b^{(l)}$

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

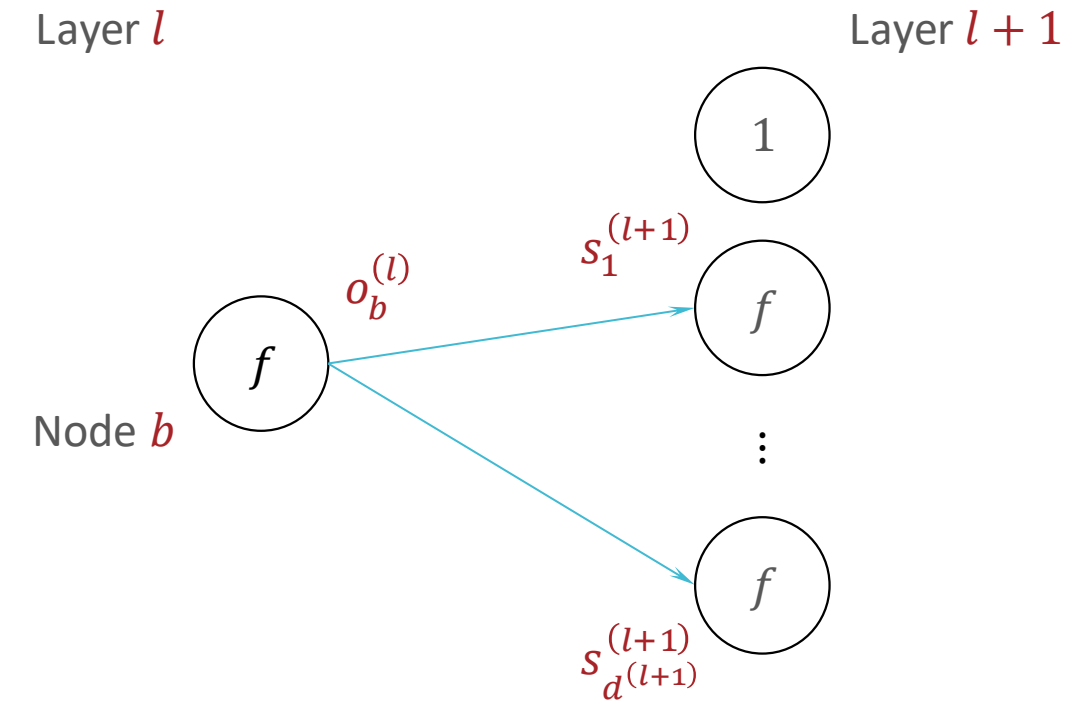
$o_b^{(l)} = f(s_b^{(l)})$

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

for example, $f(\cdot) = \tanh(\cdot) \Rightarrow 1 - \tanh(s_b^{(l)})^2$

Computing Partial Derivatives

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



Computing Partial Derivatives

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

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

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

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

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

Computing Partial Derivatives

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

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

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

Based solely on their shape alone, which of the following could be an expression for $\delta^{(l)} = \nabla_{\mathbf{s}^{(l)}} \ell^{(n)}(W_t^{(1)}, \ldots, W_t^{(L)})$? 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

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

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

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

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

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

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

$w^{(l+1)^T} \delta^{(l+1)} \in \mathbb{R}^{d^{(l)}+1}$ which is almost the right size

Computing Gradients

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

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

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

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

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

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

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

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

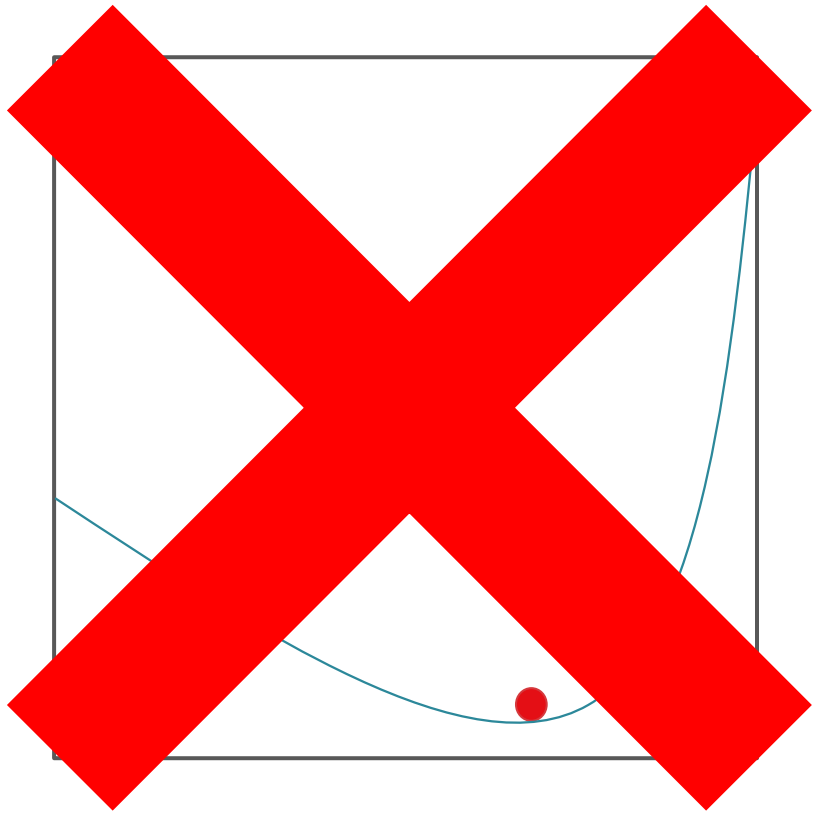
when $f(\cdot) = \tanh(\cdot)$

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

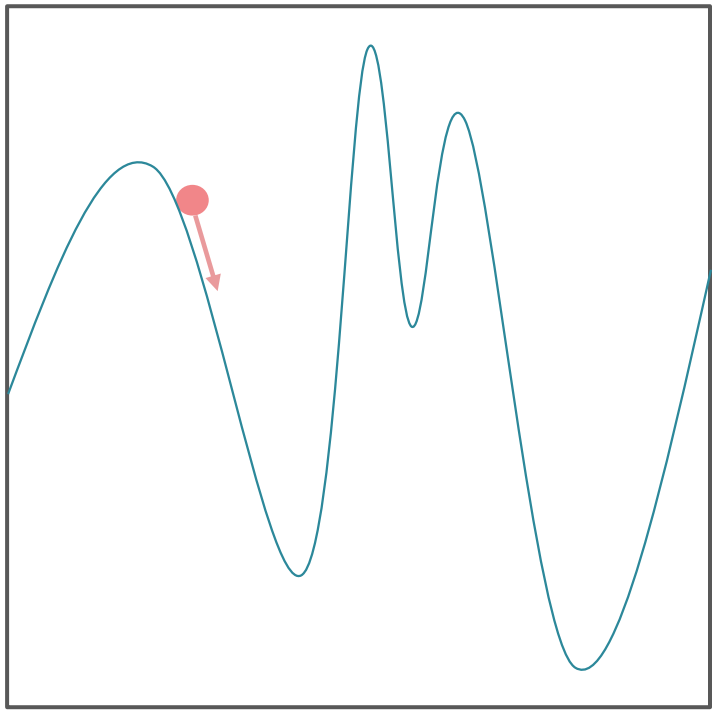
Recall: Gradient Descent

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

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

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

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

Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks

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

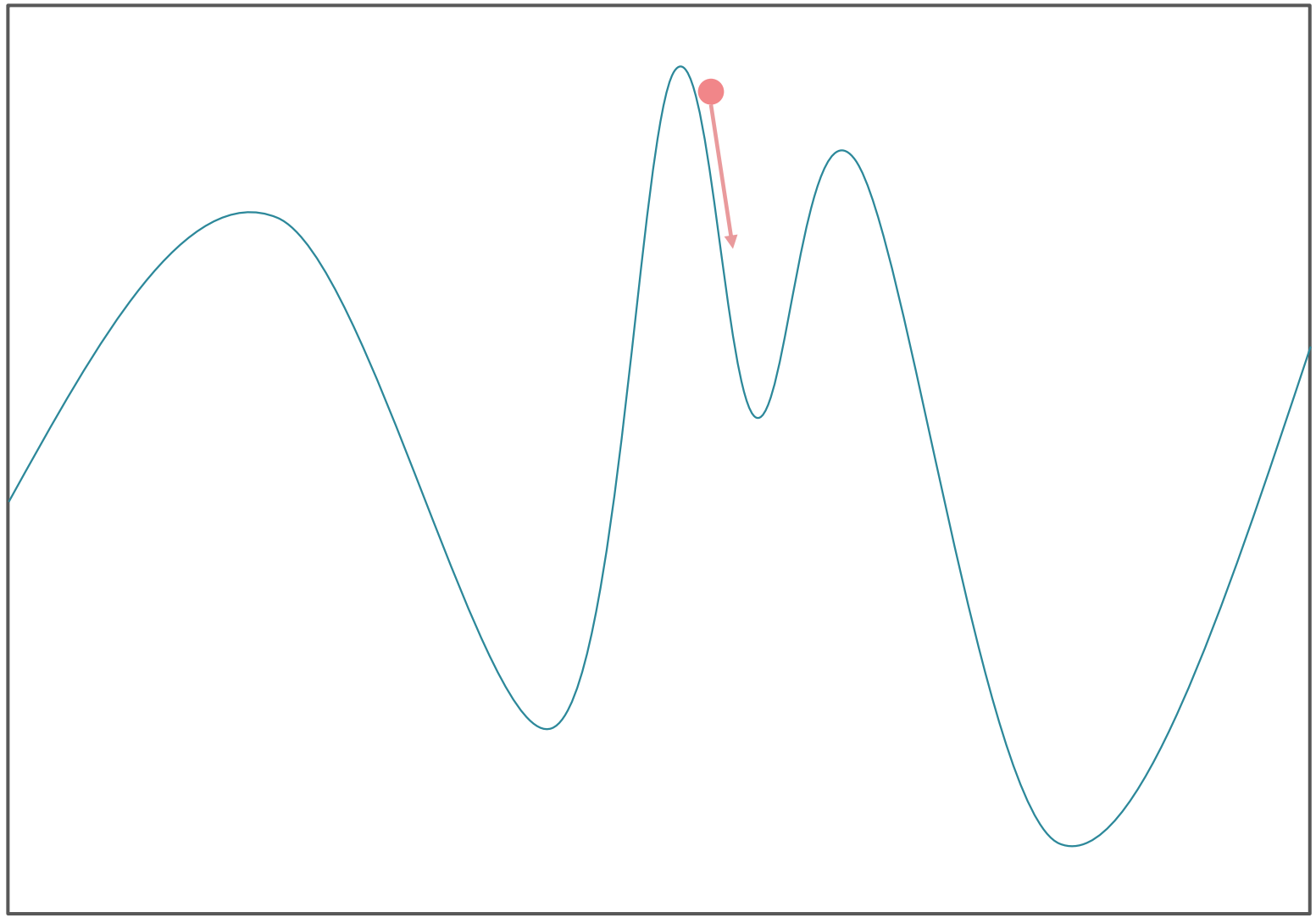
1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $G_{-1}^{(l)} = 0 \odot W^{(l)} \; \forall \; l = 1, \dots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)},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

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

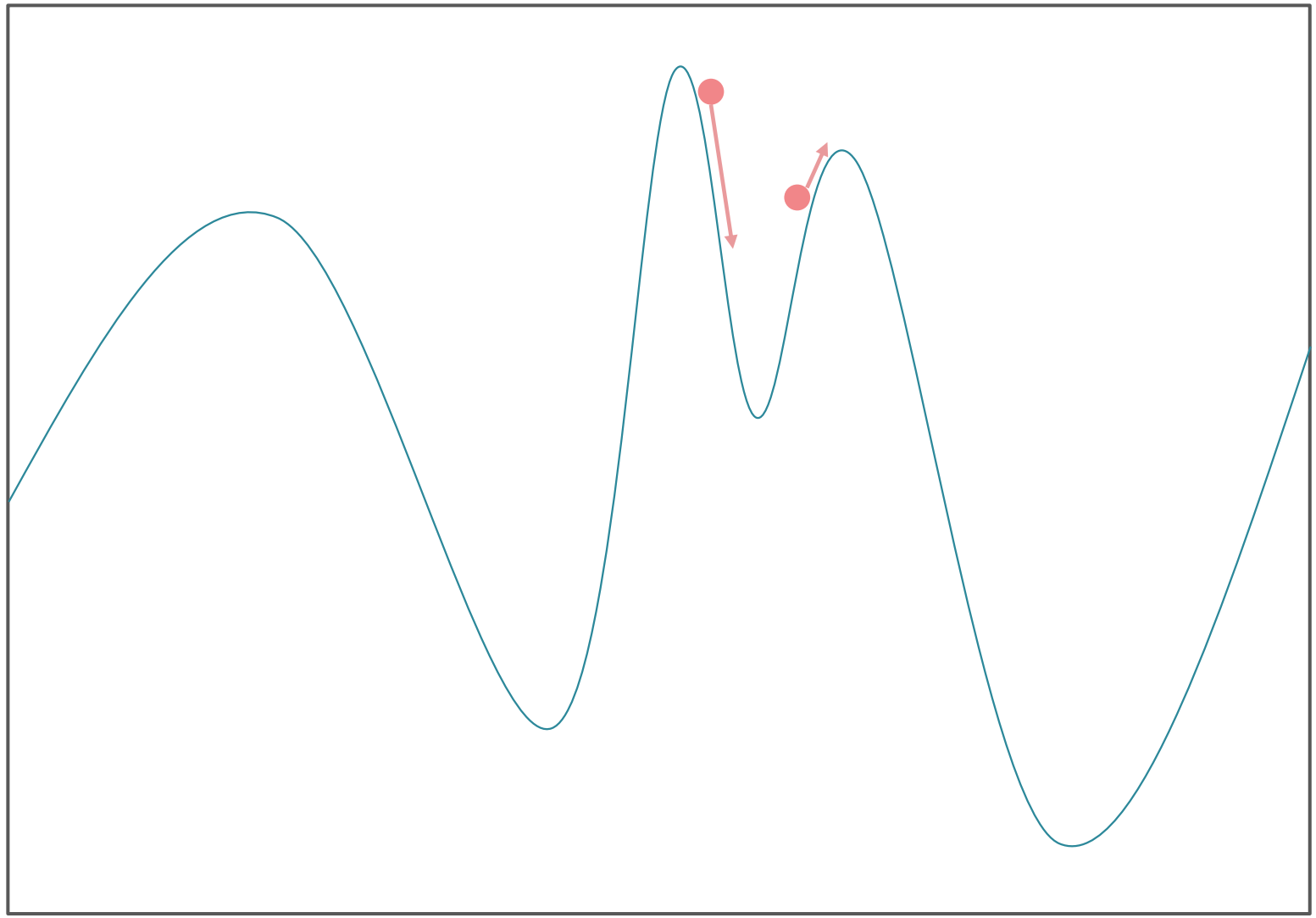
   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta_{MB}^{(0)} \left(\beta G_{t-1}^{(l)} + G_t^{(l)}\right) \forall \; l$
   d. Increment t: $t \leftarrow t + 1$

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

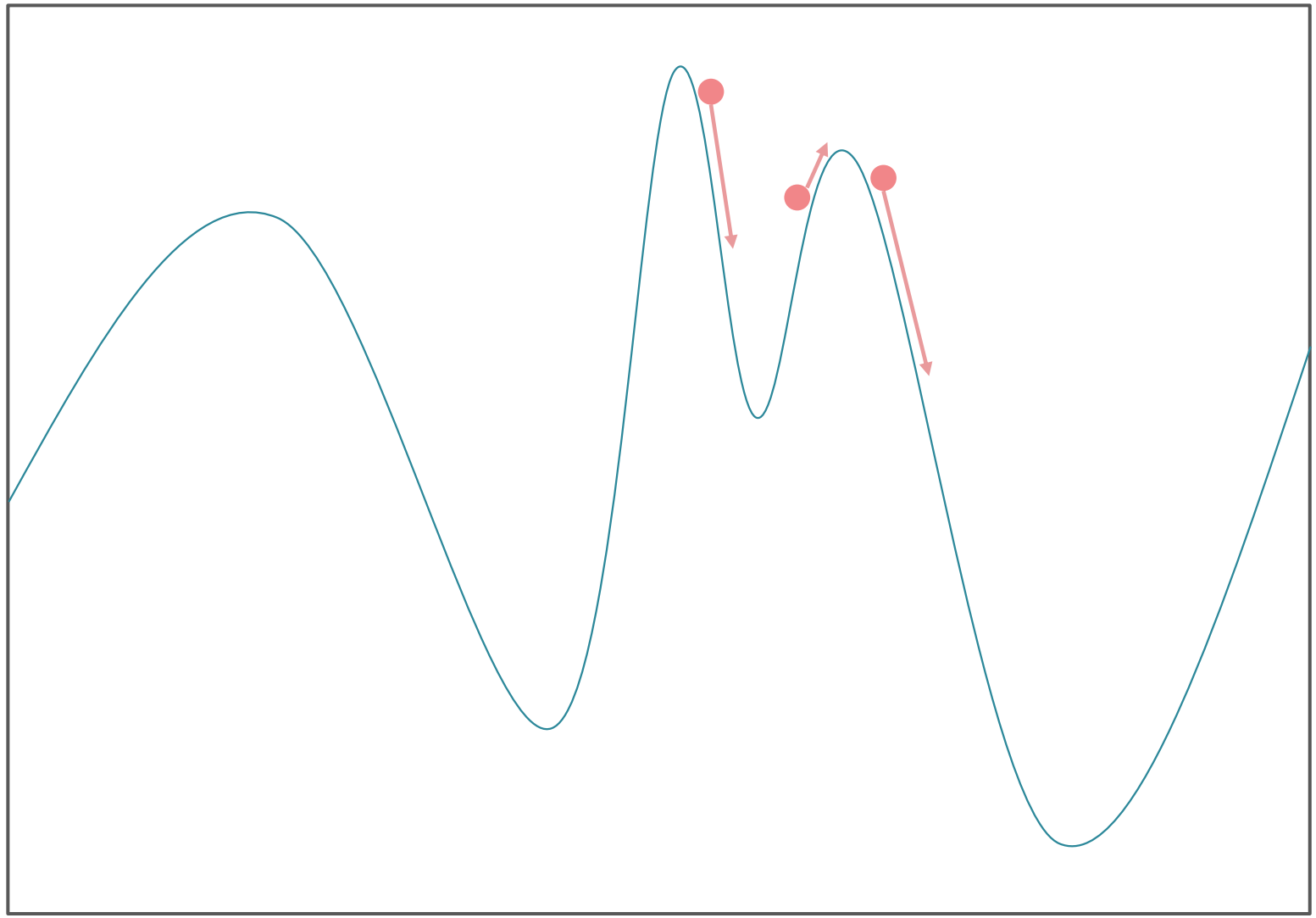
Mini-batch Stochastic Gradient Descent with Momentum for Neural Networks



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 Root Mean Square Propagation (RMSProp)

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

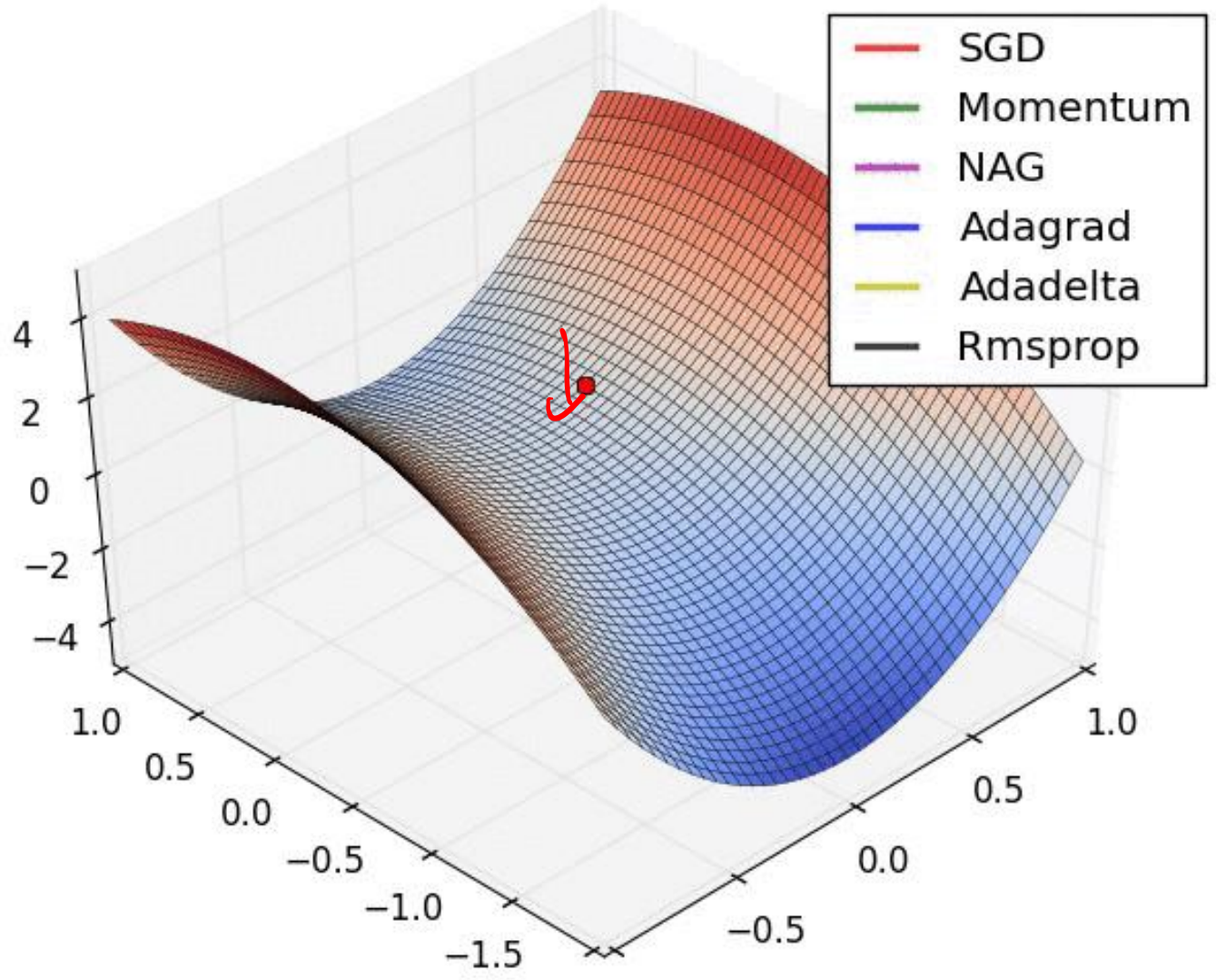
1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $S_{-1}^{(l)} = 0 \odot W^{(l)} \; \forall \, l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   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 the scaling factor: $S_t = \beta S_{t-1} + (1-\beta)(G_t \odot G_t)$
   d. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \frac{\gamma}{\sqrt{S_t}} \odot G_t$
   e. Increment t: $t \leftarrow t + 1$

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

Mini-batch Stochastic Gradient Descent with Root Mean Square Propagation (RMSProp)



Source: https://www.ruder.io/optimizing-gradient-descent/

Adam (Adaptive Moment Estimation) = SGD + Momentum + RMSProp

- Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$, decay parameters β_1 and β_2

1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $M_{-1} = S_{-1} = 0 \odot W^{(l)} \; \forall \, l = 1, \dots, L$
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 (G_t), momentum and scaling factor

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1) G_t$$

$$S_t = \beta_2 S_{t-1} + (1 - \beta_2)(G_t \odot G_t)$$

   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \frac{\eta}{\sqrt{S_t/(1-\beta_2^t)}} \odot (M_t/(1-\beta_1^t))$

   d. Increment t: $t \leftarrow t + 1$

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

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