10-301/601: Introduction to Machine Learning Lecture 15 – Differentiation

Henry Chai

5/27/25

Front Matter

- Announcements:
  - HW4 released on 5/23, due **5/28** (tomorrow) at 11:59 PM
  - Midterm on 5/30 at 9:30 AM in BH A36
    - Lectures 1 – 14 are in-scope; **this week's lectures will not be tested on the midterm**
    - Recitation on 5/29 will be a review of the practice problems

Recall: 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.

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

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

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

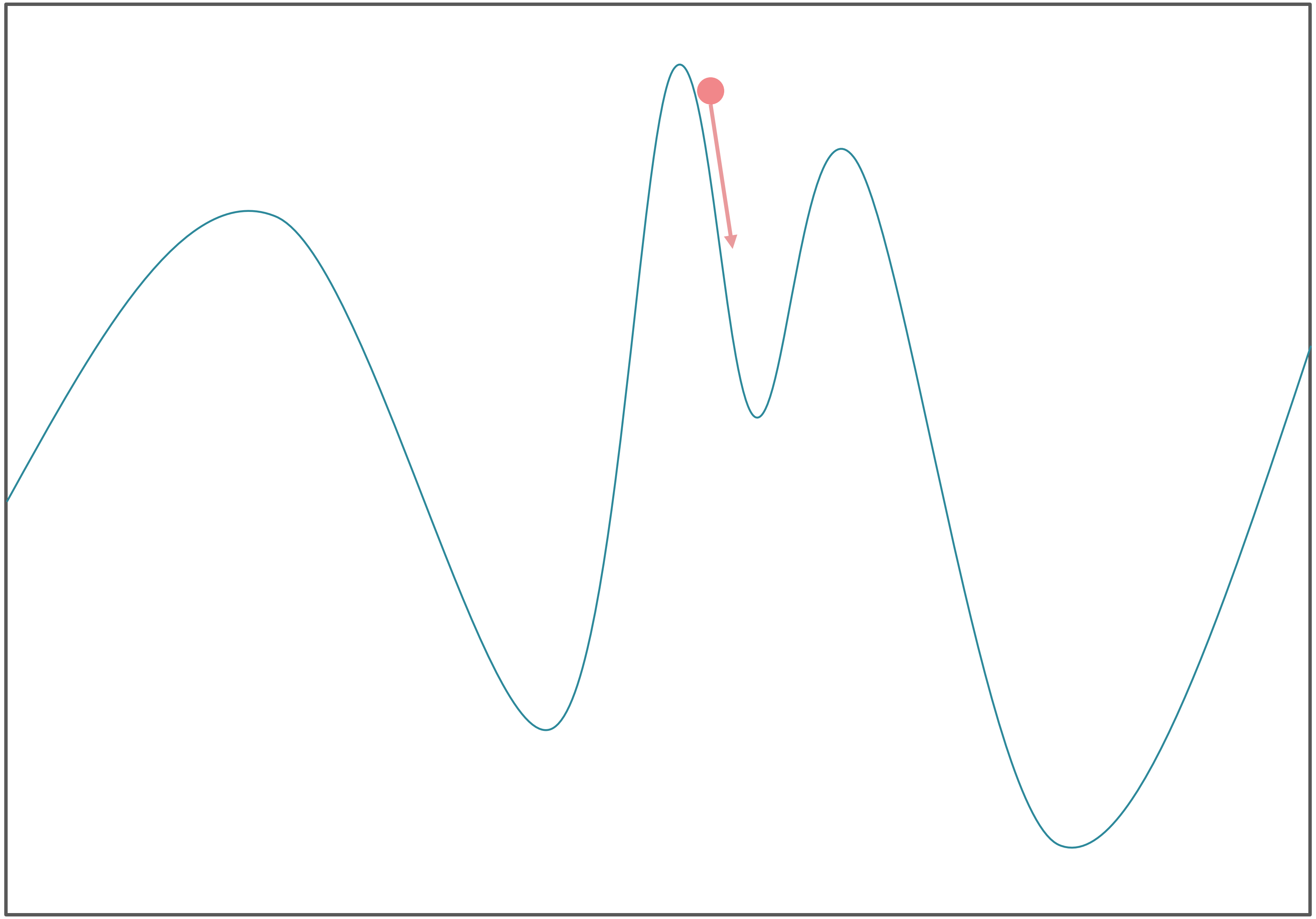
Mini-batch Stochastic Gradient Descent with Momentum for Learning

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

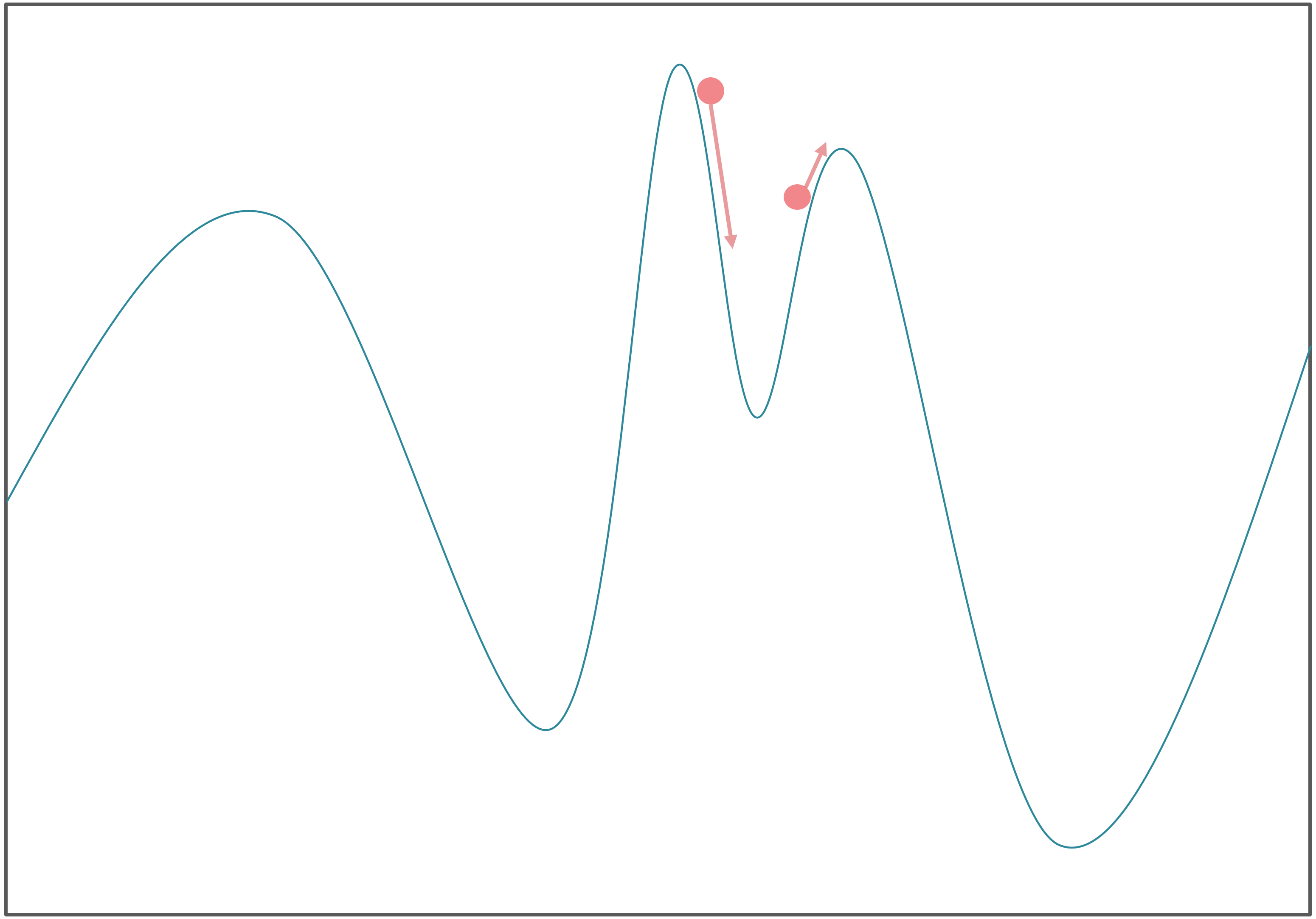
1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $G_{-1}^{(l)} = 0 \odot W^{(l)} \ \forall\, l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)},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

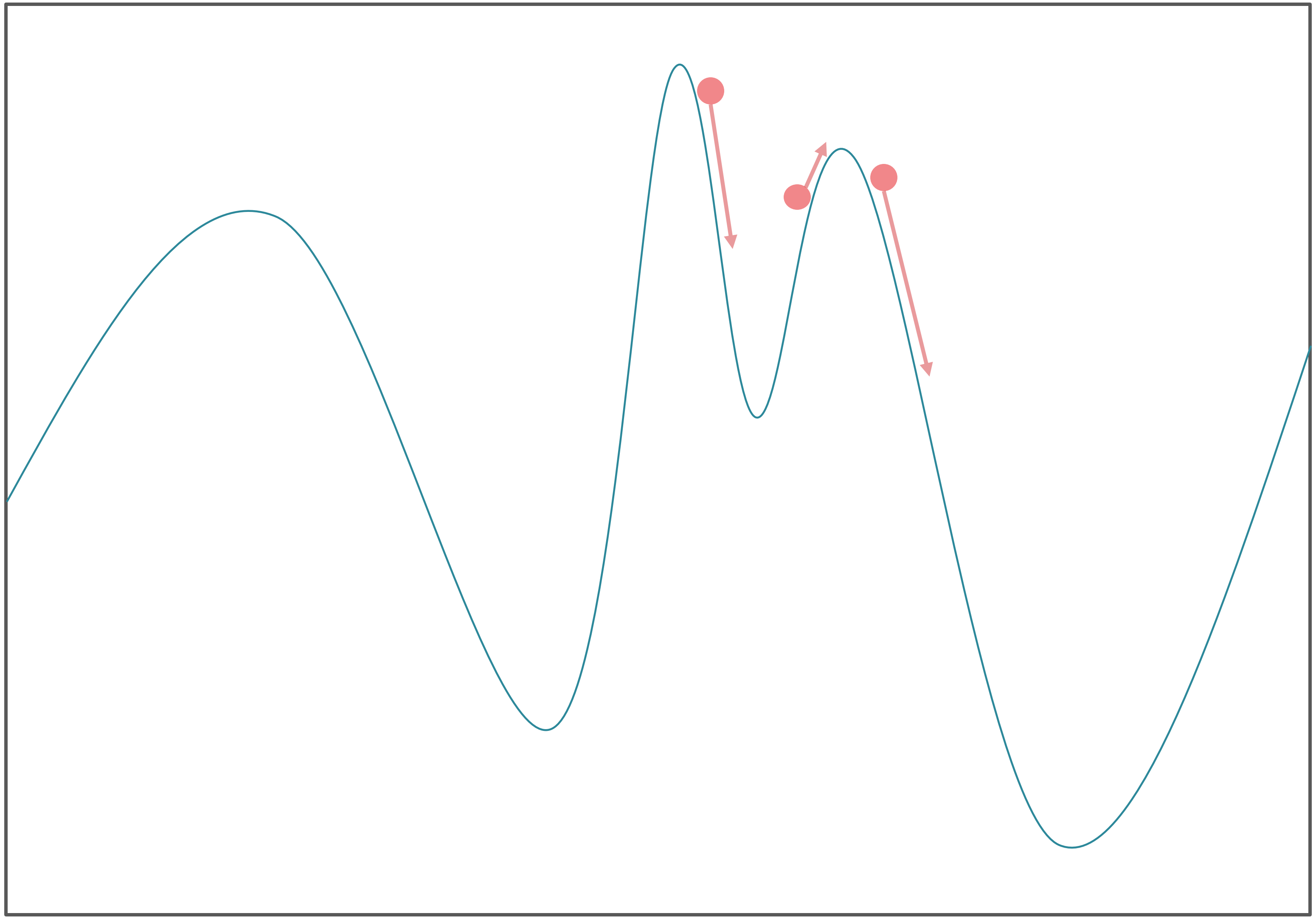
Mini-batch Stochastic Gradient Descent with Momentum for Learning



Mini-batch Stochastic Gradient Descent with Momentum for Learning



Mini-batch Stochastic Gradient Descent with Momentum for Learning



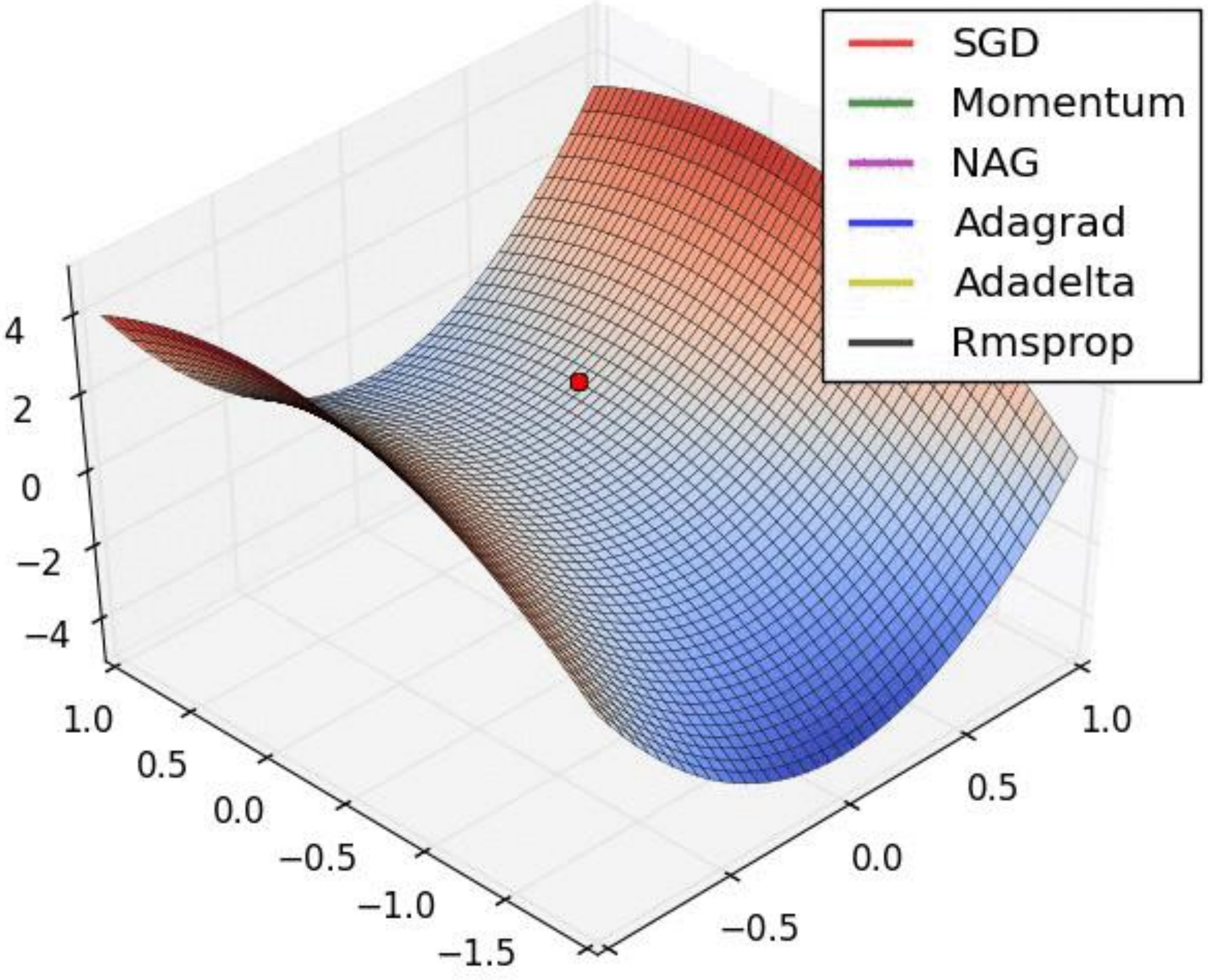
Mini-batch Stochastic Gradient Descent with Root Mean Square Propagation (RMSProp)

- 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 $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ 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 $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   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 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: $W_t^{(1)}, \dots, W_t^{(L)}$

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

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

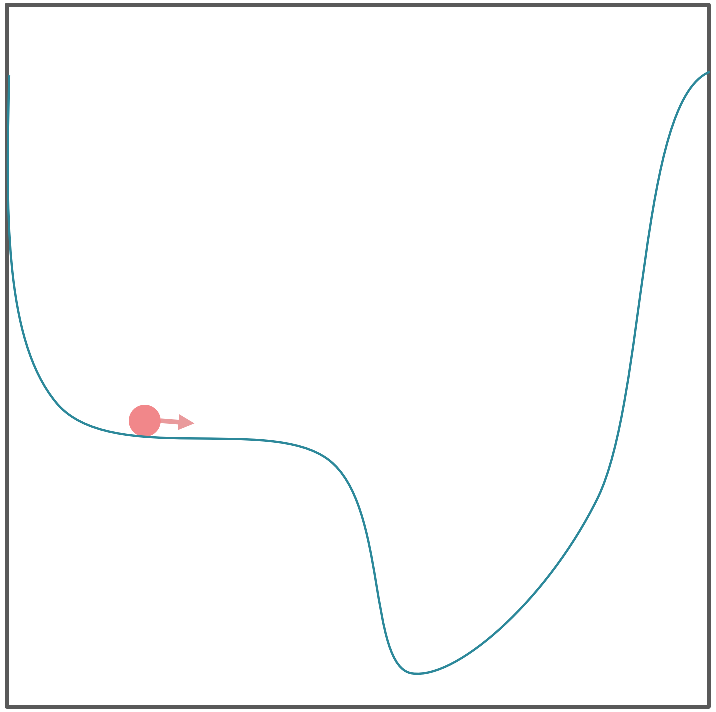
$$M_t = \beta_1 M_{t-1} + (1 - \beta_1) G_t$$
$$S_t = \beta_2 S_{t-1} + (1 - \beta_2)(G_t \odot G_t)$$

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

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

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

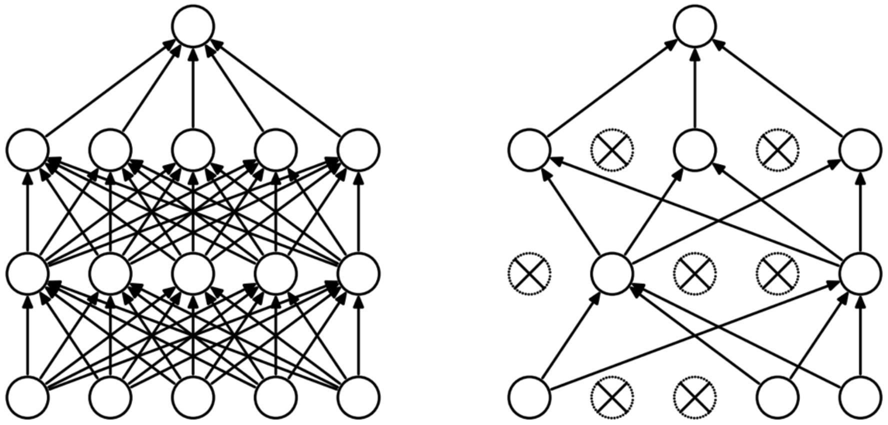
$$= \ell_{\mathcal{D}}\left(W^{(1)}, \dots, W^{(L)}\right) + \lambda_C r\left(W^{(1)}, \dots, W^{(L)}\right)$$

e.g. L2 regularization

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

Neural Networks and "Strange" Regularization (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

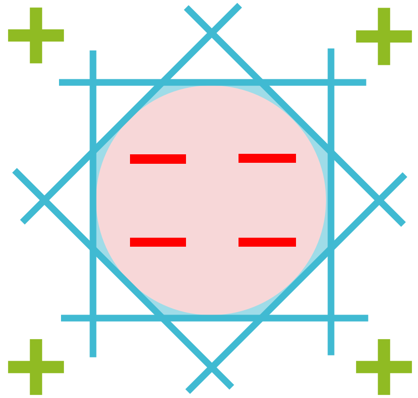


(a) Standard Neural Net (b) After applying dropout.

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

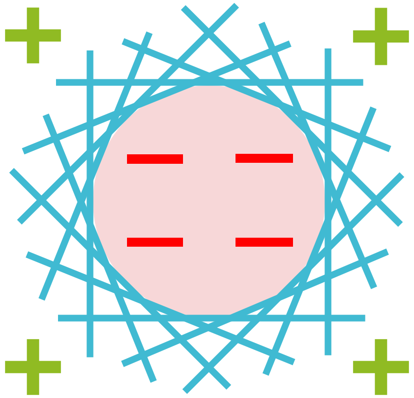
MLPs as Universal Approximators

- 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, θ, is continuous, bounded and non-constant.
- What about unbounded or discontinuous functions?
- Use more layers!

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, θ, 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.

Three Approaches to Differentiation

- Given $f\colon \mathbb{R}^D \rightarrow \mathbb{R}$, compute $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = {}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$

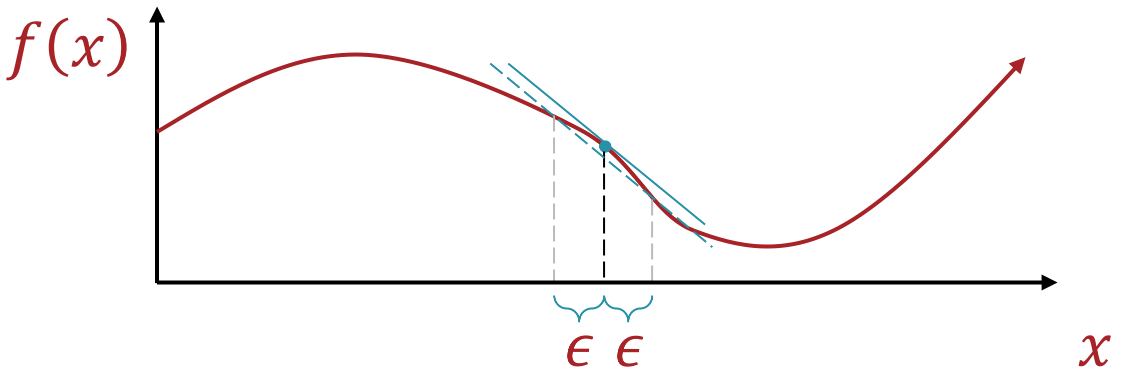
1. Finite difference method
2. Symbolic differentiation
3. Automatic differentiation (reverse mode)

Approach 1: Finite Difference Method

- Given $f\colon \mathbb{R}^D \to \mathbb{R}$, compute $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = {}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \epsilon \boldsymbol{d}_i) - f(\boldsymbol{x} - \epsilon \boldsymbol{d}_i)}{2\epsilon}$$

where $\boldsymbol{d}_i$ is a one-hot vector with a 1 in the i^{th} position



- We want ϵ to be small to get a good approximation but we run into floating point issues when ϵ is too small
- Getting the full gradient requires computing the above approximation for each dimension of the input

Approach 1: Finite Difference Method Example

- 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 $x = 2, z = 3$?

```
>>> import math
>>> y = lambda x,z:
math.exp(x*z)+(x*z)/math.log(x)+math.sin(math.log(x))/(x*z)
>>> x = 2
>>> z = 3
>>> e = 10**-8
>>> dydx = (y(x+e,z)-y(x-e,z))/(2*e)
>>> dydz = (y(x,z+e)-y(x,z-e))/(2*e)
>>> print(dydx, dydz)
```


Example courtesy of Matt Gormley

Three Approaches to Differentiation

- Given $f\colon \mathbb{R}^D \to \mathbb{R}$, compute $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = {}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$

1. Finite difference method
   - Requires the ability to call $f(\boldsymbol{x})$
   - Great for checking accuracy of implementations of more complex differentiation methods
   - Computationally expensive for high-dimensional inputs
2. Symbolic differentiation
3. Automatic differentiation (reverse mode)

Approach 2: Symbolic Differentiation

- 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 $x = 2, z = 3$?

- Looks like we're gonna need the chain rule!

Example courtesy of Matt Gormley

Approach 2: Symbolic Differentiation

- 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 $x = 2, z = 3$?

$$\frac{\partial y}{\partial x} = \frac{\partial}{\partial x}(e^{xz}) + \frac{\partial}{\partial x}\left(\frac{xz}{\ln(x)}\right) + \frac{\partial}{\partial x}\left(\frac{\sin(\ln(x))}{xz}\right)$$

$$= ze^{xz} + \frac{z}{\ln(x)} - \frac{z}{\ln(x)^2} + \frac{\cos(\ln(x))}{x^2 z} - \frac{\sin(\ln(x))}{x^2 z}$$

$$= 3e^6 + \frac{3}{\ln(2)} - \frac{3}{\ln(2)^2} + \frac{\cos(\ln(2))}{12} - \frac{\sin(\ln(2))}{12}$$

$$\frac{\partial y}{\partial z} = \frac{\partial}{\partial z}(e^{xz}) + \frac{\partial}{\partial z}\left(\frac{xz}{\ln(x)}\right) + \frac{\partial}{\partial z}\left(\frac{\sin(\ln(x))}{xz}\right)$$

$$= 2e^6 + \frac{2}{\ln(2)} - \frac{\sin(\ln(2))}{18}$$

Example courtesy of Matt Gormley

Three Approaches to Differentiation

- Given $f\colon \mathbb{R}^D \to \mathbb{R}$, compute $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = {}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$

1. Finite difference method
   - Requires the ability to call $f(\boldsymbol{x})$
   - Great for checking accuracy of implementations of more complex differentiation methods
   - Computationally expensive for high-dimensional inputs
2. Symbolic differentiation
   - Requires systematic knowledge of derivatives
   - Can be computationally expensive if poorly implemented
3. Automatic differentiation (reverse mode)

Approach 3: Automatic Differentiation (reverse mode)

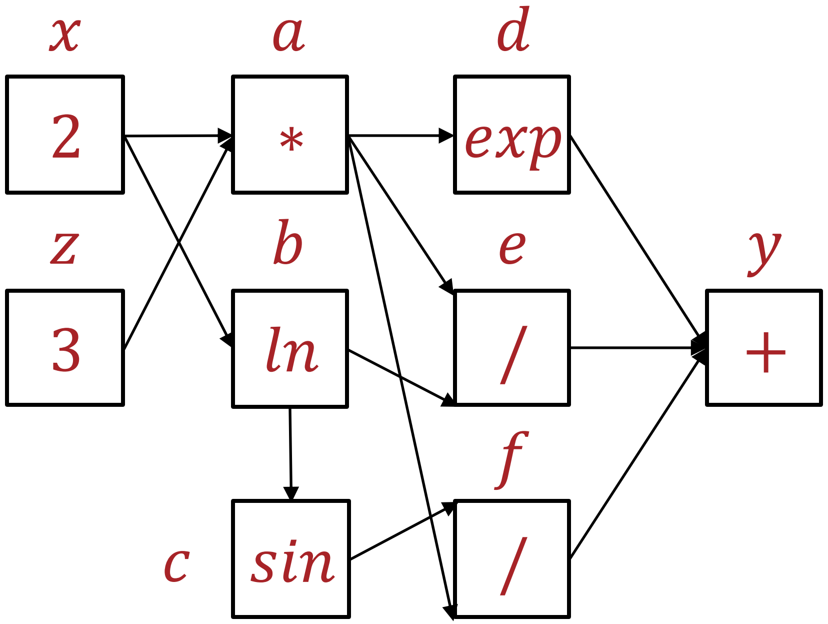
- Given

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

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

- First define some intermediate quantities, draw the computation graph and run the "forward" computation

$$a = xz$$
$$b = \ln(x)$$
$$c = \sin(b)$$
$$d = e^a$$
$$e = a/b$$
$$f = c/a$$
$$y = d + e + f$$



Example courtesy of Matt Gormley

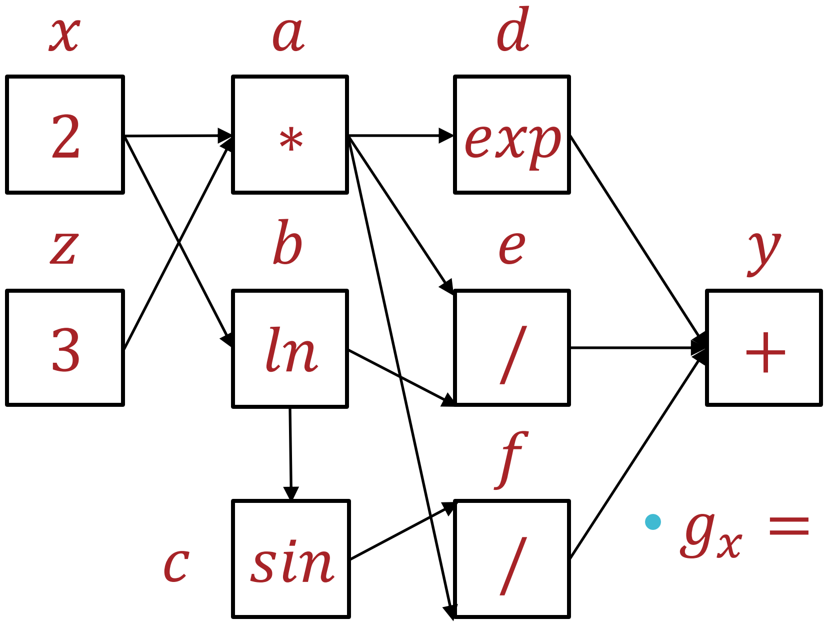
Approach 3: Automatic Differentiation (reverse mode)

- 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

Three Approaches to Differentiation

- Given $f\colon \mathbb{R}^D \rightarrow \mathbb{R}$, compute $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = {}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$

1. Finite difference method
   - Requires the ability to call $f(\boldsymbol{x})$
   - Great for checking accuracy of implementations of more complex differentiation methods
   - Computationally expensive for high-dimensional inputs
2. Symbolic differentiation
   - Requires systematic knowledge of derivatives
   - Can be computationally expensive if poorly implemented
3. Automatic differentiation (reverse mode)
   - Requires systematic knowledge of derivatives *and* an algorithm for computing $f(\boldsymbol{x})$
   - Computational cost of computing ${}^{\partial f(\boldsymbol{x})}/_{\partial \boldsymbol{x}}$ is proportional to the cost of computing $f(\boldsymbol{x})$

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

Key Takeaways

- Finite difference method is a simple but computationally expensive approximation technique
  - ***You should use this to unit test your implementation of backpropagation!***
- Symbolic differentiation is the “default” differentiation method but can also also be computationally expensive
- Automatic differentiation (reverse mode) saves intermediate quantities for computational efficiency
  - Backpropagation is an instance of automatic differentiation in the reverse mode