Neural Network Optimization

Machine Learning Problems

- **Given data:**

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $\sum_{i=1}^n \ell_i(w)$

$$\text{Logistic Loss: } \ell_i(w) = \log(1 + \exp(-y_i\, x_i^T w))$$

$$\text{Squared error Loss: } \ell_i(w) = (y_i - x_i^T w)^2$$

Machine Learning Problems

- **Given data:**

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $\sum_{i=1}^n \ell_i(w)$

$$\text{Logistic Loss: } \ell_i(w) = \log(1 + \exp(-y_i\, x_i^T w))$$

$$\text{Squared error Loss: } \ell_i(w) = (y_i - x_i^T w)^2$$

Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Big|_{w=w_t}$$

Gradient Descent

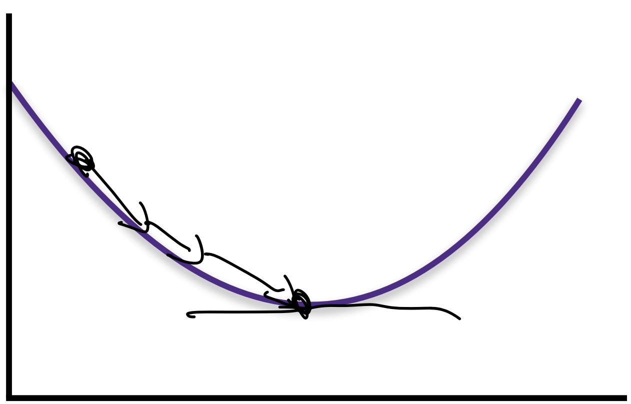
min f(X)

Initialize: $w_0 = 0$

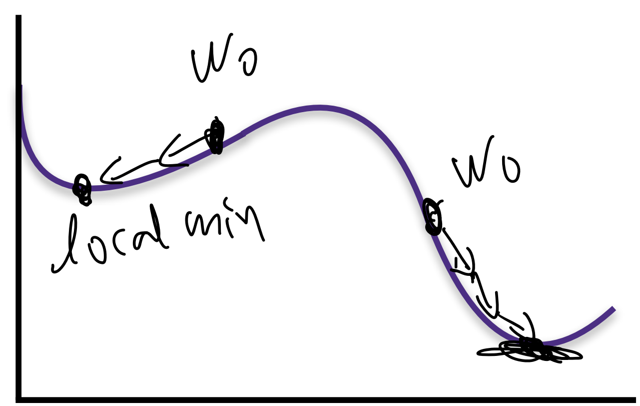
for $t = 1, 2, \ldots$

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

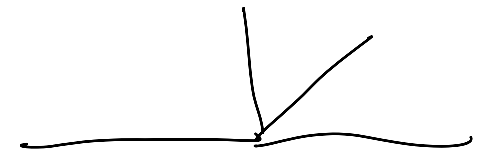
Convex Function



Non-convex Function



Sub-Gradient Descent



Initialize: $w_0 = 0$

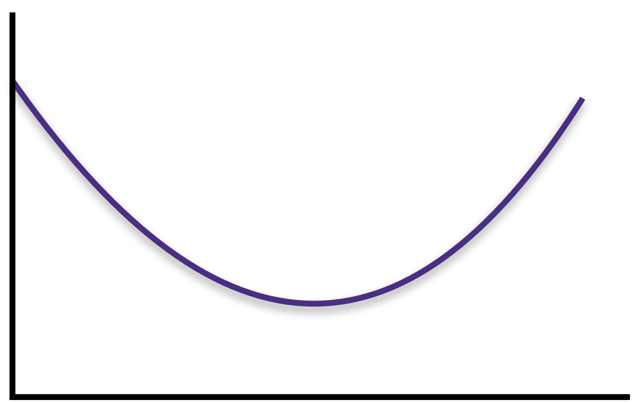
for $t = 1, 2, \ldots$

Find any g_t such that $f(y) \geq f(w_t) + g_t^\top (y - w_t)$

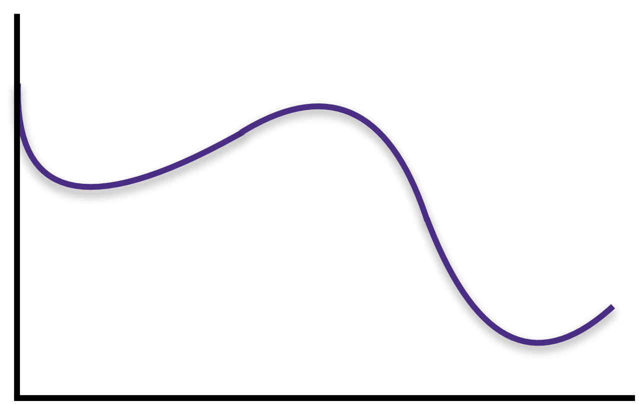
$w_{t+1} = w_t - \eta g_t$

g is a subgradient at x if $f(y) \geq f(x) + g^T(y - x)$

Convex Function



Non-convex Function



Machine Learning Problems

- **Given data:**

$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $\displaystyle\sum_{i=1}^n \ell_i(w)$

Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Big|_{w=w_t}$$

Stochastic Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \Big|_{w=w_t}$$

I_t drawn uniform at random from $\{1, \ldots, n\}$

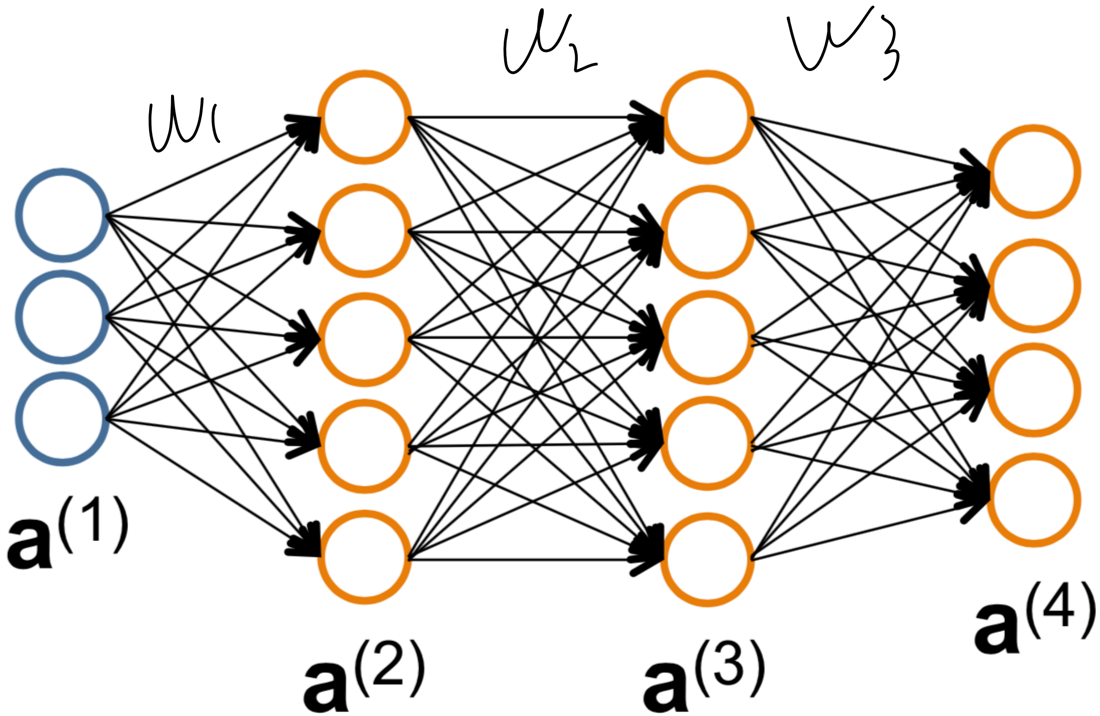
Mini-batch SGD

Instead of one iterate, average B stochastic gradient together

Advantages:

- de-noises gradient
- Matrix computations
- Parallelization

Gradient Computation on a Graph



Naive computation: node by node

L layer, gradient of every parameter $O(L)$

$\Rightarrow O(L^2)$ time

A brief history

- **Back propagation:** the workhorse for training neural networks. An algorithm that for a network with V nodes and E edges calculates that gradient in **linear time** O(V+E).

  vs. naive $O(V^2)$

- The name was introduced by Rumelhart, Hinton, Williams ’86. Same idea was rediscovered multiple times. Also mentioned in Werbos’ thesis ’74 in the context of neural networks.

- **Control theory:** Kelly ’60, Bryson ’61 [**dynamic programming**]

- **Theoretical computer science:** Baur-Strassen lemma ’83 [**algebraic circuits**]

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

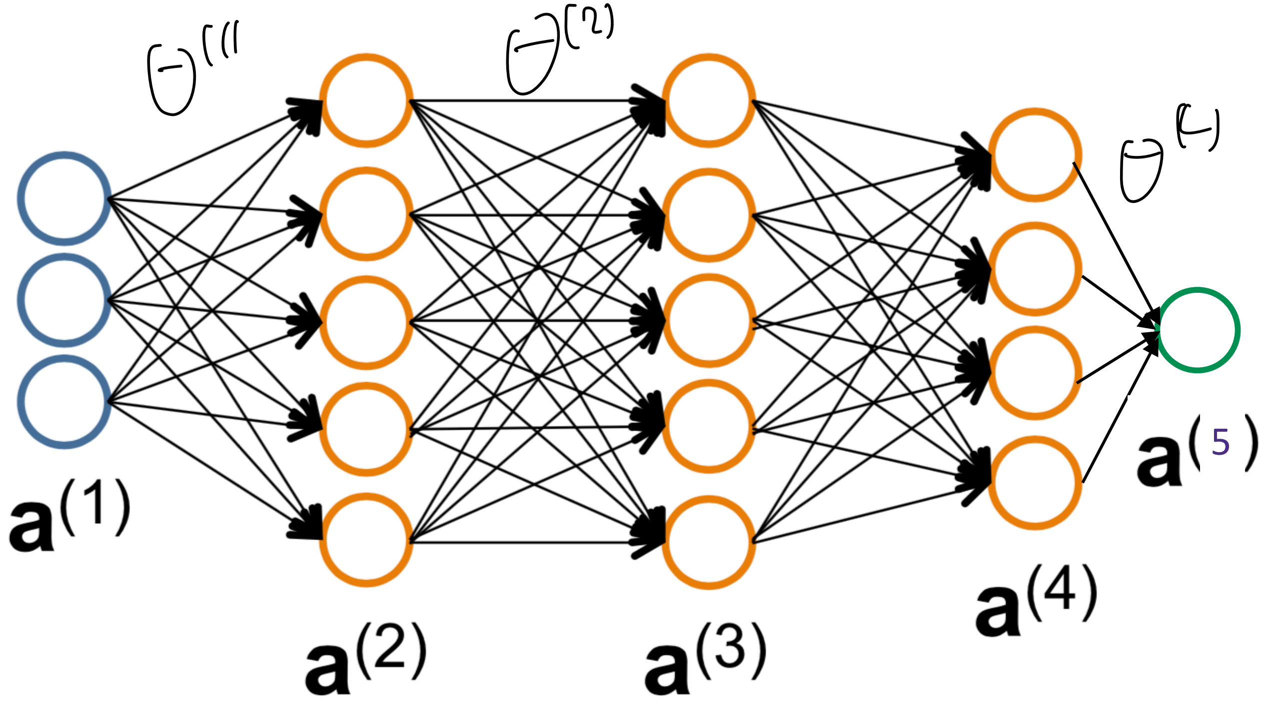
$$\vdots$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \widehat{y}) \quad \forall l$

Forward Propagation

L: # of layers
g: activation function
$z^{(l)}$: pre-activation

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

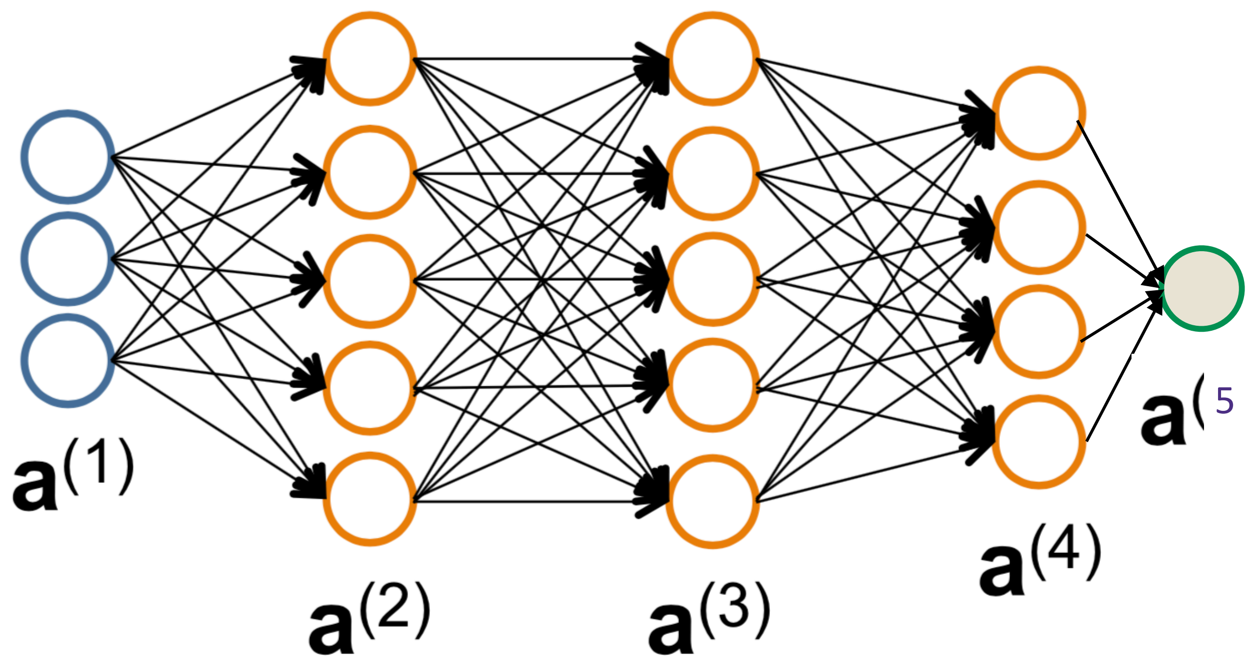
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Backprop

$x \in \mathbb{R}^d$

$\Theta^{(1)}, \ldots, \Theta^{(L)}$: parameters to train

$\Theta^{(1)}: \mathbb{R}^{m \times d}, \Theta^{(l)}: \ldots \Theta^{(L-1)} \in \mathbb{R}^{m \times m}, \Theta^{(L)}: \mathbb{R}^m$

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

Train by Stochastic Gradient Descent:

$\Theta^{(l)}_{i,j} \in \mathbb{R}$

$$\Theta^{(l)}_{i,j} \leftarrow \Theta^{(l)}_{i,j} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

Chain Rule $\quad z_i^{(l+1)} = \sum_{j=1}^{m} \Theta_{ij}^{(l)} \cdot a_j^{(l)}$

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

chain rule

$$\delta_i^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \underbrace{\frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}}}_{\delta_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$z_k^{(l+1)} = \sum_{u=1}^{m} \Theta^{(l)}_{ku} \cdot g(z_u^{(l)}) \Rightarrow \Theta^{(l)}_{ki} g'(z_i^{(l)})$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y,\hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y,\hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$= \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}\, g'(z_i^{(l)})$$

when logistic

$$= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}$$

$$L(y,\hat{y}) = y \log(\hat{y}) + (1-y)\log(1-\hat{y})$$

$$g(z) = \frac{1}{1+e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y,\hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g\left(z^{(l)}\right)$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}$$

$$\delta^{(L+1)} \to \delta^{(L)} \to \cdots \to \delta^{(1)}$$

$$\Theta^{(L)} \to \cdots \to \Theta^{(1)}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}$$

$$\delta_i^{(L+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} \left[y \log(g(z^{(L+1)})) + (1-y)\log(1 - g(z^{(L+1)})) \right]$$

$$= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1-y}{1 - g(z^{(L+1)})} g'(z^{(L+1)})$$

$$= y - g(z^{(L+1)}) = y - a^{(L+1)}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1-y)\log(1-\hat{y})$$

$$g(z) = \frac{1}{1+e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g\left(z^{(l)}\right)$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\delta^{(L+1)} = y - a^{(L+1)}$$

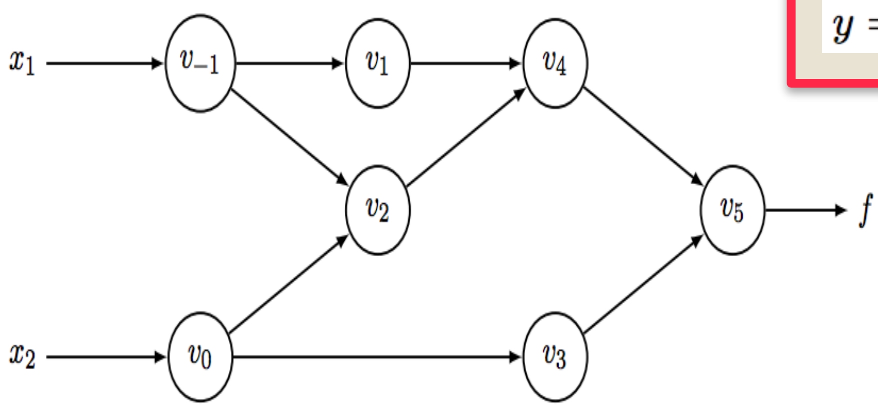
Recursive Algorithm!

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Auto-differentiation

Backprop for this simple network architecture is a special case of ***reverse-mode auto-differentiation:***



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$

Forward Primal Trace

$v_{-1} = x_1$		$= 2$
$v_0 = x_2$		$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

Reverse Adjoint (Derivative) Trace

$\bar{x}_1 = \bar{v}_{-1}$			$= \mathbf{5.5}$
$\bar{x}_2 = \bar{v}_0$			$= \mathbf{1.716}$
$\bar{v}_{-1}$	$= \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$\bar{v}_0$	$= \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
$\bar{v}_{-1}$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
$\bar{v}_2$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_1$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_3$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$
$\bar{v}_4$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$
$\bar{v}_5$	$= \bar{y}$	$= 1$	

Auto-differentiation

- Given a function, computes its partial derivatives
- Compute all of the partial derivatives of a function with (nearly) same computation runtime [Griewank ’89, Baur and Strassen ’83]
- Backbone of (applied) machine learning: Pytorch, Tensorflow, …

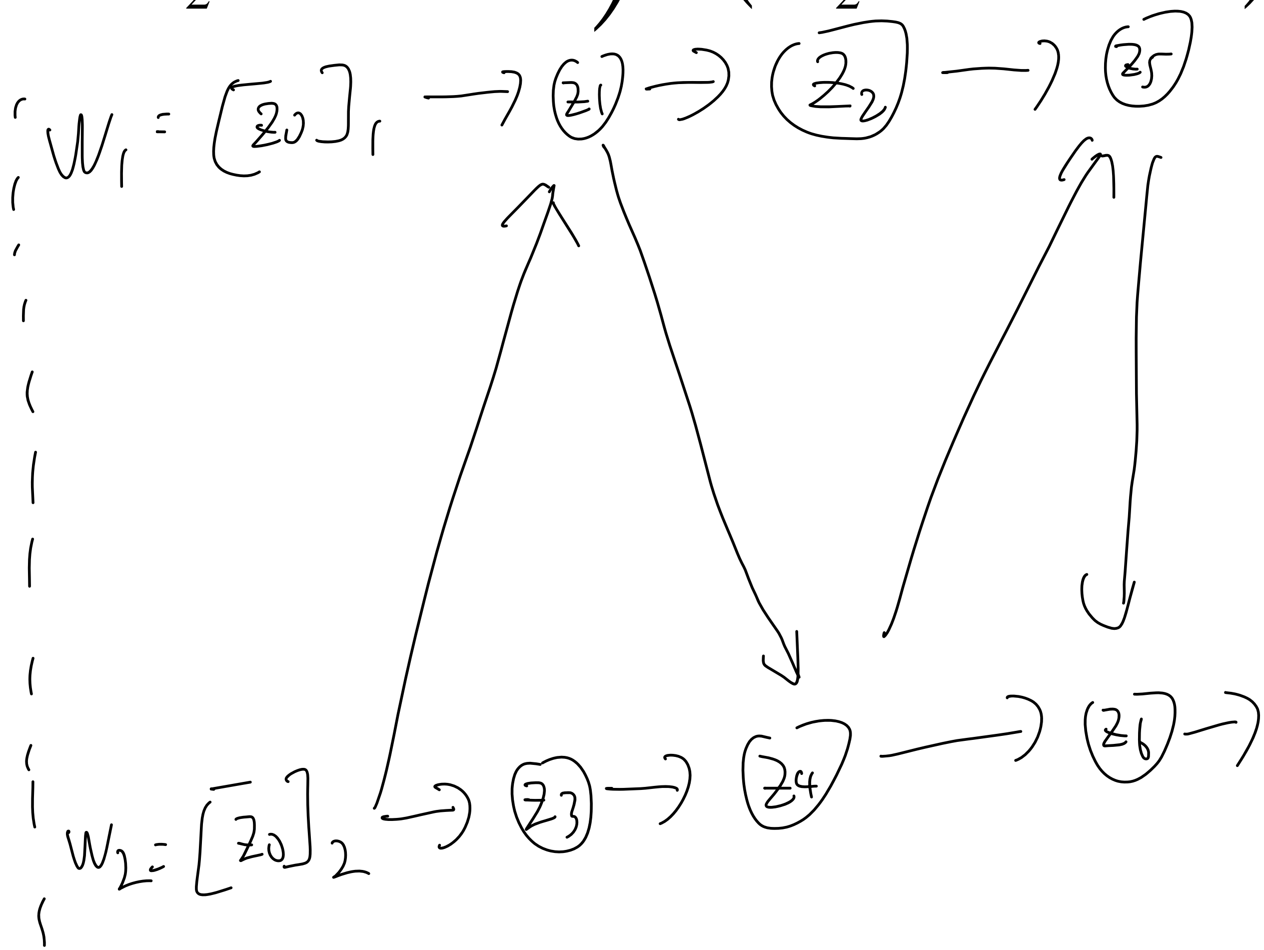
Example of Computation Graph

$$f(w_1, w_2) = \left(\sin\left(\frac{2\pi w_1}{w_2}\right) + \frac{3w_1}{w_2} - \exp(2w_2) \right) \cdot \left(\frac{3w_1}{w_2} - \exp(2w_2) \right)$$

Input: $z_0 = (w_1, w_2)$

1. $z_1 = \frac{w_1}{w_2}$
2. $z_2 = \sin(2\pi \cdot z_1)$
3. $z_3 = \exp(2w_2)$
4. $z_4 = 3z_1 - z_3$
5. $z_5 = z_2 + z_4$
6. $z_6 = z_4 \cdot z_5$

Return z_6



Computation Model

O(1)

exp, sin

- Given access to a set of differentiable real functions $h \in \mathscr{H}$
- Use functions in $\mathscr{H}$ to create intermediate variables.
- Evaluation trace:
  - All intermediate variables will be scalars; each corresponds to a node.
  - Input $z_0 = w \in \mathbb{R}^d$. $[z_0]_1 = w_1, [z_0]_2 = w_2, \ldots, [z_0]_d = w_d$
  - Step 1: $z_1 = h_1$ (a subset of variables in w)
  - ….
  - Step t: $z_t = h_t$ (a subset of variables in $z_1, \ldots, z_{t-1}, w$)
  - …
  - Step T: $z_T = h_T$ (a subset of variables in $z_1, \ldots, z_{T-1}, w$)
  - **Return:** z_T

($h_1, \ldots, h_T \in \mathscr{H}$ **)**

Computation Model

- Every $h \in \mathscr{H}$ is one of the following:
  - Type 1: An affine transformation of the inputs

    $z_4 = 3z_1 - z_3$, $z_2 + z_5$ $\quad O(1)$

    $\frac{\partial z_4}{\partial z_3} = -1$

  - Type 2: A product of variables, to some power

    $z_1 \cdot z_2$, z_4^4 z_5^6

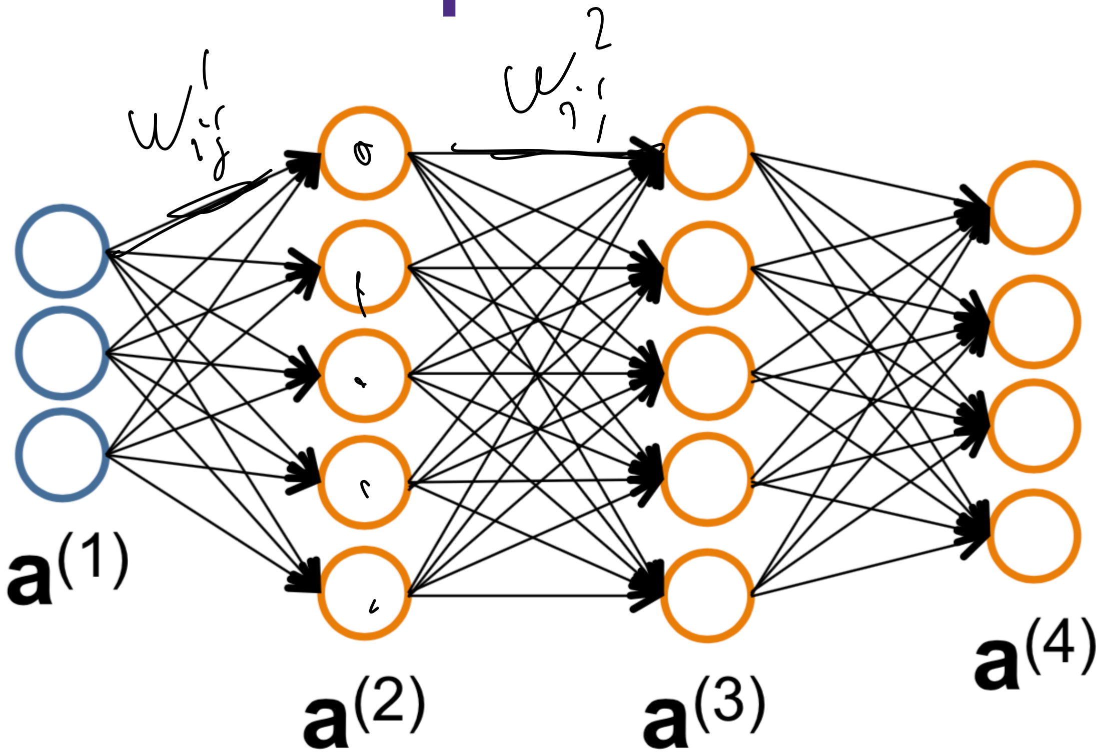
  - Type 3: A fixed set of one dimensional differentiable functions: $\sin(\cdot), \cos(\cdot), \exp(\cdot), \log(\cdot), \ldots$

    ReLU(z): if $z > 0 \Rightarrow 1$, o.w. $\Rightarrow 0$

    - We assume we can easily compute the derivatives for each of this functions. $O(1)$

    - Type 3 can be approximated by Type 1 and Type 2, using polynomials.

Neural Network Example

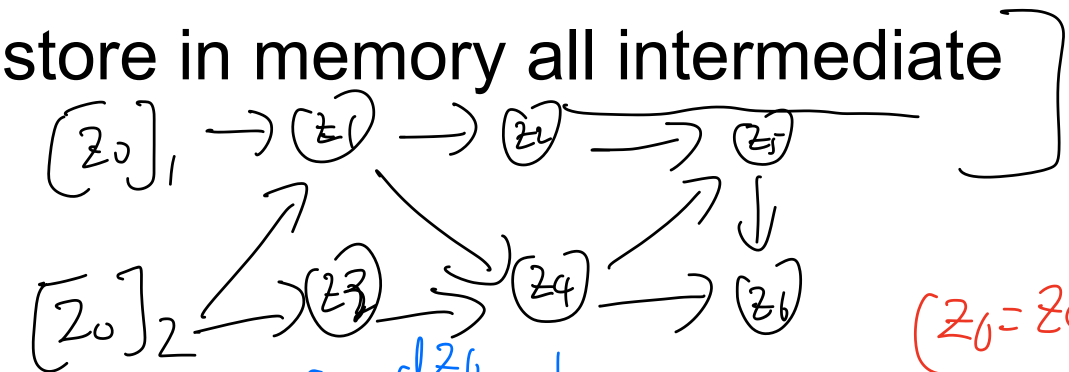


Reverse Mode of Automatic Differentiation

Goal: Compute partial derivatives of $f(w)$, i.e., df/dw.

- Step 1: computer $f(w)$ and store in memory all intermediate variables $z_1, \ldots, z_T$
- Step 2: Initialize: $\dfrac{dz_T}{dz_T} = 1.$
- Step 3: For $t = T, T-1, \ldots, 0$
  - $$\frac{dz_T}{dz_t} = \sum_{\text{c is a child of t}} \frac{dz_T}{dz_c} \cdot \frac{\partial z_c}{\partial z_t}$$

  (Child: a node z_t directly points to)
- Step 4: Return $\dfrac{dz_T}{dz_0} = \dfrac{df}{dw}$



$(z_6 = z_4 \cdot z_5)$

(1) $\dfrac{dz_6}{dz_6} = 1$

(2) $\dfrac{dz_6}{dz_5} = \dfrac{dz_6}{dz_6} \cdot \dfrac{\partial z_6}{\partial z_5} = z_4$

(3) $\dfrac{dz_6}{dz_4} = \dfrac{dz_6}{dz_6} \cdot \dfrac{\partial z_6}{\partial z_4} + \dfrac{dz_6}{dz_5} \cdot \dfrac{\partial z_5}{\partial z_4}$

(4) $\dfrac{dz_6}{dz_3} = \dfrac{dz_6}{dz_4} \cdot \dfrac{\partial z_4}{\partial z_3}$

(5) $\dfrac{dz_6}{dz_2} = \dfrac{dz_6}{dz_5} \cdot \dfrac{\partial z_5}{\partial z_2}$

(6) $\dfrac{dz_6}{dz_1} = \dfrac{dz_6}{dz_4} \cdot \dfrac{\partial z_4}{\partial z_1} + \dfrac{dz_6}{dz_2} \cdot \dfrac{\partial z_2}{\partial z_1}$

Time Complexity

Theorem (Baur and Strassen '83, Griewak '89): Assume every h is specified as in our computational model. For $h(\cdot)$ of type 3, assume we can compute the derivative $h'(z)$ in time as the same order of computing $h(z)$. Let T denote the time to compute $f(w)$. Then the reverse mode computes df/dw in time $O(T)$.

① Time complexity: count edges

② Correctness: $\frac{dz_T}{dz_c}$ already computed, z_c is a child of z_t

⟹ we can compute $\frac{dz_T}{dz_t}$

(1) $\frac{\partial z_c}{\partial z_t}$ can be computed

- type 1 ⟹ coefficient
- type 2 ⟹ $\frac{\partial z_c}{\partial z_t} = \frac{z_c}{z_t} \cdot \alpha$
- type 3, $\frac{\partial z_c}{\partial z_t} = h'(z_t)$

example: $z_5 = z_1 \cdot z_4^2$

$\frac{\partial z_5}{\partial z_4} = 2 \cdot z_4$

Time Complexity

Clarke Differential

Subdifferential and Subgradient

Definition: Given $f : \mathbb{R}^d \to \mathbb{R}$, for every x, the subdifferential set is defined as $\partial_s f(x) \triangleq \{s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^\top (x' - x)\}$. The elements in the subdifferential set are subgradients.

Subdifferential and Subgradient

Definition: Given $f : \mathbb{R}^d \to \mathbb{R}$, for every x, the subdifferential set is defined as $\partial_s f(x) \triangleq \{s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^\top (x' - x)\}$. The elements in the subdifferential set are subgradients.

Subdifferential is not enough

Definition: Given $f : \mathbb{R}^d \to \mathbb{R}$, for every x, the subdifferential set is defined as $\partial_s f(x) \triangleq \{s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^\top (x' - x)\}$. The elements in the subdifferential set are subgradients.

Clarke Differential

Definition: Given $f : \mathbb{R}^d \to \mathbb{R}$, for every x, the Clark differential is defined as

$$\partial f(x) \triangleq \text{conv}\left(\{s \in \mathbb{R}^d : \exists \{x_i\}_{i=1}^{\infty} \to x, \{\nabla f(x_i)\}_{i=1}^{\infty} \to s\}\right).$$

The elements in the subdifferential set are subgradients.

When does Clarke differential exists

Definition (Locally Lipschitz): $f : \mathbb{R}^d \to \mathbb{R}$ is locally Lipchitz if $\forall x \in \mathbb{R}^d$, there exists a neighborhood S of x, such that f is Lipchitz in S.