

Neural Network Optimization



Machine Learning Problems

- **Given data:**

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

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

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

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

Machine Learning Problems

- Given data:

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

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

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

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

Gradient Descent:

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

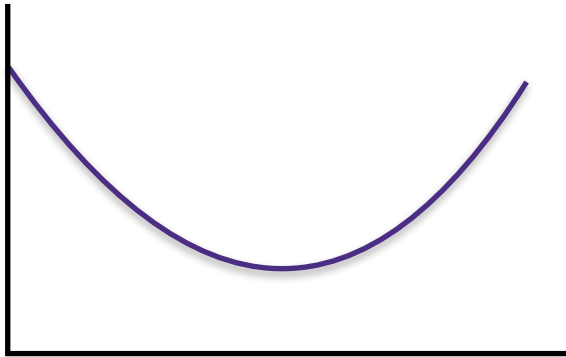
Gradient Descent

Initialize: $w_0 = 0$

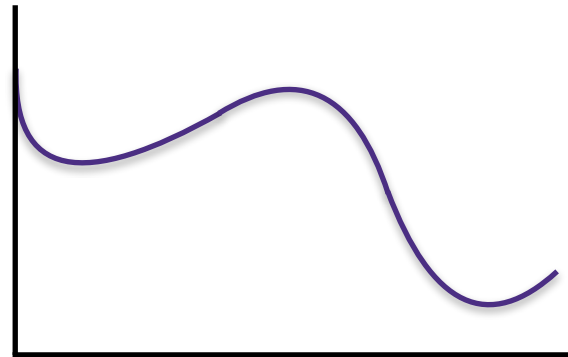
for $t = 1, 2, \dots$

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

Convex Function



Non-convex Function



Sub-Gradient Descent

Initialize: $w_0 = 0$

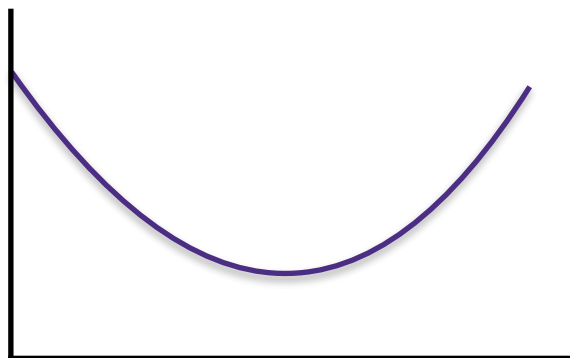
for $t = 1, 2, \dots$

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

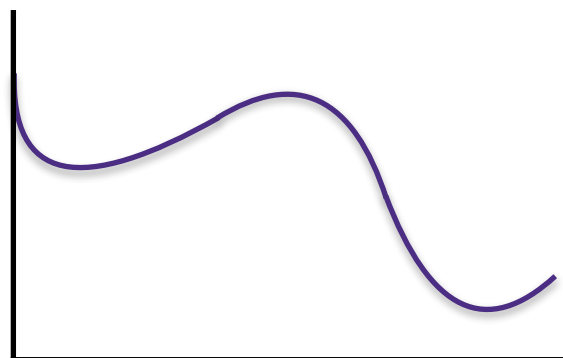
$$w_{t+1} = w_t - \eta g_t$$

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

Convex Function



Non-convex Function



Machine Learning Problems

- Given data:

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

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

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:

$$w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \Big|_{w=w_t} \quad I_t \text{ drawn uniform at random from } \{1, \dots, n\}$$

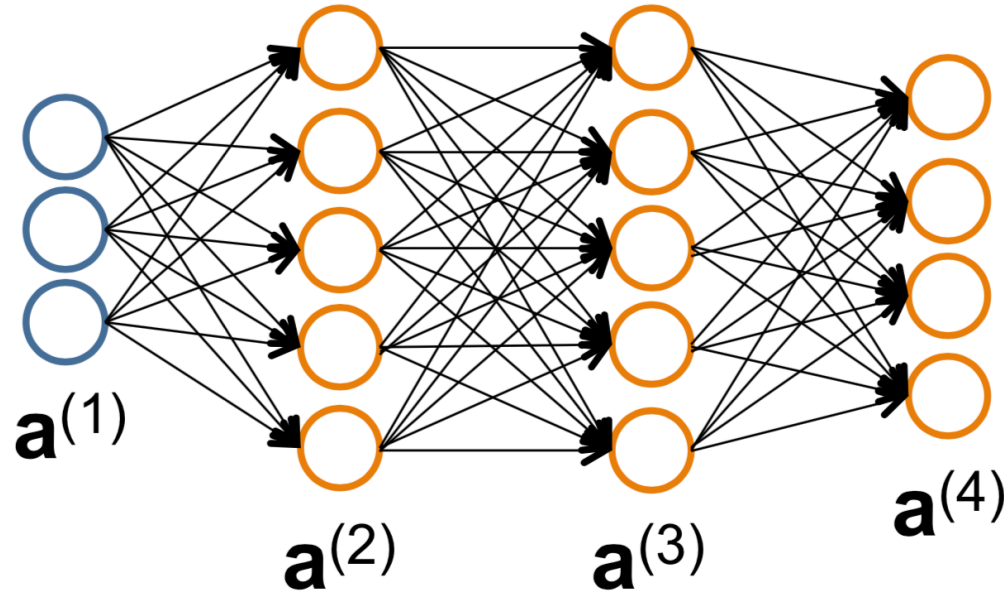
Mini-batch SGD

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

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** $O(V+E)$.
- 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**]

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

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

$$a^{(2)} = g(z^{(2)})$$

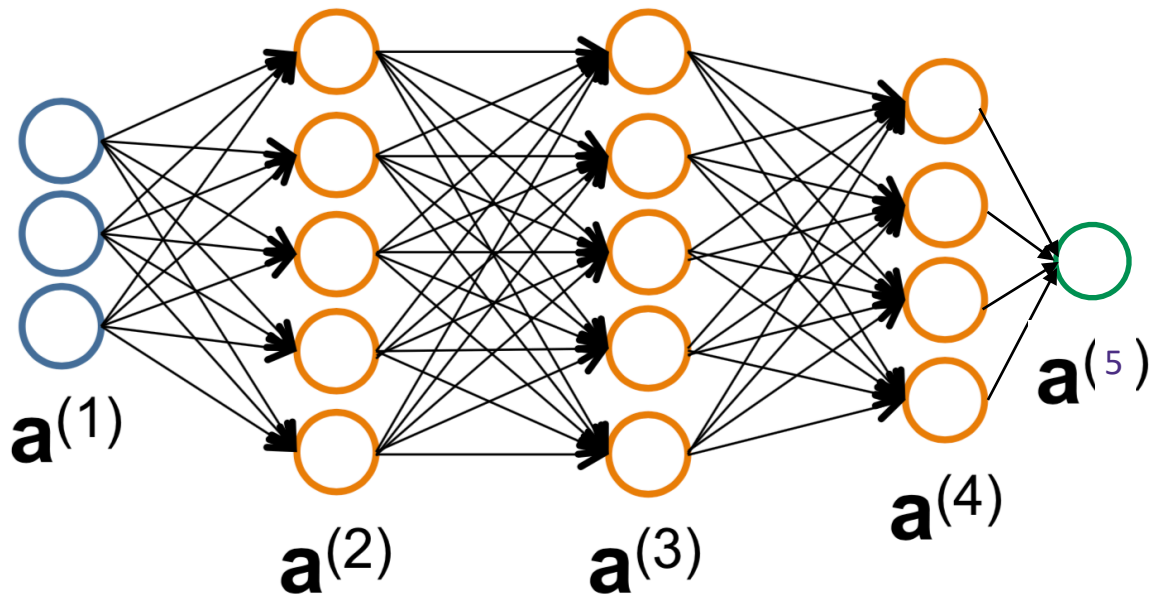
$$\vdots$$

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

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

$$\vdots$$

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



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

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

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

Forward Propagation

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

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

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$

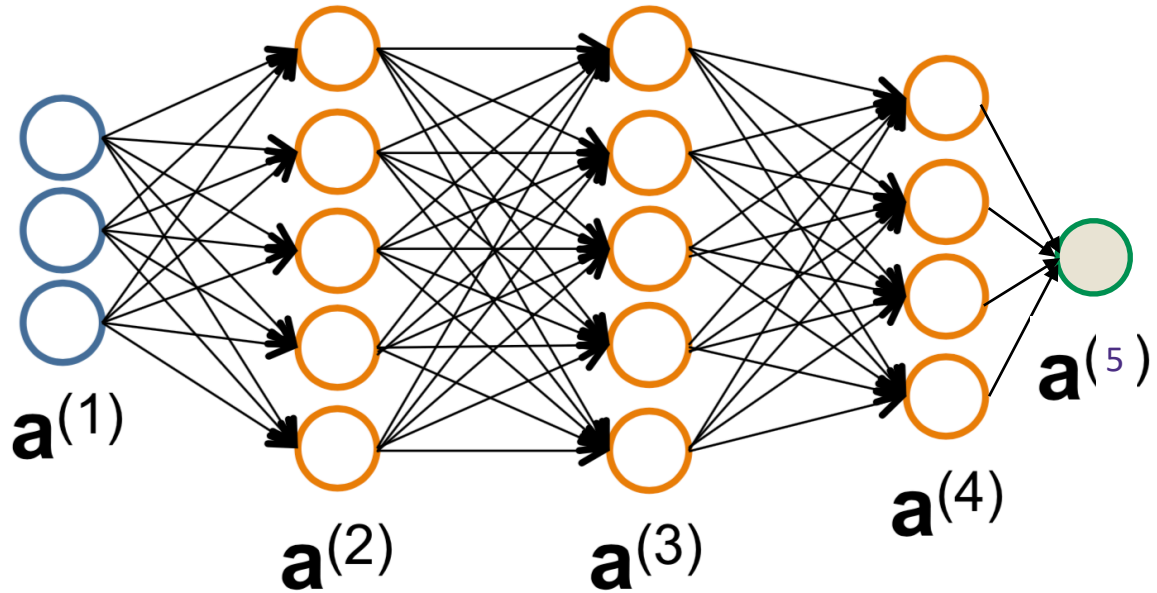
$$a^{(l)} = g(z^{(l)})$$

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

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

$$\vdots$$

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



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

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

Backprop

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

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

$$a^{(2)} = g(z^{(2)})$$

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

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

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

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

Train by Stochastic Gradient Descent:

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

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

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

Backprop

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

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

$$a^{(2)} = g(z^{(2)})$$

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

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

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

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

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

Train by Stochastic Gradient Descent:

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

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

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

Backprop

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

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

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$

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

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

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

$$\vdots$$

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

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

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

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

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

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

Backprop

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

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

$$a^{(2)} = g(z^{(2)})$$

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

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

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

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

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

$$\begin{aligned} \delta_i^{(l)} &= \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \\ &= \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} g'(z_i^{(l)}) \\ &= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} \end{aligned}$$

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

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

Backprop

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

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

$$a^{(2)} = g(z^{(2)})$$

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

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

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

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

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

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

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

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

Backprop

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

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

$$a^{(2)} = g(z^{(2)})$$

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

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

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

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

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

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

$$\begin{aligned} \delta_i^{(L+1)} &= \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} [y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)}))] \\ &= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)}) \\ &= y - g(z^{(L+1)}) = y - a^{(L+1)} \end{aligned}$$

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

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

Backprop

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

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

$$a^{(2)} = g(z^{(2)})$$

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

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

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

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

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

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

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

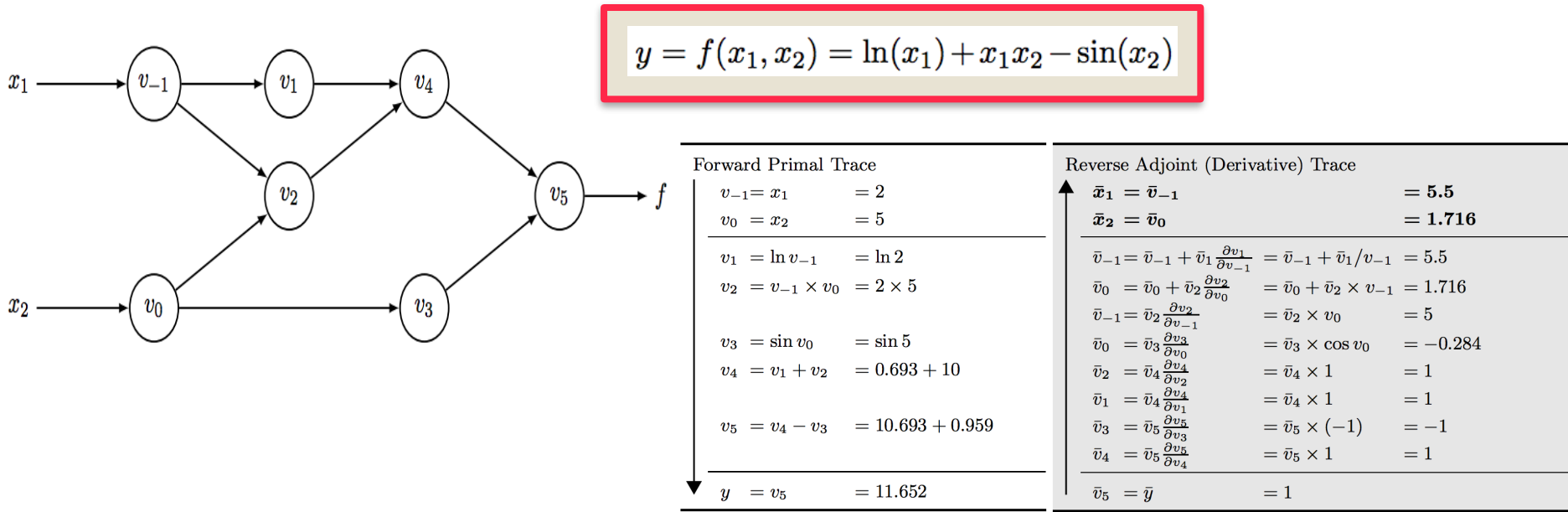
Recursive Algorithm!

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

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

Auto-differentiation

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



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: $z_0 = (w_1, w_2)$

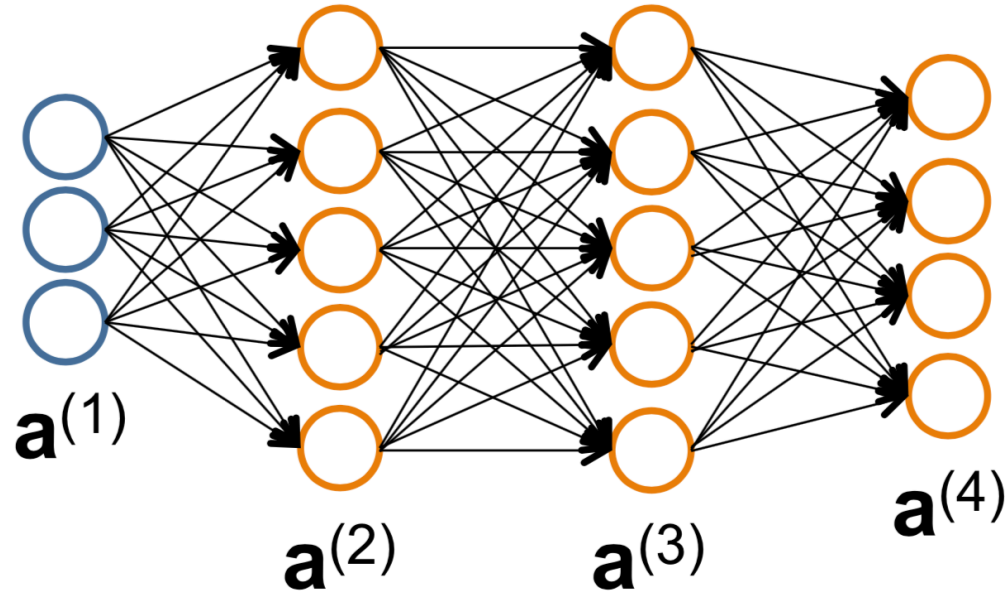
Computation Model

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

Computation Model

- Every $h \in \mathcal{H}$ is one of the following:
 - Type 1: An affine transformation of the inputs
 - Type 2: A product of variables, to some power
 - Type 3: A fixed set of one dimensional differentiable functions: $\sin(\cdot)$, $\cos(\cdot)$, $\exp(\cdot)$, $\log(\cdot)$, $\dots$
 - We assume we can easily compute the derivatives for each of this functions.
 - 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: compute $f(w)$ and store in memory all intermediate variables $z_1, \dots, z_T$

- Step 2: Initialize: $\frac{dz_T}{dz_T} = 1$.

- Step 3: For $t = T, T-1, \dots, 0$

- $$\frac{dz_T}{dz_t} = \sum_{\text{c is a child of t}} \frac{dz_T}{dz_c} \cdot \frac{\partial z_c}{\partial z_t}$$

(Child: a node z_t directly points to)

- Step 4: Return $\frac{dz_T}{dz_0} = \frac{df}{dw}$

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 $O(T)$.

Time Complexity
