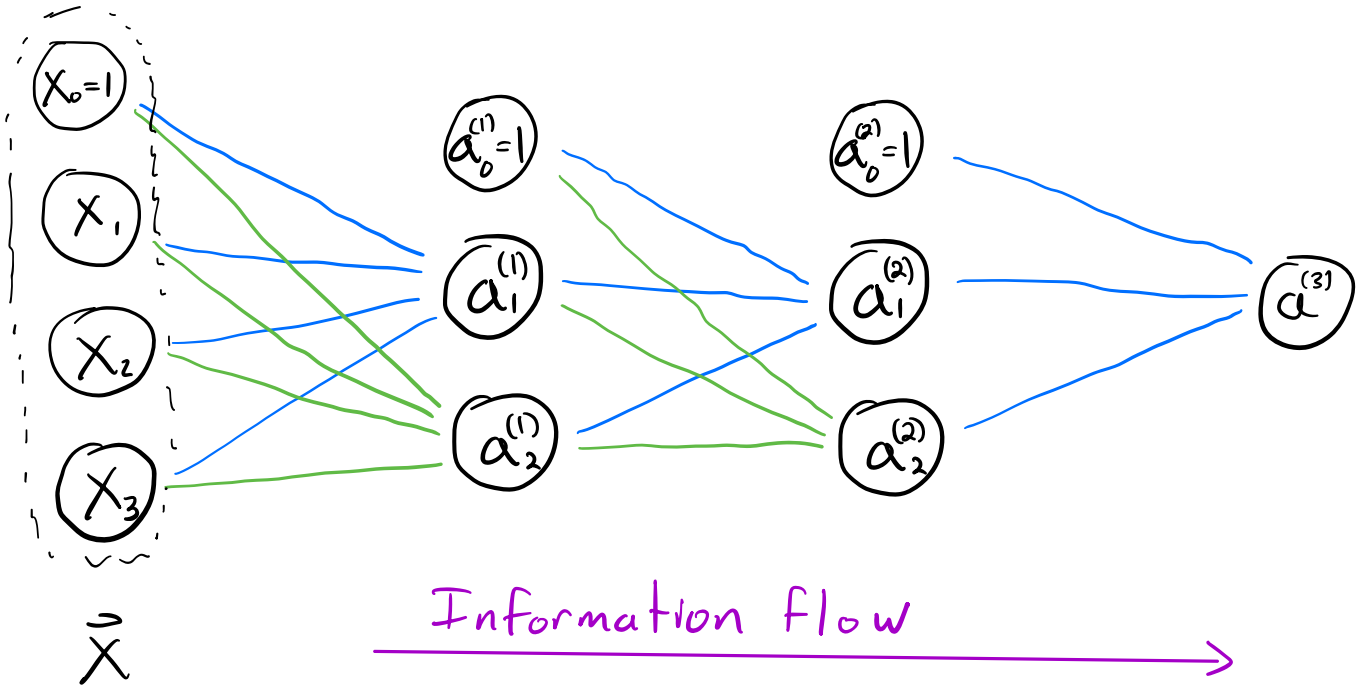
Neural Networks (NN)

A NN is a function $f(\vec{x})$

Ex: $f(\vec{x}) = a^{(3)}$ $\quad d = 3$



ERM with NNs (High Level)

$D = ((\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n))$

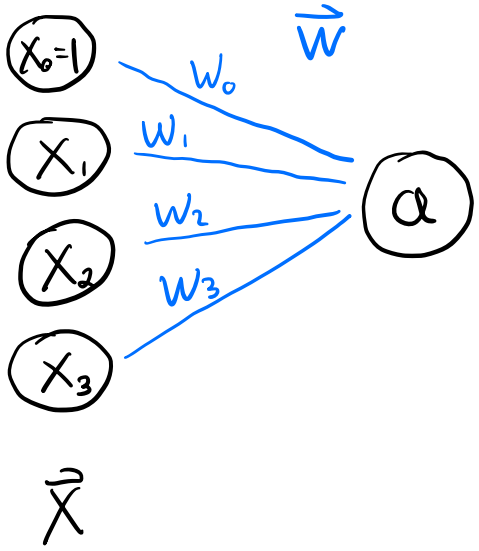
$A(D) = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \hat{L}(f)$

$\mathcal{F} = \{f \mid f: \mathcal{X} \to \mathcal{Y}$ and f is a NN with a specific architecture$\}$

every $f \in \mathcal{F}$ is defined by B weight matrices $W^{(1)}, \ldots, W^{(B)}$

Previously Seen functions as NNs

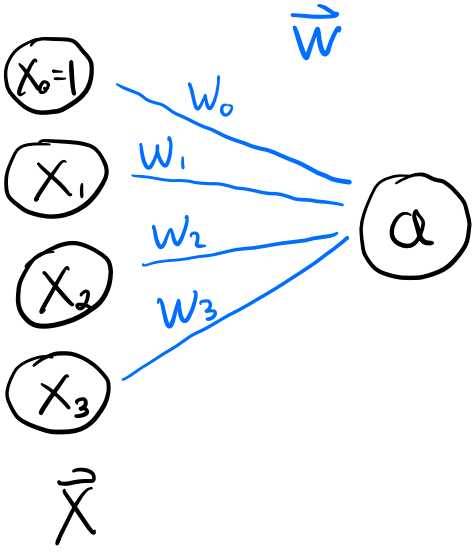
Linear: $f(\vec{x}) = \vec{x}^T \vec{w}$, $d = 3$



$$f(\vec{x}) = a = \vec{x}^T \vec{w}$$
$$= w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$$
$$= h(\vec{x}^T \vec{w})$$

$h(z) = z$

Sigmoid: $f(\vec{x}) = \sigma(\vec{x}^T \vec{w})$

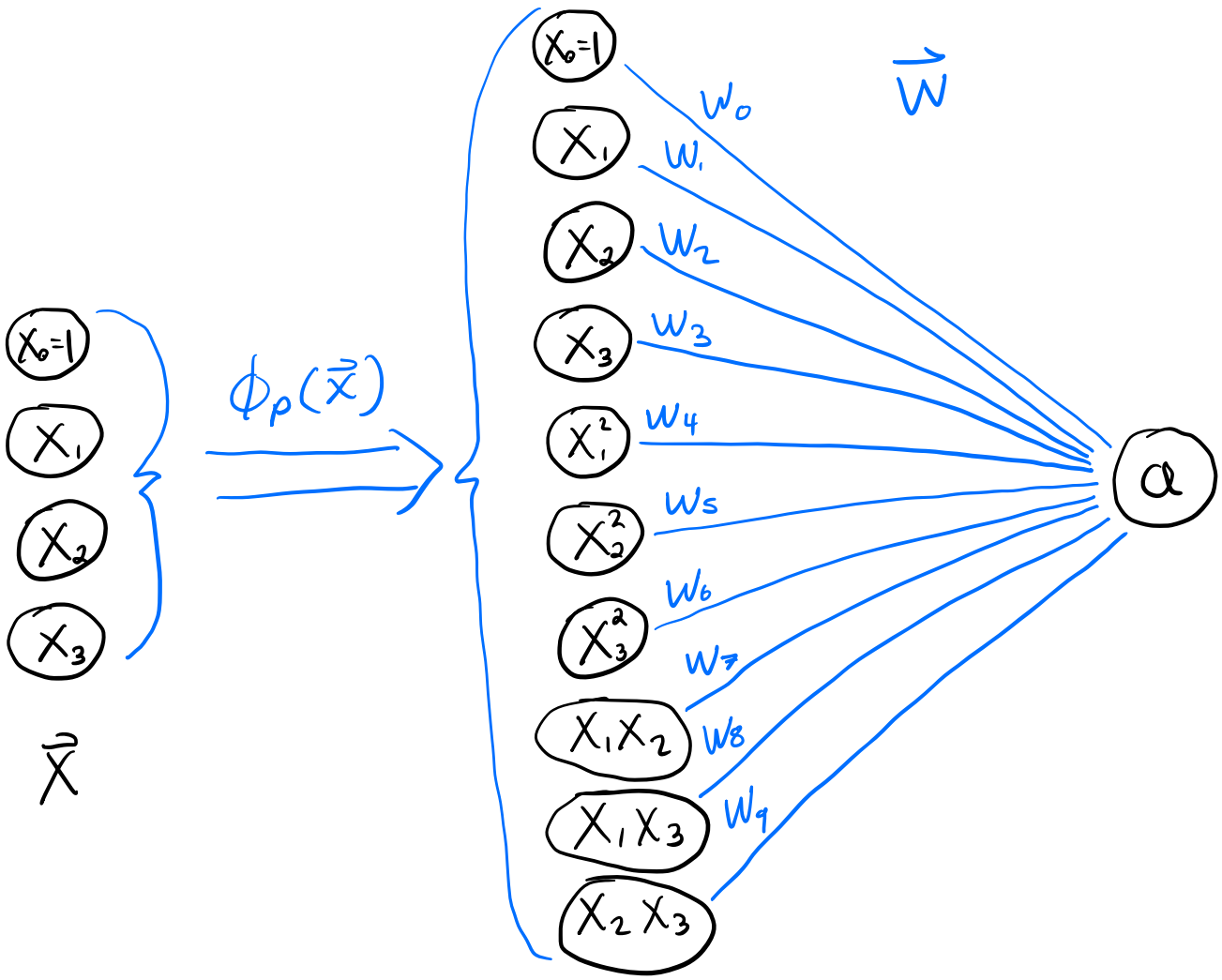


$$f(\vec{x}) = a = \sigma(\vec{x}^T \vec{w})$$
$$= \sigma(w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3)$$
$$= h(\vec{x}^T \vec{w})$$

$h = \sigma$

Linear (with polynomial features):

$$f(\vec{x}) = \phi_p(\vec{x})^T \vec{w} = a \qquad , d = 3, p = 2$$



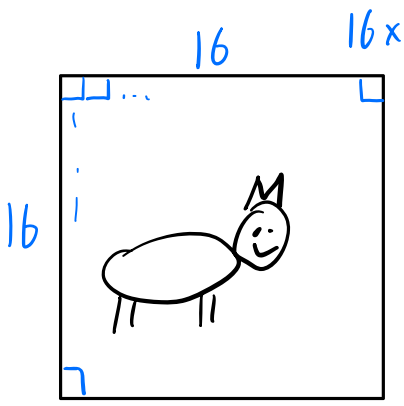
Many of the new features in $\phi_p(x)$ may not useful

When p is large this is very wasteful

Ex: Binary Classification (Cat or No Cat)

$y = \{0, 1\}$ $\quad x = \mathbb{R}^{256+1}$

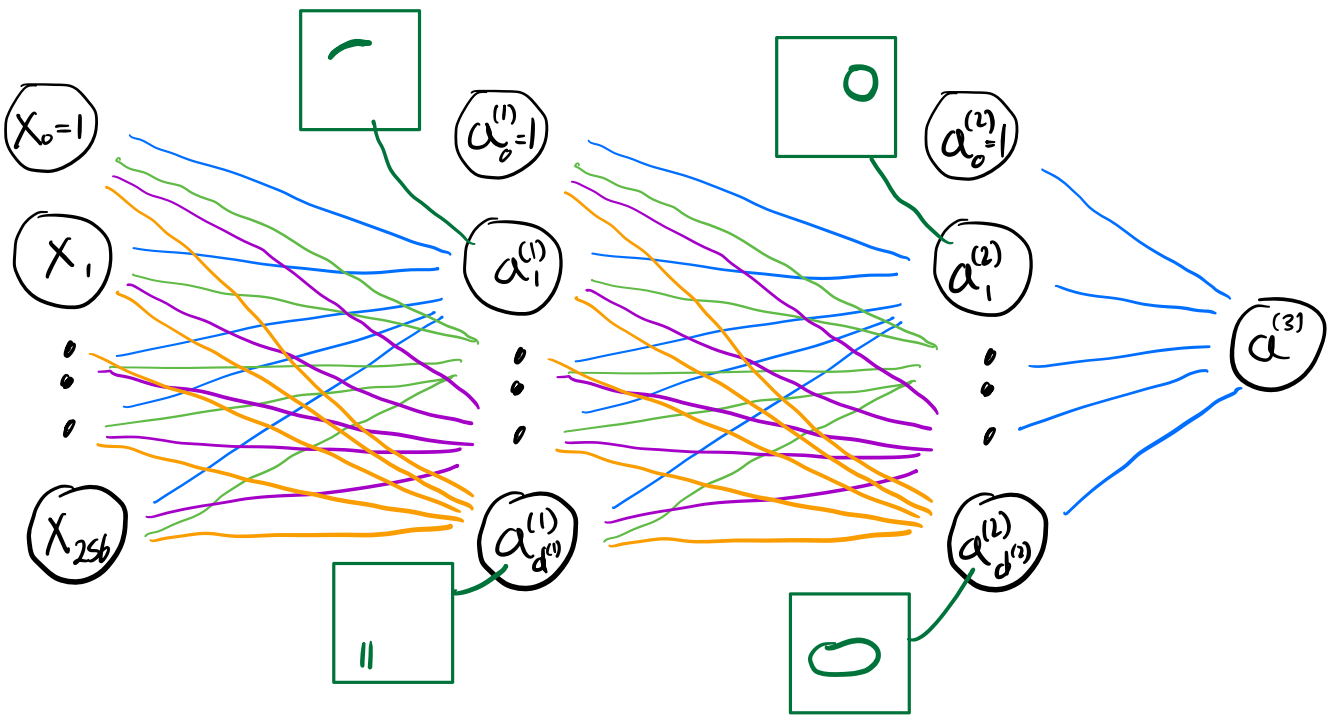
0 → No cat, 1 → Cat



$16 \times 16 = 256 = d$

$$\begin{pmatrix} x_0 = 1 \\ x_1 \\ \vdots \\ x_{256} \end{pmatrix} = \vec{x}$$

$f(\vec{x}) = a^{(3)}$ probability of a cat

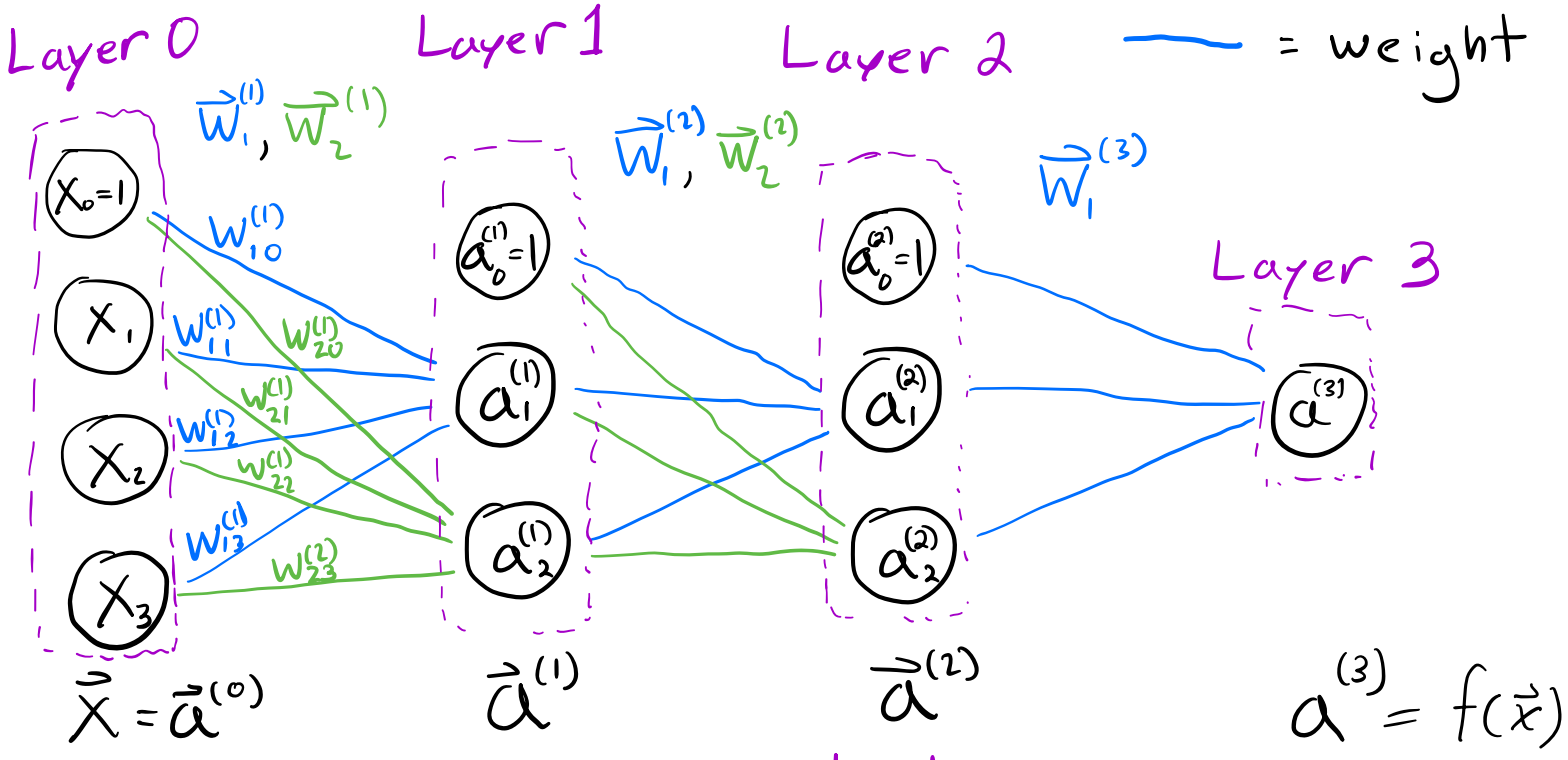


NN Definition: Specific Example

$d = 3$

◯ = Neuron

— = weight



activations

$$\vec{a}^{(1)} = (a_0^{(1)} = 1, a_1^{(1)}, a_2^{(1)})$$

$$a_1^{(1)} = h^{(1)}(z_1^{(1)}), \quad a_2^{(1)} = h^{(1)}(z_2^{(1)})$$

$h^{(1)}$: activation function Ex: Sigmoid, ReLU

and $z_1^{(1)} = \vec{x}^T \vec{w}_1^{(1)}$ $\quad z_2^{(1)} = \vec{x}^T \vec{w}_2^{(1)}$

$z_1^{(1)}, z_2^{(1)}$: Pre-Activations

$\vec{x} = \vec{a}^{(0)}$

$\vec{a}^{(0)} = \mathbb{R}^{d+1}, \ \vec{w}_1^{(1)}, \vec{w}_2^{(1)} \in \mathbb{R}^{d+1}, \ \vec{a}^{(1)} \in \mathbb{R}^{2+1}$

$$\vec{a}^{(2)} = \left(a_0^{(2)} = 1, a_1^{(2)}, a_2^{(2)} \right)$$

$$a_1^{(2)} = h^{(2)}\left(z_1^{(2)} \right), \quad a_2^{(2)} = h^{(2)}\left(z_2^{(2)} \right)$$

$$z_1^{(2)} = \left(\vec{a}^{(1)} \right)^T \vec{w}_1^{(2)} \qquad z_2^{(1)} = \left(\vec{a}^{(1)} \right)^T \vec{w}_2^{(2)}$$

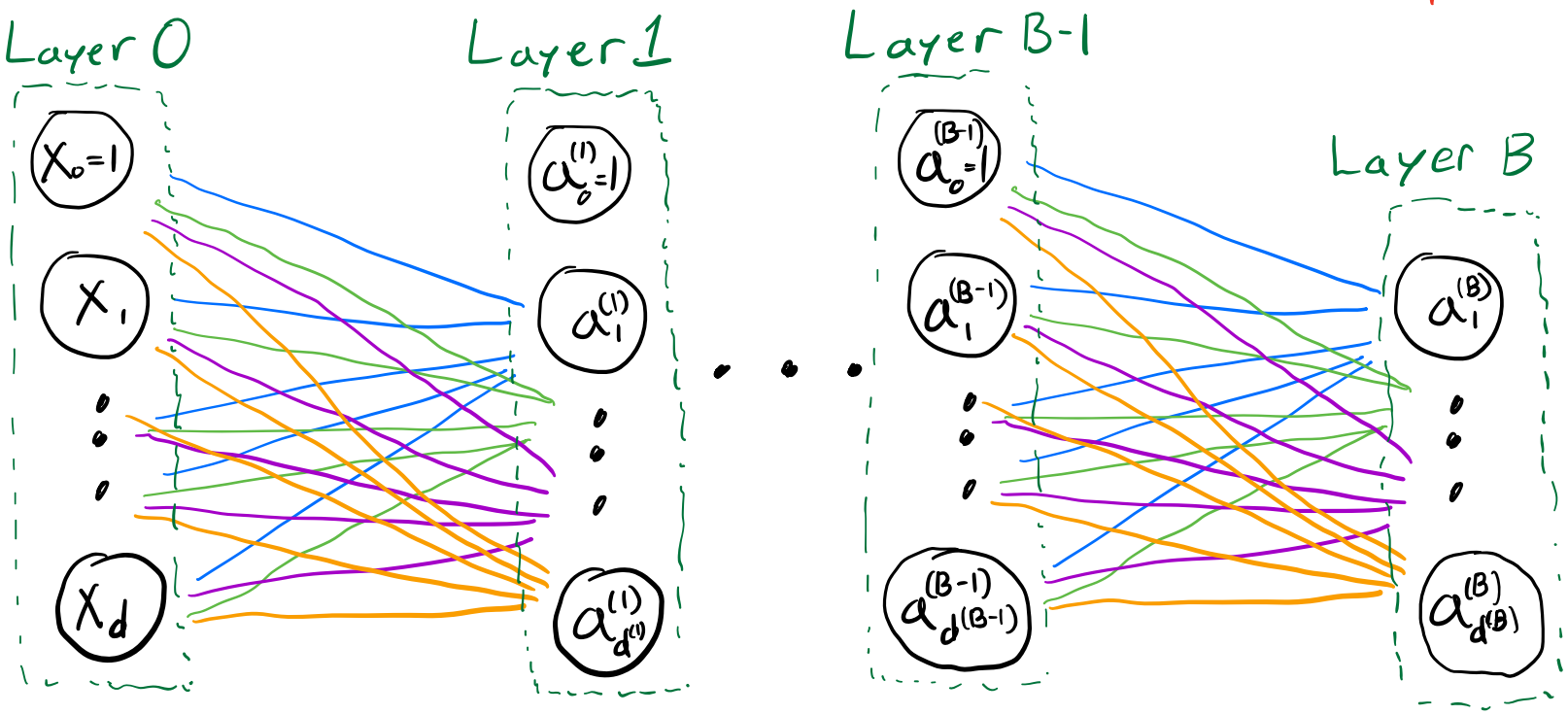
$$\vec{w}_1^{(2)}, \vec{w}_2^{(2)} \in \mathbb{R}^{2+1}, \quad \vec{a}^{(2)} \in \mathbb{R}^{2+1}$$

$$a^{(3)} = h^{(3)}\left(z^{(3)} \right), \quad z^{(3)} = \left(\vec{a}^{(2)} \right)^T \vec{w}^{(3)}$$

$$\vec{w}^{(3)} \in \mathbb{R}^{2+1}, \; a^{(3)} \in \mathbb{R}$$

In general:

No Bias in Layer B



For layer $b \in \{1, \ldots, B\}$, $d^{(b)}$ = # of neurons

Weights

$$\vec{w}_1^{(b)}, \ldots, \vec{w}_{d^{(b)}}^{(b)} \in \mathbb{R}^{d^{(b-1)}+1}$$

Pre-activations

$$\vec{z}^{(b)} = \left(z_1^{(b)}, \ldots, z_{d^{(b)}}^{(b)}\right)^T \in \mathbb{R}^{d^{(b)}}$$

$$z_j^{(b)} = \left(\vec{a}^{(b-1)}\right)^T \vec{w}_j^{(b)} \quad \text{for } j \in \{1, \ldots, d^{(b)}\}$$

Activations

$$\vec{a}^{(0)} = \vec{x} \in \mathbb{R}^{d+1} = \mathbb{R}^{d^{(0)}+1} \qquad d^{(0)} = d$$

$$\vec{a}^{(b)} = \left(a_0^{(b)} = 1, a_1^{(b)}, \ldots, a_{d^{(b)}}^{(b)}\right)^T \in \mathbb{R}^{d^{(b)}+1} \quad \text{if } b \neq B$$

$$a_j^{(b)} = h^{(b)}(z_j^{(b)}) \quad \text{for} \quad j \in \{1, \ldots, d^{(b)}\}$$

$$\vec{a}^{(B)} = (a_1^{(B)}, \ldots, a_{d^{(B)}}^{(B)})^T \in \mathbb{R}^{d^{(B)}}$$

Activation function

$h^{(b)} : \mathbb{R} \to \mathbb{R}$ usually non-linear

NN: $f(\vec{x}) = \vec{a}^{(B)}$

Defining the number of layers B and the number of neurons in each layer $d^{(1)}, \ldots, d^{(B)}$, and the activation functions:

$h^{(1)}, \ldots, h^{(B)}$ defines a "NN architecture"

Matrix Notation

Represent $\vec{w}_1^{(b)}, \ldots, \vec{w}_{d^{(b)}}^{(b)}$ as a matrix:

$$W^{(b)} = \begin{bmatrix} | & | & & | \\ \vec{w}_1^{(b)} & \vec{w}_2^{(b)} & \cdots & \vec{w}_{d^{(b)}}^{(b)} \\ | & | & & | \end{bmatrix}$$

$$\begin{bmatrix} — (\vec{a}^{(b-1)})^T — \end{bmatrix} \begin{bmatrix} | \\ \vec{w}_j^{(b)} \\ | \end{bmatrix}$$

$$\vec{a}^{(b)} = h^{(b)}\left((\vec{a}^{(b-1)})^T W^{(b)}\right)$$

$$h^{(b)}(\vec{v}) = h^{(b)}(v_1, \ldots, v_d) = \left(h^{(b)}(v_1), \ldots, h^{(b)}(v_d)\right)$$

$$f(\vec{x}) = \vec{a}^{(B)} = h^{(B)}\left(h^{(B-1)}\left(\cdots h^{(2)}\left(\underbrace{h^{(1)}\left(\vec{x}^T W^{(1)}\right)}_{\vec{a}^{(1)}} W^{(2)}\right)\cdots W^{(B-1)}\right) W^{(B)}\right)$$

If $h^{(b)}(z) = z$

Then $f(\vec{x}) = \vec{x}^T W^{(1)} W^{(2)} W^{(3)} \cdots W^{(B)}$

$$= \underbrace{\vec{x}^T W}_{\in \mathbb{R}^{d^{(B)}}} \quad W \in \mathbb{R}^{(d+1) \times d^{(B)}}$$

($d+1$: rows; $d^{(B)}$: columns)

Just $d^{(B)}$ linear functions

i.e., multi-output linear Regression

ERM With Neural Networks

$$D = ((\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n))$$

$$\mathcal{A}(D) = \hat{f} = \underset{f \in \tilde{F}}{\operatorname{argmin}} \hat{L}(f) \quad \text{where} \quad \hat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\vec{x}_i), y_i)$$

$\tilde{F} = \{ f \mid f : \mathcal{X} \to \mathcal{Y}$ and f is a NN with a specific architecture$\}$

every $f_{W^{(1)}, \ldots, W^{(B)}} \in \tilde{F}$ is defined by B weight matrices $W^{(1)}, \ldots, W^{(B)}$

$$\mathcal{A}(D) = \hat{f} \quad \text{where} \quad \hat{f}(\vec{x}) = \vec{a}^{(B)} \quad \text{is defined by}$$

$$\hat{W}^{(1)}, \ldots, \hat{W}^{(B)} = \underset{W^{(1)}, \ldots, W^{(B)}}{\operatorname{argmin}} \underbrace{\hat{L}\left(f_{W^{(1)}, \ldots, W^{(B)}}\right)}_{= \hat{L}(W^{(1)}, \ldots, W^{(B)})}$$

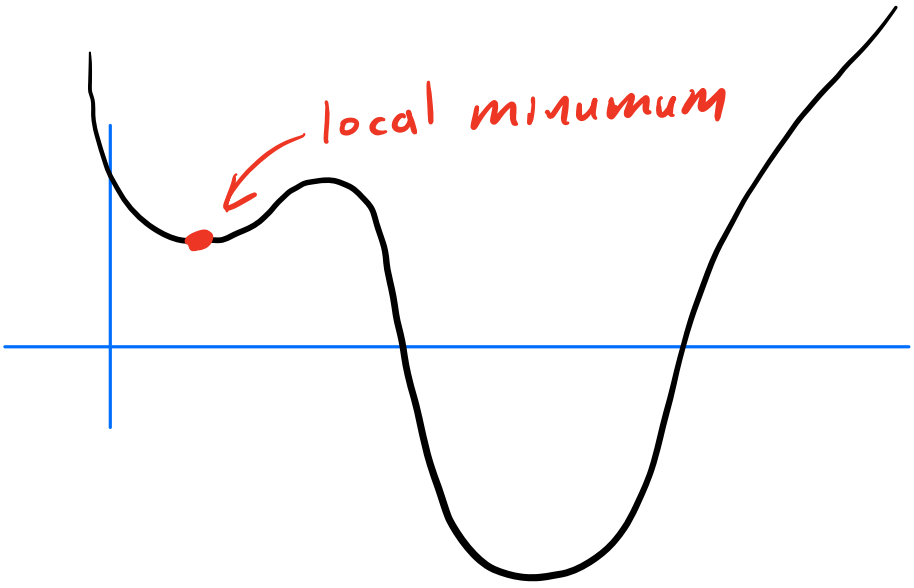
Usually no closed form solution

$$\left(\vec{w}_j^{(b)}\right)^{(t+1)} = \left(\vec{w}_j^{(b)}\right)^{(t)} - \eta^{(t)} \nabla_{\vec{w}_j^{(b)}} \hat{L}\left(\left(W^{(1)}\right)^{(t)}, \ldots, \left(W^{(B)}\right)^{(t)}\right)$$

$$b \in \{1, \ldots, B\}, \quad j \in \{0, \ldots, d^{(b-1)}\}$$

Back propagation

$\hat{L}(W^{(1)}, \ldots, W^{(B)})$ is usually not convex



Softmax

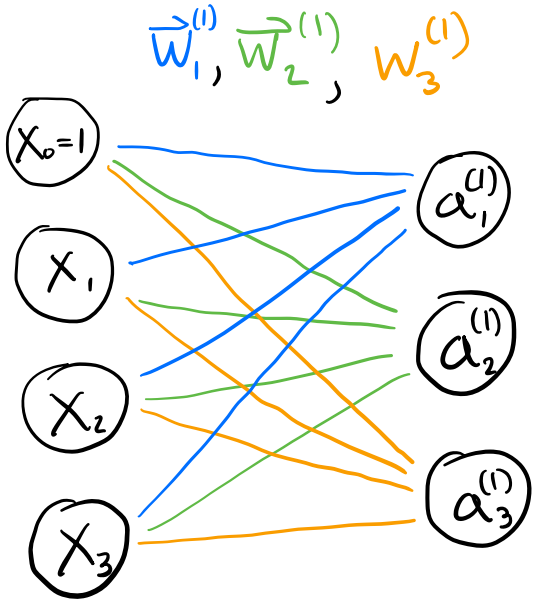
$d=3, k=3,$

$$\sigma(\vec{x}^T\vec{w}_1^{(1)}, \vec{x}^T\vec{w}_2^{(1)}, \vec{x}^T\vec{w}_3^{(1)})$$

$$= \left(\sigma_1(\vec{x}^T\vec{w}_1^{(1)}, \vec{x}^T\vec{w}_2^{(1)}, \vec{x}^T\vec{w}_3^{(1)}), \sigma_2(\vec{x}^T\vec{w}_1^{(1)}, \vec{x}^T\vec{w}_2^{(1)}, \vec{x}^T\vec{w}_3^{(1)}), \sigma_3(\vec{x}^T\vec{w}_1^{(1)}, \vec{x}^T\vec{w}_2^{(1)}, \vec{x}^T\vec{w}_3^{(1)})\right)^T$$

$\vec{x}^T\vec{w}_1^{(1)} \in \mathbb{R}$

Softmax cannot be implemented by a single-layer NN



$$a_1^{(1)} = h^{(1)}(\vec{x}^T\vec{w}_1^{(1)})$$

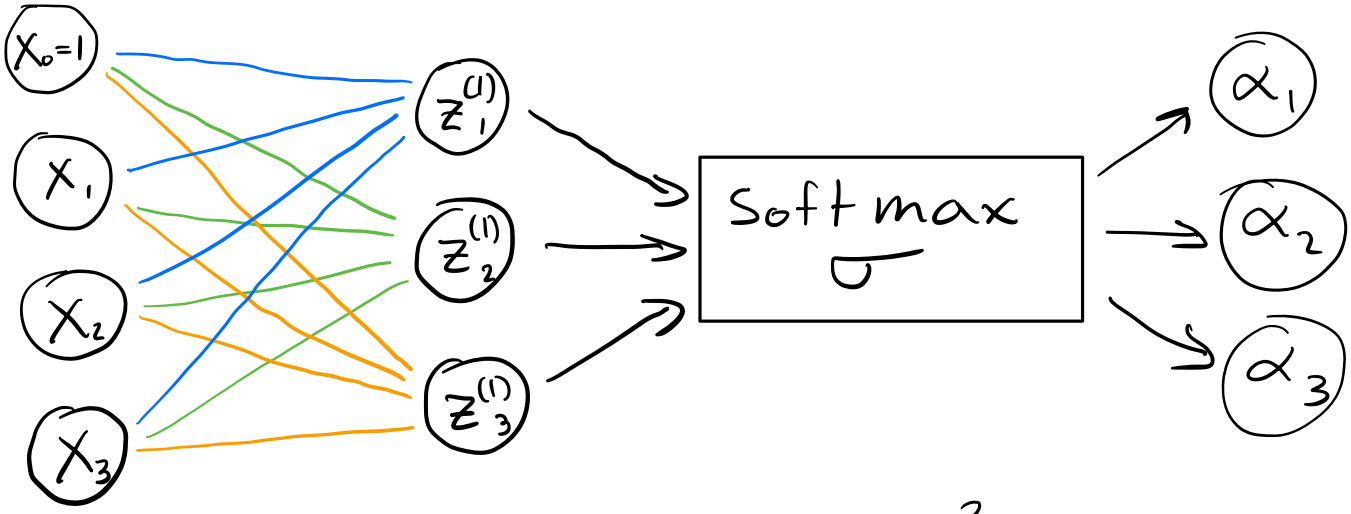
$$a_2^{(1)} = h^{(1)}(\vec{x}^T\vec{w}_2^{(1)})$$

$$a_3^{(1)} = h^{(1)}(\vec{x}^T\vec{w}_3^{(1)})$$

if $h^{(1)}(z) = z$

$$f(\vec{x}) = \vec{a}^{(1)} = (\vec{x}^T\vec{w}_1^{(1)}, \vec{x}^T\vec{w}_2^{(1)}, \vec{x}^T\vec{w}_3^{(1)})^T$$

$\sigma(f(\vec{x}))$ post-processing step



$$\sum_{q=1}^{3} \alpha_q = 1 \,, \quad \alpha_q \in [0,1]$$

$$\sigma\left(z_1^{(1)}, z_2^{(1)}, z_3^{(1)}\right)$$

$$= \left(\sigma_1\left(z_1^{(1)}, z_2^{(1)}, z_3^{(1)}\right), \sigma_2\left(z_1^{(1)}, z_2^{(1)}, z_3^{(1)}\right) \; \sigma_3\left(z_1^{(1)}, z_2^{(1)}, z_3^{(1)}\right) \right)^T$$

$$= \left(\frac{\exp\left(z_1^{(1)}\right)}{\sum_{q=1}^{3} \exp\left(z_q^{(1)}\right)} \,, \frac{\exp\left(z_2^{(1)}\right)}{\sum_{q=1}^{3} \exp\left(z_q^{(1)}\right)} \,, \frac{\exp\left(z_3^{(1)}\right)}{\sum_{q=1}^{3} \exp\left(z_q^{(1)}\right)} \right)^T$$

Notes

- Companies release models with various levels of access

  Black box access (No architecture or weights):

  Chat GPT, Gemini, Claude

  Open Source (Architecture and weights):

  LLaMA, Gemma

- We studied a NN called a Multilayer Perceptron (MLP)

  Other types of NNs include: CNN, RNN, Transformer

  "GPT": Generative Pre-trained Transformer

- Deep learning refers to using NNs with many layers (B is large)