# Neural Networks (part 1)

E190BO

Sept 2, 2026

How to fit a model to data?

How to compute gradient of complex functions?

initialize weights $\triangleq [w_1, w_2 \dots w_m]$

```
for i in range(numIters):
    grad = calcGradient(...)
    weights = weights - γ · grad
```

Backpropagation = computation graph + chain rule

4 steps: ① decompose into composition of simple

② derive derivative of each simple function

③ compute intermediate outputs and

④ multiply derivatives

Example (intuition)

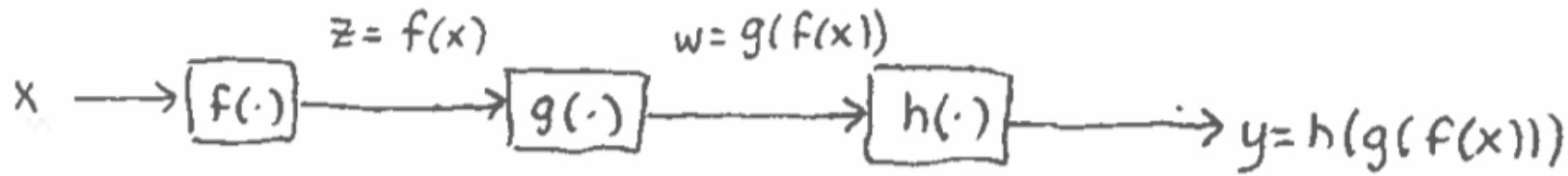
$$y = [\ln(x^2+1)]^2 \quad \Rightarrow \quad \frac{dy}{dx} = 2\ln(x^2+1) \cdot \frac{1}{x^2+1} \cdot 2x$$

$$y = h(g(f(x)))$$

Recall chain rule:

$$\frac{d}{dx} g(f(x)) = g'(f(x))f'(x$$

computation graph:




$$f'(x) \cdot g'(z) \cdot h'(w) = \frac{dy}{dx}$$

# How to verify that gradients are correct?

On a single $(\bar{x}, y)$ data point:

1. compute $\frac{dJ}{d\bar{w}}$ = calc Gradient $(\bar{x}, y, \text{weights})$
2. compute approximate $\frac{d\hat{J}}{d\bar{w}}$ using predict() function

$$J_i^+ = \text{loss}(y, \text{predict}(\bar{x}, \text{weights} + \epsilon \cdot \delta_i))$$

$$J_i^- = \text{loss}(y, \text{predict}(\bar{x}, \text{weights} - \epsilon \cdot \delta_i))$$

where $\delta_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i^{th}$ elemen

$\epsilon$ very small (e.g. 1e-7

3. check that $\frac{dJ}{d\bar{w}} \approx \frac{d\hat{J}}{d\bar{w}}$

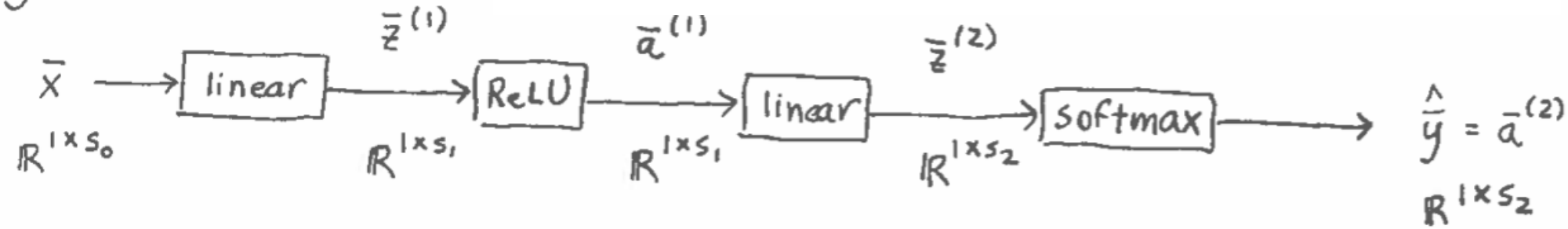
Goal: train a neural network classifier

| | |
|---|---|
| Model | NN classifier |
| Loss | CE |
| Optimizer | mini-batch GD |

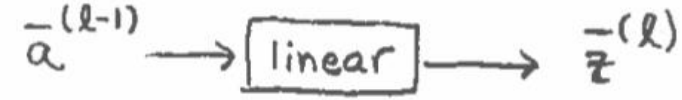
## Model

Dense Neural Network:
- loosely based on biological neurons
- consists of layers of neurons
- each layer consists of linear operation with nonlinear activation function
- # layers and layer sizes are design params

Running example:



## linear




single input $\quad \bar{z}^{(l)} = \bar{a}^{(l-1)} W^{(l)} + \bar{b}^{(l)}$

- $\bar{a}^{(l-1)}$: activations of $l^{th}$ layer
- $W^{(l)}$: weights in $l^{th}$ layer
- $\bar{b}^{(l)}$: bias in $l^{th}$ layer

multiple inputs $\quad Z^{(l)} = A^{(l-1)} W^{(l)} + \bar{b}^{(l)}$

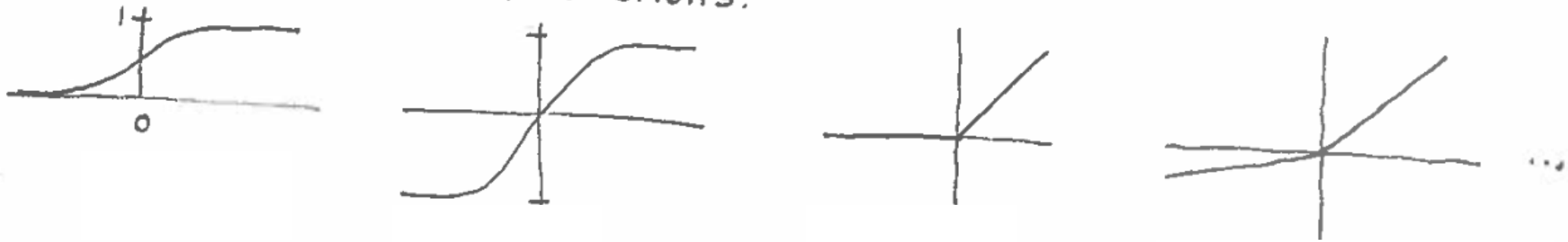
$Z^{(l)} \in \mathbb{R}^{N \times s_l}$

## ReLU

$$\text{ReLU}(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Comments:
- elementwise nonlinearity
- easy to compute

Common nonlinear activation functions:



## Softmax

$$\text{softmax}(\underset{\uparrow}{\bar{z}})_i = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \dots + e^{z_C}}$$
$$\mathbb{R}^C$$

Comments:
- maps $C$ numbers to valid

  i.e. $\sum_i \text{softmax}(\bar{z})_i = 1$

  $0 \le \text{softmax}(\bar{z})_i \le 1$
- higher number → higher probability
- note that $\frac{e^{z_i}}{\sum_j e^{z_j}} = \frac{e^{z_i - k}}{\sum_j e^{z_j - k}}$ ← $k = \max_i z_i$ to avoid
- used at last layer for

## Loss

Cross Entropy Loss: $$J_{CE} = \frac{1}{N} \sum_{i=0}^{N-1} \left( - \sum_{c=1}^{C} y_{ic} \cdot \log(\hat{y}_{ic}) \right)$$

where $C \triangleq$ # output classes

$y_{ic} \triangleq \begin{cases} 1 & \text{if } y_i \text{ has class label } c \\ 0 & \text{otherwise} \end{cases}$

$\hat{y}_{ic} \triangleq$ predicted prob that $\vec{x}_i$ is class $c$

Comments:
- If $y_i = c$, $\text{loss}_i = -\log(\hat{y}_{ic})$
- common loss for classification tasks

## Optimizer

initialize weights = $[\, W^{(1)}, \vec{b}^{(1)}, W^{(2)}, \vec{b}^{(2)} \,]$

for i in range (numIters):

    grad = calcGradient (...)

    weights = weights - $\gamma \cdot$ grad

How to calculate $\frac{\partial J}{\partial W^{(1)}}$, $\frac{\partial J}{\partial \vec{b}^{(1)}}$, $\frac{\partial J}{\partial W^{(2)}}$, $\frac{\partial J}{\partial \vec{b}^{(2)}}$ ?

Consider a single training example $(\bar{x}, y)$:

