Neural Networks (part 1)

E190BO

Sept 2, 2026

How to fit a model to data? Gradient Descent

How to compute gradient of complex functions? Backpropagation

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 functions

② derive derivative of each simple function

③ compute intermediate outputs and derivatives

④ multiply derivatives (chain rule)

Example (intuition)

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

$y = h(g(f(x))) \Rightarrow \frac{dy}{dx} = h'(g(f(x)))\, g'(f(x)) \cdot f'(x)$

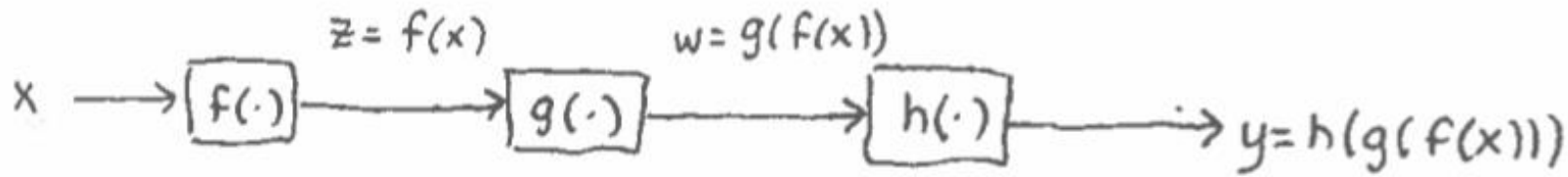
where $f(x) = x^2+1$
$g(x) = \ln x$
$h(x) = x^2$

where $f'(x) = 2x$
$g'(x) = \frac{1}{x}$
$h'(x) = 2x$

Recall chain rule:

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

computation graph:



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

$\Downarrow \quad \Downarrow \quad \Downarrow$

$\frac{dz}{dx} \quad \frac{dw}{dz} \quad \frac{dy}{dw}$

How to verify that gradients are correct? Gradient checking

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

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

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

$$\frac{d\hat{J}}{dw_i} = \frac{J^+ - J^-}{2\epsilon}$$

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

ϵ very small (e.g. 1e-7

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

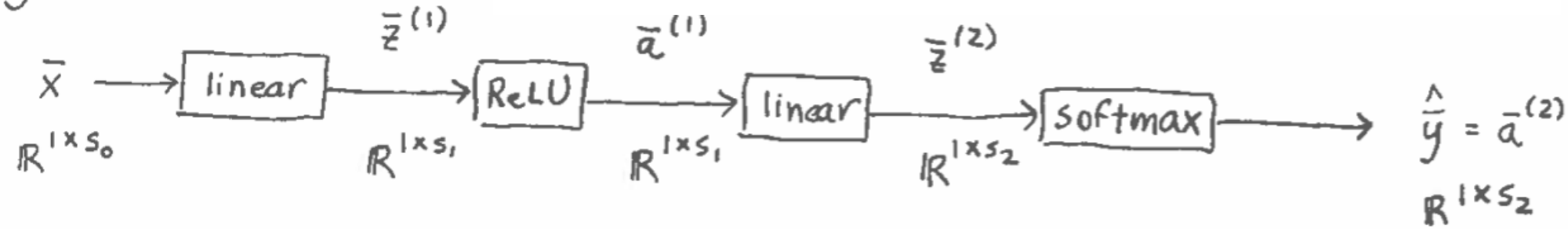
Goal: train a neural network classifier

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

$\bar{a}^{(l-1)} \longrightarrow \boxed{\text{linear}} \longrightarrow \bar{z}^{(l)}$

$\mathbb{R}^{1 \times s_{l-1}}$ $\quad$ $\mathbb{R}^{1 \times s_l}$

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

- $\bar{a}^{(l-1)}$: activations of l^{th} layer, $\mathbb{R}^{1 \times s_{l-1}}$
- $W^{(l)}$: weights in l^{th} layer, $\mathbb{R}^{s_{l-1} \times s_l}$
- $\bar{b}^{(l)}$: bias in l^{th} layer, $\mathbb{R}^{1 \times s_l}$

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

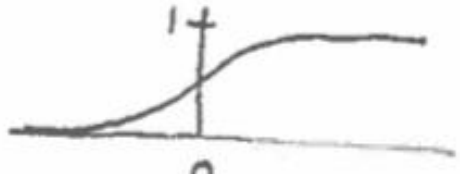
- $Z^{(l)}$: $\mathbb{R}^{N \times s_l}$
- $A^{(l-1)}$: $\mathbb{R}^{N \times s_{l-1}}$
- $W^{(l)}$: $\mathbb{R}^{s_{l-1} \times s_l}$
- $\bar{b}^{(l)}$: $\mathbb{R}^{1 \times s_l}$
- $+$: use broadcasting

ReLU

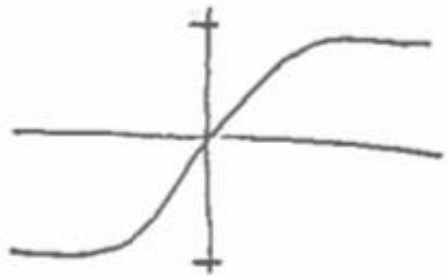
$$\text{ReLU}(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

Comments:
- elementwise nonlinearity
- easy to compute

Common nonlinear activation functions:



sigmoid


tanh


ReLU


Leaky ReLU

...

Softmax

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

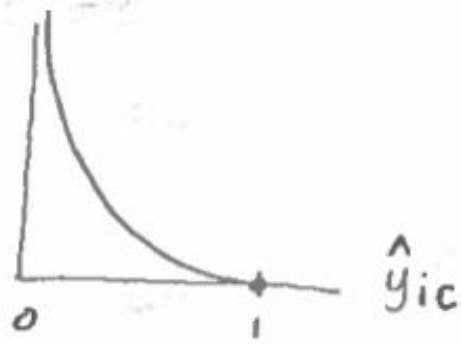
Comments:
- maps C numbers to valid prob distribution

  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 numerical overflow / underflow
- used at last layer for classification tasks

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

$y_{ic} \triangleq \begin{cases} 1 & \text{if } y_i \text{ has class label } c \\ 0 & \text{otherwise} \end{cases}$ ← "one-hot encoding"

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

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

Optimizer

$\mathbb{R}^{S_0 \times S_1}$ $\mathbb{R}^{1 \times S_1}$ $\mathbb{R}^{S_1 \times S_2}$ $\mathbb{R}^{1 \times S_2}$

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 $\frac{\partial J}{\partial W^{(1)}}$, $\frac{\partial J}{\partial \vec{b}^{(1)}}$, $\frac{\partial J}{\partial W^{(2)}}$, $\frac{\partial J}{\partial \vec{b}^{(2)}}$? Backpropagation.

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

