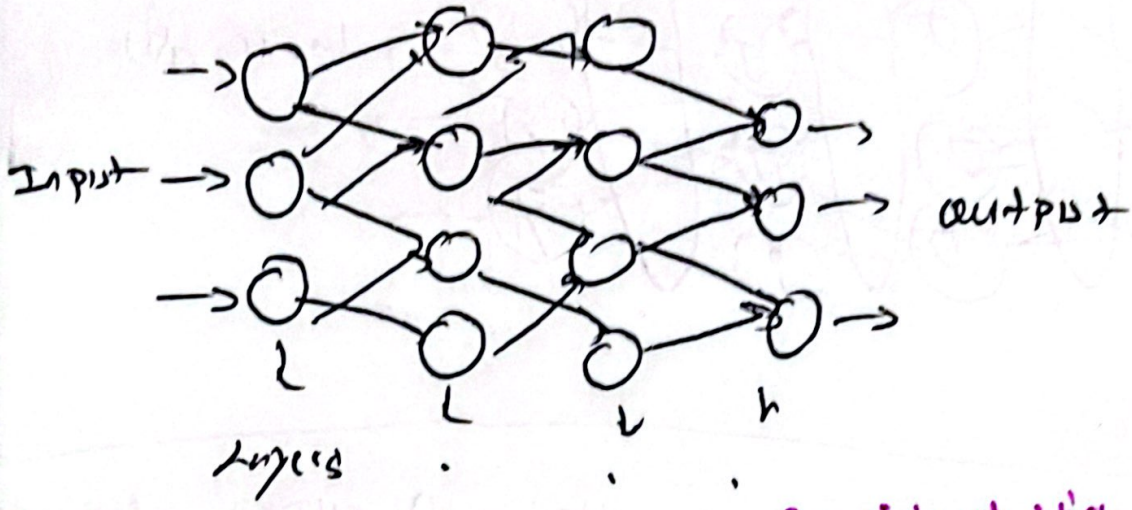
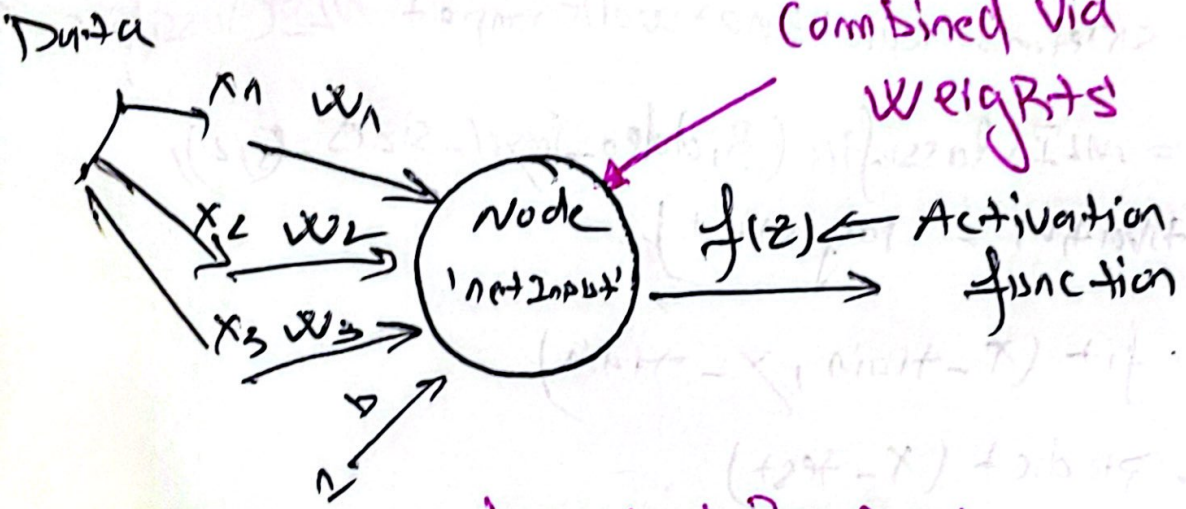
① Neural Net Structure By Zimbo





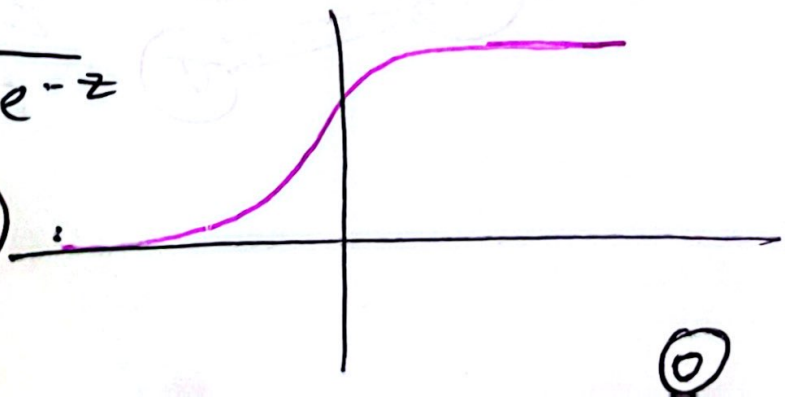
↳ Relation to Logistic Regression

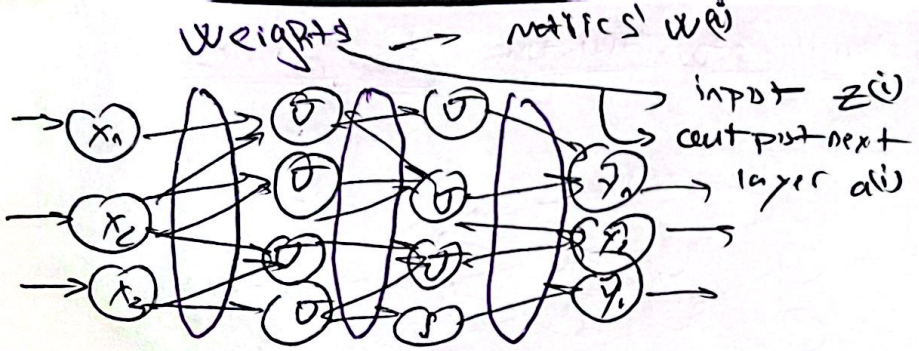
$$f(z) = \frac{1}{1+e^{-z}} \qquad z = b + \sum_{i=1}^{m} x_i w_i + b$$

→ sigmoid function

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

$$\sigma'(z) = \sigma(z)\,(1-\sigma(z))$$





$z^{(1)} = x w^{(1)}$ →

$z^{(2)} = a^2 w^{(2)}$ →

$z^n = a^{(n-1)} w^{(n)}$

→ Put training inp

→ compar →

→ Adjust and re

→ Backpropagat

output

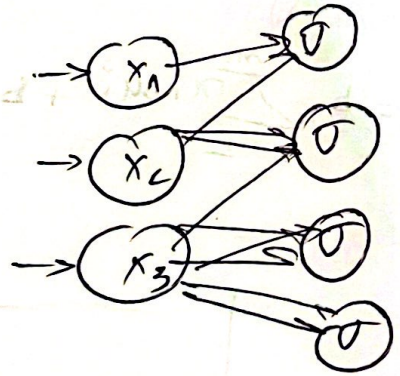
y_1

0

C> $\frac{\delta E}{\delta w_k}$

code

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(5,2),
  activation='logistic')
mlp.fit(X_train, Y_train)
mlp.predict(X_test)
```



$w^{(1)}$ 3×4 matr

$z^{(1)}$ 4 vector

a^1 4 vectors

$x = a^0$

$z^1 = x w^{(1)}$

$a^1 = \sigma(z^1)$

$z^{(1)} = x w^{(1)} \rightarrow a^{(1)} = \sigma(z^{(1)})$

$z^{(2)} = a^{2} w^{(2)} \rightarrow a^{2} = \sigma(z^{(2)})$

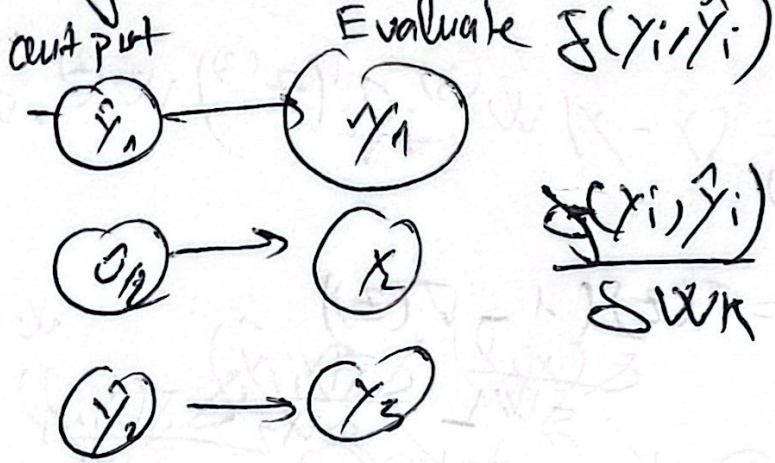
$z^{n} = a^{(n-2)} w^{(n-1)} \rightarrow \hat{y} = \text{softmax}(z^{n})$

→ Put training inputs → Recurpt

→ compar → loss function

→ Adjust and repeat

→ Backpropagation → How to make adjusment



C> $\frac{\partial J}{\partial w_k}$

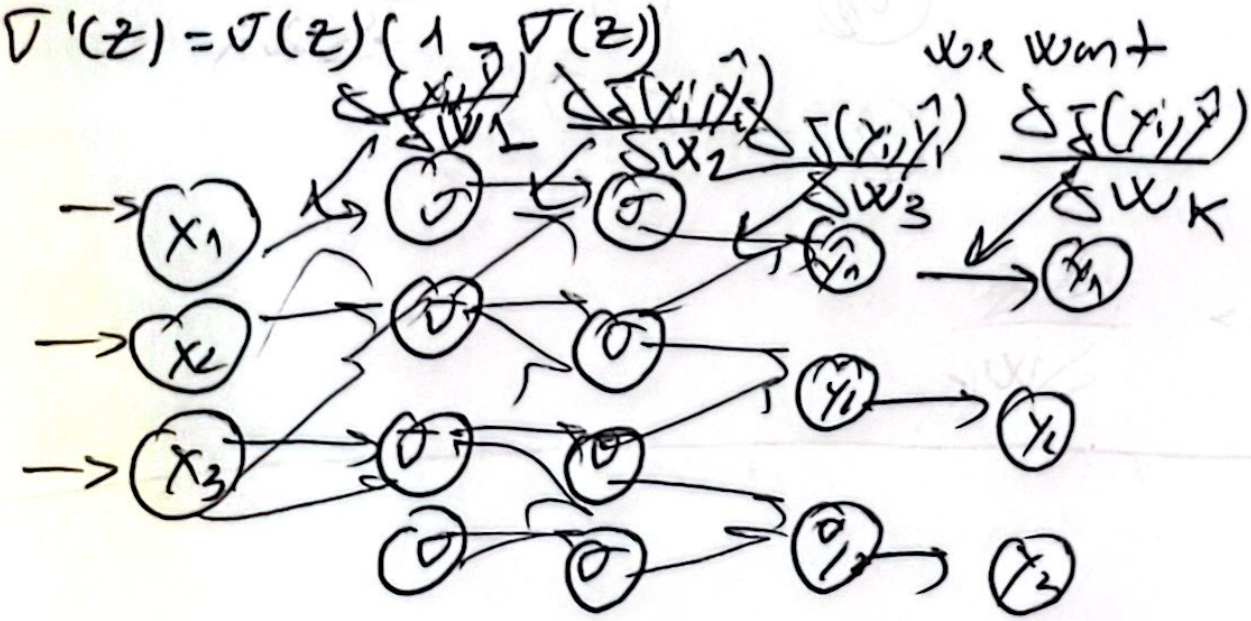
Back propagation : Feedforward network

- $\frac{\partial E}{\partial w^{(n)}} = (\hat{y} - y)\, a^{(n-1)}$
- $\frac{\partial E}{\partial w^{(3)}} = (\hat{y} - y)\, a^{(2)}$

$$\frac{\partial E}{\partial w^{(2)}} = (\hat{y} - y) \cdot w^{(3)} \cdot \sigma'(z^{(3)})\, a^{(1)}$$

$$\frac{\partial E}{\partial w^{(1)}} = (\hat{y} - y)\, w^{(3)} \cdot \sigma'(z^{(3)})\, w^{(2)} \cdot \sigma'(z^{(2)})\, x$$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$



→ mathematics understanding

$$\hat{y} = f(w_1 x_1 \text{ —— } w_n x_n + b)$$

→ loss function

$$L = \frac{1}{2}(y - \hat{y})^2$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w}$$

$$= -(y - \hat{y})$$

$$w = w - \eta \frac{\partial L}{\partial w}$$

→ vanish gradient

$$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)} \quad (6)$$

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

$$\frac{\partial L}{\partial w} = \prod_i f'(z^{(l)}) w^{(l)}$$

$$f'(z) = f(z)(1 - f(z))$$

$$0 < f'(z) \leq 0.25$$

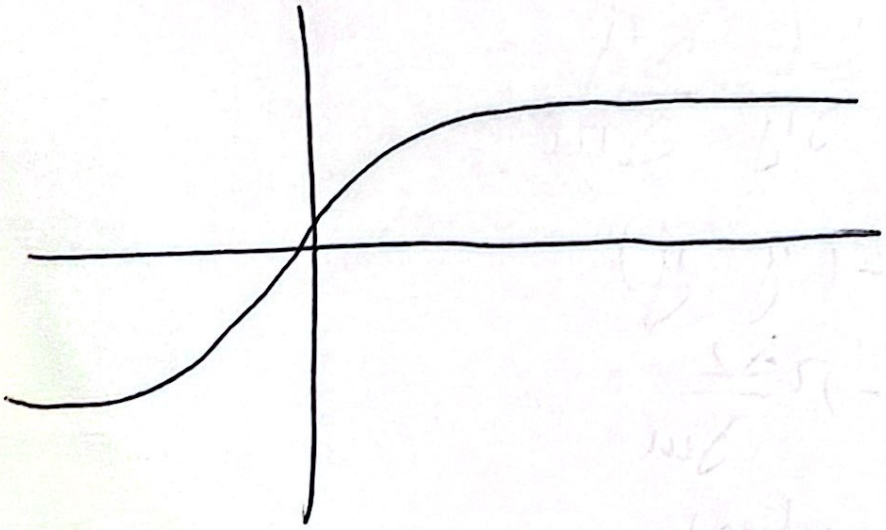
Grad → 0 → How to fix

activate ReLU $f(z) = \max(0, z)$

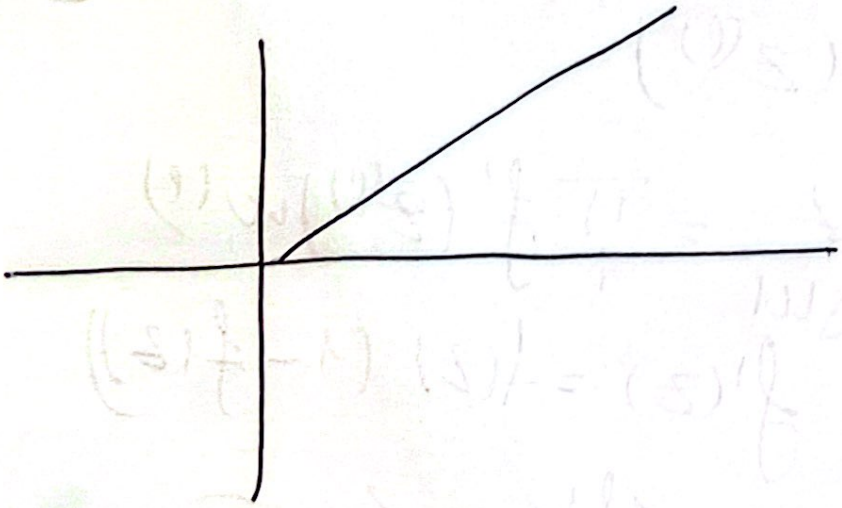
Raz derivate 1 when $z > 1$

→ Activation function

$$\tan R(z) = \frac{\sin R(z)}{\cos R(z)} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

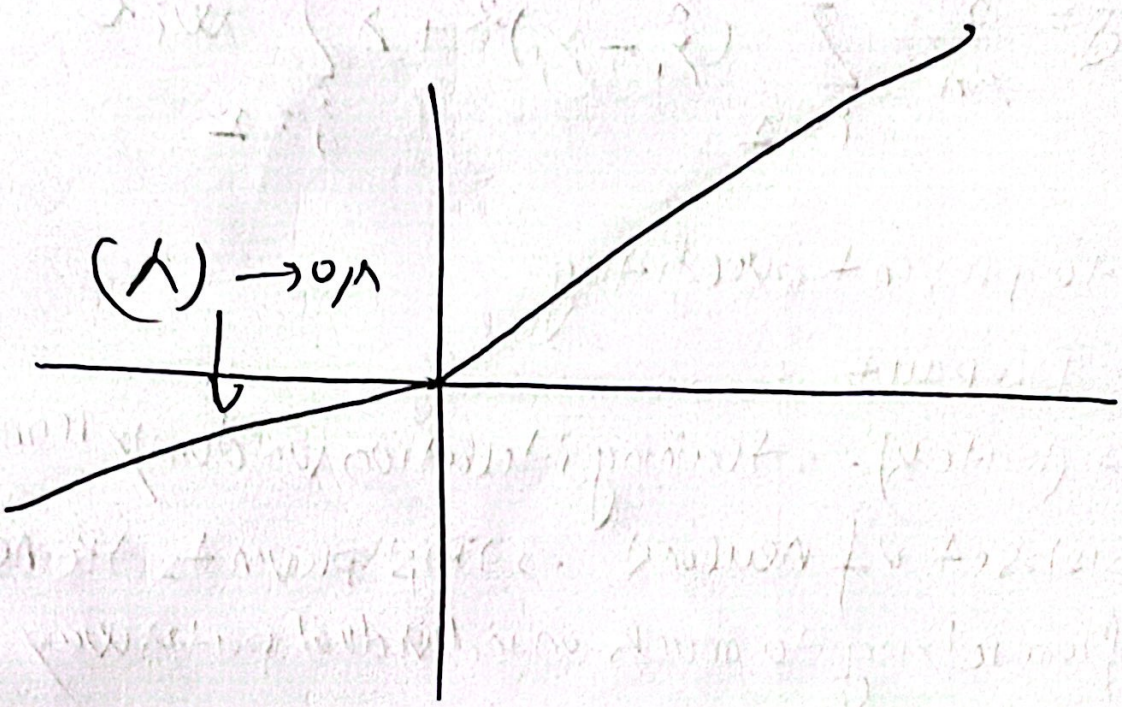


(ReLU) → $\text{ReLU}(z) \begin{cases} 0 & z < 0 \\ z, & z \geq 0 \end{cases} = \max(0, z)$



(Leaky) (LReLU)

$$LReLU(z) = \begin{cases} \alpha z & z < 0 \\ z & z \geq 0 \end{cases} \quad \max(\alpha z, z) \text{ for } \alpha < 1$$



(Optimizers)

$$\min_{W} Loss(W)$$

$$W = W - \alpha \frac{\partial Loss}{\partial W}$$

α → Learning Rate

↳ GD $\boxed{W = W - \alpha \nabla Loss(x_i)}$

Cons: Err are Alot

Regularization

Regularization → Reduce error

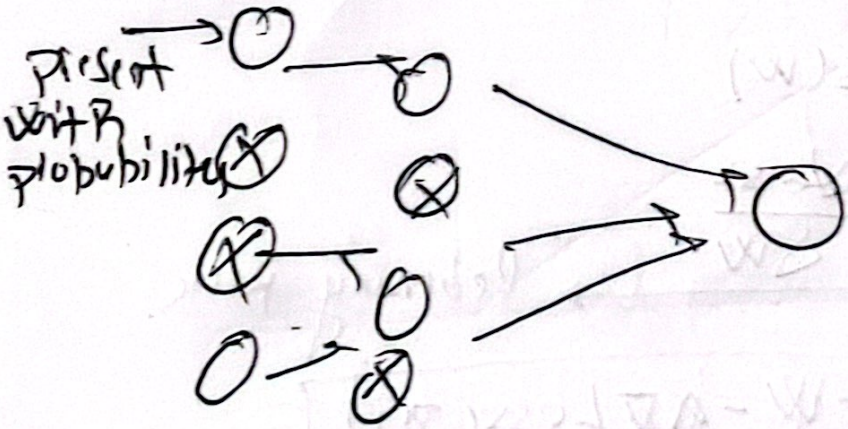
Cost-function

$$J = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{m} w_i^2$$

→ to prevent overfitting

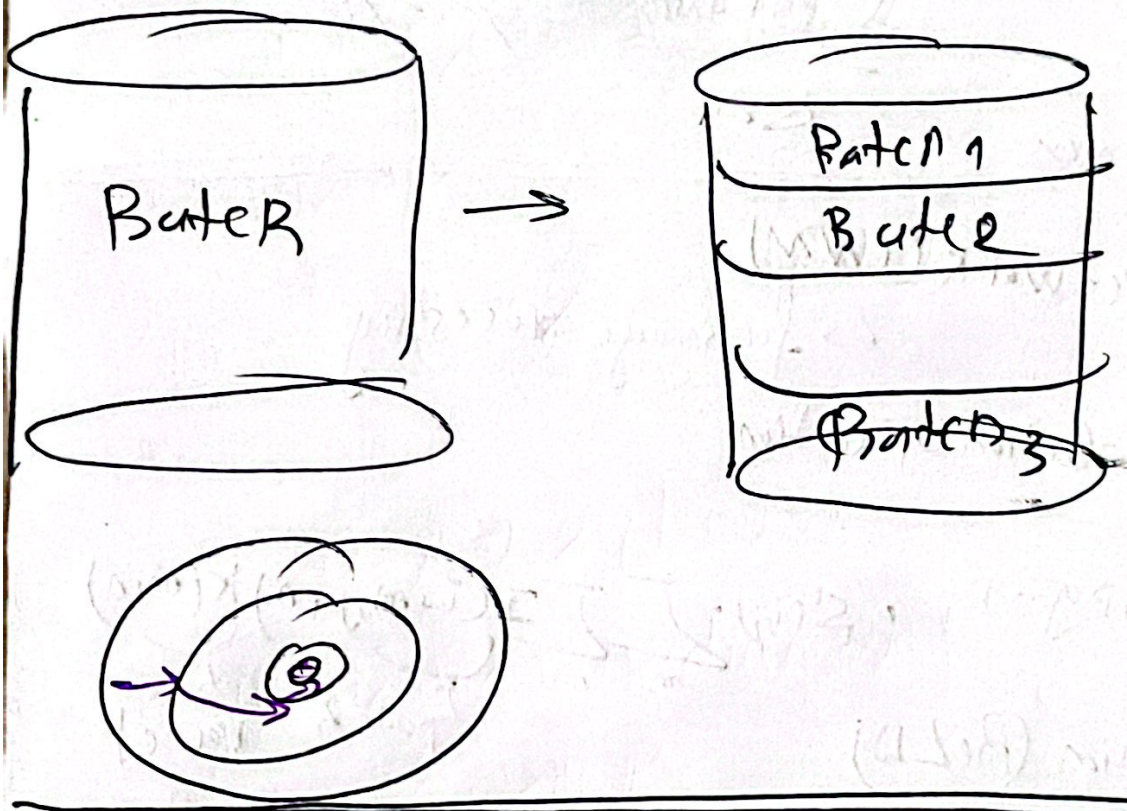
→ Dropout

↳ (batch) → training iteration randomly remove subset of neurons → this prevents the neuro from relying too much on individual pathway make it more "robust"



Data Shuffling

- Randomizing + Reorder
  - prevent overfitting
- Improve (SGD)
- Reduce Bias



→ for (0,1) classification problems → Sigmoid activation ≡ Logestics Regression

1+ Take a vector with length to number of categories

2+ categorie ≡ (1)

Renting $\begin{bmatrix}1\\0\\0\end{bmatrix}$ saving $\begin{bmatrix}0\\1\\0\end{bmatrix}$

Mortage $\begin{bmatrix}0\\0\\1\end{bmatrix}$

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

0 ↙ ↘ 1

^

for loss function we use "CE"

$$CE = \sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

$$\frac{\partial CE}{\partial Softmax} \cdot \frac{\partial Softmax}{\partial z_i} = \hat{y}_i - y_i$$

Neural network (CNN)

↳ for image processing

① Convolution Operation

$$(Img)(jpg\cdots) \rightarrow S(i,j) = \sum_m \sum_n I(\underbrace{i+m, j+n}_{\text{pixel}}) \underbrace{K(m,n)}_{\text{kernel}}$$

② Activation (ReLU)

$$f(x) = \max(0, x)$$

③ Pooling → Reducing the Dimention

④ fully connected layer

$$Y = Wx + b$$

$$P(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Vanished Gradint problem

Reduce this

"Fine-Tuning" ⇒ pre-trained network

↳

① Pretained Model

$$f_\theta(x) = y$$

↳ model parameter

θ^* → optimized

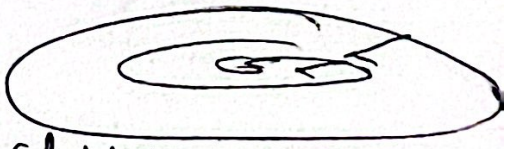
$$\theta^* = \arg\min \sum_{(x,y)\in D} \mathcal{L}(f_\theta(x), y)$$

↳ optimization

$$\theta^*_{fine\text{-}tune} = \arg\min \sum_{(x,y)\in D} \mathcal{L}(f_\theta(x), y)$$

↳ SGD

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(f_\theta(x), y)$$



② Regularization to Reduce overfitting

$$\mathcal{L}_{new} = \mathcal{L} + \lambda \|\theta\|^2$$

③

$$f_{\theta'}(x) \approx f_{\theta_{fine}}(x)$$

$$\theta_{fine} \simeq \theta + \Delta\theta$$

RNN, RL, GAN

(RNN) → for sequential data (time series, text, speech) → allowing info persistence

$$h_t = \sigma(W_R R_{t-1} + W_x x_t + b_R)$$

$$y_t = \sigma(W_y R_t + b_y)$$

Application → language modeling
↳ machine tla
↳ speech Rec

---

2) (RL)

$$V^{\pi}(s) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s\right]$$

$\gamma \in [0, 1]$

$\pi(a, s')$ proba for $\rightarrow a$, $\rightarrow s'$

⋮

Reda azzdine