Contents

1 Basics & Feed Forward Networks

1.1 Multilayer Perceptron

Name	Symbol	Dimension
Input Vector	$\boldsymbol{x}$	$N_{in} \times 1$
Weights	$\boldsymbol{W}$	$N_{out} \times N_{in}$
Bias	$\boldsymbol{b}$	$N_{out} \times 1$
Pre-activation	$\boldsymbol{z}$	$N_{out} \times 1$
Activation	$\sigma(\cdot)$	$N_{out} \times 1$
Output	$\boldsymbol{h} = f(\boldsymbol{x})$	$N_{out} \times 1$
Number of neurons	D	
Number of hidden layers	K	

Maybe an image here?

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N_{in}} \\ w_{21} & w_{22} & \cdots & w_{2N_{in}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N_{out}1} & w_{N_{out}2} & \cdots & w_{N_{out}N_{in}} \end{bmatrix} = \begin{bmatrix} w_{1\leftarrow 1} & w_{1\leftarrow 2} & \cdots & w_{1\leftarrow N_{in}} \\ w_{2\leftarrow 1} & w_{2\leftarrow 2} & \cdots & w_{2\leftarrow N_{in}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N_{out}\leftarrow 1} & w_{N_{out}\leftarrow 2} & \cdots & w_{N_{out}\leftarrow N_{in}} \end{bmatrix}$$

Here, $w_{j\leftarrow i}$ denotes the weight from neuron $i^{(l-1)}$ to neuron $j^{(l)}$.

$$\boldsymbol{h}^{(l)} = \sigma \underbrace{\left(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right)}_{\boldsymbol{z}^{(l)}=\text{pre-activation}}, \quad \boldsymbol{y} = f_{\boldsymbol{\theta}(\boldsymbol{x})}$$

1.2 Parameter Count

$$3D + 1 + (K-1)D(D+1)$$

1.3 Activation Functions

Name	Formula
Sigmoid	$\sigma(\mathbf{h}) = \frac{1}{1+e^{-\mathbf{h}}}$
Tanh	$\tanh(\mathbf{h}) = \frac{e^{\mathbf{h}}-e^{-\mathbf{h}}}{e^{\mathbf{h}}+e^{-\mathbf{h}}}$
ReLU	$\text{ReLU}(\mathbf{h}) = \max(0, \mathbf{h})$
Softmax (Output Layer)	$\pi_d = \frac{e^{h_d}}{\sum_j e^{h_j}}$

1.4 Backpropagation

Chain Rule:

$$\ldots$$

2 Training & Optimization

2.1 Loss Functions $L(\phi)$

Given dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$:

Name	Formula	Regression/Classification	Notes
...	...	...	...

2.2 Gradient Descent Updates

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

2.3 Optimizers

SGD + Momentum:

$$\begin{aligned} m_{t+1} &= \beta m_t + (1 - \beta) \nabla L(\theta_t) \\ \theta_{t+1} &= \theta_t - \eta m_{t+1} \end{aligned}$$

Adam:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\ \hat{m}_t &= m_t / (1 - \beta_1^t) \\ \hat{v}_t &= v_t / (1 - \beta_2^t) \\ \theta_{t+1} &= \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned}$$

3 Regularization & Initialization

3.1 Bias-Variance Tradeoff

$E_{gen} = \text{Bias}^2 + \text{Variance} + \text{Noise}$.

3.2 Regularization

- **L2 (Weight Decay):** $L' = L + \frac{\lambda}{2}||\mathbf{w}||^2$. Gradient update adds $+\lambda\mathbf{w}$.
- **L1 (Lasso):** $L' = L + \frac{\lambda}{2}||\mathbf{w}||_1$. Induces sparsity.
- **Dropout:** Randomly zero activations with prob p. Scale activations by $1/(1-p)$ during training to maintain magnitude.
- **Batch Normalization:** Normalize inputs per mini-batch (μ_B, σ_B). Learnable parameters γ, β. At test time, use running stats.
- **Data Augmentation:** Increase dataset size via transformations (flips, crops, noise).
- ...

3.3 Weight Initialization

Variance matching to maintain signal magnitude.

He (Kaiming) Init: For ReLU.

$$\sigma^2 = \frac{2}{D_{in}}$$

. **Glorot (Xavier) Init:** For Sigmoid/Tanh.

$$\sigma^2 = \frac{2}{D_{in} + D_{out}}$$

.

4 Convolutional Neural Networks

...