Summary of Activation Functions, Loss Functions, and Optimizers

Contents

1 Activation Functions **1**
- 1.1 Sigmoid . . . 1
- 1.2 Tanh . . . 1
- 1.3 ReLU . . . 1
- 1.4 Leaky ReLU . . . 1
- 1.5 Softmax . . . 2
- 1.6 GELU . . . 2

2 Loss Functions **2**
- 2.1 Mean Squared Error (MSE) . . . 2
- 2.2 Mean Absolute Error (MAE) . . . 2
- 2.3 Huber Loss . . . 2
- 2.4 Cross Entropy Loss . . . 2
- 2.5 Focal Loss . . . 2
- 2.6 Dice / IoU Loss . . . 3
- 2.7 KL Divergence . . . 3

3 Optimizers **4**
- 3.1 Stochastic Gradient Descent (SGD) . . . 4
- 3.2 Momentum . . . 5
- 3.3 Adagrad . . . 5
- 3.4 RMSProp . . . 5
- 3.5 Adam . . . 5
- 3.6 AdamW . . . 5
- 3.7 LAMB . . . 5

4 Summary Tables **6**

5 Forward Pass **7**

6 Backpropagation **7**

7 Weight and Bias Updates **7**

8 General Structure of Forward Pass and Backpropagation **8**
- 8.1 Forward Pass . . . 9
- 8.2 Backpropagation . . . 9
- 8.3 Universality of the Process . . . 9
- 8.4 Variations . . . 10
- 8.5 Summary . . . 10

List of Figures

List of Tables

1 Forward Pass Computations and Data Structures 7
2 Comprehensive Backpropagation Steps 8
3 Weight and Bias Update Steps 8

1 Activation Functions

1.1 Sigmoid

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

Applications: Binary classification outputs, logistic regression, simple neural networks. **Advantages:**

- Smooth, differentiable.
- Maps input to $(0, 1)$ range for probability interpretation.

Disadvantages:

- Vanishing gradients for large $|x|$.
- Non-zero-centered outputs slow convergence.

1.2 Tanh

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Applications: Hidden layers of RNNs, MLPs. **Advantages:**

- Zero-centered outputs.
- Stronger gradients than Sigmoid.

Disadvantages:

- Still suffers from vanishing gradients.
- Costlier than ReLU.

1.3 ReLU

$$\text{ReLU}(x) = \max(0, x)$$

Applications: CNNs, fully connected layers in deep networks. **Advantages:**

- Simple and efficient.
- Sparse activations reduce computation.
- Mitigates vanishing gradient problem.

Disadvantages:

- "Dying ReLU" problem for negative inputs.
- Unbounded positive outputs.

1.4 Leaky ReLU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

Applications: CNNs where ReLU causes dead neurons. **Advantages:** Avoids dying ReLU; small slope for negatives. **Disadvantages:** Requires α tuning.

1.5 Softmax

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Applications: Multiclass classification output layers. **Advantages:** Converts logits to probabilities. **Disadvantages:** Sensitive to large logits; saturation slows learning.

1.6 GELU

$$\text{GELU}(x) = x\Phi(x)$$

Applications: Transformers, BERT, Vision Transformers. **Advantages:** Smooth and probabilistic activation. **Disadvantages:** More computation-heavy.

2 Loss Functions

2.1 Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Applications: Regression problems, autoencoders. **Advantages:** Penalizes large errors heavily; convex. **Disadvantages:** Sensitive to outliers.

2.2 Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Applications: Robust regression. **Advantages:** Robust to outliers. **Disadvantages:** Non-differentiable at 0.

2.3 Huber Loss

$$L_\delta(e) = \begin{cases} \frac{1}{2}e^2, & |e| \le \delta \\ \delta(|e| - \frac{1}{2}\delta), & |e| > \delta \end{cases}$$

Applications: Regression with moderate outliers. **Advantages:** Combines smoothness (MSE) and robustness (MAE). **Disadvantages:** Requires tuning δ.

2.4 Cross Entropy Loss

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log p_{ik}$$

Applications: Classification (binary, multiclass, multi-label). **Advantages:** Probabilistic; aligns with maximum likelihood. **Disadvantages:** Sensitive to noisy labels; overconfident predictions.

2.5 Focal Loss

$$L = -\alpha(1 - p_t)^\gamma \log(p_t)$$

Applications: Object detection (Mask R-CNN, RetinaNet). **Advantages:** Handles class imbalance. **Disadvantages:** Hyperparameters α, γ need tuning.

2.6 Dice / IoU Loss

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|}, \qquad L = 1 - \text{Dice}$$

Applications: Image segmentation (Mask R-CNN, U-Net). **Advantages:** Works well with imbalanced masks. **Disadvantages:** Non-linear, harder optimization.

2.7 KL Divergence

$$D_{KL}(P \| Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Applications: Variational Autoencoders, distillation. **Advantages:** Measures information difference. **Disadvantages:** Asymmetric; can be unstable.

Loss Functions

Loss	Typical Use	Advantages	Disadvantages
MSE ($\frac{1}{N}\sum(y-\hat{y})^2$)	Regression	Convex, smooth, strong penalty on large errors	Sensitive to outliers
MAE ($\frac{1}{N}\sum\lvert y-\hat{y}\rvert$)	Robust regression	Robust to outliers	Non-differentiable at 0; slower convergence
Huber / Smooth-L_1	Regression, box regression	Combines MSE and MAE; robust and smooth	Requires tuning of δ
Binary CE	Binary classification	Probabilistic; aligns with MLE	Sensitive to noisy labels
Multiclass CE	Multiclass classification	Standard loss with softmax; probabilistic interpretation	Overconfident predictions are heavily penalized
BCEWithLogits	Multi-label classification	Numerically stable (sigmoid + CE combined)	Ignores label dependencies
Focal Loss	Imbalanced classification/detection	Focuses learning on hard examples	Requires tuning of α and γ
Dice Loss	Image segmentation (masks)	Handles class imbalance; measures overlap directly	Unstable for very small targets
IoU (Jaccard) Loss	Segmentation / bounding boxes	Directly optimizes IoU metric	Non-smooth; slower early convergence
Smooth-L_1 (boxes)	Object detection (R-CNNs)	Robust to outliers; standard for box regression	Scale-sensitive
KL Divergence	VAEs, knowledge distillation	Measures divergence between distributions	Asymmetric; unstable when $Q \approx 0$
Triplet Loss	Metric learning	Learns discriminative embedding spaces	Needs triplet mining; margin tuning
Contrastive Loss	Siamese networks / similarity tasks	Learns pairwise distance relationships	Requires balanced positive/negative pairs
Cosine Embedding	Text/vision embeddings	Scale-invariant and simple	Loses magnitude information
Perceptual Loss	Super-resolution / style transfer	Encourages perceptual similarity using deep features	Computationally heavy; needs pretrained ϕ
Total Variation	Image smoothing / denoising	Removes noise, encourages smoothness	Can over-smooth fine details

3 Optimizers

3.1 Stochastic Gradient Descent (SGD)

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

Applications: Classical ML, CNNs. **Advantages:** Simple, effective. **Disadvantages:** Sensitive to learning rate; oscillates.

3.2 Momentum

$$v_t = \beta v_{t-1} + (1-\beta)\nabla_\theta J(\theta_t), \quad \theta_{t+1} = \theta_t - \eta v_t$$

Applications: Deep CNNs. **Advantages:** Faster convergence; less noise. **Disadvantages:** Needs tuning of β.

3.3 Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla_\theta J(\theta_t)$$

Applications: Sparse features (e.g. NLP embeddings). **Advantages:** Adaptive learning rate. **Disadvantages:** Learning rate decays too fast.

3.4 RMSProp

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)g_t^2, \quad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Applications: RNNs, time series, non-stationary data. **Advantages:** Stable, adaptive. **Disadvantages:** Sensitive to β.

3.5 Adam

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

Applications: Deep learning, NLP, Transformers. **Advantages:** Combines momentum + adaptive rate. **Disadvantages:** May generalize poorly.

3.6 AdamW

$$\theta_{t+1} = \theta_t - \eta\left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda\theta_t\right)$$

Applications: Transformers, BERT, ViTs. **Advantages:** Decoupled weight decay → better generalization. **Disadvantages:** Slightly more computation.

3.7 LAMB

$$r_t = \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \quad \theta_{t+1} = \theta_t - \eta\frac{\|\theta_t\|}{\|r_t\|} r_t$$

Applications: Large-batch Transformer training. **Advantages:** Enables distributed training. **Disadvantages:** Complex; more tuning.

4 Summary Tables

Activation Functions

Function	Pros	Cons
Sigmoid	Probabilistic, smooth	Vanishing gradient
Tanh	Zero-centered	Saturation at extremes
ReLU	Sparse, fast	Dead neurons
Leaky ReLU	Fixes dead ReLU	Hyperparam α
Softmax	Probabilities	Saturation
GELU	Smooth	Slower compute

Optimizers

Optimizer	Pros	Cons
SGD	Simple, reliable	Slow, oscillates
Momentum	Smooth convergence	Needs tuning
RMSProp	Stable	Sensitive β
Adam	Fast, adaptive	May overfit
AdamW	Best generalization	Slightly slower

5 Forward Pass

The forward pass computes activations layer by layer using learned weights and biases. Each layer applies linear transformations followed by nonlinear activation functions (e.g., sigmoid, ReLU, etc.).

Table 1: Forward Pass Computations and Data Structures

Step	**Calculation**	**Data Structure**	**Multiplication Type**
Forward Pass			
Input to 1st hidden layer (HI_1)	$HI_1 = W_1 \cdot X + B_1$	HI_1: Matrix, W_1: Matrix, X: Matrix, B_1: Vector	**Dot Product** (between W_1 and X)
Output of 1st hidden layer (HO_1)	$HO_1 = \sigma(HI_1)$	HO_1: Vector	**Element-wise** (Sigmoid applied element-wise)
Input to 2nd hidden layer (HI_2)	$HI_2 = W_2 \cdot HO_1 + B_2$	HI_2: Vector, W_2: Matrix, HO_1: Vector, B_2: Vector	**Dot Product** (between W_2 and HO_1)
Output of 2nd hidden layer (HO_2)	$HO_2 = \sigma(HI_2)$	HO_2: Vector	**Element-wise** (Sigmoid applied element-wise)
Input to output layer (HO_final)	$HO_{final} = W_3 \cdot HO_2 + B_3$	HO_{final}: Vector, W_3: Matrix, HO_2: Vector, B_3: Vector	**Dot Product** (between W_3 and HO_2)
Final output $\hat{Y}$	$\hat{Y} = \sigma(HO_{final})$	$\hat{Y}$: Vector	**Element-wise** (Sigmoid applied element-wise)
Error Calculation	$E = Y - \hat{Y}$	E: Vector, Y: Vector, $\hat{Y}$: Vector	**Element-wise** (Subtraction)

6 Backpropagation

$$\boxed{Error\_Layer\_N = \sigma'_d(Input\_N)\left(Error\_Layer\_N + 1(W\_N + 1)^T\right), \qquad Input\_N = WX + B.}$$

7 Weight and Bias Updates

$$W_{new} = W_{old} + lr \cdot Own\_Error \cdot Own\_Input^T$$

$$Own\_Input = X, \quad Own\_Input\_N = HO_{N-1} = \sigma(HI_{N-1})$$

Table 2: Comprehensive Backpropagation Steps

Step	**Calculation**	**Data Structure**	**Multiplication Type**
Backpropagation			
Error at output layer	$err_{HO_{final}} = E \cdot \sigma'(HO_{final})$	$err_{HO_{final}}$: Vector, E: Vector, $\sigma'(HO_{final})$: Vector	**Element-wise** (Multiplication)
Error in 2nd hidden layer (err_HO_2)	$err_{HO_2} = err_{HO_{final}} \cdot W_3^T \cdot \sigma'(HI_2)$	err_{HO_2}: Vector, W_3^T: Matrix (transpose), $\sigma'(HI_2)$: Vector	**Dot Product** (between $err_{HO_{final}}$ and W_3^T) followed by **Element-wise** multiplication with $\sigma'(HI_2)$
Error in 1st hidden layer (err_HO_1)	$err_{HO_1} = err_{HO_2} \cdot W_2^T \cdot \sigma'(HI_1)$	err_{HO_1}: Vector, W_2^T: Matrix (transpose), $\sigma'(HI_1)$: Vector	**Dot Product** (between err_{HO_2} and W_2^T) followed by **Element-wise** multiplication with $\sigma'(HI_1)$
Error Calculation	$E = Y - \hat{Y}$	E: Vector, Y: Vector, $\hat{Y}$: Vector	**Element-wise** (Subtraction)

Table 3: Weight and Bias Update Steps

Step	**Calculation**	**Data Structure**	**Multiplication Type**
Weight and Bias Updates			
Update W_1 and B_1	$W_1 = W_1 + lr \cdot X^T \cdot err_{HO_1}$, $B_1 = B_1 + lr \cdot err_{HO_1}$	W_1: Matrix, X^T: Matrix (transpose), err_{HO_1}: Vector	**Dot Product** (between X^T and err_{HO_1})
Update W_2 and B_2	$W_2 = W_2 + lr \cdot HO_1^T \cdot err_{HO_2}$, $B_2 = B_2 + lr \cdot err_{HO_2}$	W_2: Matrix, HO_1^T: Matrix (transpose), err_{HO_2}: Vector	**Dot Product** (between HO_1^T and err_{HO_2})
Update W_3 and B_3	$W_3 = W_3 + lr \cdot HO_2^T \cdot err_{HO_{final}}$, $B_3 = B_3 + lr \cdot err_{HO_{final}}$	W_3: Matrix, HO_2^T: Matrix (transpose), $err_{HO_{final}}$: Vector	**Dot Product** (between HO_2^T and $err_{HO_{final}}$)

8 General Structure of Forward Pass and Backpropagation

The general structure of the forward pass and backpropagation is the same for all basic neural networks, regardless of how many hidden layers or neurons are used. The following principles hold true for most feedforward neural networks (also known as **multilayer perceptrons (MLPs)**).

8.1 Forward Pass

Basic Structure

- The network consists of an input layer, one or more hidden layers, and an output layer.
- Each layer computes a weighted sum of its inputs (or the outputs of the previous layer), adds a bias, and applies an activation function (such as sigmoid, ReLU, or tanh) to produce its output.

Computation Flow

- Data flows forward through the network from the input layer to the output layer.
- The output of one layer becomes the input to the next layer.

8.2 Backpropagation

Error Propagation

- After computing the final output during the forward pass, the network compares the predicted output with the expected (target) output.
- The error is calculated using a loss function (for example, mean squared error or cross-entropy).
- This error is propagated backward from the output layer to the hidden layers, adjusting the neuron weights to minimize the error.

Weight Update

- The gradients (rate of change of error with respect to weights) are computed using the chain rule of calculus.
- The weights and biases are updated using gradient descent or its variants (e.g., stochastic gradient descent).

General Steps

1. Compute the error at the output layer.
2. Backpropagate the error through each preceding layer.
3. Update the weights and biases to reduce the overall error.

8.3 Universality of the Process

The process of forward pass and backpropagation described above holds for all basic neural networks, regardless of:

- The number of hidden layers.
- The number of neurons per layer.
- The activation function used.

8.4 Variations

While the core steps remain the same, there are variations among different network types and techniques:

- **Activation Functions:** Modern networks often use ReLU (Rectified Linear Unit) or tanh instead of sigmoid in hidden layers.
- **Loss Functions:** Classification problems typically use cross-entropy loss, while regression problems often use mean squared error (MSE).
- **Learning Algorithms:** Standard gradient descent can be replaced by advanced optimizers like Adam, RMSProp, or Adagrad, which adaptively adjust the learning rate.

8.5 Summary

- The concept of forward pass and backpropagation is **universal** to most feedforward neural networks.
- Differences arise mainly in the architecture (number of layers or neurons) and the type of activation, loss function, and optimization algorithm used.
- As networks become deeper and more complex, the same principles apply, but across more layers.

Examples

- **Deep Neural Networks (DNNs):** Contain more hidden layers, but the core forward and backward propagation remain identical.
- **Convolutional Neural Networks (CNNs)** and **Recurrent Neural Networks (RNNs):** Both employ forward and backward propagation with specialized modifications.

In essence, for any basic feedforward neural network, the process of forward pass and backpropagation remains fundamentally the same.

Index

Activation Functions, 1
- GELU, 2
- Leaky ReLU, 1
- ReLU, 1
- Sigmoid, 1
- Softmax, 2
- Tanh, 1

Backpropagation, 7

Forward Pass, 7

Loss Functions, 2
- Cross Entropy, 2
- Dice Loss, 3
- Focal Loss, 2
- Huber Loss, 2
- IoU Loss, 3
- KL Divergence, 3
- MAE, 2
- MSE, 2

Optimizers, 4
- Adagrad, 5
- Adam, 5
- AdamW, 5
- LAMB, 5
- Momentum, 5
- RMSProp, 5
- SGD, 4

Weight and Bias Updates, 7