CNN and Neural Network — A Mathematical View

Tutorial 2

April 4, 2025

Sigmoid Activation Problem

Question:

Given:

- Weights $w = [0.2, 0.3, 0.9]$
- Bias $b = 0.5$
- Input $x = [0.5, 0.6, 0.1]$

What is the output y using the sigmoid activation function?

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

Solution: Sigmoid Activation

Step 1: Dot Product

$$w \cdot x = (0.2 \times 0.5) + (0.3 \times 0.6) + (0.9 \times 0.1) = 0.1 + 0.18 + 0.09 = 0.37$$

Step 2: Add Bias

$$z = w \cdot x + b = 0.37 + 0.5 = 0.87$$

Step 3: Apply Sigmoid Function

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-0.87}} \approx \frac{1}{1 + 0.419} = \frac{1}{1.419} \approx \boxed{0.705}$$

Final Output: $y \approx 0.705$

Activation Output Scenario — Classification Problem

Problem: A binary classification network uses ReLU as activation in its output layer. Given the following scenario:

- Final output (before activation): $x = -2.5$
- Activation used: **ReLU**

Question:

What will be the final output, and how will it affect classification?

Options:

1. The output is 0, which might misclassify negative scores.
2. The output remains -2.5.
3. The ReLU activation ensures proper binary classification.
4. None of the above.

Solution: Activation Output Scenario

Given:

- Pre-activation output: $x = -2.5$
- Activation: ReLU $f(x) = \max(0, x)$

So,

$$\text{ReLU}(-2.5) = 0$$

Interpretation:

- Output becomes **0**, which may be interpreted as a low probability or a non-positive class.
- Using ReLU in the output layer of a classification model is not suitable — use **Sigmoid or Softmax** instead.

Correct answer: Option 1.

Problem 1: Medical Diagnosis

Scenario: Predict diabetes (1) or no diabetes (0) using:

- Features: Blood sugar $= 0.6$, BMI $= 0.8$
- Weights: $w = [0.5, -0.3]$, Bias: $b = 0.2$
- Custom Activation:

$$\text{CustomSig}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z^2 & \text{if } 0 \leq z \leq 1 \\ 1 & \text{if } z > 1 \end{cases}$$

- Decision threshold: 0.5

Task:

1. Compute neuron output
2. Classify the patient

Solution 1: Medical Diagnosis

Step 1: Weighted Sum

$$z = (0.5 \times 0.6) + (-0.3 \times 0.8) + 0.2 = 0.3 - 0.24 + 0.2 = 0.26$$

Step 2: Apply CustomSig

$$\text{Since } 0 \leq 0.26 \leq 1:$$

$$y_{\text{pred}} = z^2 = (0.26)^2 = 0.0676$$

Step 3: Classification

$$0.0676 < 0.5 \Rightarrow \boxed{\text{Class 0 (No diabetes)}}$$

Classification with Custom Activation

You are given a neural network with:

- Inputs: $x_1 = -2, \quad x_2 = 3, \quad x_3 = 1$
- Weights: $w_1 = 0.5, \quad w_2 = -1.0, \quad w_3 = 2.0$
- Bias: $b = -1.5$

The activation function is defined as:

$$f(z) = \frac{1 - e^{-z^2}}{1 + e^{-z^2}} \quad \text{(Squashed Gaussian-like activation)}$$

Tasks:

1. Compute the net input z
2. Compute the output $f(z)$
3. Classify as:
   - Class 1 if $f(z) > 0.7$
   - Class 0 otherwise

Solution: Classification

Step 1: Compute net input

$$z = x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

$$= (-2)(0.5) + (3)(-1.0) + (1)(2.0) + (-1.5) = -1 - 3 + 2 - 1.5 = -3.5$$

Step 2: Compute Activation

$$f(z) = \frac{1 - e^{-(-3.5)^2}}{1 + e^{-(-3.5)^2}} = \frac{1 - e^{-12.25}}{1 + e^{-12.25}} \approx \frac{1 - 4.8 \times 10^{-6}}{1 + 4.8 \times 10^{-6}} \approx \frac{0.999995}{1.0000048}$$

$$f(z) \approx 0.99999$$

Step 3: Classification

$$f(z) = 0.99999 > 0.7 \Rightarrow \textbf{Class 1}$$

Final Answer: Class 1

Problem Setup

Given:

- Network architecture: 2 inputs $\rightarrow$ 2 hidden neurons $\rightarrow$ 1 output
- All neurons share:
  - Weights: $\mathbf{w} = [0.5, -0.5]$
  - Bias: $b = 0.2$
  - Activation: Sigmoid ($\sigma(z) = \frac{1}{1+e^{-z}}$)
- Input vector: $\mathbf{x} = [1, -1]$

Task: Compute hidden layer outputs (h_1, h_2) and final output (o_1).

Step 1: Hidden Layer Calculation

For both hidden neurons (same weights/bias):

$$
\begin{aligned}
z_{\text{hidden}} &= \mathbf{w} \cdot \mathbf{x} + b \\
&= (0.5 \times 1) + (-0.5 \times -1) + 0.2 \\
&= 0.5 + 0.5 + 0.2 = 1.2
\end{aligned}
$$

$$
h_1 = h_2 = \sigma(1.2) = \frac{1}{1 + e^{-1.2}} \approx \boxed{0.768}
$$

Step 2: Output Layer Calculation

Output neuron computation:

$$
\begin{aligned}
z_{\text{output}} &= \mathbf{w} \cdot [h_1, h_2] + b \\
&= (0.5 \times 0.768) + (-0.5 \times 0.768) + 0.2 \\
&= 0.384 - 0.384 + 0.2 = 0.2
\end{aligned}
$$

$$
o_1 = \sigma(0.2) = \frac{1}{1 + e^{-0.2}} \approx \boxed{0.550}
$$

Question 4: Learning Rate Behavior (Updated)

Loss Function:

$$L(w) = (w - 4)^2$$

Initial weight: $w = 0$

Perform 3 updates using stochastic gradient descent (SGD) with:

- (a) Learning rate $\eta = 0.1$
- (b) Learning rate $\eta = 1.0$

Q:

- (i) Compute the gradient $\frac{dL}{dw}$ manually.
- (ii) Perform 3 steps of SGD updates and observe the behavior of weight convergence.

Answer 4: Learning Rate Behavior

Step (i): Gradient Calculation

$$L(w) = (w-4)^2 \Rightarrow \frac{dL}{dw} = 2(w-4)$$

(a) SGD with $\eta = 0.1$, Initial $w = 0$:

$$\text{Step 1: } w_1 = 0 - 0.1 \cdot 2(0-4) = 0.8$$

$$\text{Step 2: } w_2 = 0.8 - 0.1 \cdot 2(0.8-4) = 1.44$$

$$\text{Step 3: } w_3 = 1.44 - 0.1 \cdot 2(1.44-4) = \boxed{1.952}$$

(b) SGD with $\eta = 1.0$, Initial $w = 0$:

$$\text{Step 1: } w_1 = 0 - 1.0 \cdot 2(0-4) = 8$$

$$\text{Step 2: } w_2 = 8 - 1.0 \cdot 2(8-4) = 0$$

$$\text{Step 3: } w_3 = 0 - 1.0 \cdot 2(0-4) = 8$$

Observation:

- $\eta = 0.1$: Smooth convergence toward 4
- $\eta = 1.0$: Oscillates $\rightarrow$ does not converge

Convolution Output Dimension Problem

Question:

Suppose that we have an input image of size 125×49, a filter of size 5×5, padding $P = 2$, and stride $S = 2$. What are the output dimensions?

Formula:

$$\left[\frac{n + 2p - f}{s} + 1\right] \times \left[\frac{n + 2p - f}{s} + 1\right]$$

Note: This formula is typically used for square inputs. For a non-square input like this, apply it separately to height and width.

Step-by-Step Solution

Given:

- Input size: 125×49
- Filter size: 5×5
- Padding: $p = 2$
- Stride: $s = 2$

Output Height:

$$\left[\frac{125 + 2 \times 2 - 5}{2} + 1\right] = \left[\frac{124}{2} + 1\right] = 62 + 1 = \boxed{63}$$

Output Width:

$$\left[\frac{49 + 2 \times 2 - 5}{2} + 1\right] = \left[\frac{48}{2} + 1\right] = 24 + 1 = \boxed{25}$$

Final Output Dimensions: $\boxed{63 \times 25}$

Problem 1: MNIST Classifier

Architecture:

- Input: 28×28 grayscale (1 channel), Batch Size = 1, Bias: N
- CONV-5-16 → POOL-2 → CONV-3-32 → POOL-2 → FC-128 → FC-10

Task: For each layer calculate:

1. Number of weights and biases
2. Feature map dimensions
3. Total trainable parameters

Notation Help:

CONV-K-N = K×K conv with N filters (stride=1, padding=0)
POOL-K = K×K pooling (stride=K, padding=0)

Solution 1: MNIST Calculations

Layer	Weights	Biases	Feature Map Size
Input	-	-	28×28×1
CONV-5-16	(5×5×1)×16 = 400	16	(28-5+1)×(28-5+1)×16 = 24×24×16
POOL-2	-	-	(24/2)×(24/2)×16 = 12×12×16
CONV-3-32	(3×3×16)×32 = 4,608	32	(12-3+1)×(12-3+1)×32 = 10×10×32
POOL-2	-	-	(10/2)×(10/2)×32 = 5×5×32
FC-128	(5×5×32)×128 = 102,400	128	128
FC-10	128×10 = 1,280	10	10

Total Parameters:

400 + 16 + 4,608 + 32 + 102,400 + 128 + 1,280 + 10 = **108,874**

Solution 1: MNIST Calculations

Model: "functional_4"

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(1, 28, 28, 1)	0
conv2d_11 (Conv2D)	(1, 24, 24, 16)	416
max_pooling2d_10 (MaxPooling2D)	(1, 12, 12, 16)	0
conv2d_12 (Conv2D)	(1, 10, 10, 32)	4,640
max_pooling2d_11 (MaxPooling2D)	(1, 5, 5, 32)	0
flatten_4 (Flatten)	(1, 800)	0
dense_8 (Dense)	(1, 128)	102,528
dense_9 (Dense)	(1, 10)	1,290

Total params: 108,874 (425.29 KB)
Trainable params: 108,874 (425.29 KB)
Non-trainable params: 0 (0.00 B)

Problem 2: CIFAR-10 Compact CNN

Architecture:

- Input: 32×32 RGB (3 channels)
- CONV-3-8 $\rightarrow$ POOL-2 $\rightarrow$ CONV-5-16 $\rightarrow$ POOL-2 $\rightarrow$ FC-64 $\rightarrow$ FC-10

Tasks:

1. Compute all parameters
2. Track spatial reduction
3. Compare efficiency with Problem 1

Solution 2: CIFAR-10 Calculations

Layer	Weights	Biases	Feature Map Size
Input	-	-	32×32×3
CONV-3-8	(3×3×3)×8 = 216	8	(32-3+1)×(32-3+1)×8 = 30×30×8
POOL-2	-	-	(30/2)×(30/2)×8 = 15×15×8
CONV-5-16	(5×5×8)×16 = 3,200	16	(15-5+1)×(15-5+1)×16 = 11×11×16
POOL-2	-	-	(11/2)×(11/2)×16 = 5×5×16 (floor division)
FC-64	(5×5×16)×64 = 25,600	64	64
FC-10	64×10 = 640	10	10

Total Parameters:

216 + 8 + 3,200 + 16 + 25,600 + 64 + 640 + 10 = **29,754**

Total Parameters

$$216 + 8 + 3,200 + 16 + 25,600 + 64 + 640 + 10 = \boxed{29,754}$$

Comparison

72% fewer parameters than Problem 1's 108,874

Solution 1: MNIST Calculations

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(1, 32, 32, 3)	0
conv2d (Conv2D)	(1, 30, 30, 8)	224
max_pooling2d (MaxPooling2D)	(1, 15, 15, 8)	0
conv2d_1 (Conv2D)	(1, 11, 11, 16)	3,216
max_pooling2d_1 (MaxPooling2D)	(1, 5, 5, 16)	0
flatten (Flatten)	(1, 400)	0
dense (Dense)	(1, 64)	25,664
dense_1 (Dense)	(1, 10)	650

Total params: 29,754 (116.23 KB)
Trainable params: 29,754 (116.23 KB)
Non-trainable params: 0 (0.00 B)

Key Formulas Explained

Convolutional Layers (CONV-K-N)

- **Weights**: $(K \times K \times C_{\text{in}}) \times N$
  - K: Kernel size (e.g., 3 for 3×3 filter)
  - C_{in}: Input channels (e.g., 1 for grayscale, 3 for RGB)
  - N: Number of filters
  - Example: CONV-3-8 on 32×32 RGB image: $(3 \times 3 \times 3) \times 8 = 216$ weights
- **Biases**: N
  - One bias per filter, added to all positions in the output feature map
- **Feature Map**: $(W - K + 1) \times (H - K + 1) \times N$
  - W, H: Input width/height
  - Example: 32×32 input with CONV-3-0: $(32 - 3 + 1) = 30 \rightarrow$ 30×30 output

Pooling Layers (POOL-K)

- **Feature Map**: $\left\lfloor \frac{W}{K} \right\rfloor \times \left\lfloor \frac{H}{K} \right\rfloor \times C$
  - K: Pool size (e.g., 2 for 2×2 pooling)
  - C: Number of input channels (preserved)
  - Uses floor division ($\lfloor \rfloor$) for non-integer results

What are Trainable Parameters?

- Parameters that are **learned and updated** during training via backpropagation.
- Adjusted using optimizers like SGD, Adam, etc.

Examples:

- Weights and biases in convolutional layers.
- Fully connected (Dense) layer weights.
- γ and β in Batch Normalization.

What are Non-Trainable Parameters?

- Parameters that are **not updated** during training.
- May be computed internally or frozen intentionally.

Examples:

- Running mean/variance in BatchNorm.
- Layers frozen from pretrained models.
- Hyperparameters: kernel size, stride, etc.
- MaxPool / AvgPool layers (no weights).

Comparison Table

Type	**Examples**	**Learnable?**
Conv weights/bias	Filters, biases	Yes
Dense weights	Fully connected layers	Yes
BatchNorm γ, β	Scale and shift	Yes
BatchNorm stats	Mean, variance	No
Pooling layers	MaxPool, AvgPool	No
Frozen layers	Pretrained weights	No

Quick Summary

- Trainable: Learnable weights updated during training.
- Non-trainable: Static or frozen values.
- Helps optimize memory and computational cost.