1 Complete Machine Learning Numericals - Step-by-Step Solutions

1.1 Comprehensive Study Guide for BTech Final Year Exams

1.1.1 All Formulas & Detailed Calculations

1.2 PART 1: ACTIVATION FUNCTIONS

1.2.1 1. Sigmoid Activation Function

Formula:

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

Properties: - Output range: (0, 1) - Used for binary classification output layers - Smooth, differentiable curve

---

Example 1: Calculate sigmoid(2)
Given: x = 2
Step-by-Step Solution:

1. Calculate the exponent: $-x = -2$
2. Compute e^{-2} (where $e \approx 2.71828$):

$$e^{-2} = \frac{1}{e^2} = \frac{1}{7.389} \approx 0.1353$$

3. Add 1 to the result:

$$1 + e^{-2} = 1 + 0.1353 = 1.1353$$

4. Calculate reciprocal:

$$\sigma(2) = \frac{1}{1.1353} \approx 0.8808$$

Final Result:

$$\boxed{\sigma(2) \approx 0.8808}$$

Interpretation: 88.08% probability (high activation)

Example 2: Calculate sigmoid(-1)
Given: x = -1
Step-by-Step Solution:

1. Calculate the exponent: $-x = -(-1) = 1$
2. Compute e^1:
$$e^1 \approx 2.7183$$
3. Add 1:
$$1 + e^1 = 1 + 2.7183 = 3.7183$$
4. Calculate reciprocal:
$$\sigma(-1) = \frac{1}{3.7183} \approx 0.2689$$

Final Result:
$$\boxed{\sigma(-1) \approx 0.2689}$$

Interpretation: 26.89% probability (low activation)

1.2.2 2. Tanh (Hyperbolic Tangent) Activation Function

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

Properties: - Output range: (-1, 1) - Centered at zero (zero mean) - Better for hidden layers than sigmoid

---

Example 1: Calculate tanh(1)
Given: x = 1
Step-by-Step Solution:

1. Compute exponentials:
   - $e^1 \approx 2.7183$
   - $e^{-1} = \frac{1}{e} \approx 0.3679$
2. Calculate numerator:
$$e^1 - e^{-1} = 2.7183 - 0.3679 = 2.3504$$
3. Calculate denominator:
$$e^1 + e^{-1} = 2.7183 + 0.3679 = 3.0862$$

4. Divide:
$$\tanh(1) = \frac{2.3504}{3.0862} \approx 0.7616$$

Final Result:
$$\boxed{\tanh(1) \approx 0.7616}$$

Example 2: Calculate tanh(0.5)
Given: x = 0.5
Step-by-Step Solution:

1. Compute exponentials:
   - $e^{0.5} \approx 1.6487$
   - $e^{-0.5} \approx 0.6065$
2. Calculate numerator:
$$e^{0.5} - e^{-0.5} = 1.6487 - 0.6065 = 1.0422$$
3. Calculate denominator:
$$e^{0.5} + e^{-0.5} = 1.6487 + 0.6065 = 2.2552$$
4. Divide:
$$\tanh(0.5) = \frac{1.0422}{2.2552} \approx 0.4621$$

Final Result:
$$\boxed{\tanh(0.5) \approx 0.4621}$$

1.2.3 3. ReLU (Rectified Linear Unit) Activation Function

Formula:
$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} = \max(0, x)$$

Properties: - Output range: [0,) - Most popular in modern deep networks - Computationally efficient

---

Example: Calculate ReLU for x = -3, 0, 4
Solution:

- **For x = -3:** Since $-3 < 0$,

$$\text{ReLU}(-3) = 0$$

- **For x = 0:** Since $0 \geq 0$,

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

- **For x = 4:** Since $4 > 0$,

$$\text{ReLU}(4) = 4$$

Final Results:

$$\boxed{\text{ReLU}(-3) = 0, \text{ReLU}(0) = 0, \text{ReLU}(4) = 4}$$

1.2.4 4. Softmax Activation Function

Formula: For input vector $\mathbf{z} = [z_1, z_2, ..., z_n]$:

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

Properties: - Output range: (0, 1) for each element - Sum of all outputs = 1 (probability distribution) - Used for multi-class classification

---

Example 1: Calculate softmax([1, 2, 3])
Given: $\mathbf{z} = [1, 2, 3]$
Step-by-Step Solution:

1. Compute exponentials for each element:
   - $e^1 \approx 2.7183$
   - $e^2 \approx 7.3891$
   - $e^3 \approx 20.0855$
2. Sum all exponentials:

$$\sum_{j=1}^{3} e^{z_j} = 2.7183 + 7.3891 + 20.0855 = 30.1929$$

3. Calculate softmax for each element:
   - $$\text{softmax}(1) = \frac{2.7183}{30.1929} = 0.0901$$

- $$\text{softmax}(2) = \frac{7.3891}{30.1929} = 0.2447$$
- $$\text{softmax}(3) = \frac{20.0855}{30.1929} = 0.6652$$

Final Result:

$$\boxed{\text{softmax}([1, 2, 3]) = [0.0901, 0.2447, 0.6652]}$$

Verification: $0.0901 + 0.2447 + 0.6652 = 1.0000$

Example 2: Calculate softmax([0, 0, 0])
Given: $\mathbf{z} = [0, 0, 0]$ (equal values)
Step-by-Step Solution:

1. Compute exponentials:
   - $e^0 = 1$
   - $e^0 = 1$
   - $e^0 = 1$
2. Sum:
   $$\sum e^{z_j} = 1 + 1 + 1 = 3$$
3. Calculate softmax:
   - $$\text{softmax}(0) = \frac{1}{3} \approx 0.3333$$
   - $$\text{softmax}(0) = \frac{1}{3} \approx 0.3333$$
   - $$\text{softmax}(0) = \frac{1}{3} \approx 0.3333$$

Final Result:

$$\boxed{\text{softmax}([0, 0, 0]) = [0.3333, 0.3333, 0.3333]}$$

Interpretation: Uniform probability (no preference, equal likelihood)

1.3 PART 2: LOSS FUNCTIONS

1.3.1 1. Mean Squared Error (MSE) Loss

Formula:

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

Where: - n = number of samples - y_i = actual value - $\hat{y}_i$ = predicted value

Properties: - Used for regression tasks - Penalizes large errors more heavily - Range: [0,)

Example: Multiple Predictions

Given: - Target outputs: $y = [0.4, 0.8, 0.1, 0.7]$ - Predicted outputs: $\hat{y} = [0.5, 0.7, 0.2, 0.6]$ - Number of samples: $n = 4$

Step-by-Step Solution:

1. Calculate error for each sample:
   - Sample 1: $y_1 - \hat{y}_1 = 0.4 - 0.5 = -0.1$
   - Sample 2: $y_2 - \hat{y}_2 = 0.8 - 0.7 = 0.1$
   - Sample 3: $y_3 - \hat{y}_3 = 0.1 - 0.2 = -0.1$
   - Sample 4: $y_4 - \hat{y}_4 = 0.7 - 0.6 = 0.1$
2. Square each error:
   - $(-0.1)^2 = 0.01$
   - $(0.1)^2 = 0.01$
   - $(-0.1)^2 = 0.01$
   - $(0.1)^2 = 0.01$
3. Sum squared errors:

$$\sum_{i=1}^{4}(y_i - \hat{y}_i)^2 = 0.01 + 0.01 + 0.01 + 0.01 = 0.04$$

4. Divide by number of samples:

$$L_{\text{MSE}} = \frac{0.04}{4} = 0.01$$

Final Result:

$$\boxed{L_{\text{MSE}} = 0.01}$$

1.3.2 2. Mean Absolute Error (MAE) Loss

Formula:

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

Properties: - Used for regression tasks - Robust to outliers - Less sensitive to large errors than MSE

Example: Multiple Predictions (Same Data)

Given: - Target outputs: $y = [0.4, 0.8, 0.1, 0.7]$ - Predicted outputs: $\hat{y} = [0.5, 0.7, 0.2, 0.6]$ - Number of samples: $n = 4$

Step-by-Step Solution:

1. Calculate absolute error for each sample:
   - $|0.4 - 0.5| = |-0.1| = 0.1$
   - $|0.8 - 0.7| = |0.1| = 0.1$
   - $|0.1 - 0.2| = |-0.1| = 0.1$
   - $|0.7 - 0.6| = |0.1| = 0.1$
2. Sum absolute errors:

$$\sum_{i=1}^{4} |y_i - \hat{y}_i| = 0.1 + 0.1 + 0.1 + 0.1 = 0.4$$

3. Divide by number of samples:

$$L_{\text{MAE}} = \frac{0.4}{4} = 0.1$$

Final Result:

$$\boxed{L_{\text{MAE}} = 0.1}$$

1.3.3 3. Binary Cross-Entropy (BCE) Loss

Formula:

$$L_{\text{BCE}} = -[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})]$$

Where: - $y \in \{0, 1\}$ (binary target) - $\hat{y} \in (0, 1)$ (predicted probability)

Properties: - Used for binary classification - Natural for probability outputs - Range: [0,)

Example 1: Positive Case (y=1)
Given: - Target: $y = 1$ - Prediction: $\hat{y} = 0.72$
Step-by-Step Solution:

1. Note that $(1 - y) = 1 - 1 = 0$, so the second term becomes zero:

$$L_{\text{BCE}} = -[1 \cdot \log(0.72) + 0 \cdot \log(1 - 0.72)]$$

$$L_{\text{BCE}} = -[1 \cdot \log(0.72)]$$

2. Calculate $\log(0.72)$ (natural logarithm):

$$\ln(0.72) \approx -0.3285$$

3. Apply formula:

$$L_{\text{BCE}} = -(-0.3285) = 0.3285$$

Final Result:

$$\boxed{L_{\text{BCE}} = 0.3285}$$

Interpretation: Low loss (good prediction - model predicted high probability for positive class)

Example 2: Negative Case (y=0)
Given: - Target: $y = 0$ - Prediction: $\hat{y} = 0.15$
Step-by-Step Solution:

1. Note that $y = 0$, so first term becomes zero:

$$L_{\text{BCE}} = -[0 \cdot \log(0.15) + (1 - 0) \cdot \log(1 - 0.15)]$$

$$L_{\text{BCE}} = -[\log(0.85)]$$

2. Calculate $\log(0.85)$:

$$\ln(0.85) \approx -0.1625$$

3. Apply formula:

$$L_{\text{BCE}} = -(-0.1625) = 0.1625$$

Final Result:

$$\boxed{L_{\text{BCE}} = 0.1625}$$

Interpretation: Low loss (good prediction - model predicted low probability for negative class)

1.4 PART 3: NLP TECHNIQUES

1.4.1 1. Bag of Words (BoW)

Concept: Count frequency of each word in a document
Formula:

$$\text{BoW}[i] = \text{count of word}_i \text{ in document}$$

Example: Three Sentences
Corpus: 1. "Text processing is necessary" 2. "Text processing is necessary and important" 3. "Text processing is easy"
Step-by-Step Solution:

1. Create vocabulary (unique words):

$$\text{Vocabulary} = \{\text{text, processing, is, necessary, and, important, easy}\}$$

$$|V| = 7 \text{ words}$$

2. Create BoW vector for each document by counting:

   Document 1: "Text processing is necessary"

   - text: 1, processing: 1, is: 1, necessary: 1, and: 0, important: 0, easy: 0

   $$\text{BoW}_1 = [1, 1, 1, 1, 0, 0, 0]$$

   Document 2: "Text processing is necessary and important"

   - text: 1, processing: 1, is: 1, necessary: 1, and: 1, important: 1, easy: 0

   $$\text{BoW}_2 = [1, 1, 1, 1, 1, 1, 0]$$

   Document 3: "Text processing is easy"

   - text: 1, processing: 1, is: 1, necessary: 0, and: 0, important: 0, easy: 1

   $$\text{BoW}_3 = [1, 1, 1, 0, 0, 0, 1]$$

Final Result:
[]@ ¿p() * 0.1562 ¿p() * 0.0938 ¿p() * 0.1875 ¿p() * 0.0625 ¿p() * 0.1719 ¿p() * 0.0625 ¿p() * 0.1719 ¿p() * 0.0938@ Document
text
processing
is
necessary
and
important

easy
1 1 1 1 1 0 0 0
2 1 1 1 1 1 1 0
3 1 1 1 0 0 0 1

1.4.2 2. TF-IDF (Term Frequency - Inverse Document Frequency)

Formulas:

$$TF(t,d) = \frac{\text{count of term } t \text{ in document } d}{\text{total terms in document } d}$$

$$IDF(t) = \log\left(\frac{\text{total documents}}{\text{documents containing term } t}\right)$$

$$\text{TF-IDF}(t,d) = TF(t,d) \times IDF(t)$$

Example: Two Documents

Corpus: - Doc 1: "Text processing is necessary" (4 words) - Doc 2: "Text processing is necessary and important" (6 words) - Total documents: $N = 2$

Step 1: Calculate TF (Term Frequency)

For each word, calculate: $TF = \frac{\text{word count}}{\text{total words in doc}}$

[]@lllll@ Term Count Doc 1 TF Doc 1 Count Doc 2 TF Doc 2
text 1 $1/4 = 0.25$ 1 $1/6 \approx 0.167$
$processing 11/4 = 0.2511/6 \approx 0.167$
$is 11/4 = 0.2511/6 \approx 0.167$
$necessary 11/4 = 0.2511/6 \approx 0.167$
$and 0011/6 \approx 0.167$
$important 0011/6 \approx 0.167$

Step 2: Calculate IDF (Inverse Document Frequency)

For each word: $IDF = \log\left(\frac{N}{\text{doc count}}\right)$

[]@lll@ Term Doc Count IDF = log(2/count)
text 2 (in both) $\log(2/2) = \log(1) = 0$
processing 2 (in both) $\log(2/2) = 0$
is 2 (in both) $\log(2/2) = 0$
necessary 2 (in both) $\log(2/2) = 0$
and 1 (only Doc 2) $\log(2/1) = \log(2) \approx 0.693$
$important 1 (only Doc 2) log(2/1) \approx 0.693$

Step 3: Calculate TF-IDF = TF × IDF

[]@lll@ Term TF-IDF Doc 1 TF-IDF Doc 2
text $0.25 \times 0 = 0$ $0.167 \times 0 = 0$
processing $0.25 \times 0 = 0$ $0.167 \times 0 = 0$
is $0.25 \times 0 = 0$ $0.167 \times 0 = 0$
necessary $0.25 \times 0 = 0$ $0.167 \times 0 = 0$
and $0 \times 0.693 = 0$ $0.167 \times 0.693 \approx 0.116$
$important 0 0.693 = 0 0.167 0.693 \approx 0.116$

Final Result: - Common words (text, processing, is, necessary) have TF-IDF = 0 - Unique words (and, important) to Doc 2 have highest scores - TF-IDF$_{\text{Doc1}} = [0, 0, 0, 0, 0, 0]$ - TF-IDF$_{\text{Doc2}} = [0, 0, 0, 0, 0.116, 0.116]$

1.4.3 3. Co-occurrence Matrix

Concept: Count how often words appear together within a context window
Formula:

$$\text{Co-occurrence}[i, j] = \text{count of times word}_i \text{ appears near word}_j$$

Example: Context Window = 1
Sentences: - “India won the match.” - “I like the match.”
Words: [India, won, the, match, I, like]
Step-by-Step Solution:
For each word, count neighbors within distance 1:
Building the matrix:
Neighbors of “India”: [won] Neighbors of “won”: [India, the] Neighbors of “the”: [won, match] (from sentence 1), [I, match] (from sentence 2) Neighbors of “match”: [the] (twice) Neighbors of “I”: [like, the] Neighbors of “like”: [I, the]
Co-occurrence Matrix (6×6):
[]@lllllll@ India won the match I like
India 0 1 0 0 0 0
won 1 0 1 0 0 0
the 0 1 0 2 1 1
match 0 0 2 0 0 0
I 0 0 1 0 0 1
like 0 0 1 0 1 0

Interpretation: - “India” appears near “won” 1 time - “the” appears near “match” 2 times (appears in both sentences) - “I” appears near “like” 1 time

---

1.4.4 4. Word Embedding (Cosine Similarity)

Concept: Measure semantic similarity between word vectors
Formula:

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum A_i B_i}{\sqrt{\sum A_i^2} \times \sqrt{\sum B_i^2}}$$

Example: Similarity between "India" and "country"
Given Word Embeddings: - V(India) = [0.2, -0.5, 0.3, 0.1] - V(country) = [0.3, -0.4, 0.2, 0.2]
Step-by-Step Solution:

1. Calculate dot product (A · B):

$$A \cdot B = (0.2)(0.3) + (-0.5)(-0.4) + (0.3)(0.2) + (0.1)(0.2)$$

$$= 0.06 + 0.20 + 0.06 + 0.02 = 0.34$$

2. Calculate magnitude of A: $||A|| = \sqrt{\sum A_i^2}$

$$||A|| = \sqrt{(0.2)^2 + (-0.5)^2 + (0.3)^2 + (0.1)^2}$$

$$= \sqrt{0.04 + 0.25 + 0.09 + 0.01} = \sqrt{0.39} \approx 0.624$$

3. Calculate magnitude of B: $||B|| = \sqrt{\sum B_i^2}$

$$||B|| = \sqrt{(0.3)^2 + (-0.4)^2 + (0.2)^2 + (0.2)^2}$$

$$= \sqrt{0.09 + 0.16 + 0.04 + 0.04} = \sqrt{0.33} \approx 0.574$$

4. Calculate cosine similarity:

$$\text{Similarity} = \frac{0.34}{0.624 \times 0.574} = \frac{0.34}{0.358} \approx 0.949$$

Final Result:

$$\boxed{\text{Cosine Similarity} \approx 0.949 \text{ (very high - semantically similar)}}$$

1.5 PART 4: GRADIENT DESCENT

1.5.1 Gradient Descent - Weight Update Algorithm

Formula:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

Where: - w = weight parameter - η = learning rate (step size) - $\frac{\partial L}{\partial w}$ = gradient (partial derivative of loss with respect to weight)

Complete Example: Linear Regression with 3 Iterations

Setup: - Model: $\hat{y} = w \cdot x + b$ (simple linear) - Training data point: x = 2, y = 4 (actual) - Initial parameters: $w_0 = 0.5$, $b_0 = 0.1$ - Learning rate: $\eta = 0.01$ - Loss function: $L = (y - \hat{y})^2$ (MSE)

ITERATION 1:

Step 1: Forward Pass - Calculate Prediction

$$\hat{y} = w \cdot x + b = 0.5 \times 2 + 0.1 = 1.0 + 0.1 = 1.1$$

Step 2: Calculate Loss

$$L = (y - \hat{y})^2 = (4 - 1.1)^2 = (2.9)^2 = 8.41$$

Step 3: Compute Gradients using Chain Rule

Gradient of loss w.r.t. prediction:

$$\frac{\partial L}{\partial \hat{y}} = -2(y - \hat{y}) = -2(4 - 1.1) = -2(2.9) = -5.8$$

Gradient of prediction w.r.t. weight:

$$\frac{\partial \hat{y}}{\partial w} = x = 2$$

Gradient of loss w.r.t. weight (chain rule):

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w} = -5.8 \times 2 = -11.6$$

Gradient of loss w.r.t. bias:

$$\frac{\partial \hat{y}}{\partial b} = 1$$

$$\frac{\partial L}{\partial b} = -5.8 \times 1 = -5.8$$

Step 4: Update Parameters

Update weight:

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

$$w_1 = 0.5 - 0.01 \times (-11.6) = 0.5 + 0.116 = 0.616$$

Update bias:

$$b_{\text{new}} = b_{\text{old}} - \eta \cdot \frac{\partial L}{\partial b}$$

$$b_1 = 0.1 - 0.01 \times (-5.8) = 0.1 + 0.058 = 0.158$$

Iteration 1 Results: $w = 0.616$, $b = 0.158$, $\hat{y} = 1.39$, $L = 6.81$

ITERATION 2:
Step 1: Forward Pass

$$\hat{y} = 0.616 \times 2 + 0.158 = 1.232 + 0.158 = 1.39$$

Step 2: Calculate Loss

$$L = (4 - 1.39)^2 = (2.61)^2 = 6.8121$$

Step 3: Compute Gradients

$$\frac{\partial L}{\partial \hat{y}} = -2(4 - 1.39) = -5.22$$

$$\frac{\partial L}{\partial w} = -5.22 \times 2 = -10.44$$

$$\frac{\partial L}{\partial b} = -5.22 \times 1 = -5.22$$

Step 4: Update Parameters

$$w_2 = 0.616 - 0.01 \times (-10.44) = 0.616 + 0.1044 = 0.7204$$

$$b_2 = 0.158 - 0.01 \times (-5.22) = 0.158 + 0.0522 = 0.2102$$

Iteration 2 Results: $w = 0.7204$, $b = 0.2102$, $\hat{y} = 1.641$, $L = 5.564$

ITERATION 3:
Step 1: Forward Pass

$$\hat{y} = 0.7204 \times 2 + 0.2102 = 1.4408 + 0.2102 = 1.6510$$

Step 2: Calculate Loss

$$L = (4 - 1.6510)^2 = (2.349)^2 = 5.517$$

Step 3: Compute Gradients

$$\frac{\partial L}{\partial \hat{y}} = -2(4 - 1.6510) = -4.698$$

$$\frac{\partial L}{\partial w} = -4.698 \times 2 = -9.396$$

Step 4: Update Parameters

$$w_3 = 0.7204 - 0.01 \times (-9.396) = 0.7204 + 0.09396 = 0.8144$$

$$b_3 = 0.2102 - 0.01 \times (-4.698) = 0.2102 + 0.04698 = 0.2572$$

Iteration 3 Results: $w = 0.8144$, $b = 0.2572$, $\hat{y} = 1.889$, $L = 4.428$

Summary Table: Gradient Descent Convergence

[]@lllll@

Iteration	Weight (w)	Bias (b)	Prediction (ŷ)	Loss
0 (Initial)	0.5	0.1	1.1	8.41
1	0.616	0.158	1.39	6.81
2	0.7204	0.2102	1.641	5.564
3	0.8144	0.2572	1.889	4.428

Observations: - Loss decreases at each iteration: 8.41 → 6.81 → 5.564 → 4.428 - Prediction moves toward target (4): 1.1 → 1.39 → 1.641 → 1.889 - Weights update in correct direction to minimize loss

---

1.6 PART 5: CONVOLUTIONAL NEURAL NETWORKS (CNN)

1.6.1 VGG-19 Forward Propagation - Complete Step-by-Step

Architecture Overview: - Input: $224 \times 224 \times 3$ (RGB image) - 5 Convolutional Blocks with MaxPooling - 3 Fully Connected Layers - Output: 1000 classes (ImageNet)

BLOCK 1: EDGE DETECTION

Layer 1.1: Convolution (64 filters, 3×3, padding=SAME) - Input shape: $224 \times 224 \times 3$ - Filters: 64 different 3×3×3 kernels - Padding: SAME (preserve dimensions) - Activation: ReLU - Output shape: $\mathbf{224 \times 224 \times 64}$

Calculation:

$$\text{Output Height} = \frac{\text{Input Height} - \text{Kernel Size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

$$= \frac{224 - 3 + 2 \times 1}{1} + 1 = \frac{223}{1} + 1 = 224$$

Layer 1.2: Convolution (64 filters, 3×3) - Input: $224 \times 224 \times 64$ - Output: $\mathbf{224 \times 224 \times 64}$

Layer 1.3: MaxPooling (2×2, stride 2) - Input: $224 \times 224 \times 64$ - Pool size: 2×2 - Stride: 2 - Output Height: $\frac{224}{2} = 112$ - Output: $\mathbf{112 \times 112 \times 64}$

BLOCK 2: TEXTURE DETECTION

Layers 2.1-2.2: Convolution (128 filters, 3×3) - Input: $112 \times 112 \times 64$ - Output: $\mathbf{112 \times 112 \times 128}$

Layer 2.3: MaxPooling (2×2, stride 2) - Input: $112 \times 112 \times 128$ - Output Height: $\frac{112}{2} = 56$ - Output: $\mathbf{56 \times 56 \times 128}$

BLOCK 3: PATTERN RECOGNITION

Layers 3.1-3.4: Convolution (256 filters, 3×3) × 4 - Input: $56 \times 56 \times 128$ - After Conv 1: $56 \times 56 \times 256$ - After Conv 2: $56 \times 56 \times 256$ - After Conv 3: $56 \times 56 \times 256$ - After Conv 4: $\mathbf{56 \times 56 \times 256}$

Layer 3.5: MaxPooling (2×2, stride 2) - Output Height: $\frac{56}{2} = 28$ - Output: $\mathbf{28 \times 28 \times 256}$

BLOCK 4: OBJECT PART DETECTION

Layers 4.1-4.4: Convolution (512 filters, 3×3) × 4 - Input: $28 \times 28 \times 256$ - Output: $\mathbf{28 \times 28 \times 512}$

Layer 4.5: MaxPooling (2×2, stride 2) - Output Height: $\frac{28}{2} = 14$ - Output: $\mathbf{14 \times 14 \times 512}$

BLOCK 5: SEMANTIC FEATURE EXTRACTION

Layers 5.1-5.4: Convolution (512 filters, 3×3) × 4 - Input: $14 \times 14 \times 512$ - Output: $\mathbf{14 \times 14 \times 512}$

Layer 5.5: MaxPooling (2×2, stride 2) - Output Height: $\frac{14}{2} = 7$ - Output: $\mathbf{7 \times 7 \times 512}$

FLATTENING LAYER

Operation: Convert 3D tensor to 1D vector

$$\text{Flattened Size} = \text{Height} \times \text{Width} \times \text{Channels}$$

$$= 7 \times 7 \times 512 = 49 \times 512 = 25,088$$

- Input: $7 \times 7 \times 512$ (3D)
- Output: **1 × 25,088** (1D vector)

FULLY CONNECTED LAYERS

FC Layer 1: - Input: 1 × 25,088 - Neurons: 4,096 - Activation: ReLU - Output: **1 × 4,096**

Calculation for first neuron:

$$\mathrm{FC}_1[0] = \mathrm{ReLU}\left(\sum_{i=0}^{25087} w_i \cdot x_i + b_0\right)$$

$$= \max(0, w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_{25088} \cdot x_{25088} + b)$$

FC Layer 2: - Input: 1 × 4,096 - Neurons: 4,096 - Activation: ReLU - Output: **1 × 4,096**

Output Layer (Classification): - Input: 1 × 4,096 - Neurons: 1,000 (ImageNet classes) - Activation: Softmax - Output: **1 × 1,000**

$$\text{Output} = \text{softmax}\left(\sum_{j=0}^{4095} w_j \cdot \mathrm{FC}_2[j] + b\right)$$

Result: Probability distribution over 1,000 classes - Example: [0.02, 0.01, 0.85, 0.05, . . .] = 85% dog

---

1.6.2 Complete VGG-19 Dimension Tracking

[]@llllll@ Layer Type Filters Operation Input Size Output Size
0 Input - RGB Image - **224×224×3**
1.1-1.2 Conv 64×3×3 2 layers, ReLU 224×224×3 224×224×64
1.3 MaxPool 2×2 Stride 2 224×224×64 **112×112×64**
2.1-2.2 Conv 128×3×3 2 layers, ReLU 112×112×64 112×112×128
2.3 MaxPool 2×2 Stride 2 112×112×128 **56×56×128**
3.1-3.4 Conv 256×3×3 4 layers, ReLU 56×56×128 56×56×256
3.5 MaxPool 2×2 Stride 2 56×56×256 **28×28×256**
4.1-4.4 Conv 512×3×3 4 layers, ReLU 28×28×256 28×28×512
4.5 MaxPool 2×2 Stride 2 28×28×512 **14×14×512**
5.1-5.4 Conv 512×3×3 4 layers, ReLU 14×14×512 14×14×512
5.5 MaxPool 2×2 Stride 2 14×14×512 **7×7×512**
Flatten - - Reshape 7×7×512 **1×25,088**
FC1 Dense 4,096 ReLU 25,088 **1×4,096**
FC2 Dense 4,096 ReLU 4,096 **1×4,096**
Output Dense 1,000 Softmax 4,096 **1×1,000**

1.6.3 Convolution Numerical Example

Input Feature Map (5×5):

```
2  4  1  3  2
1  3  5  2  1
4  2  1  3  4
2  3  4  1  2
1  2  3  4  5
```

3×3 Filter (Edge Detector):

```
1   0  -1
1   0  -1
1   0  -1
```

Step 1: Apply filter at position (0,0) with stride 1
Multiply corresponding elements:

```
2×1 + 4×0 + 1×(-1) = 2 + 0 - 1 = 1
1×1 + 3×0 + 5×(-1) = 1 + 0 - 5 = -4
4×1 + 2×0 + 1×(-1) = 4 + 0 - 1 = 3
```

Sum: $1 + (-4) + 3 = 0$
Step 2: Slide filter across entire feature map
Output positions: (5-3+1) × (5-3+1) = 3 × 3
Convolution Output (before activation):

```
0   -2   1
2    1   2
1    3   0
```

After ReLU Activation (max(0, x)):

```
0   0   1
2   1   2
1   3   0
```

1.6.4 Max Pooling Numerical Example

Input (4×4):

```
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
```

2×2 Max Pooling with stride 2:
Take maximum from each 2×2 window:
Window 1 (top-left):

```
1  2
5  6
```

Max = 6
Window 2 (top-right):

```
3  4
7  8
```

Max = 8
Window 3 (bottom-left):

```
9  10
13 14
```

Max = 14
Window 4 (bottom-right):

```
11 12
15 16
```

Max = 16
Pooling Output (2×2):

```
6   8
14  16
```

Dimension: $\frac{4}{2} = 2 \times 2$

1.6.5 Parameter Calculation for VGG-19

Convolution Layer Formula:

$$\text{Params} = (\text{Height} \times \text{Width} \times \text{Input Channels} + 1) \times \text{Output Filters}$$

Example - Block 1, Layer 1.1: - Kernel size: 3×3 - Input channels: 3 (RGB) - Output filters: 64 - Bias: 1 per filter

$$\text{Params} = (3 \times 3 \times 3 + 1) \times 64 = (27 + 1) \times 64 = 28 \times 64 = 1,792$$

Fully Connected Layer Formula:

$$\text{Params} = (\text{Input Neurons} + 1) \times \text{Output Neurons}$$

Example - FC Layer 1: - Input neurons: 25,088 (from flattened 7×7×512) - Output neurons: 4,096 - Bias: 1 per neuron

$$\text{Params} = (25,088 + 1) \times 4,096 = 25,089 \times 4,096 = 102,764,544$$

Total VGG-19 Parameters (approximate):

$$\text{Total} \approx 144 \text{ million parameters}$$

Note: Majority of parameters (~119M) are in FC layers!

1.7 IMPORTANT EXAM NOTES

Key Concepts to Remember:

1. **Activation Functions:**
   - Sigmoid: Output (0,1), used for binary classification
   - Tanh: Output (-1,1), better for hidden layers
   - ReLU: Output [0,), most efficient for deep networks
   - Softmax: Probability distribution, multi-class output
2. **Loss Functions:**
   - MSE: For regression, penalizes large errors heavily
   - MAE: For regression, robust to outliers
   - BCE: For binary classification, uses probabilities
3. **NLP Techniques:**
   - BoW: Simple counting, sparse vectors
   - TF-IDF: Weights words by importance
   - Word2Vec/CBOW: Dense semantic representations
4. **Gradient Descent:**
   - Always shows gradients step-by-step
   - Loss should decrease monotonically

- Learning rate affects convergence speed

5. **CNN Forward Pass:**

   - Track dimensions at each layer
   - Conv preserves dimensions (with padding)
   - Pooling halves spatial dimensions
   - Flattening converts 3D to 1D
   - FC layers for classification

Complete Study Guide with All Formulas & Step-by-Step Calculations Prepared for BTech Final Year - Advanced Machine Learning K. J. Somaiya College of Engineering, Mumbai November 25, 2025