Linear and Softmax Layers in BERT Fine-tuning: Formulas and Functions

In BERT fine-tuning, the **linear and softmax layers** form the classification head that processes the final hidden representations from the transformer to produce predictions. Here's a comprehensive explanation of their mathematical formulas and functions:

Linear Layer (Classification Head)

The linear layer is a fully connected layer that transforms the BERT output representation into class scores.

Formula:
$z = W \cdot h + b$

Where:

- z is the output logits (raw prediction scores)
- W is the weight matrix with dimensions $num\_classes \times hidden\_size$
- h is the hidden representation from BERT (typically the [CLS] token representation)
- b is the bias vector with dimensions $num\_classes$

What it does:

- **Transforms representations**: Converts the high-dimensional BERT output (usually 768 dimensions for BERT-base) into class-specific scores [1] [2]
- **Learns task-specific mappings**: The weight matrix W learns which features in the BERT representation are most important for each class
- **Provides linear transformation**: Each output neuron computes a weighted sum of all input features plus a bias term

Softmax Layer (Activation Function)

The softmax function converts the raw logits from the linear layer into probability distributions.

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

Where:

- z_i is the logit for class i
- K is the total number of classes
- e is the exponential function

What it does:

- **Normalizes outputs**: Ensures all class probabilities sum to 1[3] [4]
- **Amplifies differences**: Uses exponential function to emphasize higher-scoring classes
- **Enables probabilistic interpretation**: Converts raw scores into meaningful probabilities for decision-making

Combined Process in BERT Fine-tuning

The complete classification pipeline follows this sequence:

1. **BERT Processing**: Input text → BERT layers → [CLS] token representation h
2. **Linear Transformation**: $z = W \cdot h + b$
3. **Softmax Activation**: $p = softmax(z)$
4. **Prediction**: $\hat{y} = \arg\max(p)$

Training Process

During fine-tuning, these layers are optimized using:

- **Cross-entropy loss**: $L = -\sum_{i=1}^{K} y_i \log(p_i)$
- **Backpropagation**: Gradients flow back through softmax → linear → BERT layers
- **Parameter updates**: Both the linear layer weights and BERT parameters are updated

Practical Implementation

In practice, the linear and softmax layers work together as follows[2] [5]:

```
# Conceptual implementation
linear_output = torch.matmul(bert_output, weight_matrix) + bias
probabilities = torch.softmax(linear_output, dim=-1)
```

Key Functions

Linear Layer Functions:

- Feature extraction and dimensionality reduction
- Task-specific weight learning
- Bias adjustment for class imbalances

Softmax Layer Functions:

- Probability normalization
- Confidence scoring
- Multi-class decision making

Variations and Enhancements

Recent research has explored alternatives to the standard linear+softmax approach:

- **Multiple linear layers**: Adding depth to the classification head [2]
- **Attention mechanisms**: Using attention to weight different parts of the representation [4]
- **Alternative activation functions**: Exploring functions beyond softmax for specific tasks [6]

The linear and softmax layers, while conceptually simple, are crucial for translating BERT's rich contextual representations into task-specific predictions. Their mathematical simplicity belies their importance in the overall fine-tuning process, serving as the bridge between pre-trained language understanding and downstream task performance.

⁂

1. https://www.semanticscholar.org/paper/b964afe5b755022f1f1e6915d23df9a7f65c911c
2. https://ieeexplore.ieee.org/document/10742347/
3. https://www.mdpi.com/1424-8220/23/3/1481
4. https://journals.sagepub.com/doi/10.3233/KES-230066
5. https://www.mdpi.com/2078-2489/14/8/467
6. https://arxiv.org/abs/2408.08803