lab1

group9	姓名
Team Member 1	贾世安(JIA SHIAN)
Team Member 2	陶毅诚(TAO YICHENG)

question1

1. what an embedding layer does?

- An embedding layer is a neural network layer that maps each integer to a high-dimensional vector. The purpose of the embedding layer is to learn a representation of the words that captures their semantic meaning. This is important because it allows the LSTM to better understand the relationships between words in the text.

2. why we cannot just feed the integers from the tokenizer direct to the LSTM.

- **Lack of semantic meaning**: Integers produced by a tokenizer do not inherently carry any semantic meaning. They are simply numerical representations assigned to different words based on their position in the vocabulary. LSTM or recurrent layers are designed to capture sequential dependencies and patterns in data, but they are not equipped to interpret the semantic relationships between words.
- **High-dimensional input**: Tokenizers typically assign a unique integer to each word in the vocabulary, resulting in a large range of integers. Directly feeding these high-dimensional integer sequences to an LSTM would require the network to handle a high number of unique inputs, which can be computationally expensive and lead to overfitting due to the large number of parameters.
- **Loss of word relationships**: When tokenizing text, words are often split into their individual units (tokens) based on punctuation and other rules. By feeding only the integers to an LSTM, we lose the information about the relationships between the tokens within a word. This can lead to a loss of important semantic information and hinder the model's ability to understand the context and meaning of the text.

question2

1. Explain why we don't need to chop up our tokens into groups of 5 tokens to predict the 6th for transformers, but must do so for LSTMs.

- The difference in handling token sequences between transformers and LSTMs is primarily due to their architectural design and the way they process information.
- Transformers are based on the self-attention mechanism and operate on the entire sequence of tokens at once. They have a parallel processing capability that allows them to capture dependencies and relationships between tokens across long distances. This means that transformers can consider the context of any token within the entire sequence when making predictions. Therefore, there is no need to chop up tokens into groups to predict the next token in a transformer model.
- On the other hand, LSTMs are sequential models that process token sequences one element at a time in a sequential manner. They maintain an internal state that carries information from previous tokens to inform the prediction of the next token. LSTMs have a fixed memory size, and long sequences can cause memory limitations and computational inefficiency.
- To overcome these limitations, when using LSTMs or other recurrent neural networks (RNNs), token sequences are often divided into smaller chunks or groups. By breaking the sequence into chunks, the memory constraints are reduced, and the model can process the tokens in smaller, manageable parts. Each chunk is processed sequentially, and the hidden state from one chunk is passed to the next as a form of context.

question3

1. In our network we have used a one-hot approach; our network will have over 50,000 outputs, where one of them will be set to "1" and the rest to "0" when training. Why can't we just have one output, where the target value is the index of the next word?

- Dimensionality: By having a single output, you are essentially trying to represent over 50,000 different classes with a single value, which would be challenging for a neural network to learn. Each word would need to be represented by a unique value within a limited range, making it difficult for the network to capture the nuances and relationships between different words.
- Loss Function: In order to train a neural network, you need to define a loss function that quantifies the difference between the predicted output and the true target. With a single output, you would need to devise a loss function that can handle such a high-dimensional target space. This would likely be complex and less intuitive compared to using a one-hot encoding, which allows for a straightforward comparison between the predicted and true values.

- Model Interpretability: Using a one-hot encoding provides a clear interpretation of the network's output. Each output neuron corresponds to a specific word, and the neuron with the highest value indicates the predicted word. This makes it easier to understand and analyze the model's predictions. With a single output, it would be more challenging to interpret and analyze the network's behavior.
- Generalization: One-hot encoding allows for better generalization. Each output neuron is responsible for predicting a specific word, enabling the model to learn distinct patterns and relationships between words. This allows the model to generalize its knowledge to unseen data. A single output may struggle to capture the complexity of word relationships and may result in poorer generalization performance.
- Overall, using a one-hot approach with multiple outputs is a more effective and commonly used technique for word prediction tasks. It provides a more expressive representation, facilitates training, and allows for better interpretation and generalization capabilities compared to using a single output with the target value as the index of the next word.

2. Why do we use softmax and categorical cross entropy for the activation function and loss function?

- for the activation function: The softmax activation function is used to convert the output of a neural network into a probability distribution over multiple classes. It ensures that the predicted probabilities sum up to 1 and are in the range [0, 1]. in this lab, we need to use a probability model to choose which output with the maximum probability based on the current series of words, so softmax function is the best choice for us to get the most suitable word. Another advatage for softmax function is that it is differential and it is easier for model to optimize the parameter together with cross-entrop loss function through gradient-based optimization algorithms like stochastic gradient descent (SGD).
- for the loss function: the categorical cross-entropy measures the dissimilarity between the predicted probabilities (y_pred) and the true labels (y_true). It penalizes large errors and encourages the model to assign high probabilities to the correct classes which is suitable here. And categorical cross-entropy is a differentiable function, enabling backpropagation and gradient-based optimization to update the model's parameters. Another important thing is that the categorical cross-entropy loss aligns with the probabilistic interpretation of softmax. It measures the dissimilarity between the predicted probabilities and the true class probabilities, driving the model to improve its predictions towards the true distribution.(cross-entropy loss function and softmax activation function have a good cooperation)