Language Modeling

Amirmahdi Aramideh

August 2024

1. Problem Statement

Language modeling is a core task in natural language processing (NLP) that involves predicting the next word in a sequence given the context of previous words. This task is critical in applications like machine translation, text generation, and autocomplete systems.

However, language modeling presents significant challenges, particularly with capturing both **short-term** and **long-term dependencies** in text. Many traditional models, such as n-grams, struggle to handle long-range dependencies and larger contexts due to their limited scope. More advanced models like **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM)** networks address this issue by learning from sequences of arbitrary length. However, even these models can suffer from overfitting and training inefficiencies, especially when dealing with large datasets like WikiText-2.

In this project, two models were developed:

1. A **base model** using a standard LSTM architecture.
2. An **improved model** using techniques from the **AWD-LSTM** (ASGD Weight-Dropped LSTM) paper to further enhance performance and generalization.

2. Base Model: LSTM

The base model is built on the **Long Short-Term Memory (LSTM)** architecture. LSTM networks are a type of recurrent neural network (RNN) specifically designed to handle long-term dependencies in sequential data, making them ideal for language modeling.

Why LSTM?

RNNs can theoretically capture dependencies between words over time, but they suffer from the **vanishing gradient problem**, which makes it difficult to learn and remember information from earlier time steps in long sequences. LSTMs address this by incorporating a **memory cell** mechanism with three gates—input, forget, and output—that regulate the flow of information, allowing the model to **selectively retain** or **discard information** as needed. This enables LSTMs to handle longer-term dependencies more effectively than traditional RNNs.

Architecture:

1. **Input Embeddings**:
   - The input to the model is a sequence of word embeddings. These embeddings represent each word as a dense vector of fixed size, capturing semantic relationships between words (e.g., "king" and "queen" might have similar embeddings).
2. **LSTM Layer**:
   - A two-layer LSTM processes the sequence of word embeddings. Each LSTM cell takes the previous word's hidden state as input, along with the current word's embedding, and updates its memory and hidden states based on the context from the sequence.
3. **Fully Connected Layer**:
   - The final hidden state from the LSTM is passed to a fully connected (dense) layer, which converts it into a probability distribution over the vocabulary. This allows the model to predict the likelihood of each possible word in the vocabulary as the next word in the sequence.

Training:

- The model was trained on the **WikiText-2** dataset, a popular benchmark for language modeling tasks.

- **Perplexity** was used as the evaluation metric, where a lower perplexity score indicates better performance in predicting the next word in the sequence.
- Several hyperparameters were optimized, including the number of LSTM layers, learning rate, batch size, sequence length, and dropout rates.

Base Model Results:

- **Validation Perplexity**: 128.3
- **Test Perplexity**: 121.7 These results indicate that the base LSTM model was able to capture important patterns and dependencies in the text. However, the perplexity scores suggest that there was room for improvement in terms of the model's ability to generalize, especially on the validation set.

3. Dataset: WikiText-2

The **WikiText-2** dataset is a widely used corpus for language modeling and other NLP tasks. It consists of **cleaned, pre-tokenized text** from Wikipedia articles, making it a challenging dataset for models to learn long-range dependencies due to its diverse vocabulary and complex sentence structures.

- **Training set**: Approximately 2 million tokens.
- **Validation set**: Several hundred thousand tokens.
- **Test set**: Several hundred thousand tokens.
- **Vocabulary size**: 33,000 words.

The dataset provides a good balance of formal and informal language, making it suitable for testing the ability of language models to predict the next word in various contexts. The complexity of the dataset requires the model to generalize well across different domains and writing styles.

4. Use of AWD-LSTM

In the second phase of the project, we implemented techniques from the **AWD-LSTM** model, introduced by Stephen Merity et al. in their paper "Regularizing and Optimizing LSTM Language Models" (2017). The AWD-LSTM architecture is designed to improve the performance of LSTM networks for language modeling tasks by addressing common challenges such as overfitting, gradient instability, and inefficient training dynamics.

LSTMs are highly effective at modeling long-range dependencies in sequences, but they are prone to overfitting, especially on small or medium-sized datasets. Furthermore, LSTMs can face training challenges like **vanishing/exploding gradients**, which make it difficult for the network to learn effectively over long time steps. The AWD-LSTM model incorporates various **regularization** and **optimization** strategies to mitigate these issues, allowing for better generalization and more efficient training.

Motivation for AWD-LSTM

The core idea behind AWD-LSTM is to enhance the training and generalization capacity of LSTMs for language modeling by applying advanced regularization techniques and more effective optimization methods. Traditional LSTMs can overfit easily and tend to be sensitive to hyperparameters like learning rate and dropout rate. This makes them more difficult to train, especially on challenging datasets like WikiText-2, which requires balancing both short- and long-term dependencies in natural language.

To address these challenges, AWD-LSTM introduces a number of key innovations that make LSTM training more robust and enable better generalization to unseen data.

Key Techniques in AWD-LSTM

AWD-LSTM builds on traditional LSTM architecture but incorporates several **regularization** and **stabilization**techniques to improve performance. Below are the main features and how they contribute to better results in language modeling:

1. **Weight Dropping (Dropout on Recurrent Weights)**:
    - **Traditional Dropout**: Dropout is a well-known regularization technique that prevents overfitting by randomly setting some neuron activations to zero during training, which forces the network to become more robust and prevents co-adaptation of neurons. However, in the case of recurrent neural networks like LSTMs, applying dropout in a naive manner can disrupt the sequential memory dynamics.
    - **Weight Dropping**: Instead of applying dropout to the neuron activations, AWD-LSTM applies dropout to the **hidden-to-hidden weights** within the LSTM itself. This is called **Weight Dropping**, and it helps regularize the recurrent connections by preventing them from memorizing the training data too closely. Weight dropping encourages the LSTM to learn more general features that can transfer better to unseen data.
        - **Key Benefit**: This method allows the LSTM to retain the long-term memory required for language modeling tasks while still benefiting from the regularization effects of dropout.
2. **ASGD (Averaged Stochastic Gradient Descent)**:
    - **Traditional Optimization Issues**: When training neural networks, particularly LSTMs, on large datasets, the choice of optimizer plays a critical role in convergence and generalization. Standard optimizers like SGD (Stochastic Gradient Descent) or Adam may lead to **overfitting** or converge to **sharp minima**, where the model performs well on training data but poorly on test data.
    - **ASGD**: To address this, AWD-LSTM employs **Averaged Stochastic Gradient Descent (ASGD)**. This method averages the model's parameters over time, which smooths the model's final parameter values and helps avoid overfitting by converging to **flatter minima** in the loss landscape. Flatter minima are associated with better generalization because small perturbations in the model's parameters don't lead to significant increases in error.
        - **Key Benefit**: ASGD allows the model to maintain higher performance on unseen data by reducing the likelihood of overfitting to the training data.
3. **DropConnect**:
    - **Standard Dropout** in fully connected layers works by zeroing out neuron activations, but **DropConnect**takes this a step further by randomly zeroing out the connections (weights) between neurons instead. This technique is applied to the weights between layers, making the model more robust by forcing it to learn more diverse representations.
    - In AWD-LSTM, DropConnect is specifically applied to the **LSTM's hidden-to-hidden connections**, making the recurrent connections more

robust and less prone to overfitting.

- **Key Benefit**: By regularizing the connections between neurons rather than the neurons themselves, DropConnect ensures that the network doesn't rely too heavily on any single connection, improving its ability to generalize.

4. **Non-monotonic Triggering for Early Stopping**:
   - **Traditional Early Stopping**: One common approach to prevent overfitting is early stopping, where training is stopped when performance on a validation set no longer improves. However, this can be tricky because the validation loss might temporarily increase or decrease during training due to the complexity of the model and the dataset.
   - **Non-monotonic Triggering**: AWD-LSTM uses **non-monotonic triggering**, a more flexible form of early stopping that allows the model to continue training as long as the performance eventually improves, even if there are temporary dips in validation performance. This helps the model avoid premature stopping while still preventing overfitting.
     - **Key Benefit**: Non-monotonic triggering allows the model to explore more during training and avoid overfitting by only stopping when there is clear evidence that further training won't improve performance.
5. **Variational Dropout**:
   - **Standard Dropout** can be too aggressive when applied to LSTMs, where it is crucial to maintain consistent internal state across multiple time steps. AWD-LSTM implements **Variational Dropout**, which applies dropout in a consistent manner across all time steps in the sequence. The same dropout mask is used for each time step, ensuring that the model retains temporal consistency in its hidden states while still benefiting from the regularization effects of dropout.
   - This technique is applied to the **recurrent connections** between LSTM layers, allowing the model to learn more robust representations over time.
     - **Key Benefit**: Variational dropout reduces the risk of overfitting without disrupting the temporal dynamics of the LSTM.
6. **Weight Tying**:
   - **Weight tying** is a technique where the weights of the input embedding layer and the output softmax layer are shared. This reduces the number of parameters in the model, making it more memory-efficient and less prone to overfitting. Since the input and output layers essentially work with the same vocabulary, sharing their weights makes sense and improves training efficiency.
   - By tying the weights between these two layers, the model

learns more compact representations, reducing overfitting and improving convergence.

    - **Key Benefit**: Reduces the overall parameter count, preventing the model from becoming too complex and overfitting the training data.

Hyperparameters:

In AWD-LSTM, several hyperparameters were fine-tuned to balance regularization and performance:

- **Embedding Dropout**: 0.1 – Regularization applied to the word embeddings.
- **Input Dropout**: 0.65 – Dropout applied to the inputs of LSTM layers.
- **Hidden Layer Dropout**: 0.3 – Dropout applied to the outputs of LSTM layers.
- **Output Dropout**: 0.4 – Dropout applied to the final output layer.
- **Weight Decay**: 1.2e-6 – Used to penalize large weights, further preventing overfitting.
- **Gradient Clipping**: 0.25 – To prevent exploding gradients, which can destabilize training.

These hyperparameters were optimized through experimentation to ensure that the AWD-LSTM model could maintain generalization while still achieving high performance on the language modeling task.

How AWD-LSTM Improves Performance

- **Reduced Overfitting**: Techniques like weight dropping, variational dropout, and DropConnect ensure that the LSTM does not memorize the training data but instead learns generalized patterns that can transfer to unseen data.
- **Improved Stability**: ASGD and gradient clipping help stabilize training by preventing the model from converging to sharp minima, ensuring that it generalizes well to validation and test data.
- **Efficient Training**: Weight tying and non-monotonic early stopping reduce the number of parameters and allow for more efficient training without sacrificing performance.
-

5. Results

Base Model (LSTM) Results:

- **Validation Perplexity**: 128.3
- **Test Perplexity**: 121.7

The base LSTM model performed reasonably well, capturing many dependencies in the text. However, the high perplexity on the validation set indicated that the model could overfit the training data, showing less generalization on unseen data.

AWD-LSTM Model Results:

- **Validation Perplexity**: 80.27
- **Test Perplexity**: 77.11

The AWD-LSTM model significantly outperformed the base LSTM model, achieving much lower perplexity on both validation and test sets. This improvement demonstrates the effectiveness of the regularization techniques and optimization strategies introduced in the AWD-LSTM paper.

Comparison of Model Results:

Model	Parameters	Validation Perplexity	Test Perplexity
LSTM (2 layers)	24.8M	128.3	121.7
AWD-LSTM (3 layers)	31.75 M	80.27	77.11

These results confirm that the AWD-LSTM model's additional techniques, such as weight-dropping, variational dropout, and weight tying, contributed to improved generalization and overall performance.

Sample Output from AWD-LSTM:

The AWD-LSTM model was also tested by generating text sequences based on given prompts. Here are a few examples:

- **Prompt**: "I think this movie is..." **Generated Output**: "I think this movie is the most beautiful thing I've ever seen."

- **Prompt**: "In this year..." **Generated Output**: "In this year, as he was in the first game of the season, and he was named the team's captain."

These outputs demonstrate the model's ability to generate coherent and contextually appropriate text.