CAP 6640 – Natural Language Processing **Spring 2025**
Homework #5 Due date: March 13, 2025

Problem 1:

1. **Bag of Vectors:** The Bag of Vectors model represents text as an unordered collection of word embeddings, making it computationally efficient for document classification. While it performs well in tasks that do not require sequential dependencies, its major drawback is the loss of word order and contextual relationships, limiting its effectiveness in complex NLP applications such as machine translation or sentiment analysis. Despite this, performance can be improved by introducing ReLU layers to add non-linearity, but it remains fundamentally constrained by its lack of sequence awareness.
2. **Window Model:** The Window Model improves upon Bag of Vectors by considering a fixed number of surrounding words, allowing it to capture local context effectively. This makes it particularly useful for single-word classification tasks such as POS tagging and NER. However, since it only considers a small window of words at a time, it struggles with long-range dependencies, making it unsuitable for applications that require a broader contextual understanding.
3. **CNNs:** CNNs process text by applying convolutional filters over word embeddings, allowing them to detect meaningful n-gram patterns such as sentiment phrases or topic-specific keywords. They excel in text classification tasks in NLP and many others as they are highly parallelizable and efficient on GPUs. However, CNNs struggle with long-range dependencies since they primarily focus on local patterns, and they require padding to handle varying sentence lengths, making them less suitable for tasks requiring a strong grasp of word order and context, such as machine translation.
4. **RNNs:** Unlike CNNs, RNNs are designed to process text sequentially, maintaining a hidden state that carries information from previous words, making them highly effective for tasks like machine translation, speech recognition, and text generation. This structure allows RNNs to model long-term dependencies, providing a more contextual understanding of language. However, they suffer from slow training speeds due to their sequential nature, making them difficult to parallelize. Additionally, they are prone to the vanishing gradient problem, which hinders their ability to capture dependencies over long sequences unless enhancements like LSTMs or GRUs are applied.

Problem 2:

Gating mechanisms in NNs control the flow of information by selectively allowing or blocking certain values. Vertical and horizontal gating are two approaches used to regulate information within deep learning architectures.

1. **Horizontal Gating:** Horizontal gating regulates information flow across time steps in sequential models such as RNNs, LSTMs, and GRUs. It helps models retain or discard past information at each time step, making it essential for handling long-term dependencies in sequential data. A key example is the forget gate in LSTMs, which determines how much past information should be kept using the formula $f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$. Horizontal gating is crucial in machine translation, speech recognition, and NLP tasks, where preserving context over time is necessary. However, training these models on very long sequences can be challenging, sometimes requiring attention mechanisms to improve performance.
2. **Vertical Gating:** Vertical gating controls information flow across layers in deep neural networks, such as CNNs and ResNets, helping regulate how much information passes from one layer to the next. It is commonly used in Highway Networks and ResNets, where gates determine whether an input is transformed or passed directly to the next layer, aiding in gradient flow and feature learning. A key example is the Highway Network, where the transform gate $T(x)$ modulates the proportion of tranformed information versus unchanged input using the formula $y = T(x) \cdot H(x) + (1 - T(x)) \cdot x$. Vertical gating is particularly useful in deep architectures to prevent vanishing gradients, making it effective for image classification and other deep learning tasks. However, it does not model temporal dependencies, as it only functions across layers, not time steps.

Problem 3:

Batch normalization is a widely used technique for CNNs to improve training stability and efficiency. It works by normalizing activations across a batch, ensuring that NN outputs have zero mean and unit variance, which helps to prevent extreme activations that could slow down learning. Additionally, batch normalization includes trainable scale γ and shift β parameters, allowing the model to adjust the normalization dynamically rather than simply standardizing all activations.

One of the main advantages of batch normalization is that it reduces internal covariate shift, a problem where changing distributions of activations across layers make training less stable. By normalizing inputs before they are passed to the next layer, batch normalization ensures that each layer receives consistently scaled data, reducing the burden on later layers to adapt to distributional shifts. This leads to faster and more stable training. Furthermore, it also mitigates vanishing and exploding gradient problems, which commonly occur in deep networks. By keeping activations within a reasonable range, it prevents gradients from shrinking too much or growing uncontrollably, which can significantly slow down or disrupt training. In addition, it makes parameter initialization less critical, since it automatically rescales outputs, reducing the need for carefully chosen weight initializations. Lastly benefit is that it allows higher learning rates without causing instability, making hyperparameter tuning more forgiving. Typically, deep NLP models require very small learning rates to avoid divergence, but batch normalization stabilizes activations, enabling the use of more aggressive learning rates that speed up convergence.

Problem 4:

Traditionally, sequence models like LSTMs and GRUs have dominated NLP due to their ability to handle sequential dependencies. However, very deep CNNs which are inspired by image-processing architectures like VGG can be highly effective for NLP, particularly in text classification tasks.

Instead of relying on word embeddings and sequential processing, very deep CNNs extract features directly from character-level representations. The model consists of multiple stacked convolutional layers, which progressively refine text representations, much like deep CNNs do for images. The deeper the network, the more hierarchical and abstract the extracted linguistic features become.

Very deep CNNs apply multiple convolutional layers, followed by pooling layers, to extract patterns in text at various levels. This process is similar to how deep CNNs for vision detect simple edges in early layers and complex objects in deeper layers. In NLP, this results in low-level character n-grams at shallow layers and high-level semantic representations at deeper layers. Local pooling at each stage reduces the temporal resolution while retaining key information. Stacking convolutional layers enables multi-scale text representation, allowing the model to capture both local and long-range dependencies. This fully convolutional architectures allows for parallel processing, significantly speeding up training. Deeper CNN architecture improves feature extraction, making the model more effective at capturing linguistic structures. By leveraging hierarchical feature learning, very deep CNNs can outperform sequence models in text classification tasks, especially when trained on large datasets.

Problem 5: Describe how QRNNs work conceptually and mathematically, including their purpose and a supporting figure.

Problem 6: Explain the role of subword information in language understanding.

Problem 7: Describe fully character-level neural machine translation.

Problem 8: Explain byte pair encoding (BPE).

Problem 9: Compare bottom-up and neural summarization.

Problem 10: Discuss a method for handling irrelevant responses, generic outputs, repetition, and inconsistent persona in natural language generation.