SEQUENCE ANALYSIS

When working with variable-length inputs, especially sequences, it's important to adapt your deep learning model design.
Unlike fixed-size data like images, sequences have varying lengths, making it challenging to apply traditional neural network architectures.

Key Challenges of variable length input:

1. Sequences can have different lengths, making it difficult to define a fixed-size input layer.
2. Elements in a sequence are often dependent on previous elements, requiring the model to capture these relationships.

PPT

- When dealing with fixed-size data like images from MNIST, CIFAR-10, and ImageNet, traditional deep learning models work well.

- However, when it comes to variable length inputs or sequences, such as reading the morning newspaper, making a bowl of cereal, listening to the radio, watching a presentation

- We need to be more creative in designing our deep learning models.

Approaches to Handling Variable Length Inputs

- **Recurrent Neural Networks (RNNs)**: RNNs are a type of neural network designed to handle sequential data. They can learn patterns in sequences of varying lengths.

- **Long Short-Term Memory (LSTM) Networks**: LSTMs are a type of RNN that can learn long-term dependencies in sequences. They are well-suited for tasks like language modeling and speech recognition.

- **Transformers**: Transformers are a type of neural network architecture that can handle sequential data. They are particularly well-suited for tasks like machine translation and text summarization.

FFN (PPT)

Feed-forward neural networks (FNNs) are a fundamental architecture in deep learning, but they have limitations when it comes to analyzing sequences.

1. Feed-forward neural networks are structured to accept inputs of a fixed size. FNNs cannot dynamically adjust to different input sizes.
2. It's even possible to deal with smaller inputs by padding zeros to the end of the input until it's the appropriate length.
3. If the input sequence exceeds the size of the input layer, the FNN can only process a limited number of elements.
4. However, if the input exceeds the size of the input layer, naively using the feedforward network no longer works.

Strategies to "Hack" Feed-Forward Networks

Here are your 4 points explained in a simple way:

1. **FFN for POS Tagging**: We use a special kind of computer program to identify the part of speech (like noun, verb, adjective) for each word in a sentence.

2. **Understanding Word Usage**: By identifying the POS, we can understand how words work together in a sentence and what they mean.

3. **Seq2Seq Problems**: This is a type of problem where we need to change one sequence of things (like words) into another sequence (like POS tags). Other examples include translating languages and summarizing text.

4. **Developing an Algorithm**: We're working on creating a computer program that can understand the meaning of words in a sentence, which is a step towards more advanced language understanding tasks.

Summary

In summary, a FFN is a foundational model in machine learning used for various tasks, including **classification**, **regression**, and **feature extraction**. Its straightforward architecture makes it a good starting point for many applications, including **NLP** tasks like part-of-speech tagging.

How To Use A Feed-Forward Neural Network (Ffn) For (Seq2seq) Tasks, Specifically For Predicting Part-Of-Speech (Pos) Tags In A Text ignoring (LTD)

1. Long-Term Dependencies: Long-term dependencies between distant words are not crucial for predicting POS tags.

2. Individual Tag Prediction: The model predicts each POS tag individually rather than all at once.

3. Limited Context: Each tag is predicted using a fixed-length context of the current word and a few preceding words.

4. Simplified Processing: This approach simplifies the task for feed-forward networks by reducing the amount of information processed at once.

5. Distributed Representations: The technique leverages distributed vector representations to enhance the prediction process.

CONTEXT WINDOW

- **Size :** When predicting the POS tag for the i-th word in the input, we use the words from the (i-n+1)-th to the i-th as input, referred to as the context window.
- **Move the Context Window :** To process the entire text, we start at the beginning and move the context window one word at a time, predicting the POS tag for the rightmost word until we reach the end.
- **Word Embeddings :** We leverage word embeddings, using dense vector representations of words that capture semantic relationships instead of one-hot vectors.
- **Reduce Dimensionality :** This approach reduces the number of parameters in our model and speeds up learning.

The passage outlines the architecture and training process for implementing a Part-of-Speech (POS) tagger using a feed-forward neural network. (SLIDE 11)

1. Network Architecture:

- Input Layer: The network uses a 3-gram context window, meaning it considers three consecutive words at a time to predict the POS tag for the target word. This allows the model to capture local context effectively.

- Word Embeddings: The model employs 300-dimensional word embeddings. Each word is represented as a dense vector of 300 values, which helps capture semantic relationships between words. With a 3-gram context window, the total input size becomes 900 dimensions (3 words × 300 dimensions).

2. Hidden Layers:

- The feed-forward network consists of two hidden layers:
- The first hidden layer has 512 neurons, The second hidden layer has 256 neurons.
- These layers allow the model to learn complex patterns and relationships in the data.

3. Output Layer:

- The output layer uses a softmax activation function to calculate the probability distribution of the POS tags. This layer outputs probabilities for 44 possible POS tags.

4. Optimizer:

- The model uses the Adam optimizer with default hyperparameter settings.
- The model is trained for a total of 1,000 epochs.
- Batch Normalization is leveraged for regularization. This technique helps in reducing internal covariate shift.

Word2vec using Gensim

- The passage describes the process of using pre-trained word embeddings from Google News with the Gensim library in Python.

- The model utilizes pre trained word embeddings generated from the Google News dataset.

- It includes vectors for 3 million words and phrases and was trained on roughly 100 billion words.

- We can use the gensim Python package to read the dataset.

```
from gensim.models import Word2Vec
model =Word2Vec.load_word2vec_format('/path/to/googlenews.bin',binary=True
```

- To optimize memory usage, especially during debugging or experimentation, the passage suggests **caching** a relevant subset of the vectors to disk.

- Caching allows the program to avoid loading the entire dataset into memory every time it runs, which can be resource-intensive and slow.

CoNLL-2000 POS dataset

- The gensim model contains 3 million words, which is larger than our dataset.
- For the sake of efficiency, we'll selectively cache word vectors for words in our dataset and discard everything else.
- To figure out which words we'd like to cache, we download the POS dataset from the CoNLL-2000 task.
- To match the formatting of the dataset to the gensim model, we'll have to do some preprocessing.
  • replace digits with '#' characters
  • combine separate words into entities where appropriate
  • utilize underscores where the raw data uses dashes

Loading words in LevelDB & Building dataset object

- The gensim model contains 3 million words, which is larger than our dataset.

Beam Search

- Beam search, a search algorithm commonly used in NLP tasks, particularly in sequence prediction models like SyntaxNet.

- We generally leverage beam searches in situations like SyntaxNet, where the output of our network at a particular step influences the inputs used in future steps

- beam search is a more sophisticated search strategy that improves upon greedy methods by maintaining multiple hypotheses at each step.
  By considering the top b most likely sequences and their probabilities, beam search allows for better decision-making in models like SyntaxNet.

A Case for Stateful Deep Learning Models

In the realm of sequence analysis, traditional feed-forward neural networks often struggle with capturing the complexities of long-term dependencies. While various techniques have been employed to adapt these networks for sequential data, such as beam searching and global normalization, the fundamental challenges remain.

<u>Limitations:</u>

Ignoring Long-Term Dependencies:

Feed-forward networks often overlook long-term dependencies in sequence tasks (e.g., Part-of-Speech tagging). This can result in lost contextual information, leading to suboptimal performance.

One-to-One Mapping Assumption:

Many traditional models assume a direct one-to-one mapping between input and output sequences. For instance, in dependency parsing, the model reformulates the problem to create a direct correlation between input configurations and output actions, which may not always be feasible.

Cases Having No One-to-One Mapping

There are several scenarios where the task cannot be simplified to a one-to-one mapping between input and output sequences. Here are a few examples:

- **Sentiment Analysis:** The model needs to analyze the entire input sequence (e.g., a review) to determine the overall sentiment (positive or negative). There is no direct mapping between individual words and the sentiment label.

- **Image Captioning:** The model takes a complex input (an image) and generates a descriptive sentence. There is no obvious mapping between pixels in the image and words in the sentence.

- **Machine Translation:** Translating sentences from one language to another (e.g., English to French) requires understanding the context and nuances of the input sentence, which cannot be reduced to a simple one-to-one mapping between words.

In these cases, traditional models that rely on one-to-one mappings are insufficient, and more complex models that can capture the relationships between input and output sequences are needed.

ARC STANDARD SYSTEM

Arc-standard system is used to analyze the structure of a sentence by figuring out word relationships using a series of actions.

- Words are placed in two areas: a stack (for working with words) and a buffer (for waiting words).

  Three possible actions are:
- Shift: Move a word from the buffer to the stack.
- Left Arc: Create a connection where the top word on the stack depends on the word below it.
- Right Arc: Create a connection where the word below the top depends on the top word.
- The goal is to figure out how all the words in the sentence connect by using these actions step-by-step.

LOCAL VS GLOBAL:

Here's a simplified explanation of **local normalization** and **global normalization**:

Local Normalization:

- In **local normalization**, the network decides the best action (like Shift or Arc) based on the current situation.
- **I**t gives each possible action a score, which is converted into probabilities using a **softmax** layer (so the total probabilities add up to 1).
- The goal is to make the network output **high probability (1)** for the correct action and **low probability (0)** for incorrect ones.
- The **cross-entropy loss** helps the network learn to make better predictions by adjusting these probabilities.

Global Normalization:

- In **global normalization**, instead of assigning probabilities for each action separately, the network looks at the **entire sequence of actions**.
- **I**t adds up the scores for each sequence of actions (hypothesis) and then applies the softmax layer.
- The goal is to select the best sequence by looking at all possible action sequences, not just individual actions.
- The same **cross-entropy loss** can be used to improve the network's predictions, but it's applied to the entire action sequence rather than each individual action.

In short:

- **Local**: Focuses on choosing the best action at each step, normalizing scores per action.
- **Global**: Looks at the entire sequence of actions and then normalizes the scores for all sequences.

PROBLEM OF GLOBAL OPTIMIZATION STRATEGY (SIMPLIFIED)

1. Challenge:
   - Intractably large number of hypothesis sequences.
   - Example: For a sentence of length 10 with 15 possible actions, there are 10^15 possible sequences.
2. Solution:
   - Use beam search with a fixed beam size to reduce complexity.
   - Continue until:
     1. End of the sentence, or
     2. The correct action sequence is no longer in the beam.
3. Loss Function:
   - Maximizes the score of the correct sequence.
   - Pushes the “gold standard” sequence higher relative to others in the beam.