Natural Language Processing (NLP) of just text, including transformers and Large Language Models (LLMs)

Adam Boult (www.bou.lt)

July 11, 2025

Contents

Preface

This is a live document, and is full of gaps, mistakes, typos etc.

Part I

Applications for time series models

Chapter 1

Data cleaning: Text

1.1 Cleaning categorial data

1.1.1 One Hot Encoding

1.2 Cleaning text data

1.2.1 Bag-of-words

1.2.2 N-grams

Introduction

We can add start and end of sentence markets. * and STOP

Generally remove punctuation

1.2.3 Feature hashing

Chapter 2

Text prediction

2.1 Text

Chapter 3

Text translation

3.1 Text

Chapter 4

Natural Language Processing (NLP)

4.1 Other

4.1.1 Probabilistic language models

Introduction

Probabilistic language models can predict future words given a history of words.

This can be used for predictive text. For example if a user types "Did you call your" we may want to estimate the probability that the next word is "child".

We can state this problem:

$P(child|did\ you\ call\ your)$

By definition this is:

$$P(child|did\ you\ call\ your) = \frac{P(did\ you\ call\ your\ child)}{P(did\ you\ call\ your)}$$

We can estimate each of these:

$$P(did\ you\ call\ your\ child) = \frac{|did\ you\ call\ your\ child|}{|5\ word\ sentences|}$$

$$P(did\ you\ call\ your) = \frac{|did\ you\ call\ your|}{|4\ word\ sentences|}$$

Data requirements

This needs a large corpus, which may not be practical.

Additionallly, the words must be indexed, and not simply stored as a bag of words.

Decomposition

We can decompose the probabilities using the chain rule.

$P(did\ you\ call\ your\ child) = P(did)P(you|did)...P(child|did\ you\ call\ your)$

$P(w_1, ..., w_k) = \prod_k p(w_k|w_1, ..., w_{k-1})$

N-grams

We can simplify the decomposition using the Markov assumption:

$P(w_k|w_1, ..., w_{k-1}) = P(w_k|w_{k-1})$

This is a 1-gram.

We can do this for n words back. This is an n-gram.

Smoothing

We can use smoothing to address small corpuses.

$P(did\ you\ call\ your\ child) = \dfrac{|did\ you\ call\ your\ child| + 1}{|5\ word\ sentences| + V}$

$P(did\ you\ call\ your) = \dfrac{|did\ you\ call\ your| + 1}{|4\ word\ sentences| + V}$

For some value V.

Perplexity

We can compare probabilistic language models using perplexity.

We can then choose the model with the lowest perplexity.

$perplexity(w_1, w_2, ..., w_n) = P(w_1, w_2, ..., w_n)^{-\dfrac{1}{n}}$

We can expand this:

$perplexity(w_1, w_2, ..., w_n) = \prod_i P(w_i|w_1, ..., w_{i-1})^{-\dfrac{1}{n}}$

Depending on which n-gram we use we can then simplify this.

4.1.2 Word2vec

4.1.3 Latent Semantic Analysis

4.2 Machine translation

4.2.1 Machine translation

Part II

Linguistics

Chapter 5

Comparative method

Chapter 6

Internal reconstruction

Chapter 7

Universal grammar

Part III

Architectures for sequences

Chapter 8

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)

8.1 Simple recurrent neural networks

8.1.1 Simple Recurrent Neural Networks (RNNs)

Introduction

Recurrent Neural Networks (RNN) are an alternative to feedforward networks.

These have loops.

Motivation

We have inputs which are not independent. For example speech input, where each input is a the recording for a length of time.

Unrolling RNNs

The activation unit takes the input, and an outcome from the previous activation unit. It then performs its activation function.

This allows information to be kept across time.

However this degrades, and relevant information was from much earlier, it will be lost.

8.1.2 Backpropagation Through Time (BPTT)

We can do backpropagation on the unrolled network, backpropagating over time.

8.2 Long Short-Term Memory (LSTM)

8.2.1 Long Short-Term Mmemory (LSTM)

Introduction

These are a more complex RNN architecture.

Cell state

Each cell has as an as input the cell state from the previous cell C_{t-1}

The LSTM cell updates the cell state to C_t and pushes it to the next cell.

Other inputs to the cell

We have x_t, the input of the cell, and h_{t-1}, the output of the previous cell.

Cell output and the output gate

We run an activation function on the cell state C_t to get a candidate output.

We multiply this by the outcome of the output gate to get the actual result.

The input gate

We create a candidate change to the state.

We multiply this by the input gate value, and add it to the state.

The forget gate

This is a multiplication factor. What % of the state should be removed?

8.3 Variants

8.3.1 Peephole LTSM

8.3.2 Gated Recurrent Units (GRUs)

8.4 Forecasting with recurrant neural networks

8.4.1 Introduction

8.5 Other

8.5.1 Attention and Neural Turing Machines

Chapter 9

Recurrent Neural Network (RNN) encoders and decoders

9.1 Recurrent Neural Network (RNN) encoders and decoders

9.1.1 Recurrent Neural Network (RNN) encoders

Final output is the encoding.

The end of the sequence is identified through an End-Of-Sequence token.

seq2seq

9.1.2 Recurrent Neural Network (RNN) decoders

Introduction

We take the encoded vector and pass this through to the decoder. This spits out decoded output.

As we output a word, the word (and previous words) are sent as inputs to the following RNN cells.

Encoding the outputs

As we create outputs, we can pass this as an encoded vector in the target language.

Chapter 10

Attention in neural networks and Transformers, and Bidirectional Encoder Representations from Transformers (BERT) and Generative Pre-trained Transformers (GPT)