CS4851/6851 - Intro to Deep Learning

Homework - 6

Student Details:

Level-of-Study	Name	Panther-ID	Email ID
Graduate Student	Kiran Kumar Reddy Donuru	002678089	kdonuru1@student.gsu.edu

1)We can represent the words in a vocabulary with binary vectors that have dimensions of the number of words in the vocabulary and all values set to zero except the one value that corresponds to the index of the given word in the sorted version of this vocabulary. This is the so-called one-in-K or one-hot encoding.

(a) Describe a representation of a document with a vector. (Think of a representation that is based on the one-hot encoding of the words in that document and has the same dimension as a single word (size of the vocabulary).

(b) Explain why this representation is problematic:

(i) Simple sentence or two

(ii) Examples of the problem(s)

(c) Provide at least two more options to fix this problem.

Text modeling is difficult because it is unstructured, and approaches such as machine learning algorithms require well-defined fixed-length inputs and outputs.

Machine learning algorithms cannot operate directly with raw text; the text must first be turned into numbers. Vectors of numbers, to be specific.
A bag-of-words is a text representation that defines the appearance of words in a document. It entails two steps:

- A list of well-known terms.
- A metric for the presence of well-known terms.

Collect data:
It is not the time to enjoy
It must be the worst day.
The light is too bad
Designing the Vocabulary:

- It
- Is
- not
- The
- Time
- to
- enjoy
- Must
- Be
- Worst

- day
- Light
- Too
- bad

Create Document Vectors:

- It =1
- Is =1
- Not =0
- The = 1
- Time =1
- to =1
- enjoy =1
- Must =1
- Be =1
- Worst =0
- Day =1
- Light =1
- Too =1
- Bad =0

Word Hashing:
A hash representation of known terms in our lexicon can be used. This solves the corpus problem since we can specify the size of the hash space, which determines the size of the vector representation of the document.

In the target hash space, words are hashed to the same integer index. The word can then be scored using a binary score or count.

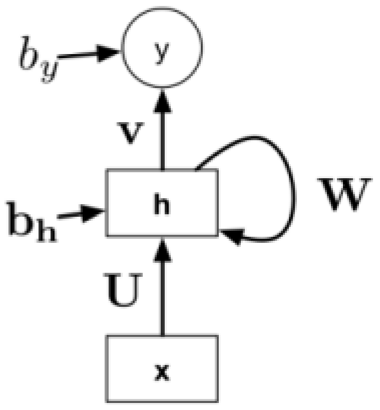
TF-IDF:
One issue with assessing word frequency is that very common terms begin to dominate the page, appearing to the model as rarer but maybe domain specialized words.

One option is to rescale the frequency of words based on how frequently they appear in all texts, such that common terms like "the" that appear frequently in all publications are punished.

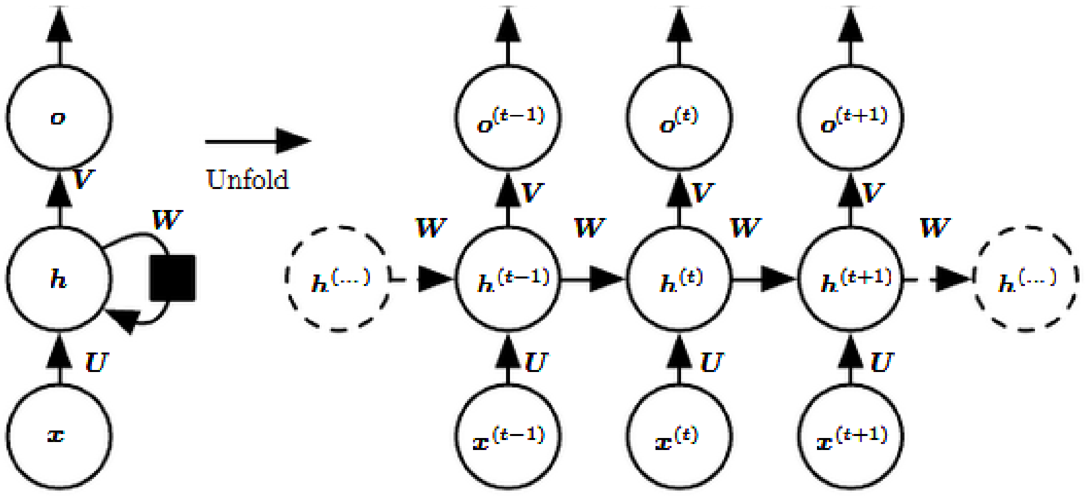
- Frequency of occurrence
- Document Frequency in the Negative

reference:https://machinelearningmastery.com/gentle-introduction-bag-words-model/

2)A recurrent network in Figure 3 takes a sequence of integers as input and at the end of the sequence, the last element produces a number between 0 and 1. What does a 0 mean? What does a 1 mean? Describe which function this network is computing (what is the meaning of this function). Assume all biases are 0, and make sure the hidden state is initialized to 0 as well. Note, that the inputs, the weights, and the hidden state are just scalars in this RNN.



RNN: A recurrent neural network is a type of neural network that is trained to handle a sequence of data with a time step-index. RNNs are frequently preferable for jobs involving sequential inputs, such as voice and language. If you want to anticipate the next word in a phrase in an NLP issue, you must first know what comes before it. RNNs are referred to as recurrent because they do the same task for each element of a sequence, with the result dependent on past calculations.



Input: x(t)is taken as the input to the network at time step t.

Hidden state:h(t) represents a hidden state at time t and serves as the network's "memory." h(t) is computed using the current input and the hidden state from the previous time step.

Weights: A weight matrix U governs the RNN's input to hidden connections, a weight matrix W governs the hidden-to-hidden recurrent connections, and a weight matrix V governs the hidden-to-output connections.

Output: o(t) represents the network's output. In the diagram, I simply added an arrow after o(t), which is also frequently vulnerable to non-linearity, especially when the network has further layers downstream.

Forward Pass:
Before we begin, we will make the following assumptions: For the buried layer, we assume the hyperbolic tangent activation function. We assume the output is discrete as if the RNN were predicting words or letters. A natural method to describe discrete variables is to consider the output o to be the un-normalized log probability of each discrete variable's potential values.
Backward Pass:
The gradient calculation entails making a forward propagation pass from left to right along the graph illustrated above, followed by a backward propagation pass from right to left. Because the forward propagation graph is fundamentally sequential, each time step can only be computed after the preceding one, the runtime is O() and cannot be decreased by parallelization. States calculated in the forward pass must be saved until utilized in the backward phase, therefore the memory cost is also O(. Back-propagation over time refers to the back-propagation method applied to the unrolled graph with an O() cost.
Computing Gradients
We must compute the gradients for our three weight matrices U, V, W, as well as the bias terms b, c, and then update them using a learning rate. The gradient, like regular back-propagation, shows us how the loss changes concerning each weight parameter.
reference:https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85

3)In this problem, you will implement a recurrent neural network that implements binary addition. The inputs are given as binary sequences, starting with the least significant binary digit. (It is easier to start from the least significant bit, just like how you did an addition in grade school.) The sequences will be padded with at least one zero on the end. For instance, the problem 100111 + 110010 = 1011001 (1) would be represented as : (a) Input 1: 1, 1, 1, 0, 0, 1, 0 (b) Input 2: 0, 1, 0, 0, 1, 1, 0 (c) Correct output: 1, 0, 0, 1, 1, 0, 1 There are two input units corresponding to the two inputs, and one output unit. Therefore, the pattern of inputs and outputs for this example would be: Design the weights and biases for an RNN which has two input units, three hidden units, and one output unit, which implements binary addition. All of the units use the hard threshold activation function. In particular, specify weight

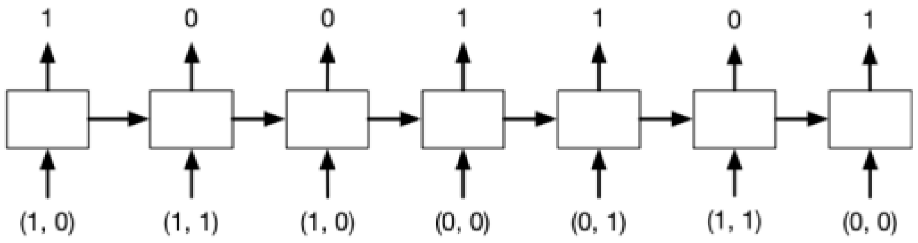
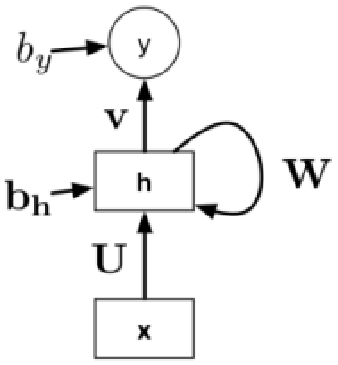


Figure 2: The RNN Binary for Problem 3

matrices U, V, and W, bias vector bh, and scalar bias by for the following architecture:



Hint: In the grade school algorithm, you add up the values in each column, including the carry. Have one of your hidden units activate if the sum is at least 1, the second one if it is at least 2, and the third one if it is 3.

4)We have learned about regularization in image processing. How does regularization help in the context of Recurrent Neural Networks?

The Recurrent Neural Network (RNN) is a neural sequence model that delivers cutting-edge performance on critical tasks such as Mikolov language modeling, speech recognition, and machine translation. It is well understood that successful neural network applications need good regularization. As a result, because big RNNs tend to overfit, practical implementations of RNNs frequently utilize models that are too small. Existing regularization approaches provide only minor gains for RNNs. In this paper, we demonstrate that dropout, when utilized appropriately, significantly minimizes overfitting in LSTMs and test it on three distinct tasks.

REGULARIZING RNNS

The LSTM contains complex dynamics that allow it to easily "memorize" information over a long period of time. A vector of memory cells stores "long-term" memories. Despite the fact that different LSTM designs differ in connection topology and activation functions, all LSTM architectures feature explicit memory cells for storing information over long periods of time. The LSTM can choose whether to overwrite the memory cell, recover it, or save it until the future time step. The following equation describes the LSTM architecture utilized in our investigations. The key contribution of this research is a formula for successfully reducing overfitting by applying dropout to LSTMs. The fundamental notion is to only apply the dropout operator to non-recurrent information, and this number is independent of the number of timesteps covered by the information. The recurrent connections are disrupted by standard dropout, making it difficult for the LSTM to learn to store information for lengthy periods of time. The LSTM can benefit from dropout regularization without loosing its valuable memorizing capacity by not utilizing dropout on recurrent connections.

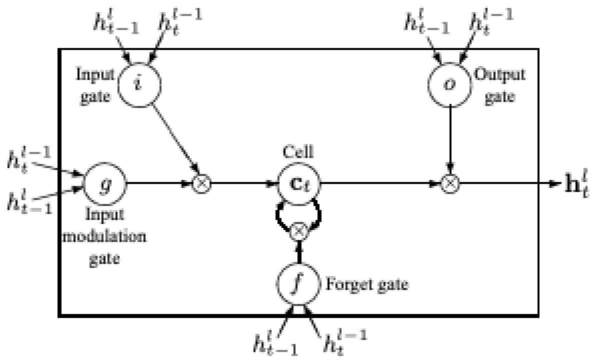
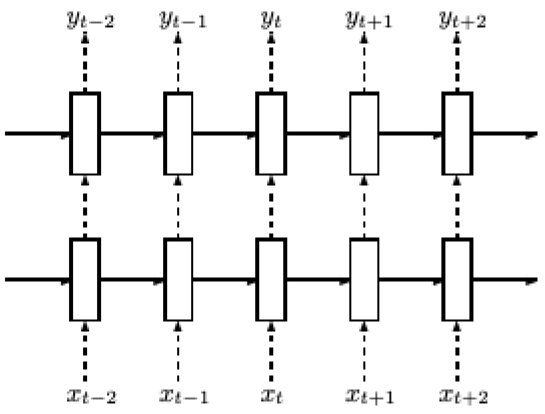


Figure 1: A graphical representation of LSTM memory cells used in this paper (there are minor differences in comparison to Graves (2013)).



Reference: https://arxiv.org/pdf/1409.2329.pdf

5)How is the teacher forcing ratio more accurate than the model output for a sequence of inputs? How can we use the teacher process to parallelize the computation?

Firstly let's try to understand what exactly is **teacher forcing.**

Teacher Forcing: Teacher forcing is a training approach for recurrent neural networks that employ ground truth as input rather than the model output from a previous time step.

Teacher Forcing for Recurrent Neural Networks: Teacher forcing is a technique for rapidly and effectively training recurrent neural network models that take as input the ground truth from a previous time step. It is a network training approach that is essential for the construction of deep learning language models used in machine translation, text summarization, and picture captioning, among other applications.

There are sequence prediction models that use the output from the last time step y(t-1) as input for the model at the current time step X(t). This type of model is common in language models that output one word at a time and use the output word as input for generating the next word in the sequence. When training the model, the same recursive output-as-input approach can be utilized, although it can lead to issues such as slow Convergence, Model shakiness, and Incompetence.

Teacher forcing is a quick and efficient method of training a recurrent neural network that leverages the result of previous time steps as input to the model. However, when the produced sequences differ from what the model saw during training, the technique might result in models that are fragile or constrained when utilized in practice. Because the outputs are probabilistic, this is frequent in the most simple of this sort of model. This form of model application is frequently referred to as an open-loop.
Reference:https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/

6)Write a report on one of the following topics:
(a) Attention Is All You Need {https://arxiv.org/pdf/1706.03762.pdf}
(b) Transformers: {https://arxiv.org/pdf/1910.03771v5.pdf}

I have gone through both the topics but the **Attention Is All You Need** topic seemed interesting to me, so am writing a report on it.

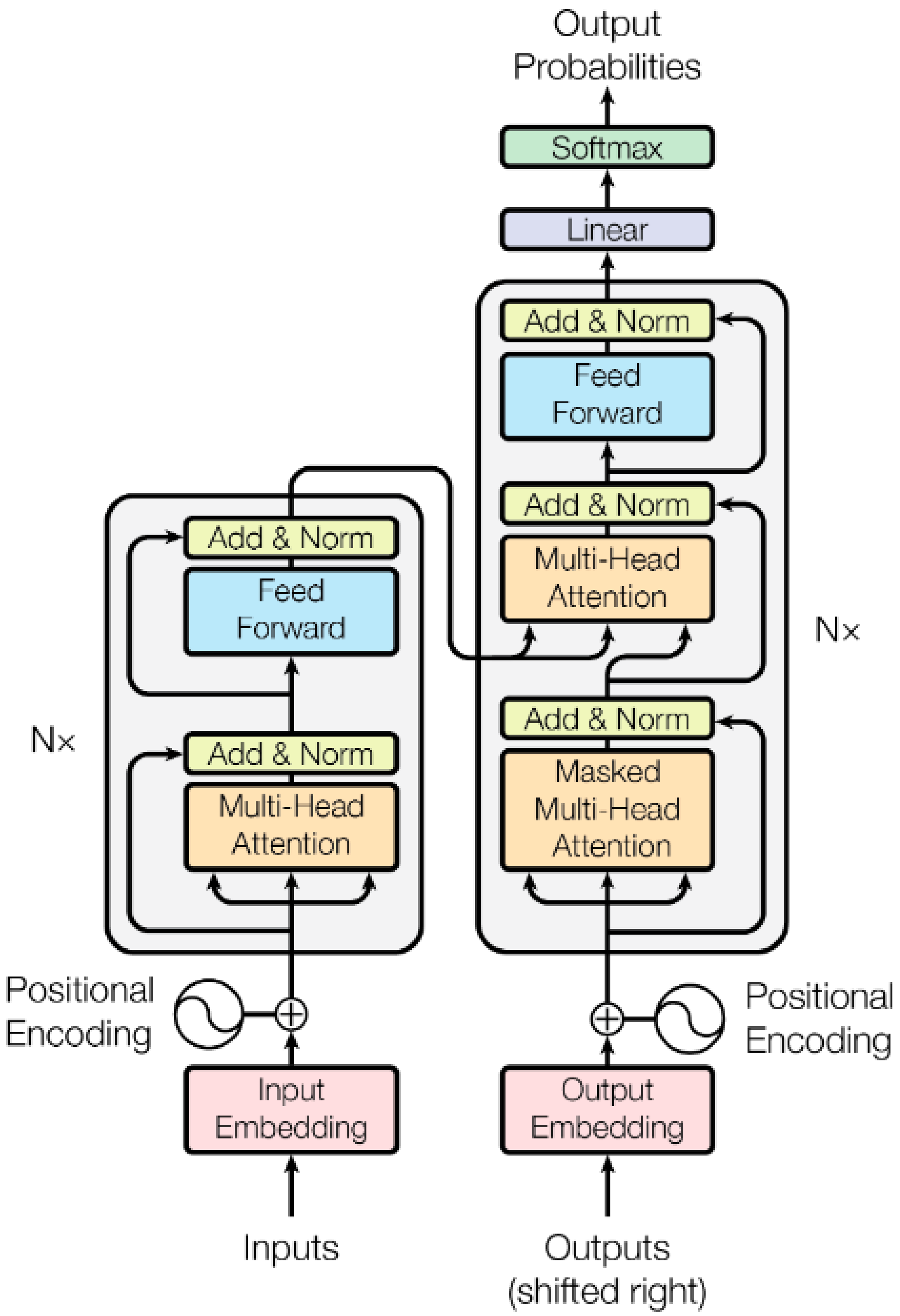
Introduction to Attention Is All You Need:

In sequence modeling and transduction challenges such as language modeling and machine translation, recurrent neural networks have become well-established as cutting-edge techniques. In most cases, recurrent models factor computation along with the symbol locations of the input and output sequences. They produce a succession of hidden states by aligning the locations to steps in computing time. In different tasks, attention mechanisms have become an intrinsic feature of appealing sequence modeling and transduction models, allowing for the modeling of dependencies regardless of their distance in the input or output sequences. The Transformer is a model architecture suggested in this study that avoids recurrence and instead relies exclusively on an attention mechanism to create global relationships between input and output.

Background:
The objective of decreasing sequential processing is also at the heart of the Extended Neural GPU, ByteNet, and ConvS2S, all of which employ convolutional neural networks as their fundamental building blocks, computing hidden representations in parallel for all input and output positions. The Transformer, on the other hand, is the first transduction model to construct representations of its input and output using just self-attention rather than sequence aligned RNNs or convolution.

Model Architecture:

The structure of most competitive neural sequence transduction models is encoder-decoder.
The encoder in this case converts an input series of symbol representations into a sequence of continuous representations. The model is auto-regressive at each phase, taking previously created symbols as an extra input when creating the next. This general design is followed by the Transformer, which employs layered self-attention and point-wise, completely linked layers for both the encoder and decoder.


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Encoder & Decoder Blocks:

Encoder: The vanilla transformer architecture encoder blocks are made up of two sublayers, a multi-head self-attention mechanism, and a position-wise completely linked feed-forward network. A residual connection is used around each sub-layer.

Decoder:
The decoder is built in the same way as the encoder. The decoder, on the other hand, employs an extra multi-head attention layer that operates on the output of an encoder block. The multi-head attention that operates on output embeddings conceals all subsequent positions for all positions, allowing a prediction to be based solely on previously viewed positions.

Attention:
Attention is used by the transformer architecture in two ways: scaled dot-product attention and multi-head attention, with multi-head attention internally using scaled dot-product attention.

Scaled Dot-Product Attention:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Multi-Head Attention:
By learning independent attention heads, multi-head attention allows you to project queries, keys, and values into higher dimensions. An attention head is essentially the outcome of a single scaled dot-product attention. A number of those heads are subsequently concatenated and weighted using another weight matrix.

Feed-Forward Networks:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Positional Encoding:

Although the transformer design does not sequentially handle the input, the information provided by the sequence should still be used. A positional encoding is a vector of values that should offer information about the relative or absolute position of a segment of the input. To compute the encodings, the architecture employs the following functions.

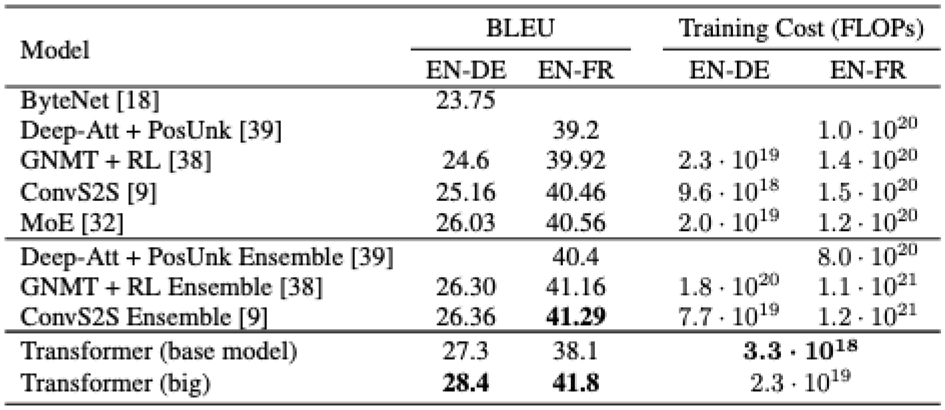
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Training:

The researchers trained using the standard WMT 2014 English-German dataset, which included around 4.5 million phrase pairings. Sentences were encoded using byte-pair encoding, which has a 37000 token common source-target vocabulary. The researchers employed the substantially bigger WMT 2014 English-French dataset, which consisted of 36M phrases and separated tokens into a 32000 word-piece vocabulary, for English-French.

Results:

The Table summarizes the findings and compares our translation quality and training costs to those of alternative model designs found in the literature. It has calculated the amount of floating-point operations required to train a model by multiplying the training period, the number of GPUs employed, and an estimate of each GPU's sustained single-precision floating-point capacity.



Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Conclusion:

According to the report, the team is optimistic about the future of attention-based models and plans to adapt them to other jobs. They want to apply the Transformer to challenges with input and output modalities other than text, as well as to examine local, limited attention techniques for effectively handling huge inputs and outputs such as graphics, audio, and video.

Code Link: https://github.com/gsu-kiranreddy/intro2DL-Repo2/tree/main/HW6