Attention & Transformers

Matt Smith

UW Data Science Club

3 April, 2019

Who am I?

4B CS and C&O student here at UWaterloo

VP of Education for UW DSC

I run the weekly DSC NLP reading group

ML engineer at Yelp where I do a lot of research into modern NLP techniques

I had a very non-standard path into ML

I usually hang out in the DSC clubroom if you want to chat about CS theory, machine learning or careers in ML

Plaza Top 3: Gol's, Nuri Village, Waterloo Star

In This Talk

Why should you attend this talk?

Continuing on from a basic understanding of RNNs, you'll see:

- An introduction to neural machine translation and other modern NLP problems
- A practically and biologically motivated introduction to attention/alignment mechanisms
- An overview of the transformer architecture, OpenAI's modification, ULMFiT and BERT
- You can listen to my Chinese and Japanese and laugh at me

Coverage

Machine Translation

Pretty obvious: Given some text in one language, convert it to another

Impossibility

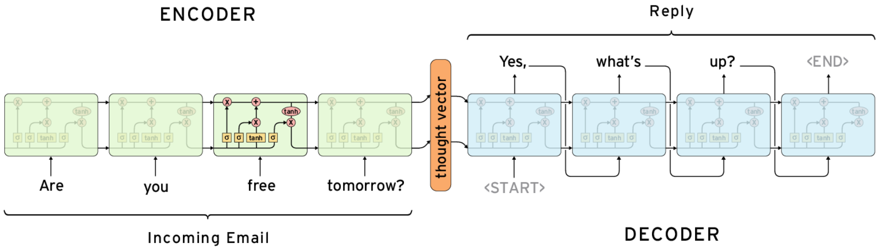
Sometimes we can't translate without assumptions. This is not fixable without changing the paradigm of translation.

- Politeness: "だ" vs "です"
- Idioms: "The cat's pyjamas" ⇒ "???"
- Different meanings/culture: "I love you" vs "我爱你"
- Slang: "飯テロ"
- Many-to-many: "Let's do it" ⇒ "それで行こう" vs "するか"

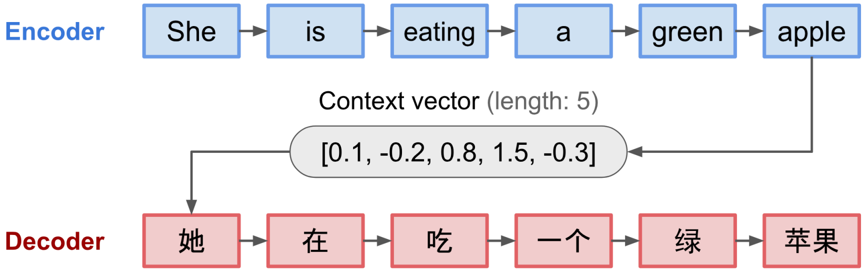
Usually evaluated by comparison with a database of human-generated candidate translations. Unsupervised learning is possible due to large amounts of parallel texts (movie subtitles, EU/UN proceedings, etc.)

Encoder-Decoder Architecture

At the time, SOTA was encoder-decoder networks:



Encoder-Decoders for Translation



Motivation for this Paper

Written soon after the inception of neural machine translation

The best neural models used two RNNs: an encoder and a decoder

The encoder reads a sentence and produces an encoding or “sentence embedding”

The decoder then reads the encoding and produces a translation

Encoder-Decoder Networks

This has a biological motivation:

- As you read words, you update your brain state
- Finally you have read the input sentence
- You write the output word-by-word, updating your brain state as you go
- Finally you output something signaling the end of the translated sentence

This is the same as what an encoder-decoder network does, except with a hidden state instead of a brain state

Problems with the Biological Motivation

This is not exactly how we translate sentences

While a doubly-fluent speaker could translate simple sentences in this way, you might struggle as the sentence length increases

A real translator might want to look back at previous words in the sentence to inform their word choice while writing the translation

Practical Problems

Researchers found that after a certain sentence length (fixed wrt the dimension of the encoding) there would be catastrophic forgetting of sentence context

This means we forget things such as the gender of the subject or the object, the verb tense, the voice or style, if we are inside a quote or a parenthetical, etc.

This causes exponentially worse performance as sentence length increases

This is what we would expect from understanding the issues with the biological motivation

In this Paper

This paper allows the decoder to determine which words in the input are relevant to each word of the output

This is akin to a human translator looking back at the input sentence to ensure consistency

For example, pluralities, tenses and genders should be constant in the translated sentence

Thus the biological issue is fixed in this new model

Also, the entire context of the sentence to be translated does not have to fit into a fixed encoding, fixing the practical issue

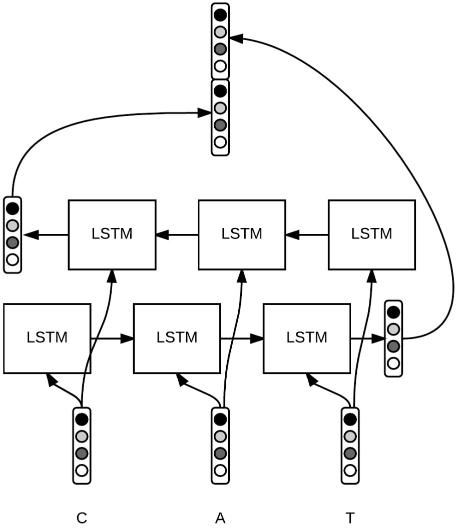
New Technique in NMP: BiRNN

Here, the authors employ a common technique in RNNs to increase performance. Usually RNNs are motivated because they seem to act like humans do when reading or listening to natural language:

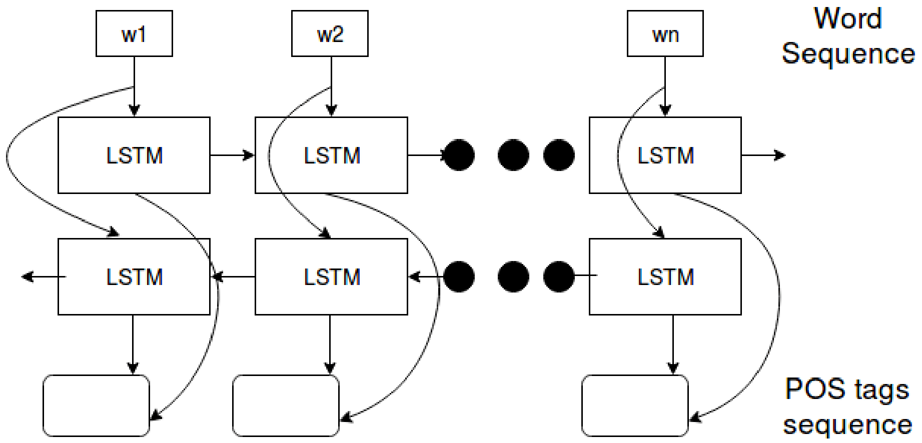
- Read word-by-word (or unit-by-unit) in “reading order”
- Keep a working memory that is used to contextualize future words

BiRNNs have the RNN read the sentence in both reading order and also in reversed order. The output vectors of the RNN are then combined (usually concatenation or addition)

BiLSTM with One Output



BiLSTM with Multiple Outputs



Why?

We throw away a bit of our biological motivation for RNNs to achieve better performance

Exercise

Why does this work?

Why?

We throw away a bit of our biological motivation for RNNs to achieve better performance

Exercise

Why does this work?

- We mentioned earlier that the vector in the encoder-decoder network was a bottle-neck for sentence length
- Similarly, the hidden state in a RNN is a bottleneck
- We often work hard to fix this bottleneck and allow our RNNs to parse longer-range dependencies (part of our motivation for LSTM/GRU)
- Single-output RNNs get context from both the beginning and the end of the sentence

New Technique in NMT: Decoder Attention

More biological motivation:

- When you read, you don't actually just read in reading order
- Your eyes move back and forth, rereading parts of the sentence sometimes if you need to get more context
- This is especially true if you're translating a sentence as languages may have different word order
- This means you don't need to keep the entire meaning of the sentence in your head, you can look back at the original input as needed!

Thus we want some sort of "attention" mechanism. The authors call this "soft-alignment" but future work frames it like a question-answering network (more on this later)

Attention

When we are trying to use a RNN make a prediction at timestep t, we take as inputs an input vector x_t and a context vector s_{t-1}

Typically in NMT, the input to the decoder at timestep t is just the output from the decoder at the previous timestep

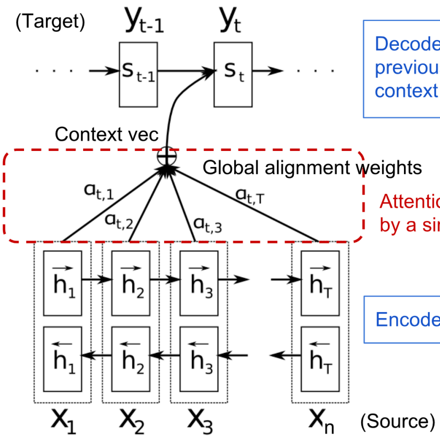
Instead, attention dynamically computes what the input to the decoder should be based on a weighted sum of the outputs of the encoder

For each encoder output h_i, a weight $\alpha_{t,i}$ is produced. Then,

$$x_t = \sum_{i=1}^{T} \alpha_{t,i} h_i'$$

where h_i is the output of the encoder at timestep i

A Graphical View of Attention



Decoder: RNN with input from previous state + dynamic context vector.

Attention layer: parameterized by a simple feed-forward network

Additive Attention

Encoder: bidirectional RNN

Image: Lilian Weng's *Attention? Attention!*

Computing attention weights

This paper computes attention weights using an attention model.

$$e_{t,i} = a([s_{t-1}; h_i])$$

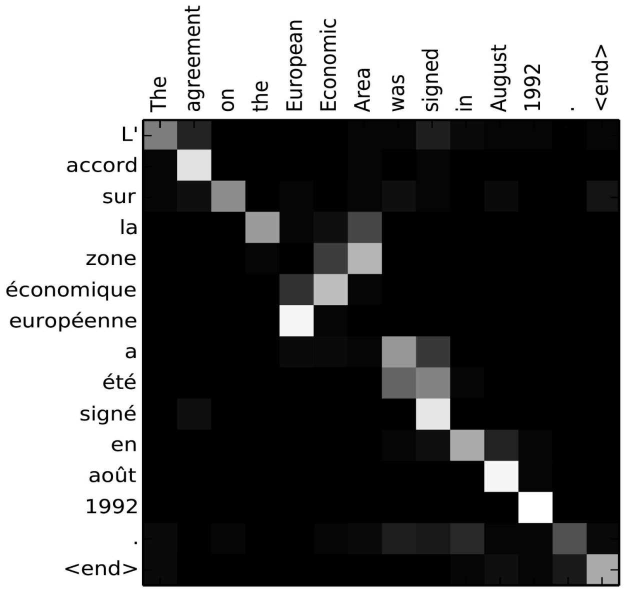
$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})}$$

In this case a is a feed-forward neural network,

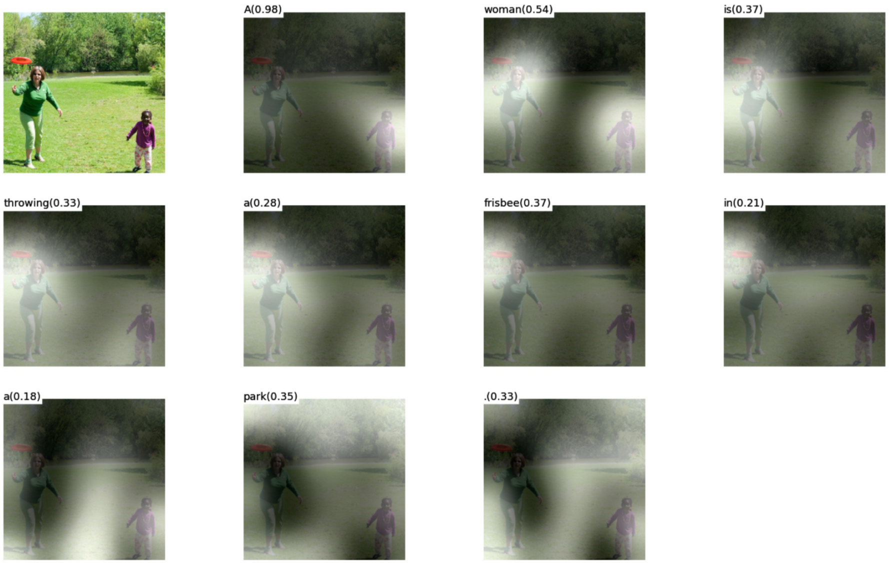
$$a([s_{t-1}; h_i]) = \sigma(\, W[s_{t-1}; h_i] + b)$$

Notes about attention: it's kind of slow. The naive implementation runs on the CPU

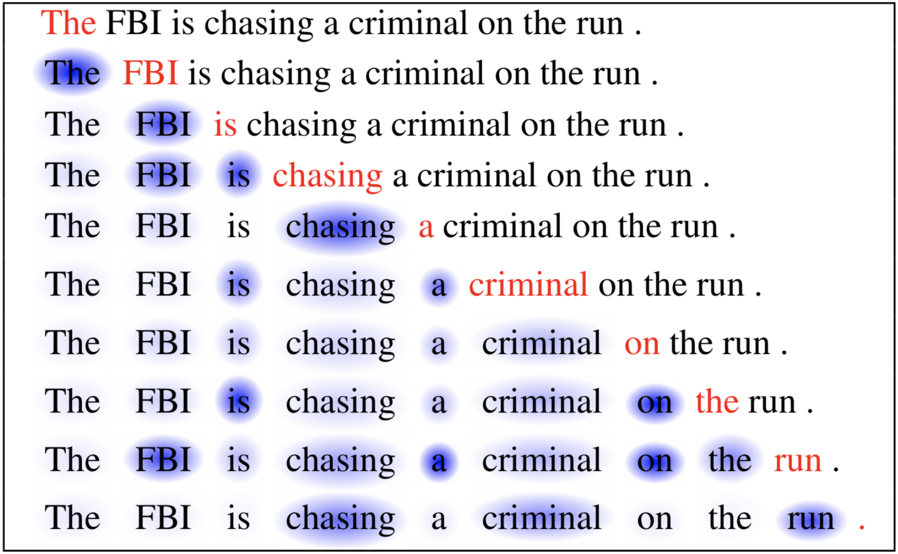
Visualizing Attention



Visualizing Attention



Visualizing Attention


The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .
The FBI is chasing a criminal on the run .

Different Kinds of Attention – Self-Attention

The previous two pictures are examples of self-attention

Instead of having the a decoder attend to an encoder, an encoder can attend to its previous layers

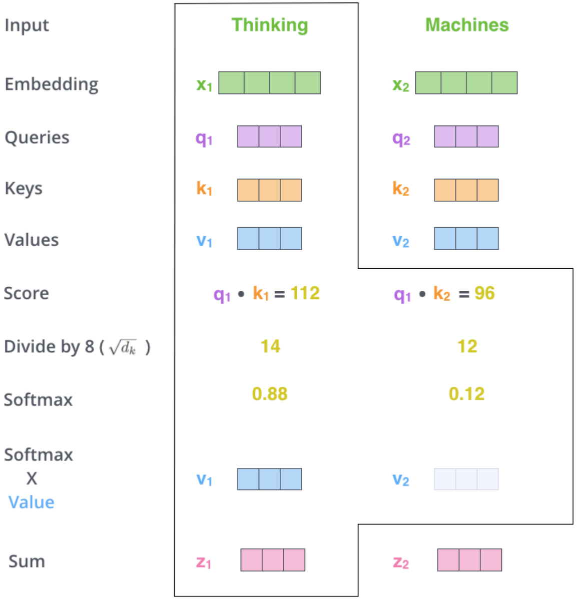
This provides a powerful and generic layer for discovering features that depend on correlations within data e.g. anaphora resolution

Anaphora Resolution

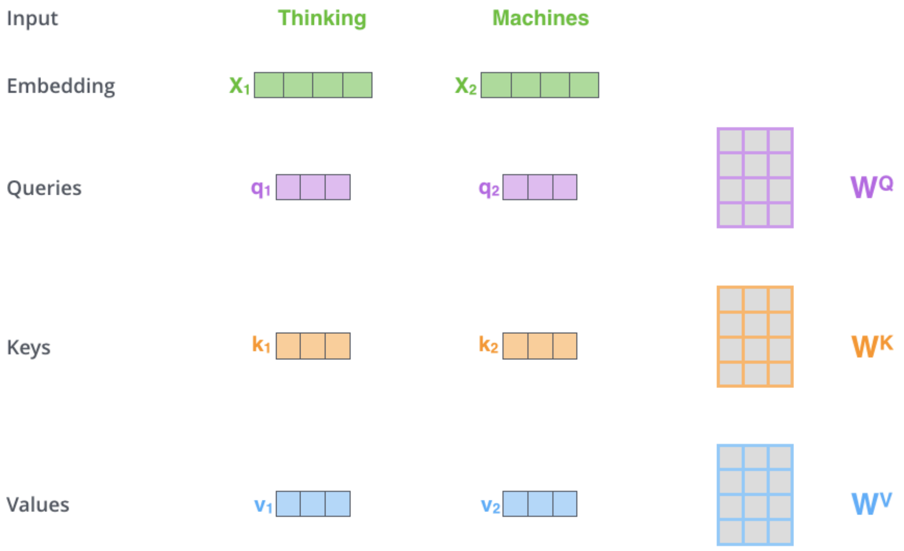
"Stacy went to the movies. She had fun."
Who does 'she' refer to? 'She' is an anaphora, and determining that 'she' = 'Stacy' is anaphora resolution

Self-attention uses a dot-product formulation that can more easily be done on GPU

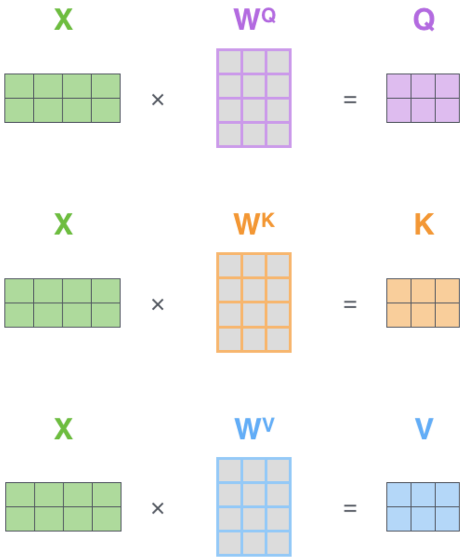
Dot-Product Attention



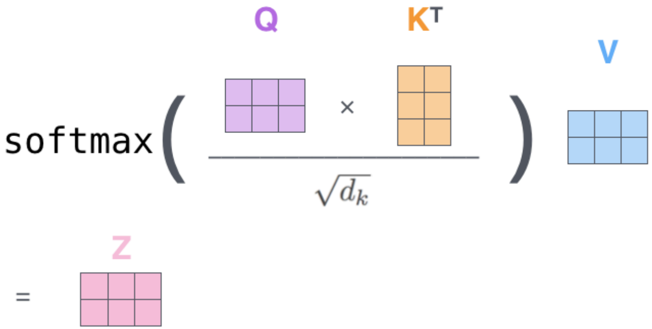
Dot-Product Attention



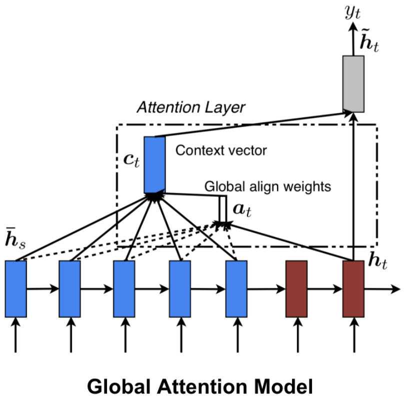
Dot-Product Attention

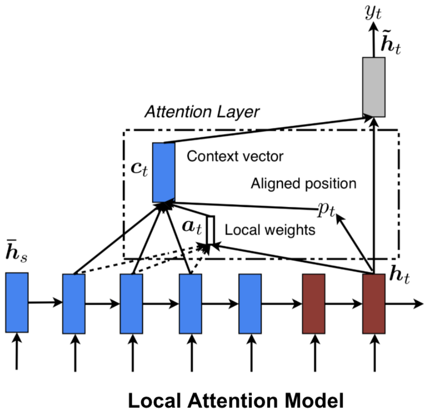


Dot-Product Attention



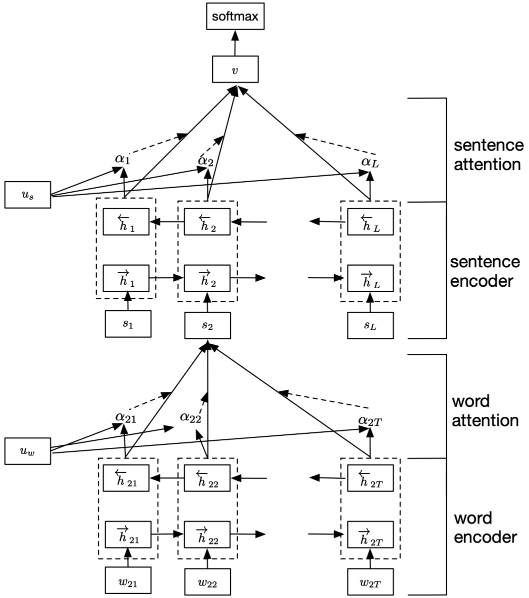
Different Kinds of Attention – Global vs Local





Coverage

Model – Practice Your Understanding of Attention



Coverage

Self-attention

RNNs take a sequence as input and output a sequence

Attention mechanisms take a sequence as input and output a sequence

#showerthoughts: what if we only used attention mechanisms?

Enter transformer networks. All of the work is done through self-attention and dense layers

We already know how scaled dot-product attention works so I'll describe it simply

Multi-head Attention Mechanism

Transformers use multiple “heads” of attention to attend to multiple concepts at once

The V, K and Q self-attention matrices are projected onto subspaces, attended to and then concatenated

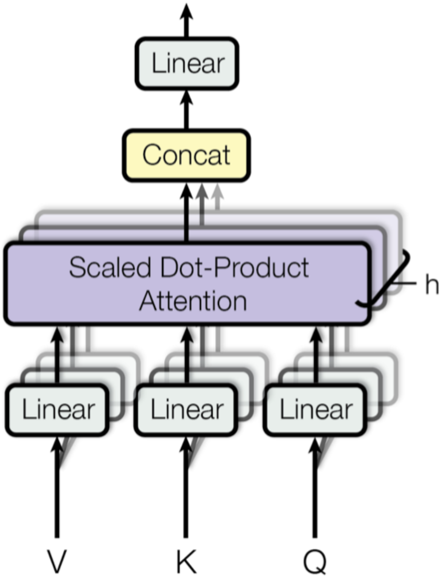
Subspace Projection

Recall that “project onto a subspace” is just a dense layer
The authors use a ReLU activation here

The intuition here is that different heads can learn different linguistic features. A parallel intuition is that this is somewhat like ensembling

The final result for each head is concatenated together and then passed through a dense layer

Multi-head Attention Mechanism



Transformers

The final model is quite complicated, but I'll throw up a picture on the next slide

This is still a encoder/decoder network. There are two notes to make here:

1) Because there's no RNNs, we can't access word order. In order to fix this, we introduce some periodic noise that the model can learn:

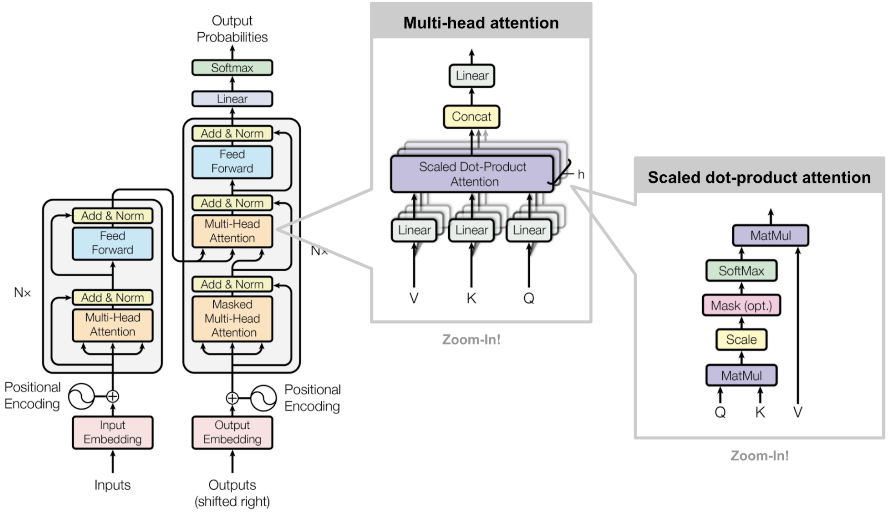
$$x_{(pos,2i)} \mathrel{+}= \sin(pos/10000^{2i/d_{model}})$$

$$x_{(pos,2i+1)} \mathrel{+}= \cos(pos/10000^{2i/d_{model}})$$

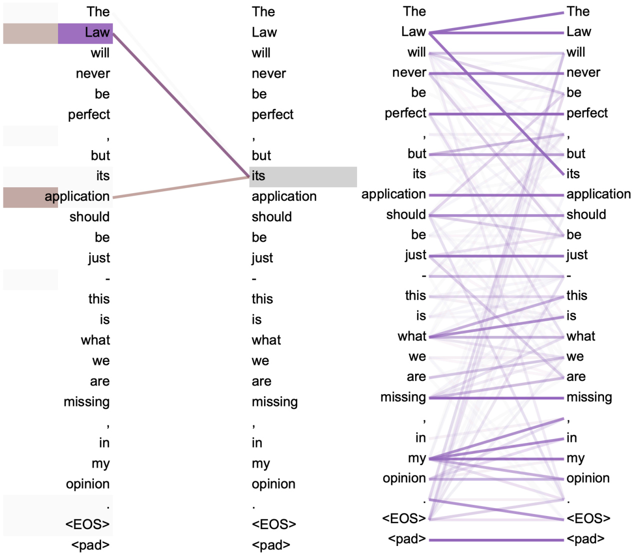
The authors also tried learned noise

2) We modify one of the attention mechanisms in the decoder to stop it from "looking into the future" (masking)

Picture of the Model



Visualizing Self-attention – Anaphora Resolution



Coverage

Why does no one use transformers?

Transformers didn't see widespread adoption after their release

Transformers have many parameters and are very deep. They require a lot of data to train

Transformers were also very slow. They're very parallelizable but this is only notable to some of the largest users

If you want to justify the slow performance of transformers, you need to have massive amounts of data and train them for long periods of time

The Idea

By training a transformer on a language modeling task, it provides a very good initialization for quickly retraining the transformer on a domain-specific task with a smaller labeled dataset

This allows us to train a transformer network with relatively small amounts of sample English text

This model in this paper is “GPT-1”

The trick is representing text in a special form such that the input shape says the same. They do this by combining input features using special delimiters

Recall: Language Modeling

Suppose I have a sentence $x_1 x_2 x_3 \ldots x_n$

Given a prefix $x_1 x_2 \ldots x_k$, I want to predict x_{k+1}

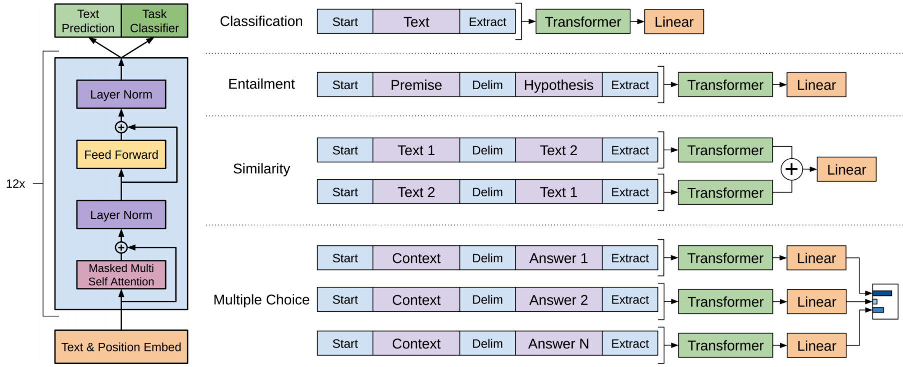
Specifically, I want to find

$$\Pr(x_{k+1} \mid x_1 x_2 \ldots x_k)$$

This is what's known as a language modeling task, and is what is used in this paper for the pretraining step and the auxiliary finetuning objective

It makes a good pretraining step as it is totally unsupervised – all you need is English (or whatever language) text

Model



Text Prediction
Task Classifier
Layer Norm
Feed Forward
Layer Norm
Masked Multi Self Attention
12x
Text & Position Embed
Classification
Start
Text
Extract
Transformer
Linear
Entailment
Start
Premise
Delim
Hypothesis
Extract
Transformer
Linear
Similarity
Start
Text 1
Delim
Text 2
Extract
Transformer
Start
Text 2
Delim
Text 1
Extract
Transformer
Linear
Multiple Choice
Start
Context
Delim
Answer 1
Extract
Transformer
Linear
Start
Context
Delim
Answer 2
Extract
Transformer
Linear
Start
Context
Delim
Answer N
Extract
Transformer
Linear

ULMFiT

This paper was inspired by a paper called *Universal Language Model Fine-tuning for Text Classification* (ULMFiT) by fast.ai

ULMFiT focused on fine-tuning recurrent language models to achieve good transfer-learning performance with little data

ULMFiT paper presents other optimizations such as per-layer learning rates, a triangular learning rate and gradual unfreezing that are not used in this paper

Instead, *Improving Language Understanding with Unsupervised Learning* uses an auxiliary objective, and adds the idea of task-aware input transformations

Coverage

Character-Aware Neural Language Models

Necessary to understand the ELMO paper

ELMO paper is required for understanding the BERT paper

BERT was SOTA until GPT-2 came out, and BERT is still very good considering it has way fewer parameters than GPT-2

Idea

Word embeddings don't work well for out-of-vocabulary words

However many languages, such as English, have strong morphological clues

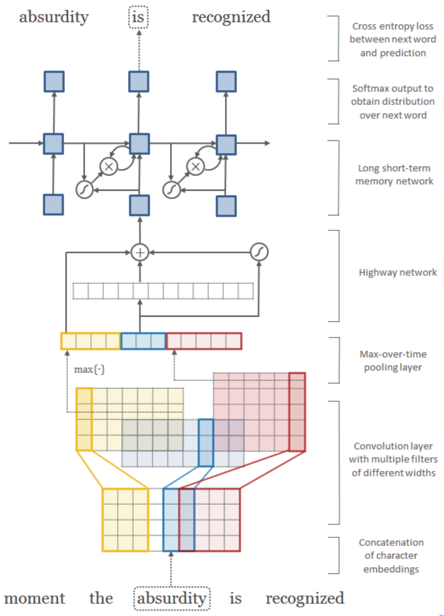
Morphological Clues

Imagine you've gone your whole life without seeing the word 'cats', but you know the word 'cat' and have seen many other examples of words and their plural form
You can still what the word 'cats' means the first time you can see it

Word embeddings throw out all of the morphological information for OOV words

The authors note that morphological features are very local, and suggest using CNNs to capture them

Model


absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

CNNs

A full treatment of CNNs would take an hour by itself

CNNs apply a sliding filter to the input to get an output of similar shape

The notable property of CNNs is that this filter naturally understands spacial relationships within the input space

In the case of NLP, the spacial relationship we care about is the arrangements of letters

CNNs are more biased towards spacial relationships than RNNs are

Highway Networks

A generalization of residual connections

Given input x and dense layer D, the output of a highway connection is

$$z := T(x)D(x) + V(x)x$$

where $T, V : \mathbb{R}^n \to [0, 1]$. Commonly we set $V(x) = 1 - T(x)$ and $T(x) = \sigma(Wx + b)$

Residual Connections

Residual connections are the special case where $T = V = 1$

Results

Embeddings that:

- Perform similarly to SOTA language models
- Have very small numbers of parameters
- Work well for OOV words
- Work well for neologisms
- Work well for common misspellings (e.g. ‘loook’)

Coverage

Deep contextualized word representations – AKA ELMO

The authors claim the problem with normal word vectors is that they are static

Each word maps to a fixed vector irrespective of context

ELMO produces contextualized word vectors, which means that the continuous representation of the word depends on the surrounding context. This accounts for polysemy

Polysemy

Polysemy is the property that a word can mean different things in different contexts
For example, ‘blue’ in ‘a blue ball’ vs ‘feeling blue’

ELMO at its core

Hey ELMo, what's the embedding of the word "stick"?
There are multiple possible embeddings! Use it in a sentence.
Oh, okay. Here: "Let's stick to improvisation in this skit"
Oh in that case, the embedding is: -0.02, -0.16, 0.12, -0.1 ....etc

Exercise

Exercise: what are the pros and cons of this?

Exercise

Exercise: what are the pros and cons of this?

Pros:

- word embeddings can take context into account
- word embeddings for words with lots of polysemy are not ‘overloaded’

Cons:

- you need to run a biLM on your input to get the word vectors
- running ahead of time + storing the result takes a lot of space

Deep Word Vectors

ELMO produces these vectors by a bidirectional language model, combining the lower layers of the model to produce the word representation

This is what is meant by a deep language model

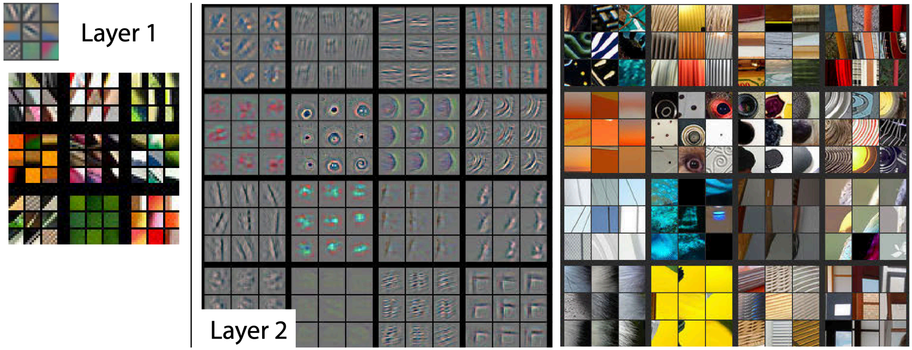
I see a similarity/inspiration to CNNs

The lower layers of CNNs compute lower-level features like edges or shapes, while the higher layers compute high-level features like objects or faces

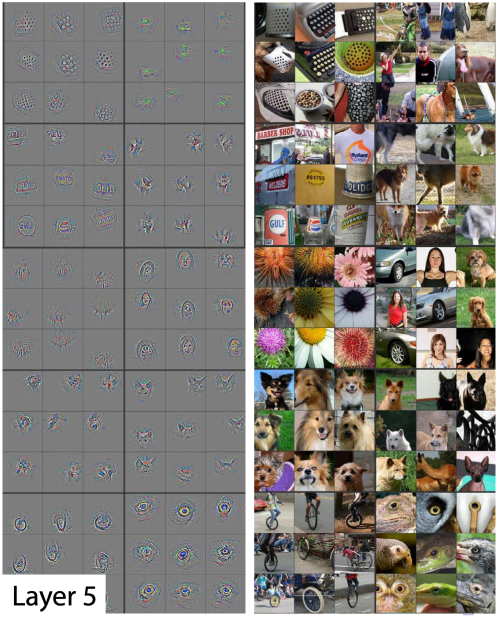
Likewise the lower layers of LMs compute low-level language features; upper layers of LMs compute high-level language features

You need both to make better word embeddings, according to this paper

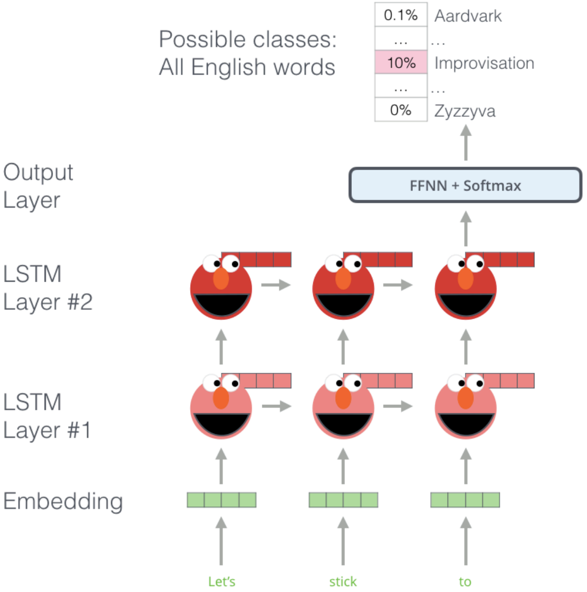
Visualizing CNN Layers



Visualizing CNN Layers



Model


Possible classes:
All English words
0.1% Aardvark
… …
10% Improvisation
… …
0% Zyzzyva
Output Layer
FFNN + Softmax
LSTM Layer #2
LSTM Layer #1
Embedding
Let's
stick
to

Getting embeddings from ELMO

The embedding of the tth word is the weighted sum of each layer's output at time t.

For example: Suppose there are L layers. If the output of layer i at word t is x_i^t, then the output embedding for the ith word is

$$\gamma \sum_{i=i}^{L} s_i x_i^t$$

γ and s_i are trainable in the model that uses ELMO weights, while the rest of the weights in the model are fixed.

Coverage

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Combines:

- Bidirectional LMs
- ULMFiT finetuning
- OpenAI-style transformers
- ELMO

Using OpenAI-style transformers and finetuning is simple enough

Bidirectionality poses a challenge...

Bidirectionality

There are two ways of transferring context from pre-trained networks: feature-based and fine-tuning

ELMO is the SOTA feature-based method, and is combines the left and right contexts, conditioned independently, to create its embeddings

GPT is conditioned on just the left-to-right context

Can we combine them?

Bidirectionality

There are two ways of transferring context from pre-trained networks: feature-based and fine-tuning

ELMO is the SOTA feature-based method, and is combines the left and right contexts, conditioned independently, to create its embeddings

GPT is conditioned on just the left-to-right context

Can we combine them?

Yes! BERT's output at every step is conditioned on both the left and right context (this is superior to just concatenating the left and right context after every layer)

Bidirectional LMs

The language model we used previously to pre-train our transformers was to predict the next word given a sentence fragment

In the bidirectional case, there is no ‘next’ word. So the authors created a new LM objective

For 15% of the tokens (word pieces in this case) in the input sentence:

- replace them with a blank token 80% of the time
- replace them with a random token 10% of the time
- do nothing 10% of the time

BERT must now predict which tokens were changed

Next Sentence Prediction Task

Because this LM task doesn't capture sentence-to-sentece relationships well, there is another pre-training task

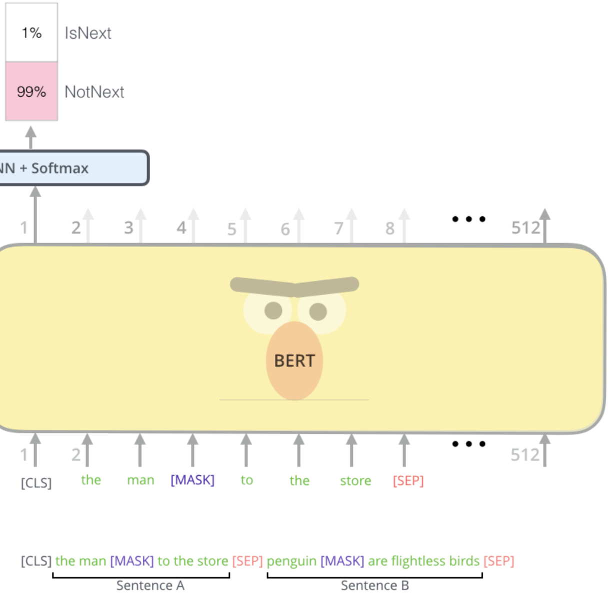
Given two masked sentences, predict whether the second one follows the first one in the training corpus or not

Next Sentence Prediction Task

Predict likelihood that sentence B belongs after sentence A

Tokenized Input

Input


1% IsNext
99% NotNext
FFNN + Softmax
1 2 3 4 5 6 7 8 ••• 512
BERT
1 2 ••• 512
[CLS] the man [MASK] to the store [SEP]
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]
Sentence A
Sentence B

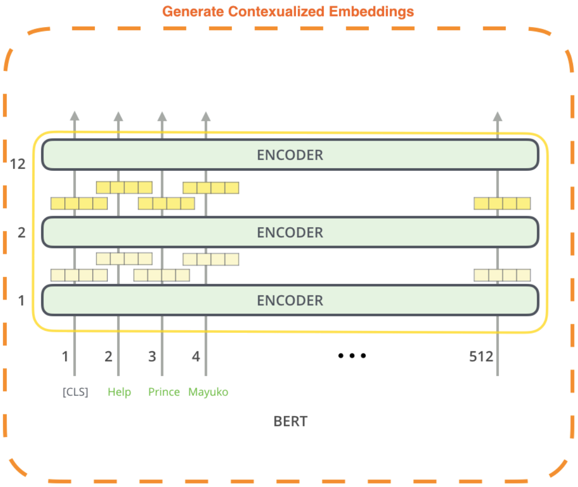
What do Onions and Ogres Have In Common?

So now we have a bidirectional LM that we can finetune in a general way using task-specific input transformations and it's great

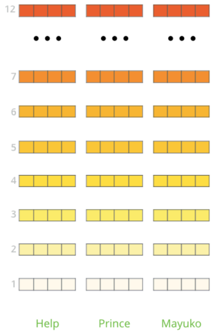
Remember in ELMO, when we used a weighted sum of the internal layers of a biLM to create better embeddings?

BERT is a biLM with layers too! So we can take an internal sum of *it's* internal layers to create (hopefully better) embeddings

BERT Embeddings

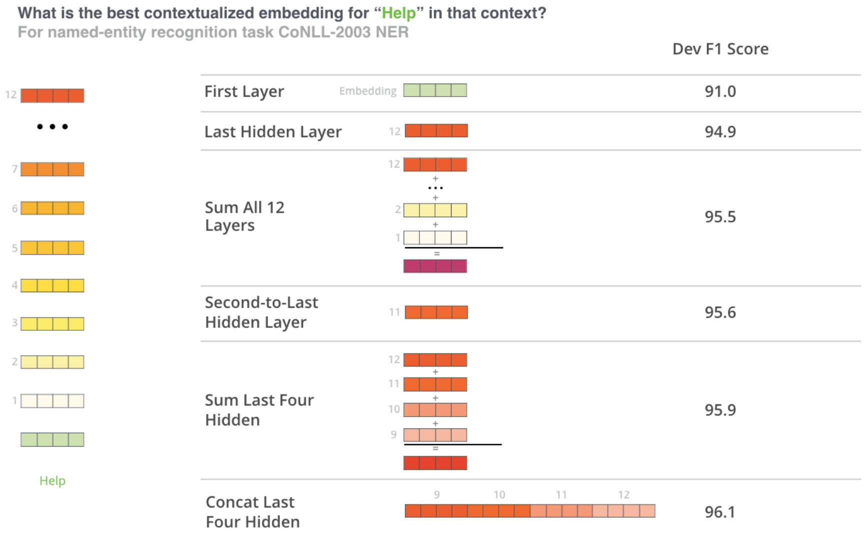


The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

BERT Embeddings Performance



What is the best contextualized embedding for "Help" in that context?

For named-entity recognition task CoNLL-2003 NER

	Dev F1 Score
First Layer	91.0
Last Hidden Layer	94.9
Sum All 12 Layers	95.5
Second-to-Last Hidden Layer	95.6
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1

12
• • •
7
6
5
4
3
2
1
Help
Embedding
12
12
+
• • •
+
2
+
1
=
11
12
+
11
+
10
+
9
=
9
10
11
12

Coverage

The Question

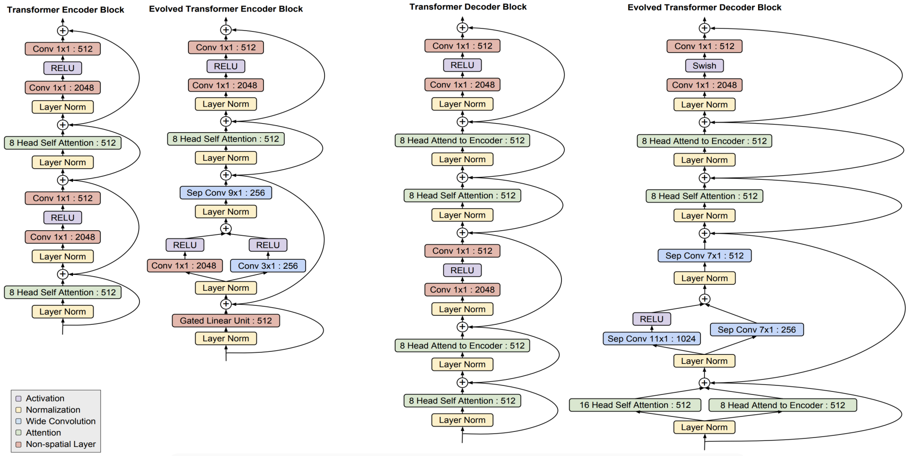
Can we use NAS (Neural Architecture Search) to find the optimal transformer network for English-to-German translation?

The Question

Can we use NAS (Neural Architecture Search) to find the optimal transformer network for English-to-German translation?

Yes, using 12 hours on 270 TPUv2s

Model


Transformer Encoder Block
Evolved Transformer Encoder Block
Transformer Decoder Block
Evolved Transformer Decoder Block
Conv 1x1 : 512
RELU
Conv 1x1 : 2048
Layer Norm
8 Head Self Attention : 512
Sep Conv 9x1 : 256
Conv 3x1 : 256
Gated Linear Unit : 512
8 Head Attend to Encoder : 512
Swish
Sep Conv 7x1 : 512
Sep Conv 11x1 : 1024
Sep Conv 7x1 : 256
16 Head Self Attention : 512
Activation
Normalization
Wide Convolution
Attention
Non-spatial Layer

Coverage

Universality

It turns out that transformers aren't universal computation devices

They struggle to generally learn some simple tasks such as copying the input to the output

By modifying transformers to give them a recurrent step, universal transformers are universal computation devices that also outperform vanilla transformers on some tasks

Recurrence in Transformers

The original transformers has a fixed number of stacked self-attention layers

The authors of *Universal Transformers* say that this stops the transformer from having the bias towards inductive or recursive behaviour that RNNs have

Universal transformers are allowed to perform stacked self-attention any number of times they want per each input word

The model predicts the probability that computation on a word should halt

Coverage

Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context

An important note about transformers is because they don't have recurrent connections, they only consider a fixed-size context

While these contexts are sufficient to get amazing results on many sentence understanding tasks, they are insufficient for others

Transformer-XL adds a recurrent mechanism to transformer networks

Understanding Long Sequences With Transformers

Suppose the transformer takes a context of length k

How to predict word x_{n+1} given words $x_1 x_2 \ldots x_n$?

Pass $x_{n-k} x_{n-k+1} \ldots x_n$ into the transformer

Thus to predict a sentence of length n requires $n - k + 1$ calls to the transformer

This also means that our training on the first/last word of a sentence is limited

Understanding Long Sequences With Transformer-XLs

Suppose the transformer takes a context of length k

How to predict word x_{n+1} given words $x_1 x_2 \dots x_n$?

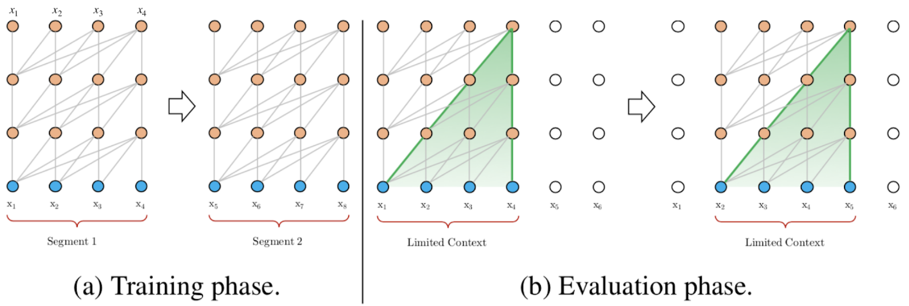
Break the sentence into blocks of length k

The prediction for the words after the start of the ith block is based on the $i - 1$th block and ith block

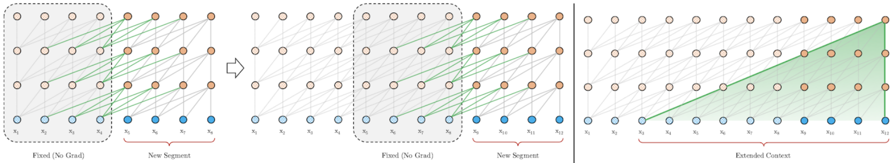
Thus to predict a sentence of length n requires n/k calls to the transformer

So we also get a nice performance boost from this :)

Vanilla Transformer Context



Transformer-XL Context



Fixed (No Grad)
New Segment
Fixed (No Grad)
New Segment
Extended Context
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}

(a) Training phase. (b) Evaluation phase.

New Positional Embeddings

Vanilla transformers used fixed per-segment noise to encode absolute word order

However since Transformer-XLs have recurrent connections, they may see these encodings repeated multiple times, causing confusion

Instead of encoding absolute positional information, we only store relative positional information

Longer Recurrent Connections in Training

The authors only passed the intermediate values from the previous segment to the next segment during training

However they found that during evaluation they could pass intermediate values from several previous blocks

This was limited only by available GPU memory as there were no bad generalization effects observed

The End

That's pretty much all I know about attention and transformers

Thanks for coming out :)

I Want to Know More

An excellent series by Jay Alammar:

- `https://jalammar.github.io/illustrated-word2vec/`
- `https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/`
- `https://jalammar.github.io/illustrated-transformer/`
- `https://jalammar.github.io/illustrated-bert/`

I'll post these slides on my personal site, `student.cs.uwaterloo.ca/~mjksmith`