Today: 6 Introduction to Transformers
6 Project Structure, Scope, Deliverables

Where are we? Working on A2, making a neural-net classifier that takes in sentences and produces a classification.

⇒ sentences encoded with pre-trained word embedding
⇒ side note: A2 also asks you to try letting the word embeddings themselves be trained/tuned.

Introduction to Transformers

- the reason I'm teaching this course is because, in my research on making therapeutic chatbots, we found a pre-trained transformer produced remarkable results - over 3 years ago. + Subsequent improvements & capabilities are both stunning & fundamental.

- transformers are the state-of-the-art method for:

1. **Classification of language** - not MLP or CNN as in A2 😊

2. **Generation of language** - which we haven't yet discussed outside of Lecture 6; I believe generation is qualitatively different than classification, even though the structures overlap; Even reading & writing are related.
Let's begin by describing a core topic in NLP: Language Models — GPT-1/2/3/4 are called "Large Language Models"; Juntak, Chap 3/13.0 3.1

"Language models are models that assign probabilities to words" ??
- probability that the next word in a sequence of words is word \( x \): e.g. \( P("\text{Proy}\) > 2
- meaning what? that if that word is added next, that the sentence, to that point:
  1. Is grammatically correct
  2. Makes sense

E.g.: I believe running water is important for ... ?
- good health - everyone - success

The task of a language model is to
- once we can do this \( G \)

Given: One or more words in a sequence
Compute: The probability that every word in the vocabulary is the next word.

i.e. assume the size (\#words) in vocabulary \( |V| = M \)

Given a sequence of \( N \) words \( X_0, X_1, \ldots X_{N-1} \)

Compute: \( P(W_0 \text{ is } x_n) \)
\( P(W_{m-1} \text{ is } x_n) \)
- aside: given that we can do this, we can use these probabilities to compute the "likelihood" of an entire sequence of words. "Likelihood that is grammatical/meaningful"

- I.e., we want a predictor that looks like this:

\[ X_0, X_1, \ldots, X_{n-1} \rightarrow \text{Predictor} \rightarrow P(w_0) \rightarrow P(w_1) \rightarrow P(w_{n-1}) \]

- does this look familiar? (A1553) (except > 1 word input)

- we want to train a neural network to do this:
  - the \( X_i \) are word embeddings, of course
  - either pretrained (by GloVe) OR, trained as part of the process

- where can we find the training data? (except now we might be unlocking something very powerful)
  - everything ever written, again!
  - amount of data is unprecedented

- The Transformer architecture that we will first use as a classifier is actually trained to do exactly this (predict prob of next word given the sequence). We can first think of it this way:
- We will (but not yet) dive into Transformer Blocks

Three important comments/insights:

1. Perhaps one reason this is called a Transformer is that, for each Transformer block, the inputs = outputs and so the information is "transformed" ("changed") but not increased or decreased. d \times n \text{ in } d \times n \text{ out} → d = embedding dim or hp

- This allows any number of Transformer blocks to be stacked

→ That is one way the big Transformers are made

- Evidence has been that more = better, if sufficient data

2. That size, \(d \times n\), or just \(N\) assembly field, is often called the context size; very important.

- Context is very important in all communication.

- For original Vaswani Transformer \(N = 512\)
  
  \[ \text{GP7-2} \quad N = 1024 \]
  
  \[ \text{GP7-3} \quad N = 2048 \]

- Is limited by the \(N^2\) complexity of attention (coming soon)

3. Observe the "language head" which, to repeat, looks just like output in assignment 7, Section 3.

- This is used when training the network to be a language model

- Consider this sentence being used to create training examples:

The smooth blue lake became choppy in the wind

Key: When we want to classify, we chop off this head and put on an MLP classifier head and train with fewer stages.

- Just like transfer learners, it can the transformer keeps its features
Training Example 1: light The smooth blue [mask] [mask] [mask]
Label: lake
-
- model computes $P(\text{lake})$ (i.e. lets off other probs)
- (normalized with softmax across all probs) $\rightarrow$ $\rightarrow$
- use "log loss" aka cross-entropy

$$\text{loss} = -\log (P(\text{lake}))$$

- for a batch of size $b$, compute the average loss $\frac{1}{b} \sum_{i=1}^{b} \text{loss}(\text{training example})$

- Side note: "teacher forcing" means always use the (right) answer to compute loss. (which is sort of obnoxious rather than using the predicted output)

- there are trillions of examples of writing, unlike most other ML problems
- that writing contains much of human knowledge!

---

- Note 1: the sequence of words input $x_0, x_1, x_2, \ldots$ does not contain any information about which word comes where, in order.
  - yet that ordering very much matters

- so in addition to the word embeddings, a positional embedding is added to the word embedding [runs don't have this problem]
- Positional embedding can be a number absolute or relative that is added to the embedding.

- Note 2: Both the word embeddings themselves and the positional embedding are learned (treated as parameters), changed through gradient descent back propagation. I.e. don't start with word embeddings.

Next lecture: the details of the Transformer Block beginning with attention.

If time, two rants:

1. This structure (just one stack of T-blocks), I believe, is the essence of Transformers. The literature is full of Encoder -> Decoder structures or encoder-only or decoder-only. This Encoder -> Decoder structure is based on how RNNs were used to process language and is not necessary for Transformers. However, it is the source of lots of confusion. See next page.

2. We will delve into what is happening thought to happen inside the Neural Network of a Transformer:
   - it is speculation.
   - the proof of success is always empirical.

That said, it is remarkable what learning to predict the next word can achieve!
4.2.3 Transformer Decoder (T-D)

We introduce a simple but effective modification to T-ED for long sequences that drops the encoder module (almost reducing model parameters by half for a given hyper-parameter set), combines the input and output sequences into a single "sentence" and is trained as a standard language model.

That is, we convert a sequence-transduction example \((m^1, ..., m^n) \mapsto (y^1, ..., y^n)\) into the sentence \((w^1, ..., w^{n+\eta+1}) = (m^1, ..., m^n, \delta, y^1, ..., y^n)\), where \(\delta\) is a special separator token and train a model to predict the next word given the previous ones:

\[
p(w^1, ..., w^{n+\eta}) = \prod_{j=1}^{n+\eta} p(w^j | w^1, ..., w^{j-1})
\]

Since the model is forced to predict the next token in the input, \(m\), as well as \(y\), error signals are propagated from both input and output time-steps during training. We also suspect that for monolingual text-to-text tasks redundant information is re-learned about language in the encoder and decoder. We believe this allows for easier optimization and empirically observe this with longer sequences (see Section 5.3). Note that because of the self-attention of the Transformer, when generating the next token, attention from both \(m\) and \(y\) are considered. At inference we provide the input sequence, \(m_i\), initially, and auto-regressively generate the output, \(y_i\), as normal.

From "Generating Wikipedia by Summarizing Long Sequences"
Liu, Saleh, Poli, Goodrich, Kaiser & Shazeer, ICLR 2018 + Arxiv.