Notes on Attention and the Transformer Model

Sham Kakade

1 Introduction

- Two "core" tasks: machine translation and language modeling.
- Many other tasks: part-of-speech tagging, named entity recognition, coreference resolution, semantic role labeling, question/answering, textual entailment, sentiment analysis, semantic parsing, etc.
- Goal today: build a language model. Why? "Representations" of the language may be helpful for many tasks.
- Interesting questions: memory, questions/answering, reasoning/logic.

2 A Short Summary of Some Improvements

- linguistics (grammars, parse trees) → statistical machine learning → "deep" models
- Brown clustering, n-gram models, IBM translation models
- Lots of work on Neural Embeddings.
- MT:
  - rule-based machine translation.
  - Statistical MT. IBM translation models.
  - Then a series of "deep learning" based approaches:
    - One first end-to-end models, with an "encoder-decoder" architecture. " Recurrent Continuous Translation Models." (Kalchbrenner & Blunsom, 2013)
    - Seq2Seq: using sequential neural models was a good first step. "Sequence to Sequence Learning with Neural Networks." (Sutskever et. al. '14)

- $*$ A series of papers started incorporating “attention”, where one directly tries to utilize long range dependencies in the representation. The idea is that these long range dependencies help when translating given words (the broader context is important). Now, all state of the art methods use some form of ”attention”. The Transformer is one of the most popular ones:
  “Attention is All you Need” (Vaswani. et. al. ’17)

- Transfer learning: how can make learning easier by transferring knowledge of one task to another? Recent exciting results showing that representations extracted from a good language model can help with this.
  - NAACL best paper:
    “Deep Contextualized Word Representations” (Peters et. al. ’18)
  - Another improvement with pretraining:
    “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” (Devlin et. al. ’18)

3 Datasets, Tasks, and some (important) details

3.1 Datasets and Objectives for Language Modelling

- machine translation: translate one sentence to another sentence. BLEU score used.

- language modeling: the goal is to learn a model over documents/ sequences, where given a document $d = w_{1:T}$ (or a sequence of words/characters) our model provides a probability $\widehat{\Pr}(d) = \widehat{Pr}(w_{1:T})$. Note that we often specify this joint distribution by the conditional distribution $\widehat{\Pr}(w_{t+1}|w_{\leq t})$ where the w’s are the words.

- The performance measure: If D is the true distribution, we measure the quality of our model by the *cross entropy rate*:

$$\begin{aligned}\text{CrossEnt}(\widehat{\Pr}||D) &:= \frac{1}{T}\mathbf{E}_{w_{1:T}\sim D}\left[-\log \widehat{Pr}(w_{1:T})\right]\\ &= \frac{1}{T}\mathbf{E}_{w_{1:T}\sim D}\left[-\sum_t \log \widehat{Pr}(w_{t+1}|w_{1:t})\right]\end{aligned}$$

- The *perplexity* is defined as $\exp(\text{CrossEnt}(\widehat{\Pr}||D))$. Intuitively, think of this as the number of plausible candidate alternative words that our model is suggesting.

- Examples:
  - Using a uniform distribution over m words gives a ppl of m.
  - Using the (estimated) unigram distribution has ppl about 1000.
  - Shannon, in his paper ””Prediction and Entropy of Printed English” (’51), estimated 0.6 to 1.3 bits/character (using human prediction of letters). This translates to 4.5 bits/word, using 1 bit/character and 4.5 characters/word. This gives a ppl of $2^{4.5} = 23$.
  - On the PTB dataset, the best ppl is about $55 - 60$ (on the validation set). The best character level entropy rate is 1.2 bits/character. This translates to about 77 ppl in perplexity units per word (to see this use $2^(1.175 * 390000/74000)$ since there are 390000 characters in validation set and 74000 words in the validation set).
  - There are other ’codings’ like BPE (byte pair encodings) and subwords. One can translate perplexities between different codings, provided they can faithfully represent the document/sequence.
- Concerns: memory and long term dependencies may not be reflected in this metric?
- Other ideas: RL, logic, meaning?
- Datasets used for language modeling:
  - Penn Tree Bank (PTB): first collection. 1M words. 10K vocab sized (based on standardization)
  - WikiText-2 (2M words) and WikiText-103 (103M words). Scraped from articles on Wikipedia passing a certain quality/length threshold, on all topics. 300k vocab size, > 3 times each.
  - Google Billion Words: web crawl, assorted topics. 1B words. 800K vocab size.
  - Books corpus: 11k public-domain novels. 1B words.
- Training:
  - GPUs/TPUs are needed.
  - Books/Billion words takes GPU weeks to a month to train (all standard models). TPU a few days.

3.2 The details matter: training and overfitting

The details do matter a lot for training language models. In contrast, in visual object recognition, once we move to the "Res-Net" style of architectures, training is relatively easy, where overfitting, hyperparameter tuning, and regularization are not major concerns. In fact, simply "early stopping" on vanilla SGD training is often non-trivially competitive on any reasonable model.

- overfitting is very real in language models, contrast to vision.
  - PTB with a trigram model (i.e. predict next word with previous two words) has 20 train ppl (in about 4 epochs) with 150 ppl on val.
  - PTB with LSTM+dropout has about 30 train ppl (in about 500 epochs) with 60 ppl on val.
- dropout: this is needed. dropout is used everywhere in these networks. $L2$ regularization alone is not comparable (very brittle and even with highly tuned it is not as good).
- (average) SGD or ADAM? Sometimes one algorithm is much better than the other.
- exploding gradients: these occur in practice.
- vanishing gradients: lots of discussion on this. unclear what is going on.
- dynamic evaluation: keep training at test time to handle topic drifts. squeezes about 5-10ppl on val (for PTB).

4 The Transformer

Let x be the input sequences of size $T \times m$, where T is the sequence length, often of length 512 and m is the vocabulary size, often in the range of 10^4 to 10^5.

Now we will describe a one hidden layer transformer, where it will predict the next word.

Parameters:

$$\begin{aligned} E &\in \mathcal{R}^{m\times d_{\text{embedding}}} \\ W_V, W_Q, W_K &\in \mathcal{R}^{d_{\text{embedding}}\times d_{\text{hidden}}} \\ W_1 &\in \mathcal{R}^{d_{\text{hidden}}\times d_1} \\ W_2 &\in \mathcal{R}^{d_1\times d_{\text{embedding}}} \end{aligned}$$

1. Embed the sequence:

$$x \leftarrow xE + P$$

so now x is of size $d_{\text{embedding}} \times T$. Here, P is the positional encoding. One common choice is:

$$\begin{aligned} P_{t,2i} &= \sin(t/10000^{2i/d_{\text{embedding}}}) \\ P_{t,2i+1} &= \cos(t/10000^{2i/d_{\text{embedding}}}) \end{aligned}$$

where i indexes the embedding dimension and t the sequence time. Note that often this is a fixed choice (and not a learned parameter).

2. Compute the 'values', 'query', and 'key':

$$V = xW_V,\ Q = xW_Q,\ K = xW_K\ ,$$

which are of size $T \times d_{\text{hidden}}$.

3. Compute the attention "weights" scheme:

$$h = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{\text{hidden}}}}\right) V$$

so h is of size $T \times d_{\text{hidden}}$. Importantly, note that QK^T is a $T \times T$ matrix. Here, abusing notation, the vector valued softmax$(\cdot)$ function is applied to every row of the matrix $QK^\top$. Recall that the vector valued softmax$(\cdot)$ function is defined so that the i-th component is:

$$[\text{softmax}(v)]_i := \exp(v_i)/\sum_j \exp(v_j)\ .$$

Note: each row of softmax(QK^T) sums to 1. The idea is that we want a convex combination of the columns of V.

4. The output after two transformations is then:

$$O = \text{ReLu}\left(\text{ReLu}(hW_1)W_2\right)$$

which is of size $T \times d_{\text{embedding}}$.

5. The prediction that the next word in the sequence, X_{T+1}, is the j-th word is then:

$$\begin{aligned} \widetilde{O} &= OE^\top \\ \widehat{\Pr}(X_{T+1} = j) &= [\text{softmax}(\widetilde{O}_T)]_j\ . \end{aligned}$$

i.e. only the last node $\widetilde{O}_T$ is used for prediction. Here, we have coupled the embedding weights and the prediction weights, where both use the matrix E.

4.1 Invariances and other observations

Define:

$$M = QK^T$$

which is of size $T \times T$.

- Suppose $P = 0$ (no positional encoding). The matrix M is shift invariant in that if translate the sequence by τ then (i, j) entry gets shifted to $(i+\tau, j+\tau)$ (provided these are in bounds). Similarly, $h_i \to h_{i+\tau}$.
- **Lemma.** Let $Q, K, V \in \mathbb{R}^{T \times d_{\text{embed}}}$, and let Π be a T-by-T permutation matrix. Then, $\sigma(\Pi Q(\Pi K)^T)\Pi V = \Pi\sigma(QK^T)V$, where $\sigma(\cdot)$ is the row-wise softmax of a matrix. In particular, suppose $P = 0$, then if we permute the sequence, i.e. $x \mapsto \Pi x$, then $h \mapsto \Pi h$.

  Proof. The LHS is $\sigma(\Pi QK^T\Pi^T)\Pi V$. It suffices to show that $\sigma(\Pi QK^T\Pi^T) = \Pi\sigma(QK^T)\Pi^T$. Indeed, letting π denote the permutation specified by Π, we have

  $$\sigma(\Pi QK^T\Pi^T)_{i,j} = \frac{e^{(QK^T)_{\pi(i),\pi(j)}}}{\sum_{j'=1}^T e^{(QK^T)_{\pi(i),\pi(j')}}}$$
  $$= \frac{e^{(QK^T)_{\pi(i),\pi(j)}}}{\sum_{j'=1}^T e^{(QK^T)_{\pi(i),j'}}} = (\Pi\sigma(QK^T)\Pi^T)_{i,j}.$$

5 Further comments:

Transformer Observations:

- hidden state interpretation and 'sequential' training/scoring: the above description is model for $\widehat{\Pr}(w_{T+1}|w_{1:T})$. We may be interested in the model predicting $\widehat{\Pr}(w_{t+1}|w_{1:t})$ (for $t < T$ where often $T = 512$), i.e. we may want to make multiple predictions simultaneously (say for training). For this, there is a way to use 'masking' with an upper triangular matrix so that (for all $t \le T$):

  $$\Pr(X_{t+1} = j|w_{1:t}) = [\text{softmax}(\widetilde{O}_t)]_j$$

- computation: the transformer computations are very efficient due to the manner in which the matrix multiplications can be parallelized. In contrast, the LSTM fundamentally needs a for-loop over the history. (The LSTM is a circuit with greater depth.)

- Transformer vs. LSTMs: hidden states vs no hidden states.
- Other models: TrellisRNN,

From the web:

- generations: a qualitative discussion.
- Elmo and Bert improvements
- page on state of the art
- MT: out of curiosity

Training:

- “big” models are best (unsurprisingly)
- dropout is critical (perhaps surprisingly)
- BPE now the norm. Do they handle the tails in a qualitatively (or quantitatively) different manner?
- Adam vs SGD (unclear); learning rate decay (not done for SGD)
- training/dev(or val)/test: one oddity is that, often, the dev and the test set come from a different distribution than the training set.

Some other thoughts:

- dynamic context evaluation: on the val/test set, update the model after each sample.
  - Let θ^g be the global model, and let θ^l be the local model.

    $$\theta^l \leftarrow \theta^l - \nabla \ell(\theta^l, (x, y)) + \lambda(\theta^g - \theta^l)$$

  - Lemep-Ziv vs. language modeling as a code?
- model averaging is important.
  KN-smoothing.

5.1 Logistic regression, model averaging: even the well-specified case is not that easy.

For logistic regression, there is a big difference between estimation with model averaging and estimation under a prior.

- Suppose you are given n samples of (x_i, y_i), with y_i's binary. And suppose we restrict to logistic classifiers:

  $$\ell(w, (x, y)) = -\log \Pr(y|x, w) = \exp(w^\top x)/Z_x$$

  and let:

  $$\text{LogLoss}(P(\cdot)) = \mathbf{E}_{(x,y)\sim D} - \log P(y|x)$$

  where D is the truth, and P is predictive distribution that we are scoring.

- (Hazan, Koren, Levy, '14). Suppose that we know $\|w^*\| \leq W_2$. Even if the true model is well specified, the best we can hope for, for any estimator $\widehat{w}_m$ based on m random samples, has regret (with high probability):

  $$\text{LogLoss}(\widehat{w}_m) - \text{LogLoss}(w^*) \leq O(\frac{1}{\sqrt{m}})\,.$$

- (Kakade, Ng '04; Foster et. al. '18) In the agnostic model (without any data generating assumptions), model averaging is much more powerful. Suppose that our "posterior" is:

  $$\Pr(w|\text{data}) = Pr(w|(x_1, y_1)\,, (x_m, y_m)) = \pi_{i=1}^m \Pr(y_i|x_i, w)\Pr(w)$$

  where $\Pr(w)$ is say a uniform prior (or Gaussian prior) which is over the ball of radius $\|w^*\|$.

- Now suppose our prediction on any (x, y) is:

  $$P_{\text{Bayes}}(y|x) = \int \Pr(y|x, w)\Pr(w|\text{data})dw$$

  then:

  $$\text{LogLoss}(P_{\text{Bayes}}) - \text{LogLoss}(w^*) \leq O(\frac{1}{m})\,.$$

6 Acknowledgements

These notes were based on discussions with Xinyi Chen, Karthik Narashiman, Cyril Zhang, and Yi Zhang.