EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Homework 6

This homework is due on Sunday, Oct 23, 2022, at 10:59PM.

Deliverables: Please submit the code/notebooks to the code assignment. Alongside the code, attach a pdf printout of the notebook to the written assignment.

1. Zachary's Karate Club

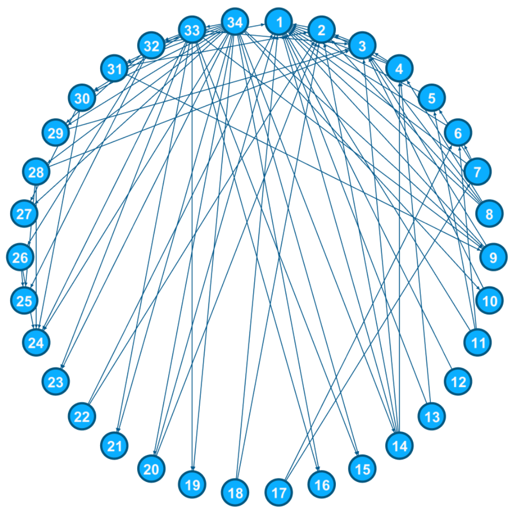

Figure 1: Zachary's Karate Club Graph

Zachary's karate club (ZKC) is a social network of a university karate club, described in the paper "An Information Flow Model for Conflict and Fission in Small Groups" by Wayne W. Zachary.

A social network captures 34 members of a karate club, documenting links between pairs of members who interacted outside the club.

During the study a conflict arose between the officer/ administrator ("John A") and the instructor "Mr. Hi", which led to the split of the club into two.

Half of the members formed a new club around Mr. Hi; members from the other part found a new instructor or gave up karate.

Based on collected data Zachary correctly assigned all but one member of the club to the groups they actually joined after the split. You could read more about it here `https://www.jstor.org/stable/3629752`, and here `https://commons.wikimedia.org/wiki/File:Social_Network_Model_of_Relationships_in_the_Karate_Club.png`

1 Implement all the TODOs in `q_zkc.ipynb`.

2 Comment on what you have learned

2. Justifying Scaled-Dot Product Attention

Suppose $q, k \in \mathbb{R}^d$ are two random vectors with $q, k\ N(\mu, \sigma^2 I)$, where $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^+$. In other words, each component q_i of q is drawn from a normal distribution with mean μ and stand deviation σ, and the same if true for k.

(a) Define $\mathbb{E}[q^T k]$ in terms of μ, σ and d.

(b) Considering a practical case where $\mu = 0$ and $\sigma = 1$, define $\text{Var}(q^T k)$ in terms of d.

(c) Continue to use the setting in part (b), where $\mu = 0$ and $\sigma = 1$. Let s be the scaling factor on the dot product. Suppose we want $\mathbb{E}[\frac{q^T k}{s}]$ to be 0, and $\text{Var}(\frac{q^T k}{s})$ to be $\sigma = 1$. What should s be in terms of d?

3. Kernelized Linear Attention

The softmax attention is widely adopted in transformers (Luong et al., 2015; Vaswani et al., 2017), however the $\mathcal{O}\left(N^2\right)$ (N stands for the sequence length) complexity in memory and computation often makes it less desirable for processing long document like a book or a passage, where the N could be beyond thousands. There is a large body of the research studying how to resolve this [1].

Under this context, this question presents a formulation of attention via the lens of the kernel. A large portion of the context is adopted from Tsai et al. (2019). In particular, attention can be seen as applying a kernel over the inputs with the kernel scores being the similarities between inputs. This formulation sheds light on individual components of the transformer's attention, and helps introduce some alternative attention mechanisms that replaces the "softmax" with linearized kernel functions, thus reducing the $\mathcal{O}\left(N^2\right)$ complexity in memory and computation.

We first review the building block in the transformer. Let $x \in \mathbb{R}^{N \times F}$ denote a sequence of N feature vectors of dimensions F. A transformer Vaswani et al. (2017) is a function $T : \mathbb{R}^{N \times F} \to \mathbb{R}^{N \times F}$ defined by the composition of L transformer layers $T_1(\cdot), \ldots, T_L(\cdot)$ as follows,

$$T_l(x) = f_l(A_l(x) + x). \tag{1}$$

The function $f_l(\cdot)$ transforms each feature independently of the others and is usually implemented with a small two-layer feedforward network. $A_l(\cdot)$ is the self attention function and is the only part of the transformer that acts across sequences.

We now focus on the the self attention module which involves softmax. The self attention function $A_l(\cdot)$ computes, for every position, a weighted average of the feature representations of all other positions with a weight proportional to a similarity score between the representations. Formally, the input sequence x is projected by three matrices $W_Q \in \mathbb{R}^{F \times D}$, $W_K \in \mathbb{R}^{F \times D}$ and $W_V \in \mathbb{R}^{F \times M}$ to corresponding representations Q, K and V. The output for all positions, $A_l(x) = V'$, is computed as follows,

$$\begin{aligned} Q = xW_Q, K = xW_K, V = xW_V, \\ A_l(x) = V' = \text{softmax}(\frac{QK^T}{\sqrt{D}})V. \end{aligned} \tag{2}$$

Note that in the previous equation, the softmax function is applied rowwise to QK^T. Following common terminology, the Q, K and V are referred to as the "queries", "keys" and "values" respectively.

Equation 2 implements a specific form of self-attention called softmax attention where the similarity score is the exponential of the dot product between a query and a key. Given that subscripting a matrix with i

[1] https://huggingface.co/blog/long-range-transformers

returns the i-th row as a vector, we can write a generalized attention equation for any similarity function as follows,

$$V_i' = \frac{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right)}. \tag{3}$$

Equation 3 is equivalent to equation 2 if we substitute the similarity function with $\text{sim}_{\text{softmax}}(q, k) = \exp(\frac{q^T k}{\sqrt{D}})$. This can lead to

$$V_i' = \frac{\sum_{j=1}^{N} \exp(\frac{Q_i^T K_j}{\sqrt{D}}) V_j}{\sum_{j=1}^{N} \exp(\frac{Q_i^T K_j}{\sqrt{D}})}. \tag{4}$$

For computing the resulting self-attended feature $A_l(x) = V'$, we need to compute all V_i' $i \in 1, ..., N$ in equation 4.

(a) Determine what conditions on the denominator in equation 3 would prevent V_i from blowing up.

(b) The definition of attention in equation 3 is generic and can be used to define several other attention implementations.

  i. One potential attention variant is the "polynomial kernel attention", where the similarity function as $\text{sim}\left(q, k\right)$ is measured by polynomial kernel $\mathcal{K}$ [2]. **Considering a special case for a "quadratic kernel attention" that the degree of "polynomial kernel attention" is set to be 2, derive the $\text{sim}\left(q, k\right)$ for "quadratic kernel attention". (NOTE: any constant factor is set to be 1.)** .

  ii. One benefit of using kernelized attention is that we can represent a kernel using a feature map $\phi\left(\cdot\right)$ [3]. **Derive the corresponding feature map $\phi\left(\cdot\right)$ for the quadratic kernel.**

  iii. **Considering a general kernel attention, where the kernel can be represented using feature map that $\mathcal{K}(q, k) = (\phi\left(q\right)^T \phi\left(k\right))$, rewrite kernel attention of equation 3 with feature map $\phi\left(\cdot\right)$.**

(c) i. We can rewrite the softmax attention in terms of equation 3 as equation 4. **For all the V_i' ($i \in \{1, ..., N\}$), derive the computational cost and memory requirement of the above softmax attention in terms of sequence length N, D and M. (NOTE: for memory requirement, think that we need to store any intermediate results for backpropagation, including all Q, K, V)**

  ii. Assume we have a kernel $\mathcal{K}$ as the similarity function and the kernel can be represented with a feature map $\phi\left(\cdot\right)$, we can rewrite equation 3 with $\text{sim}\left(x, y\right) = \mathcal{K}(x, y) = \left(\phi\left(Q_i\right)^T \phi\left(K_j\right)\right)$ in part (b). We can then further simplify it by making use of the associative property of matrix multiplication to

$$V_i' = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right)}. \tag{5}$$

  Note that the feature map $\phi\left(\cdot\right)$ is applied row-wise to the matrices Q and K.
  Considering using a linearized polynomial kernel $\phi\left(x\right)$ of degree 2, derive the computational cost and memory requirement of this kernel attention as in (5).

[2] https://en.wikipedia.org/wiki/Polynomial_kernel
[3] https://en.wikipedia.org/wiki/Kernel_method

4. Auto-encoder : Learning without Labels

So far, with supervised learning algorithms we have used labelled datasets $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ to learn a mapping f_θ from input x to labels y. In this problem, we now consider algorithms for *unsupervised learning*, where we are given only inputs x, but no labels y. At a high-level, unsupervised learning algorithms leverage some structure in the dataset to define proxy tasks, and learn *representations* that are useful for solving downstream tasks.

Autoencoders present a family of such algorithms, where we consider the problem of learning a function $f_\theta : \mathbb{R}^m \to \mathbb{R}^k$ from input x to a *intermediate representation* z of x (usually $k \ll m$). For autoencoders, we use reconstructing the original signal as a proxy task, i.e. representations are more likely to be useful for downstream tasks if they are low-dimensional but encode sufficient information to approximately reconstruct the input. Broadly, autoencoder architectures have two modules:

- An $\mathrm{encoder}$ $f_\theta : \mathbb{R}^m \to \mathbb{R}^k$ that maps input x to a representation z.
- A $\mathrm{decoder}$ $g_\phi : \mathbb{R}^k \to \mathbb{R}^m$ that maps representation z to a reconstruction $\hat{x}$ of x.

In such architectures, the parameters (θ, ϕ) are learnable, and trained with the learning objective of minimizing the reconstruction error $\ell(\hat{x}_i, x_i) = \|x - \hat{x}\|_2^2$ using gradient descent.

$$\theta_{\text{enc}}, \phi_{\text{dec}} = \operatorname*{argmin}_{\Theta,\Phi} \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{x}_i, x_i)$$
$$\hat{x} = g_\phi(f_\theta(x))$$

Note that above optimization problem does not require labels $\mathbf{y}$. In practice however, we would want to use the *pretrained* models to solve the *downstream* task at hand, e.g. classifying MNIST digits. *Linear Probing* is one such approach, where we fix the encoder weights, and learn a simple linear classifier over the features $z = \mathrm{encoder}(x)$.

(a) Designing AutoEncoders

In the accompanying notebook `q_ae.ipynb` we explore the following variants of an autoencoder architecture on a synthetic & MNIST dataset to learn representations, and compare the performance of a linear classifier with these features.

The notebook implements three different class of autoencoder architectures:

(1) **Vanilla AutoEncoder**
For high-dimensional input $x \in \mathbb{R}^m$, an interpretation of autoencoder is that of *dimensionality reduction*. Here, the encoder f_θ is a *lossy* function that maps x to lower dimensional $z \in \mathbb{R}^k$ when $k < m$. The decoder $g_\phi : \mathbb{R}^k \to \mathbb{R}^m$ is a function that maps z to a reconstruction $\hat{x}$ of x.

(2) **Denoising AutoEncoder**
To learn robust lower-dimensional representations, we can add noise to the input x before passing it to the encoder. This is known as a *denoising autoencoder*.

(3) **Masked AutoEncoder**
An alternative to adding noise to the input is to *mask* the input. That is, we randomly set some of the entries of x to zero (or a fixed special value) before passing to the encoder, and try decoding the full signal x. This is known as a *masked autoencoder*, and the proxy task is often referred to as *inpainting*.

After filling the `TODO`s in the notebook, "pretrain" models with different unsupervised learning objectives on given training set. Evaluate representation quality using a linear classifier with these features on the "downstream" classification task, for the following architectures:

(i) **Linear Autoencoders** : In these architectures don't use any non-linearities (stacking multiple layers is allowed). Train models with different bottlenecks `k ∈ [5, 10, 100, 1000]`.

(ii) **Non-Linear** : AutoEncoders that use non-linearities (stacking multiple layers is allowed), train models with different bottleneck dimensions $k \in$ `[5, 10, 100, 1000]`, `ReLU` activation.

(b) PCA & AutoEncoders

In the case where the encoder f_θ, g_ϕ are linear functions, the model is termed as a *linear autoencoder*. In particular, assume that we have data $x_i \in \mathbb{R}^m$ and consider two weight matrices: an encoder $W_1 \in \mathbb{R}^{k \times m}$ and decoder $W_2 \in \mathbb{R}^{m \times k}$ (with $k < m$). Then, a linear autoencoder learns a low-dimensional embedding of the data $\mathbf{X} \in \mathbb{R}^{m \times n}$ (which we assume is zero-centered without loss of generality) by minimizing the objective,

$$\mathcal{L}(W_1, W_2; \mathbf{X}) = \frac{1}{n}||\mathbf{X} - W_2 W_1 \mathbf{X}||_F^2 \tag{6}$$

We will assume $\sigma_1^2 > \cdots > \sigma_k^2 > 0$ are the k largest eigenvalues of $\frac{1}{n}\mathbf{X}\mathbf{X}^\top$. The assumption that the $\sigma_1, \ldots, \sigma_k$ are positive and distinct ensures identifiability of the principal components, and is common in this setting. Therefore, the top-k eigenvalues of $\mathbf{X}$ are $S = \mathrm{diag}(\sigma_1, \ldots, \sigma_\mathrm{k})$, with corresponding eigenvectors are the columns of $\mathbf{U}_k \in \mathbb{R}^{m \times k}$. A well-established result from (Baldi & Hornik, 1989) shows that principal components are the unique optimal solution to linear autoencoders (up to sign changes to the projection directions). In this subpart, we take some steps towards proving this result.

(i) Write out the first order optimality conditions that the minima of Eq. 6 would satisfy.

(ii) Show that the principal components $\mathbf{U}_k$ satisfy the optimality conditions outlined in (i).

5. Beam Search

This problem will also be covered in discussion.

When making predictions with an autoregressive sequence model, it can be intractable to decode the true most likely sequence of the model, as doing so would require exhaustively searching the tree of all $O(M^T)$ possible sequences, where M is the size of our vocabulary, and T is the max length of a sequence. We could decode our sequence by greedily decoding the most likely token each timestep, and this can work to some extent, but there are no guarantees that this sequence is the actual most likely sequence of our model.

Instead, we can use beam search to limit our search to only candidate sequences that are the most likely so far. In beam search, we keep track of the k most likely predictions of our model so far. At each timestep, we expand our predictions to all of the possible expansions of these sequences after one token, and then we keep only the top k of the most likely sequences out of these. In the end, we return the most likely sequence out of our final candidate sequences. This is also not guaranteed to be the true most likely sequence, but it is usually better than the result of just greedy decoding.

The beam search procedure can be written as the following pseudocode:

Algorithm 1 Beam Search

```
for each time step $t$ do
    for each hypothesis $y_{1:t-1,i}$ that we are tracking do
        find the top $k$ tokens $y_{t,i,1}$,...,$y_{t,i,k}$
    end for
    sort the resulting $k^2$ length $t$ sequences by their total log-probability
    store the top $k$
    advance each hypothesis to time $t+1$
end for
```

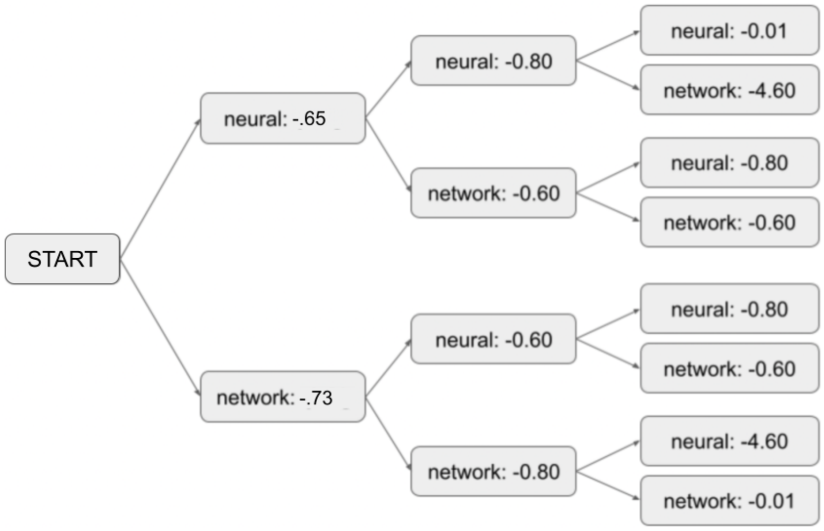

Figure 2: The numbers shown are the decoder's log probability prediction of the current token given previous tokens.

We are running the beam search to decode a sequence of length 3 using a beam search with $k = 2$. Consider predictions of a decoder in Figure 2, where each node in the tree represents the next token **log probability** prediction of one step of the decoder conditioned on previous tokens. The vocabulary consists of two words: "neural" and "network".

(a) **At timestep 1, which sequences is beam search storing?**

(b) **At timestep 2, which sequences is beam search storing?**

(c) **At timestep 3, which sequences is beam search storing?**

(d) **Does beam search return the overall most-likely sequence in this example? Explain why or why not.**

(e) **What is the runtime complexity of generating a length-T sequence with beam size k with an RNN?** Answer in terms of T and k. .

6. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

References

Baldi, P. and Hornik, K. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, 2015.

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4344–4353, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Contributors:

- Jerome Quenum.
- Olivia Watkins.
- Anant Sahai.
- Sheng Shen.
- David M. Chan.
- Shaojie Bai.
- Angelos Katharopoulos.
- Hao Peng.
- Kumar Krishna Agrawal.
- CS 182 Staff from past semesters.