CS 189 Introduction to Machine Learning HW6
Spring 2022 Marvin Zhang

Due 5/3/22 at 11:59pm

- Homework 6 consists of both written and coding questions.
- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted for the written questions.
- In all of the questions, **show your work**, not just the final answer.

Deliverables:

1. Submit a PDF of your homework ,**with an appendix listing all your code**, to the Gradescope assignment entitled "HW6 Write-Up". **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
    - In your write-up, please state with whom you worked on the homework. This should be on its own page and should be the first page that you submit.
    - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadvertently cheats. *"I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."*
    - **Replicate all your code in an appendix**. Begin code for each coding question in a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from appendix to correct questions.

1 Administrivia (2 points)

1. Please fill out the Check-In Survey if you haven't already. Please write down the 10 digit alphanumeric code you get after completing the survey.

2. **Declare and sign the following statement:**

   "I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

   Signature: ______________

   While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., anything outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are particularly severe

2 Fundamentals of Convolutional Neural Networks (8 points)

(a) **1D Convolution (2 points)**

Consider a single convolutional layer, where your input is a sequence of length 200. In other words, the input is a 200×1 tensor. Your convolution has:

- Filter size: 4×1
- Filters: 8
- Stride: 2
- No padding

What is the number of parameters (weights) in this layer, including a bias term?

What is the shape of the output tensor?

(b) **2D Convolution (2 points)**

Consider a single convolutional layer, where your input is a 28×28 pixel, RGB image. In other words, the input is a $28 \times 28 \times 3$ tensor. Your convolution has:

- Filter size: $4 \times 4 \times 3$
- Filters: 8
- Stride: 2
- No padding

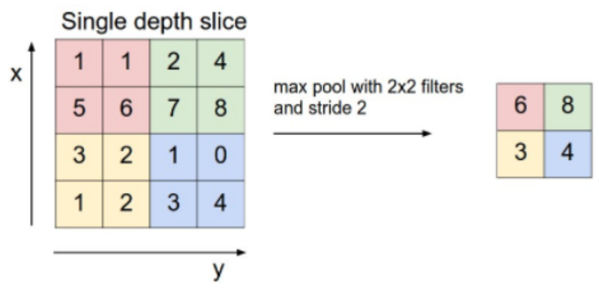
What is the number of parameters (weights) in this layer, including a bias term?

What is the shape of the output tensor?

(c) **Max/Average Pooling (2 points)**

Pooling is a downsampling technique for reducing the dimensionality of a layer's output. Pooling iterates across patches of an image similarly to a convolution, but pooling and convolutional layers compute their outputs differently: given a pooling layer B with preceding layer A, the output of B is some function (such as the max or average functions) applied to patches of A's output.

Below is an example of max-pooling on a 2-D input space with a 2×2 filter (the max function is applied to 2×2 patches of the input) and a stride of 2 (so that the sampled patches do not overlap):



Average pooling is similar except that you would take the average of each patch as its output instead of the maximum.

Consider the following two matrices:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

Apply 3×3 average pooling with a stride of 2 to each of the above matrices.

Apply 3×3 max pooling with a stride of 2 to each of the above matrices.

(d) **(2 points)** Consider a scenario in which we wish to classify a dataset of images that contain different shapes, taken at various angles/locations and containing small amounts of noise (e.g. some pixels may be missing). Why might pooling be advantageous given these distortions in our dataset? Discuss both max and average pooling.

3 Fundamentals of Attention Mechanisms (17 points)

(a) **(5 points) Self-Attention by Hand**

Suppose we are given the following keys, values, and queries. The horizontal axis represents the sequence time step. For example, the first vector in the key matrix is k_1 (key at time step 1), the second k_2 (key at time step 2), and the third k_3 (key at time step 3).

Keys:

$$\left\{ \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 \\ -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \right\}$$

Values:

$$\left\{ \begin{bmatrix} 5 \\ 4 \\ 5 \end{bmatrix}, \begin{bmatrix} 3 \\ 5 \\ 5 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} \right\}$$

Query:

$$\left\{ \begin{bmatrix} 1 \\ 0 \\ 4 \end{bmatrix}, \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}, \begin{bmatrix} 4 \\ 6 \\ 1 \end{bmatrix} \right\}$$

Calculate a_2, which is the output of the self-attention mechanism at time step 2. You can leave your answer as a linear combination between the softmax outputs and value vectors.

Hint: If you are stuck, refer to the last slide of Transformers Lecture 1.

(b) **(2 points) Masking in Self-Attention**

Sometimes, we do not want to perform self-attention between a given query and all the keys. For instance, if we are trying to use attention to create a language generation model, it does not make sense for q_2 to look at k_3 and v_3, because then the model can "cheat" during training when it needs to predict the word at time step 2 by looking at information from time step 3. Describe one way we can "mask" the v_3 component of a_2.

(c) **Importance of Scaled Dot Product Attention**

In practice, we scale each dot product $k_t^T q$ by a factor of $\sqrt{d_k}$, where d_k is the dimensionality of the keys. This is called *scaled dot product attention*. In this part, we will prove why we perform this scaling. Suppose we are performing a dot product between a key k and query q, where $k, q \in \mathcal{R}^d$ and $k, q \sim \mathcal{N}(\mu, \sigma^2 I)$

(a) **(2 points)** Compute $E[q^T k]$ in terms of μ, σ, and d

(b) **(6 points)** Compute Compute $Var[q^T k]$ in terms of μ, σ, and d

Hint: Use the trace property that $Tr(abcd) = Tr(bcda) = Tr(cdab) = Tr(dabc)$

(c) **(2 points)**Based on the variance $Var[q^T k]$, explain why we need to scale the dot product by $\sqrt{d}$

4 Vision Transformer (20 points)

Transformers applied directly to sequences of image patches can also perform very well on image classification tasks. Vision Transformer (ViT) (`https://arxiv.org/abs/2010.11929`) attains results comparable to state-of-the-art convolutional networks while requiring fewer computational resources. Here we explore applying Vision Transformer to the same handwritten digit classification task on the MNIST dataset.

The starter code can be found here. Note that you need to sign in with your UC Berkeley email to view it. Make a copy of the notebook to begin the assignment. First, we start with the building blocks for Transformer models.

(a) **(10 points) Scaled dot-product attention**

Scaled dot-product attention (`https://arxiv.org/abs/1706.03762`) is an attention mechanism where the dot products are scaled down by $\sqrt{d_k}$, the squared root of the query dimension. Formally for a sequence of length T, we have a query matrix $Q \in \mathbb{R}^{T\times d_k}$, a key $K \in \mathbb{R}^{T\times d_k}$ and a value $V \in \mathbb{R}^{T\times d_v}$ and calculate the attention matrix as:

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

The resulting attention matrix takes the following form as described in the lecture:

$$\begin{bmatrix} - & a_1^T & - \\ - & a_2^T & - \\ & \cdots & \\ - & a_T^T & - \end{bmatrix},$$

where each vector a_i describes how the i-th position in the sequence attends to all the other positions. For example a_{ij} is the relative attention weight for the i-th position attending to the j-th position.

The query Q, the key K, and the value V are each projected from the hidden state through different projection matrices (no bias in the projections). In general, d_k and d_v could be different dimensions, but in the scope of this problem they will be the same (`inner_dim` in the Colab notebook).

Deliverables. Implement (single-head) scaled dot-product attention as a module in PyTorch based on the template in the code sample, and include your code in your writeup. You may **only** use the following PyTorch functions that start with `nn.`:

- `Linear`: A fully-connected layer
- `Softmax` (activation): Rescales input so that it can be interpreted as a (discrete) probability distribution.
- `Dropout`: Takes some probability and at every iteration sets weights to zero at random with that probability (effectively regularization)

You may use other PyTorch builtin functions, such as `torch.einsum`, but not anything else from the `nn` module. For matrix operations, you may also find `torch.matmul` and `torch.transpose` useful. Note that, in the call to `forward`, `x` will have an additional mini-batch dimension in the front that your implementation needs to handle.

(b) **(5 points) Transformer layers**

The implementation for multihead attention is provided to you and uses your implementation from part (a). In PyTorch, layer normalization is available as `torch.nn.layernorm`. Putting together these building blocks, implement the Transformer encoder layer as a PyTorch module, based on the template provided in the Colab notebook.

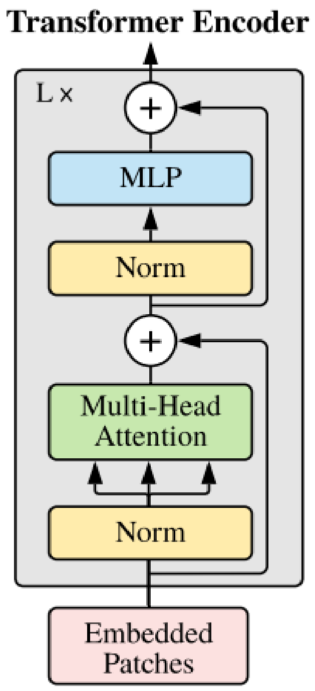


Figure 1: Each layer in the Transformer encoder consists of layer normalization, multi-head attention, another layer normalization, and a fully-connected MLP.

Residual connections. Within each layer, we are applying residual connections twice (as illustrated by "+" in Figure 1), once for multihead attention and once for the MLP.

The basic idea behind residual connections is to make each layer close to the identity map. Traditional building blocks of neural networks typically look like an affine transformations A with a nonlinearity in the middle:

$$f(x) = \text{ReLU}(A(x))$$

Residual networks modify these building blocks by adding the input x back to the output:

$$h(x) = x + B(\text{ReLU}(A(x)))$$

In this example, say x is of input dimension d_i, and A is an affine transformation that maps the input to a hidden vector of dimension d_h, *i.e.*, $A : \mathbb{R}^{d_i} \to \mathbb{R}^{d_h}$. Then, B is an affine transformation that maps back to the input dimension, *i.e.*, $B : \mathbb{R}^{d_h} \to \mathbb{R}^{d_i}$, so that $B(\text{ReLU}(A(x)))$ has the same dimension as the input x.

Thinking about the computation graph, we create a connection from the input to the output of the building block that skips the transformation. Such connections are therefore called *skip connections*. This seemingly innocuous change was hugely successful and allowed for model depths not seen previously.

Deliverables. Turn in the code you implemented for the encoder layer module. Remember to include layer normalization, dropout, and residual connections at appropriate places. See Figure 1 for the illustration of each layer in the Transformer encoder. Note that dropout is used after the attention, inside the MLP, and after the MLP.

(c) **(5 points) Vision Transformer on MNIST**

Using the transformer layer module implemented above, train the Vision Transformer model on MNIST for 10 epochs using batch size 32 (see provided Vision Transformer code in the Colab notebook). For the Vision Transformer, use 6 Transformer layers each with 8 heads, patch size 7, query dimension 64, and MLP hidden dimension 128.

Deliverables. What test accuracy does this version of the Vision Transformer model achieve? Plot the loss trajectories on both the training set and the validation set.

It should take 30-60 minutes to train the Vision Transformer model and the accuracy should be at least 94% (± 1%).