Recitation 7
Deep Learning

10-601: Introduction to Machine Learning

11/08/2024

1 PyTorch Basics

1.1 Colab Notebook

See the following Colab Notebook:
https://colab.research.google.com/drive/1yd4v5hVRDByk58xfcut2bHlbj2PYx3fF

2 Transformers

2.1 Concepts

1. What is a word embedding?

   - A word embedding is a vector representation of a word. Ideally words with similar meanings should produce vectors which are close to each other. The difficulty of this task is that context affects the meaning of a word.

2. What is attention?

   - In language models, attention is a weighting used to quantify how much other words in a sentence affect the context of a single word.

     $$\mathbf{x}_i' = \sum_j \alpha_{ij} \mathbf{x}_j$$

     Here, $\mathbf{x}_i'$ is the contextualized vector for word i, $\mathbf{x}_j$ is the vector embedding for any arbitrary word j in a sentence, and α_{ij} is the attention weight for word j.

   - For example, in the sentence 'I went to the bank to deposit my paycheck', we know bank is referring to the financial institution. However, 'bank' by itself could also refer to the side of the river. Thus, the context of this sentence tells us that bank refers to the financial institution. Specifically, the words 'deposit' and 'paycheck' indicate this. Therefore, these would have high attention weights when determining the contextualized vector for 'bank'.

3. How do you calculate attention?

   - Attention is meant to represent the relatedness between two words. One easy way to calculate this is to take the dot products of two vectors. In the case of a transformer, these vectors are the query and key vectors.

- We also want these attention weights to be fractions, to avoid $\mathbf{x}_i'$ from becoming very large or very small. This is why we use the softmax, so that the attention weights become a valid probability distribution.

$$\text{Attention}(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{k}^T}{\sqrt{d_k}}\right)\mathbf{v}$$

4. What are query, key, value vectors?
   - Query vector represents what each word is looking for when determining context.
   - Key vector represent how related a word is to a query (how relevant am I to your question?).
   - Value vector represents value of a word.
   - Analogous to looking up things in a database. I want a fancy lamp (query). Database contains keys which are easy to compare against query (i.e. small lamp, big table, twin bed). Value is the actual lamp (REGNSKUR IKEA lamp).
5. Why are there different attention heads?
   - We want to model multiple different relationships between words.
   - 'The dog chased the cat because it was hungry.' How can we figure out what 'it' is referring to?
   - We know 'dog', 'chased', 'cat' and 'hungry' are all relevant to determining what 'it' is, but it's difficult to imagine a single weighted addition can capture all of the context describing what 'it' is.
   - Easier to believe one head determines from 'dog' and 'cat' that 'it' represents one of these animals. A separate head would then determine from 'chased' and 'hungry' that the dog is hungry. Ideally, combining these insights using the feed-forward layer produces a modified 'it' vector which knows it refers to the dog.

Questions:

1. Let's say you are using a transformer for an image captioning model. Your model would therefore take in an image, and produce a text caption for the image. For each of the query, key and value vectors, which modality do they come from, image or text?
   - Query: Image or Text
   - Key: Image or Text
   - Value: Image or Text
2. You are using a text transformer to fill in the blank word in the following sentence. 'I want to go ____ in the park later. It's really nice out and I haven't worked out in a while.' What kinds of relationships might different heads within the transformer be considering to determine the best word to fill in the blank?

2.2 Parameters

Here is the illustration of a self-attention block from lecture:

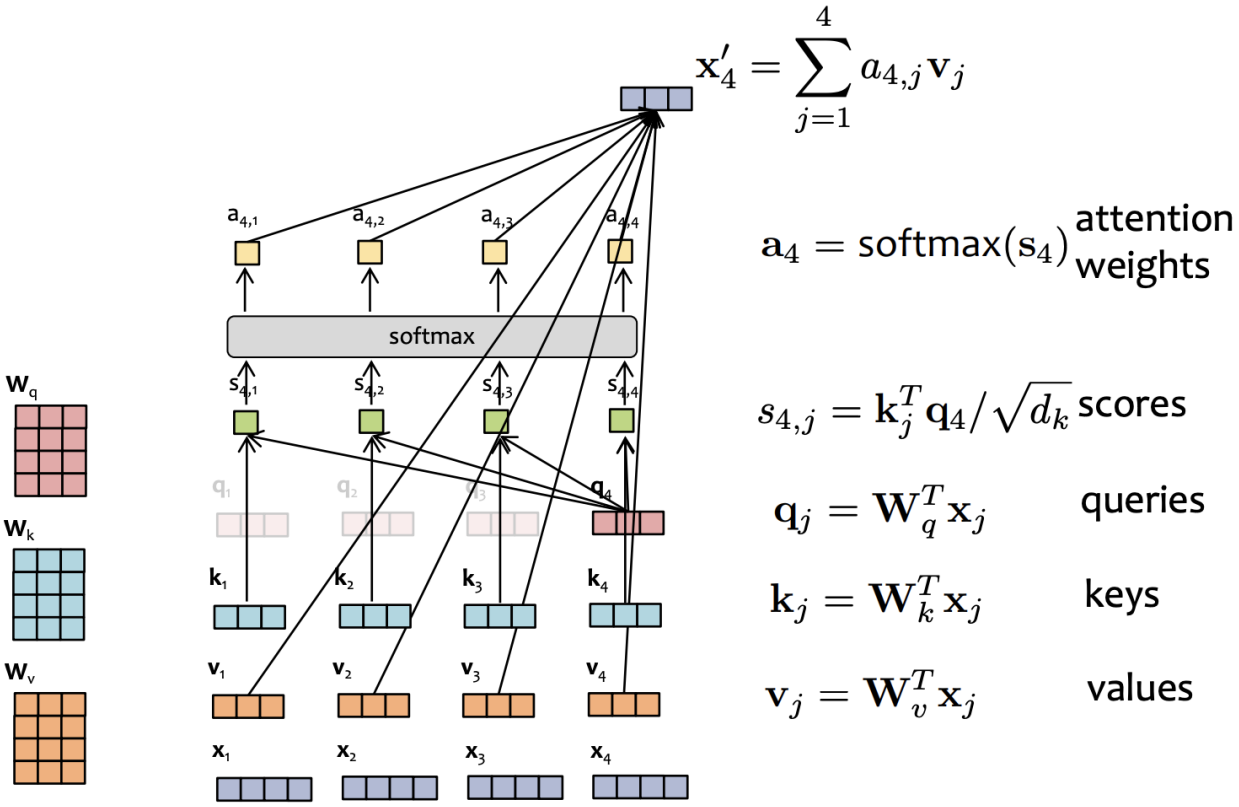


Figure 1: Self-attention block, illustrated

Let's define some terms:

- d_{model} is the dimension of each input vector $\mathbf{x}_i$
- d_k is the dimension of the query, key and value vectors.
- h is the number of attention heads

1. What is the shape of the W_q, W_k, W_v matrices (without biases)?

2. How many parameters are there in each of the W_q, W_k, W_v matrices (including biases)?

3. How many parameters are there in total for a multiheaded attention block, as illustrated in lecture?

4. Where else in the transformer are there additional parameters?

5. What is the big-O relationship between the number of parameters for a multiheaded attention block and the length of each input vector?

2.3 Links

Visualization of transformer https://jalammar.github.io/illustrated-transformer/

3 Convolutional Neural Networks

3.1 Concepts

1. What are filters?
    - Filters (also called kernels) are feature extractors in the form of a small matrix used in convolutional neural layers. They usually have a width, height, depth, stride, padding, channels (output) associated with them.
2. What are convolutions?
    - We sweep the filter around the input tensor and take element-wise product sums based on factors such as filter size, stride, padding. These output product-sums form a new tensor, which is the output of a convolutional layer.
3. How do we calculate the output shape of a convolution?
    - Given input width W_{in}, kernel width K_W, padding P, and stride S, the output width W_{out} can be calculated as:

$$W_{out} = \lfloor \frac{W_{in} - K_w + 2 \times P}{S} \rfloor + 1$$

    .
    - Output height can be calculated similarly.
4. What are some benefits of CNNs over fully connected (also called dense) layers?
    - Good for image-related machine learning (learns the kernels that do feature engineering)
    - Pseudo translational invariance
    - Parameter efficient
5. How does the number of channels vary through convolutional networks?
    - Each convolution filter will have as many channels as the input, and there will be as many filters as there are output channels.
    - Pooling and activations often maintain the number of channels.

3.2 Dance Dance Convolution

Consider the following 4 x 4 image and 2x2 filter below.

1	3	-2	4
0	8	6	5
2	1	-9	0
4	-1	3	7

1	2
-2	-1

1. Assume that there is no padding and stride = 1. What are the dimensions of the output, and what is the value in the bottom right corner of the output image?
2. Now assume that we having padding = 1. Given that, what are the new dimensions of the output, and the new value in the bottom right corner?

3.3 Parameters

Suppose that we want to classify images that belong to one of ten possible classes (i.e. `[cat, dog, bird, turtle, ..., horse]`). The images come in RGB format (one channel for each color), and are downsampled to dimension `128x128`.

Figure 2 illustrates one such image from the MS-COCO dataset[1].

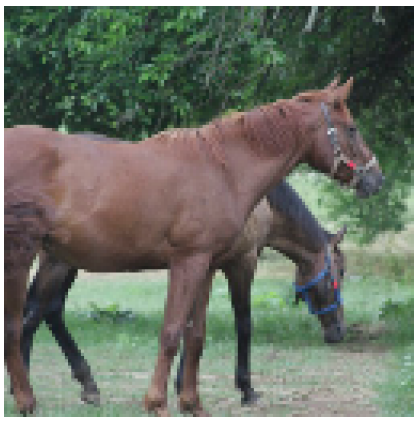


Figure 2: Image of a horse from the MS-COCO dataset, downsampled to 128x128

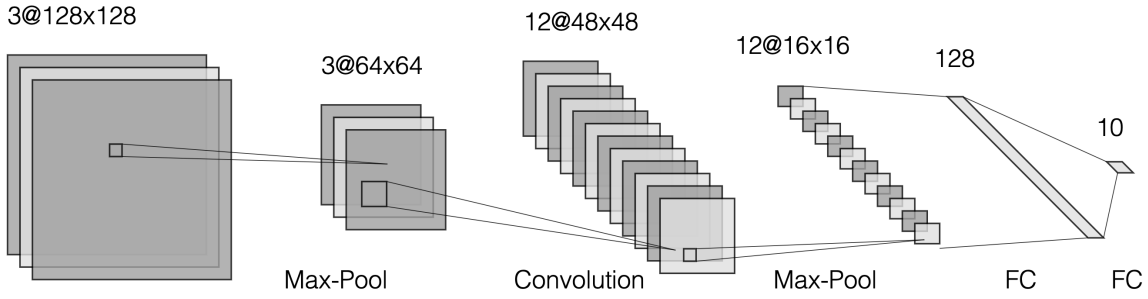
[1] https://cocodataset.org/

We construct a Convolutional Neural Network that has the following structure: the input is first max-pooled with a 2x2 filter with stride 2 and 3 output channels. The results are then sent to a convolutional layer that uses a 17x17 filter of stride 1 and 12 output channels. Those values are then passed through a max-pool with a 3x3 filter with stride 3 and also 12 output channels. The result is then flattened and passed through a fully connected layer (ReLU activation) with 128 hidden units followed by a fully connected layer (softmax activation) with 10 hidden units. We say that the final 10 hidden units thus represent the categorical probability for each of the ten classes. With enough labeled data, we can simply use some optimizer like SGD to train this model through backprogation.

Note: By default, please assume we have bias terms in all neural network layers unless explicitly stated otherwise.

1. Fill the table below with channels and dimensions of the tensors before and after every neural net operation.

Layer / Operation	**Shape**
Input	3@128 × 128
maxpool-1	(a)
conv	(b)
maxpool-2	(c)
flatten	(d)
fully-connected-1	(e)
ReLU	(f)
fully-connected-2	(g)
softmax	(h)



2. How many parameters are in this network for the convolutional components?

3. How many parameters are in this network for the fully connected (also called dense) components?

4. From these parameter calculations, what can you say about convolutional layers and fully connected layers in terms of parameter efficiency[2]? Why do you think this is the case?

3.4 Links

Visualization of convolutional filter sweep steps https://github.com/vdumoulin/conv_arithmetic

Visualization of convolutional filter smooth sweep with outputs https://www.youtube.com/watch?v=f0t-OCG79-U

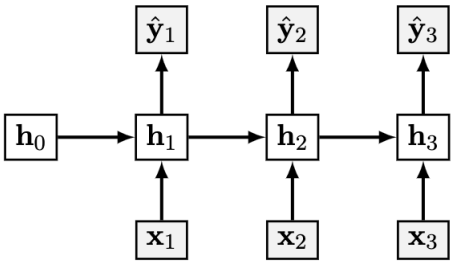
Visualization of neural network layer outputs http://cs231n.stanford.edu/

The architecture used there is (conv→ relu → conv → relu → pool) x3 → fc → softmax

[2]the ratio between the number of parameters from some layer type and the total number of parameters.

4 Recurrent Neural Networks

4.1 Sample RNN



Where the layers and their corresponding weights are given below:

$$\mathbf{x}_t \in \mathbb{R}^3 \qquad \mathbf{W}_{hx} \in \mathbb{R}^{4\times 3}$$
$$\mathbf{h}_t \in \mathbb{R}^4 \qquad \mathbf{W}_{yh} \in \mathbb{R}^{2\times 4}$$
$$\mathbf{y}_t, \hat{\mathbf{y}}_t \in \mathbb{R}^2 \qquad \mathbf{W}_{hh} \in \mathbb{R}^{4\times 4}$$

$$\begin{aligned}\hat{\mathbf{y}}_t &= \sigma(\mathbf{o}_t)\\ \mathbf{o}_t &= \mathbf{W}_{yh}\mathbf{h}_t\\ \mathbf{h}_t &= \psi(\mathbf{z}_t)\\ \mathbf{z}_t &= \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t\end{aligned}$$

Where σ and ψ are activations.

1. Redraw the above diagram in a compact form such that we don't need to unroll it across several timesteps.

4.2 Concepts

1. What are recurrent neural networks?
   - A recurrent neural network (RNN) can be characterized by connections between nodes creating a cycle [3]. Outputs from some nodes can affect subsequent computations. This allows it to exhibit temporal dynamic behavior.

[3] Article linked here.

- the recurrent nature makes them useful when the input is sequential (or temporal).

2. How do they use both inputs and previous outputs?
   - Hidden nodes have two sets of weights, one to process input from the current timestep, and one to process their own outputs from the previous timestep.
3. How do we optimize RNNs?
   - Applying chain rule to the 'unrolled' RNN (as above) is no different than a regular feed forward neural network aside from the fact that the same parameters are repeated throughout the network at each timestep.
   - Called as backpropagation through time (BPTT).