THE DEEP LEARNING DUMP

ERNEST YEUNG ERNESTYALUMNI@GMAIL.COM

If you find these notes useful, feel free to donate directly and easily through PayPal. PayPal does charge a fee, so I also have a Venmo, ernestyalumni, and CashApp (via email mailto:ernestyalumni@gmail.com).

Otherwise, under the *open-source MIT license*, feel free to copy, edit, paste, make your own versions, share, use as you wish.

gmail : ernestyalumni
linkedin : ernestyalumni
twitter : ernestyalumni

Contents

Abstract. Everything Deep Learning, Deep Neural Networks

Part 1. Deep Neural Networks

1. Cross Entropy Loss

See Pytorch "CrossEntropyLoss".

Consider the classification problem such that $y \in S$ where S is some finite set, i.e. $S \in \mathbf{FiniteSet}$. Let $C \equiv$ total number of "classes" or elements in this set S.

For what Pytorch calls `minibatch` in the documentation for "CrossEntropyLoss", let `minibatch` $\equiv N$ for our notation (the reason is we can say we have N samples in our batch).

$d_1, \dots, d_k$ with $K \geq 1$ for the K-dimensional case, with K being the number of additional "features" for a single data point x. We've seen K be denoted D in other literature.

The Pytorch literature says that the input into `torch.nn.CrossEntropyLoss` is a "Tensor" of size (C) for unbatched input, (minibatch, C) or $(\text{minibatch}, C, d_1, d_2, \dots d_K)$. The documentation says the last "being useful for higher dimension inputs". I interpret this as being for a single data point with multiple features.

Consider these 2 examples: for a classification problem with $C = 3$ such as cat, dog, or mouse, if there are no additional features, we could imagine that for training data point X,

$$X \mapsto h \in \mathbb{R}^C \mapsto y,\ y \in S \text{ s.t. } |S| = C$$

where $h \in \mathbb{R}^C$ are the so-called "logits" generated by the "forward pass" (the linear transformations and nonlinear element wise mappings of the model) which then result into a prediction for a $y \in S$. The input is of size (N, C) as there are no further additional features to consider.

Consider a grayscale image (i.e. each pixel has a single value for brightness, 0 to 255). Generalize this to each pixel having a value in $\{0, \dots P_b - 1\}$. If the image has "dimensions" $H \times W = 28 \times 28$ for example, then $d_1 = H$ and $d_2 = W$ and $K = 2$. So in this case the input dimensions are $(N, C, d_1 = H, d_2 = W)$.

2. Gaussian Processes

Yang (2021) for Tensor Programs I[1]

Part 2. Recurrent Neural Networks

3. Long Short Term Memory, LSTM

The NVIDIA CUDNN Documentation, in 7.1.2.8. `cudnnRNNMode_t` says these following equations apply for the 4-gate Long Short-Term Memory (LSTM) network with no peephole connections, and for the default mode of `CUDNN_RNN_DOUBLE_BIAS`:

$$\begin{aligned}
i_t &= \sigma(W_i x_t + R_i h_{t-1} + b_{Wi} + b_{Ri}) = \sigma(W_i x_t + b_{Wi} + R_i h_{t-1} + b_{Ri}) \\
f_t &= \sigma(W_f x_t + R_f h_{t-1} + b_{Wf} + b_{Rf}) = \sigma(W_f x_t + b_{Wf} + R_f h_{t-1} + b_{Rf}) \\
o_t &= \sigma(W_o x_t + R_o h_{t-1} + b_{Wo} + b_{Ro}) = \sigma(W_o x_t + b_{Wo} + R_o h_{t-1} + b_{Ro}) \\
c'_t &= \tanh(W_c x_t + R_c h_{t-1} + b_{Wc} + b_{Rc}) = \tanh(W_c x_t + b_{Wc} + R_c h_{t-1} + b_{Rc}) \\
c_t &= f_c \circ c_{t-1} + i_t \circ c'_t \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned} \tag{1}$$

where $\circ$ denotes an element-wise operation.

Compare this with the expression given in PyTorch, LSTM and they are similar, but $c'_t \equiv g_t$ and $\odot$ denotes the element-wise operation. In Eq. 1,
x_t is input at time t,
i_t is the so-called input gate,
f_t is the so-called forget gate,
g_t is the so-called cell gate,

Date: 19 July 2023.
Key words and phrases. Deep Learning, Deep Neural Networks.

o_t is the so-called output gate,

and

$$\sigma \equiv \text{ sigmoid operator, s.t. } \sigma(x) = \frac{1}{1+\exp(-1)} \tag{2}$$

is the sigmoid function.

Also, output hidden (state of each) layer h_t can be multiplied by a learnable projection matrix $h_t = W_{hr} h_t$.

Let us write Eq. 1 in the following manner:

$$\begin{aligned} &(x_t, h_{t-1}) \xrightarrow{\sigma \odot (W_i, b_{Wi})(R_i, b_{Ri})} && i_t \\ &(x_t, h_{t-1}) \xrightarrow{\sigma \odot (W_f, b_{Wf})(R_f, b_{Rf})} && f_t \\ &(x_t, h_{t-1}) \xrightarrow{\sigma \odot (W_o, b_{Wo})(R_o, b_{Ro})} && o_t \\ &(x_t, h_{t-1}) \xrightarrow{\sigma \odot (W_c, b_{Wc})(R_c, b_{Rc})} && c'_t \equiv g_t \end{aligned} \tag{3}$$

$$\begin{aligned} &(f_t, c_{t-1}, i_t, c'_t) \longrightarrow && c_t \\ &(o_t, c_t) \longrightarrow && h_t \end{aligned}$$

From this we can clearly see that $\forall t \in \{0, 1, \dots T-1$ where T is the *sequence length* (Pytorch denotes this as L = sequence length in the Pytorch LSTM documentation) the inputs in are (x_t, h_{t-1}) and the output is h_t and that c_t is also effectively an output for the next time step $t+1$.

EY: 20230903 It is not clear however, what form does c_{-1} takes that'd be necessary for the computation of c_0 for $t = 0$. While one could maybe make some guess or ansatz about what h_{-1} to use initially and compute i_0, f_0, o_0 to use for some initialized values for the operators W, R, and biases b's, does one have to make a guess or ansatz about c_{-1} as well?

3.1. Interpreting NVIDIA `cudnn` Legacy API.

Let
N = number of samples in a batch
T = length of sequence
D = number of features; size of a (single) input vector

Implied strides would be $(TD, D, 1)$.

L = number of layers, $b = \begin{cases} 1 & \text{if unidirectional} \\ 2 & \text{if bidirectional} \end{cases}$

$(bL, N, \begin{cases} P & \text{if LSTM of} \\ H & \text{hidden size} \end{cases})$

Consider strides: $(\{NP, NH\}, \{P, H\}, 1)$

Cell dimensions or cells of rank R
(bL, N, H)

$x, hx, cx, W \mapsto$ forward $\mapsto y, hy, cy$

$x, hx \mapsto$ backward on W $\mapsto y, \partial W \equiv dW$

if $o = [y, hy, cy] = F(x, hx, cx) = F(z)$, then

backwardData $\mapsto \left(\frac{\partial o_i}{\partial z_j}\right)^T \delta_{\text{out}}$, where $\delta_{\text{out}} \equiv \text{grad} l$ and $|\text{grad} l| = m \times 1$

$\delta_{\text{out}} \equiv dy, dhy, dcy$

gradient results $\left(\frac{\partial o_i}{\partial z_j}\right)^T \equiv dx, dhx, dcx$

$y, dy, hx, dhy, cx, dcy, w \mapsto$ backward on Data $\mapsto dx, dhx, dcx$

Part 3. Transformer Networks

Including *Attention*

4. Transformers

4.1. Input.

See Turner (2023) [2].

Let input data s.t. sequence of N $\mathbf{x}_n^{(0)}$ of dim. D, $n = 0, 1, \dots N-1$, $\mathbf{x}_n^{(0)} \in F^D$, where F is some field (i.e. some data type such as float, double, etc.).

Let matrix $X^{(0)} \in F^{D \times N}$ or $\text{Mat}_F(D, N)$, a sequence of N arrays of dim. D collected into a matrix.

Let $M \in 0, 1, \dots$ i.e. $M \in \mathbb{Z}^+$.

The goal is to map $X^{(0)}$ to $X^{(M)} \in \text{Mat}_F(D, N)$ i.e. $X^{(M)}$ of size $D \times M$ s.t. since $x_n = X^{(M)}_{;n}$ is a vector of features representing the sequence at location of n in the sequence.

4.2. Attention $A^{(m)}$.

Consider output vector at location n, $\mathbf{y}_n^{(m)}$, where

$$\mathbf{y}_n^{(m)} = \mathbf{x}_{n'}^{(m-1)} A^{(m)}_{n'n}, \quad n' = 0, 1, \dots N-1 \tag{4}$$

(Eq. (1) of Turner (2023) [2]), where $A^{(m)}_{n'n} = A^{(m)}$ is called the attention matrix, $A^{(m)} \in \text{Mat}_F(N, N)$ and normalizes over its columns:

$$\sum_{n'=1}^{N} A^{(m)}_{n'n} = 1 \tag{5}$$

4.3. Projection of Q, K, V, queries, keys, and values.

Recall an input $\mathbf{x}_n = X^{(M)}_{;n} \in F^D$. Recall the linear transform resulting so-called queries or query vectors:

$$\mathbf{q}^{(m)}_{h;n} = U^{(m)}_{q;h} \mathbf{x}_n^{(m-1)} \in F^K, \quad U^{(m)}_{q;h} \in \text{Mat}_F(K, D)$$

where $h = 0, 1, \dots H-1$ with H heads in Turner's notation (Turner (2023)[2]). Compare this with NVIDIA's notation, [3], $i = 0, 1, \dots$ `nHeads` $- 1$, so that $H \equiv$ `nHeads`.

Generalize K in dimensiosn $k \times D$ of $U_{q;h}^{(m)}$ to `qSize` $\equiv D_q$, i.e.

$$\mathbf{q} \equiv \mathbf{q}_{h;n}^{(m)} = U_{q;h}^{(m)} \mathbf{x}_n^{(m-1)} \in F^{D_q}, \quad U_{q;h}^{(m)} \in \mathrm{Mat}_F(D_q, D)$$

See 7.2.45. `cudnnSetAttnDescriptor`

References

[1] Greg Yang. "Tensor Programs I: Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes." arXiv:1910.12478v3 8 May 2021
[2] Richard E. Turner. "An Introduction to Transformers". arXiv:2304.10557v3 cs.LG 4 Jul 2023
[3] NVIDIA. "7.2.45 `cudnnSetAttnDescriptor`". cuDNN API Documentation. 7.2.45. `cudnnSetAttnDescriptor`
[4] Hasim Sak, Andrew Senior, Francoise Beaufays. "Long Short-Term Memory Based Recurrent Neural Network Architectures For Large Vocabulary Speech Recognition." `https://arxiv.org/abs/1402.1128`