BPTT DERIVATIONS

ADITYA SINGH

Abstract. We derive the Backpropagation Through Time (BPTT) algorithm. We follow the notation and outline of BPTT from section 10.2 of [1], but an effort is also made to follow PyTorch conventions so that the code in RNN_backprop.py implements the formulas exactly as they are written here. A highlight is an application of the chain rule with tensors in Section 3.8 to derive the gradient of the loss with respect to the embedding table, which is not often discussed in textbooks.

Contents

1. Introduction 1
2. Plan of Attack 3
3. Derivations 3
3.1. Logits 3
3.2. Hidden states 4
3.3. Hidden-to-logit bias vector 4
3.4. Input-to-hidden bias vector 4
3.5. Hidden-to-logit weight matrix 5
3.6. Hidden-to-hidden weight matrix 5
3.7. Input-to-hidden weight matrix 6
3.8. Embedding matrix 6
4. Summary 7
References 7

1. Introduction

We will derive the BPTT algorithm for the RNN depicted in Figure 1. Let τ denote the number of tokens the RNN processes in a single forward pass. We refer to the embeddings of these tokens as $x^{(1)}, \ldots, x^{(\tau)}$, where the superscript denotes the timestamp. For each timestep $t = 1, \ldots, \tau$, a probability vector $\hat{y}^{(t)}$ is outputted over the next token. Let the label for each timestep be denoted as $y^{(t)}$. To compute the loss $L^{(t)}$ at each time step, we take the negative natural log of the component of $\hat{y}^{(t)}$ corresponding to $y^{(t)}$ (cross entropy loss), which we denote as

$$L^{(t)} = -log(\Pr_{model}(y^{(t)}|x^{(1)}, \ldots, x^{(t)})). \tag{1.1}$$

Date: September 2024.

The total loss L is then given by averaging over the losses at each timestep:

$$L = \frac{1}{\tau} \sum_{t=1}^{\tau} L^{(t)}. \tag{1.2}$$

For the forward pass, we assume the hyperbolic tangent activation function. We begin with specifying an initial hidden state $h^{(0)}$. For $t = 1, \ldots, \tau$, we forward propagate as follows:

$$x^{(t)} = E[input^{(t)}] \tag{1.3}$$

$$a^{(t)} = h^{(t-1)} W^\top + x^{(t)} U^\top + b \tag{1.4}$$

$$h^{(t)} = \tanh(a^{(t)}) \tag{1.5}$$

$$o^{(t)} = h^{(t)} V^\top + c \tag{1.6}$$

$$\hat{y}^{(t)} = softmax(o^{(t)}) \tag{1.7}$$

The computational graph for the RNN is shown below:

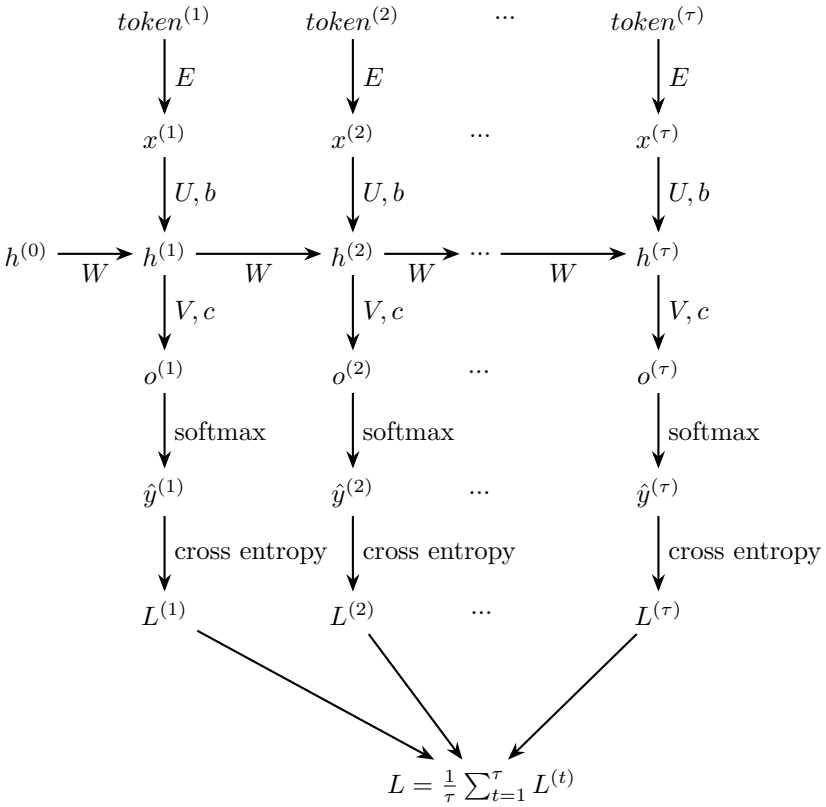

FIGURE 1. Computational graph of the RNN

2. Plan of Attack

Our task is to compute the gradients of the loss L with respect to the hidden-to-logit linear layer parameters (V and c), the embedding-to-hidden linear layer parameters (U and b), the hidden-to-hidden linear layer parameters (W), and the embedding table E.

As an intermediate step, we will compute the gradients of the loss with respect to all the internal nodes – the logits $o^{(1)}, \dots, o^{(\tau)}$ and the hidden states $h^{(1)}, \dots, h^{(\tau)}$ – and then express our desired gradients in terms of these.

We first backpropagate to the logits in parallel. However, to obtain the hidden state partials, parallelization is impossible since the gradient of the loss w.r.t $h^{(t)}$ will depend on the partial w.r.t $h^{(t+1)}$ (see Figure 1). Hence, we will have to sequentially compute $(DL)_{h^{(\tau)}}, \dots, (DL)_{h^{(1)}}$, and it is this restriction that gives the backward pass a $O(\tau)$ runtime (and memory cost) just as the forward pass.

Once the partials with respect to the internal nodes have been computed and stored, obtaining our desired gradients is easy.

3. Derivations

All one dimensional vectors are viewed as row vectors (including the gradient of a scalar-valued function), in line with PyTorch conventions. We use the notation D for derivative, i.e., $(Df)_p$ denotes the Jacobian of the function f at the point p. Many abuses of notation are committed in hopes of a smoother reading experience, such as a scalar minus a vector or the Hadamard product of a column vector and matrix – in all such cases, the operation intended is given by applying PyTorch broadcasting rules.

3.1. **Logits.** Fix $1 \le t \le \tau$, and letting i index the elements of $o^{(t)}$, fix i. We wish to compute $(DL)_{o_i^{(t)}}$. Observe by definition of the softmax function that $o_i^{(t)}$ feeds into each component of $\hat{y}^{(t)}$. Hence,

$$\begin{aligned}(DL)_{o_i^{(t)}} &= \sum_k (DL)_{\hat{y}_k^{(t)}} (D\hat{y}_k^{(t)})_{o_i^{(t)}} \\ &= \sum_k (DL)_{L^{(t)}} (DL^{(t)})_{\hat{y}_k^{(t)}} (D\hat{y}_k^{(t)})_{o_i^{(t)}} \\ &= \frac{1}{\tau} \sum_k (DL^{(t)})_{\hat{y}_k^{(t)}} (D\hat{y}_k^{(t)})_{o_i^{(t)}}. \end{aligned} \tag{3.1}$$

We now divide the analysis into two cases. The first case is if $i = y^{(t)}$. Then $L^{(t)} = -\log(\hat{y}_i^{(t)})$ and so (3.1) becomes

$$(DL)_{o_i^{(t)}} = \frac{1}{\tau}(DL^{(t)})_{\hat{y}_i^{(t)}} (D\hat{y}_i^{(t)})_{o_i^{(t)}} = -\frac{1}{\tau}\frac{1}{\hat{y}_i^{(t)}}(\hat{y}_i^{(t)} - (\hat{y}_i^{(t)})^2) = \frac{1}{\tau}(\hat{y}_i^{(t)} - 1). \tag{3.2}$$

The second case is if $i \neq y^{(t)}$. Then (3.1) becomes

$$(DL)_{o_i^{(t)}} = \frac{1}{\tau}(DL^{(t)})_{\hat{y}_{y^{(t)}}^{(t)}} (D\hat{y}_{y^{(t)}}^{(t)})_{o_i^{(t)}} = -\frac{1}{\tau}\frac{1}{\hat{y}_{y^{(t)}}^{(t)}}(\hat{y}_{y^{(t)}}^{(t)} \hat{y}_i^{(t)}) = \frac{1}{\tau}\hat{y}_i^{(t)}. \tag{3.3}$$

Using indicator notation we may write

$$(DL)_{o_i^{(t)}} = \frac{1}{\tau}(\hat{y}_i^{(t)} - \mathbf{1}_{\{i=y^{(t)}\}}). \tag{3.4}$$

3.2. **Hidden states.** We begin with the hidden state $h^{(\tau)}$ at the final time step. Observe from Figure 1 that the only node in the computational graph it feeds in to is the logits $o^{(\tau)}$. Hence,

$$(DL)_{h^{(\tau)}} = (DL)_{o^{(\tau)}}(Do^{(\tau)})_{h^{(\tau)}} = (DL)_{o^{(\tau)}}V. \tag{3.5}$$

When we are not at the last time step, the hidden state node not only feeds into the logits node at the current time step, but also the hidden state at the next time step (Figure 1). For $t = \tau - 1, \dots, 1$ we have

$$(DL)_{h^{(t)}} = (DL)_{h^{(t+1)}}(Dh^{(t+1)})_{h^{(t)}} + (DL)_{o^{(t)}}(Do^{(t)})_{h^{(t)}}. \tag{3.6}$$

Since $h^{(t+1)}$ is obtained from $h^{(t)}$ via the intermediate variable $a^{(t+1)}$, we have

$$(Dh^{(t+1)})_{h^{(t)}} = (Dh^{(t+1)})_{a^{(t+1)}}(Da^{(t+1)})_{h^{(t)}} = \operatorname{diag}[1 - (h^{(t+1)})^2]W. \tag{3.7}$$

All together, (3.6) becomes

$$\begin{aligned}(DL)_{h^{(t)}} &= (DL)_{h^{(t+1)}}\operatorname{diag}\left[1 - (h^{(t+1)})^2\right]W + (DL)_{o^{(t)}}V \\ &= (DL)_{h^{(t+1)}}\left(\left[1 - (h^{(t+1)})^2\right] \odot W\right) + (DL)_{o^{(t)}}V. \end{aligned} \tag{3.8}$$

3.3. **Hidden-to-logit bias vector.** Now that that the partials with respect to the internal nodes have been computed, we begin computing the gradients with respect to the parameters we actually wish to optimize. Recall the mantra that "the gradient with respect to a repeated parameter is the sum of the gradients with respect to the parameter each time it appears." Since the RNN shares weights and biases across all time steps, each of these parameters appear τ times. In computing $(DL)_c$, we denote the usage of the bias vector c at each time step via the dummy variable $c^{(t)}$, so that

$$(DL)_c = \sum_{t=1}^{\tau}(DL)_{c^{(t)}}. \tag{3.9}$$

More formally, one may visualize a computational graph in which a node c is connected to τ separate nodes $c^{(t)}$, $t = 1, \dots, \tau$, via the identity map, where each $c^{(t)}$ is used only at its current time step. Then (3.9) follows via the chain rule. The only node $c^{(t)}$ connects to is $o^{(t)}$ (Figure 1), so

$$(DL)_c = \sum_t (DL)_{c^{(t)}} = \sum_t (DL)_{o^{(t)}}(Do^{(t)})_{c^{(t)}}. \tag{3.10}$$

But $(Do^{(t)})_{c^{(t)}}$ is just the identity matrix, so (3.10) reduces to

$$(DL)_c = \sum_t (DL)_{c^{(t)}} = \sum_t (DL)_{o^{(t)}}. \tag{3.11}$$

3.4. **Input-to-hidden bias vector.** The only node $b^{(t)}$ connects to is $h^{(t)}$ (Figure 1), so

$$\begin{aligned}(DL)_b &= \sum_t (DL)_{b^{(t)}} = \sum_t (DL)_{h^{(t)}}(Dh^{(t)})_{a^{(t)}}(Da^{(t)})_{b^{(t)}} \\ &= \sum_t (DL)_{h^{(t)}}\operatorname{diag}\left[1 - (h^{(t)})^2\right]. \end{aligned} \tag{3.12}$$

3.5. **Hidden-to-logit weight matrix.** The only node $V^{(t)}$ feeds into is $o^{(t)}$ (Figure 1), so

$$(DL)_V = \sum_t (DL)_{V^{(t)}} = \sum_t \sum_i (DL)_{o_i^{(t)}} (Do_i^{(t)})_{V^{(t)}}. \tag{3.13}$$

Since $o_i^{(t)} = \sum_j v_{ij} h_j^{(t)}$, we see that $D(o_i^{(t)})_{V^{(t)}}$ is given by the following matrix:

$$\begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_1^{(t)} & h_2^{(t)} & \dots & h_n^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix},$$

where the vector $h^{(t)}$ appears in the ith row and n is the dimension of the hidden layer. Hence, the inner sum over i in (3.13) produces a matrix with ijth component equal to $(DL)_{o_i^{(t)}} h_j^{(t)}$. Observe such matrix may be expressed via the outer product $(DL)_{o^{(t)}} \otimes h^{(t)}$, so we may write

$$(DL)_V = \sum_t (DL)_{o^{(t)}} \otimes h^{(t)}. \tag{3.14}$$

3.6. **Hidden-to-hidden weight matrix.** The only node $W^{(t)}$ feeds into is $h^{(t+1)}$, so

$$\begin{aligned} (DL)_W &= \sum_{t=0}^{\tau-1} (DL)_{W^{(t)}} = \sum_{t=0}^{\tau-1} \sum_i (DL)_{h_i^{(t+1)}} (Dh_i^{(t+1)})_{a_i^{(t+1)}} (Da_i^{(t+1)})_{W^{(t)}} \\ &= \sum_{t=0}^{\tau-1} \sum_i (DL)_{h_i^{(t+1)}} \operatorname{diag}\left[1 - (h^{(t+1)})^2\right] (Da_i^{(t+1)})_{W^{(t)}} \\ &= \sum_{t=0}^{\tau-1} \sum_i ((DL)_{h_i^{(t+1)}} \left[1 - (h^{(t+1)})^2\right])^\top \odot (Da_i^{(t+1)})_{W^{(t)}}. \end{aligned} \tag{3.15}$$

As $a_i^{(t+1)} = \sum_j W_{ij} h_j^{(t)}$, we see that $(Da_i^{(t+1)})_{W^{(t)}}$ is the matrix

$$\begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_1^{(t)} & h_2^{(t)} & \dots & h_n^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix},$$

where the vector $h^{(t)}$ appears in the ith row. Hence, the inner sum over i in (3.15) produces the matrix whose ijth entry is $(DL)_{h_i^{(t+1)}} (1 - (h_i^{(t+1)})^2) h_j^{(t)}$. We observe that such matrix may be expressed via the outer product $\left[(DL)_{h^{(t+1)}} \odot (1 - h^{(t+1)})^2\right] \otimes$

$h^{(t)}$, and conclude that

$$(DL)_W = \sum_{t=0}^{\tau-1} \left[(DL)_{h^{(t+1)}} \odot (1 - (h^{(t+1)})^2) \right] \otimes h^{(t)}. \tag{3.16}$$

3.7. **Input-to-hidden weight matrix.** The only node $U^{(t)}$ feeds into is $h^{(t)}$ (Figure 1), so

$$(DL)_U = \sum_t (DL)_t = \sum_t \sum_i (DL)_{h_i^{(t)}} (Dh^{(t)})_{a_i^{(t)}} (Da_i^{(t)})_{U^{(t)}}. \tag{3.17}$$

By identical reasoning to Section 3.6, we see that

$$(DL)_U = \sum_t \left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2) \right] \otimes x^{(t)}. \tag{3.18}$$

3.8. **Embedding matrix.** The only node $E^{(t)}$ feeds into is $x^{(t)}$. There are two ways in which we may apply the chain rule. The first approach is to differentiate with respect to each component of $x^{(t)}$, but since each component of $x^{(t)}$ feeds into every component of $a^{(t)}$, this devolves into the triple sum shibboleth

$$(DL)_E = \sum_t \sum_i \sum_m (DL)_{a_m^{(t)}} (Da_m^{(t)})_{x_i^{(t)}} (Dx_i^{(t)})_{E^{(t)}}, \tag{3.19}$$

from which we will have to claw our way back out to a more efficient expression. Instead we will differentiate with respect to the vector $x^{(t)}$ as a whole, that is,

$$\begin{aligned} (DL)_E &= \sum_t (DL)_{h^{(t)}} (Dh^{(t)})_{a^{(t)}} (Da^{(t)})_{x^{(t)}} (Dx^{(t)})_{E^{(t)}} \\ &= \sum_t \left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2) \right] U (Dx^{(t)})_{E^{(t)}}. \end{aligned} \tag{3.20}$$

Observe that $(Dx^{(t)})_{E^{(t)}}$ is now a tensor, which we visualize by placing a vector v_{ij} of dimension equal to $x^{(t)}$ at each index ij of $E^{(t)}$, whose kth component is given by $(Dx_k^{(t)})_{E_{ij}^{(t)}}$. Denoting the dimension of $E^{(t)}$ by $V \times d\_model$, we see that $(Dx^{(t)})_{E^{(t)}}$ is a vertical stack of V many square matrices of dimension $d\_model \times d\_model$, all of which are the zero matrix except for the matrix placed at the row corresponding to $x^{(t)}$, which is the identity. Right multiplication of the row vector $\left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2) \right] U$ by the tensor $(Dx^{(t)})_{E^{(t)}}$ consists of matrix multiplication with each vertically stacked matrix of the tensor, resulting in a $V \times d\_model$ matrix, with all rows identically zero except for the row corresponding to $x^{(t)}$, which is equal to $\left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2) \right] U$. In words, we may summarize (3.20) in words as $(DL)_E$ is the $V \times d\_model$ matrix with row i equal to $\left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2) \right] U$ if i corresponds to the row of some $x^{(t)}$, $1 \leq t \leq \tau$, and equal to the zero vector otherwise. Mathematically, we may write

$$(DL)_E = \sum_t \mathbf{e}_{i_t}^\top \left(\left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2) \right] U \right), \tag{3.21}$$

where $\mathbf{e}_{i_t}^\top$ is the one-hot vector with a one at position $i^{(t)}$, where $i^{(t)}$ is the index corresponding to $x^{(t)}$.

Remark 3.22. If the reader is uncomfortable with representing the Jacobian as a tensor and then applying the chain rule, try the following. The $V \times d\_model$ matrix $E^{(t)}$ may be viewed as a vector $e^{(t)}$ in $\mathbb{R}^{d\_model \times V}$ by arranging the rows of the matrix side by side. The Jacobian of the projection map taking $e^{(t)}$ to $x^{(t)}$ is then a $d\_model \times (d\_model \times V)$ matrix. One can then proceed with standard matrix multiplication of $\left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2)\right] U$ and the Jacobian to obtain a $1 \times (d\_model \times V)$ composite Jacobian. Consistent with the unfolding of the matrix $E^{(t)}$ to the vector $e^{(t)}$ by placing the rows side by side, we fold the $1 \times (d\_model \times V)$ Jacobian back into the shape of $E^{(t)}$ by vertically stacking the V many $1 \times d\_model$ vectors placed side by side, obtaining $(DL)_{E^{(t)}}$. Observe how in Section 3.8, the manner in which the tensor $(Dx^{(t)})_{E^{(t)}}$ is defined as well as how it right multiplies the row vector $\left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2)\right] U$ produces the exact same result.

4. Summary

We may summarize the backward pass in the following nine equations:

$$(DL)_{o_i^{(t)}} = \frac{1}{\tau}(\hat{y}_i^{(t)} - \mathbf{1}_{\{i=y^{(t)}\}}). \tag{4.1}$$

$$(DL)_{h^{(\tau)}} = (DL)_{o^{(\tau)}}(Do^{(\tau)})_{h^{(\tau)}} = (DL)_{o^{(\tau)}} V. \tag{4.2}$$

For $t = \tau - 1, \ldots, 1$,

$$(DL)_{h^{(t)}} = (DL)_{h^{(t+1)}} \left(\left[1 - (h^{(t+1)})^2\right] \odot W \right) + (DL)_{o^{(t)}} V. \tag{4.3}$$

$$(DL)_c = \sum_t (DL)_{o^{(t)}}. \tag{4.4}$$

$$(DL)_b = \sum_t (DL)_{h^{(t)}} \operatorname{diag}\left[1 - (h^{(t)})^2\right]. \tag{4.5}$$

$$(DL)_V = \sum_t (DL)_{o^{(t)}} \otimes h^{(t)}. \tag{4.6}$$

$$(DL)_W = \sum_{t=0}^{\tau-1} \left[(DL)_{h^{(t+1)}} \odot (1 - (h^{(t+1)})^2)\right] \otimes h^{(t)}. \tag{4.7}$$

$$(DL)_U = \sum_t \left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2)\right] \otimes x^{(t)}. \tag{4.8}$$

$$(DL)_E = \sum_t \mathbf{e}_{i_t}^\top \left(\left[(DL)_{h^{(t)}} \odot (1 - (h^{(t)})^2)\right] U \right). \tag{4.9}$$

References

[1] Goodfellow et al. Deep Learning. 2016. http://www.deeplearningbook.org