01_Long_Short_Term_Memory_Gated_Recurrent_Unit

March 23, 2024

1 Vanishing/exploding gradients

- In order to study the vanishing/exploding gradients problem, consider a multilayer perceptron with $L > 2$ layers and a single unit per layer.
- More concretely, let $a_0 \in \mathbb{R}$ denote the input to the multilayer perceptron and let $a_L \in \mathbb{R}$ denote the corresponding output.
- For every $l \in \{1, \dots, L\}$, let a_l be given by

$$a_l = \sigma(w_l a_{l-1} + b_l),$$

where σ is an activation function, $w_l \in \mathbb{R}$ is the weight for layer l, and $b_l \in \mathbb{R}$ is the bias for layer l.

- For convenience, for every $l \in \{1, \dots, L\}$, let z_l be given by

$$z_l = w_l a_{l-1} + b_l,$$

so that

$$a_l = \sigma(z_l).$$

- We will compute the partial derivative $\partial a_L / \partial w_1$, which intuitively represents the impact of an infinitesimally small increase in w_1 on the output of the network a_L for a given input a_0.
- First, note that w_1 only affects a_L through a_1. By the chain rule,

$$\frac{\partial a_L}{\partial w_1} = \frac{\partial a_L}{\partial a_1} \frac{\partial a_1}{\partial w_1}.$$

- Second, note that w_1 only affects a_1 through z_1. By the chain rule,

$$\frac{\partial a_1}{\partial w_1} = \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_1} = \sigma'(z_1) a_0,$$

where σ' is the derivate of the activation function σ.

- Third, for every $l \in \{1, \dots, L-1\}$, note that a_l only affects a_L through a_{l+1}. By the chain rule,

$$\frac{\partial a_L}{\partial a_l} = \frac{\partial a_L}{\partial a_{l+1}} \frac{\partial a_{l+1}}{\partial a_l}.$$

- Inductively, the previous result implies that

$$\frac{\partial a_L}{\partial a_1} = \frac{\partial a_L}{\partial a_{L-1}} \frac{\partial a_{L-1}}{\partial a_{L-2}} \cdots \frac{\partial a_3}{\partial a_2} \frac{\partial a_2}{\partial a_1} = \prod_{l=2}^{L} \frac{\partial a_l}{\partial a_{l-1}}.$$

- Fourth, note that a_{l-1} only affects a_l through z_l for every $l \in \{1, \dots, L\}$. By the chain rule,

$$\frac{\partial a_l}{\partial a_{l-1}} = \frac{\partial a_l}{\partial z_l} \frac{\partial z_l}{\partial a_{l-1}} = \sigma'(z_l) w_l.$$

- Finally, by combining the previous results,

$$\frac{\partial a_L}{\partial w_1} = \left[\prod_{l=2}^{L} \sigma'(z_l) w_l \right] \sigma'(z_1) a_0.$$

- The expression above is a product of L terms, each of which is a product of two terms.
- If $L = 16$, consider that $(1/2)^L \approx 0.000015$ and $2^L \approx 65536$.
- There are many situations where $\partial a_L / \partial w_1$ may vanish (become close to zero). For example,
  - If $\sigma = \text{ReLU}$, then $|\sigma'(z_l) w_l| = 0$ whenever $z_l < 0$ and $w_l \in \mathbb{R}$.
  - If $\sigma = \text{sigmoid}$, then $|\sigma'(z_l) w_l| \approx 0$ whenever $|z_l| \gg 0$ and $|w_l| \ll 1/|\sigma'(z_l)|$.
- There are may situations where $\partial a_L / \partial w_1$ may explode (become very large in magnitude). For example,
  - If $\sigma = \text{ReLU}$, then $|\sigma'(z_l) w_l| \gg 1$ whenever $z_l > 0$ and $|w_l| \gg 1$.
  - If $\sigma = \text{sigmoid}$, then $|\sigma'(z_l) w_l| \gg 1$ whenever $z_l \in \mathbb{R}$ and $|w_l| \gg 1/|\sigma'(z_l)|$.
- Because the weight w_1 only affects the loss through the output of the network a_L for a given input a_0, the magnitude of the partial derivative $\partial a_L / \partial w_1$ is related to how much the weight w_1 is updated in a given iteration of gradient descent.
- A vanishing partial derivative may lead to slow progress, whereas an exploding partial derivative may lead to unstable progress.
- The previous analysis can be generalized to show that any parameter in a multilayer perceptron can have issues with vanishing/exploding partial derivatives, specially when the number of layers L is large.
- Unfolding a recurrent neural network to compute a prediction for a sequence of T input vectors is roughly related to employing a multilayer perceptron with at least T layers.
- Therefore, recurrent neural networks trained to make predictions for long input sequences are highly susceptible to vanishing/exploding gradients problems.
- Based on a careful mathematical analysis of the gradients (of the loss with respect to the parameters) of recurrent neural networks, long short-term memory networks (LSTMs) were developed to mitigate the vanishing gradients problem.

- Long short-term memory networks are one of the most influential neural network architectures of all time.

2 Sequence Element Classification

- We will study how long short-term memory networks can be used for sequence element classification.
- A sequence element classification dataset is a sequence of examples. Each example is a pair of sequences of vectors. Both sequences in an example have the same number of elements.
- For instance, if the first element of an example is a sequence of vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$ with T elements, where $\mathbf{x}_i \in \mathbb{R}^d$, then the second element of the same example may be a sequence of (one-hot encoded) vectors $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T$, where $\mathbf{y}_i \in \mathbb{R}^q$.
- Assuming the previous example is part of the training dataset, a sequence element classification model would attempt to predict the target vector $\mathbf{y}_t \in \mathbb{R}^q$ given the sequence of input vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t$, for every t. In this context, t is called the number of *time steps*.
- As always, the objective is to find a a model that generalizes well (makes good predictions for unseen input sequences).

3 Long Short-Term Memory Networks: Overview

- Long short-term memory networks are able to make predictions based on an entire sequence of input vectors rather than a single vector.
- A long short-term memory network summarizes a given sequence $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t$ using a hidden state $\mathbf{h}_t \in \mathbb{R}^h$ and a cell state $\mathbf{c}_t \in \mathbb{R}^h$, where h is a hyperparameter.
- More concretely, the initial states are typically given by $\mathbf{h}_0 = \mathbf{0}$ and $\mathbf{c}_0 = \mathbf{0}$ and, for every $t > 0$, the hiden state $\mathbf{h}_t$ and the cell state $\mathbf{c}_t$ are obtained by using a (learned) function f to compute

$$\mathbf{h}_t, \mathbf{c}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}).$$

- After computing the hidden state $\mathbf{h}_t$ based on the input sequence $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t$, one or more fully connected layers can be used to compute the logits vector $\mathbf{o}_t$ and the corresponding prediction $\hat{\mathbf{y}}_t = \mathrm{softmax}(\mathbf{o}_t)$.

4 Long Short-Term Memory Networks

- We will present the vectorized implementation of long short-term memory networks, which enables minibatch stochastic gradient descent.
- Suppose that a batch of n examples from the (sequence element classification) dataset is organized into a sequence of input matrices $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_T$ and a sequence of target matrices $\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_T$, where $\mathbf{X}_i \in \mathbb{R}^{n \times d}$ and $\mathbf{Y}_i \in \mathbb{R}^{n \times q}$.

- More concretely, the input matrices $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_T$ contain a single sequence of (transposed) input vectors in a given row, and the target matrices $\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_T$ contain the corresponding sequence of (transposed) target vectors in the same row.
- Note that the input matrices could be further organized into a single $T \times n \times d$ tensor $[\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_T]$. Similarly, the target matrices could be organized into a single $T \times n \times q$ tensor $[\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_T]$.
- Let the initial hidden state matrix $\mathbf{H}_0 \in \mathbb{R}^{n \times h}$ and the the initial cell state matrix $\mathbf{C}_0 \in \mathbb{R}^{n \times h}$ be matrices filled with zeros, where h is a hyperparameter.
- A long short-term memory network layer has three so-called gates: input gate, forget gate, and output gate.
- For any time step $t > 0$, the input gate matrix $\mathbf{I}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{I}_t = \text{sigmoid}\left(\mathbf{X}_t \mathbf{W}^{(II)} + \mathbf{H}_{t-1} \mathbf{W}^{(RI)} + \mathbf{B}^{(I)}\right),$$

where $\mathbf{W}^{(II)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(RI)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B}^{(I)} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the parameter matrix $\mathbf{B}^{(I)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(I)} \in \mathbb{R}^h$ across n rows.

- For any time step $t > 0$, the forget gate matrix $\mathbf{F}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{F}_t = \text{sigmoid}\left(\mathbf{X}_t \mathbf{W}^{(IF)} + \mathbf{H}_{t-1} \mathbf{W}^{(RF)} + \mathbf{B}^{(F)}\right),$$

where $\mathbf{W}^{(IF)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(RF)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B}^{(F)} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the parameter matrix $\mathbf{B}^{(F)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(F)} \in \mathbb{R}^h$ across n rows.

- For any time step $t > 0$, the output gate matrix $\mathbf{O}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{O}_t = \text{sigmoid}\left(\mathbf{X}_t \mathbf{W}^{(IO)} + \mathbf{H}_{t-1} \mathbf{W}^{(RO)} + \mathbf{B}^{(O)}\right),$$

where $\mathbf{W}^{(IO)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(RO)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B}^{(O)} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the parameter matrix $\mathbf{B}^{(O)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(O)} \in \mathbb{R}^h$ across n rows.

- For any time step $t > 0$, the **candidate** cell state matrix $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n \times h}$ is given by

$$\tilde{\mathbf{C}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}^{(IC)} + \mathbf{H}_{t-1} \mathbf{W}^{(RC)} + \mathbf{B}^{(C)}\right),$$

where $\mathbf{W}^{(IC)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(RC)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B}^{(C)} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the parameter matrix $\mathbf{B}^{(C)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(C)} \in \mathbb{R}^h$ across n rows.

- For any given time step $t > 0$, the cell state matrix $\mathbf{C}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t,$$

where $\odot$ denotes elementwise multiplication.

- Intuitively, the forget gate decides whether the previous cell state should be kept, and the input gate controls whether the current candidate cell state should be added to the cell state.
- Finally, for a given time step $t > 0$, the hidden state matrix $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t),$$

where $\odot$ denotes elementwise multiplication.

- Intuitively, the output gate decides whether the current cell state should become the current output of the layer.
- The following image illustrates a long short-term memory layer. The so-called input node corresponds to what we called the candidate cell state.
- In summary, a long short-term memory layer can be interpreted as a differentiable electronic circuit that can learn when to store, erase, update, and output a state stored in a memory cell.
- For the sake of simplicity, suppose that this so-called long short-term memory network layer is followed by a single fully connected layer. In that case, for every $t > 0$, the logits matrix $\mathbf{O}_t$ is given by

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}^{(2)} + \mathbf{B}^{(2)},$$

where $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ and $\mathbf{B}^{(2)} \in \mathbb{R}^{n \times q}$ are parameter matrices, and the matrix $\mathbf{B}^{(2)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(2)} \in \mathbb{R}^{q}$ across n rows.

- For every $t > 0$, the prediction matrix $\hat{\mathbf{Y}}_t$ is given by

$$\hat{\mathbf{Y}}_t = \text{softmax}(\mathbf{O}_t),$$

where the softmax function is applied individually to each **row** of the logits matrix $\mathbf{O}_t$.

- Let $l(\hat{\mathbf{Y}}, \mathbf{Y})$ denote the cross-entropy loss between a prediction matrix $\hat{\mathbf{Y}} \in \mathbb{R}^{n \times q}$ and a target matrix $\mathbf{Y} \in \mathbb{R}^{n \times q}$.
- A long short-term memory network can be trained by minimizing the average loss across time steps given by

$$\frac{1}{T} \sum_{t=1}^{T} l(\hat{\mathbf{Y}}_t, \mathbf{Y}_t),$$

where each prediction matrix $\hat{\mathbf{Y}}_t$ depends on the parameters of the network and the sequence of input matrices $\mathbf{X}_1, \ldots, \mathbf{X}_t$.

5 Gated Recurrent Unit

- A gated recurrent unit (GRU) network is a simplified alternative to the (much older) long short-term memory network.
- We will present the vectorized implementation of gated recurrent unit networks, which enables minibatch stochastic gradient descent.
- Suppose that a batch of n examples from the (sequence element classification) dataset is organized into a sequence of input matrices $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_T$ and a sequence of target matrices $\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_T$, where $\mathbf{X}_i \in \mathbb{R}^{n \times d}$ and $\mathbf{Y}_i \in \mathbb{R}^{n \times q}$.
- Let the initial hidden state matrix $\mathbf{H}_0 \in \mathbb{R}^{n \times h}$ be a matrix filled with zeros, where h is a hyperparameter.
- A gated recurrent unit layer has two so-called gates: reset gate and update gate.
- For any time step $t > 0$, the reset gate matrix $\mathbf{R}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{R}_t = \text{sigmoid}\left(\mathbf{X}_t \mathbf{W}^{(IR)} + \mathbf{H}_{t-1} \mathbf{W}^{(RR)} + \mathbf{B}^{(R)}\right),$$

where $\mathbf{W}^{(IR)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(RR)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B}^{(R)} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the parameter matrix $\mathbf{B}^{(R)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(R)} \in \mathbb{R}^h$ across n rows.

- For any time step \$ t > 0\$, the update gate matrix $\mathbf{Z}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{Z}_t = \text{sigmoid}\left(\mathbf{X}_t \mathbf{W}^{(IZ)} + \mathbf{H}_{t-1} \mathbf{W}^{(RZ)} + \mathbf{B}^{(Z)}\right),$$

where $\mathbf{W}^{(IZ)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(RZ)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B}^{(Z)} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the parameter matrix $\mathbf{B}^{(Z)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(Z)} \in \mathbb{R}^h$ across n rows.

- For any time step $t > 0$, the **candidate** hidden state matrix $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ is given by

$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}^{(IH)} + \left(\mathbf{R}_t \odot \mathbf{H}_{t-1}\right) \mathbf{W}^{(RH)} + \mathbf{B}^{(H)}\right),$$

where $\odot$ denotes elementwise multiplication, $\mathbf{W}^{(IH)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(RH)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B}^{(H)} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the parameter matrix $\mathbf{B}^{(H)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(H)} \in \mathbb{R}^h$ across n rows.

- Intuitively, the reset gate decides whether the previous hidden state will be used to compute the current candidate hidden state.
- Finally, for a given time step $t > 0$, the hidden state matrix $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ is given by

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t,$$

where $\odot$ denotes elementwise multiplication.

- Intuitively, the update gate decides how much of the current hidden state comes from the previous hidden state and how much comes from the current candidate hidden state.
- The following image illustrates a gated recurrent unit layer.
- For the sake of simplicity, suppose that this so-called gated recurrent unit layer is followed by a single fully connected layer. In that case, for every $t > 0$, the logits matrix $\mathbf{O}_t$ is given by

$$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}^{(2)} + \mathbf{B}^{(2)},$$

where $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ and $\mathbf{B}^{(2)} \in \mathbb{R}^{n \times q}$ are parameter matrices, and the matrix $\mathbf{B}^{(2)}$ is obtained by transposing and replicating the same parameter vector $\mathbf{b}^{(2)} \in \mathbb{R}^{q}$ across n rows.

- For every $t > 0$, the prediction matrix $\hat{\mathbf{Y}}_t$ is given by

$$\hat{\mathbf{Y}}_t = \text{softmax}(\mathbf{O}_t),$$

where the softmax function is applied individually to each **row** of the logits matrix $\mathbf{O}_t$.

- Let $l(\hat{\mathbf{Y}}, \mathbf{Y})$ denote the cross-entropy loss between a prediction matrix $\hat{\mathbf{Y}} \in \mathbb{R}^{n \times q}$ and a target matrix $\mathbf{Y} \in \mathbb{R}^{n \times q}$.
- A gated recurrent unit network can be trained by minimizing the average loss across time steps given by

$$\frac{1}{T} \sum_{t=1}^{T} l(\hat{\mathbf{Y}}_t, \mathbf{Y}_t),$$

where each prediction matrix $\hat{\mathbf{Y}}_t$ depends on the parameters of the network and the sequence of input matrices $\mathbf{X}_1, \ldots, \mathbf{X}_t$.

6 Recommended reading

- Neural networks and deep learning: Chapter 5 (The vanishing gradient problem).
- Dive into Deep Learning: Chapters 10.1 and 10.2.

7 [Storing this notebook as a pdf]

- In order to store this notebook as a pdf, you will need to hide the images included in the previous cells using the following syntax:
  - `<!--- ![Image caption.](https://link.to.image) --->`

[ ]:
```
%%capture
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic
```

```
# Set the path to this notebook below (add \ before spaces). The output `pdf`␣
↪will be stored in the corresponding folder.
!jupyter nbconvert --to pdf /content/gdrive/My\ Drive/Colab\ Notebooks/nndl/
↪week_12/lecture/01_Long_Short_Term_Memory_Gated_Recurrent_Unit.ipynb

# If having issues, save this notebook (File > Save) and restart the session␣
↪(Runtime > Restart session) before running this cell. To debug, remove the␣
↪first line (`%%capture`).
```