01_Recurrent_Neural_Networks

March 1, 2024

1 Sequence Element Classification

- We will study how recurrent neural networks can be used for sequence element classification, but the ideas involved will generalize to other types of sequence tasks.
- 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).

2 Recurrent Neural Networks: Overview

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

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

- After computing the hidden vector $\mathbf{h}_t$ 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 $\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{o}_t)$.

3 Recurrent Neural Networks

- We will present the vectorized implementation of recurrent neural 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 $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T$ and a sequence of target matrices $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_T$, 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 $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T$ contain a single sequence of (transposed) input vectors in a given row, and the target matrices $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_T$ 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 $[\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T]$. Similarly, the target matrices could be organized into a single $T \times n \times q$ tensor $[\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_T]$.
- 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.
- For any time step $t > 0$, suppose that the hidden state matrix $\mathbf{H}_t$ is given by

$$\mathbf{H}_t = \sigma(\mathbf{X}_t \mathbf{W}^{(I)} + \mathbf{H}_{t-1} \mathbf{W}^{(R)} + \mathbf{B}),$$

where σ is an activation function (typically, $\sigma = \tanh$), $\mathbf{W}^{(I)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}^{(R)} \in \mathbb{R}^{h \times h}$, and $\mathbf{B} \in \mathbb{R}^{n \times h}$ are parameter matrices, and the matrix $\mathbf{B}$ is obtained by transposing and replicating the same parameter vector $\mathbb{b} \in \mathbb{R}^h$ across n rows.

- For the sake of simplicity, suppose that this so-called recurrent 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 recurrent neural 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 $\mathbf{X}_1, \ldots, \mathbf{X}_t$.

4 Recurrent Neural Networks: Unfolding

- Let $[\mathbf{A} : \mathbf{B}]$ denote the matrix that results from concatenating the matrix $\mathbf{A}$ with the matrix $\mathbf{B}$ across columns, and $[\mathbf{A}; \mathbf{B}]$ denote the matrix that results from concatenating the matrix $\mathbf{A}$ with the matrix $\mathbf{B}$ across rows.
- For every $t > 0$, the hidden state matrix $\mathbf{H}_t$ is also given by

$$\mathbf{H}_t = \sigma([\mathbf{X}_t : \mathbf{H}_{t-1}][\mathbf{W}^{(I)}; \mathbf{W}^{(R)}] + \mathbf{B}).$$

- In words, a *recurrent layer* simply concatenates the current input $\mathbf{x}_t$ vector with the previous hidden state $\mathbf{h}_{t-1}$ before employing a fully connected layer parameterized by $[\mathbf{W}^{(I)}; \mathbf{W}^{(R)}]$ and $\mathbf{B}$ in order to compute the current hidden state $\mathbf{h}_t$, from which the current prediction vector $\hat{\mathbf{y}}_t$ is obtained through (one or more) fully connected layers.
- The following image illustrates a recurrent neural network *unfolded* for three time steps.
- Note how the same parameters are used at every time step t, so that the number of parameters is independent of the length of the input sequence.
- Note how a hidden state $\mathbf{h}_t$ is forced to summarize information about a sequence of input vectors $\mathbf{x}_1, \dots, \mathbf{x}_t$ in order to allow predicting the target vector $\mathbf{y}_t$.

5 Recurrent neural language model

- Recall that a language model assigns a probability to each possible sequence of tokens (text).
- By the chain rule of probability, for every sequence of tokens $x_1, \dots, x_T$, a language model only needs to assign a probability $\mathbb{P}(X_1 = x_1)$ to x_1 being the first token and, for every $t > 1$, a probability $\mathbb{P}(X_t = x_t \mid X_1 = x_1, \dots, X_{t-1} = x_{t-1})$ to x_t being the t-th token if the previous tokens are $x_1, \dots, x_{t-1}$.
- An autoregressive language model can be used to generate text by sampling the first token x_1 from the distribution for X_1 and, for every $t > 1$, sampling the token x_t from the distribution for X_t given $X_1 = x_1, \dots, X_{t-1} = x_{t-1}$.
- A recurrent neural network trained to predict the next (one-hot encoded) token $\mathbf{y}_t = \mathbf{x}_{t+1}$ given a sequence of (one-hot encoded) tokens $\mathbf{x}_1, \dots, \mathbf{x}_t$ can be used as an autoregressive language model, since its prediction vector $\hat{\mathbf{y}}_t$ can be interpreted as a probability distribution over the next token given the previous tokens.
- Large language models employ the same principle.

6 Recommended reading

- Dive into Deep Learning: Chapters 9.4, 9.5, and 9.6.

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_10/lecture/01_Recurrent_Neural_Networks.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`).
```