Tittel

Forfatter

1 Introduction

2 Background

3 Method

The purpose of this project, as mentioned earlier, is to understand the math and implementations underlying a recurrent neural network (RNN). For the implementation of the RNN, no traditional machine learning (ML) libraries has been utilised. The RNN model and its surroundings are implemented using Python, where numpy **NUMPY** is the library taking care of the mathematical implementations such as vector and matrix multiplication. The RNN is implemented in its most basic form, often denoted as a *vanilla RNN*. Nevertheless, the implementation handles a big variety of data types, and allows for configurability of optimisation algorithms and activation functions. The focus of the implementation has been natural language data, but the model can also be used for other time series data, such as learning to predict interest rates, or mimic and generate oscillations like sine waves. None of the implementation exploits graphical processing units (GPUs), and is therefore slower than more advanced implmentations where the power of GPUs is taken advantage of.

3.1 Python implementation

Model architecture

The RNN is built as a class, where all nodes and layers are built simply by initialising matrices of dimensions corresponding to the layers. Optimiser, loss function and activation function(s) are initialised at the time of model initialisation. Training and inference is done using these functions. The parameters of the model are always 5 variables: Weights from input to hidden layer (also called the recurrent layer), denoted `w_xh` in code and `U` in equations; weights from hidden to hidden, denoted `w_hh` in code and `W` in equations; and weights from hidden to output layer, denoted `w_hy` in code and `V` in equations. The bias on the hidden layer is denoted `b_hh` in code and `b` in equations and the bias on the output layer is denoted `b_hy` in code and `c` in equations. The weights and biases mentioned here are the only parameters learned through experience. Non-learned parameters, called hyper-parameters, are explained below.

One forward pass of a data sample can be explained in a few equations:

$$\mathbf{U} = \text{weights input} \rightarrow \text{hidden} \tag{1}$$
$$\mathbf{W} = \text{weights hidden} \rightarrow \text{hidden} \tag{2}$$
$$\mathbf{V} = \text{weights hidden} \rightarrow \text{output} \tag{3}$$
$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(\mathbf{t-1})} + \mathbf{U}\mathbf{x}^{(\mathbf{t})} \tag{4}$$
$$\mathbf{h}^{(t)} = activation(\mathbf{a}^{(t)}) \tag{5}$$
$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \tag{6}$$
$$\hat{\mathbf{y}}^{(t)} = postprocessing(\mathbf{o}^{(t)}) \tag{7}$$

As can be seen, the output $o^{(t)}$ (at time t), is dependent on the previous data sample.

Random control

All experiments using the model is seeded, to ensure reproducable results. Initialisation values of all parameters (not hyperparameters) are drawn from the unit normal distribution check this , to a value between 0 and 1, multiplied by a scale specified by the user.

4 Results

Appendix - some math notes

Backpropagation through time - BPTT

Definitions:

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(\mathbf{t-1})} + \mathbf{U}\mathbf{x}^{(\mathbf{t})} \tag{8}$$
$$\mathbf{h}^{(t)} = activation(\mathbf{a}^{(t)}) \tag{9}$$
$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \tag{10}$$
$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{o}^{(t)}) \tag{11}$$
$$\mathbf{U} = \text{input weights} \tag{12}$$
$$\mathbf{W} = \text{hidden weights} \tag{13}$$
$$\mathbf{V} = \text{output weights} \tag{14}$$

The goal is to maximize the probability of the observed data by estimating parameters (here: $\mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{b}, \mathbf{c}$). The estimated parameters yielding the highest maximum likelihood are called the maximum likelihood estimates. These parameters can be estimated by minimizing the cross-entropy between the model distribution and the data. This cross-entropy function, which is a function of inputs ($\mathbf{x}$) and outputs ($\mathbf{y}$), can be seen in (8).

$$C\left(\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(\tau)}\}\right) \tag{15}$$
$$= \sum_t C^{(t)} \tag{16}$$
$$= -\sum_t log\ p\left(y^{(t)} | \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(t)}\}\right) \tag{17}$$
$$\tag{18}$$

Note: $y^{(t)}$ is an entry in the output vector $\hat{\mathbf{y}}^{(t)}$

The cost function is the negative log-likelihood function. Minimising this function is the same as maximum the likelihood of the parameters - not due to the log, but due to the negative prefix. The log in log-likelihood works, but I dont know why it is used?

Below we derive the gradients of the nodes in the computational graph from the deep learning book. These gradients must propagate backwards through time, from time $t = \tau$ down to $t = 0$.

The gradient of the cost function at the output, $\mathbf{o}$, at time t is

$$\nabla_{\mathbf{o}^{(t)}} C = \frac{\delta C}{\delta o^{(t)}} = \frac{\delta C}{\delta C^{(t)}} \frac{\delta C^{(t)}}{\delta o^{(t)}} = \hat{y}^{(t)} - \mathbf{1}_{i=y^{(t)}} \tag{19}$$

We have found a general expression for the gradien at the $\mathbf{o}^{(t)}$-nodes. The next step is to find an expression for the gradient of the final hidden (computational) node, at time τ: $\mathbf{h}^{(\tau)}$. Its only descendant is $\mathbf{o}^{(\tau)}$, which means its gradient is solely dependent on this node, which makes it a good starting point for the later gradient calculations.

$$\nabla_{\mathbf{h}^{(\tau)}} C = (\nabla_{\mathbf{o}^{(\tau)}} C) \frac{\delta \mathbf{o}^{(\tau)}}{\delta \mathbf{h}^{(\tau)}} \tag{20}$$

$$= (\nabla_{\mathbf{o}^{(\tau)}} C) \mathbf{V} \tag{21}$$

$$\nabla_{\mathbf{h}^{(\tau)}} C = \mathbf{V}^\top (\nabla_{\mathbf{o}^{(\tau)}} C) \tag{22}$$

Where all the right hand side terms are known from before.

The only nodes that need gradient computation now, are all the hidden states preceding the last. I.e., for $\mathbf{h}^{(t)}$, where $t = \{0, ..., \tau - 1\}$. For these time steps, the gradient is influenced by both the gradient at $\mathbf{o}^{(t)}$, as well as all the preceding hidden state gradients. Remember that the preceding hidden state of $\mathbf{h}^{(t)}$ is $\mathbf{h}^{(t+1)}$, which has preceding hidden state $\mathbf{h}^{(t+2)}$, and so on. We are calculating the gradient starting from $t = \tau - 1$, working our way down to $t = 0$:

$$\nabla_{\mathbf{h}^{(\tau-1)}} C = \nabla_{\mathbf{o}^{(\tau-1)}} C \frac{\delta \mathbf{o}^{(\tau-1)}}{\delta \mathbf{h}^{(\tau-1)}} + \nabla_{\mathbf{h}^{(\tau)}} C \frac{\delta \mathbf{h}^{(\tau)}}{\delta \mathbf{h}^{(\tau-1)}} \tag{23}$$

$$\nabla_{\mathbf{h}^{(\tau-2)}} C = \nabla_{\mathbf{o}^{(\tau-2)}} C \frac{\delta \mathbf{o}^{(\tau-2)}}{\delta \mathbf{h}^{(\tau-2)}} + \nabla_{\mathbf{h}^{(\tau-1)}} C \frac{\delta \mathbf{h}^{(\tau-1)}}{\delta \mathbf{h}^{(\tau-2)}} \tag{24}$$

$$\nabla_{\mathbf{h}^{(\tau-3)}} C = \nabla_{\mathbf{o}^{(\tau-3)}} C \frac{\delta \mathbf{o}^{(\tau-3)}}{\delta \mathbf{h}^{(\tau-3)}} + \nabla_{\mathbf{h}^{(\tau-2)}} C \frac{\delta \mathbf{h}^{(\tau-2)}}{\delta \mathbf{h}^{(\tau-3)}} \tag{25}$$

$$= \nabla_{\mathbf{o}^{(\tau-3)}} C \frac{\delta \mathbf{o}^{(\tau-3)}}{\delta \mathbf{h}^{(\tau-3)}} + \left(\nabla_{\mathbf{o}^{(\tau-2)}} C \frac{\delta \mathbf{o}^{(\tau-2)}}{\delta \mathbf{h}^{(\tau-2)}} + \nabla_{\mathbf{h}^{(\tau-1)}} C \frac{\delta \mathbf{h}^{(\tau-1)}}{\delta \mathbf{h}^{(\tau-2)}} \right) \frac{\delta \mathbf{h}^{(\tau-2)}}{\delta \mathbf{h}^{(\tau-3)}} \tag{26}$$

$$= \nabla_{\mathbf{o}^{(\tau-3)}} C \frac{\delta \mathbf{o}^{(\tau-3)}}{\delta \mathbf{h}^{(\tau-3)}} + \left(\nabla_{\mathbf{o}^{(\tau-2)}} C \frac{\delta \mathbf{o}^{(\tau-2)}}{\delta \mathbf{h}^{(\tau-2)}} + \left(\nabla_{\mathbf{o}^{(\tau-1)}} C \frac{\delta \mathbf{o}^{(\tau-1)}}{\delta \mathbf{h}^{(\tau-1)}} + \nabla_{\mathbf{h}^{(\tau)}} C \frac{\delta \mathbf{h}^{(\tau)}}{\delta \mathbf{h}^{(\tau-1)}} \right) \frac{\delta \mathbf{h}^{(\tau-1)}}{\delta \mathbf{h}^{(\tau-2)}} \right) \frac{\delta \mathbf{h}^{(\tau-2)}}{\delta \mathbf{h}^{(\tau-3)}} \tag{27}$$

Generally:

$$\nabla_{\mathbf{h}^{(t)}} C = \nabla_{\mathbf{o}^{(t)}} C \frac{\delta \mathbf{o}^{(t)}}{\delta \mathbf{h}^{(t)}} + \nabla_{\mathbf{h}^{(t+1)}} C \frac{\delta \mathbf{h}^{(t+1)}}{\delta \mathbf{h}^{(t)}} \tag{28}$$

Some parts of the equations above have been colored to emphasize the parts that we take with us from one gradient calculation to the next.

It is observable that the gradient at time t is indeed dependent on all later time steps $t+1, t+2, t+n$, as well as the current output. The influence from current output can be seen before the first (+), while the influence from previous states can be observed after the first (+).

The final step is to calculate the gradient of the parameter nodes $\mathbf{U}, \mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}$. To find these gradients, we must differentiate $\mathbf{h}^{(t)}$. Calculating the derivative of $\mathbf{h}^{(t)}$ involves differentiating the activation function using the chain rule. In the equations below, the chain rule is applied to differentiate the activations with respect to different parameters, but the activation function itself is not differentiated because it depends on which activation function in use. It is instead denoted as $\nabla_{activation}$.

$$
\begin{aligned}
\nabla_{\mathbf{c}} C &= \sum_t \left(\nabla_{\mathbf{o}^{(t)}} C\right) \frac{\delta \mathbf{o}^{(t)}}{\delta \mathbf{c}^{(t)}} &&= \sum_t \left(\nabla_{\mathbf{o}^{(t)}} C\right) \cdot 1 \\
\nabla_{\mathbf{V}} C &= \sum_t \left(\nabla_{\mathbf{o}^{(t)}} C\right) \frac{\delta \mathbf{o}^{(t)}}{\delta \mathbf{h}^{(t)}} &&= \sum_t \left(\nabla_{\mathbf{o}^{(t)}} C\right) \mathbf{h^{(t)}} \\
\nabla_{\mathbf{b}} C &= \sum_t \left(\nabla_{\mathbf{h}^{(t)}} C\right) \frac{\delta \mathbf{h}^{(t)}}{\delta \mathbf{b}^{(t)}} &&= \sum_t \left(\nabla_{\mathbf{h}^{(t)}} C\right) \left(\nabla_{activation}\right) \cdot 1 \\
\nabla_{\mathbf{W}} C &= \sum_t \left(\nabla_{\mathbf{h}^{(t)}} C\right) \frac{\delta \mathbf{h}^{(t)}}{\delta \mathbf{W}^{(t)}} &&= \sum_t \left(\nabla_{\mathbf{h}^{(t)}} C\right) \left(\nabla_{activation}\right) \cdot \mathbf{h}^{(t-1)} \\
\nabla_{\mathbf{U}} C &= \sum_t \left(\nabla_{\mathbf{h}^{(t)}} C\right) \frac{\delta \mathbf{h}^{(t)}}{\delta \mathbf{U}^{(t)}} &&= \sum_t \left(\nabla_{\mathbf{h}^{(t)}} C\right) \left(\nabla_{activation}\right) \cdot \mathbf{x}^{(t-1)}
\end{aligned}
$$

The code written for BPTT is tricky to read. Below is conversions from math to code for gradient calculation of eq. (21):

$$
\begin{aligned}
\nabla_{\mathbf{o}^{(t)}} C &= \hat{y}^{(t)} - \mathbf{1}_{i=y^{(t)}} && \Rightarrow \texttt{w\_hy} \\
\frac{\delta \mathbf{o}^{(t)}}{\delta \mathbf{h}^{(t)}} &= \mathbf{V} && \Rightarrow \texttt{grad\_o\_Cost} \\
\nabla_{\mathbf{h}^{(t+1)}} C &= \nabla_{\mathbf{o}^{(t+1)}} C \frac{\delta \mathbf{o}^{(t+1)}}{\delta \mathbf{h}^{(t+1)}} + \nabla_{\mathbf{h}^{(\tau)}} C \frac{\delta \mathbf{h}^{(t+2)}}{\delta \mathbf{h}^{(t+1)}} && \Rightarrow \texttt{prev\_grad\_h\_Cost} \\
\frac{\delta \mathbf{h}^{(t+1)}}{\delta \mathbf{h}^{(t)}} &= \mathbf{W} && \Rightarrow \texttt{w\_hh} \\
\nabla_{\mathbf{h}^{(t)}} C & && \Rightarrow \texttt{grad\_h\_Cost}
\end{aligned}
$$

$$
\begin{aligned}
z_t &= Ux_t + Wh_{t-1} + b_t \\
h_t &= \sigma(z_t) \\
o_t &= Vh_t + c_t \\
y_t &= \sigma(Vh_t + c_t) \\
C^t &= C(y_t, \hat{y}_t) \\
\frac{\delta C_t}{\delta W} &= \frac{\delta C_t}{\delta y_t} \cdot \sum_{k=1}^{n} \left[\frac{\delta h_t}{\delta h_\cdot} \cdot\right] \frac{\delta h_k}{W}
\end{aligned}
$$