;

AIM 5011-1: Natural Language Processing (3 credits)
Assignment 2

Fall 2025

adam.faulkner@yu.edu
Course Github site

Class hours: Tuesdays 5:30-7:30pm
Class Room: SCW - 245 Lexington Ave. Rm 101

---

1 Instructions

Complete the following *by hand* and show all of your work in a separate scratch sheet or in the question itself. Use a calculator for complex calculations such as square roots and sigmoid. Scan or take a picture of the completed sheet and upload it to Canvas. **Due: Tuesday, October 7th**

2 RNNs

Refer to Jurafsky & Martin Chapter 13 section 13.1 and to the in-class whiteboard walkthrough.

In the RNN architecture, an input vector representing the current input is multiplied by a weight matrix and then passed through a non-linear activation function to compute the values for a layer of hidden units in much the same manner as a traditional FFN. But, in an RNN, the hidden layer from the previous time step provides a form of memory, or context, that encodes earlier processing and informs the decisions to be made at later points in time. This allows us to model the long-distance dependencies typical of natural language since the the context embodied in the previous hidden layer can include information extending back to the beginning of the sequence.

For this exercise, our RNN consists of the following:

- An input sequence $x = \begin{bmatrix} 3 & 4 & 5 & 6 \end{bmatrix}$
- An activation function $ReLU(z) = max(z, 0)$
- A hidden state layer h_0 initialized to $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

- Three weight matrices W, U, and V

  $W \in \mathbb{R}^{d_h \times d_{in}} =$

  $$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

  $U \in \mathbb{R}^{d_h \times d_h} =$

  $$\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

  $V \in \mathbb{R}^{d_h \times d_h} =$

  $$\begin{bmatrix} -1 & 1 \end{bmatrix}$$

- A hidden layer calculation for timestep t

  $$h_t = ReLU(Uh_{t-1} + Wx_t)$$

- A calculation for output y_t

  $$Vh_t$$

  (normally this is wrapped in a sigmoid but, for the purposes of this exercise, we ignore that)

To calculate y_1, we first generate a new hidden state h_1 using the inner part of the calculation h_t

$$h_1 = (\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} 3) = (\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 3 \\ 6 \end{bmatrix}) = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

and the applying ReLU

$$ReLU(\begin{bmatrix} 3 \\ 6 \end{bmatrix}) = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

The new hidden state h_1 is linearly combined with V to get our first output y_1 using calculation Vh_t

$$y_1 = (\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \end{bmatrix}) = 3$$

Now calculate y_2 using our new hidden state layer $h_1 = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$ and our next input $x_2 = 4$

3 LSTMs

Refer to Jurafsky & Martin Chapter 8 section 13.5, the whiteboard walkthrough in class, and this blogpost.

LSTMs utilize *forget*, *input*, and *output* gates to control the flow of information through the network. This question involves just the input gate. During the LSTM gating calculations, determining which information we're going to store in an updated cell state happens in two stages. First, the input gate calculation, a sigmoid, outputs *which* values to update as a binary mask —

so, output $[0, 1, 0]$ means that only the second value will be updated. Second, a *tanh* layer creates a vector of new candidate values that could be added to the state. In this exercise we're going to be calculating just the first part, the input gate, which is given as

$$i_t = \sigma(W_i \cdot [h_t - 1, x_t] + b_i)$$

Suppose we have the following values for the weight matrix W_i, the input vector x_t, the hidden state $h_t - 1$, and the bias vector b_i

$$W_i = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$x_t = \begin{bmatrix} 7 & 7 & 7 \end{bmatrix}$$

$$h_t - 1 = \begin{bmatrix} 8 & 8 & 8 \end{bmatrix}$$

$$b_i = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

1. Calculate the binary mask vector by first finding the value for $W_i \cdot [h_t - 1, x_t] + b_i$ which involves multiplying the weight matrix by the concatenation of $h_t - 1$ and x_t and then adding the bias b_i vector to the result. Then, apply the sigmoid σ function elementwise to the output of $W_i \cdot [h_t - 1, x_t] + b_i$ to get the binary mask.

2. How should the binary mask vector you calculated for (1) be interpreted?

4 Self-Attention

Jurafsky & Martin chapter 8 provide a set of equations (8.0-8.14) for computing self-attention. We'll be making use of these equations but with a few small modifications to make calculation-by-hand easier. Instead of scaling by the square root of the dimensionality of our feature matrix, as in Jurafsky & Martin eq. 8.11, we define a function *scale* as

$$scale = \frac{x}{\lfloor 2 \rfloor}$$

Function *scale(x)* returns the greatest integer that is less than or equal to the fraction $\frac{x}{2}$. So, $scale(3) = \frac{3}{2} = 1.5 = 1$ since 1 is the greatest integer that is less than or equal to 1.5.
Also, instead of using traditional *softmax* as given in Jurafsky & Martin eq. 8.12, we will approximate softmax by using 3^x rather than e^x.
We have query, key, and value matrices with the following values:

$$W^Q = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$W^K = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$W^V = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We have a feature matrix X:

$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ 2 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Next, we use Jurafsky and Martin eq. 8.9 to calculate our key, query and value vectors.

$$Q = x_i W^Q = \begin{bmatrix} 2 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 3 & 1 & 2 & 1 \end{bmatrix}$$

$$K = x_i W^K = \begin{bmatrix} 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 \end{bmatrix}$$

$$V = x_i W^V = \begin{bmatrix} 20 & 0 & 0 & 20 \\ 0 & 0 & 10 & 10 \\ 1 & 10 & 0 & 0 \end{bmatrix}$$

Given these elements, calculate a final set of attention-weighted features using the following steps.

Step 1: Matrix multiplication: K^T and Q. Transpose K and multiply by Q. The result should be a $4x4$ matrix.

Step 2: Scale each element of the output of Step 1 using the *scale* function.

Step 3: Pseudo-softmax. Calculate 3^x on each member of the cells in the output of Step 2. Then sum the columns. Divide each member of each column by its respective sum. The result is our Attention Weight Matrix.

Step 4: Multiply the value matrix V with the Attention Weight Matrix from Step 3. The result should be a $3x4$ set of attention-weighted features.