CSS605 Natural Language Processing for Smart Assistants
Assignment 1

Noah Kuntner
noahkuntner.2020@mitb.smu.edu.sg

June 16, 2021

Note: Given information is written in shaded boxes, with responses clearly marked by \Answer:"

Question 1 Back Propagation

Let us consider the following single layer feed-forward neural network namely multi-layer perceptron or MLP. The forward propagation for an input x_n through the network is defined as:

$$x_n \overset{V}{\rightarrow} a_n \overset{g(.)}{\rightarrow} Z_n \overset{w}{\rightarrow} b_n \overset{f(.)}{\rightarrow} y_n \qquad (1)$$

V and W are the layer weights, and g(.) and f(.) are the activations (applied element-wise).

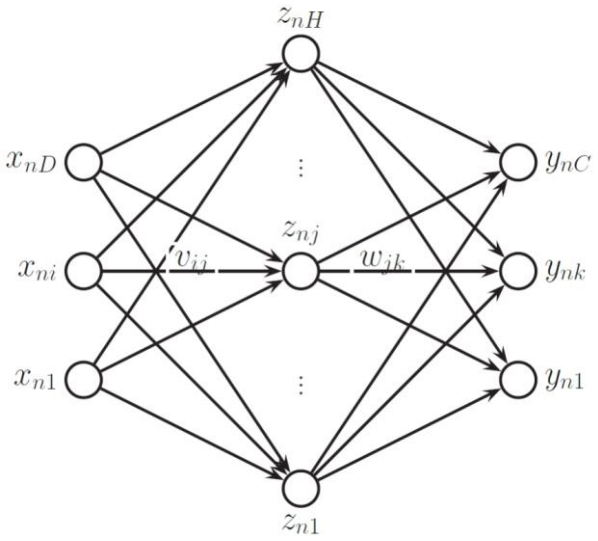

Figure 1: A single layer MLP

1a) Proof of Equation 2

Linear regression and logistic regression are examples of generalized linear models or GLMs. The output layer of a neural network can be modeled as a GLM as shown in the lecture. It was also mentioned that for the GLMs, the gradients of the loss with respect to the parameters of the last layer are:

$$\nabla_{w,k} L(W,V) = (\hat{y}_k - y_k)z \qquad (2)$$

where z is the input vector to the GLM layer, W is the matrix containing the associated weight vectors, and $\hat{y}_k$ is the prediction at the k^{th} output unit (e.g., for k^{th} class).
In this exercise, you have to prove Equation 2, when the GLM is a multi-class logistic regression (i.e., f(.) in Equation 1 is a Softmax function) defined by the following equation:

$$\hat{y}_k = p(y = k \mid z, W, V) = \text{softmax}\,(b)_k = \frac{\exp(W_k^T z)}{\sum_{k'=1}^{K} \exp\,(W_{1,}^T z)} \qquad (3)$$

where K is the total number of classes. Show your derivation.

Answer:

We can model the aforementioned GLM as such:

$$x_n \overset{V}{\rightarrow} a_n \overset{g(.)}{\rightarrow} Z_n \overset{w}{\rightarrow} b_n \overset{f(.)}{\rightarrow} y_n$$

The notation used:

$$a = Vx + c_1$$
$$x_n = \text{ Input Vector}$$
$$\mathbf{z} = g(\boldsymbol{a})$$
$$\boldsymbol{b} = \boldsymbol{W}\mathbf{z} + \boldsymbol{c}_2$$
$$\widehat{\boldsymbol{y}} = f(\boldsymbol{b}) = \text{softmax}\,(\boldsymbol{b})$$

Referring to session 2, slide 60, we obtain:

$$J(\theta) = -\sum\sum y_{nk} \ln\,(\widehat{y}_{nk}(\theta)), \text{ where } \theta = (V, W)$$

As we have,

$$\nabla_{w,k} J(\theta) = \frac{\partial J(\theta)_n}{\partial w} = \frac{\partial J(\theta)_n}{\partial b_{n,k}} \frac{\partial b_{n,k}}{\partial w}$$

Now, we must divide the process between the first and second step in the aforementioned equation:

Solving for the 2nd term:

$$\frac{\partial b_{n,k}}{\partial w} = \sum_i \frac{\partial \sum_i \, w_{ji} z_i}{\partial w_{ji}} = z$$

For the 1st term, this gets:

$$\frac{\partial J(\theta)_n}{\partial b_{n,k}} = \sum_k \frac{\partial J(\theta)_k}{\partial \widehat{y}_k} \frac{\partial \widehat{y}_k}{\partial b_{n,k}}$$

Now solving for the term $\frac{\partial y_k}{\partial b_k}$ using the quotient rule:

$$\frac{\partial y_k}{\partial b_{n,k}} = \frac{\partial \frac{e^{b_k}}{\sum_{k'} \, e^{b_{k'}}}}{\partial b_k} = \frac{\frac{\partial e^{b_k}}{\partial b_k} \sum_{k'} \, b_{k'} - e^{b_k} \frac{\partial \sum_{k'} \, b_{k'}}{\partial b_k}}{\left(\sum_{k'} \, e^{b_{k'}}\right)^2} = \frac{-e^{b_k} \frac{\partial \sum_{k'} \, e^{b_{k'}}}{\partial b_k}}{\left(\sum_{k'} \, e^{b_{k'}}\right)^2}$$

This leads to two cases, where either $k' \neq k$ (1) or $k' = k$ (2).

If $k' \neq k$ we have:

$$\frac{\partial y_k}{\partial b_k} = \frac{-e^{b_k} \frac{\partial \sum_{k'} \, e^{b_{k'}}}{\partial b_{k'}}}{\left(\sum_{k'} \, e^{b_{k'}}\right)^2} = \frac{-e^{b_k} e^{b_{k'}}}{\left(\sum_{k'} \, e^{b_{k'}}\right)^2} = -\hat{y}_{k'} \hat{y}_k$$

If $k' = k$ we have:

$$\frac{\partial y_k}{\partial b_k} = -\frac{\hat{y}_k}{\sum_{k'} e^{b_{k'}}} \frac{\partial \sum_{k'} e^{b_{k'}}}{\partial b_k} = -\hat{y}_k \left(\frac{\frac{\partial e^{b_{k'}}}{\partial b_k} - \sum_{k'} e^{b_{k'}}}{\sum_{k'} e^{b_{k'}}} \right) = -\hat{y}_k \frac{e^{b_{k'}} - \sum_{k'} e^{b_{k'}}}{\sum_{k'} e^{b_{k'}}} = -\hat{y}_k(\hat{y}_k - 1)$$

Which leads to:

$$-\hat{y}_k(\hat{y}_k - 1) = \hat{y}_k(1 - \hat{y}_k)$$

Thus, generalizing we have:

$$\frac{\partial y_k}{\partial b_k} = \begin{cases} -\hat{y}_{k'}\hat{y}_k, k' \neq k, (1) \\ y_k(1 - y_k), k' = k, (2) \end{cases} = y_k\left(\delta_{k,k'} - \hat{y}_{k'}\right)$$

Inserting this into our initial equation for the first term

$$\frac{\partial J(\theta)_n}{\partial b_{n,k}} = \sum_k \frac{\partial J(\theta)_k}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial b_{n,k}}$$

We obtain,

$$\frac{\partial J}{\partial b_k} = \sum_k \frac{\partial J(\theta)_k}{\partial y_k} \frac{\partial y_k}{\partial b_k} = \sum_k \frac{\partial y_k \ln(\hat{y}_k(\theta))}{\partial y_k} y_k\left(\delta_{k,k'} - y_k\right)$$

Now finally, proving that the GLM in Equation 2 is a multi-class logistic regression:

$$\frac{\partial J(\theta)_k}{\partial b_k} = \sum_k -\frac{y_k}{\hat{y}_k} y_k\left(\delta_{k,k'} - y_k\right) = -y_k(1 - y_k) + \sum_{k \neq k'} y_k \hat{y}_k = \hat{y}_k - y_k$$

$$\nabla_{w,k} J(\theta) = z(\hat{y}_k - y_k)$$

1b) Backpropagation using Chain Rule

Now using Backpropagation (chain rule of derivatives), derive the partial derivatives of the loss function with respect to the inner layer weights.

Answer:

Following the same approach as in 1a using the chain, we obtain:

$$\nabla_v J(\theta) = \left(\frac{\partial \boldsymbol{J}(\theta)_k}{\partial \boldsymbol{a}}\right) \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{V}} = \left(\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{b}} \frac{\partial \boldsymbol{b}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \boldsymbol{a}}\right) \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{V}} \quad \text{(a)}$$

The last part can easily be simplified to:

$$\boldsymbol{x} = \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{V}} \quad \text{(b)}$$

The term within the brackets can be plugged into the prior in 1a proven equation:

$$\frac{\partial J(\theta)_k}{\partial b_k} = \cdots = \hat{y}_k - y_k \qquad \text{(c)}$$

We obtain:

$$\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{b}} \frac{\partial \boldsymbol{b}}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \boldsymbol{a}} = (\hat{\boldsymbol{y}} - \boldsymbol{y}) \boldsymbol{W} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{a}} = (\hat{\boldsymbol{y}} - \boldsymbol{y}) \boldsymbol{W} \frac{\partial g(\boldsymbol{a})}{\partial \boldsymbol{a}} \qquad \text{(d)}$$

g(a) is the 1st activation function, but as the problem specifies the softmax for the last layer and not distinctively for the first step, we leave g(a) as it is.

Lastly, plugging our findings for (c) and (d) into (a), we obtain a final matrix multiplication form:

$$\nabla_v J(\theta) = (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

Question 2 Word embeddings

Word embeddings is one of the fundamental ideas in deep learning based NLP. As introduced in the class, word embedding lookup operation can be thought of as one dot product (linear layer) operation as exemplied below. In other words, word2vec models (CBOW, skipgram) are examples of one-layer MLP.

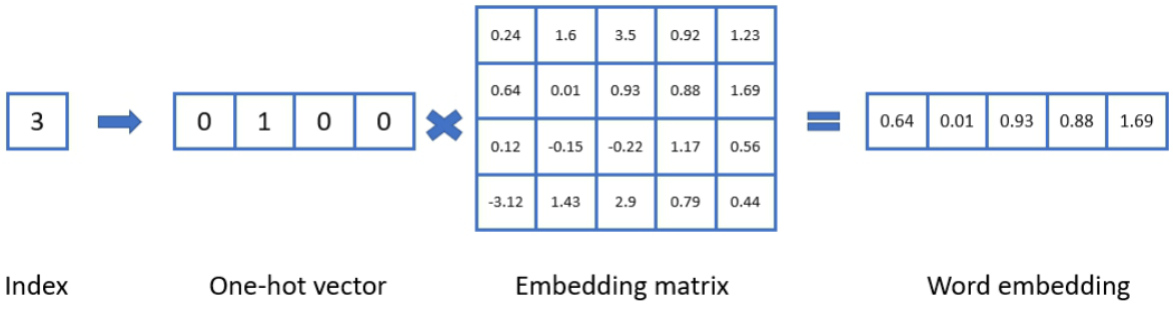

Figure 2: Word embedding lookup

2a) word2vec model

What is the most expensive operation in training a word2vec model and why?

Answer:
The Softmax function used in training is easily the most computationally expensive operation.

Herewith, we have:

$$p(w_{c,j} = w_{O,c} \mid w_I) = \frac{\exp u_{c,j}}{\sum_{j'=1}^{V} \exp\ u_{j'}}$$

Each time weights have to be updated the whole vocabulary must be traversed to update and normalize the scores of each word. This becomes computationally expensive as the size of the total vocabulary increases, whereas the time complexity is O(n), where n is the size of the vocabulary.

2b) Negative sampling

Can you explain how negative sampling solves the above issue?

Answer:
Negative sampling includes randomly sampling K number of words from the vocabulary and determining whether words are neighbours or not through binary classification. This step simplifies the process, as it skips the Softmax-step and changes the problem to a logistic regression, which strongly speeds up the process.

Thus, we have:

$$p(t = 1 \mid w_l) = \text{sig}\left(u_{c,j}\right)$$

The time complexity is followingly reduced to O(K), where K is the the number of words sampled, as the network only uses subsets of the weights in the computation and not of the whole vocabulary.

2c) GloVe embeddings

If 100-dim GloVe embeddings are trained on Twitter dataset, return the nearest 5 tokens to the following words:

Answer:
I import the pre-trained gensim model of Glove embeddings for the Twitter dataset.

```
import gensim.downloader
glove_vectors=gensim.downloader.load('glove-twitter-100')
```

Followingly, to find the 5 closest tokens to the specific token, I am using the "most_similar"-function, which calculates the cosine similarity of a given word and its corresponding vector.

Here we have:

```
for word in ["good", "language"]:

    response = glove_vectors.most_similar(word)
    nearest_neighbors = 5
    for k,v in enumerate(response[:nearest_neighbors]):
        print(f'{v[0]} is the {k} closest to', f'{word}')
        print(f'Its score is {v[1]:.3f}')
```

i) good

Answer:

The five closest words to good are ‘great’, ‘well’, ‘better’, ‘nice’ and ‘too’.

```
great is the 0 closest to good
Its score is 0.900
well is the 1 closest to good
Its score is 0.884
better is the 2 closest to good
Its score is 0.878
nice is the 3 closest to good
Its score is 0.872
too is the 4 closest to good
Its score is 0.851
```

ii) language:

Answer:

The five closest words to language are ‘english’, ‘translation’, ‘languages’, ‘speaking’, ‘’ and ‘speak’, which are all words strongly related to language.

```
english is the 0 closest to language
Its score is 0.809
translation is the 1 closest to language
Its score is 0.793
languages is the 2 closest to language
Its score is 0.788
speaking is the 3 closest to language
Its score is 0.753
speak is the 4 closest to language
Its score is 0.753
```

Question 3 RNN as language model

RNNs have been the workhorse for many problems in NLP. They can naturally be used for encoding a sequence of linguistic units (eg. Characters, words, sentences), for classifying, for generation, and many more. In this exercise, we will use RNN for character level language modeling as shown in Figure 3 below.

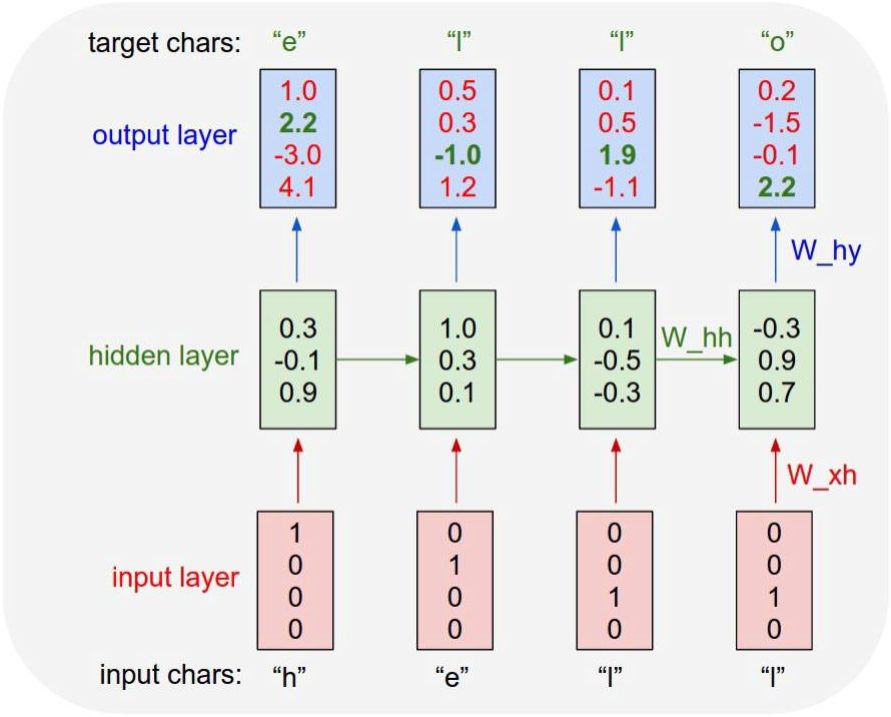

Figure 3: RNN for character level language modeling and generation

3a) Unidirectional RNN composition function

If W_{xh} and W_{hh} are the weight matrices for input to hidden layer and hidden to hidden layer, write down the composition function of a vanilla unidirectional RNN model.

Answer: To be clear, the notation used in regard to Recurrent Neural Networks is the following.

Definitions Used:

$X_t = Input\ at\ each\ interval\ time\ t$
$h_t = Hidden\ value\ at\ each\ interval\ time\ t$
$f(.) = Hidden\ layer\ activation\ function$
$g(.) = \ Output\ layer\ activation\ function$
$Y_t = Output\ at\ each\ interval\ time\ t$
$W_{xh} = Weight\ matrix\ from\ input\ to\ hidden\ layer$
$W_{hh} = Weight\ matrix\ from\ hidden\ layer\ to\ hidden\ layer$
$W_{hy} = Weight\ matrix\ from\ hidden\ layer\ to\ output\ layer$

The composition function of a vanilla unidirectional RNN model is defined as:

$$h_t = f(W_{xh} \times X_t + W_{hh} \times h_{t-1})$$

$$Y_t = g(W_{hy} \times h_t)$$

3b) Bidirectional RNN composition function

Can you write the composition functions for a bidirectional RNN?

Response: A bidirectional RNN diers from a unidirectional RNN in that the latter only considers the impact of the previous hidden layer on the next hidden layer, but a bidirectional RNN considers the impact in both directions.

$h'_t = \ Hidden\ value\ at\ each\ time\ interval\ t$
$W'_{xh} = Weight\ matrix\ from\ the\ hidden\ layer\ to\ the\ opposite\ direction$
$W'_{hh} = Weight\ matrix\ between\ hidden\ layers\ from\ the\ opposite\ direction$
$W'_{hy} = Weight\ matrix\ from\ hidden\ to\ output\ layer\ from\ opposite\ direction$

The composition function of a bidirectional RNN model is:

$$h_t = f(W_{xh} \times X_t + W_{hh} \times h_{t-1})$$
$$h'_t = f(W'_{xh} \times X_t + W'_{hh} \times h_{t+1})$$
$$Y_t = g\left(W_{hy} \times h_t + W'_{hy} \times h'_t\right)$$

3c) RNN LM Output layer

In language modeling (or generation), we use the current hidden state h_t to generate next character/token as show in Figure 3. Write the output layer for this RNN LM.

Response:

$Y^t =$ Predicted word at each time interval t
$N =$ Total number of nodes
$\hat{y}_n^t =$ The n^{th} node's output at each time interval t
$b_{y,n} = Bias\ coeficient\ of\ the\ output\ layer\ of\ node\ n$
$W_{hy,n} = \ The\ weights\ of\ the\ hidden\ to\ output\ layers$
$h^t =$ Hidden state at each time interval t

The output layer of the RNN LM contains N nodes, whereas N is the size of the vocabulary, and the output is simply the argmax of all the nodes.

Herewith, each single node uses a Softmax activation function for the activation resulting from the weight matrix W_{hy}, and the hidden state h_t and the bias coefficient $b_{y,n}$.

The predicted word at time interval t:

$$Y^t = \text{argmax}\,(\hat{y}_n^t)$$

The n^{th} node's output:

$$\hat{y}_n^t = \text{Softmax}\left(W_{hy,n} \cdot h_{t,n} + b_{y,n}\right)$$

$$= \frac{\exp\left(W_{hy,n} \cdot h_{t,n} + b_{y,n}\right)}{\sum_{n'}^{N} \exp\left(W_{hy,n'} \cdot h_{t,n'} + b_{y,n'}\right)}$$

3d) Vanishing Gradient

Explain why Gradient Vanishing is a problem in particular with RNNs?

Answer:

The gradient vanishing problem for RNNs exists largely due to:

1. **Long sequences**
2. **Backpropagation through time**

In a vanilla unidirectional RNN model exist three different weight matrices, which are continuously updated.

1. W_{hh}
2. W_{xh}
3. W_{hy}

To illustrate within the weight matrix W_{xh}, by backpropagating at time t, the resulting gradient is:

$$\frac{\partial L_t}{\partial W_{xh}} = \sum_{k=0}^{t} \frac{\partial L_t}{\partial Y_t} \times \frac{Y_t}{S_t} \times \left(\prod_{j=k+1}^{t} \frac{\partial S_j}{\partial S_{j-1}} \right) \times \frac{\partial S_k}{\partial W_{xh}}$$

To obtain the gradient generated by the loss function at each time interval *t*, one needs the product of all information from 1 to time t-1. Additionally, the activation function between hidden layer to hidden layer is usually a sigmoid or tanh, which tend to zero when *t* is large and naturally take values that are smaller than 1.

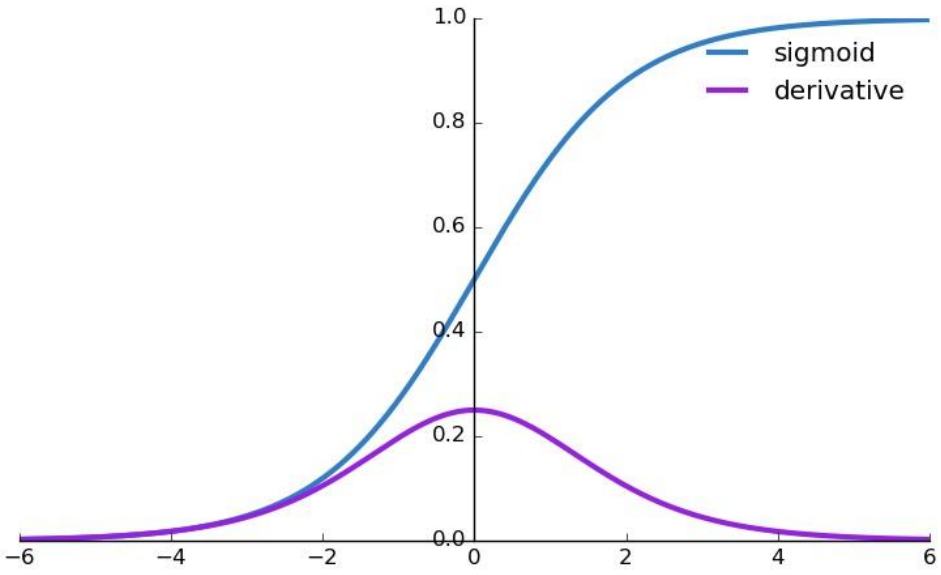

Figure 4: Sigmoid Derivative

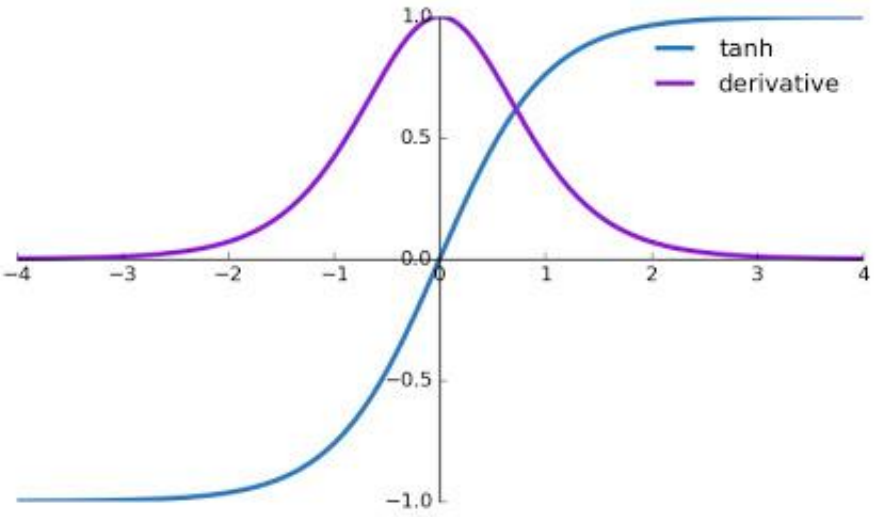

Figure 4: Tanh Derivative

It can be seen again that both sigmoid′ and tanh′ both tend to zero when t is large, which results in the aforementioned vanishing gradient problem.

$$\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} = \prod_{j=k+1}^{t} (tanh' \mid sigmoid') W_{xh}$$

The vanishing gradient is a typical problem for all deep feedforward networks, whereas this vanishing gradient within larger RNNs is aggravated due to high interconnectedness.

Question 4 Seq2Seq for NMT

In Machine Translation, our goal is to convert a sentence from the source language (e.g., French) to the target language (e.g., English). In this exercise, we will learn to build a Neural Machine Translation (NMT) system with a sequence-to-sequence (Seq2Seq) model.

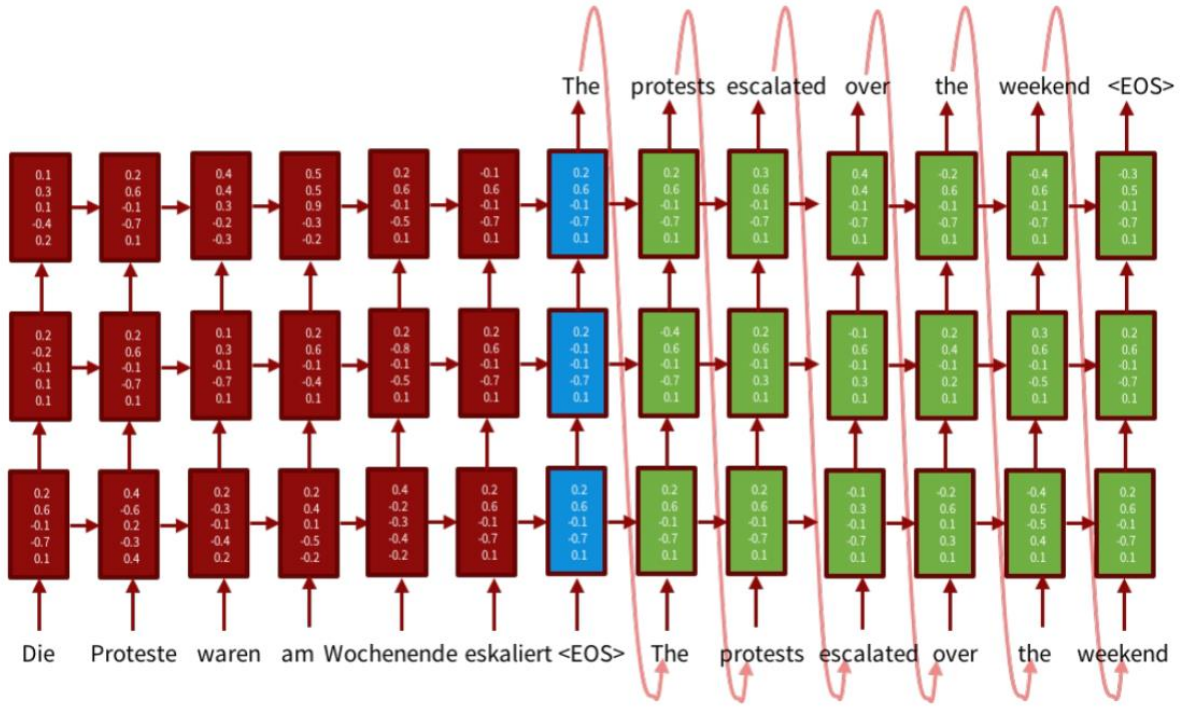

Figure 5: A Seq2Seq model for NMT

4a) Source Encoder

The NMT model uses a Bidirectional LSTM to encode a source language sentence. Given a source sentence with m tokens $s = x_1, \dots, x_m$, first show how the encoder encodes the sentence.

The model for the neural machine translation will be constructed as displayed in the Figure 5. The bidirectional LSTM-model us in fact two separate LSTM models working together - they do not share hidden state or weights, with one training the other in order, $x_1 \rightarrow x_m$, thus designated as forward layer, whereas the other one is trained in the order $x_m \rightarrow x_1$, named the backward layer. The encoder, thus, encodes the text as a vector.

As for the forward layer, the first input is the word vector of the token x_1, and it followingly outputs the hidden state h_1. This is repeated until the last token m, which takes the word vector of token x_m as input and outputs the last hidden state h_m.

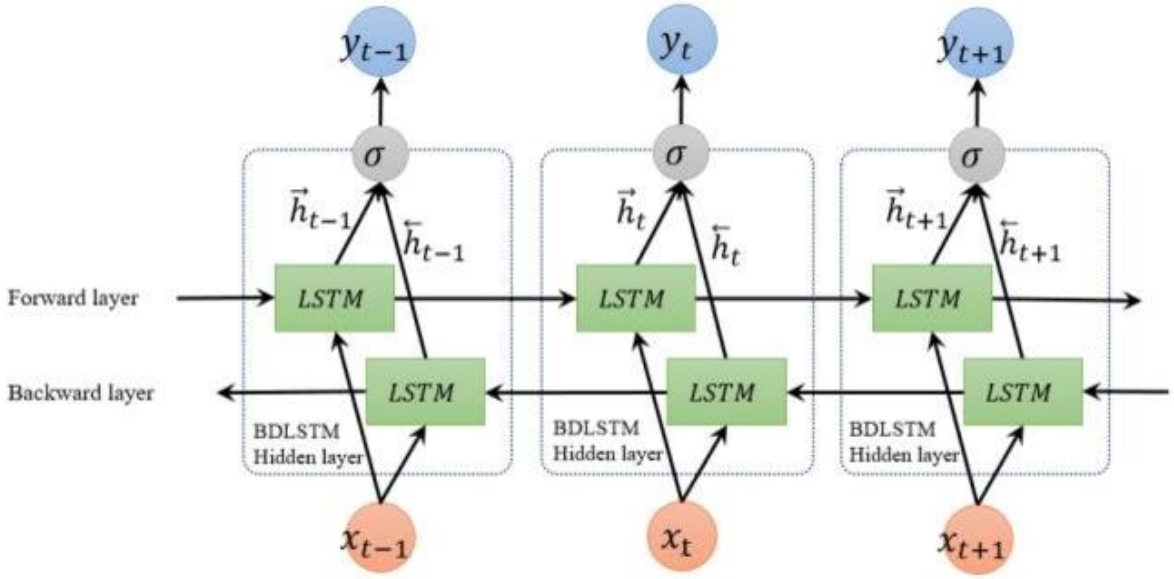

Figure 6: Bidirectional LSTM Illustration

For the backward layer, this process works in the opposite way. The first input is the word vector of token x_m, and it outputs the hidden state h_m. It then repeats this with the reverse strategy until token x_1, which takes as input the word vector of the token x_1 and returns the final hidden state h'_1. In general, for a single bidirectional LSTM, the source sentence's encoding is the concatenation of the last hidden states for both LSTM models, $[h_m, h'_1]$

In cases with more layers, the input will be the concatenation of $[h_1, h'_1]$.

4b) Target Decoder

The decoder (language model) is a unidirectional LSTM. Write down the decoder with auto-regressive factorization for the target sentence $t = y_1, \dots, y_n$ without considering the source.

Answer:

By using the decoder for the prediction and not the training, the model has no source text to start from, thus the decoder network does not use the previous context vector to learn from the encoder network, but it begins with a special character the marks the beginning of the target sentence → BoS. Thus, the decoder is initially presenting from the piece of text a vector with no a priori knowledge.

e.g.:
Considering a target sentence of $t = y_1, y_2, \dots, y_n$, the auto-regressive factorization for the target sentence denotes:

$$\hat{y}_t = \text{argmax}\left(P(\hat{y}_t)\right) = \prod_{i=1}^{t} P(\hat{y}_i \mid y_0, y_1, \dots, y_{i-1}, T) \quad (29)$$

4c) Conditional Language Modeling as MT

Now modify the decoder LM equation to make it a conditional language model by conditioning it on the source sentence s.

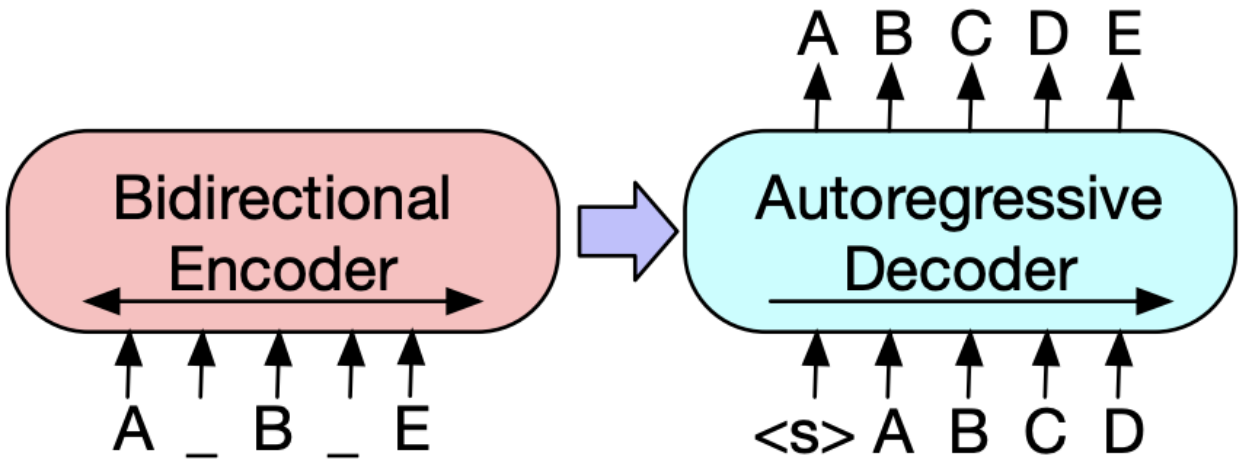

Figure 7: Bidirectional Auto Regressive Transformer Illustration

Answer:

The predicting decoder takes as input the context vector V learned from the encoder. Thus, the prediction of the previous word will be used in predicting the next word. This is illustrated in the BART-modelling in the former figure 7.

The prediction is thus of arbitrary value at time t:

$$\hat{y}_t = \text{argmax}\left(P(\hat{y}_t)\right)$$
$$= \prod_{i=1}^{t} P(\hat{y}_i \mid \hat{y}_0, \hat{y}_1, \ldots, \hat{y}_{i-1}, V)$$

4d) NMT Loss

Write the Maximum Likelihood or Cross Entropy loss function of the NMT model for the input-output pair (s;t).

Answer: The maximum likelihood function for a NMT model with n input-output pairs is defined as:

$$L_{MLE}(\theta^*) = -\sum_{i=1}^{n} \log\left(P\left(t^i \mid s^i; \theta^*\right)\right) \quad (31)$$

By which, we have $\theta^* \rightarrow$

$$\theta^* = \text{argmax} \prod_{i=1}^{n} P(t^i \mid s^i; \theta) \quad (32)$$

Herewith, we have the cross-entropy loss function to be the total negative log likelihoods of each pairing of input-output values, defined as:

$$\mathcal{L} = -\sum_{i=1}^{n} y_{s^i,t^i} \log(\hat{y}_{s,t})$$

By which, the notation is defined as:

$$\hat{y}_{s^i,t^i} = \text{Predicted probability of output } t^i \text{ given input } s^i$$

$$y_{s^i,t^i} = \begin{cases} 1 & \text{if prediction is correct;} \\ 0 & \text{otherwise} \end{cases}$$