Popular Machine Learning Methods: Idea, Practice and Math

Recurrent Neural Networks

Yuxiao Huang

Data Science, Columbian College of Arts & Sciences
George Washington University

Fall 2020

Reference

- This set of slices was largely built on the following 7 wonderful books and a wide range of fabulous papers:

  - **HML** Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)
  - **PML** Python Machine Learning (3rd Edition)
  - **ESL** The Elements of Statistical Learning (2nd Edition)
  - **PRML** Pattern Recognition and Machine Learning
  - **NND** Neural Network Design (2nd Edition)
  - **LFD** Learning From Data
  - **RL** Reinforcement Learning: An Introduction (2nd Edition)

- For most materials covered in the slides, we will specify their corresponding books and papers for further reference.

Code Example & Case Study

- See related code example in github repository: /p3_c2_s4_recurrent_neural_networks/code_example
- See related case study in github repository: /p3_c2_s4_recurrent_neural_networks/case_study

Table of Contents

Learning Objectives: Expectation

- It is **expected** to understand
  - the architecture and idea of short-term Recurrent Neural Networks (RNNs)
  - the architecture and idea of long-term RNNs:
    - Long Short-Term Memory (LSTM)
    - Gated Recurrent Unit (GRU)
  - the architecture and idea of Encoder-Decoder Networks
  - the architecture and idea of Bidirectional RNNs
  - the architecture and idea of Attention Mechanisms
  - the architecture and idea of Transformer
  - the good practices for building RNNs
  - the idea of and good practices for transfer learning using state-of-the-art pretrained RNNs

Learning Objectives: Recommendation

- It is **recommended** to understand
  - the backpropagation for RNNs:
    - Backpropagation Through Time (BPTT)
    - Real-Time Recurrent Learning (RTRL)

IMDB 50K Movie Review Dataset

Figure 1: Kaggle competition: IMDB dataset of 50K movie reviews. Picture courtesy of Kaggle.

- IMDB 50K movie review dataset: a large movie review dataset for binary sentiment classification:
  - features: movie review (in text)
  - target: sentiment for the movie (positive or negative)

Recurrent Perceptron

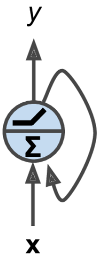

Figure 2: A recurrent perceptron. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- In *Feedforward Neural Networks*, information always passes forward (from lower layer to higher layer).
- In *Recurrent Neural Networks* (RNNs), information also passes backward (from higher layer to lower layer).
- Fig. 2 shows a recurrent perceptron where the output of the output layer goes back to the input of the input layer.

Unrolling Recurrent Perceptron Through Time

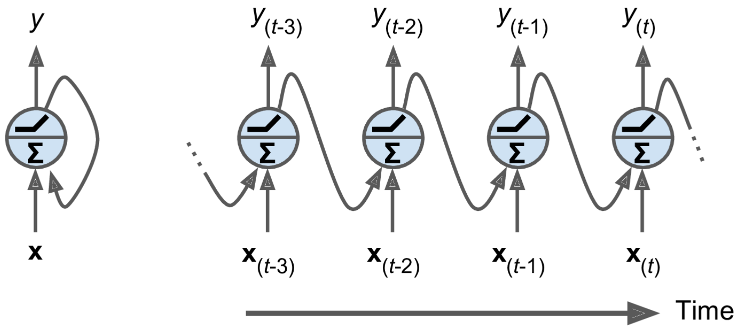

Figure 3: A recurrent perceptron unrolled through time. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- We can represent the left-most recurrent perceptron in fig. 3 as a function of time:
  - at time step 0 (the first time step), the perceptron receives input at time 0
  - at time step t (where $t > 0$), the perceptron receives not only input at time t, $\mathbf{x}_{(t)}$, but also its output at time $t-1$, $\mathbf{y}_{(t-1)}$
- This representation is called *Unrolling the Network Through Time*.

Recurrent Layer

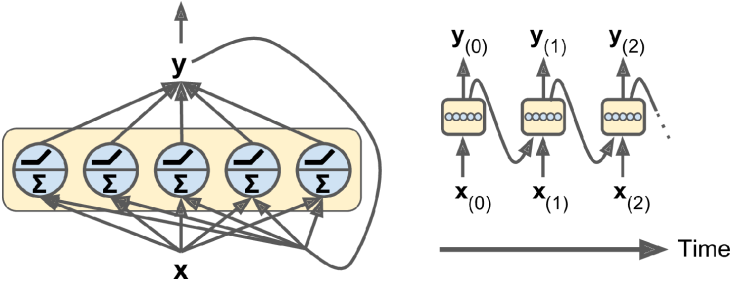

Figure 4: A layer of recurrent perceptrons unrolled through time. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- We can create a layer of recurrent perceptrons and represent it as a function of time:
  - at time step 0 (the first time step), each perceptron on the layer receives input at time 0, $\mathbf{x}_{(0)}$
  - at time step t (where $t > 0$), each perceptron on the layer receives:
    - input at time t, $\mathbf{x}_{(t)}$
    - output at time $t-1$, $\mathbf{y}_{(t-1)}$

Output of a Recurrent Layer

- The output of a recurrent layer at time step t, $\mathbf{y}_{(t)}$, can be written as

$$\mathbf{y}_{(t)} = \mathbf{f}\left(\mathbf{W}_x \mathbf{x}_{(t)}^\intercal + \mathbf{W}_y \mathbf{y}_{(t-1)}^\intercal + \mathbf{b}\right). \tag{1}$$

  Here:
  - $\mathbf{f}$ is the activation function (e.g., ReLU) of the layer
  - $\mathbf{x}_{(t)}$ is the input at time t
  - $\mathbf{W}_x$ is the connecting weight between the perceptrons on the layer and $\mathbf{x}_{(t)}$
  - $\mathbf{y}_{(t-1)}$ is the output of the layer at time $t-1$
  - $\mathbf{W}_y$ is the connecting weight between the perceptrons on the layer and $\mathbf{y}_{(t-1)}$
  - $\mathbf{b}$ is the bias of the perceptrons on the layer
- The recursive relationship in eq. (1) shows that $\mathbf{y}_{(t)}$ is a function of the input across all the previous time steps, $\mathbf{x}_{(0)}, \mathbf{x}_{(1)}, \cdots, \mathbf{x}_{(t-1)}$.

Memory Cell

- As mentioned earlier, the output of a recurrent perceptron at a time step is a function of the input across all the previous time steps.
- That is, a recurrent perceptron has a form of memory that can preserve some state (more on this later) across time steps.
- Indeed, a recurrent perceptron is one kind of memory cell that can only learn short patterns (typically about 10 steps long).
- This kind of cell is also called *Short-Term Memory Cell*.
- There are more complex memory cells that allow longer patterns (typically about 10 times longer, more on this later).
- This kind of cell is also called *Long-Term Memory Cell*.

Output and Hidden State

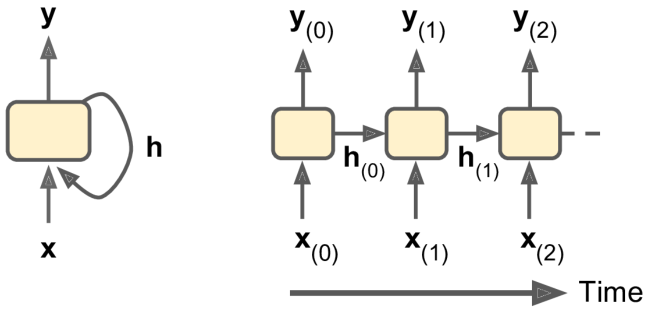

Figure 5: Output and hidden state of a memory cell. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- A memory cell's state at time step t, $\mathbf{h}_{(t)}$, is a function of the input at t, $\mathbf{x}_t$, and its state at $t-1$, $\mathbf{h}_{(t-1)}$.
- A memory cell's output at time step t, $\mathbf{y}_{(t)}$, is also a function of the input at t, $\mathbf{x}_t$, and its output at $t-1$, $\mathbf{y}_{(t-1)}$.
- While $\mathbf{h}_{(t)}$ and $\mathbf{y}_{(t)}$ could be the same in simple cells (left panel in fig. 5), they might be different in more complex cells (right panel in the figure).

Input and Output Sequences

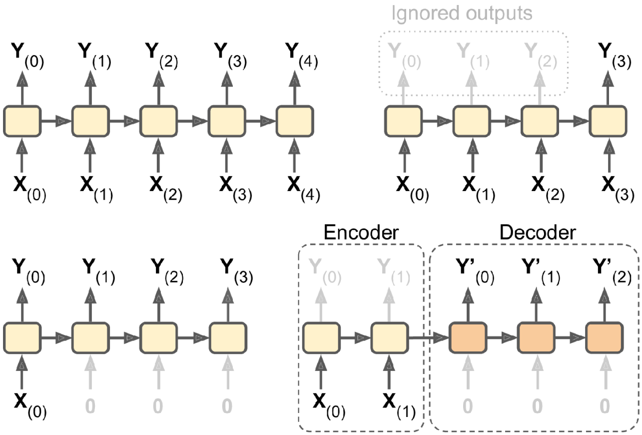

Figure 6: Four types of input and output sequences of RNNs. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

Input and Output Sequences

- *Sequence-to-Sequence Network* (top-left panel in fig. 6):
  - RNNs that take as input a sequence and output a sequence
  - e.g., given the input (a sequence), predict the output (also a sequence) that is some time steps later than the input
- *Sequence-to-Vector Network* (top-right panel in fig. 6):
  - RNNs that take as input a sequence and output a vector
  - e.g., given the book review (a sequence), predict the book rating (a vector)
- *Vector-to-Sequence Network* (bottom-left panel in fig. 6):
  - RNNs that take as input a vector and output a sequence
  - e.g., given an image (a vector), predict its caption (a sequence)
- *Encoder-Decoder Network* (bottom-right panel in fig. 6):
  - RNNs that begin with a sequence-to-vector network, named *Encoder*, and end with a vector-to-sequence network, named *Decoder*
  - e.g., given a sentence in one language (a sequence), encode it into a latent representation (a vector), which is then decoded into a sentence in another language (a sequence)
  - sequence-to-sequence network may not work here since we may have to see the whole sentence to know its meaning

Deep RNNs

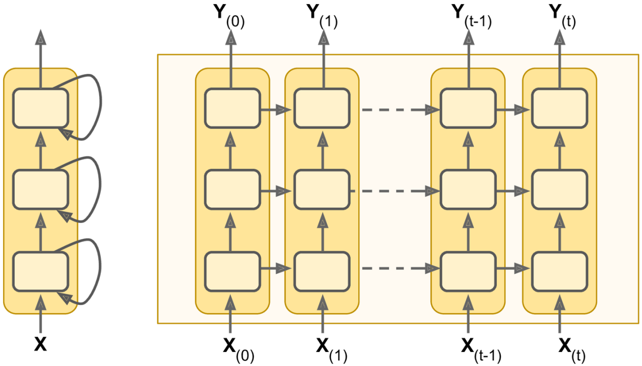

Figure 7: A deep RNN with three layers. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- Previously we have been focusing on RNNs that have only one hidden layer.
- Similar to DNNs, RNNs can also have many hidden layers, making them *Deep RNNs*.
- Fig. 7 snows a deep RNN with three layers.

Further Reading

- See HML: Chap 15 for a high-level description of a special backpropagation particularly designed for RNNs, named *Backpropagation Through Time* (BPTT).
- See NND: Chap 14 for:
  - a detailed discussion of BPTT
  - a detailed discussion of another backpropagation, also particularly designed for RNNs, named *Real-Time Recurrent Learning* (RTRL)
  - the pros and cons of BPTT and RTRL

The Short-Term Memory Problem

- After we pass data to RNNs, the data will go through a series of transformations (e.g., when calculating the net input and activation on each layer).
- Such transformations can also repeat themselves since we feed the output of RNNs back to the input layer.
- As a result, when RNNs receive the later elements of a long sequence, the fed back output may be very different from the earlier elements of the long sequence.
- In other words, we can think of RNNs as *Dory the fish* who, when translating the last word in a long sentence, already forgets about the first word in the sentence.
- This is called the *Short-Term Memory Problem*.

Long-Term Memory Cell

- To address the short-term memory problem of short-term memory cells, many cells with long-term memory have been proposed.
- Here we will introduce two popular long-term memory cells:
  - *Long Short-Term Memory* (LSTM) cell [Hochreiter and Schmidhuber, 1997]
  - *Gated Recurrent Unit* (GRU) cell [Cho et al., 2014]
- Compared to short-term memory cells, LSTM and GRU usually perform much better:
  - training LSTM and GRU could converge faster
  - LSTM and GRU could detect long-term dependencies in the data

LSTM Cell, As a Blackbox

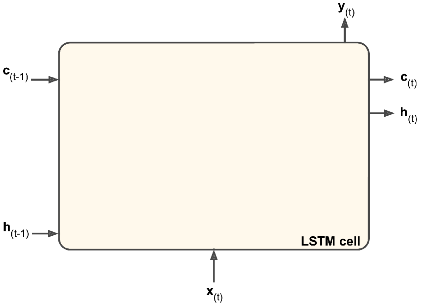

Figure 8: LSTM cell, as a blackbox. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- If we were to treat LSTM as a blackbox and only focus on its input and output, then the only difference between LSTM and short-term memory cell would be:
  - short-term memory cell has input $\mathbf{x}_{(t)}$, output $\mathbf{y}_{(t)}$ and short-term states $\mathbf{h}_{(t-1)}$ and $\mathbf{h}_{(t)}$
  - LSTM has extra long-term states $\mathbf{c}_{(t-1)}$ and $\mathbf{c}_{(t)}$
- The long-term states are the reason why LSTM works better on long sequences.

LSTM Cell: Architecture

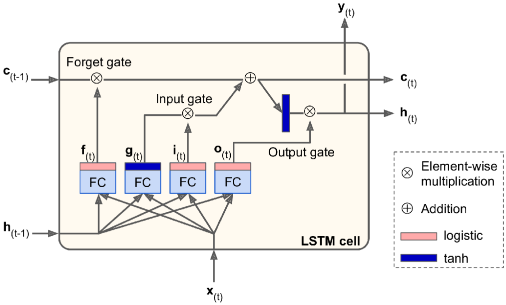

Figure 9: The architecture of LSTM cell. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- There are four kinds of fully connected layers in LSTM, which all take as input $\mathbf{h}_{(t-1)}$ and $\mathbf{x}_{(t)}$.
- Concretely:
  - the *Main Layer* outputs the activation ($\texttt{tanh}$), $\mathbf{g}_{(t)}$
  - the *Forget Gate Controller* outputs the activation ($\texttt{sigmoid}$), $\mathbf{f}_{(t)}$
  - the *Input Gate Controller* outputs the activation ($\texttt{sigmoid}$), $\mathbf{i}_{(t)}$
  - The *Output Gate Controller* outputs the activation ($\texttt{sigmoid}$), $\mathbf{o}_{(t)}$

LSTM Cell: Architecture

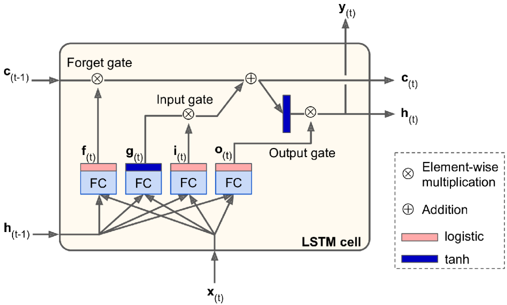

Figure 9: The architecture of LSTM cell. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- There are three major steps in LSTM:
  - recognize important input, with the *Input Gate* (taking as input $\mathbf{g}_{(t)}$ and $\mathbf{i}_{(t)}$)
  - store the important input in the long-term state for as long as it is needed, with the *Forget Gate* (taking as input $\mathbf{c}_{(t-1)}$ and $\mathbf{f}_{(t)}$)
  - extract the important input whenever it is needed, with the *Output Gate* (taking as input $\tan(\mathbf{c}_{(t)})$ and $\mathbf{o}_{(t)}$)

LSTM Cell: Math

- The output of the input gate controller at time step t, $\mathbf{i}_{(t)}$, is

$$\mathbf{i}_{(t)} = \text{sigmoid}\left(\mathbf{W}_{xi}\mathbf{x}_{(t)}^{\intercal} + \mathbf{W}_{hi}\mathbf{h}_{(t-1)}^{\intercal} + \mathbf{b}_i\right). \tag{2}$$

- The output of the forget gate controller at time step t, $\mathbf{f}_{(t)}$, is

$$\mathbf{f}_{(t)} = \text{sigmoid}\left(\mathbf{W}_{xf}\mathbf{x}_{(t)}^{\intercal} + \mathbf{W}_{hf}\mathbf{h}_{(t-1)}^{\intercal} + \mathbf{b}_f\right). \tag{3}$$

- The output of the output gate controller at time step t, $\mathbf{o}_{(t)}$, is

$$\mathbf{o}_{(t)} = \text{sigmoid}\left(\mathbf{W}_{xo}\mathbf{x}_{(t)}^{\intercal} + \mathbf{W}_{ho}\mathbf{h}_{(t-1)}^{\intercal} + \mathbf{b}_o\right). \tag{4}$$

- The output of the main layer at time step t, $\mathbf{g}_{(t)}$, is

$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}\mathbf{x}_{(t)}^{\intercal} + \mathbf{W}_{hg}\mathbf{h}_{(t-1)}^{\intercal} + \mathbf{b}_g\right). \tag{5}$$

- The long-term state at time step t, $\mathbf{c}_{(t)}$, is

$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}. \tag{6}$$

- The short-term state and output at time step t, $\mathbf{h}_{(t)}$ and $\mathbf{y}_{(t)}$, are

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\left(\mathbf{c}_{(t)}\right). \tag{7}$$

LSTM Cell: Math

- In eqs. (2) to (5):
  - $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$ and $\mathbf{W}_{xg}$ are the connecting weights between the input at time step t, $\mathbf{x}_{(t)}$, and the input gate controller, forget gate controller, output gate controller and the main layer
  - $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$ and $\mathbf{W}_{hg}$ are the connecting weights between the short-term state at time step $t-1$, $\mathbf{h}_{(t-1)}$, and the input gate controller, forget gate controller, output gate controller and the main layer
  - $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$ and $\mathbf{b}_g$ are the biases of the input gate controller, forget gate controller, output gate controller and the main layer
- In eq. (6):
  - $\mathbf{f}_{(t)}$, $\mathbf{i}_{(t)}$ and $\mathbf{g}_{(t)}$ are the output of the forget gate controller, input gate controller and the main layer at time step t
  - $\mathbf{c}_{(t-1)}$ is the long-term state at time step $t-1$
- In eq. (7):
  - $\mathbf{o}_{(t)}$ is the output of the output gate controller at time step t
  - $\mathbf{c}_{(t)}$ is the long-term state at time step t

Peephole Connections

- In regular LSTM, the input/forget/output gate controller only receives the input at time step t, $\mathbf{x}_{(t)}$, and the short-term state at time step $t-1$, $\mathbf{h}_{(t-1)}$.
- An extension of LSTM with extra connections called *Peephole Connections* was proposed in [Gers and Schmidhuber, 2000]:
  - the long-term state at time step $t-1$, $\mathbf{c}_{(t-1)}$, is added as an input to the input and forget gate controller
  - the long-term state at time step t, $\mathbf{c}_{(t)}$, is added as an input to the output gate controller

GRU Cell, As a Blackbox

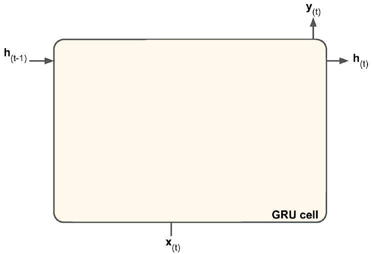

Figure 10: GRU cell, as a blackbox. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- If we were to treat GRU as a blackbox and only focus on its input and output, then there would be no difference between GRU and short-term memory cell:
  - both cells have input $\mathbf{x}_{(t)}$, output $\mathbf{y}_{(t)}$ and short-term states $\mathbf{h}_{(t-1)}$ and $\mathbf{h}_{(t)}$
- Note that LSTM has extra long-term states $\mathbf{c}_{(t-1)}$ and $\mathbf{c}_{(t)}$.

GRU Cell: Architecture

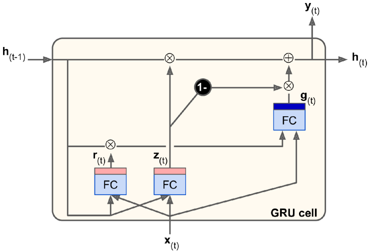

Figure 11: The architecture of GRU cell. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- There are three kinds of fully connected layers in GRU:
  - a gate controller (receiving $\mathbf{h}_{(t-1)}$ and $\mathbf{x}_{(t)}$) outputs the activation ($\texttt{sigmoid}$), $\mathbf{z}_{(t)}$
  - a gate controller (receiving $\mathbf{h}_{(t-1)}$ and $\mathbf{x}_{(t)}$) outputs the activation ($\texttt{sigmoid}$), $\mathbf{r}_{(t)}$
  - the *Main Layer* (receiving $\mathbf{r}_{(t)}$ and $\mathbf{x}_{(t)}$) outputs the activation ($\texttt{tanh}$), $\mathbf{g}_{(t)}$

GRU Cell: Architecture

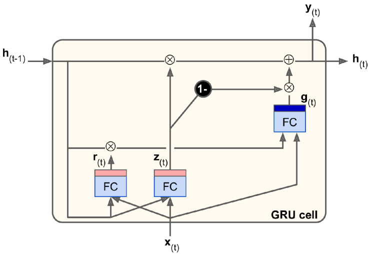

Figure 11: The architecture of GRU cell. Picture courtesy of *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).*

- There are three major steps in GRU:
  - recognize important input (that should be used when producing the output), with gate controller that outputs $\mathbf{z}_{(t)}$ and main layer that outputs $\mathbf{g}_{(t)}$
  - store the important input (that should be used when producing the output) in the state for as long as it is needed, with gate controller that outputs $\mathbf{z}_{(t)}$ (the 1- operation allows the controller to switch between its two functions)
  - recognize important input (that should be used when producing $\mathbf{g}_{(t)}$), with gate controller that outputs $\mathbf{r}_{(t)}$

GRU Cell: Math

- The output of a gate controller at time step t, $\mathbf{z}_{(t)}$, is

$$\mathbf{z}_{(t)} = \text{sigmoid}\left(\mathbf{W}_{xz}\mathbf{x}_{(t)}^{\intercal} + \mathbf{W}_{hz}\mathbf{h}_{(t-1)}^{\intercal} + \mathbf{b}_{z}\right). \tag{8}$$

- The output of a gate controller at time step t, $\mathbf{r}_{(t)}$, is

$$\mathbf{r}_{(t)} = \text{sigmoid}\left(\mathbf{W}_{xr}\mathbf{x}_{(t)}^{\intercal} + \mathbf{W}_{hr}\mathbf{h}_{(t-1)}^{\intercal} + \mathbf{b}_{r}\right). \tag{9}$$

- The output of the main layer at time step t, $\mathbf{g}_{(t)}$, is

$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}\mathbf{x}_{(t)}^{\intercal} + \mathbf{W}_{hg}\left(\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}\right)^{\intercal} + \mathbf{b}_{g}\right). \tag{10}$$

- The short-term state and output at time step t, $\mathbf{h}_{(t)}$ and $\mathbf{y}_{(t)}$, are

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + \left(1 - \mathbf{z}_{(t)}\right) \otimes \mathbf{g}_{(t)}. \tag{11}$$

GRU Cell: Math

- In eqs. (8) to (10):
  - $\mathbf{W}_{xz}$, $\mathbf{W}_{xr}$ and $\mathbf{W}_{xg}$ are the connecting weights between the input at time step t, $\mathbf{x}_{(t)}$, and the two gate controllers and the main layer
  - $\mathbf{W}_{hz}$, $\mathbf{W}_{hr}$ and $\mathbf{W}_{hg}$ are the connecting weights between the short-term state at time step $t-1$, $\mathbf{h}_{(t-1)}$, and the two gate controllers and the main layer
  - $\mathbf{b}_z$, $\mathbf{b}_r$ and $\mathbf{b}_g$ are the biases of the two gate controllers and the main layer
- In eq. (10):
  - $\mathbf{r}_{(t)}$ is the output of a gate controller at time step t
- In eq. (11):
  - $\mathbf{z}_{(t)}$ is the output of a gate controller at time step t
  - $\mathbf{g}_{(t)}$ is the output of the main layer at time step t

Encoder-Decoder based Neural Machine Translation

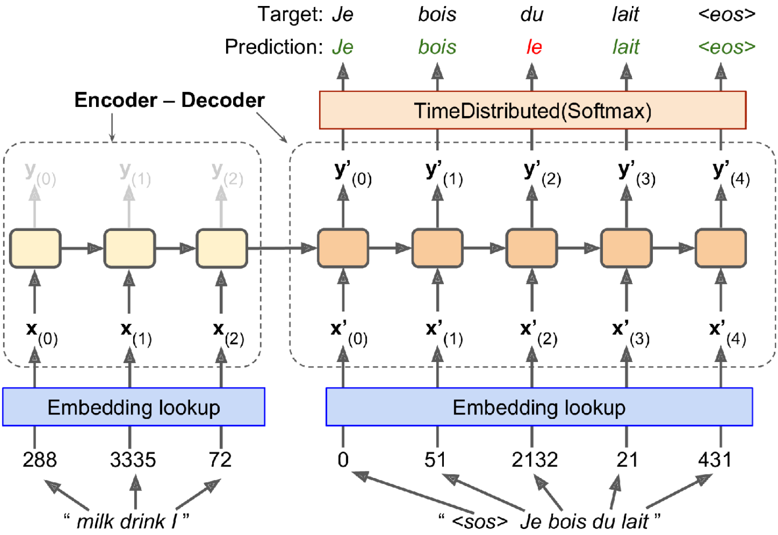

Figure 12: An encoder-decoder based neural machine translation model.

Encoder-Decoder based Neural Machine Translation

- Fig. 12 shows that, the encoder takes as input a reversed English sentence and outputs the hidden states (a representation of the input sentence).
- The input sentence was reversed because the decoder needs to translate the last word in the reversed sentence (i.e., first word in the original sentence) first.
- The decoder takes as input the hidden states of the encoder and
  - the target French sentence shifted by one step (during training), where $< \text{sos} >$ stands for *start of sentence*
  - the output of the decoder at the previous step, as shown in fig. 13 (during testing)
- The decoder outputs the translated French sentence where $< \text{eos} >$ stands for *end of sentence*.

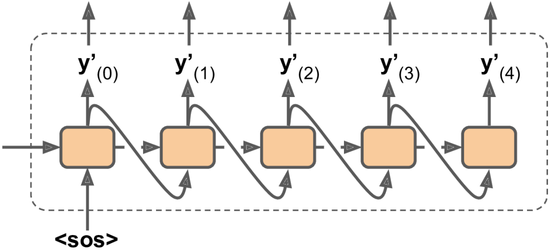

Figure 13: The input of decoder during testing.

The Idea of Bidirectional RNNs

- For all the RNNs discussed so far, their recurrent layers only look at the past inputs to predict the output:
  - "I have not had any meal today so I am ______."
  - based on the words prior to the blank, *hungry* is more likely than *full*
- However, sometimes we may have to look at the future inputs to predict the output:
  - "I am ______ because I have not had any meal today."
  - if we were to only consider the words before the blank, *hungry* and *full* would be equally likely (which is wrong)
  - if, instead, we can somehow also consider the words after the blank, *hungry* will be more likely than *full* (which is correct)
- The idea of *Bidirectional RNNs* is using both the past and future inputs (hence the name) for prediction.

The Architecture of Bidirectional RNNs

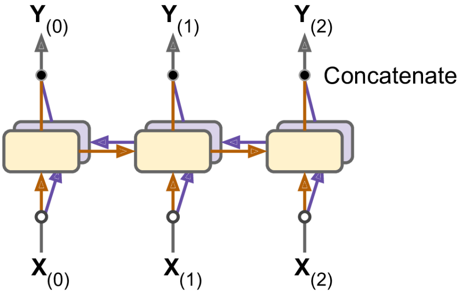

Figure 14: A bidirectional recurrent layer.

- The key building block in bidirectional RNNs is the *Bidirectional Recurrent Layer* (see fig. 14).
- A bidirectional recurrent layer comprises two recurrent layers:
  - one passes the input from left to right (to consider inputs before the blank)
  - one passes the input from right to left (to consider inputs after the blank)
- The output of the two recurrent layers will then be combined (typically concatenated).

Attention Mechanisms: Idea

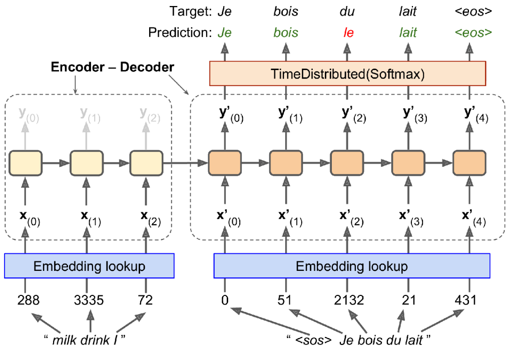

Figure 12: An encoder-decoder based neural machine translation model.

- As shown in fig. 12, in standard encoder-decoder based neural machine translation model, the path from "milk" to "lait" is fairly long.
- This long path could compromise the translation accuracy due to the short-term memory limitations of RNNs.
- The idea of the *Attention Mechanism* is that, at the time step where the decoder should output "lait", it will focus its attention on "milk" (so as to shorten the path between the two).

Attention Mechanisms: Architecture

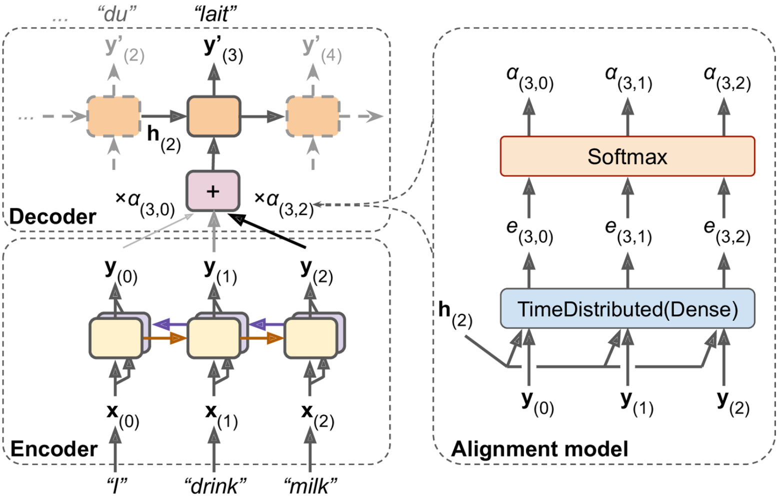

Figure 15: An encoder-decoder based neural machine translation model with attention.

Attention Mechanisms: Architecture

- Unlike in standard encoder-decoder neural machine translation model where the decoder only receives the hidden state from the encoder, in attention mechanism the decoder at each time step also receives the output of the encoder across each time step.
- At each time step, the decoder calculates a weighted sum of the encoder outputs.
- This weighted sum determines which words the decoder should focus on at this time step.
- Concretely, weight $\alpha_{(t,i)}$ is the weight of the i^{th} encoder output at the t^{th} decoder time step.
- For example, in fig. 15:
  - $\alpha_{(3,0)}$ is the weight of the 0^{th} encoder output at the 3^{rd} decoder time step
  - $\alpha_{(3,1)}$ is the weight of the 1^{st} encoder output at the 3^{rd} decoder time step
  - $\alpha_{(3,2)}$ is the weight of the 2^{nd} encoder output at the 3^{rd} decoder time step
- If $\alpha_{(3,2)}$ is much larger than $\alpha_{(3,0)}$ and $\alpha_{(3,1)}$, the decoder will pay more attention to the 2^{nd} encoder output, and in turn, the 2^{nd} encoder input (word "milk").

Bahdanau Attention

- The weights in fig. 15, $\alpha_{(t,i)}$, are calculated by a small neural network named *Attention Layer* (a.k.a., *Alignment Model*).
- As shown in fig. 15, an attention layer comprises:
  - a TimeDistributed Dense Layer
  - a Softmax Layer
- For a decoder time step (e.g., 3), the time-distributed dense layer:
  - takes as input the encoder output and the decoder's previous hidden state (e.g., $\mathbf{h}_{(2)}$)
  - outputs a score (e.g., $e_{(3,2)}$) for each encoder output, measuring how well the encoder output aligns with the decoder's previous hidden state (the higher the better aligned)
- For a decoder time step (e.g., 3), the softmax layer:
  - takes as input the output of the TimeDistributed layer
  - outputs a probability distribution (i.e., the weights $\alpha_{(t,i)}$)
- This particular attention mechanism is called *Bahdanau Attention* (a.k.a., *Concatenative Attention* or *Additive Attention*).

Other Attention Mechanisms

- Inspired by Bahdanau Attention, other attention mechanisms have also been proposed.
- Since the idea of the attention mechanism is measuring the similarity between an encoder output and the decoder's previous hidden state, a more straightforward way (compared to Bahdanau attention) for doing so is calculating the dot product of the two vectors, as:
  - dot product is a fairly good measurement for similarity
  - dot product can be calculated in parallel (so that it is much faster to compute)
- An extension of the above dot product:
  1. first uses a time-distributed dense layer to conduct a linear transformation (i.e., weighted sum) for the encoder outputs
  2. then calculates the dot product of the transformed encoder outputs and decoder's previous hidden state
- This particular mechanism is called *Luong Attention* (a.k.a., *Multiplicative Attention)*.
- Other differences between Luong attention and Bahdanau attention include:
  - Luong attention uses decoder's current hidden state, $\mathbf{h}_{(t)}$, rather than the previous hidden state, $\mathbf{h}_{(t-1)}$, to calculate the similarity score, $e_{(t,i)}$
  - Luong attention uses the output of the attention mechanism, $\widetilde{\mathbf{h}}_{(t)}$, to calculate the decode predictions, rather than decoder's current hidden state

Attention Mechanisms: Math

- Concretely, the similarity score between encoder's i^{th} output $(\mathbf{y}_{(i)})$ and decoder's t^{th} hidden state $(\mathbf{h}_{(t)})$, $e_{(t,i)}$, is calculated as:

$$e_{(t,i)} = \begin{cases} \mathbf{h}_{(t)}^{\intercal} \mathbf{y}_{(i)}, & \text{dot} \\ \mathbf{h}_{(t)}^{\intercal} \mathbf{W} \mathbf{y}_{(i)}, & \text{general} \\ \mathbf{v}^{\intercal} \tanh\left(\mathbf{W}\left[\mathbf{h}_{(t)}, \mathbf{y}_{(i)}\right]\right). & \text{concat} \end{cases} \tag{12}$$

  Here:
  - $\mathbf{W}$ is the weight matrix
  - $\mathbf{v}^{\intercal}$ is a scaling parameter vector
- The weight between encoder's i^{th} output $(\mathbf{y}_{(i)})$ and decoder's t^{th} hidden state $(\mathbf{h}_{(t)})$, $\alpha_{(t,i)}$, is calculated as:

$$\alpha_{(t,i)} = \frac{\exp\left(e_{(t,i)}\right)}{\sum_i e_{(t,i)}}. \tag{13}$$

- The output of the attention mechanism at time step t, $\widetilde{\mathbf{h}}_{(t)}$, is calculated as:

$$\widetilde{\mathbf{h}}_{(t)} = \sum_i \alpha_{(t,i)} \mathbf{y}_{(i)}. \tag{14}$$

Bibliography I

Cho, K., Merriënboer, B. V., Bahdanau, B., and Bengio, Y. (2014).
On the Properties of Neural Machine Translation: Encoder-Decoder Approaches.
arXiv preprint arXiv:1409.1259.

Gers, F. A. and Schmidhuber, J. (2000).
Recurrent Nets that Time and Count.
In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194. IEEE.

Hochreiter, S. and Schmidhuber, J. (1997).
Long Short-Term Memory.
Neural computation, 9(8):1735–1780.