2022

Deep Learning based Text Processing

Lec 12: Sequence to Sequence Model with RNN

hsyi@kisti.re.kr

Hongsuk Yi (이홍석)

Overview of Course (III)

- Introduction to Recurrent Neural Network
  - ✓ Simple RNN, BPTT, Memory Cell
  - ✓ Code: Implementing an RNN with Keras
- Introduction to Long-Short Term Memroy
  - ✓ Cell state, LSTM, and GRU, and Applications
  - ✓ A Visual Guide to Recurrent Layers in Keras
  - ✓ Code: A simple LSTM layers
- Text generation with RNN
  - ✓ Tokenizer, Character-Level Language model
  - ✓ Code: Alice's Adventures in Wonderland
- Sequence to Sequence Learning model with RNN
  - ✓ Introduction to Seq2Seq and Attention model
  - ✓ Code: Character-Level Neural Machine Translation

Reviewing the last class:

Character-level language model

KISTI SINCE 1962

"Modeling word probabilities is really difficult"

	Supervised learning	Sequence modelling
Data	$\{x, y\}_i$	$\{x\}_i$
Model	$y \approx f_\theta(x)$	$p(x) \approx f_\theta(x)$
Loss	$\mathcal{L}(\theta) = \sum_{i=1}^{N} l(f_\theta(x_i), y_i)$	$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(f_\theta(x_i))$
Optimisation	$\theta^* = \arg\min_\theta \mathcal{L}(\theta)$	$\theta^* = \arg\max_\theta \mathcal{L}(\theta)$

Simplest model:
Assume independence of words

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t)$$

p(“modeling”) × p(“word”) × p(“probabilities”) × p(“is”) × p(“really”) × p(“difficult”)

Word	$p(x_i)$
the	0.049
be	0.028
...	...
really	0.0005
...	...

More realistic model:

Assume conditional dependence of words

$$p(x_T) = p(x_T | x_1, \ldots, x_{T-1})$$

Modeling word probabilities is really ?

Context | Target

Target	p(x|context)
difficult	0.01
hard	0.009
fun	0.005
...	...
easy	0.00001

KISTI SINCE 1962

The chain rule

Computing the joint p(**x**) from conditionals

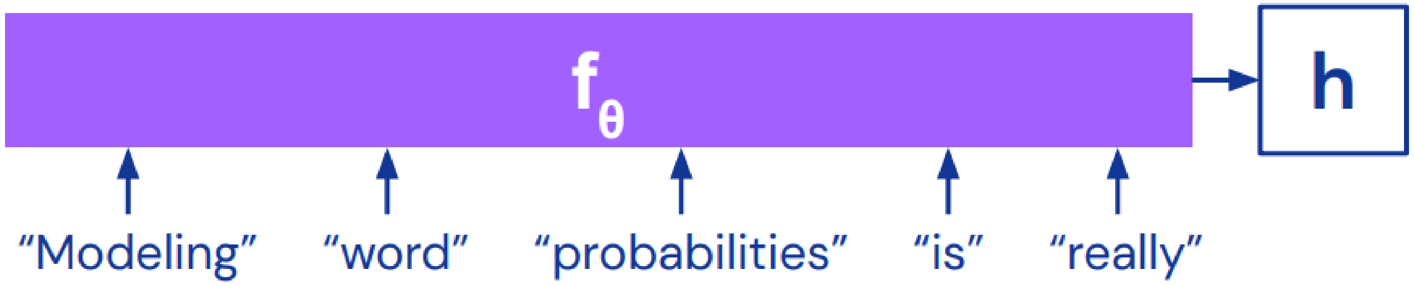
$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_1, ..., x_{t-1})$$

Modeling	$p(x_1)$
Modeling **word**	$p(x_2 \| x_1)$
Modeling word **probabilities**	$p(x_3 \| x_2, x_1)$
Modeling word probabilities **is**	$p(x_4 \| x_3, x_2, x_1)$
Modeling word probabilities is **really**	$p(x_5 \| x_4, x_3, x_2, x_1)$
Modeling word probabilities is really **difficult**	$p(x_6 \| x_5, x_4, x_3, x_2, x_1)$

Recurrent Neural Networks (RNNs)

Learning to model word probabilities

✓ Vectorising the context



f_θ summarises the context in $\boxed{h}$ such that:

$$p(x_t | x_1, ..., x_{t-1}) \approx p(x_t | h)$$

Desirable properties for f_θ:

- Order matters
- Variable length
- Learnable (differentiable)

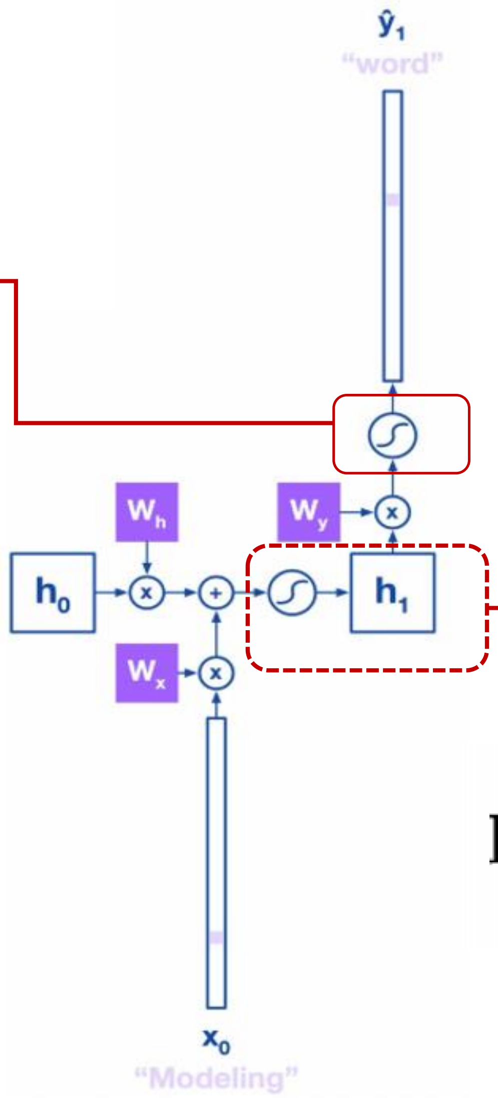
Recurrent Neural Networks (RNNs)

KISTI SINCE 1962

RNNs predict the target **y** (the word) from the state **h.**

$$p(\mathbf{y_{t+1}}) = \boxed{softmax}(\mathbf{W}_y\mathbf{h}_t)$$

Softmax ensures we obtain a distribution over all possible words.



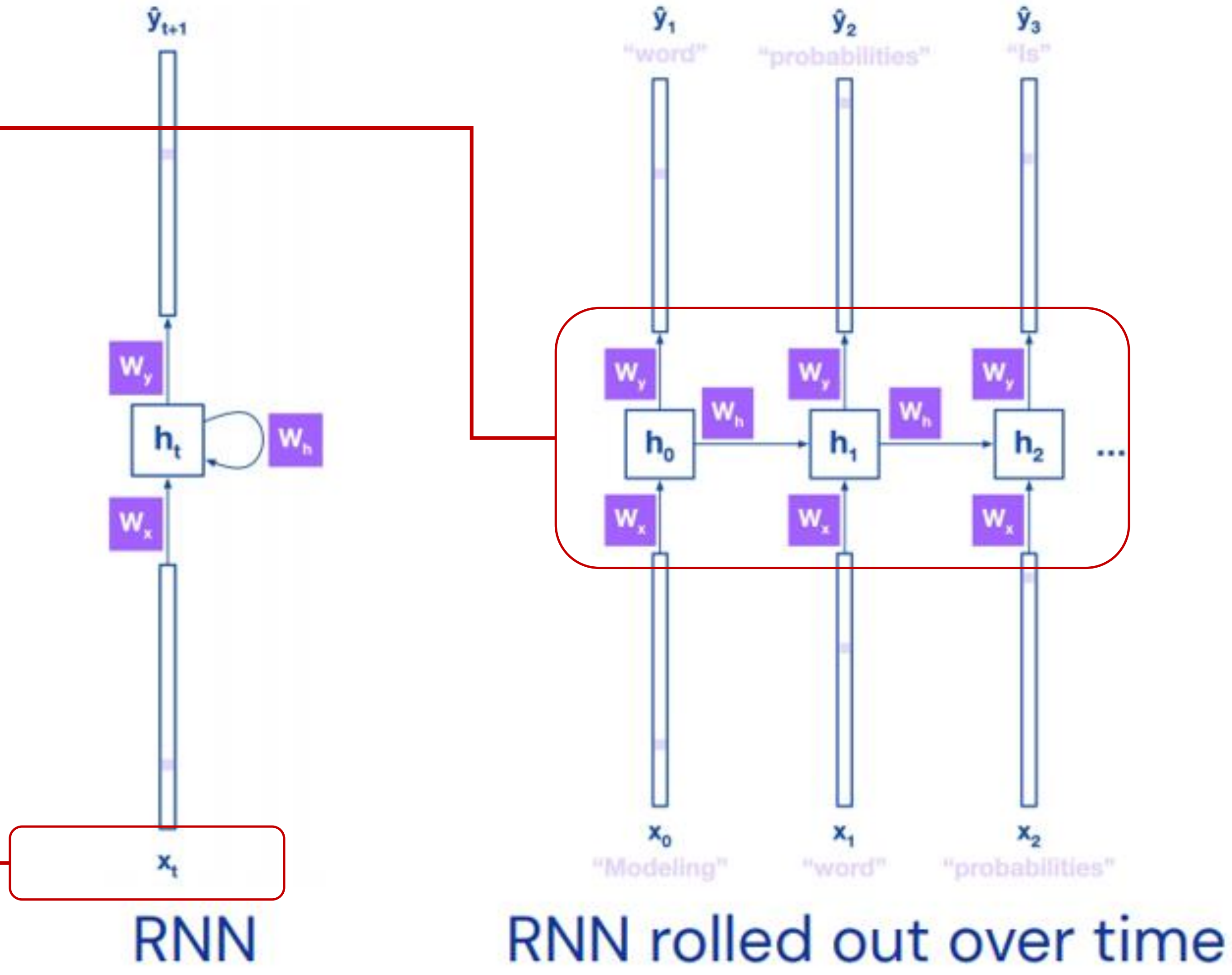
Persistent state variable h stores information from the context observed so far

$$\mathbf{h}_t = \boxed{\tanh}(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t)$$

Recurrent Neural Networks (RNNs)

Weights are shared over time steps

Input next word in sentence x_1



Loss: Cross Entropy

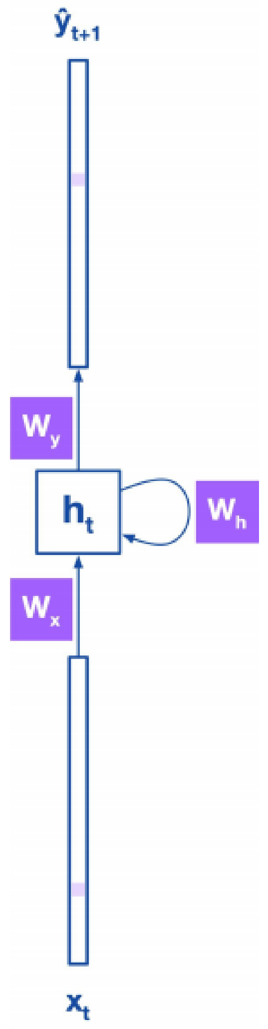
Next word prediction is essentially a classification task where the number of classes is the size of the vocabulary.

As such we use the cross-entropy loss:

For one word: $$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

For the sentence: $$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{t=1}^{T} \mathbf{y}_t \log \hat{\mathbf{y}}_t$$

With parameters $\theta = \{\mathbf{W}_y, \mathbf{W}_x, \mathbf{W}_h\}$



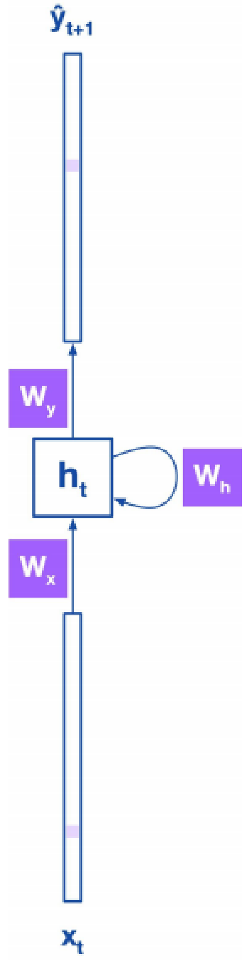
Differentiating weights (w_y, w_x, w_h) from each other

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

$$p(\mathbf{x}_{t+1}) = softmax(\mathbf{W}_y \mathbf{h}_t)$$

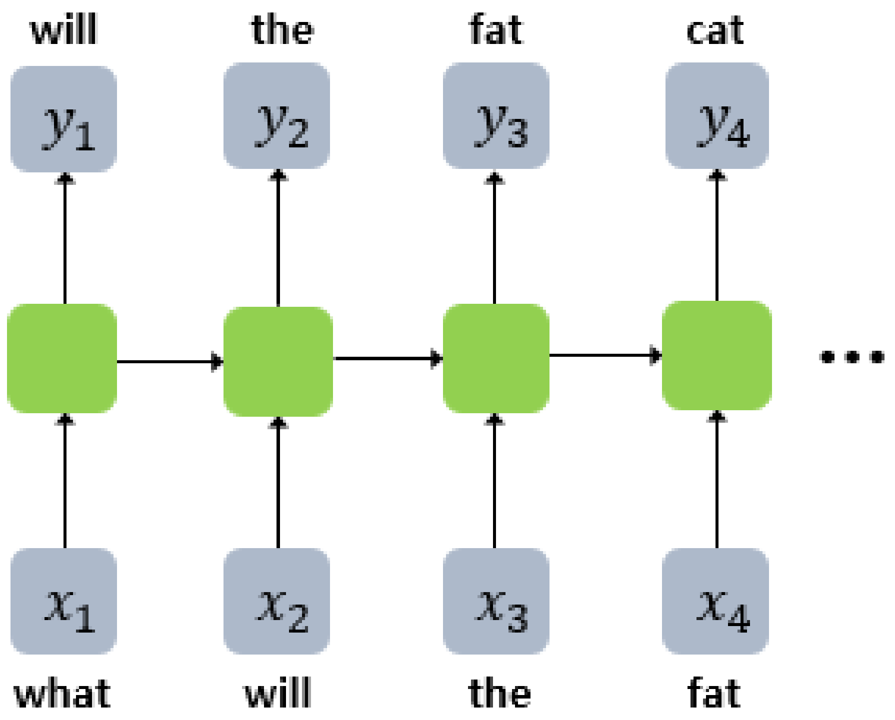
$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

$$\frac{\partial \mathbf{L}}{\partial W} = \sum_{i=0}^{T} \frac{\partial \mathcal{L}_i}{\partial W} \propto \sum_{i=0}^{T} \left(\prod_{i=k+1}^{y} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$



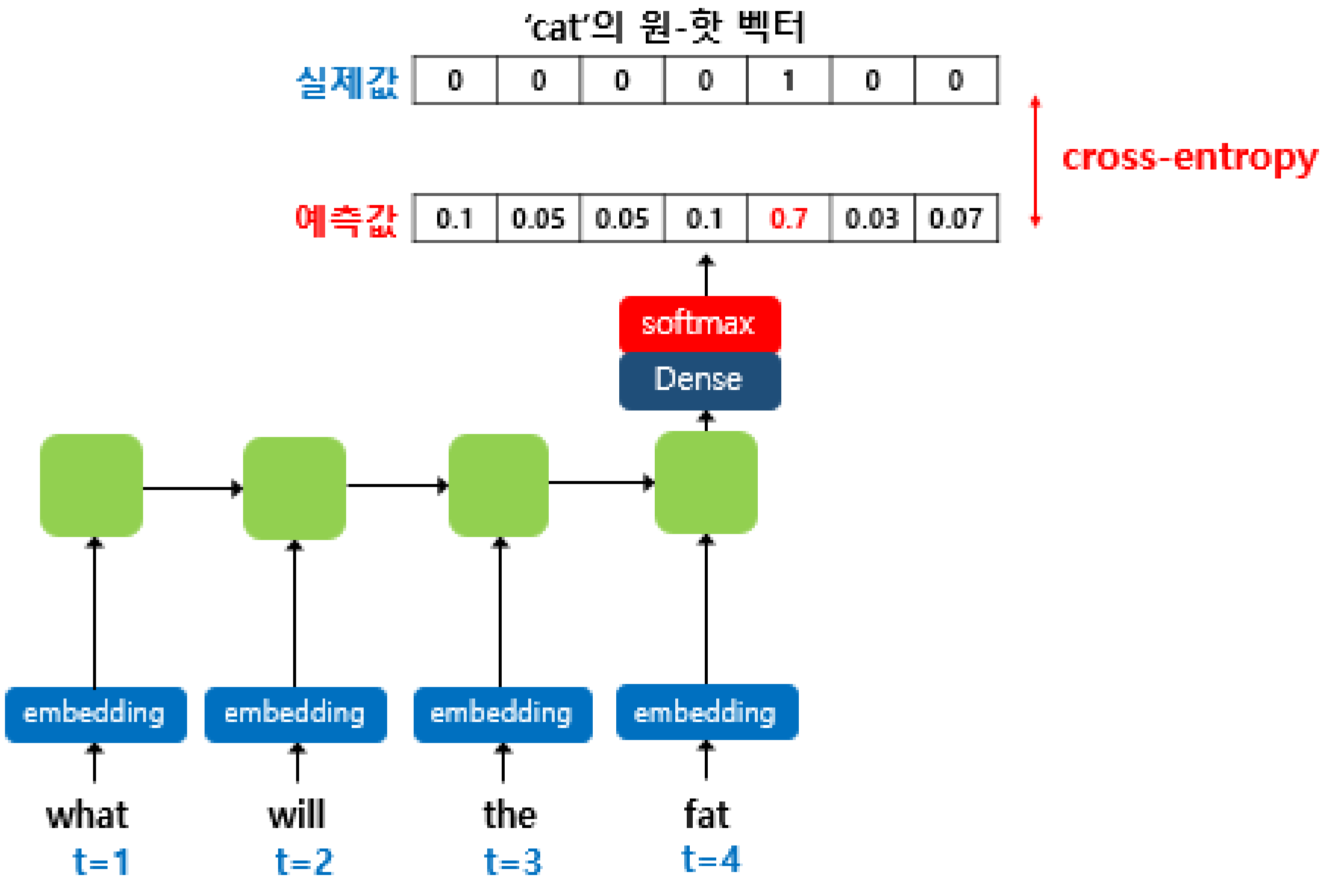
❖ A model that predicts the next word from a word sequence

예문 : 'what will the fat cat sit on'

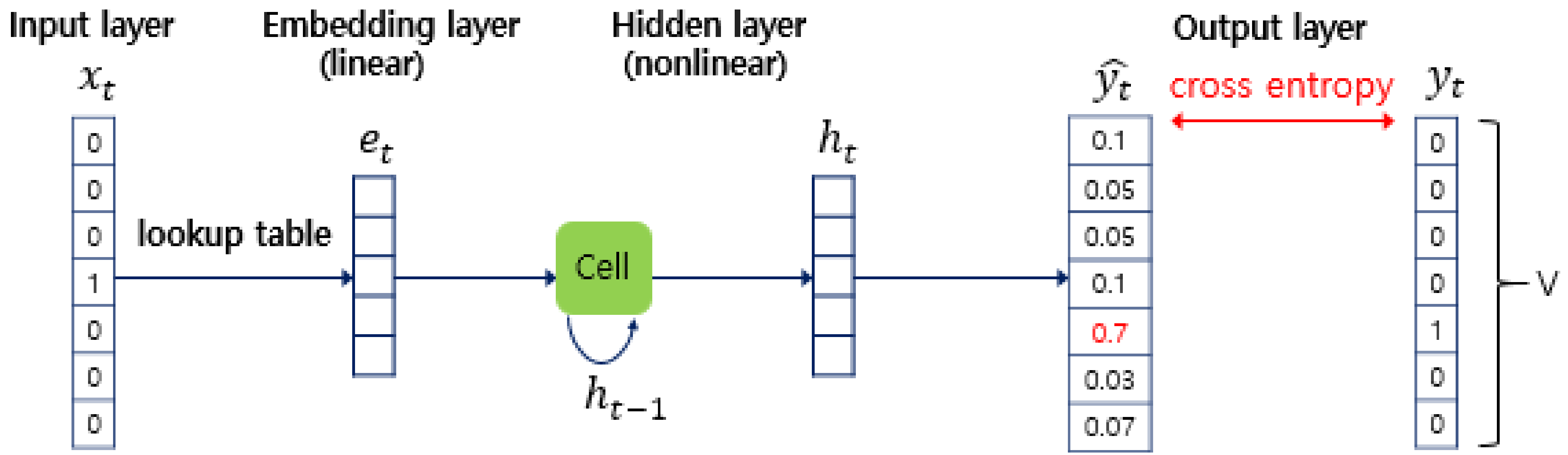


RNNLM: Teacher Forcing Learning

- The model does not use the predicted value at point t as input at point t+1, but uses the label at point t+1 as input at point t+1

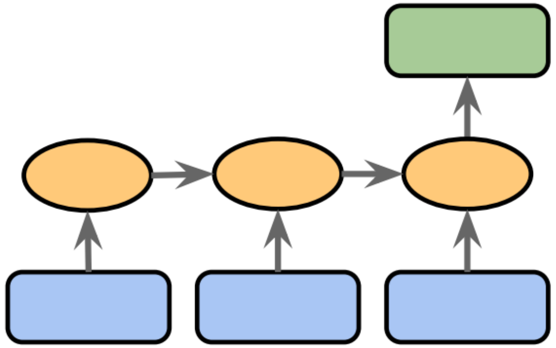


RNNLM Training Process

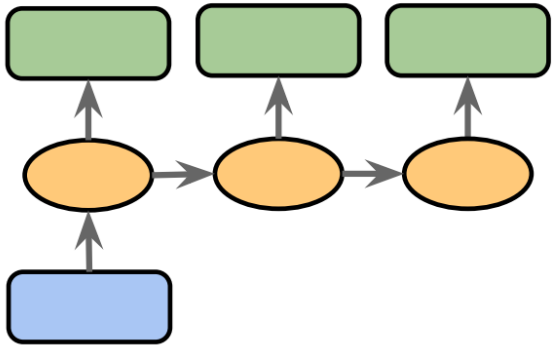


Sequence to Sequence
Neural Machine Translation

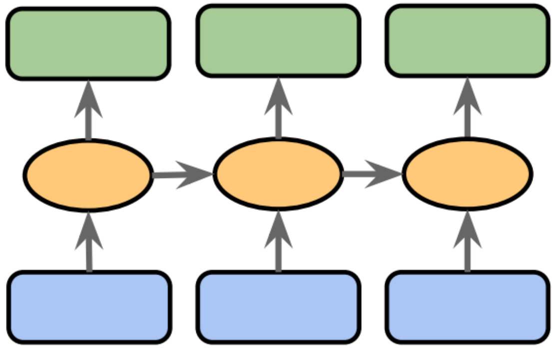
Different Types of Sequence Modeling Tasks



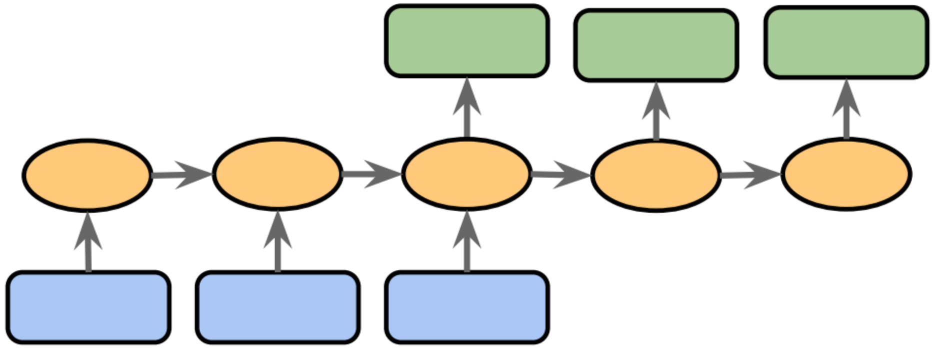
many-to-one



one-to-many



many-to-many

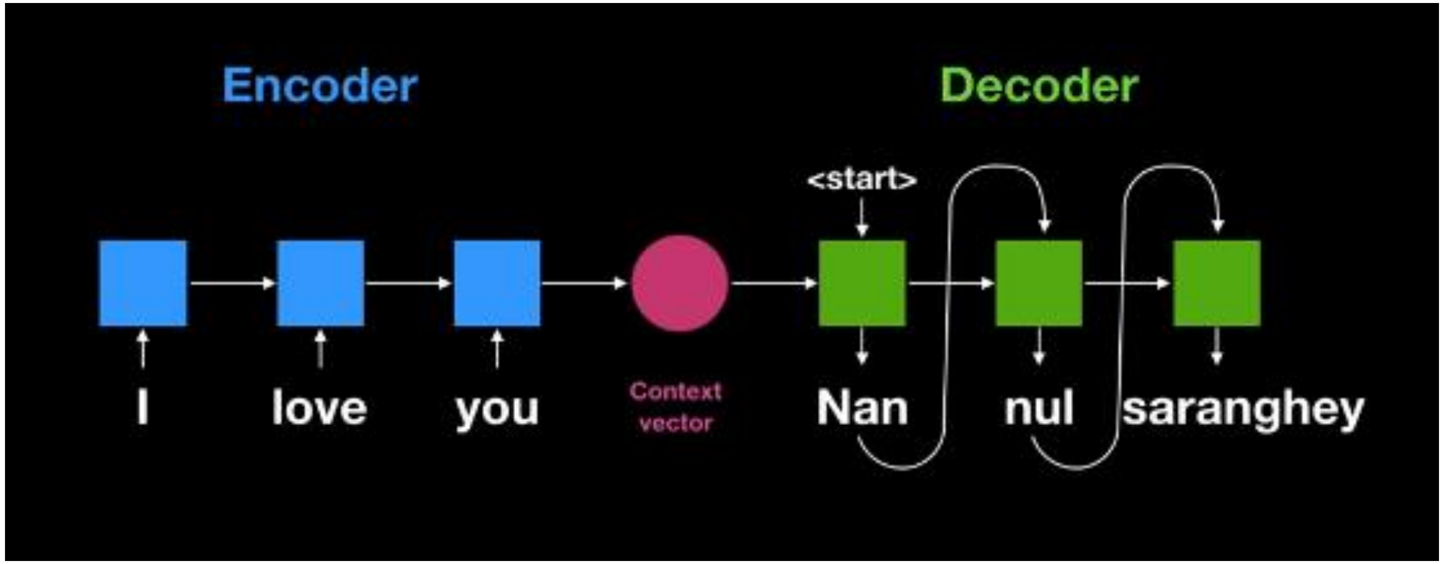


many-to-many

How do I translate the sentence by machine?

- The seq2seq model compresses the input sequence into one fixed-size vector representation, called the context vector, through which the decoder produces the output sequence.

context vector: the final RNN cell states "I love you".



(source) https://www.kaggle.com/code/jeongwonkim10516/attention-mechanism-for-nlp-beginners/notebook

What is a Seq2Seq model?

❖ Sequence-to-sequence learning (Seq2Seq)

- ✓ Seq2Seq is about training models to convert sequences from one domain (e.g. sentences in English) to sequences in another domain (e.g. the same sentences translated to French).

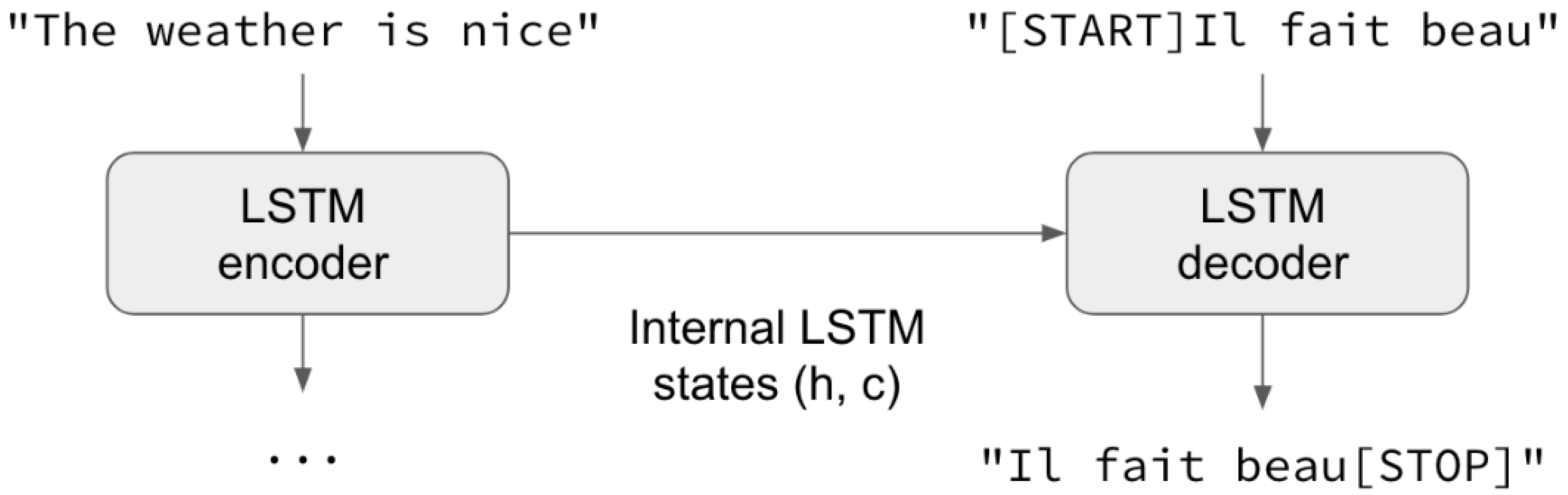
```
"the cat sat on the mat" -> [Seq2Seq model] -> "le chat etait assis sur le tapis"
```

- ✓ This can be used for machine translation or for free-from question answering (generating a natural language answer given a natural language question)
- ✓ in general, it is applicable any time you need to generate text

The general case: canonical sequence-to-sequence

❖ machine translation

✓ input sequences and output sequences have different lengths



How does a classic Seq2Seq model work?

❖ A Seq2Seq model usually consists of:

✓ an **Encoder**

- The **encoder** processes all the inputs by transforming them into a single vector, called **context** (usually with a length of 256, 512, or 1024).

✓ a **Decoder**

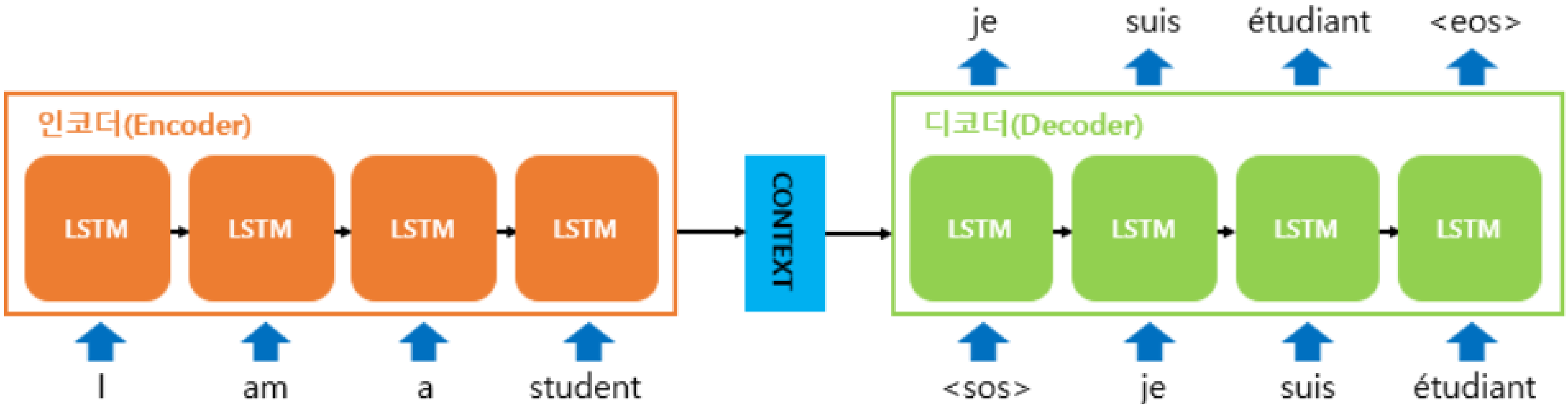
- The context contains all the information that the encoder was able to detect from the input.

✓ a **Context** (*vector*)

- Finally, the vector is sent to the **decoder** which formulates the output sequence.

context vector

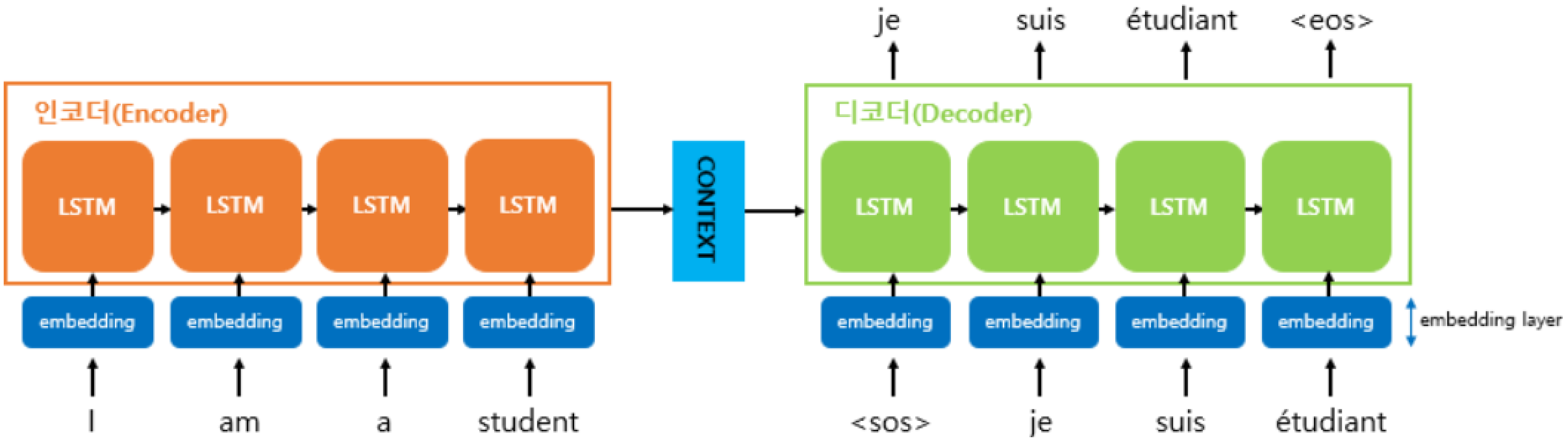
✓ The context vector is the first hidden state of the decoder RNN cell



✓ Decoder is essentially an RNNLM (RNN Language Model)

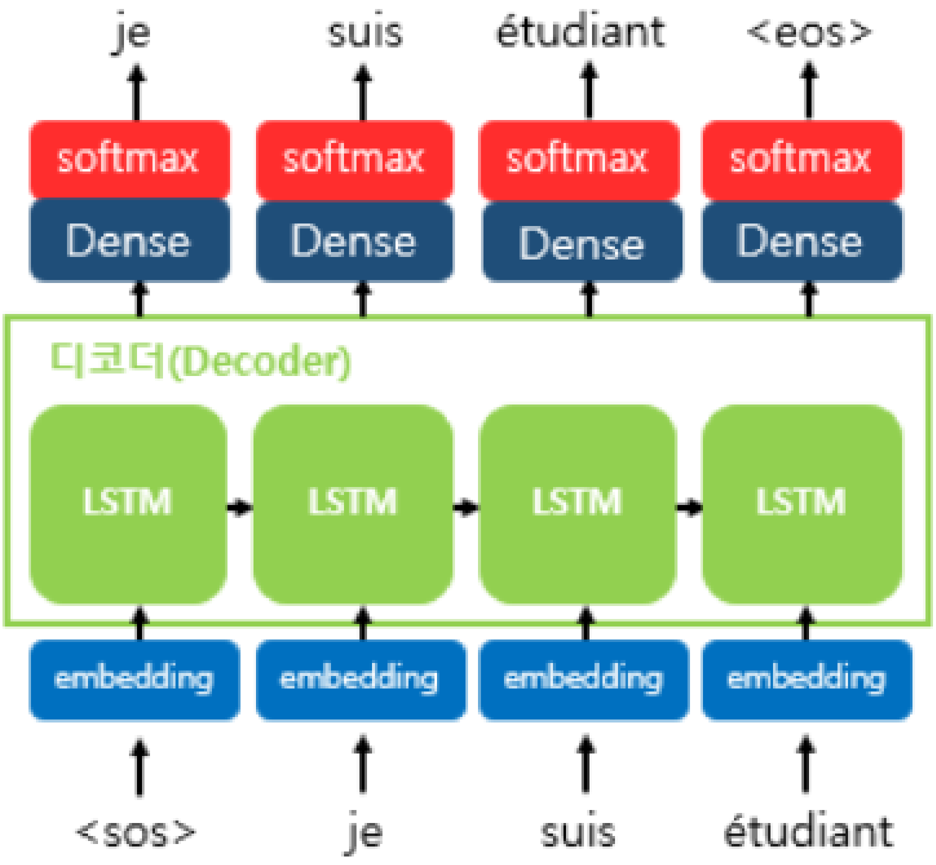
- Many-to-Many

Word embedding and an example of Seq2Seq

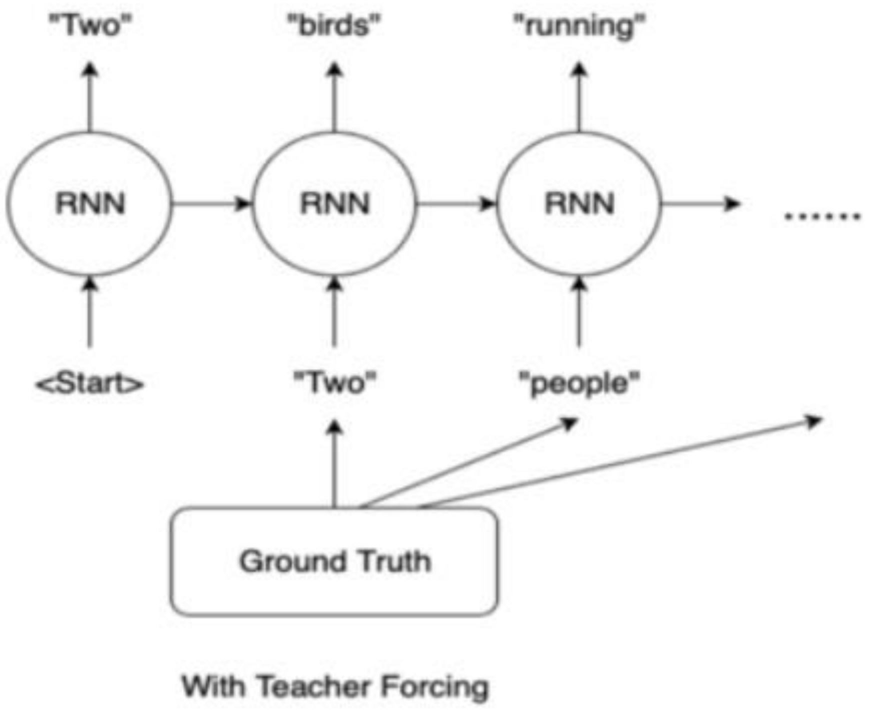


Seq2Seq: Decoder part

❖ Softmax for next prediction word

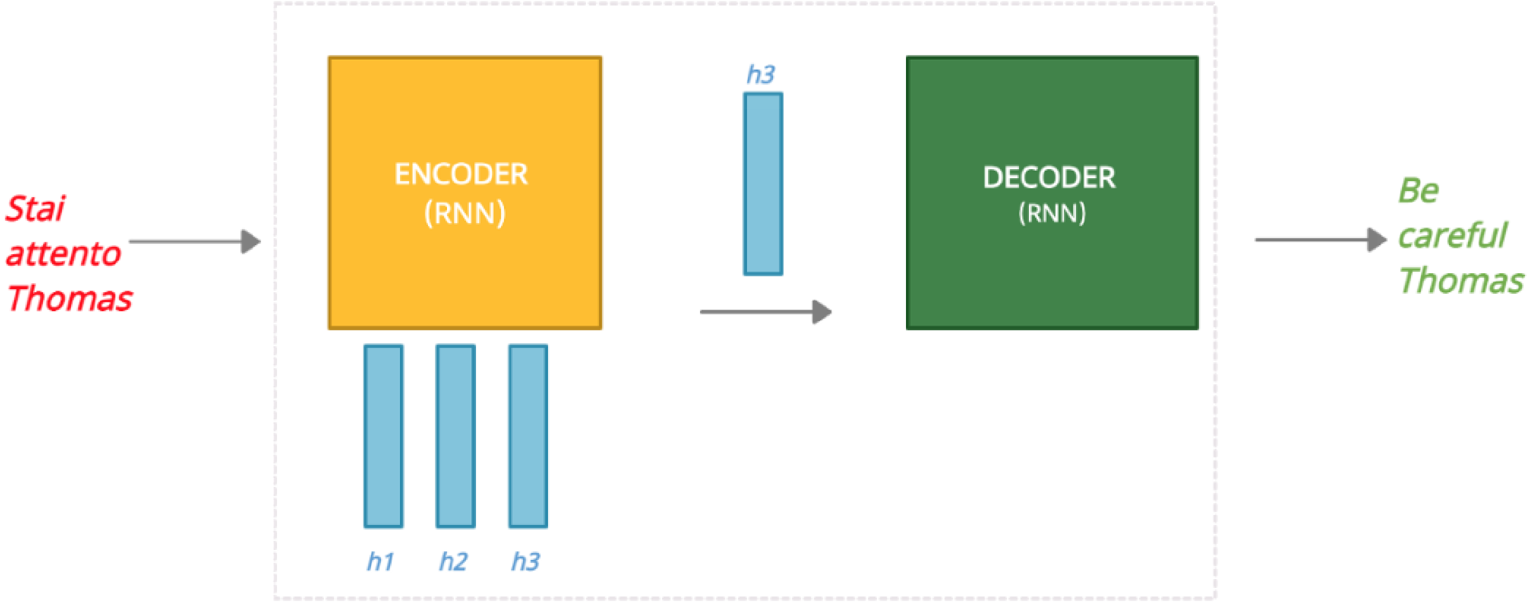


Teacher Forecing Learning



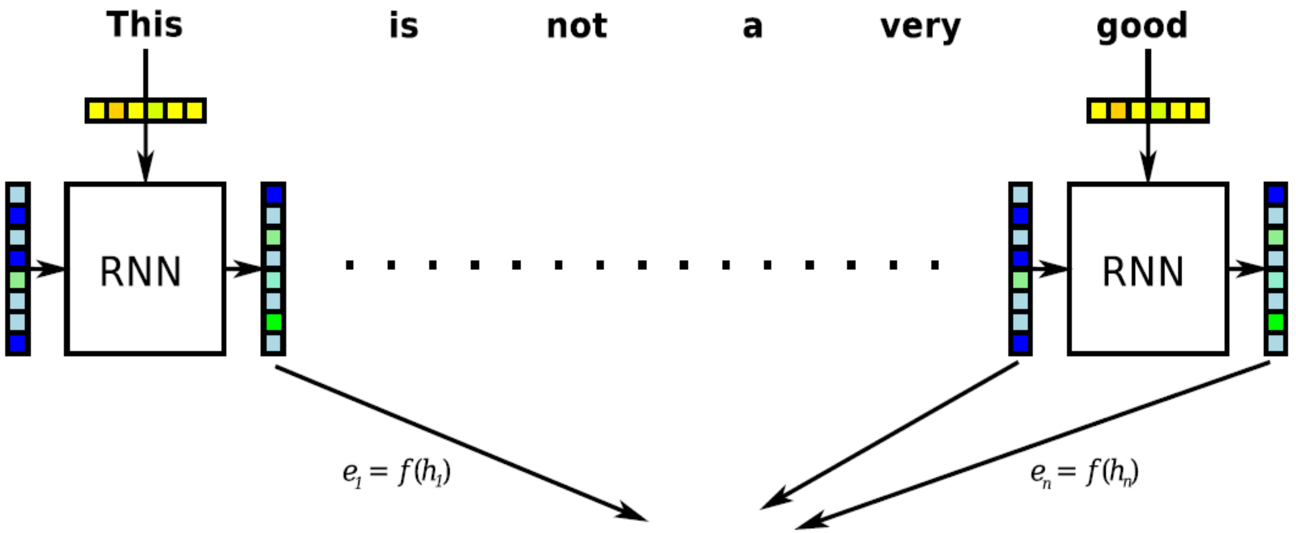
Limitation of Seq2Seq model

- Main problem with seq2seq models
  - ✓ compress all the information into one fixed-size vector results in information loss.
- This is the problem that attention solves!
  - ✓ The last **hidden state (*h3*)** becomes the content that is sent to the decoder
  - ✓ the encoder is “forced” to send only a single vector, regardless of the length of our input



❖ Attention mechanism proposed by Bahdanau et al. (2015)

✓ We compute a "summary" (weighted average) of the states which correspond to some notion of "importance"



$$\alpha_i = \frac{\exp e_i}{\sum_{j=1}^{n} \exp e_j}$$

image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

❖ A general formulation

- ✓ for the attention weights, we apply the softmax
- ✓ the "summary" is computed as a weighted sum

"importance score"

$$\boxed{\alpha_i} = \frac{\exp \boxed{e_i}}{\sum_{j=1}^{n} \exp e_j} \Rightarrow \boxed{s} = \sum_{i=1}^{n} \boxed{\alpha_i \boldsymbol{h}_i}$$

attention weights

"summary"

each RNN state h_i

image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

[HW4] Let's Code!

Character-Level Neural Machine Translation

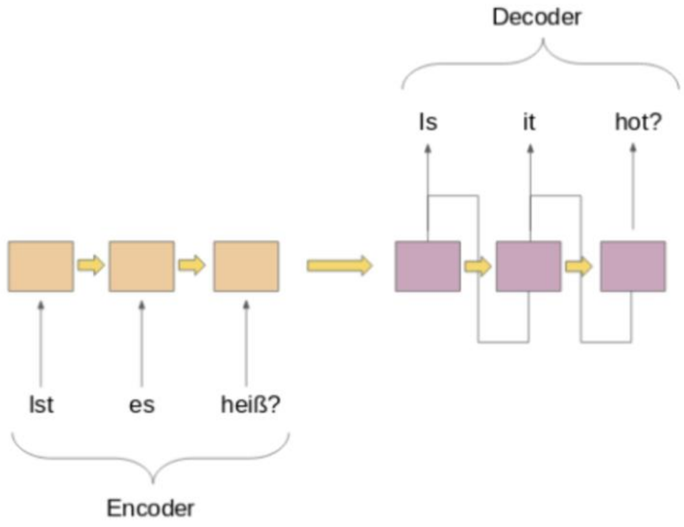
Neural Machine Translation

- Seq2Seq
  - ✓ https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
- Seq2seq may basically have different lengths of the input sequence and the output sequence
- Corpus with two or more languages in parallel
  - ✓ http://www.manythings.org/anki
    - fra-eng.zip
    - kor-eng.zip

Keras, Tensorflow api

In [2]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding, Bidirectional,
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
from tensorflow.keras import optimizers
import matplotlib.pyplot as plt
```



In [5]:
```
data = read_text("kor.txt")
kor_eng = to_lines(data)
kor_eng = np.array(kor_eng)
print(len(kor_eng))
kor_eng.shape
```

3729

Out [5]: (3729, 3)

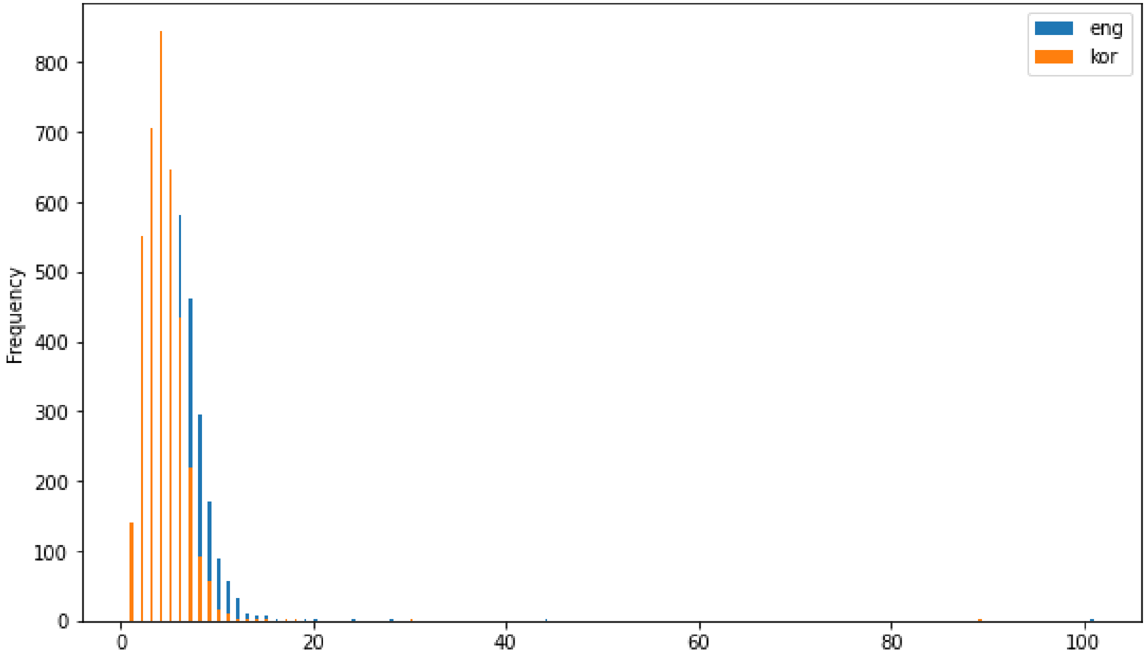
```
kor_eng
```

인 것 같아.',
'CC-BY 2.0 (France) Attribution: tatoeba.org #953635 (CK) & #8384140 (Eunhee)'],

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3729 entries, 0 to 3728
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   eng     3729 non-null   int64
 1   kor     3729 non-null   int64
dtypes: int64(2)
memory usage: 58.4 KB
```

```
df.plot.hist(bins = 300,figsize=(10, 6))
plt.show()
```



Tokenizer

✓ The maximum length of the Korean sentences is 15 and that of the English is 20.

In [15]:
```
# function to build a tokenizer
def tokenization(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

In [16]:
```
# prepare english tokenizer
eng_tokenizer = tokenization(kor_eng[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1

eng_length = 20

print('English Vocabulary Size: %d' % eng_vocab_size)
```

```
English Vocabulary Size: 2561
```

In [18]:

```
# encode and pad sequences
def encode_sequences(tokenizer, length, lines):
    # integer encode sequences
    seq = tokenizer.texts_to_sequences(lines)
    # pad sequences with 0 values
    seq = pad_sequences(seq, maxlen=length, padding='post')
    return seq
```

We will now split the data into train and test set for model training and evaluation, respectively.

In [20]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(kor_eng, test_size=0.2, random_state = 12)
```

```
# build NMT model
def build_model(in_vocab, out_vocab, in_timesteps, out_timesteps, units):
    model = Sequential()
    model.add(Embedding(in_vocab, units, input_length=in_timesteps, mask_zero=True))
    model.add(LSTM(units))
    model.add(RepeatVector(out_timesteps))
    model.add(LSTM(units, return_sequences=True))
    model.add(Dense(out_vocab, activation='softmax'))
    return model
```

Model and Compile

We are using RMSprop optimizer in this model as it is usually a good choice for recurrent neural networks.

```
model = build_model(kor_vocab_size, eng_vocab_size, kor_length, eng_length, 64)
rms = optimizers.RMSprop(learning_rate=0.001)
model.compile(optimizer=rms, loss='sparse_categorical_crossentropy',metrics=['acc'])
```

loss='sparse_categorical_crossentropy': because it allows us to use the target sequence as it is instead of one hot encoded format.

- One hot encoding the target sequences with such a huge vocabulary might consume our system's entire memory.
- We will train it for 30 epochs and with a batch size of 64.

Train the model

```
history = model.fit(trainX, trainY.reshape(trainY.shape[0], trainY.shape[1], 1),
        epochs=30, batch_size=64,
        validation_split = 0.2,   verbose=1)
```

```
38/38 [==============================] - 1s 22ms/step - loss: 1.5617 - acc:
1.8888 - val_acc: 0.7428
```

Let's compare the training loss and the validation loss.

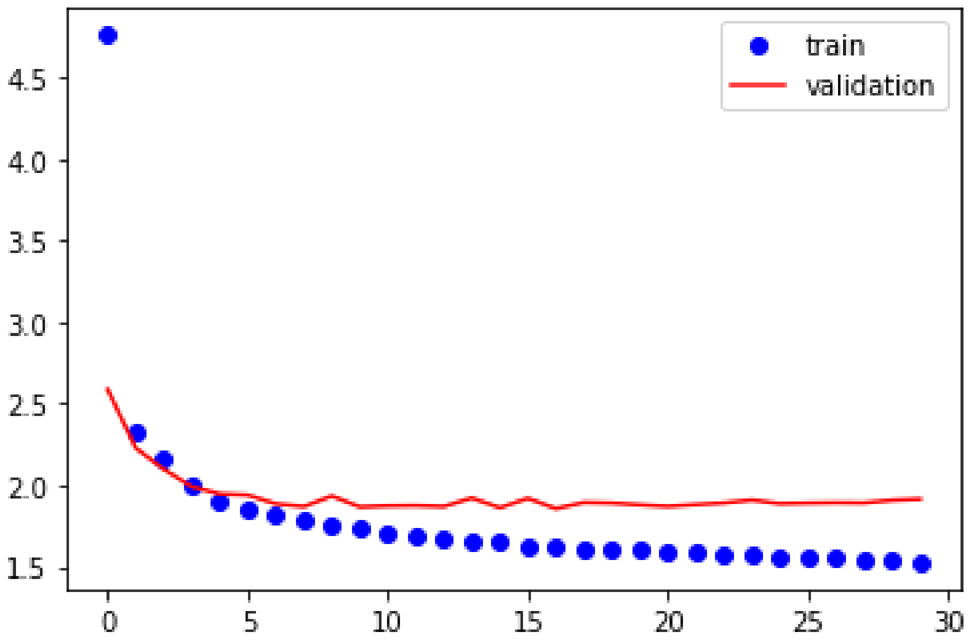
In [28]:
```
history_dict = history.history
history_dict.keys()
```

Out [28]:
```
dict_keys(['loss', 'acc', 'val_loss', 'val_acc'])
```

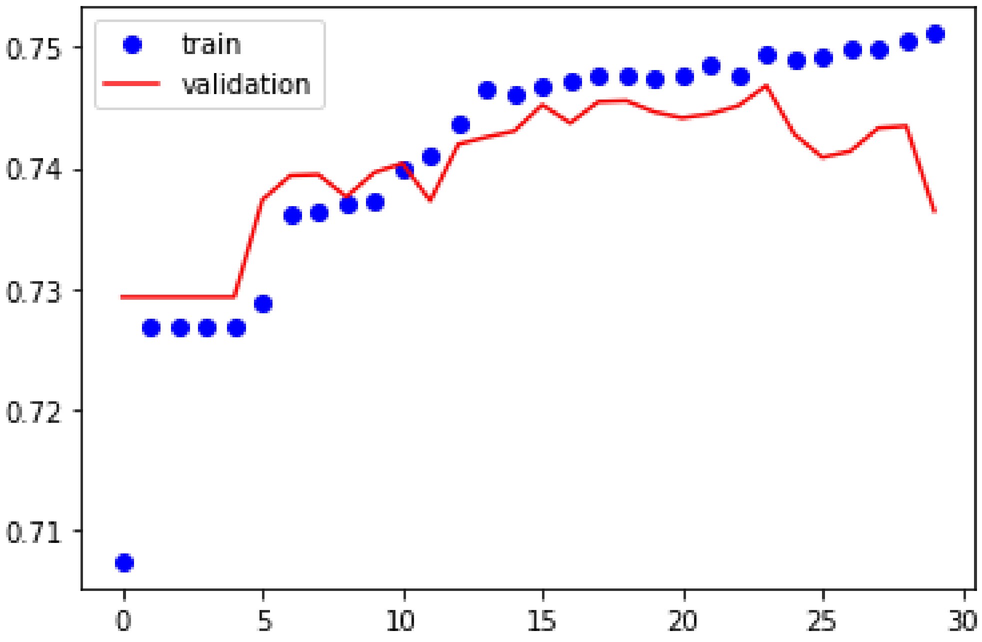
In [30]:
```
plt.plot(history.history['loss'],'bo')
plt.plot(history.history['val_loss'],'r')
plt.legend(['train','validation'])
plt.show()
```

Loss and Accuracy

Loss



Accuracy



2022

Korea Institute of Science and Technology Information

TRUST KISTI