Engaging Content
Engaging People

Neural Machine Translation

Qun Liu, Peyman Passban

ADAPT Centre, Dublin City University

29 January 2018, at DeepHack.Babel, MIPT

Ireland's European Structural and Investment Funds Programmes 2014-2020
Co-funded by the Irish Government and the European Union
European Union
European Regional Development Fund

The ADAPT Centre is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund.

Content

- Background: Machine Translation and Neural Network
- Transition: From Discrete Spaces to Continuous Spaces
- Neural Machine Translation: MT in a Continuous Space
- Implementing Seq2Seq models with PyTorch
- Conclusion

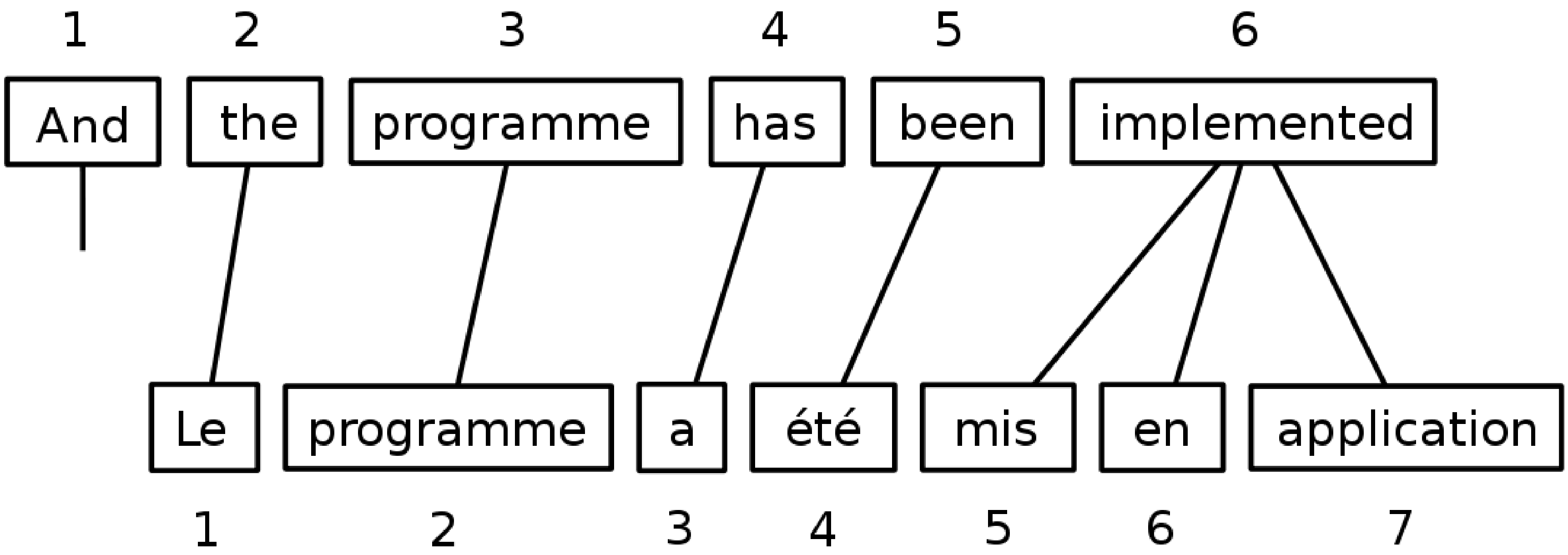
Background: Machine Translation and Neural Network

- Statistical Machine Translation (SMT)
- Deep Learning (DL) and Neural Network (NN)
- The Gap between DL and MT

Parallel Corpus

		facing with the swelling flow of through traffic zooming past their doors .		retahíla de inconvenientes que más y más gente tiene que soportar por el tráfico que pasa por delante_de sus casas , que aumenta a_diario .
5	#77501757	Weekend traffic bans and traffic **jams** are a curse to road transport .	#74765580	Las prohibiciones de conducir los fines de semana y los embotellamientos asolan el transporte por carretera .
6	#79500725	Some people also want to recoup the cost of traffic **jams** from those who get stuck in them , according to the ' polluter pays ' principle .	#76764676	Algunos son partidarios de que incluso los costes ocasionados por los atascos se carguen a el ciudadano que se encuentra atrapado en ellos , de conformidad con el principio de que " quien contamina paga " .
7	#79500765	I think this is an excellent principle and I would like to see it applied in full , but not to traffic **jams** .	#76764713	Me parece un principio acertado y estoy dispuesta a aplicarlo íntegramente , pero no sobre los atascos , ya_que éstos son un claro indicio de el fracaso de la política gubernamental en_materia_de infraestructuras .
8	#79500768	Traffic **jams** are indicative of failed government policy on the infrastructure front , which is why the government itself , certainly in the Netherlands , must be regarded as the polluter .	#76764747	Por eso es preciso subrayar que en estos casos quien contamina es el propio Gobierno , a el menos en los Países_Bajos .
9	#81309716	This would increase traffic **jams** , weaken road safety and increase costs .	#78586130	Esto aumentaría los atascos , mermaría la seguridad vial e incrementaría los costes .
10	#81997391	In the previous legislature , Parliament gave its opinion on the Commission ' s proposals on the simplification of vertical directives on sugar , honey , fruit juices , milk and **jams** .	#79281114	En efecto , durante la precedente legislatura , el Parlamento se manifestó sobre las propuestas de la Comisión relativas a la simplificación de directivas verticales sobre el azúcar , la miel , los zumos de frutas , la leche y las confituras .
11	#81998167	For **jams** , I personally reintroduced an amendment that was not accepted by the Committee on the Environment , Public Health and Consumer Policy , but which I hold to .	#79281936	Para las confituras , yo personalmente volví a introducir una enmienda que no fue aceptada por la Comisión_de_Medio_Ambiente , Salud_Pública y Política_de_el_Consumidor , pero que es importante para mí .
12	#81998209	It concerns not accepting the general use of a chemical flavouring in **jams** and marmalades , that is vanillin .	#79281966	Se trata de no aceptar la utilización generalizada de un aroma químico en las confituras y " marmalades " , a saber , la vainillina .
13	#82800065	This is highlighted particularly in towns where it is necessary to find ways of solving environmental problems and the difficulties caused by traffic **jams** .	#80085988	Esto se pone_de_relieve aún más en las ciudades , en las que hay que encontrar medios para eliminar los inconvenientes derivados de los problemas medioambientales y de la congestión de el tráfico .

Word Alignment



Phrase Table

Source	Target	p(e|f)
den Vorschlag	the proposal	0.6227
den Vorschlag	‘s proposal	0.1068
den Vorschlag	a proposal	0.0341
den Vorschlag	the idea	0.0250
den Vorschlag	this proposal	0.0227
den Vorschlag	proposal	0.0205
den Vorschlag	of the proposal	0.0159
den Vorschlag	the proposals	0.0159

Decoding Process

Maria *no dio una bofetada a la bruja verde*

Build translation left to right

Select a phrase to translate

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

Decoding Process

Maria no dio una bofetada a la bruja verde

Mary

Build translation left to right

- Select a phrase to translate
- Find the translation for the phrase

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

Decoding Process

Maria no dio una bofetada a la bruja verde

Mary

Build translation left to right

- Select a phrase to translate
- Find the translation for the phrase
- Add the phrase to the end of the partial translation

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

Decoding Process

Maria *no dio una bofetada a la bruja verde*

Mary

Build translation left to right
- Select a phrase to translate
- Find the translation for the phrase
- Add the phrase to the end of the partial translation
- Mark words as translated

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

Decoding Process

Maria no dio una bofetada a la bruja verde

Mary did not

One to many translation

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

Decoding Process

Maria no dio una bofetada a la bruja verde

Mary did not slap

Many to one translation

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

Decoding Process

Maria no dio una bofetada a la bruja verde

Mary did not slap the

Many to one translation

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

Decoding Process

Maria no dio una bofetada a la bruja verde

Mary did not slap the green

Reordering

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

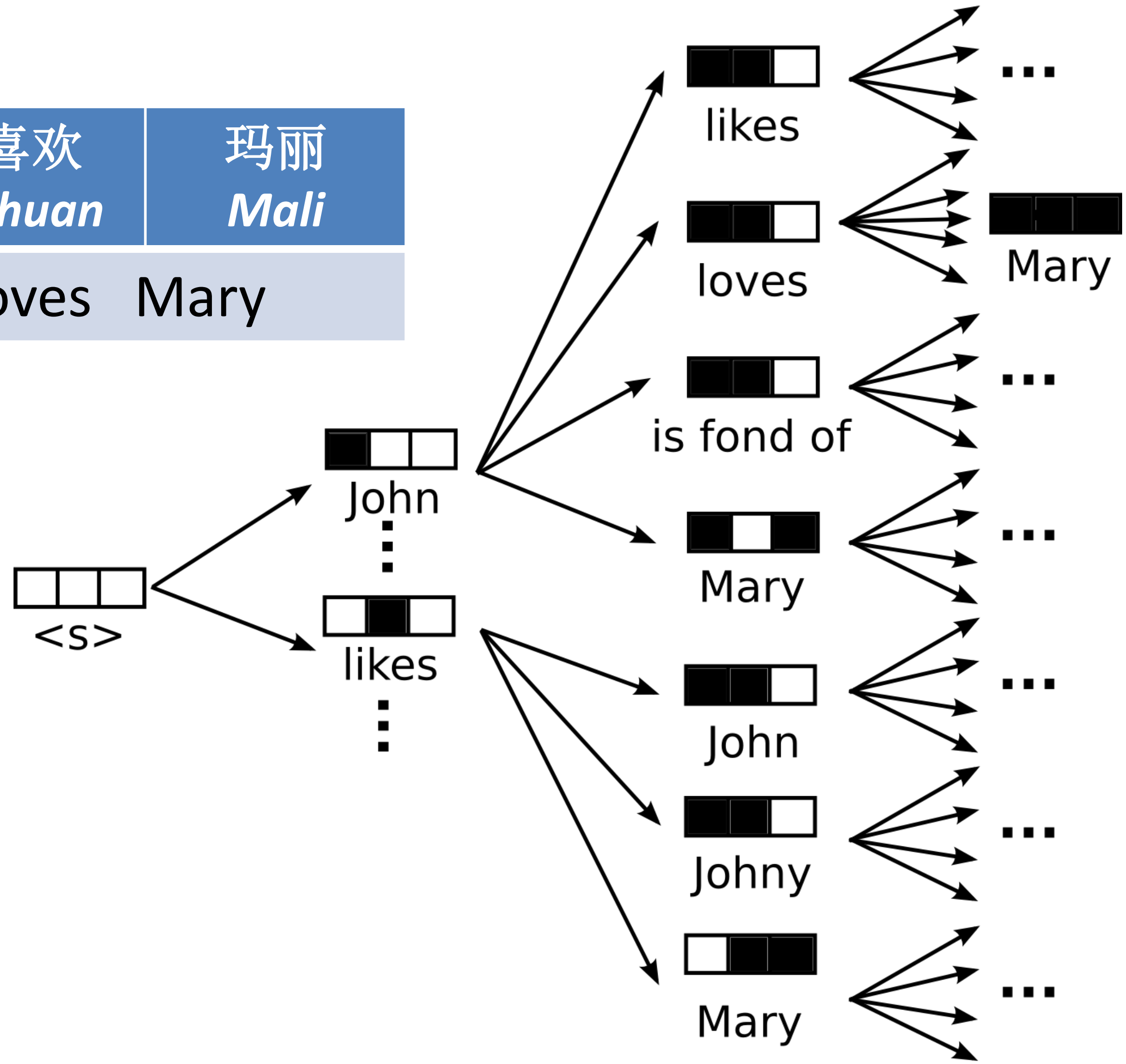
Decoding Process

Translation finished!

Maria	Mary
no	did not
dio una bofetada	slap
a la	the
bruja	witch
verde	green

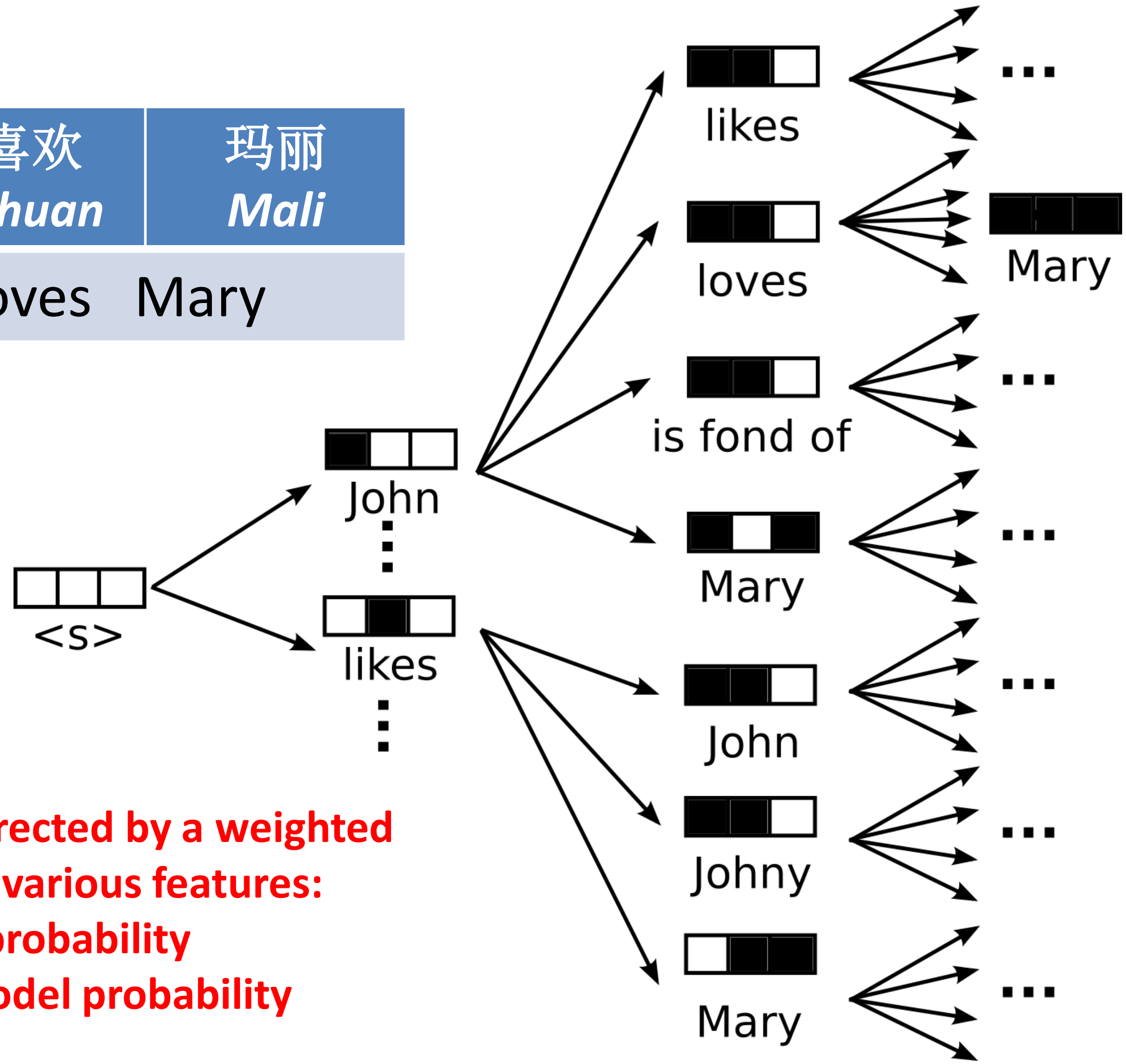
Search Space for Phrase-based SMT

约翰 *Yuehan*	喜欢 *xihuan*	玛丽 *Mali*
John	loves	Mary



Search Space for Phrase-based SMT

约翰 *Yuehan*	喜欢 *xihuan*	玛丽 *Mali*
John	loves	Mary



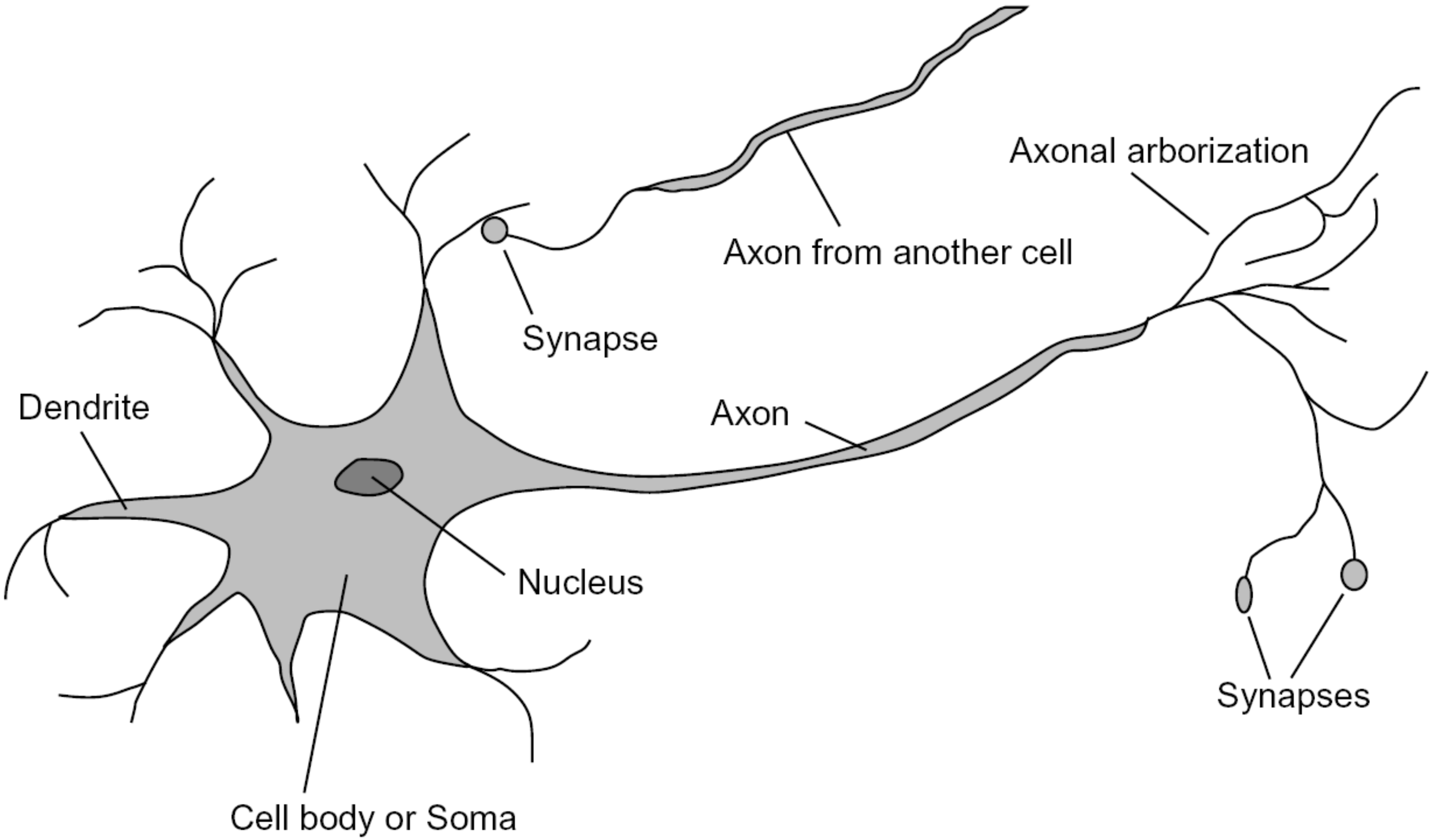
The search is directed by a weighted combination of various features:

- **Translation probability**
- **Language model probability**
- **……**

Background: Machine Translation and Neural Network

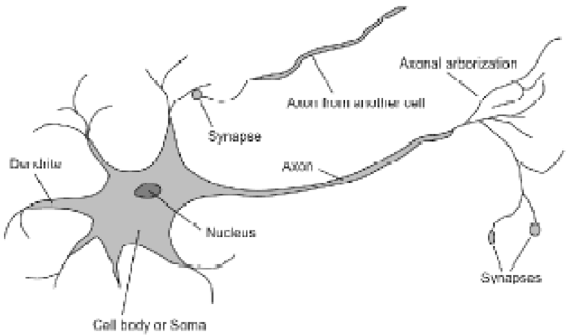
- **Statistical Machine Translation (SMT)**
- **Deep Learning (DL) and Neural Network (NN)**
  - **(slides taken from Kevin Duh's presentation)**
- **The Gap between DL and MT**

Human Neurons - Very Loose Inspiration



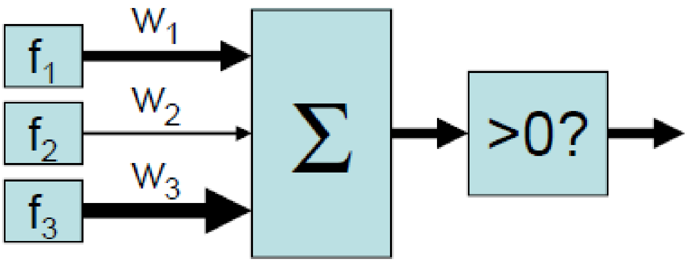
Perceptrons - Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

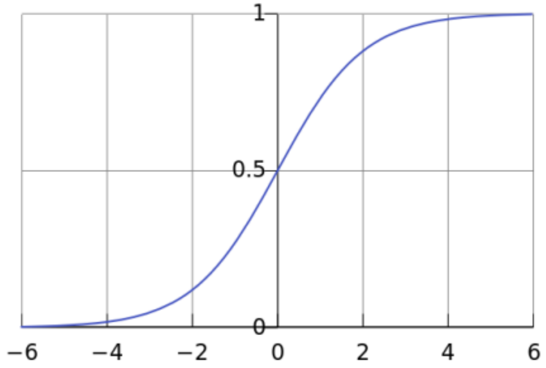
- If the activation is:
  - Positive, output +1
  - Negative, output -1



Logistic Regression (1-layer net)

Function model: $f(x) = \sigma(w^T \cdot x)$

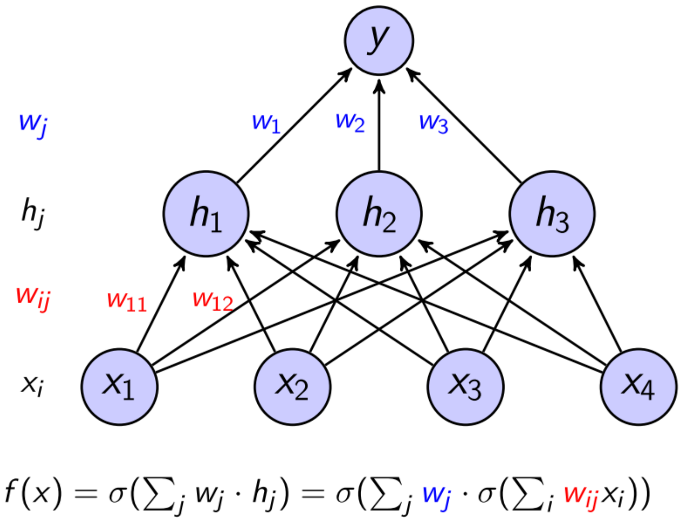
- Parameters: vector $w \in R^d$
- σ is a non-linearity, e.g. sigmoid:
- $\sigma(z) = 1/(1 + exp(-z))$



- Non-linearity will be important in expressiveness
- multi-layer nets. Other non-linearities, e.g.,
- $tanh(z) = (e^z - e^{-z})/(e^z + e^{-z})$

Extracted from Kevin Duh's slides in DL4MT Winter School

2-layer Neural Networks



$$f(x) = \sigma(\textstyle\sum_j w_j \cdot h_j) = \sigma(\textstyle\sum_j w_j \cdot \sigma(\textstyle\sum_i w_{ij} x_i))$$

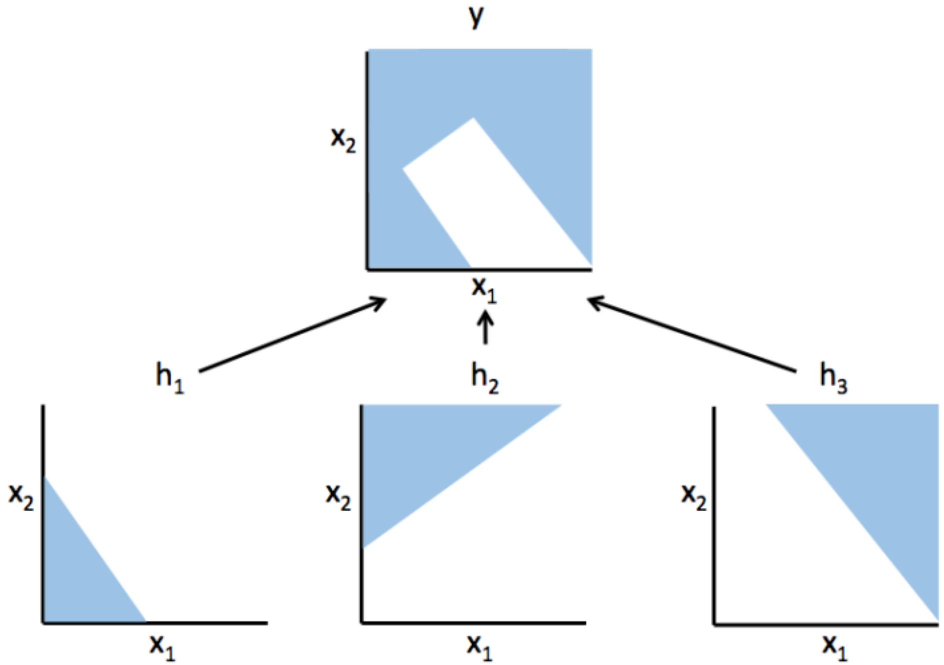
Hidden units h_j's can be viewed as new "features" from combining x_i's

Called Multilayer Perceptron (MLP), but more like multilayer logistic regression

Extracted from Kevin Duh's slides in DL4MT Winter School

Expressive Power of Non-linearity

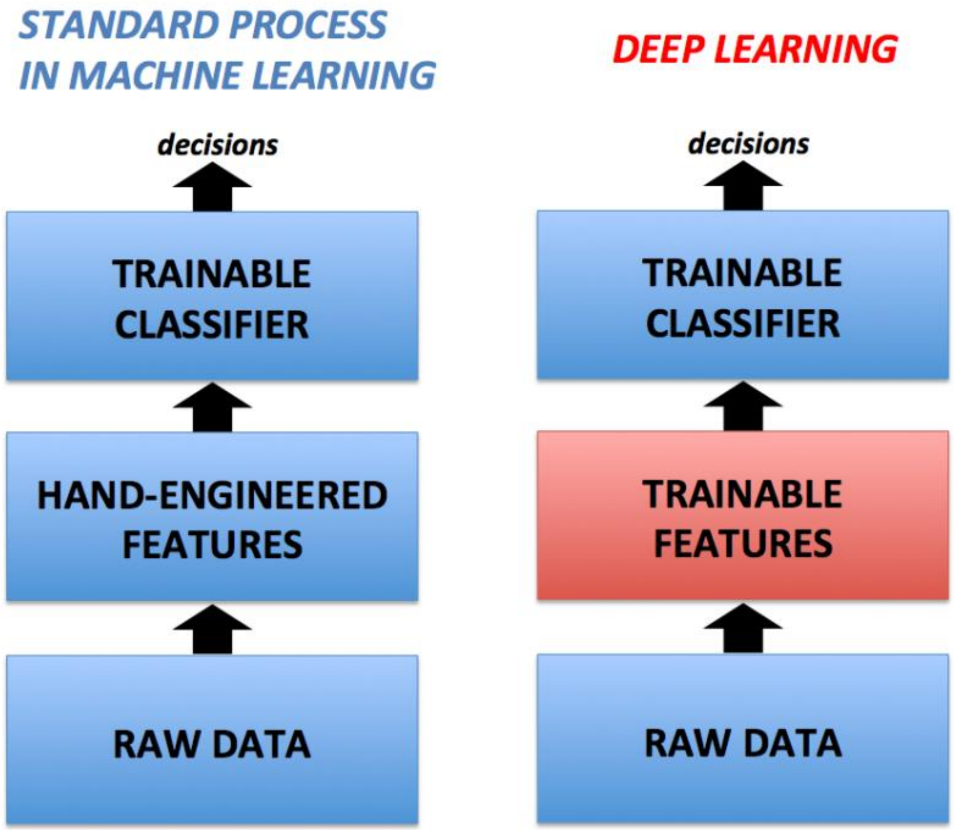
- A deeper architecture is more expressive than a shallow one given same number of nodes [Bishop, 1995]
  - 1-layer nets only model linear hyperplanes
  - 2-layer nets can model any continuous function (given sufficient nodes)
  - >3-layer nets can do so with fewer nodes



Extracted from Kevin Duh's slides in DL4MT Winter School

What is Deep Learning?

A family of methods that uses deep architectures to learn high-level feature representations



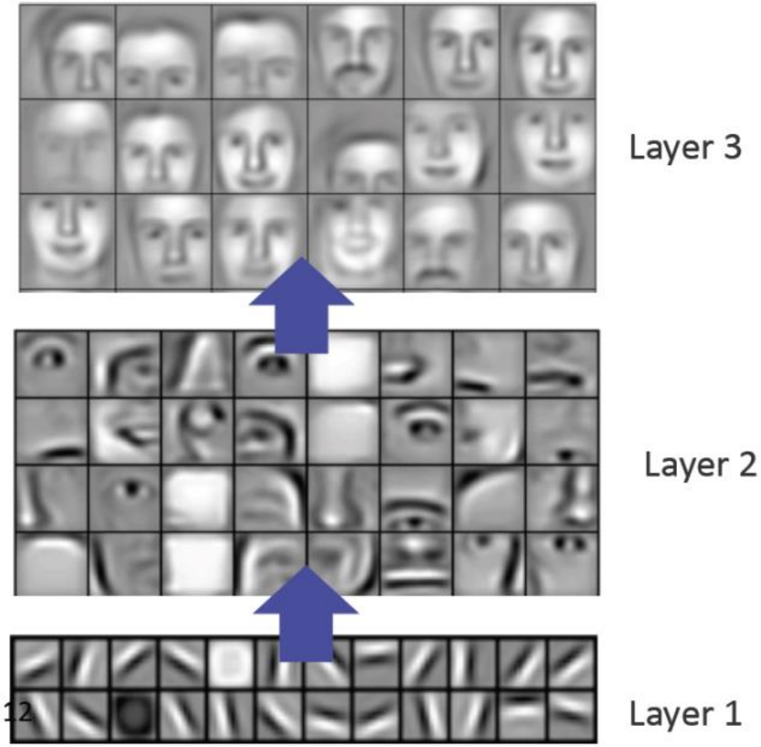
Extracted from Kevin Duh's slides in DL4MT Winter School

Automatically Trained Features in FR

Automatically trained features make sense! [Lee et al., 2009]

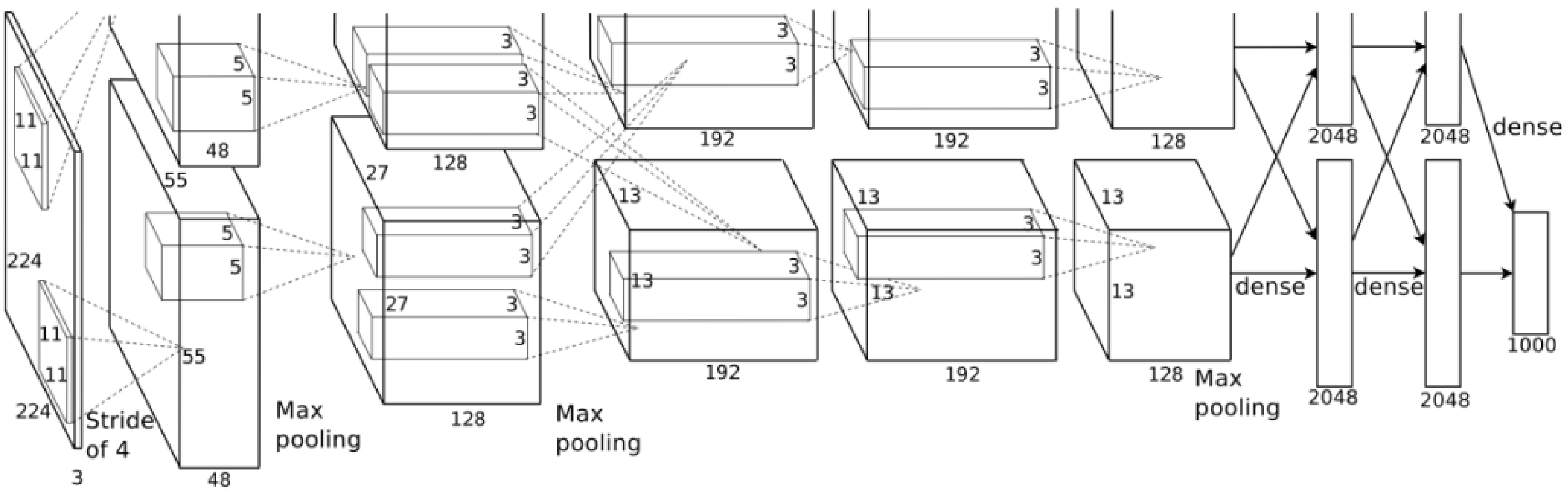
Input: Images (raw pixels)

→ Output: Features of Edges, Body Parts, Full Faces



Extracted from Kevin Duh's slides in DL4MT Winter School

Current models are becoming more complex



- AlexNet for image classification [Krizhevsky et al., 2012]

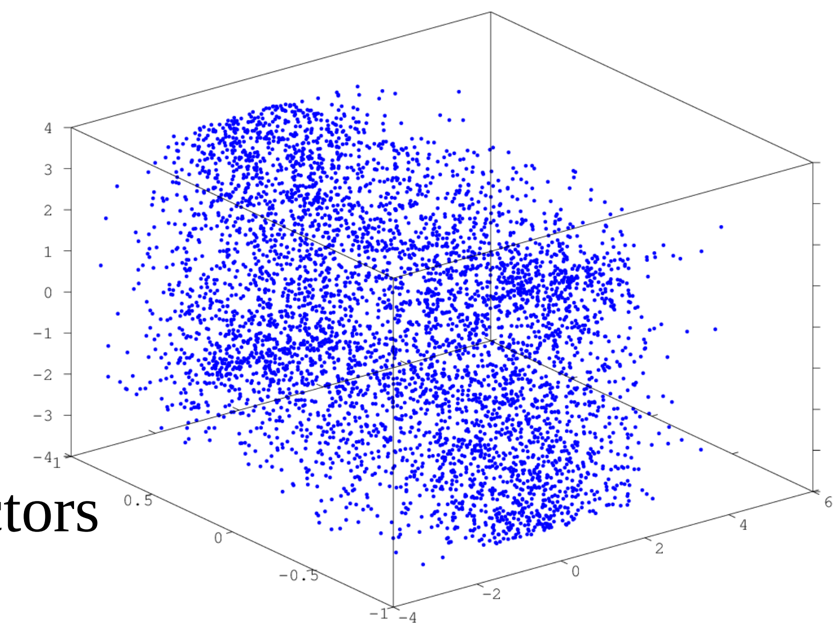
Extracted from Kevin Duh's slides in DL4MT Winter School

Background: Machine Translation and Neural Network

- **Statistical Machine Translation (SMT)**
- **Deep Learning (DL) and Neural Network (NN)**
- **The Gap between DL and MT**

The Gap between DL and MT

Discrete symbols



Continuous vectors

Content

- **Background: Machine Translation and Neural Network**
- **Transition: From Discrete Spaces to Continuous Spaces**
- **Neural Machine Translation: MT in a Continuous Space**
- **Implementing Seq2Seq models with PyTorch**
- **Conclusion**

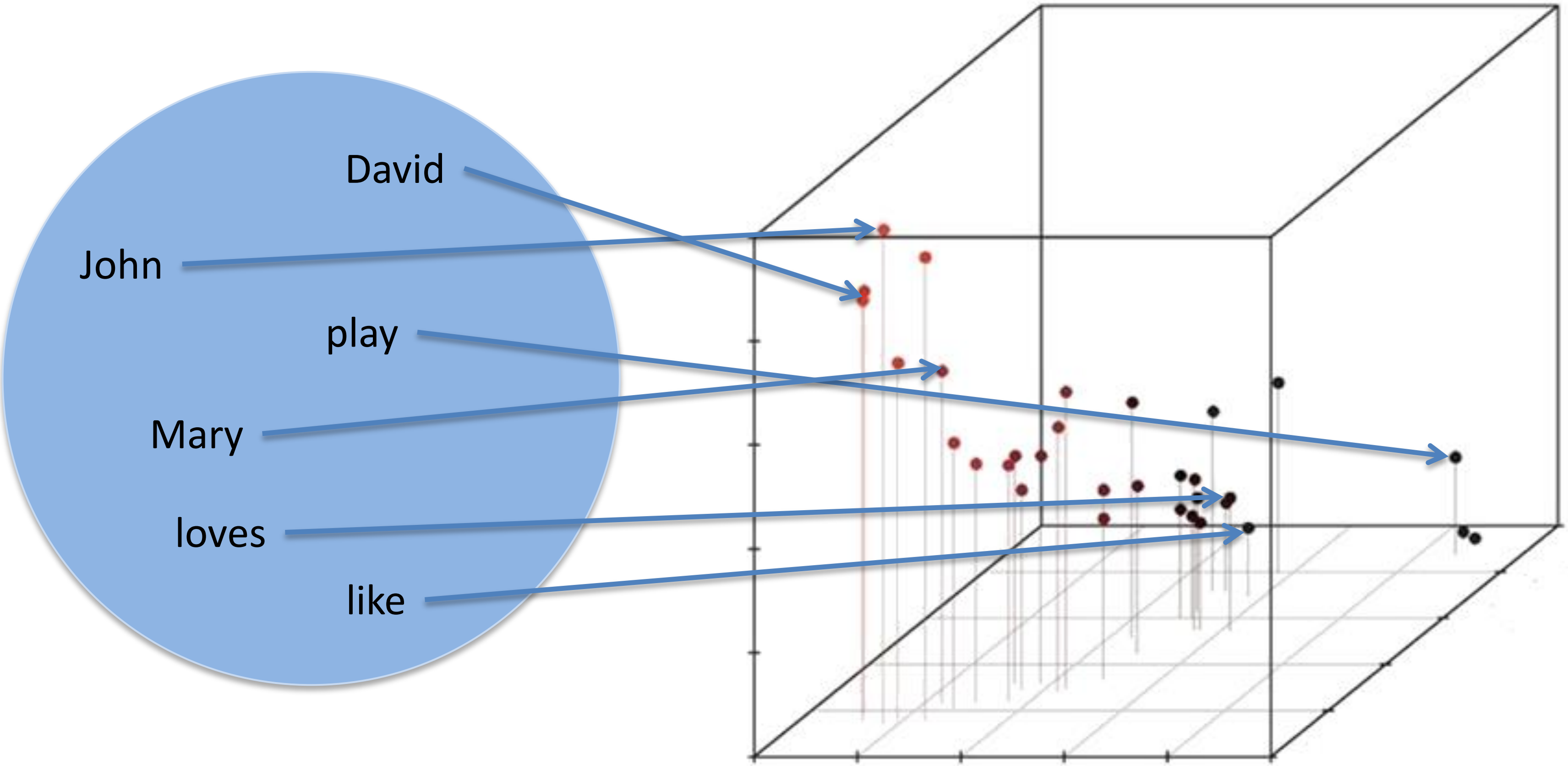
Transition From Discrete Space to Continuous Space

- **Word Embedding**

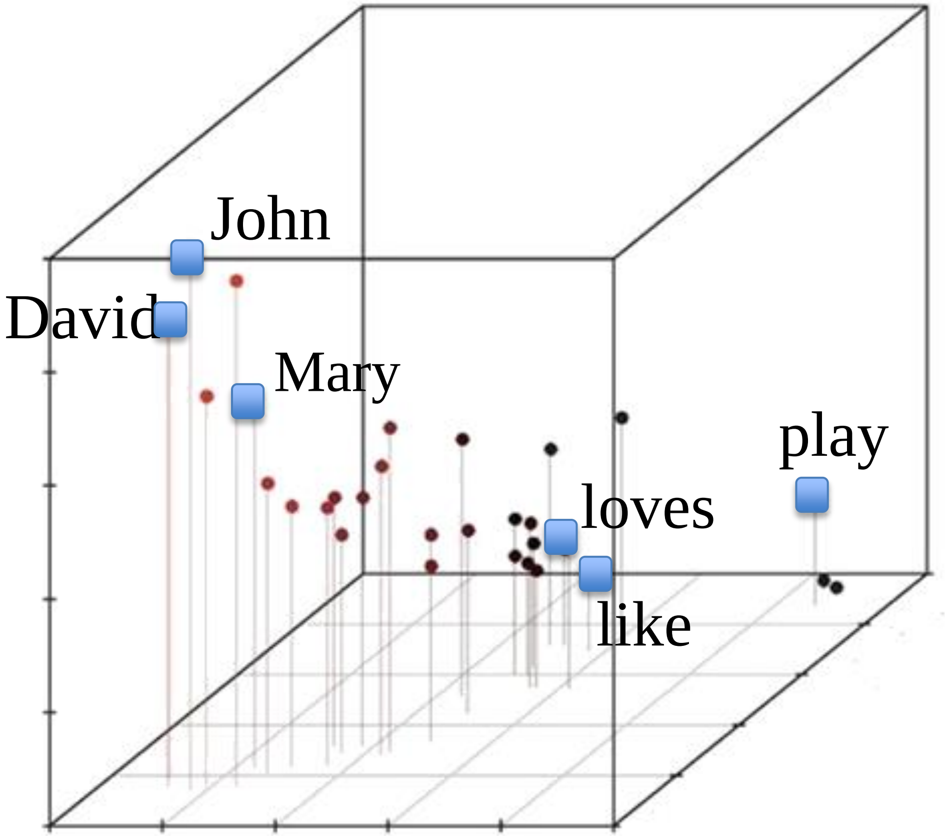
    ➔ **Express a word in a continuous space**

- **Neural Language Model**

Express a word in a continuous space

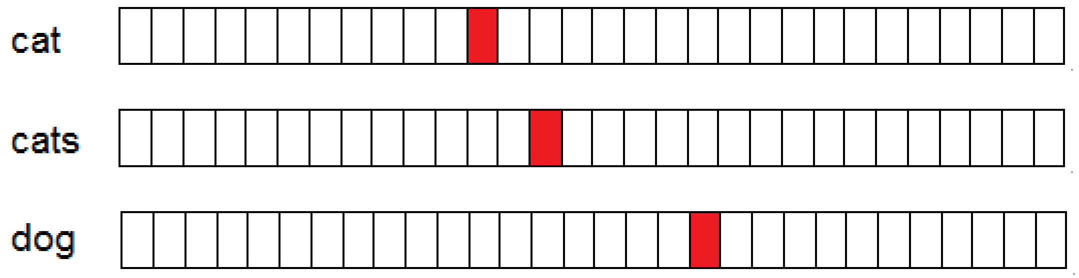


Express a word in a continuous space



One-Hot Vector

- The dimension of the vector is the vocabulary size
- Each dimension is correspondent to a word
- Each word is represented as a vector that:
  - the element is equal to 1 at the dimension which is correspondent to that word
  - All the other elements are equal to 0



One-Hot Vector: Weakness

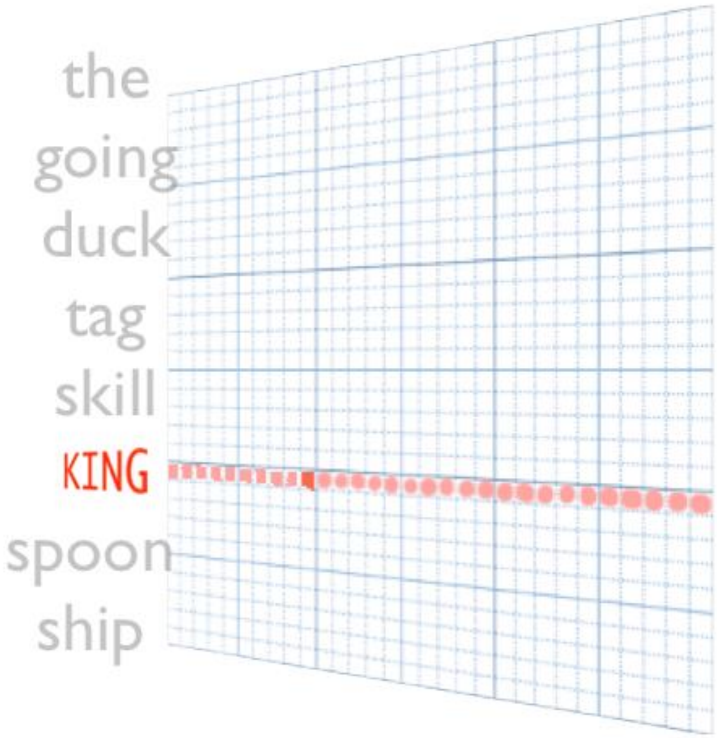
- The dimension is very high (equal to the vocabulary size / ≈100k)
- Very little information is carried by a one-hot vector
  - No syntactic information
  - No semantic information
  - No lexical information

Distributional Semantic Models

- Assumption: Words that are used and occur in the same contexts tend to purport similar meanings
- A typical model: Context Window:
  - A word is represented as the sum/average/tf-idf of the one-hot vectors appearing in the windows surrounding its every occurrence in the corpus
  - Effective for word similarity measurement
  - LSA can be used to reduce the dimension
- Weakness
  - Not compositional
  - Reverse Mapping is not supported

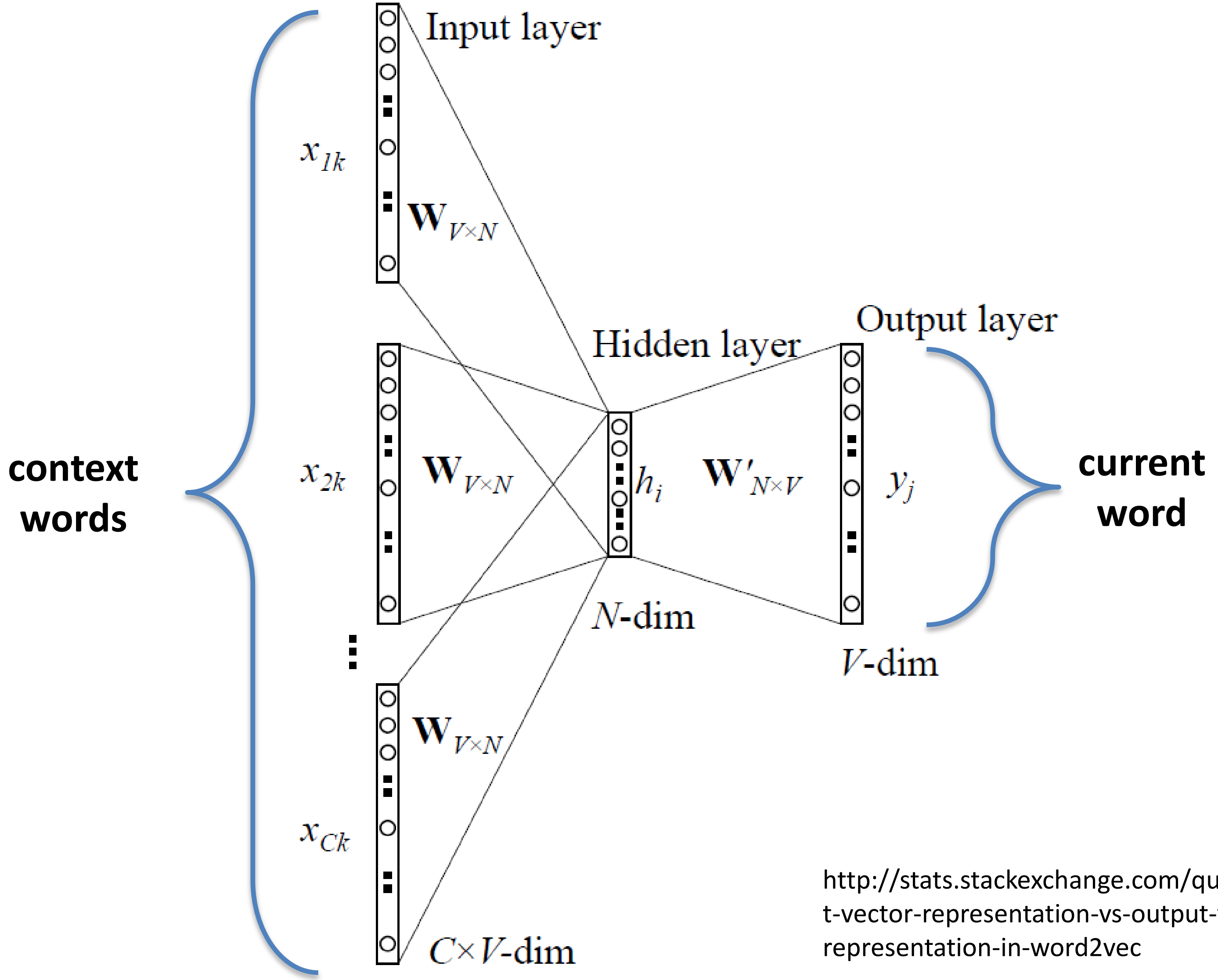
Word2Vec: Word Embedding by Neural Networks

- A word is represented by a dense vector (usually several hundreds dimensions)
- The Word2Vec matrix are trained by a 2-layer neural network



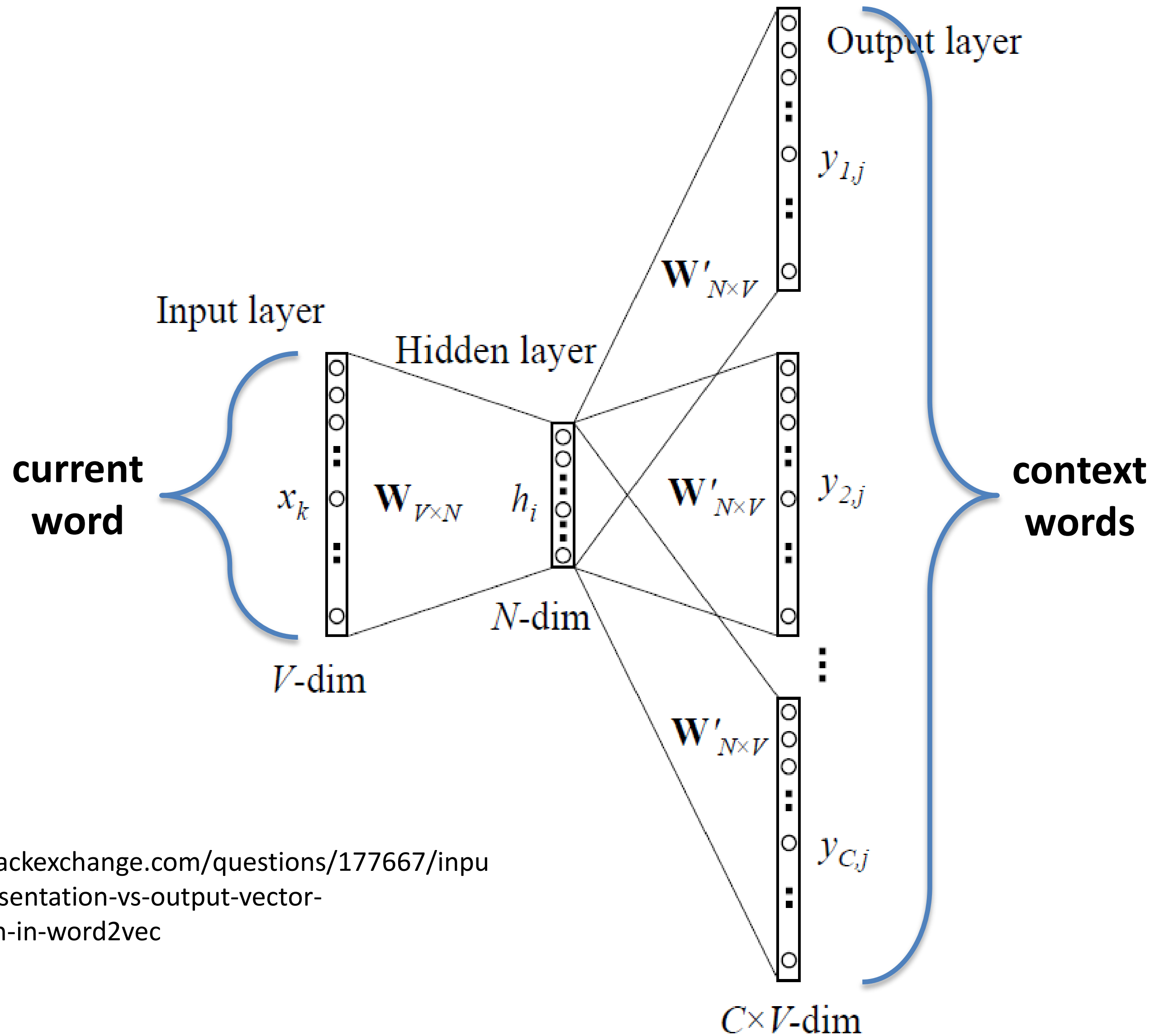
Extracted from Christopher Moody's slides

Word2Vec: CBOW



http://stats.stackexchange.com/questions/177667/input-vector-representation-vs-output-vector-representation-in-word2vec

Word2Vec: Skip-gram



http://stats.stackexchange.com/questions/177667/input-vector-representation-vs-output-vector-representation-in-word2vec

Transition From Discrete Space to Continuous Space

- **Word Embedding**
  - ➔**Express a word in a continuous space**
- **Neural Language Model**
  - ➔**Express a sentence in a continuous space**

Language Models

- Given a sentence: $w_1 w_2 w_3 \dots w_n$, a language model is:

$$p(w_i | \, w_1 \dots w_{i-1})$$

- N-gram Language Model:

$$p(w_i | \, w_1 \dots w_{i-1}) \approx p(w_i | \, w_{i-N+1} \dots w_{i-1})$$

Markov Chain Assumption

N-Gram Model

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

A part of the parameter matrix of a bigram language model

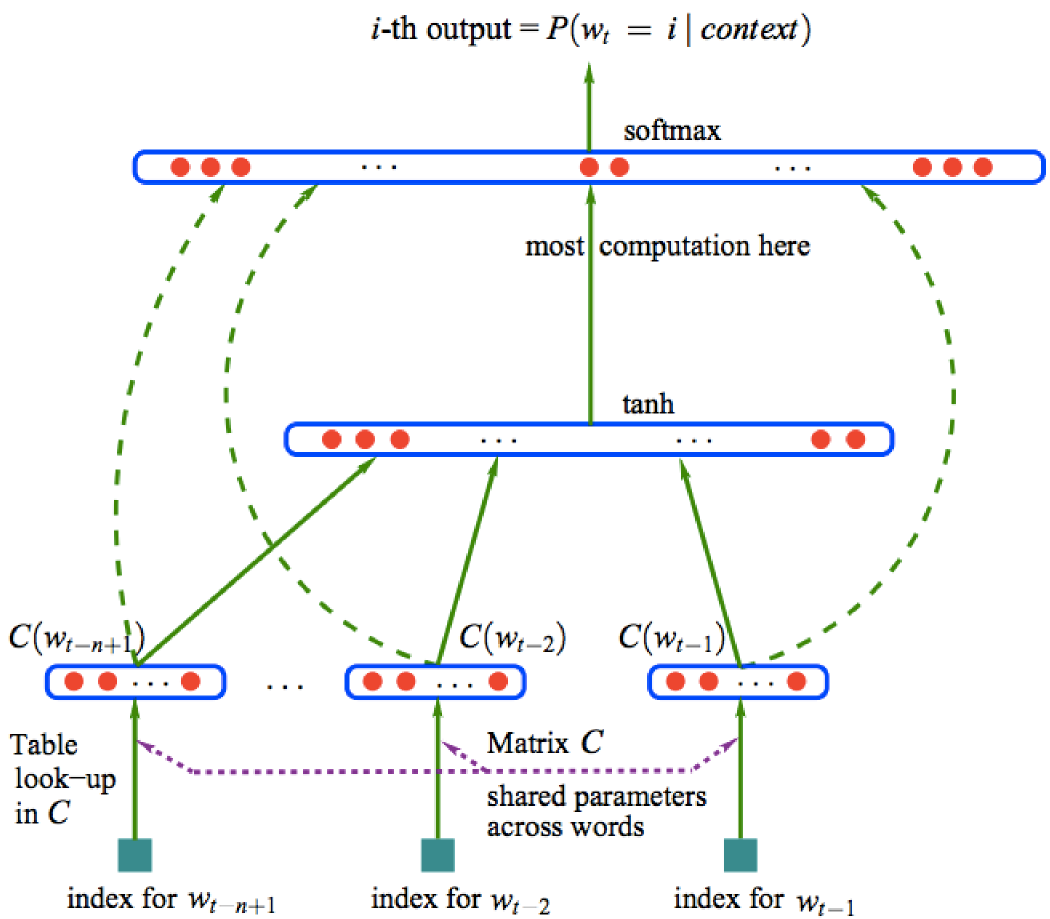
N-Gram Model

Normalize
on all words

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

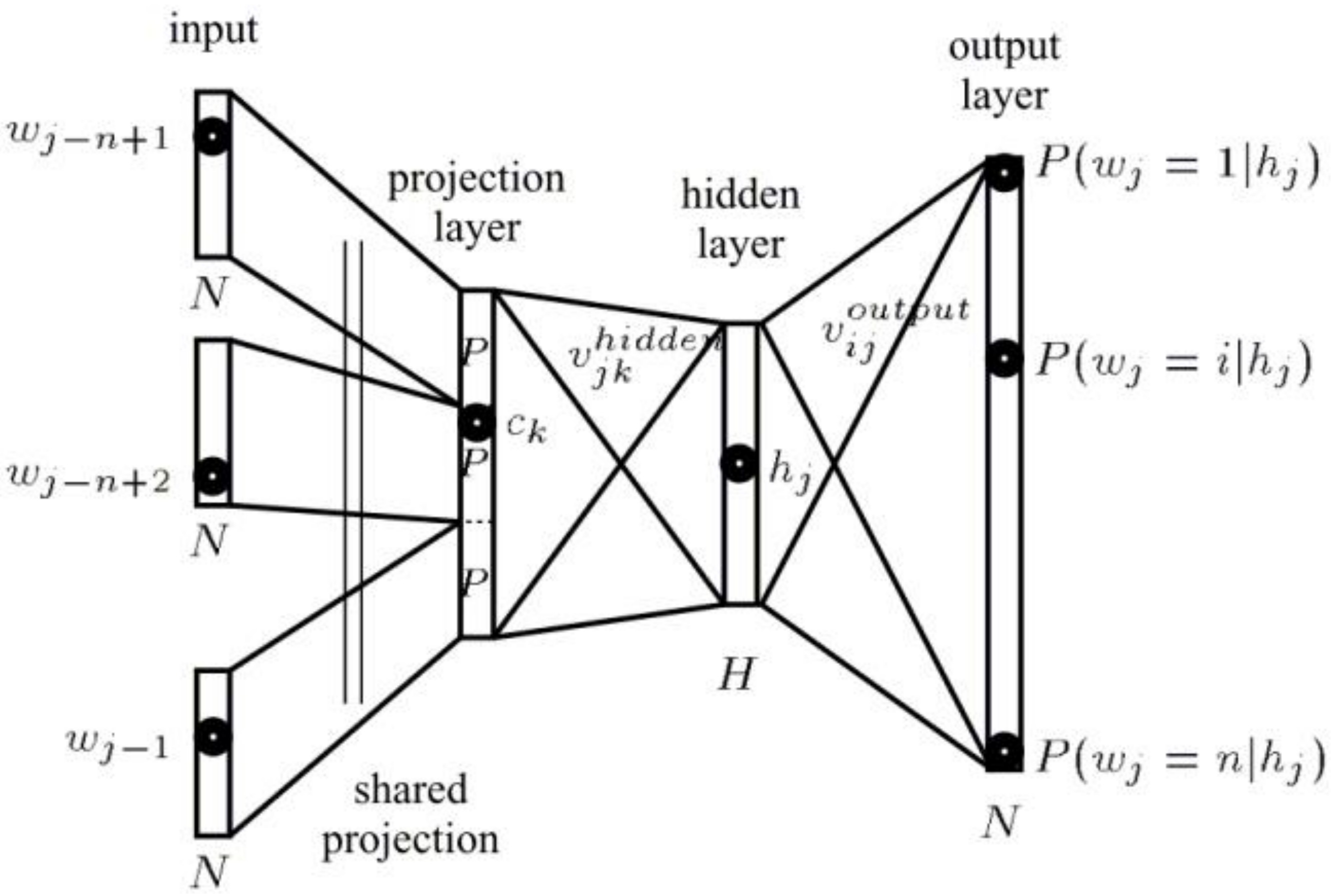
A part of the parameter matrix of a bigram language model

Feed Forward Neural Network LM



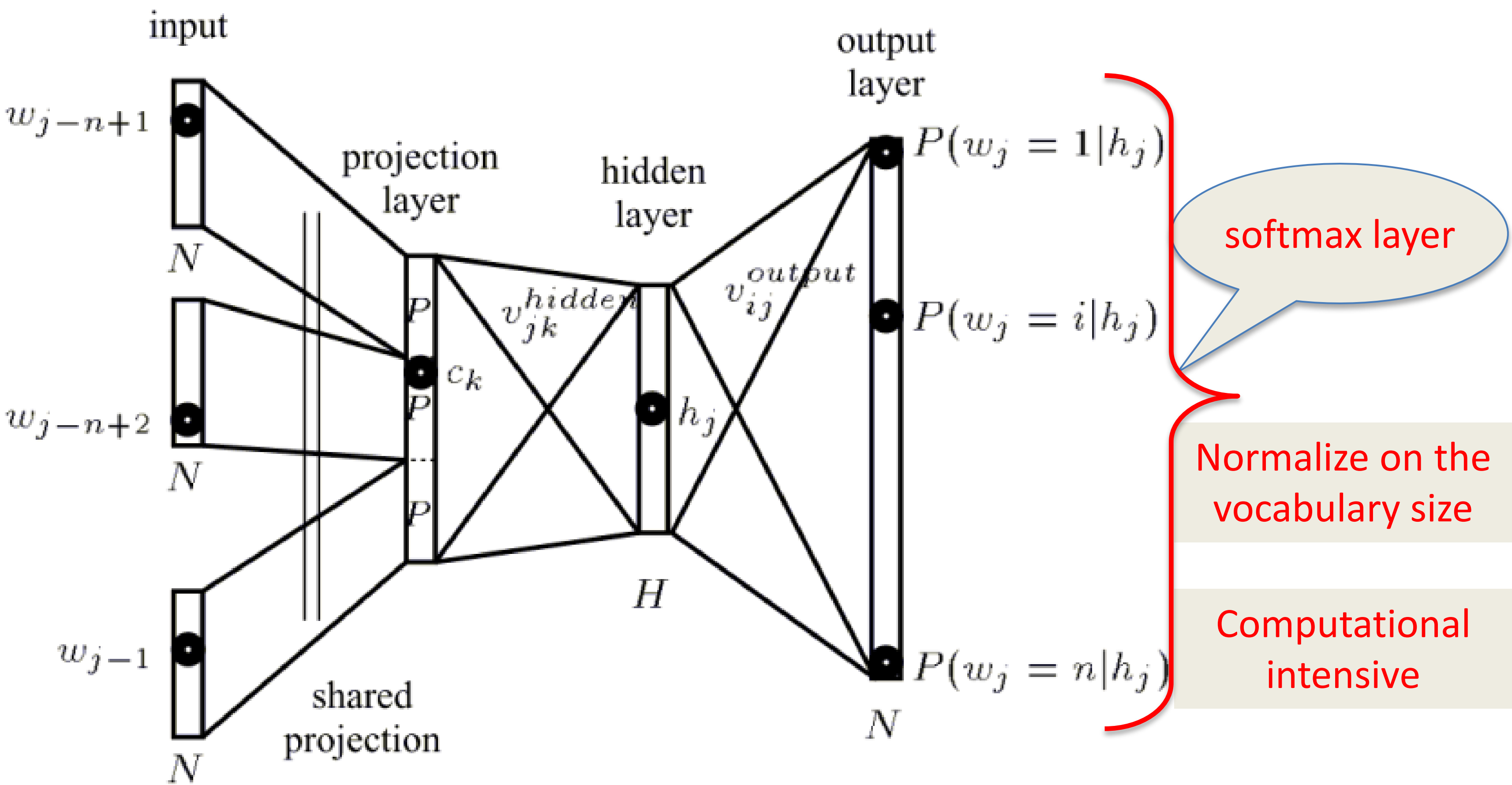
[Bengio et al., 2003]

Feed Forward Neural Network LM



[Bengio et al., 2003]

Feed Forward Neural Network LM



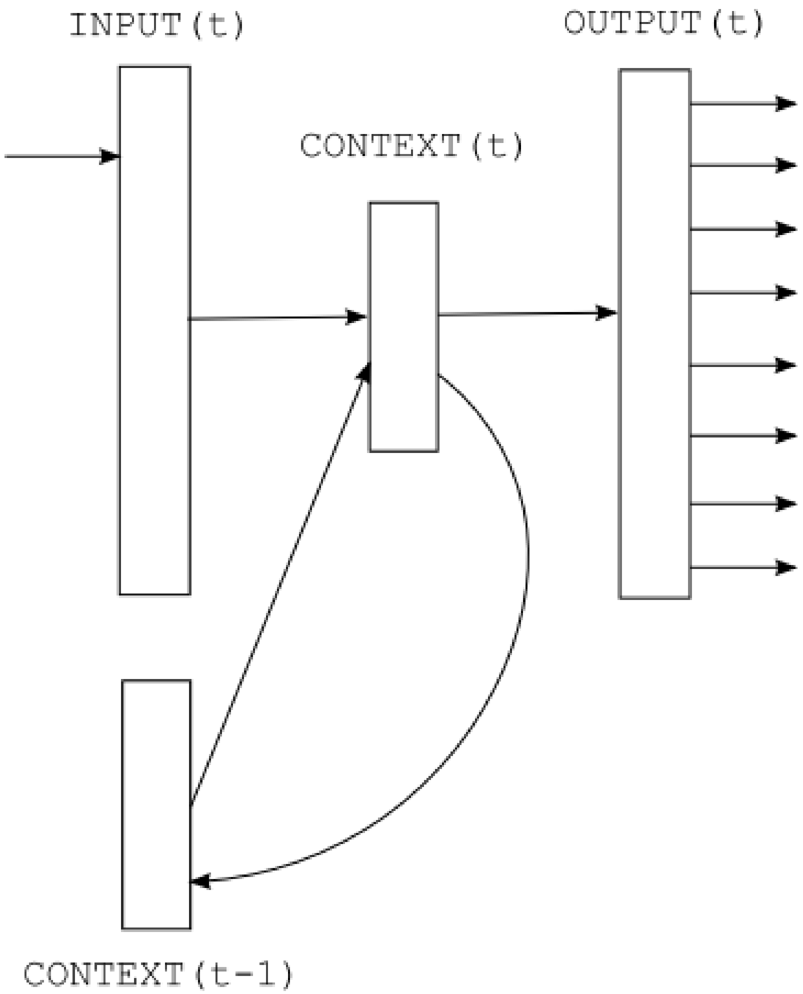
[Bengio et al., 2003]

Feed Forward Neural Network LM

- One shortcoming of FFNN LM is that it can only take limited length of history, just like N-gram LM
- An improved NN LM is proposed to solve this problem:

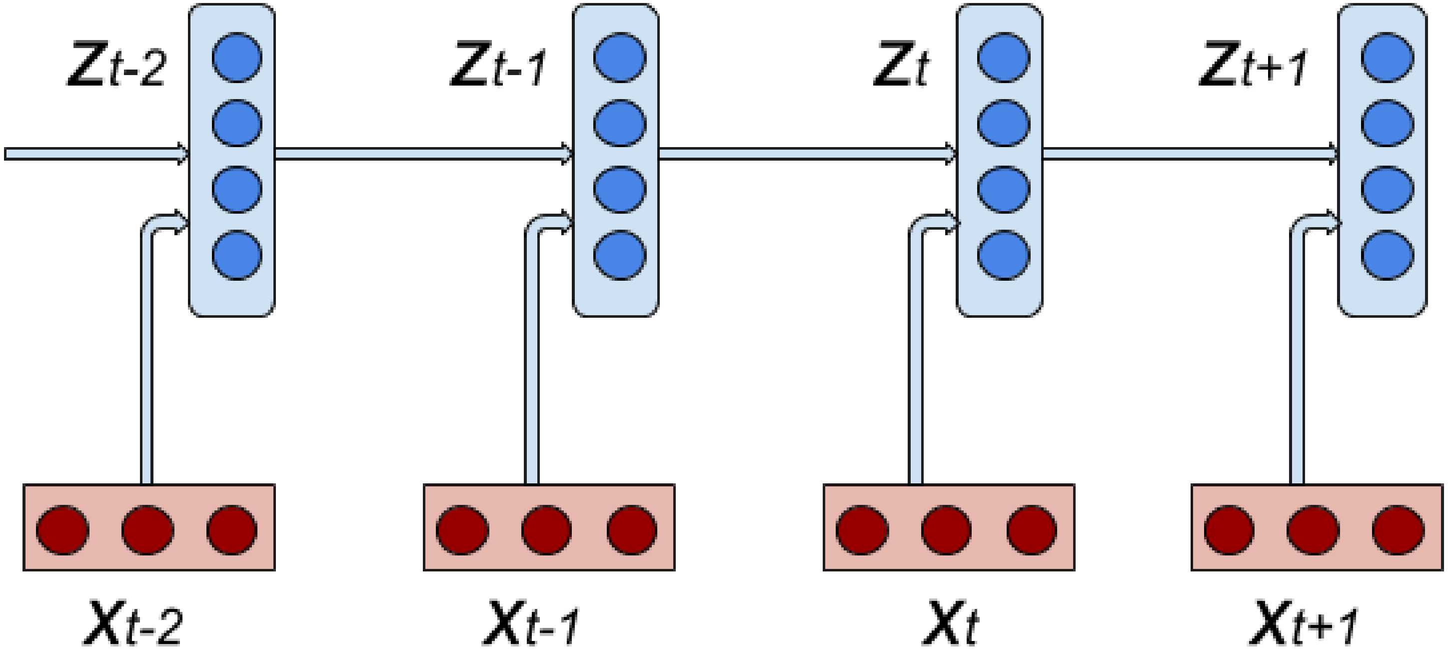
Recurrent Neural Network LM

Recurrent Neural Network LM



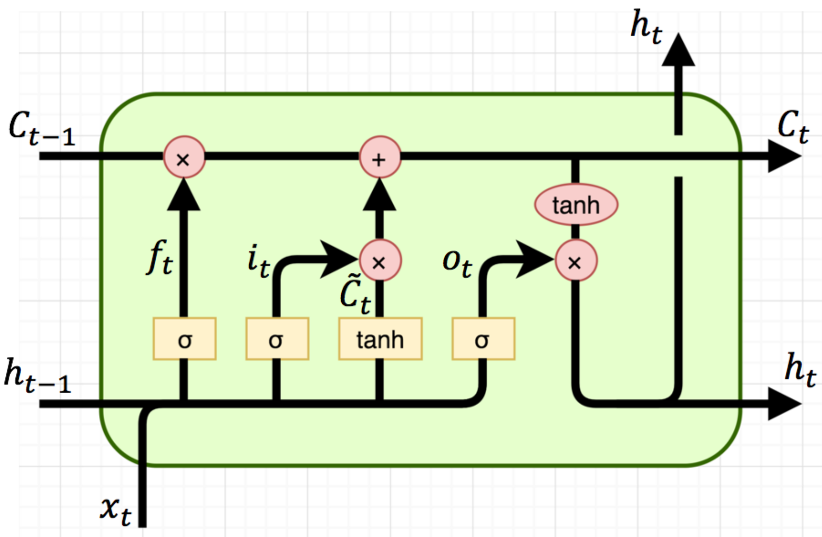
Recurrent Neural Network LM

Unfold the RNN LM along the timeline:

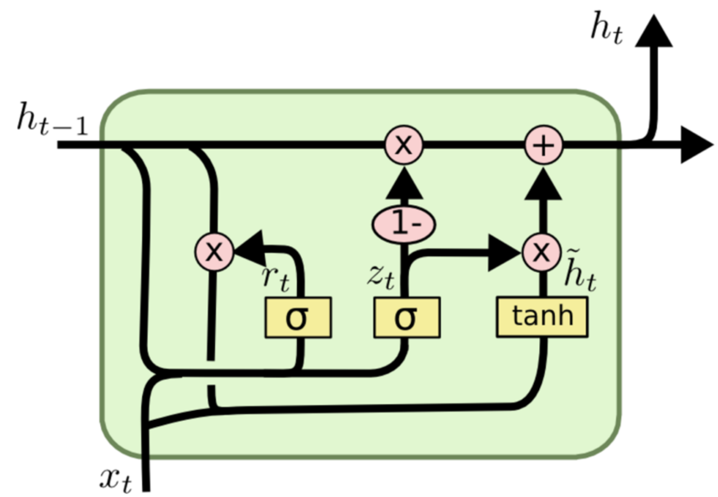


LSTM & GRU: Improved Implementation of RNN

- Mitigating gradient vanishing and exploding
- Long distance dependency



(a) Long Short-Term Memory

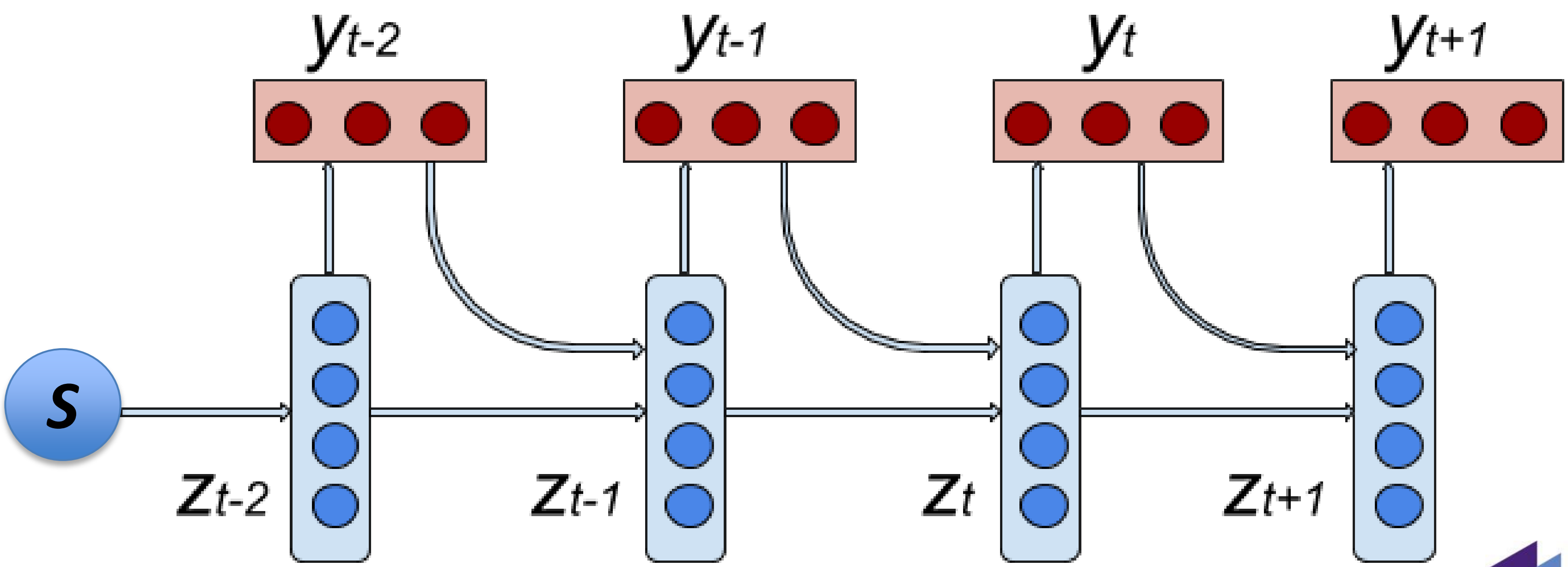


(b) Gated Recurrent Unit

Language Model for Generation

- Given language model $p(w_i|\, w_1 \ldots w_{i-1})$ and a history, we can generate the next word with highest LM score:

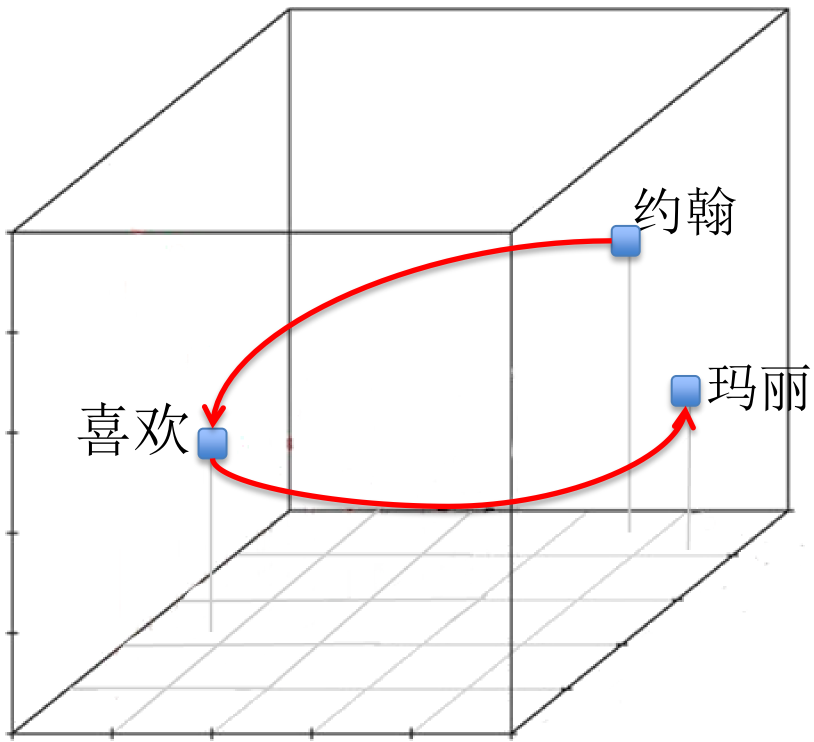
$$w_t = \underset{w'_t \in V}{\operatorname{argmax}}\, p(w'_t|\, w_1 \ldots w_{i-1})$$



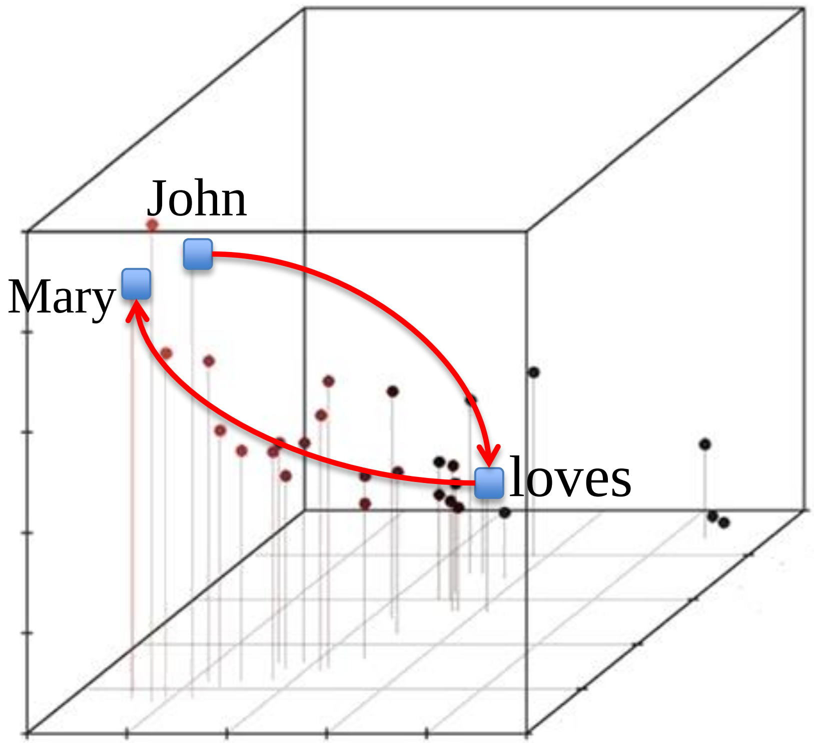
Content

- Background: Machine Translation and Neural Network
- Transition: From Discrete Spaces to Continuous Spaces
- Neural Machine Translation: MT in a Continuous Space
- Implementing Seq2Seq models with PyTorch
- Conclusion

Neural Machine Translation: MT in a Continuous Space



Chinese Space

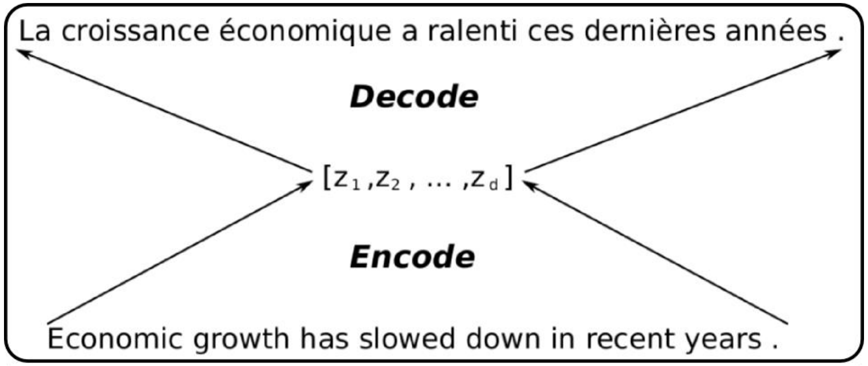


English Space

Neural Machine Translation: MT in a Continuous Space

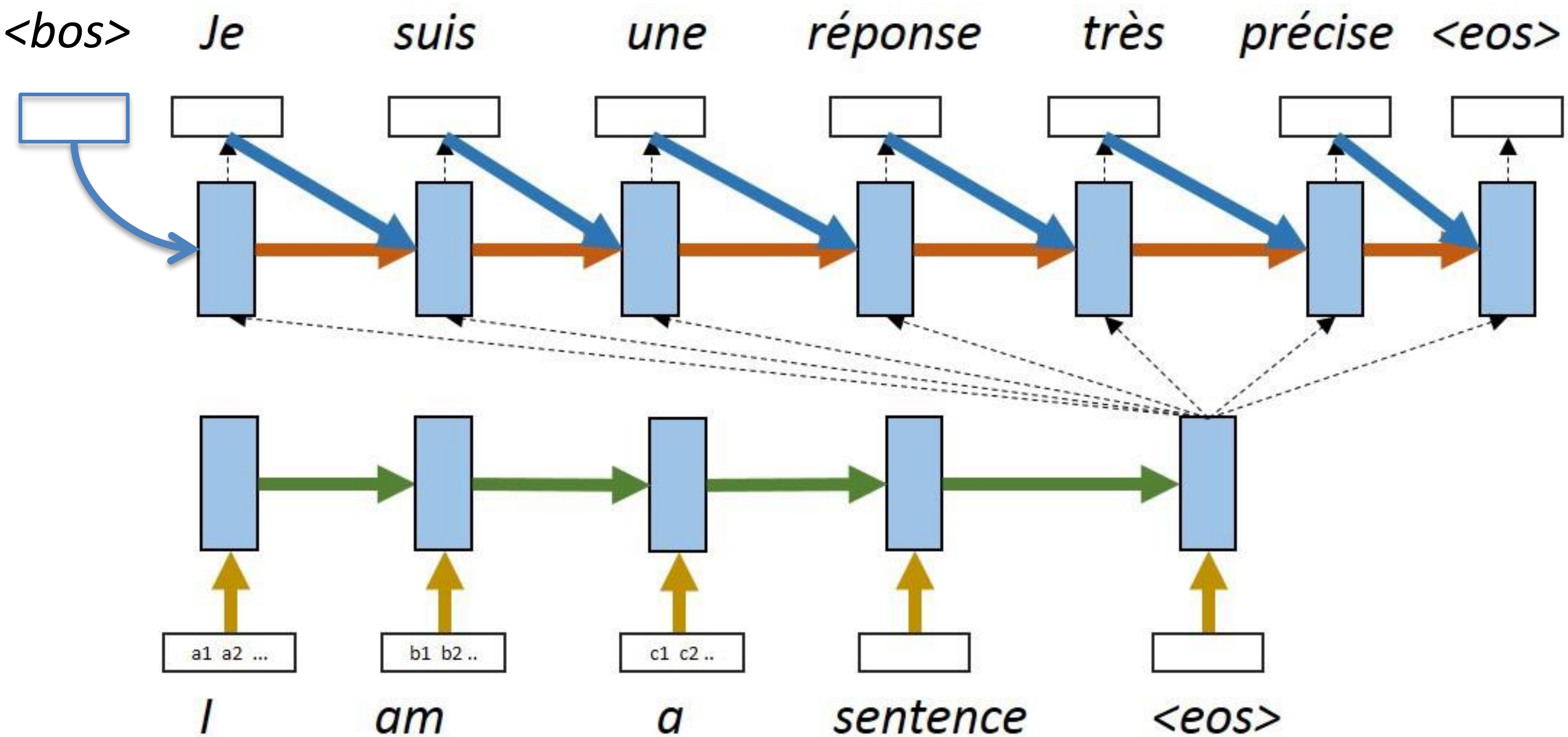
- **Neural Machine Translation (NMT)**
- **Attention-based NMT**

Neural Machine Translation



- The same things with SMT:
  - Trained with a parallel corpus
  - The input and output are word sequences
- The difference with SMT:
  - A single, large neural network
  - All the internal computing is conducted on real values without symbols
  - No word-alignment
  - No phrase table or rule table
  - No n-gram language model

Neural Machine Translation



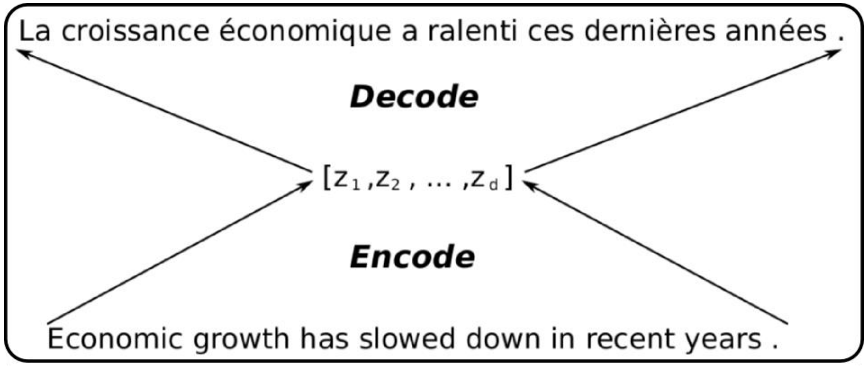
https://medium.com/@felixhill/deep-consequences-fa823a588e97#.sqlkiwvho

Neural Machine Translation: MT in a Continuous Space

- **Neural Machine Translation (NMT)**
- **Attention-based NMT**

Weakness of the simple NMT model

- The only connection between the source sentence and the target sentence is the single vector representation of the source sentence

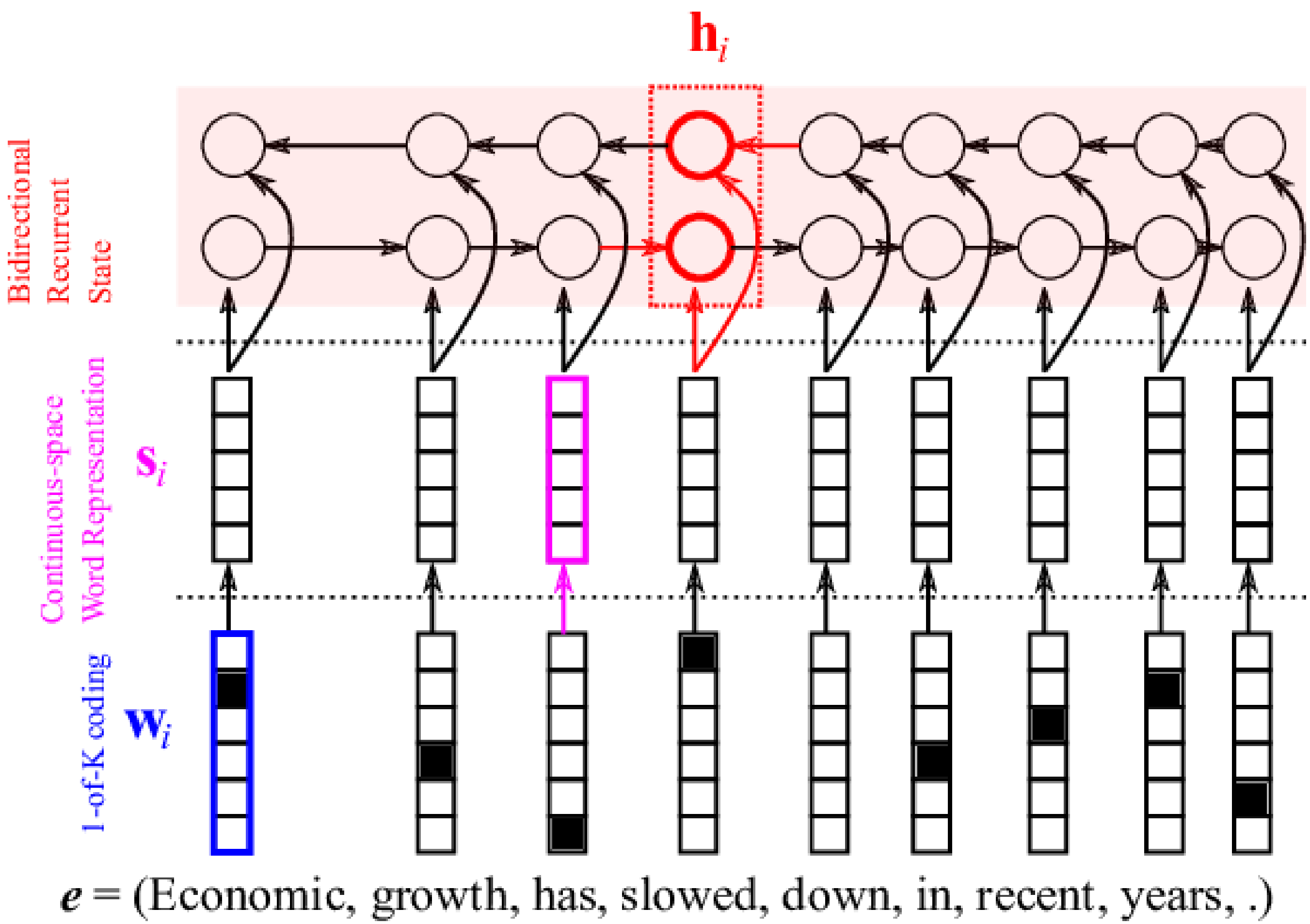


- It is hard for this fix-length vector to capture the meaning of the variable-length sentence, especially when the sentence is very long
- When the sentence becomes longer, the translation quality drops dramatically

Attention-based Model: Improvement

- Keep the states for all words rather than the final state only
- Use bi-directional RNN to replace single directional RNN
- Use an attention mechanism as a soft alignment between the source words and target words

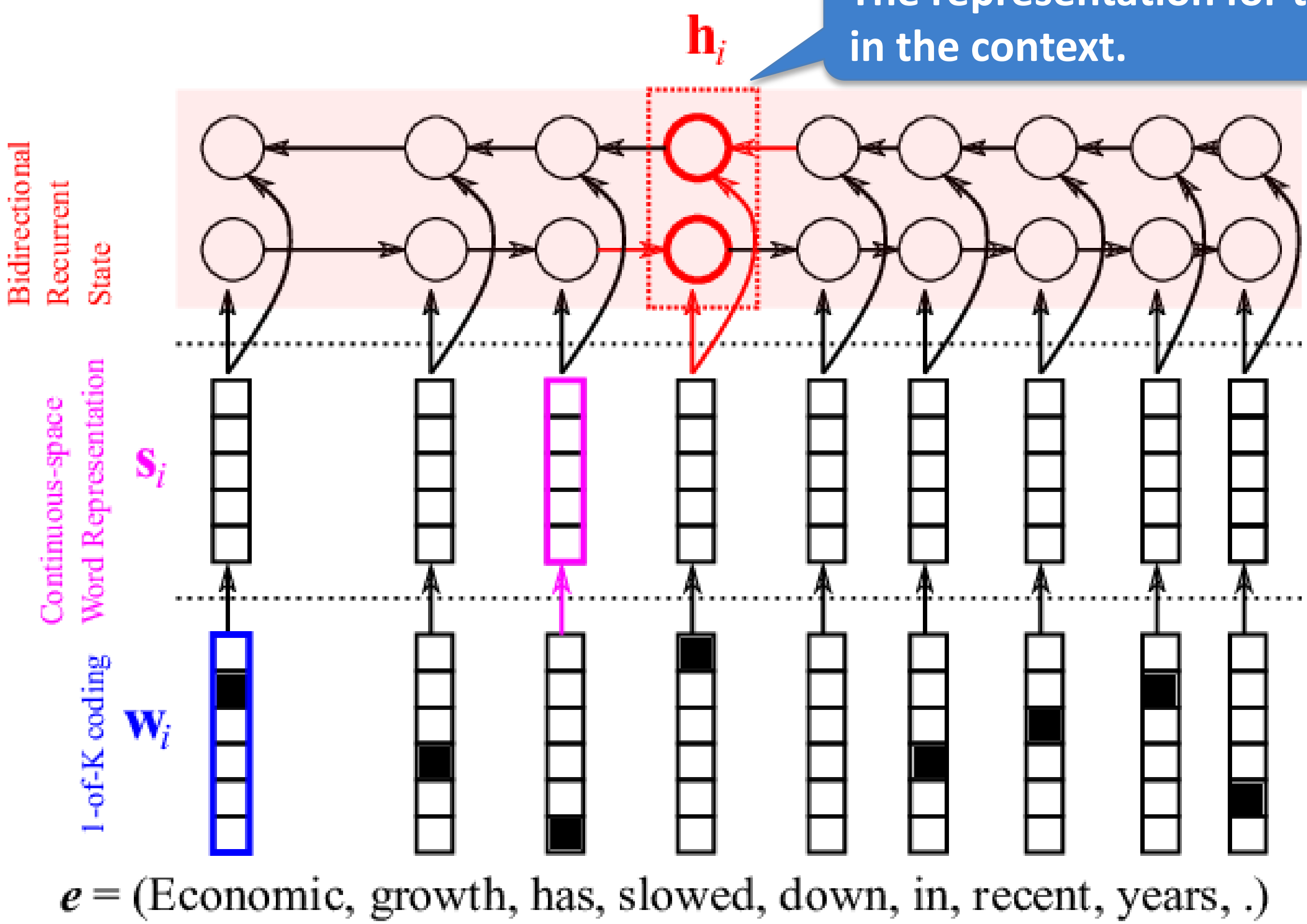
Bi-directional RNN



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

Bi-directional RNN

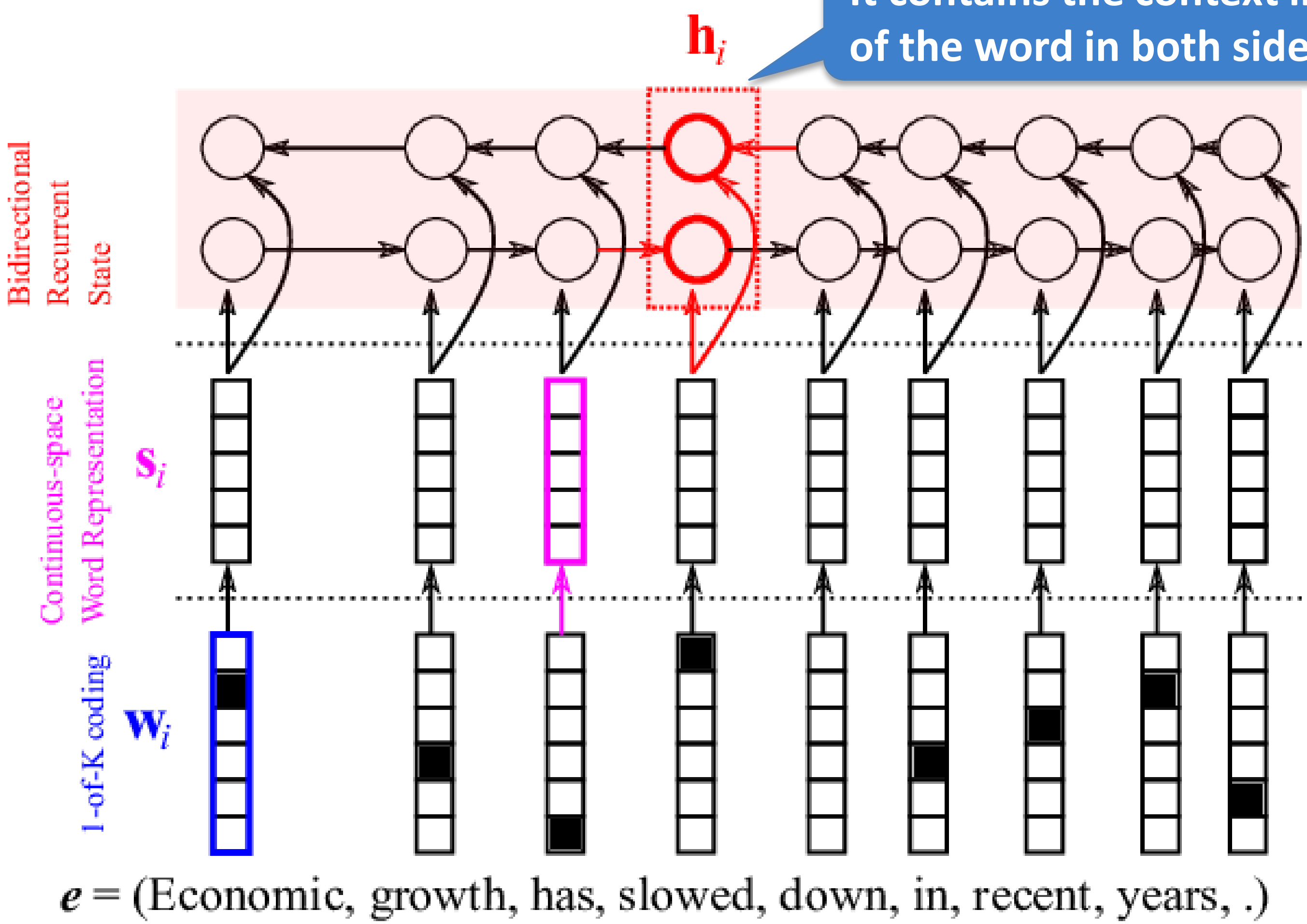
The representation for the word in the context.



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

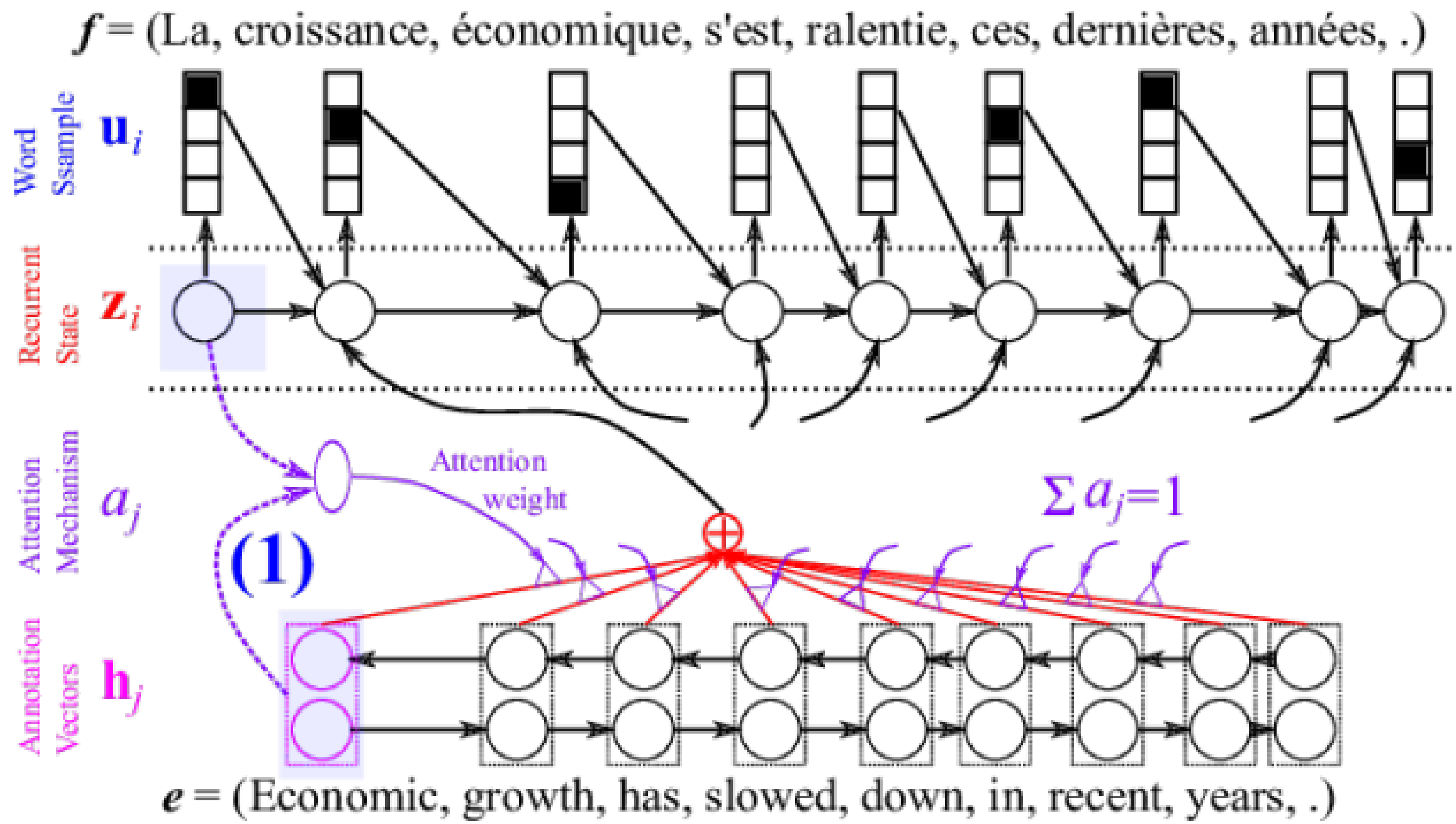
Bi-directional RNN

It contains the context information of the word in both sides



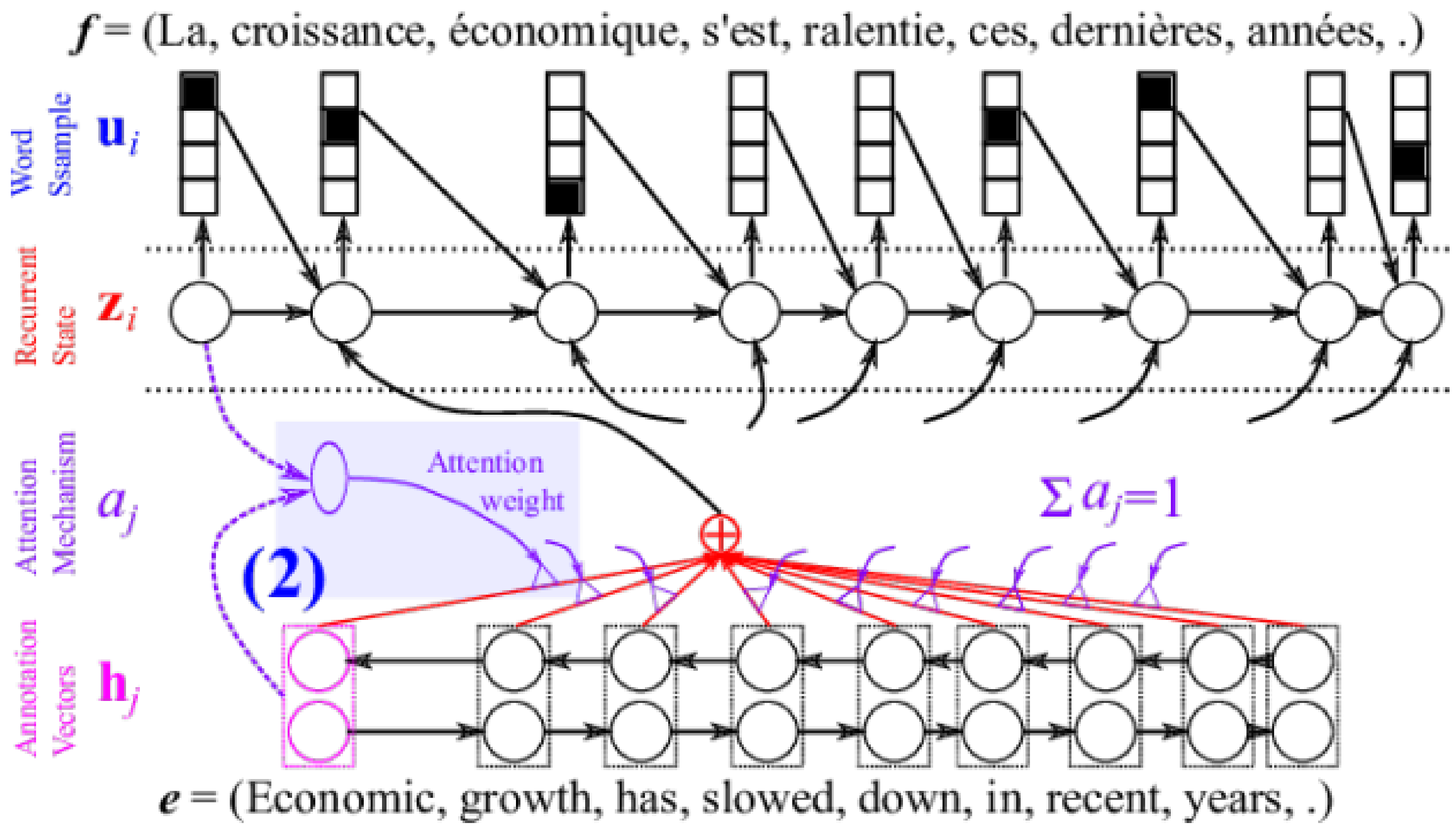
https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

Attention for NMT



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

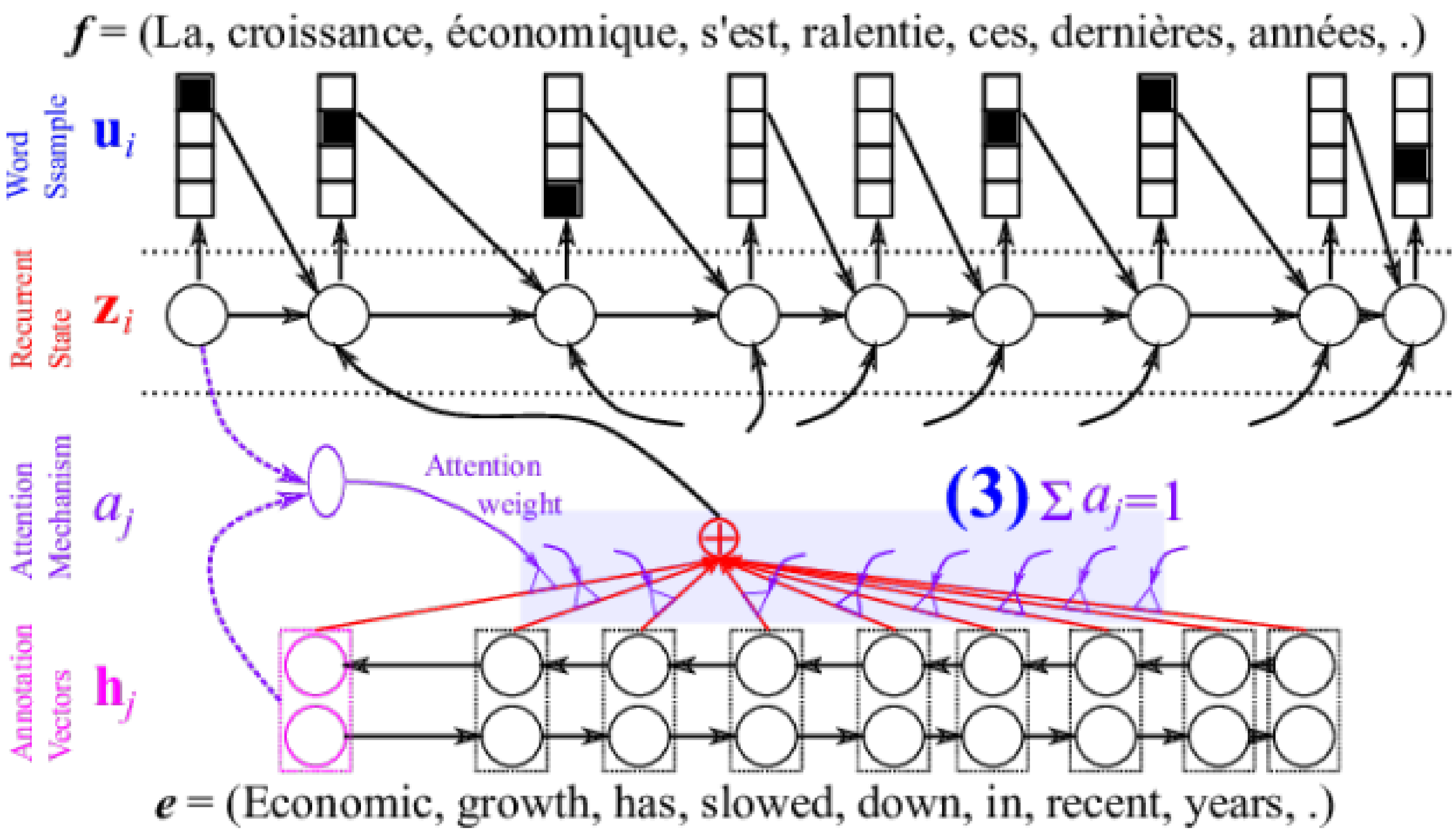
Attention for NMT



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

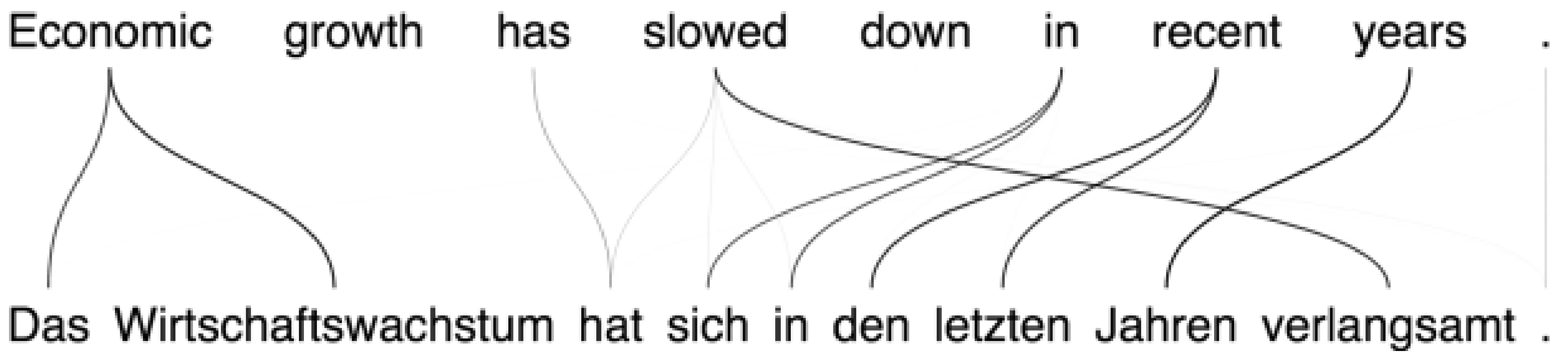
Attention for NMT

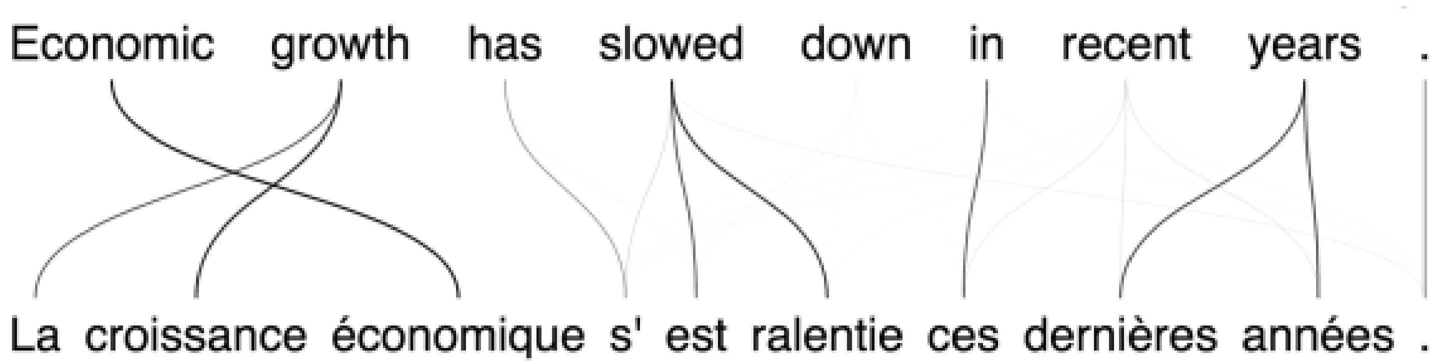
www.adaptcentre.ie



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

Soft Alignments by Attention Mechanism





https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

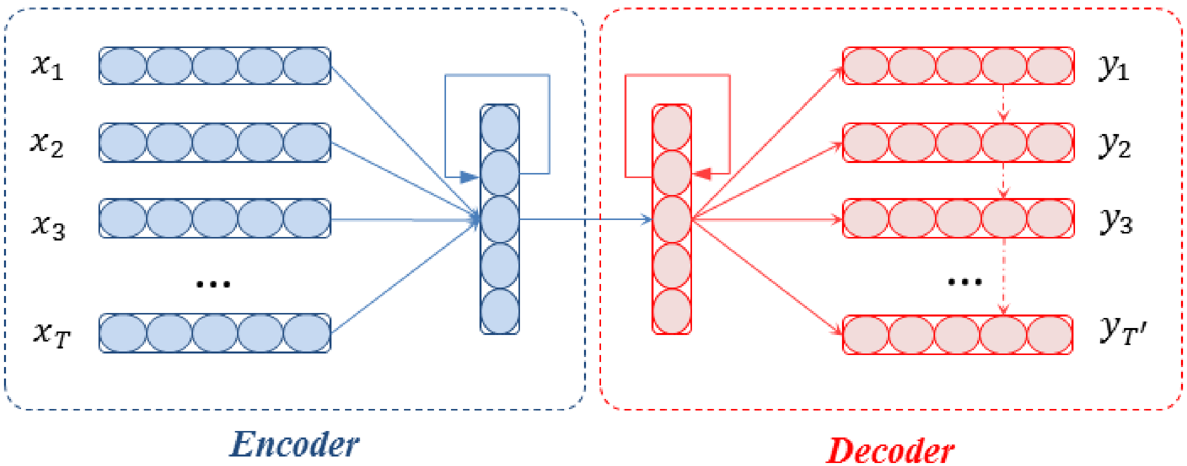
Attention-based NMT

- The attention-based NMT is very successful
- It’s performance has outperformed the SoA of SMT
- Attention mechanism is used in many DL tasks, such as image caption generation

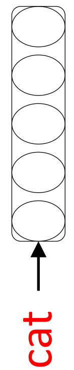
Content

- Background: Machine Translation and Neural Network
- Transition: From Discrete Spaces to Continuous Spaces
- Neural Machine Translation: MT in a Continuous Space
- Implementing Seq2Seq models with PyTorch
- Conclusion

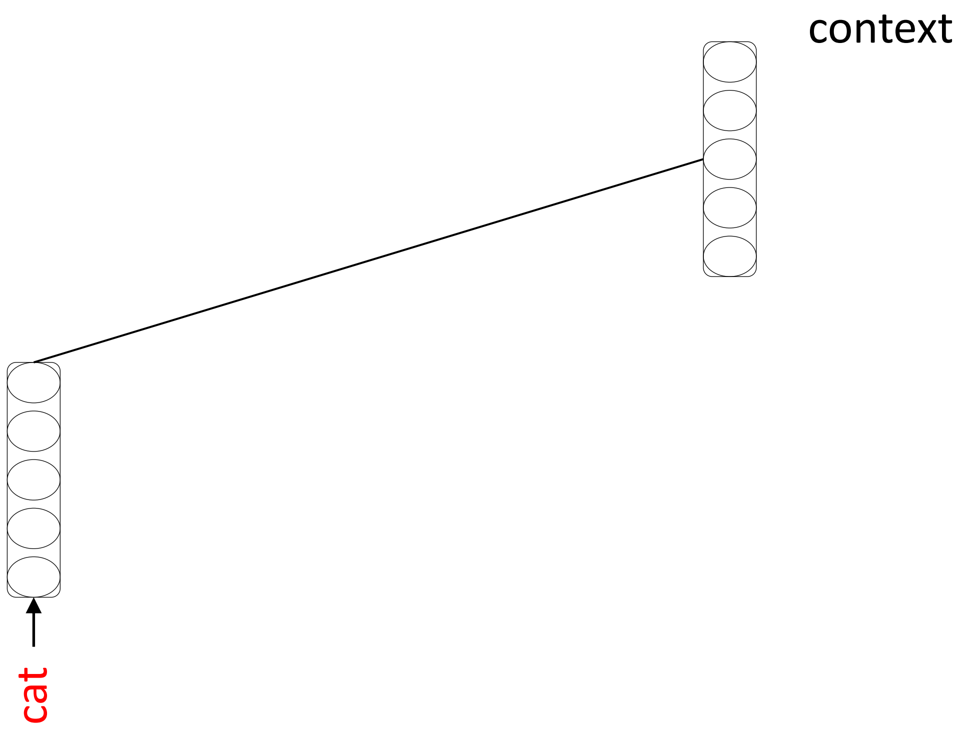
Encoder-Decoder Model



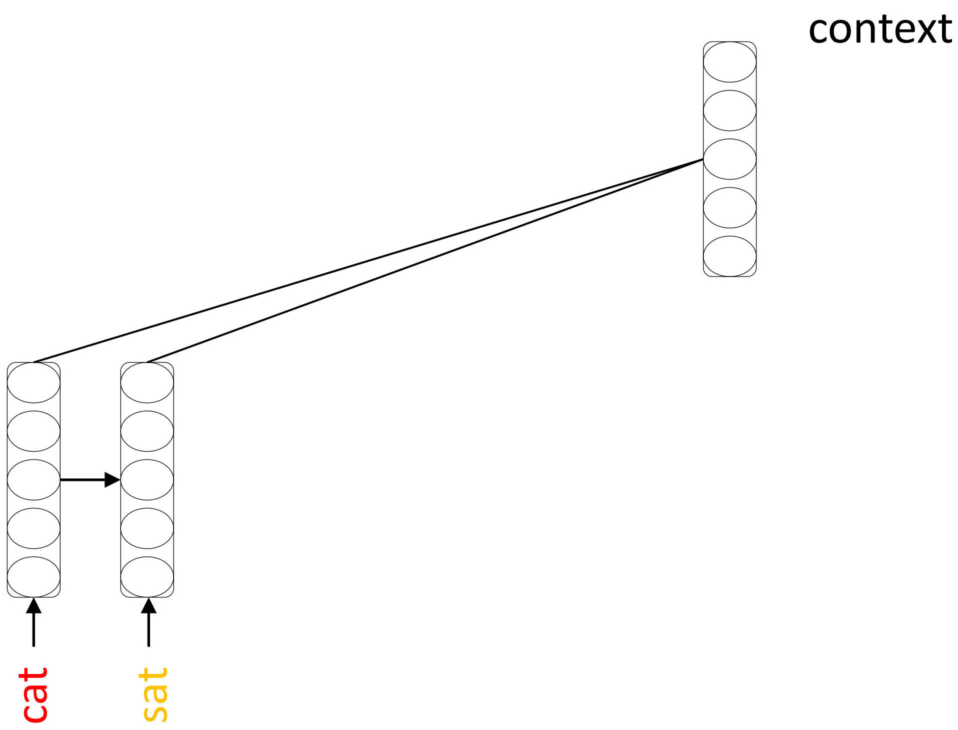
Encoding



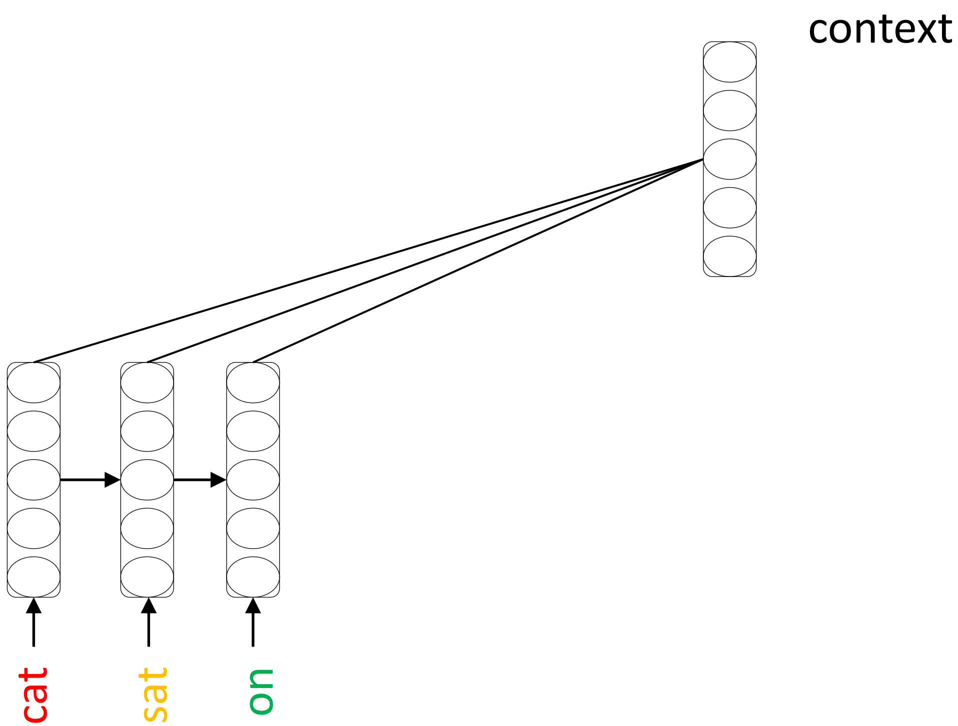
Encoding



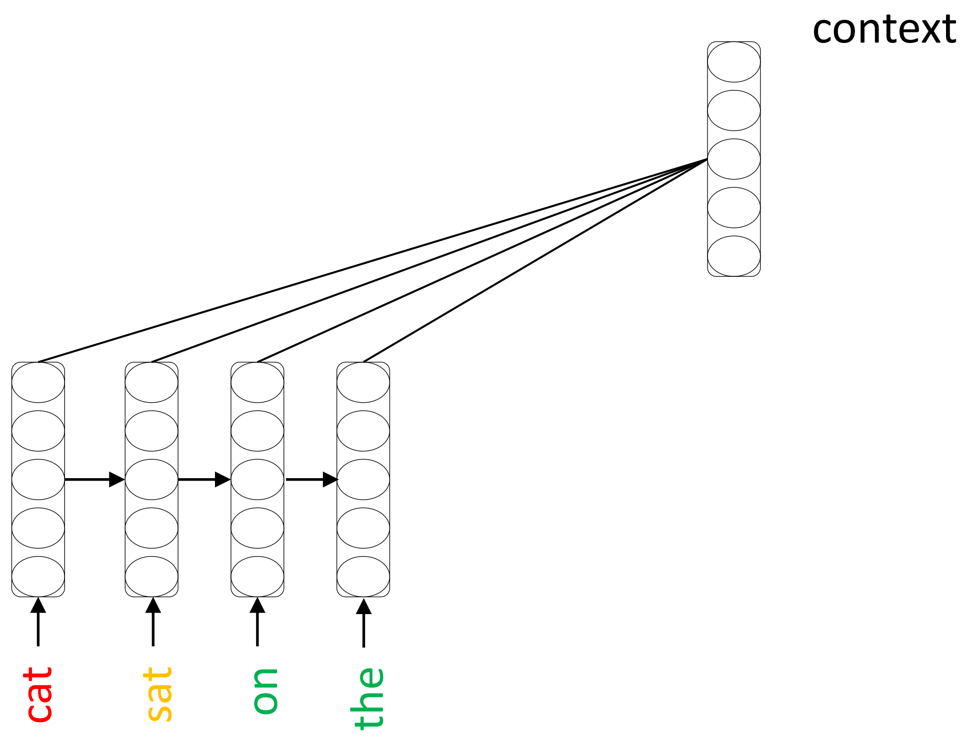
Encoding



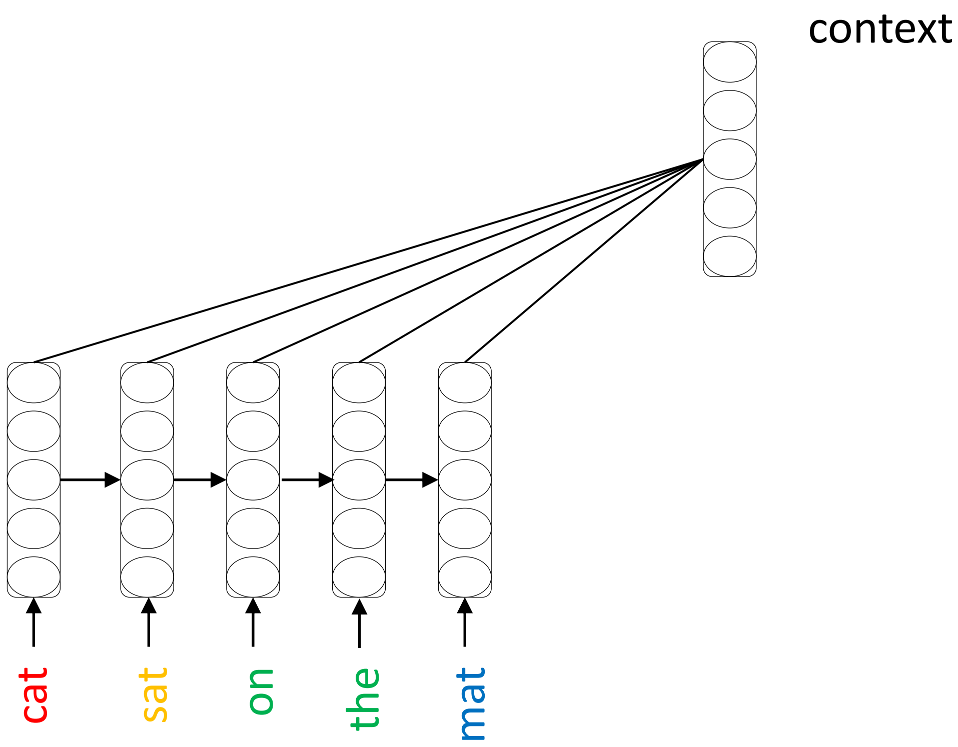
Encoding



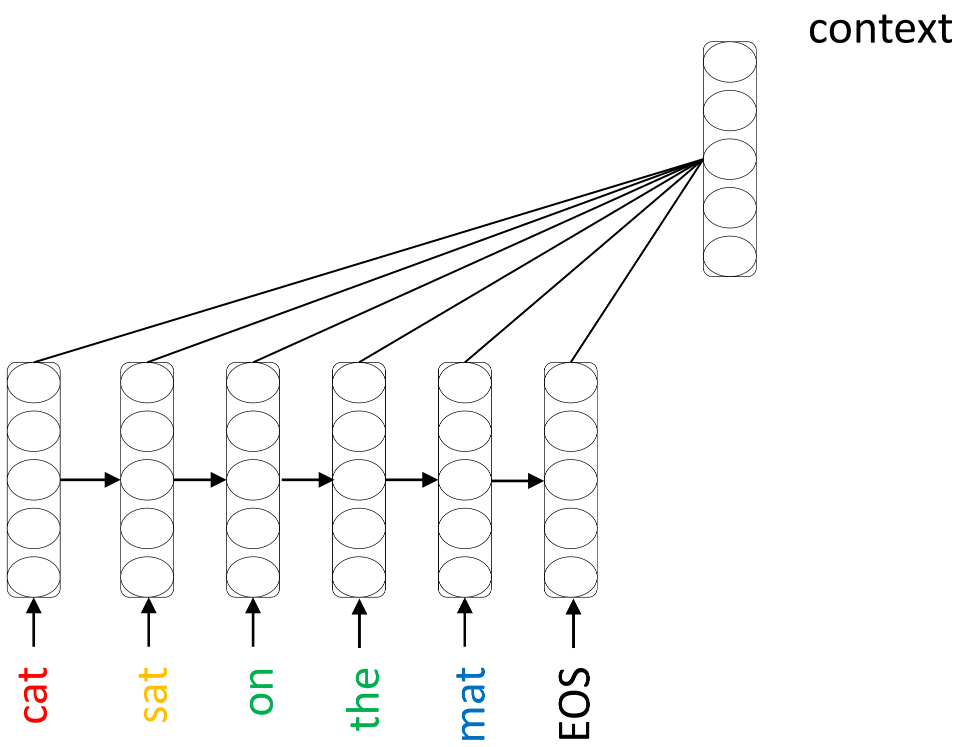
Encoding



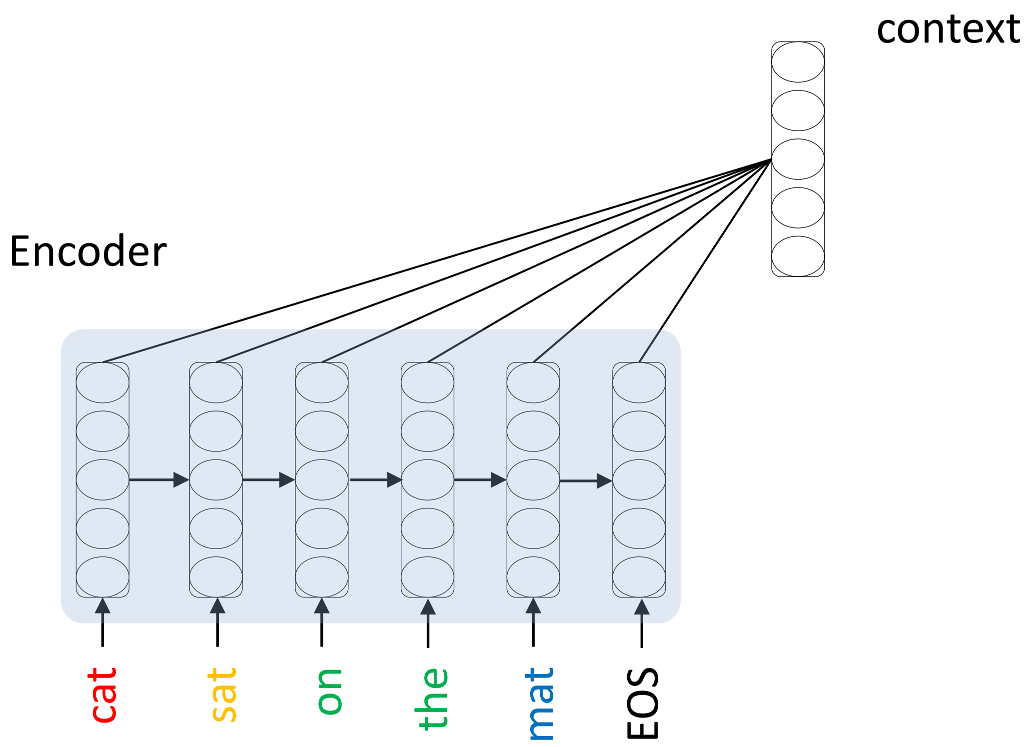
Encoding



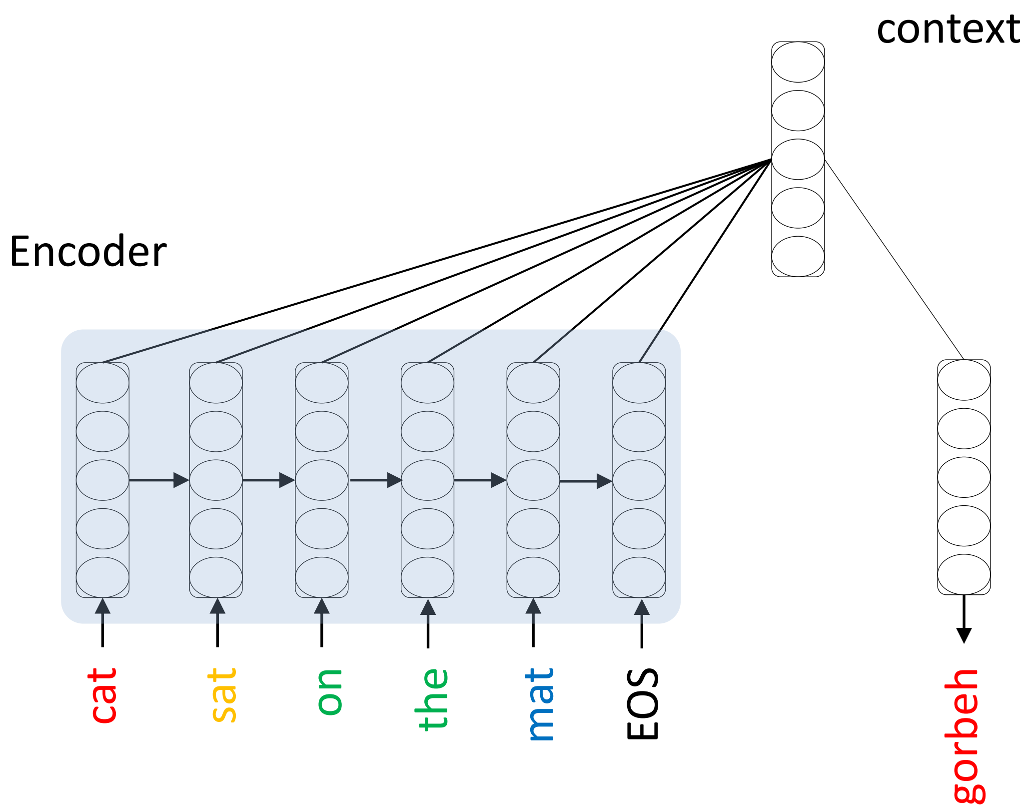
Encoding (Done!)



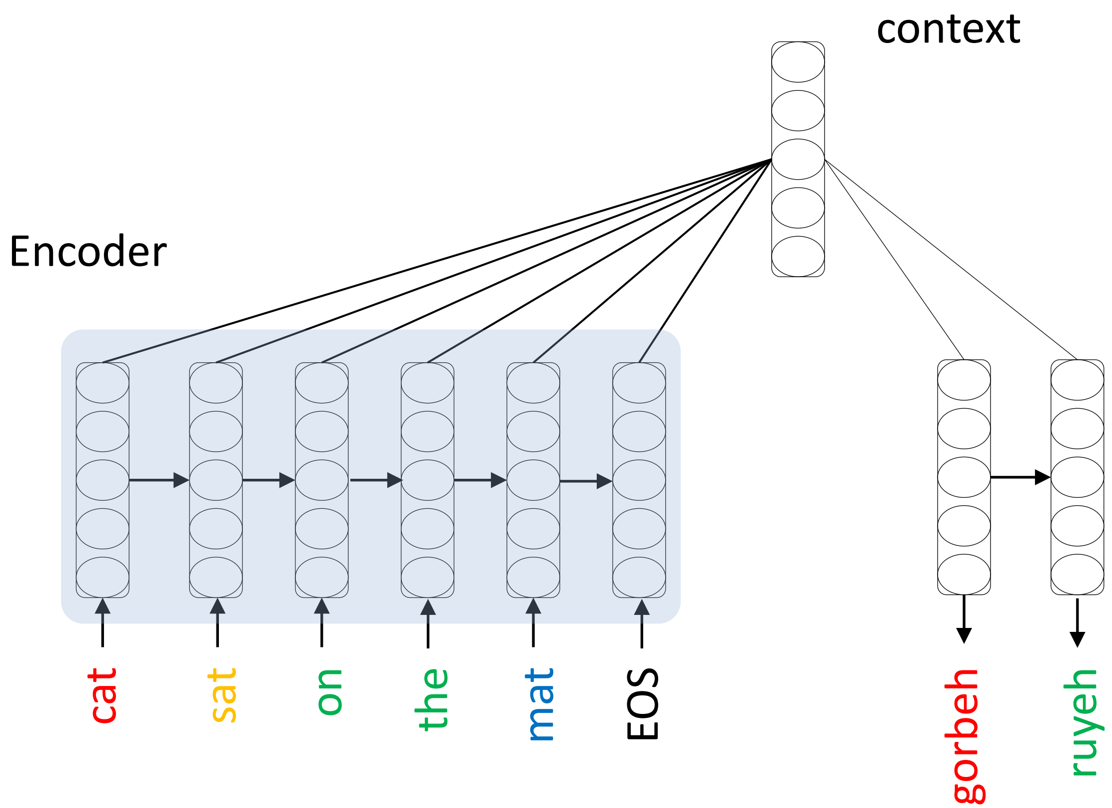
Encoding (Done!)



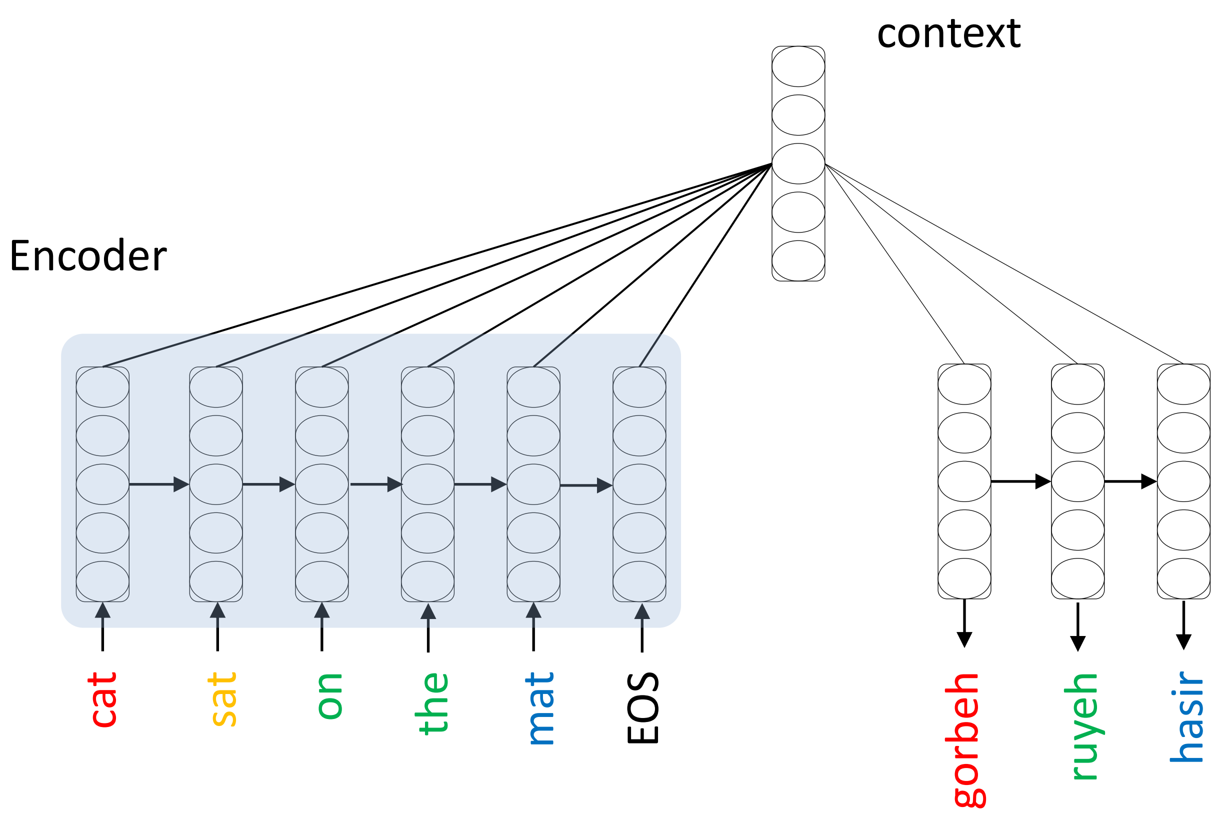
Decoding



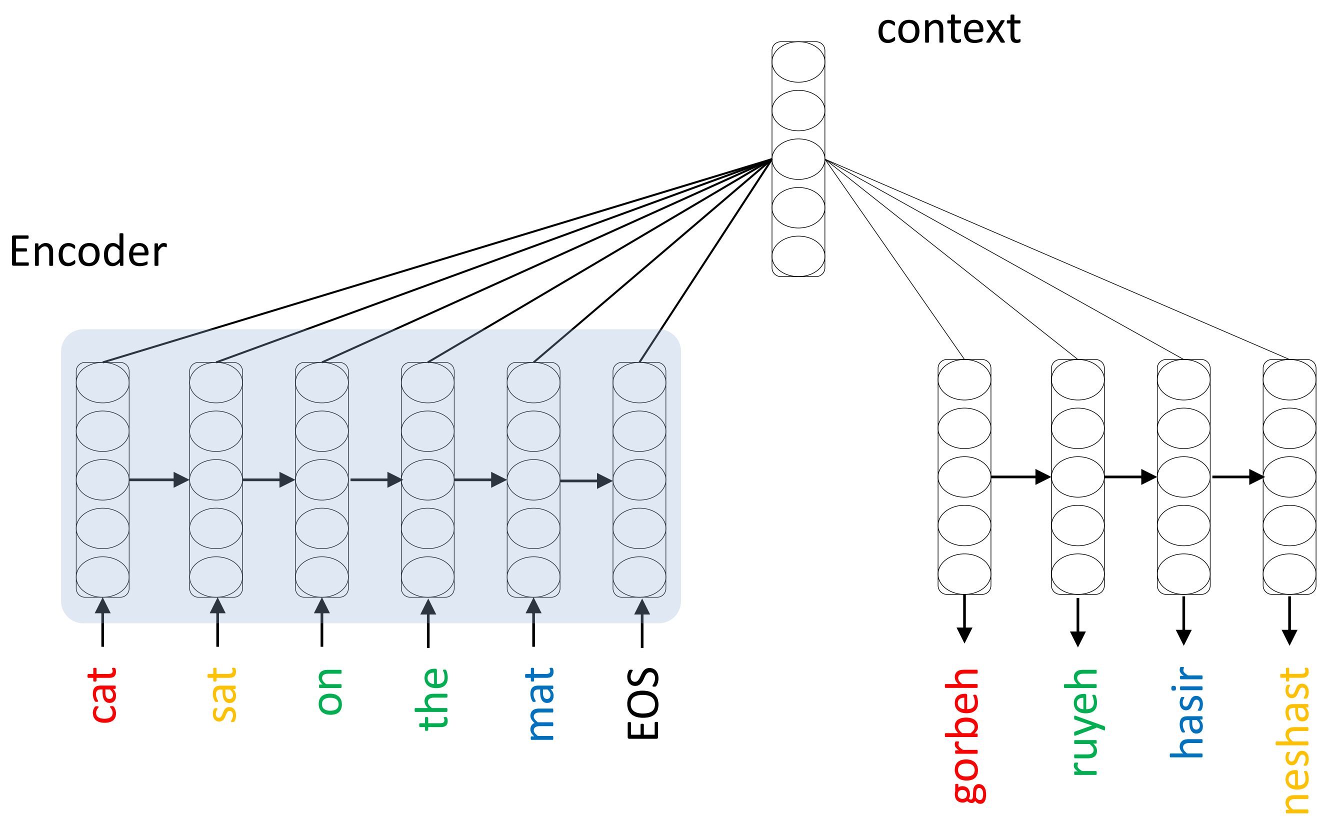
Decoding



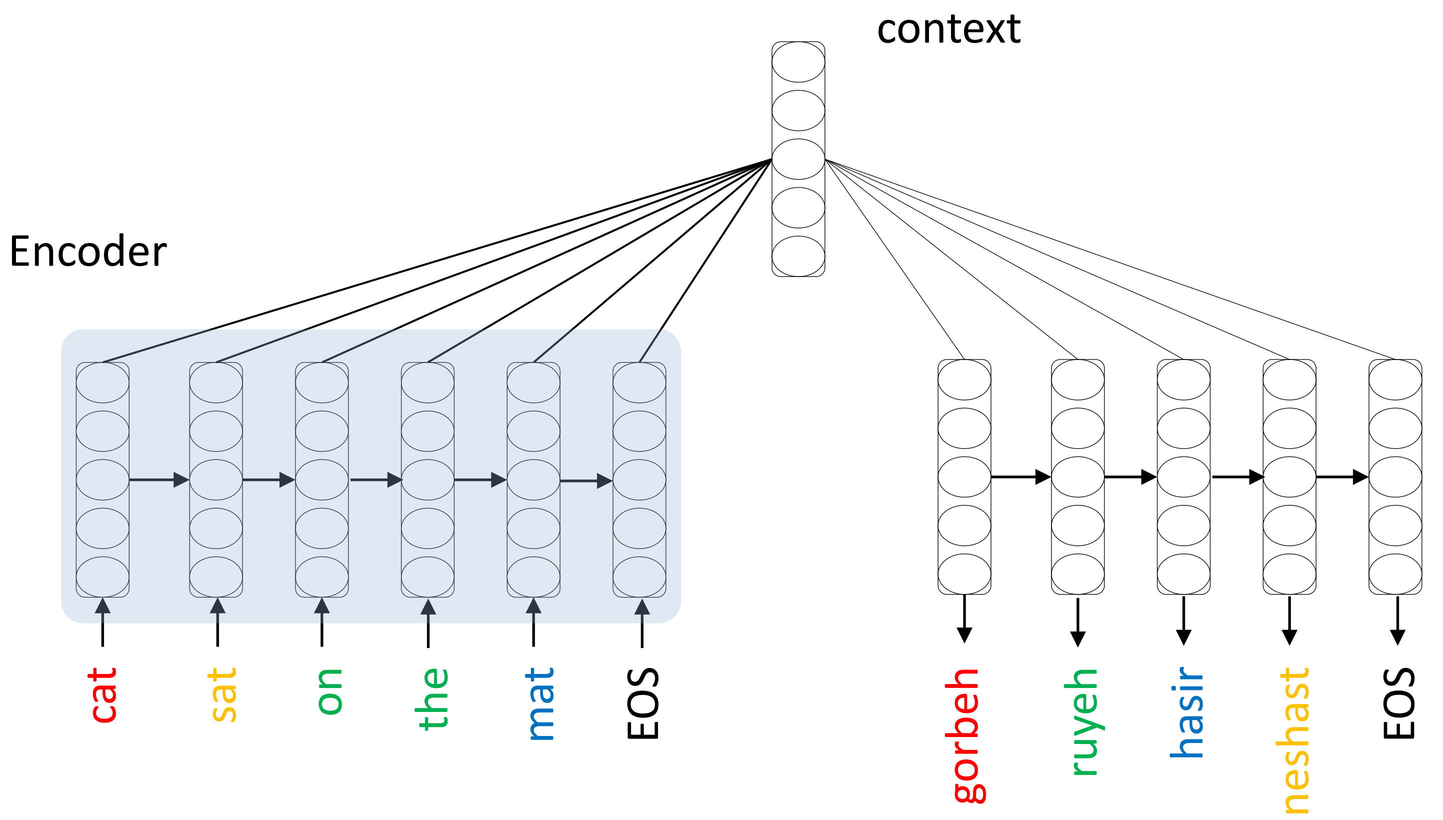
Decoding



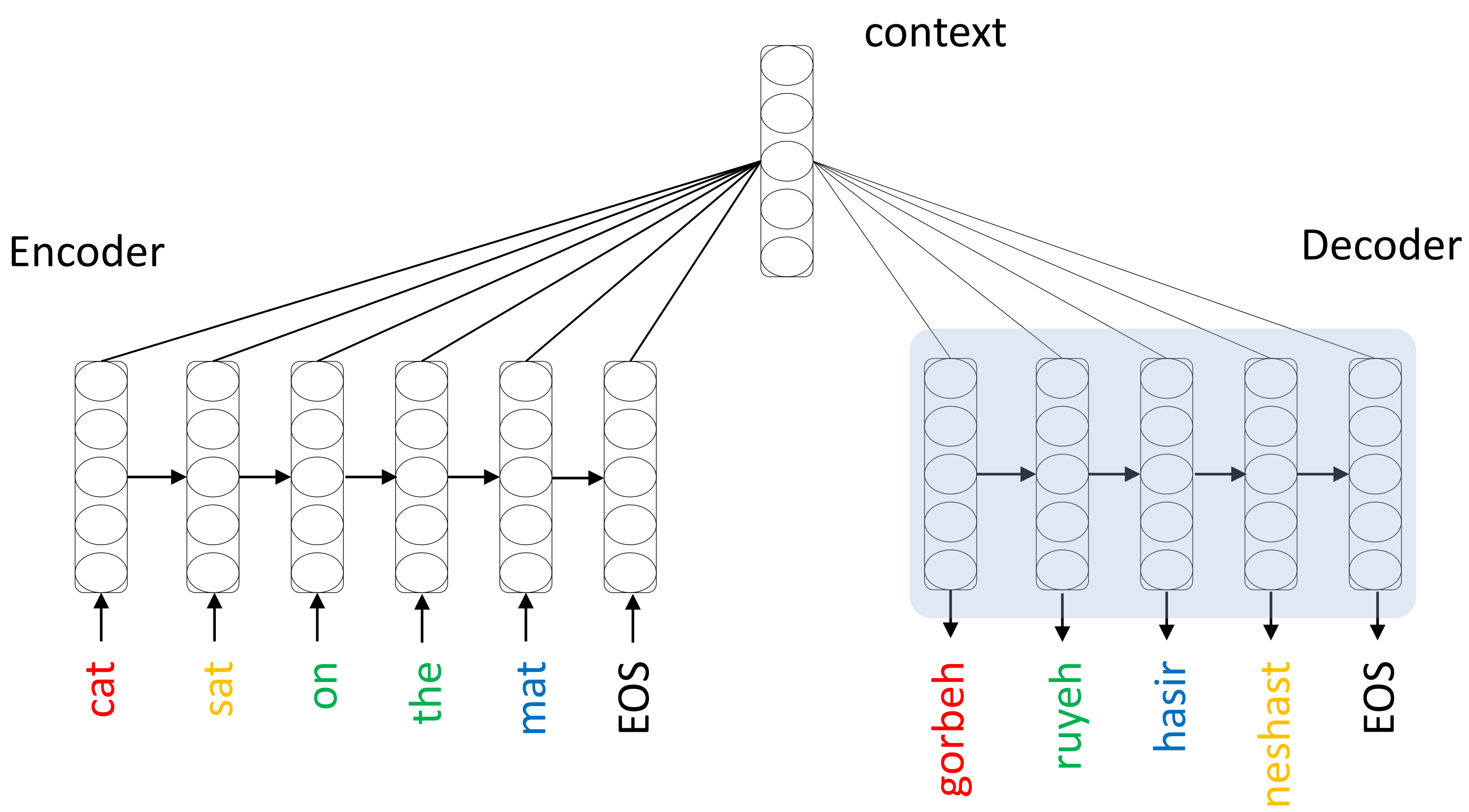
Decoding


context
Encoder
cat
sat
on
the
mat
EOS
gorbeh
ruyeh
hasir
neshast

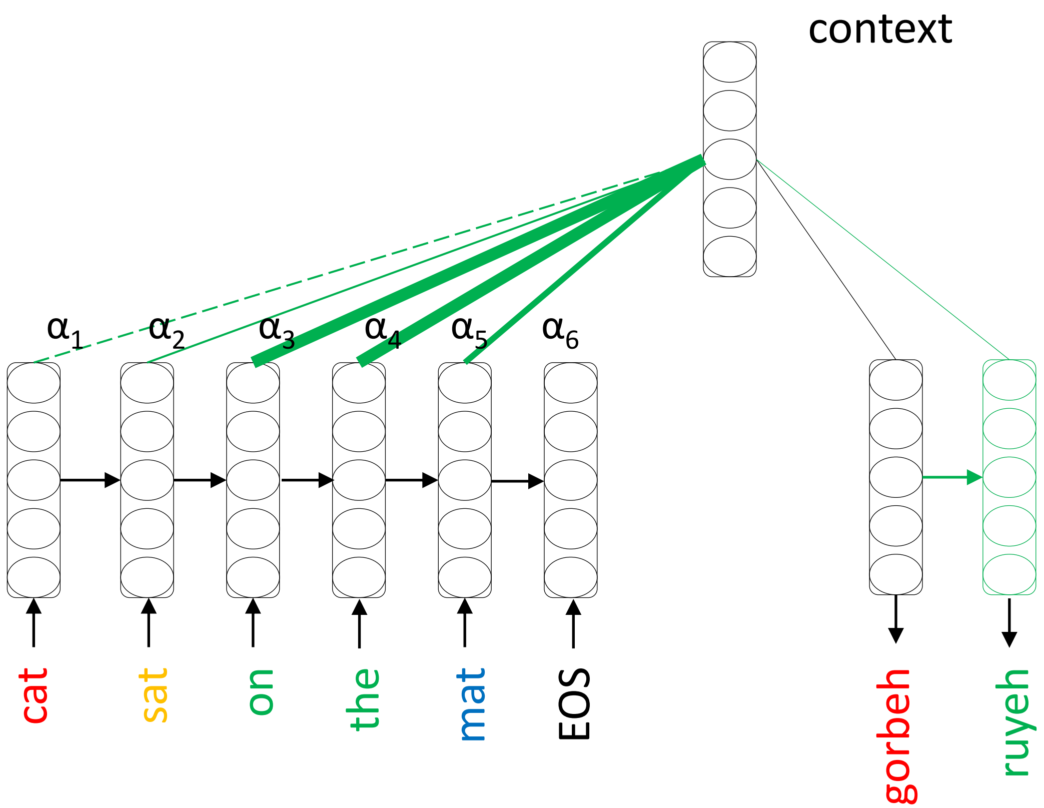
Decoding


context
Encoder
cat
sat
on
the
mat
EOS
gorbeh
ruyeh
hasir
neshast
EOS

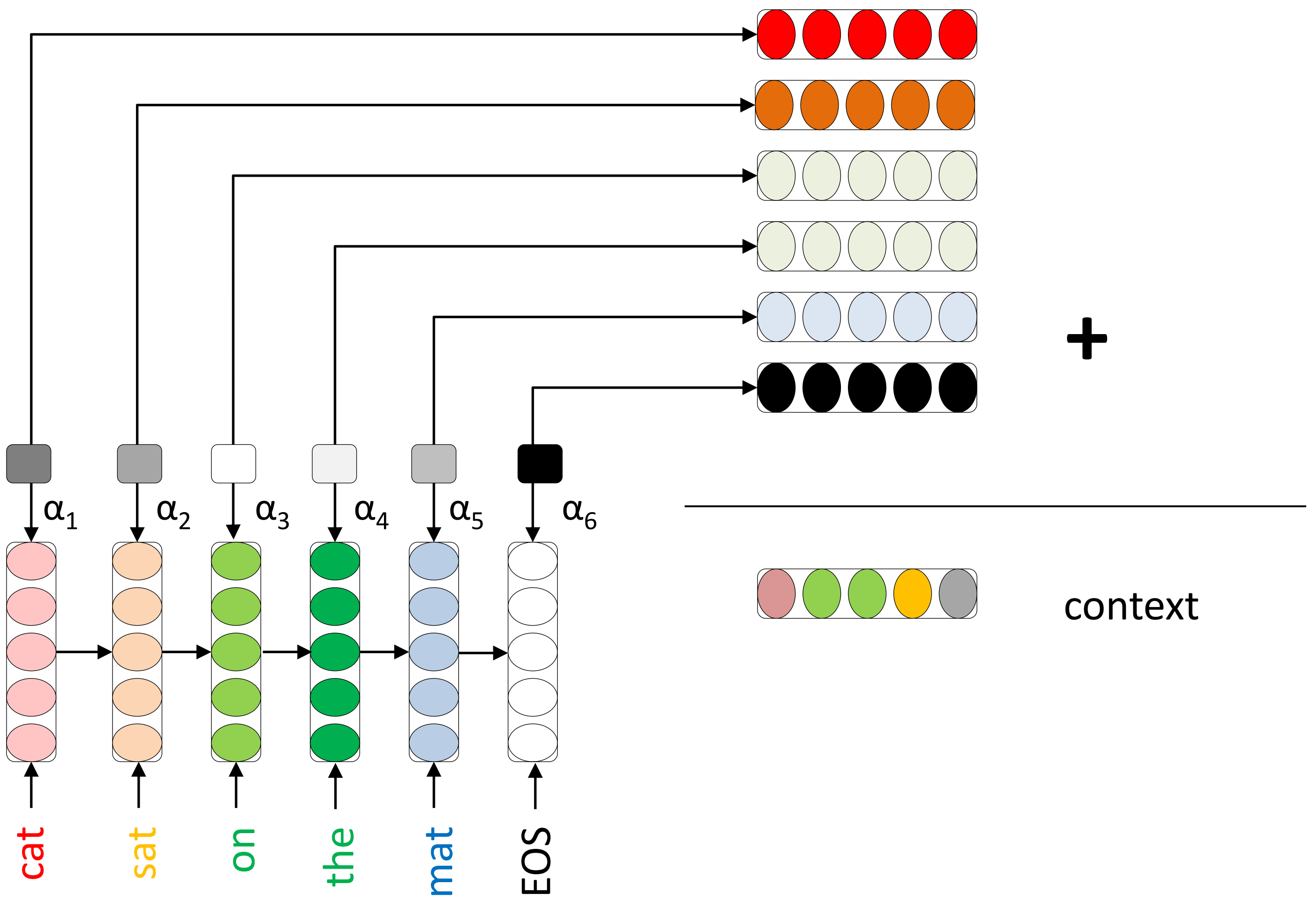
Decoding (Done!)



Attention!


context
α_1
α_2
α_3
α_4
α_5
α_6
cat
sat
on
the
mat
EOS
gorbeh
ruyeh

Attention!



+
α_1
α_2
α_3
α_4
α_5
α_6
context
cat
sat
on
the
mat
EOS

Encoder


```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```

Encoder


```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```

Encoder


```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```

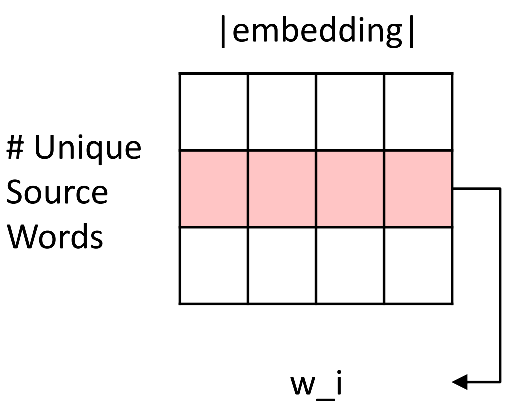
Encoder


```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```

https://stackoverflow.com/questions/222877/what-does-super-do-in-python

Encoder

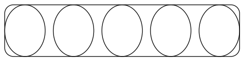

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```



Encoder

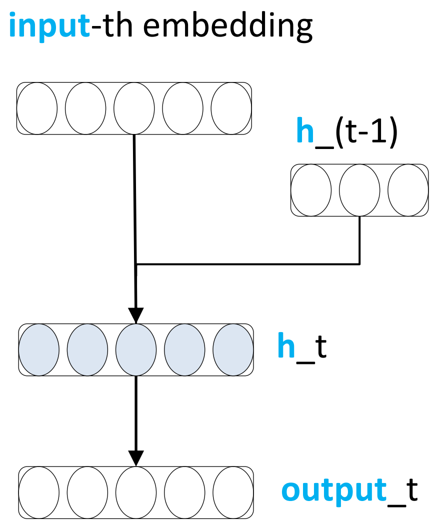

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```

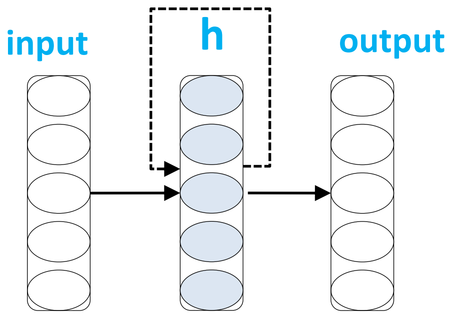
input-th embedding



Encoder

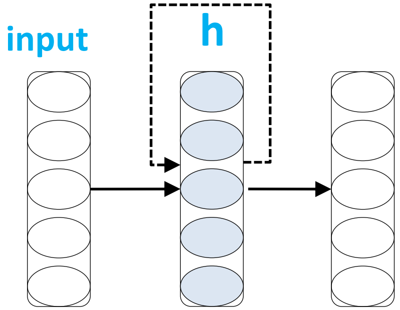

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```





Encoder


```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding (input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.gru(output, hidden)
        return output, hidden

    def initHidden(self):
        result = Variable(torch.zeros(1, 1, self.hidden_size))
        if use_cuda:
            return result.cuda()
        else:
            return result
```



Decoder+Attention


```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)
```

Decoder+Attention


```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)
```

Decoder+Attention


```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)
```

Two embedding tables!?

Decoder+Attention

www.adaptcentre.ie

```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1, max_length=MAX_LENGTH):
        super(AttnDecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size)
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, self.output_size)
```

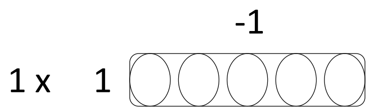
Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

index (digit)

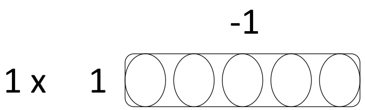
Decoder+Attention


```python
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```



Decoder+Attention


```python
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

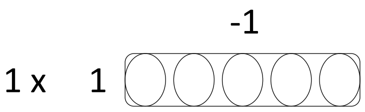


embedded: 1 x 1 x -1

embedded[0]: 1 x -1

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

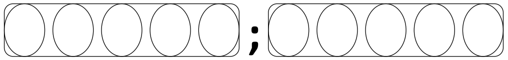


embedded: 1 x 1 x -1
embedded[0]: 1 x -1
hidden[0]: 1 x -1

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

decoder's state:

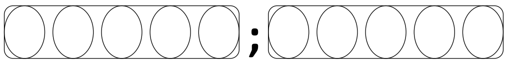


embedded[0] ; hidden[0]

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

decoder's state:



embedded[0] ; hidden[0]

```
self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
```

Decoder+Attention

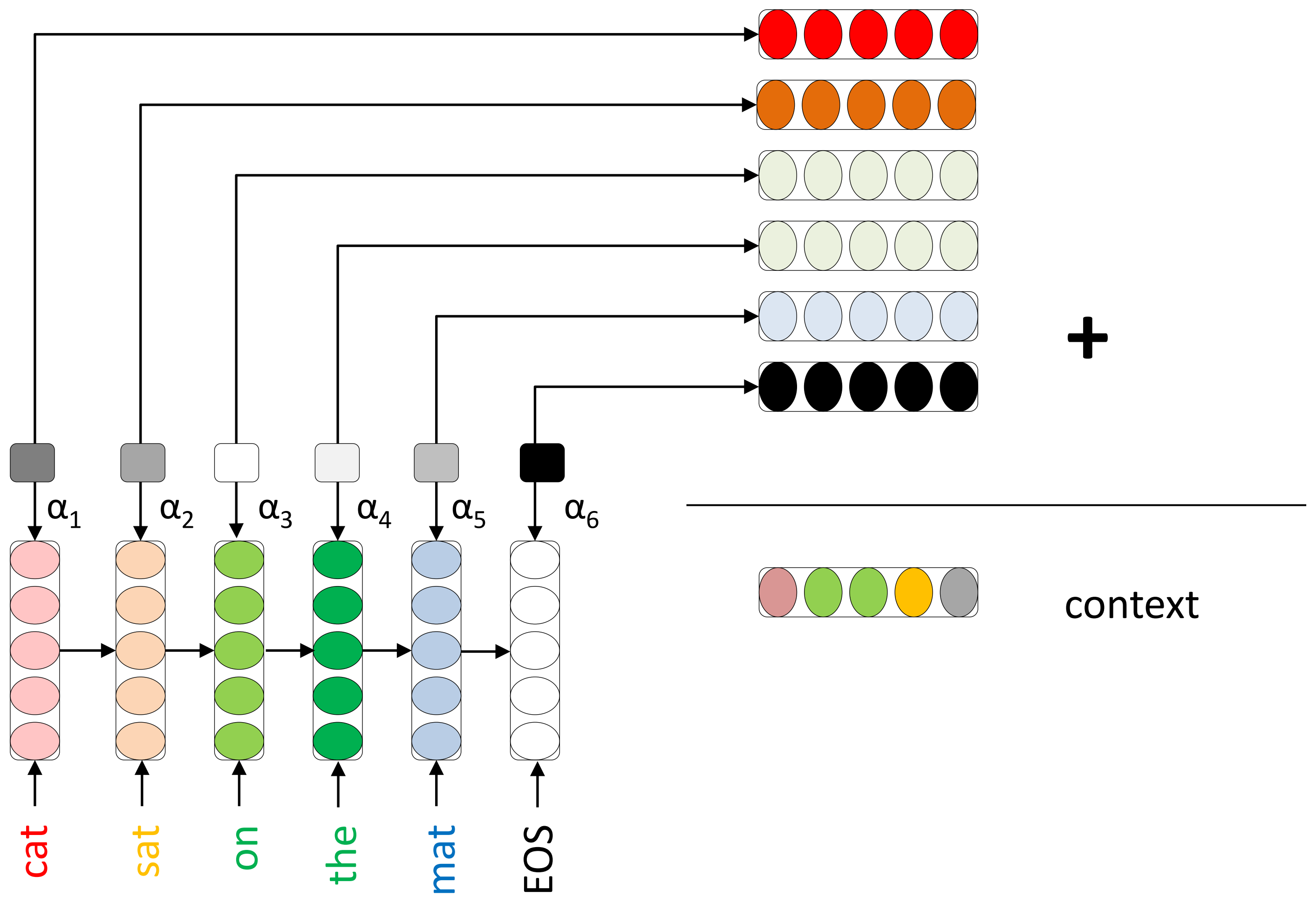

```python
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

```python
self.attn = nn.Linear(self.hidden_size * 2, self.max_length)
```

Softmax (`self.attn = nn.Linear(self.hidden_size * 2, self.max_length)`)

1 x max_length

Attention!



Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

1 x max_length

↓

unsqueeze(0)

↓

1 x 1 x max_length

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

1 x max_length x |embed|

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

context:

1 x 1 x |embed|

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}, \mathbf{c}\right),$$

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, EMNLP, 2014.

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

context:
1 x 1 x |embed|

```
self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size)
```

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

context:

1 x 1 x |embed|

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

context:

1 x 1 x |embed|

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

context:

1 x 1 x |embed|

Decoder+Attention


```
def forward(self, input, hidden, encoder_outputs):
    embedded = self.embedding(input).view(1, 1, -1)
    embedded = self.dropout(embedded)

    attn_weights = F.softmax(
        self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)
    attn_applied = torch.bmm(attn_weights.unsqueeze(0),
                             encoder_outputs.unsqueeze(0))

    output = torch.cat((embedded[0], attn_applied[0]), 1)
    output = self.attn_combine(output).unsqueeze(0)

    output = F.relu(output)
    output, hidden = self.gru(output, hidden)

    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights
```

context:

1 x 1 x |embed|

Decoder+Attention


```
    output = F.log_softmax(self.out(output[0]), dim=1)
    return output, hidden, attn_weights

def initHidden(self):
    result = Variable(torch.zeros(1, 1, self.hidden_size))
    if use_cuda:
        return result.cuda()
    else:
        return result
```

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

pair:
[[a, b, c], [a’, b’, c’, d’]]

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

training_pair[0]:
[a, b, c]
training_pair[1]:
[a’, b’, c’, d’]

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

training_pair[0]:
[a, b, c]
training_pair[1]:
[a’, b’, c’, d’]

Putting together


```
def trainIters(encoder, decoder, n_iters, print_every=1000, plot_every=100,
learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0  # Reset every print_every
    plot_loss_total = 0  # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)
    training_pairs = [variablesFromPair(random.choice(pairs))
                      for i in range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_variable = training_pair[0]
        target_variable = training_pair[1]

        loss = train(input_variable, target_variable, encoder,
                     decoder, encoder_optimizer, decoder_optimizer, criterion)
        print_loss_total += loss
        plot_loss_total += loss
```

training_pair[0]:
[a, b, c]
training_pair[1]:
[a’, b’, c’, d’]

Putting together


```
def train(input_variable, target_variable, encoder, decoder, encoder_optimizer,
decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_variable.size()[0]
    target_length = target_variable.size()[0]

    encoder_outputs = Variable(torch.zeros(max_length, encoder.hidden_size))
    encoder_outputs = encoder_outputs.cuda() if use_cuda else encoder_outputs

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_variable[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0][0]

    decoder_input = Variable(torch.LongTensor([[SOS_token]]))
    decoder_input = decoder_input.cuda() if use_cuda else decoder_input
```

Putting together

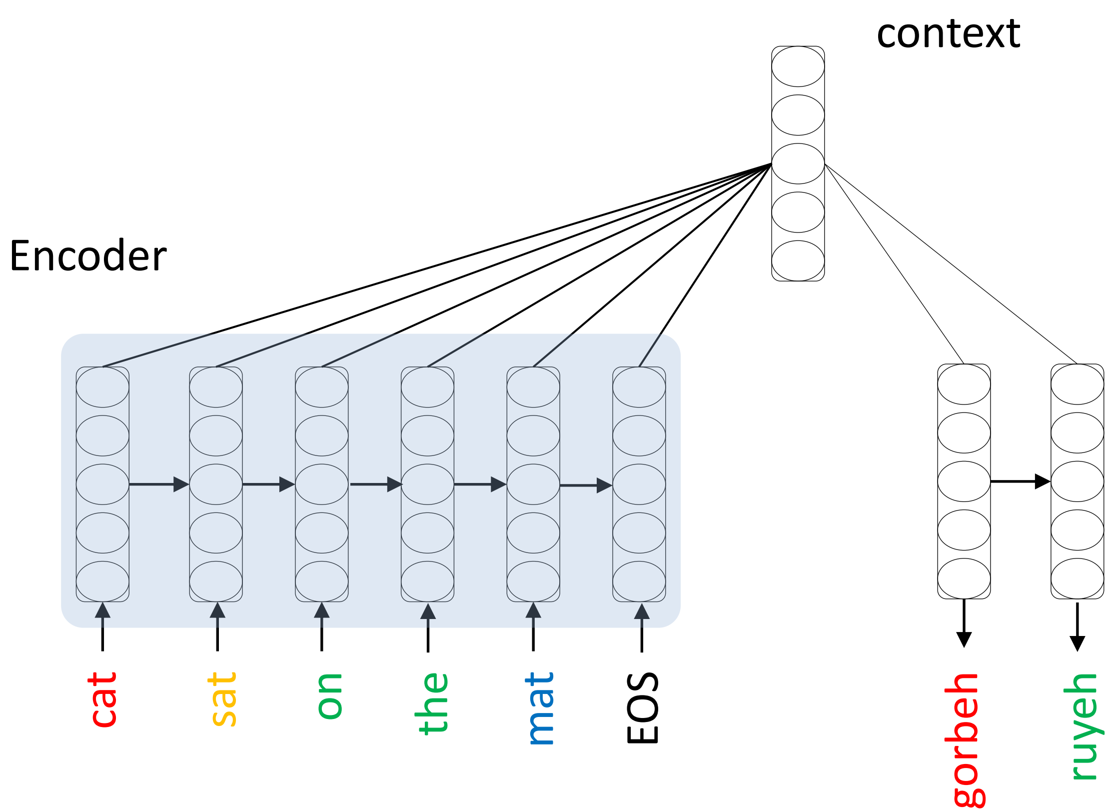

```
def train(input_variable, target_variable, encoder, decoder, encoder_optimizer,
decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_variable.size()[0]
    target_length = target_variable.size()[0]

    encoder_outputs = Variable(torch.zeros(max_length, encoder.hidden_size))
    encoder_outputs = encoder_outputs.cuda() if use_cuda else encoder_outputs

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_variable[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0][0]

    decoder_input = Variable(torch.LongTensor([[SOS_token]]))
    decoder_input = decoder_input.cuda() if use_cuda else decoder_input
```

training_pair[0]:
[a, b, c]
training_pair[1]:
[a’, b’, c’, d’]

Putting together


```
def train(input_variable, target_variable, encoder, decoder, encoder_optimizer,
decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_variable.size()[0]
    target_length = target_variable.size()[0]

    encoder_outputs = Variable(torch.zeros(max_length, encoder.hidden_size))
    encoder_outputs = encoder_outputs.cuda() if use_cuda else encoder_outputs

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_variable[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0][0]

    decoder_input = Variable(torch.LongTensor([[SOS_token]]))
    decoder_input = decoder_input.cuda() if use_cuda else decoder_input
```

Decoding



Putting together


```
def train(input_variable, target_variable, encoder, decoder, encoder_optimizer,
decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_variable.size()[0]
    target_length = target_variable.size()[0]

    encoder_outputs = Variable(torch.zeros(max_length, encoder.hidden_size))
    encoder_outputs = encoder_outputs.cuda() if use_cuda else encoder_outputs

    loss = 0

    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_variable[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0][0]

    decoder_input = Variable(torch.LongTensor([[SOS_token]]))
    decoder_input = decoder_input.cuda() if use_cuda else decoder_input
```

training_pair[0]:
[a, b, c]
training_pair[0][0]:
[a]
word embedding

Putting together


```
decoder_hidden = encoder_hidden

use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_variable[di])
        decoder_input = target_variable[di]  # Teacher forcing
else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.data.topk(1)
        ni = topi[0][0]

        decoder_input = Variable(torch.LongTensor([[ni]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        loss += criterion(decoder_output, target_variable[di])
        if ni == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.data[0] / target_length
```

init
the
decoder!

Putting together


```python
decoder_hidden = encoder_hidden

use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_variable[di])
        decoder_input = target_variable[di]  # Teacher forcing
else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.data.topk(1)
        ni = topi[0][0]

        decoder_input = Variable(torch.LongTensor([[ni]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        loss += criterion(decoder_output, target_variable[di])
        if ni == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.data[0] / target_length
```

Putting together


```
decoder_hidden = encoder_hidden

use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_variable[di])
        decoder_input = target_variable[di]  # Teacher forcing
else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.data.topk(1)
        ni = topi[0][0]

        decoder_input = Variable(torch.LongTensor([[ni]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        loss += criterion(decoder_output, target_variable[di])
        if ni == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.data[0] / target_length
```

Putting together


```
decoder_hidden = encoder_hidden

use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_variable[di])
        decoder_input = target_variable[di]  # Teacher forcing
else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.data.topk(1)
        ni = topi[0][0]

        decoder_input = Variable(torch.LongTensor([[ni]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        loss += criterion(decoder_output, target_variable[di])
        if ni == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.data[0] / target_length
```

Putting together


```
decoder_hidden = encoder_hidden

use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_variable[di])
        decoder_input = target_variable[di]  # Teacher forcing
else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.data.topk(1)
        ni = topi[0][0]

        decoder_input = Variable(torch.LongTensor([[ni]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        loss += criterion(decoder_output, target_variable[di])
        if ni == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.data[0] / target_length
```

Putting together


```
decoder_hidden = encoder_hidden

use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_variable[di])
        decoder_input = target_variable[di]  # Teacher forcing
else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.data.topk(1)
        ni = topi[0][0]

        decoder_input = Variable(torch.LongTensor([[ni]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        loss += criterion(decoder_output, target_variable[di])
        if ni == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.data[0] / target_length
```

Putting together


```
decoder_hidden = encoder_hidden

use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        loss += criterion(decoder_output, target_variable[di])
        decoder_input = target_variable[di]  # Teacher forcing
else:
    # Without teacher forcing: use its own predictions as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.data.topk(1)
        ni = topi[0][0]

        decoder_input = Variable(torch.LongTensor([[ni]]))
        decoder_input = decoder_input.cuda() if use_cuda else decoder_input

        loss += criterion(decoder_output, target_variable[di])
        if ni == EOS_token:
            break

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.data[0] / target_length
```

Content

- **Background: Machine Translation and Neural Network**
- **Transition: From Discrete Spaces to Continuous Spaces**
- **Neural Machine Translation: MT in a Continuous Space**
- **Implementing Seq2Seq models with PyTorch**
- **Conclusion**

Conclusion

- MT is a task defined in a discrete space
- In a deep learning framework, the MT is converted to a task defined in a continuous space
- Word embedding is used to map a word to a vector
- Recurrent Neural Network is used to model the word sequence
- Encoder-Decoder (or Sequence-to-Sequence) model is proposed for neural machine translation
- Attention-based mechanism is used to provide soft alignment for NMT
- NMT has outperformed SMT and still has huge potential

Further topics

- Subword level and character level models
  - Morphologically rich languages
  - Out-of-Vocabulary problem
- Multitask and Multiway models
  - Sharing parameters among Multiple MT models
  - Low resource or zero-shot language pairs
- Pure attention models
  - Higher performance

Thanks

Q&A

Speaker: Qun Liu

Email: qun.liu@dcu.ie

Speaker: Peyman Passban

Email: pe.psbn@gmail.com