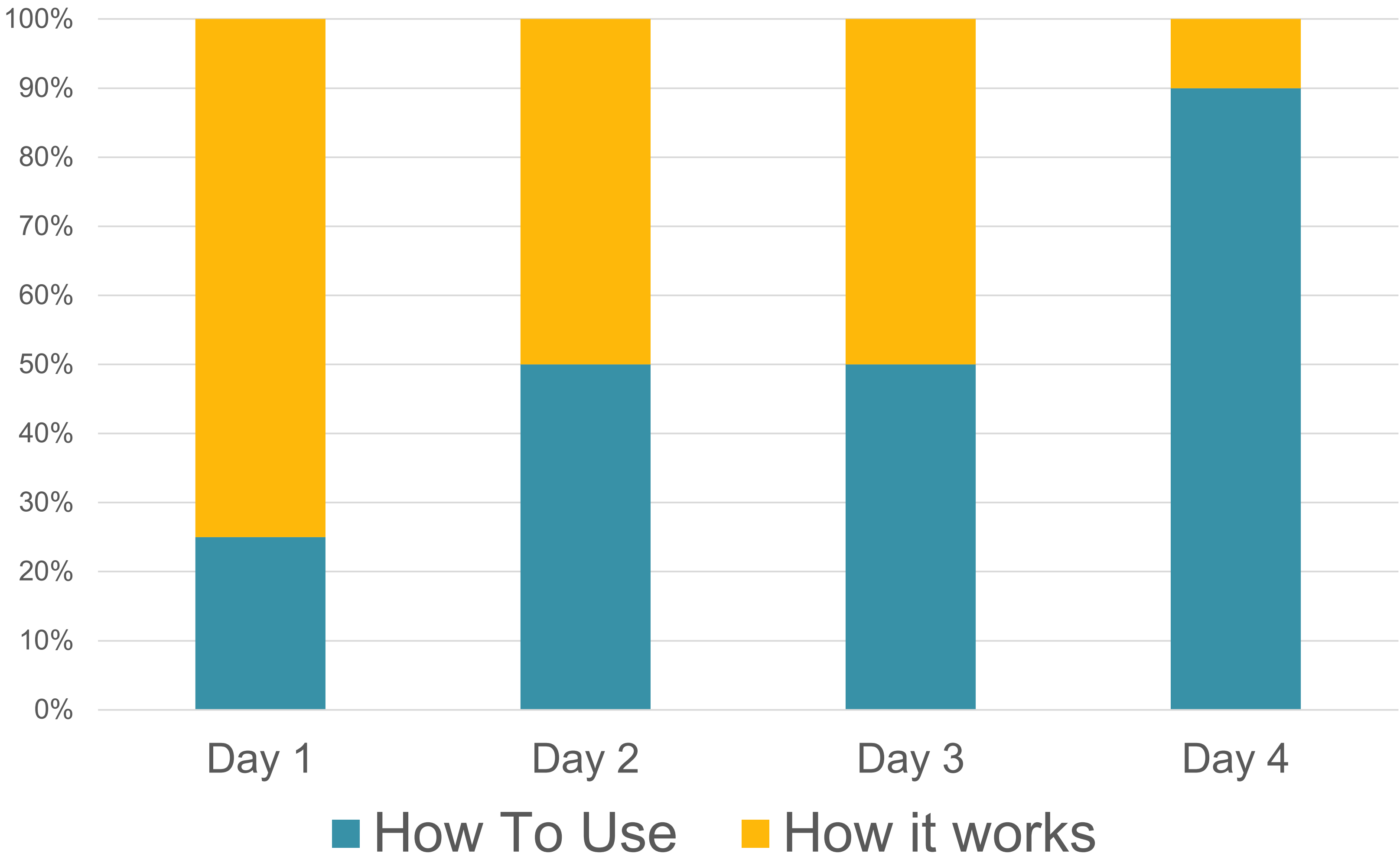
Text Processing Using Machine Learning

Classic DNN for Text Processing

Dr Wang Aobo
aobo.wang@nus.edu.sg

TPML Agenda


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Day 1
Day 2
Day 3
Day 4
How To Use
How it works

Agenda

- Data Splits and Evaluation
- Evaluation and Optimization
  - Over-/Underfitting
  - Regularization and Dropout
- CNN for Text Classification
  - Convolutional Kernels for Text
- **Workshop**
- RNN and LSTM
  - RNN text Encoder
  - LSTM for Text Processing
- SeqtoSeq model
  - Encoder Decoder Model
- **Workshop**

What can NLP do?

2 basic Use Cases:

1. **Automatically put text into categories- Classification**
   - Sentiment detection
   - Spam email detection
   - Emotion detection

2. **Extract specific information from the text- Extraction**
   - Named Entity extraction from sentence

What can NLP do?

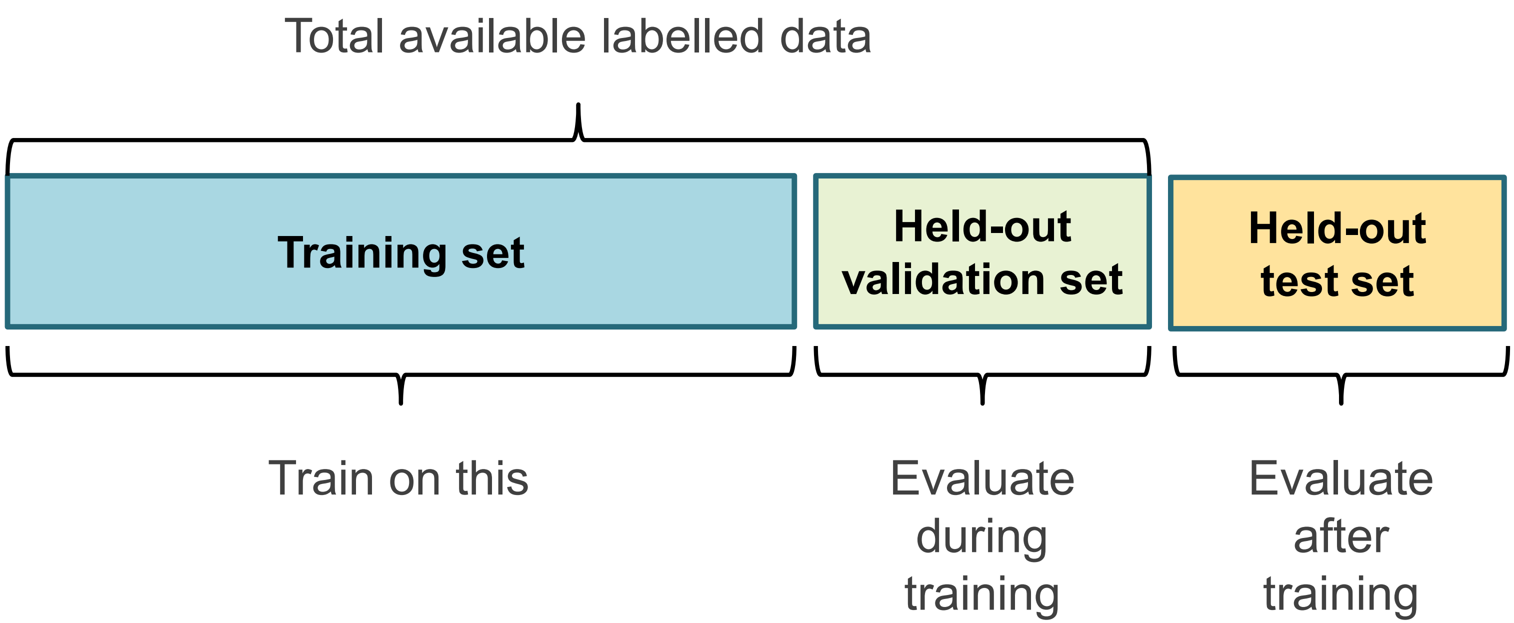
Other *Fancier* Use Cases:

1. **Object Classification/Clustering & Recommendation**
2. **Search Engine**
3. **Question Answering System**
4. **Voice Assistant**
5. **Machine Translation**
6. **Grammar Error Correction & Language Learning**
7. **Chatting Robots**
8. **"*Fake Articles*" Generation & Detection**
9. **....**

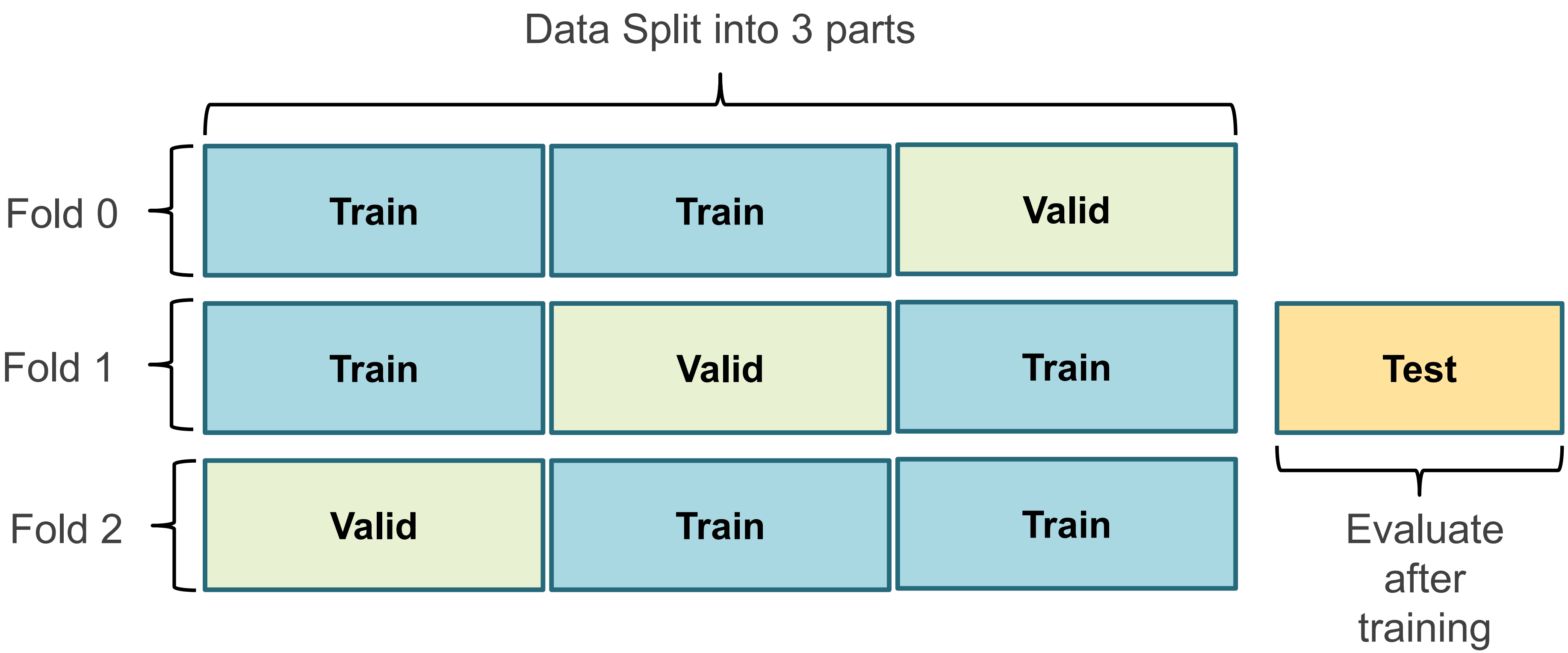
Data Splits

DNN are always Supervised, except for Reinforcement Learning

Hold-Out Evaluation



K-Fold Cross Validation Evaluation



K-Fold Cross Validation Evaluation

```
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.model_selection import KFold
>>> import torch

>>> x, y = torch.rand(10, 2).numpy(), torch.rand(10).numpy()
>>> print(x.shape, y.shape)
(10, 2) (10,)

# Split out the held-out test set first.
>>> split = 0.2
>>> x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=split)

>>> kf = KFold(n_splits=3)
>>> three_folds = {i:{'x_train': x_train[train_idx], 'x_valid': x_train[valid_idx],
...                   'y_train': x_train[train_idx], 'y_valid': x_train[valid_idx]}
...                for i, (train_idx, valid_idx) in enumerate(kf.split(x_train))}
```

Evaluation & Optimization

REGULARIZATION AND DROPOUT

Bias-Variance Tadeoff

- "*A **small network**, with say one hidden unit **is likely to be biased**, since the repertoire of available functions spanned by $f(x, w)$ over allowable weights will in this case be quite limited.*"

- "*if we **overparameterize**, via a large number of hidden units and associated weights, then bias will be reduced (… **with enough weights and hidden units, the network will interpolate the data**) but there is then the **danger of significant variance** contribution to the mean-square error*"

(German et al. 1992)

Over-/Underfitting

*"I believe that only **out-of-date** methods such as NaïveBayes, LogR, SVM are facing under-/overfitting problems.*

***Latest** Deep Learning methods have no such problems, thus dominating in world of Machine Learning."*

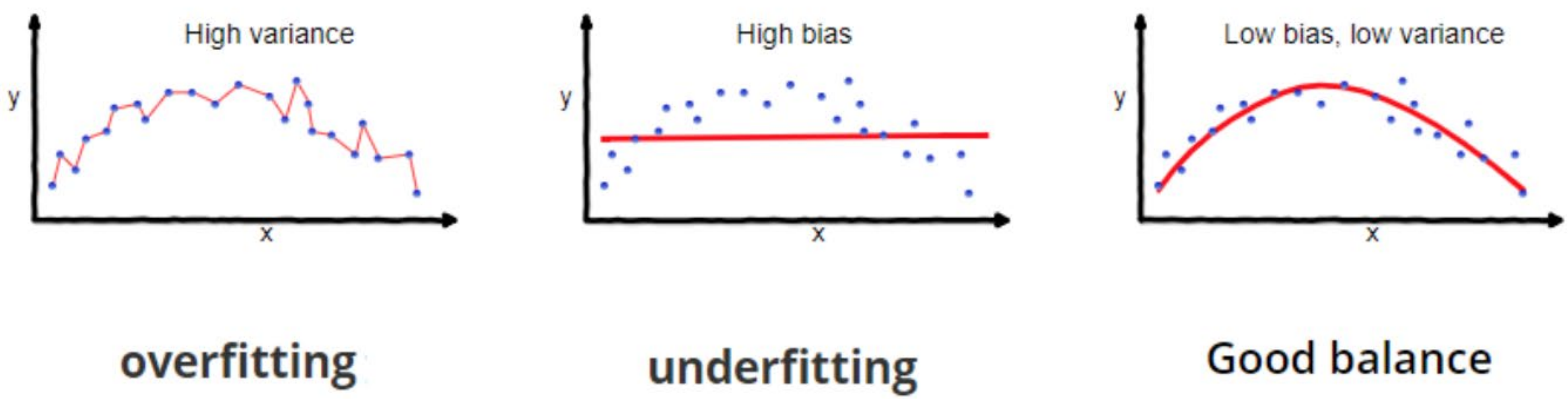


Image from https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

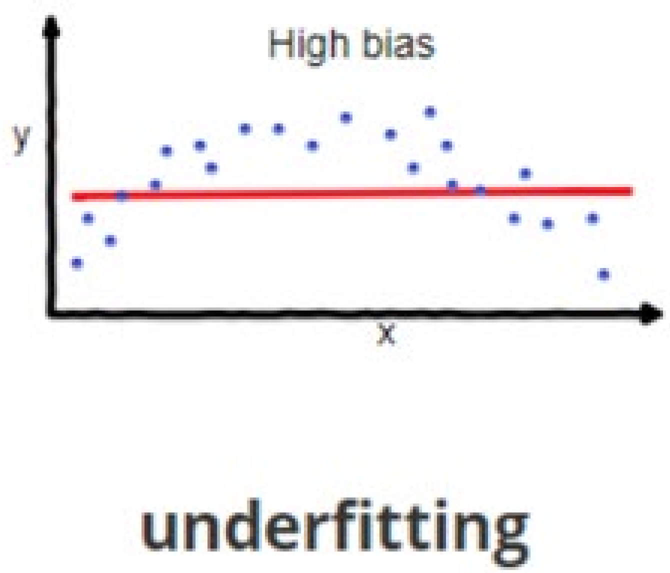
Optimization and Generalization

- **Optimization:** *process of adjusting a model to get the best performance possible on the training data (train and valid data)*

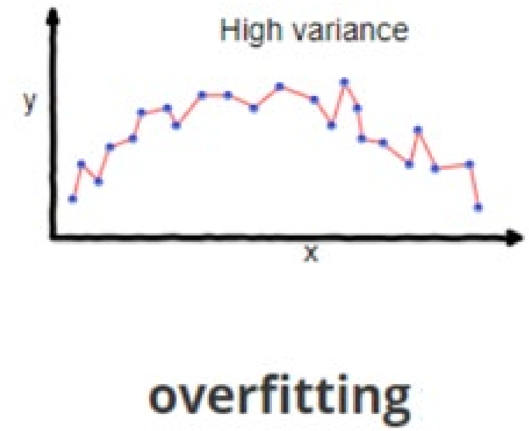
- **Generalization:** *how well trained model performs on data it has never seen before (test data)*

Overcoming Underfitting

- **What to do when model is underfitting (not optimal)?**
  - Train longer
  - Increase the complexity of models (# of Layers, # of neuros per layer)
  - Improve the data quality by removing the noisy data



Overcoming Overfitting



- **Reducing complexity of the model**
  - i.e. no. of layers and no. of units per layer

- **Weights regularization** (i.e. adding a cost associated with having large weights) put constraints on complexity of the model by forcing its weights to take small values

- **Dropping out** (i.e. randomly setting activated outputs to zero) is effective in regularizing the model

Weights regularization

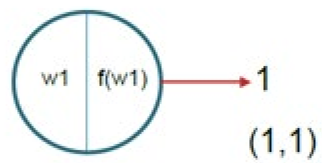
- A network with **large** network **weights** can be a sign of an **unstable** network where small changes in the input can lead to large changes in the output

	Like x1	Hate x2	Good x3	Enjoy x4	Bad x5
Like	1	0	0	0	0
Hate	0	1	0	0	0
Good	0	0	1	0	0
Enjoy	0	0	0	1	0
Bad	0	0	0	0	1

(1,5)

weight
6
-6
6
6
-6

(5,1)



- **Encourage** the network to keep the **weights small**

Weights regularization

- **Cost function = Loss (say, binary cross entropy) + Regularization term**

- **Due to the addition of this regularization term, the values of weight matrices decrease**

- **L1:** $$Cost\ function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

  – the weights may be reduced to zero

- **L2:** $$Cost\ function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

  – forces the weights to decrease towards zero (but not exactly zero).

Weights regularization

- Without any regularization
- When w=2, we get the minimum loss

Gradient Descent

- Minimize a "fake" lost function

$$L = w^2 - 4w + 6$$

$$\frac{dL}{dw} = 2w - 4$$

- Apply Delta rule

$$w^{new} = w^{old} - \eta \frac{dL}{dw}$$

$\eta = 0.3 \quad w_{init} = 3$

- Iterations

$$w^{new} = 3.00 - 0.3(2 * 3.00 - 4) = 2.40$$
$$w^{new} = 2.40 - 0.3(2 * 2.40 - 4) = 2.16$$
$$w^{new} = 2.16 - 0.3(2 * 2.16 - 4) = 2.06$$
$$w^{new} = 2.06 - 0.3(2 * 2.06 - 4) = 2.02$$
$$w^{new} = 2.02 - 0.3(2 * 2.02 - 4) = 2.00$$
$$w^{new} = 2.00 - 0.3(2 * 2.00 - 4) = 2.00$$
$$w^{new} = 2.00 - 0.3(2 * 2.00 - 4) = 2.00$$

Weights regularization

- **Apply L1:** $Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$

$L = w^2 - 4w + 6$ Let $\frac{\lambda}{2m} = 2$

$$L = w^2 - 4w + 6 + 2w$$

$$\frac{dL}{dw} = 2w - 4 + 2 = 2w-2$$

- **Apply Delta rule**

$$w^{new} = w^{old} - \eta \frac{dL}{dw}$$

$\eta = 0.3$ $w_{init} = 3$

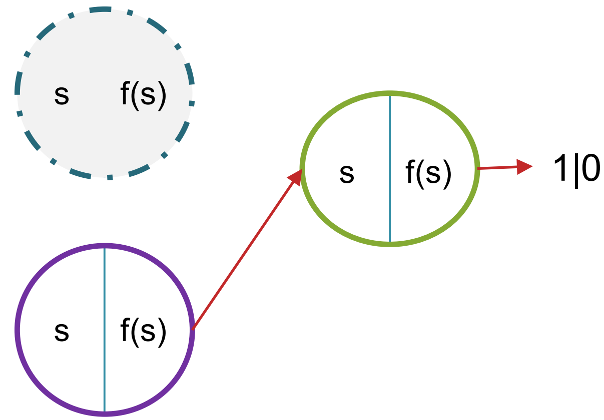
- **Iterations**

$$w^{new} = 3.00 - 0.3(2 * 3.00 - 2) = 1.80$$
$$w^{new} = 1.80 - 0.3(2 * 1.80 - 2) = 1.32$$
$$w^{new} = 1.32 - 0.3(2 * 1.32 - 2) = 1.13$$
$$w^{new} = 1.13 - 0.3(2 * 1.13 - 2) = 1.05$$
$$w^{new} = 1.05 - 0.3(2 * 1.05 - 2) = 1.02$$
$$w^{new} = 1.02 - 0.3(2 * 1.02 - 2) = 1.01$$
$$w^{new} = 1.01 - 0.3(2 * 1.01 - 2) = 1.00$$

w may be reduced to zero given different λ

Weights regularization

- Word level classification
  - let *n=26 << vocabulary_size*
  - If *wb=0* by **L1**
  - Model Complexity Reduced



	a x1	b x2	c x3	…	z x26
like	0	0	0	…	0
hate	1	0	0	…	0
good	0	0	0	…	0
enjoy	0	0	0	…	0
bad	0	1	0	…	0

(1,26)

weight	weight
0	wp1
0	wp2
0	wp3
…	…
0	wp26

(26,2)

weight
wg1
wg2

(2,1)

1|0

(1,1)

Weights regularization

- **Apply L2:** $Cost\ function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$

$$L = w^2 - 4w + 6 + w^2$$

$$\frac{dL}{dw} = 2w - 4 + 2w = 4w - 4$$

- **Apply Delta rule**

$$w^{new} = w^{old} - \eta \frac{dL}{dw}$$

$\eta = 0.3$ $\quad w_{init} = 3$

- **Iterations**

$$w^{new} = 3.00 - 0.3(4 * 3.00 - 4) = 0.60$$
$$w^{new} = 0.60 - 0.3(4 * 0.60 - 4) = 0.12$$
$$w^{new} = 0.12 - 0.3(4 * 0.12 - 4) = 1.66$$
$$w^{new} = 1.66 - 0.3(4 * 1.66 - 4) = 0.87$$
$$w^{new} = 0.87 - 0.3(4 * 0.87 - 4) = 1.03$$
$$w^{new} = 1.03 - 0.3(4 * 1.03 - 4) = 0.99$$
$$w^{new} = 0.99 - 0.3(4 * 0.99 - 4) = 1.00$$

w have no chance to be 0

Overcoming Overfitting

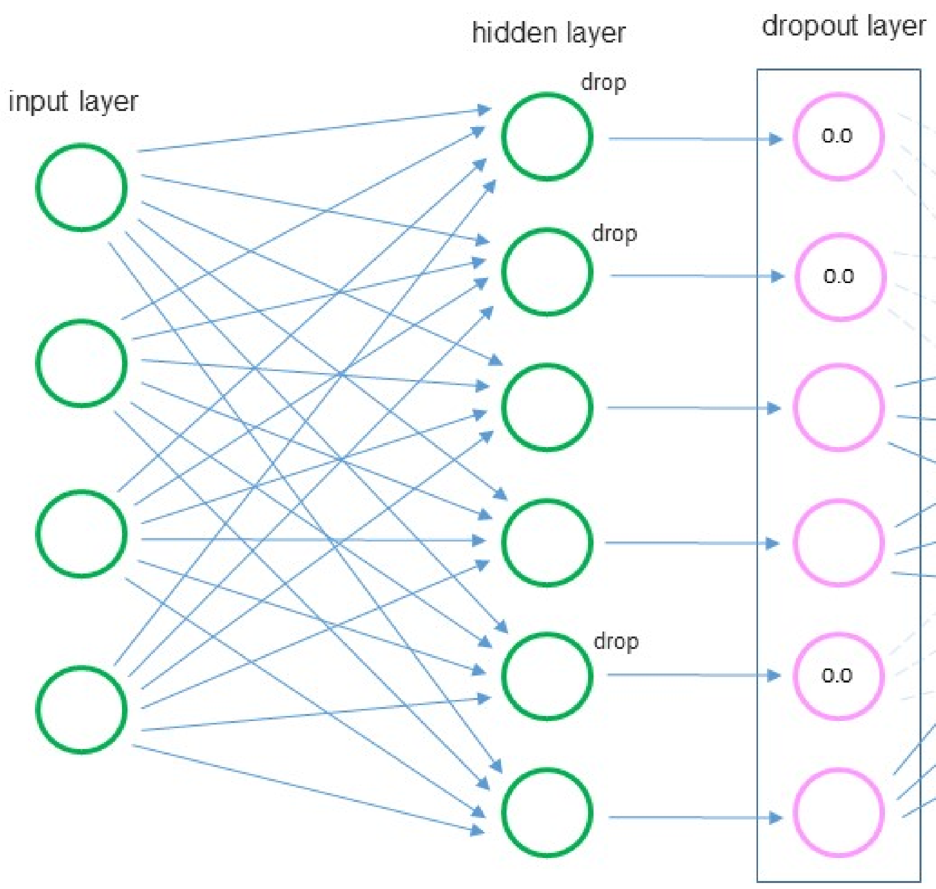
- **Reducing complexity of the model**
  - i.e. no. of layers and no. of units per layer

- **Weights regularization**
  - forcing its weights to take **small** values
  - L1 reduces no. of units per layer
  - **L2** is preferred when no need to simplify the model (as L1 does)
  - λ **usually takes {0.01 , 0.1 , 1}**

- **Dropping out** (i.e. randomly setting activated outputs to zero) is effective in regularizing the model

Dropout Layer

- **Prevent from Overfitting**

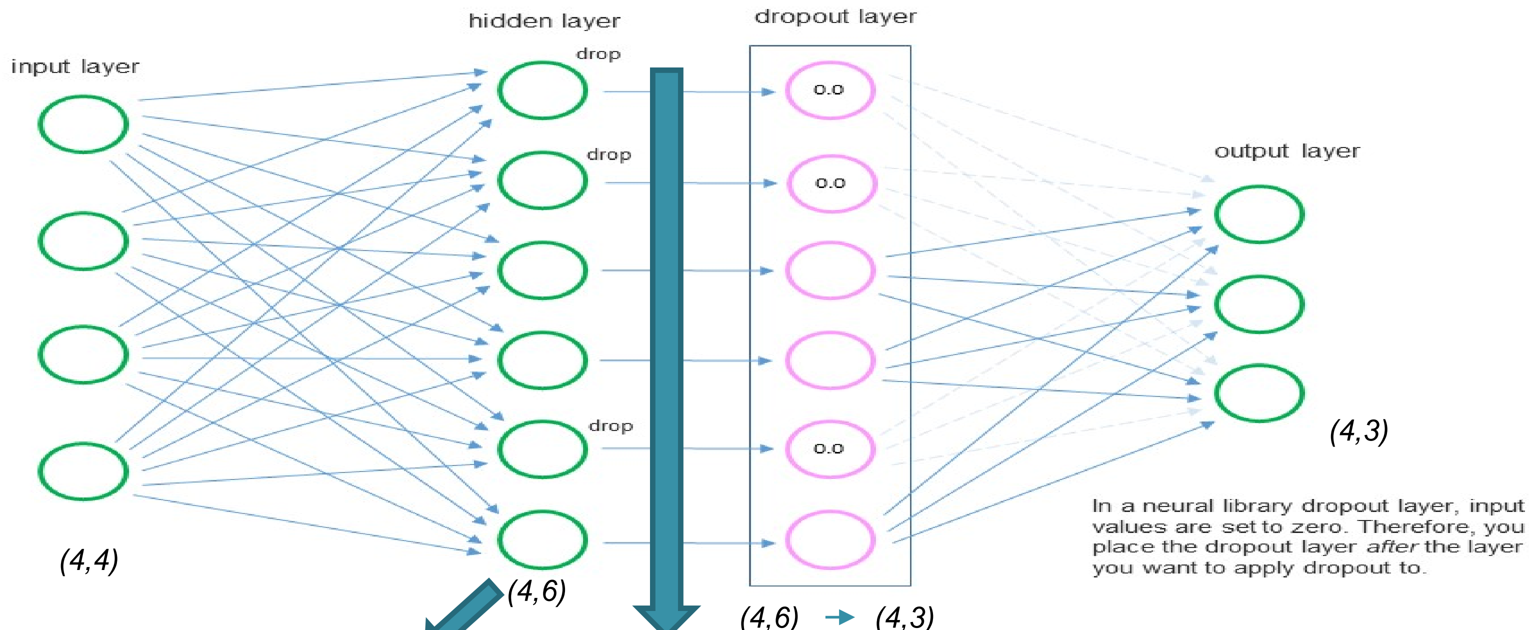


In a neural library dropout layer, input values are set to zero. Therefore, you place the dropout layer *after* the layer you want to apply dropout to.

Image from jamesmccaffrey

Dropout Layer

- **Prevent from Overfitting**



0.2	-1.1	0.3	-0.7	0.9	-0.4
0.9	-1.3	0.4	-0.9	0.9	-0.3
0.3	-3.0	0.5	-0.2	-0.1	0.3
0.5	-0.1	0.2	0.1	-0.5	0.5

(4,6)

·

0	0	0	0	0	0
0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	1

(6,6)

=

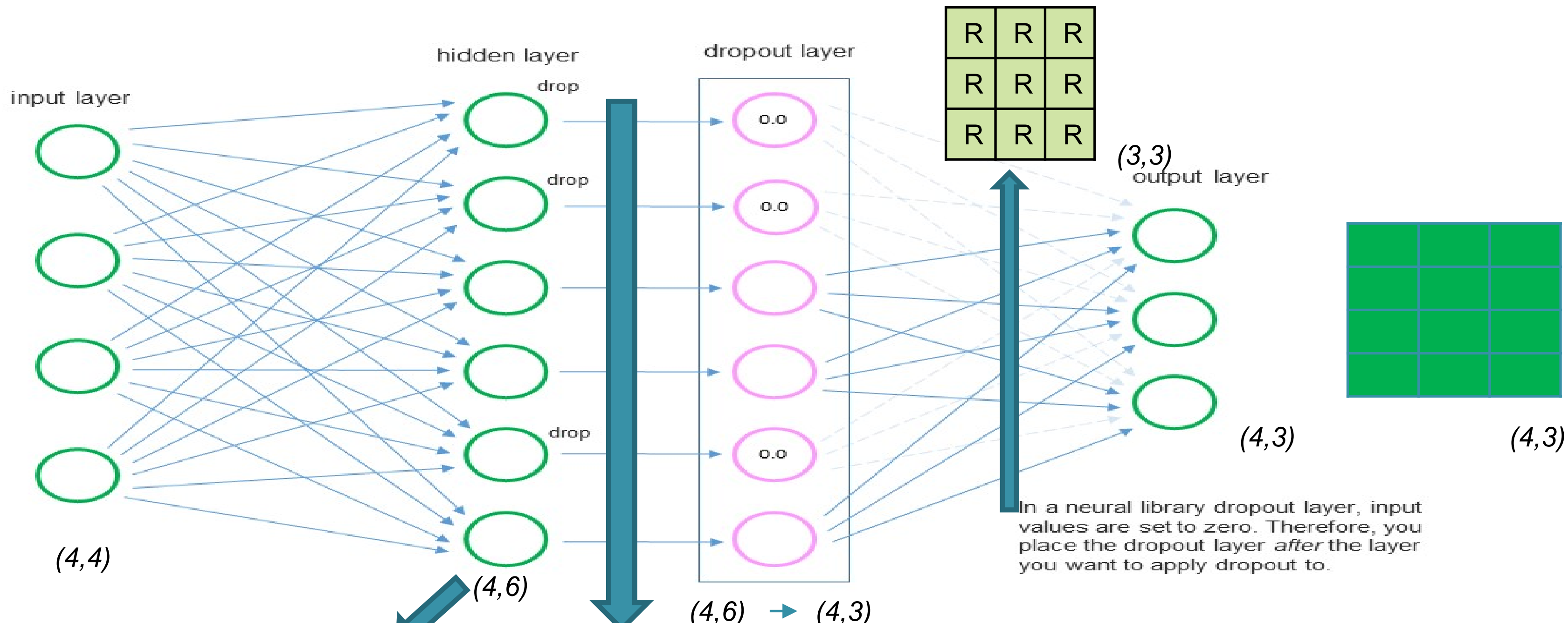
0	0	0.3	-0.7	0	-0.4
0	0	0.4	-0.9	0	-0.3
0	0	0.5	-0.2	0	0.3
0	0	0.2	0.1	0	0.5

(4,6)

P=3/6=0.5 in this case

Dropout Layer

- **Prevent from Overfitting**



0.2	-1.1	0.3	-0.7	0.9	-0.4
0.9	-1.3	0.4	-0.9	0.9	-0.3
0.3	-3.0	0.5	-0.2	-0.1	0.3
0.5	-0.1	0.2	0.1	-0.5	0.5

(4,6)

•

0	0	0	0	0	0
0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	1

(6,6)

›

0.3	-0.7	-0.4
0.4	-0.9	-0.3
0.5	-0.2	0.3
0.2	0.1	0.5

(4,3)

P=3/6=0.5
in this case

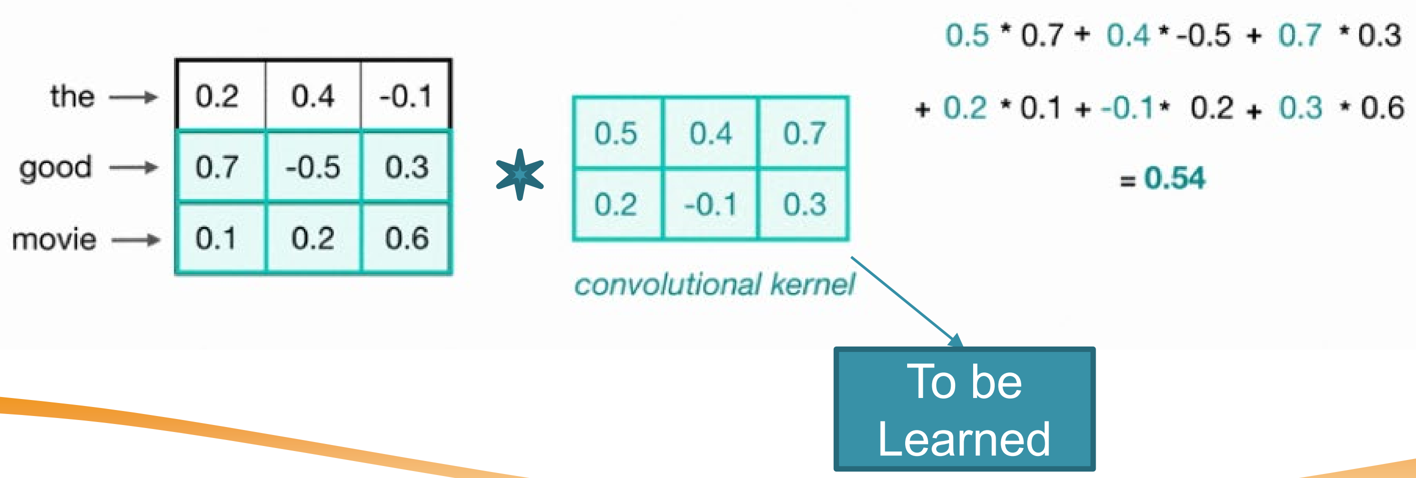
Agenda

- Data Splits and Evaluation
- Evaluation and Optimization
- **CNN for Text Classification**
  - **Convolutional Kernels for Text**
- Workshop
- RNN and LSTM
  - RNN text Encoder
  - LSTM for Text Processing
- SeqtoSeq model
  - Encoder Decoder Model
- Workshop

CNN for Text Classification

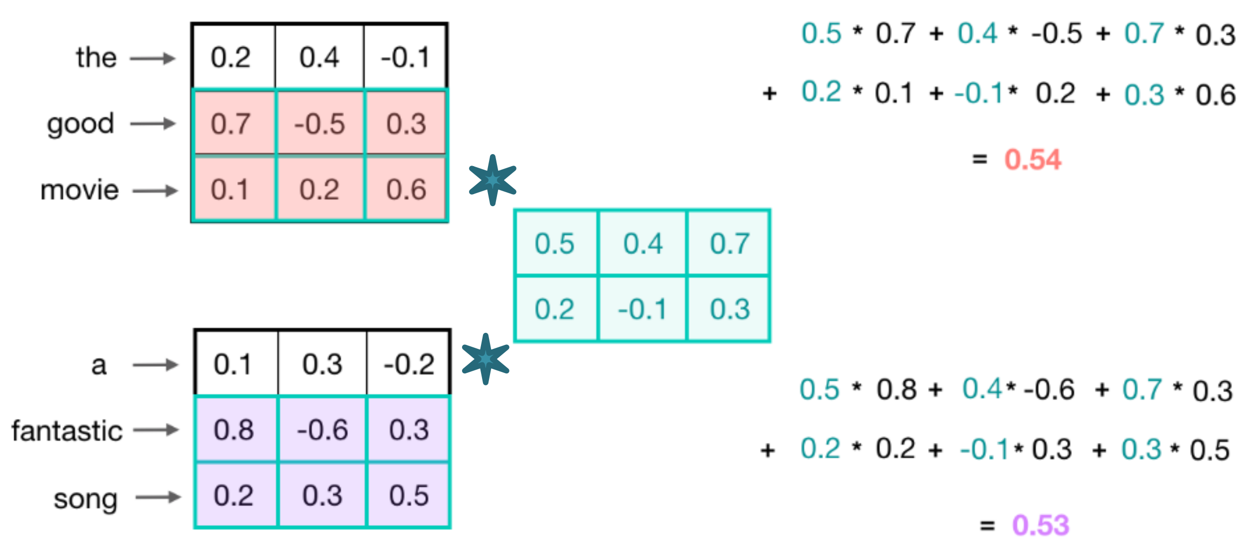
- **Kernel (Ngram feature extractor)**
  - Kernel Matrix (Ngram , length of wordEmbedding)
  - Sliding window towards the (**1-D**) direction of Text
  - **Weights** to be learned
  - **Not** ~~matrix multiplication (Dot Product)~~
  - But **Convolution**

$$\{ (\text{ngram, wordEmb}) * (\text{ngram,wordEmb}) \}_{\text{k times}} = (k,1)$$



CNN for Text Classification

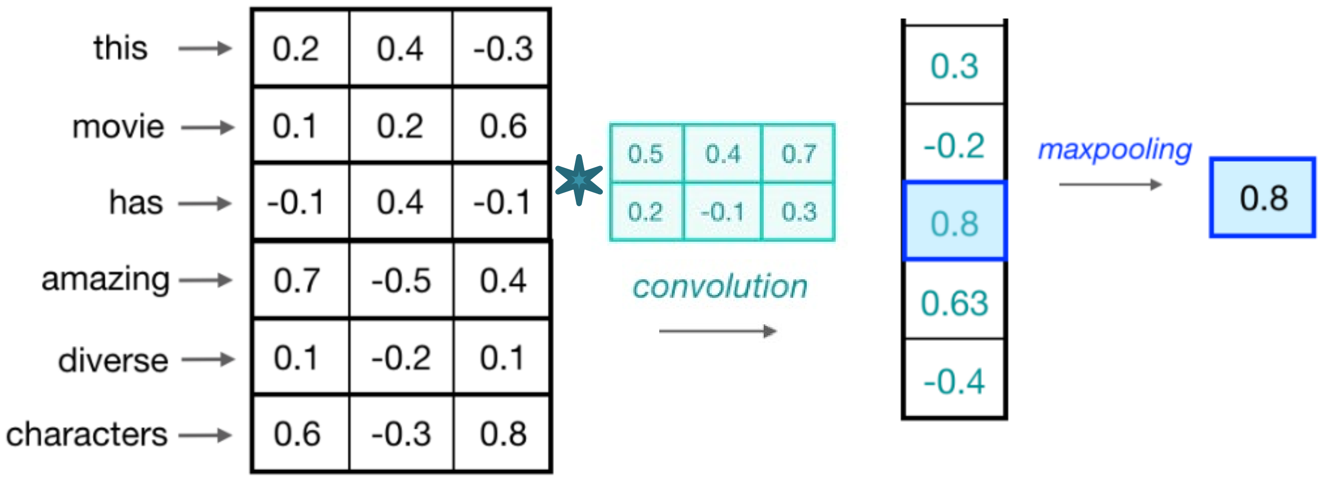
- **Kernel (Ngram feature extractor)**
  - Sliding window towards the (**1-D**) direction of Text
  - **Weights** to be learned
  - **Weights** to be shared



To be shared and learned

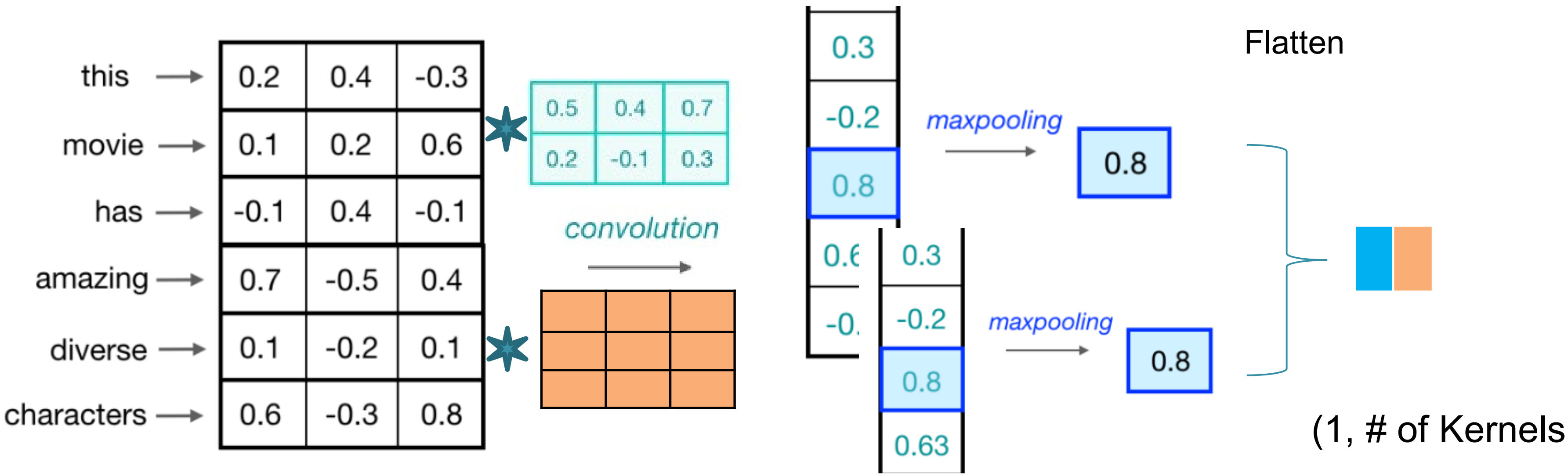
CNN for Text Classification

- **Kernel (Ngram feature extractor)**
  - Sliding window towards the (**1-D**) direction of Text
  - **Weights** to be learned and shared
  - **MaxPooling** the results (re-shaping the rows)



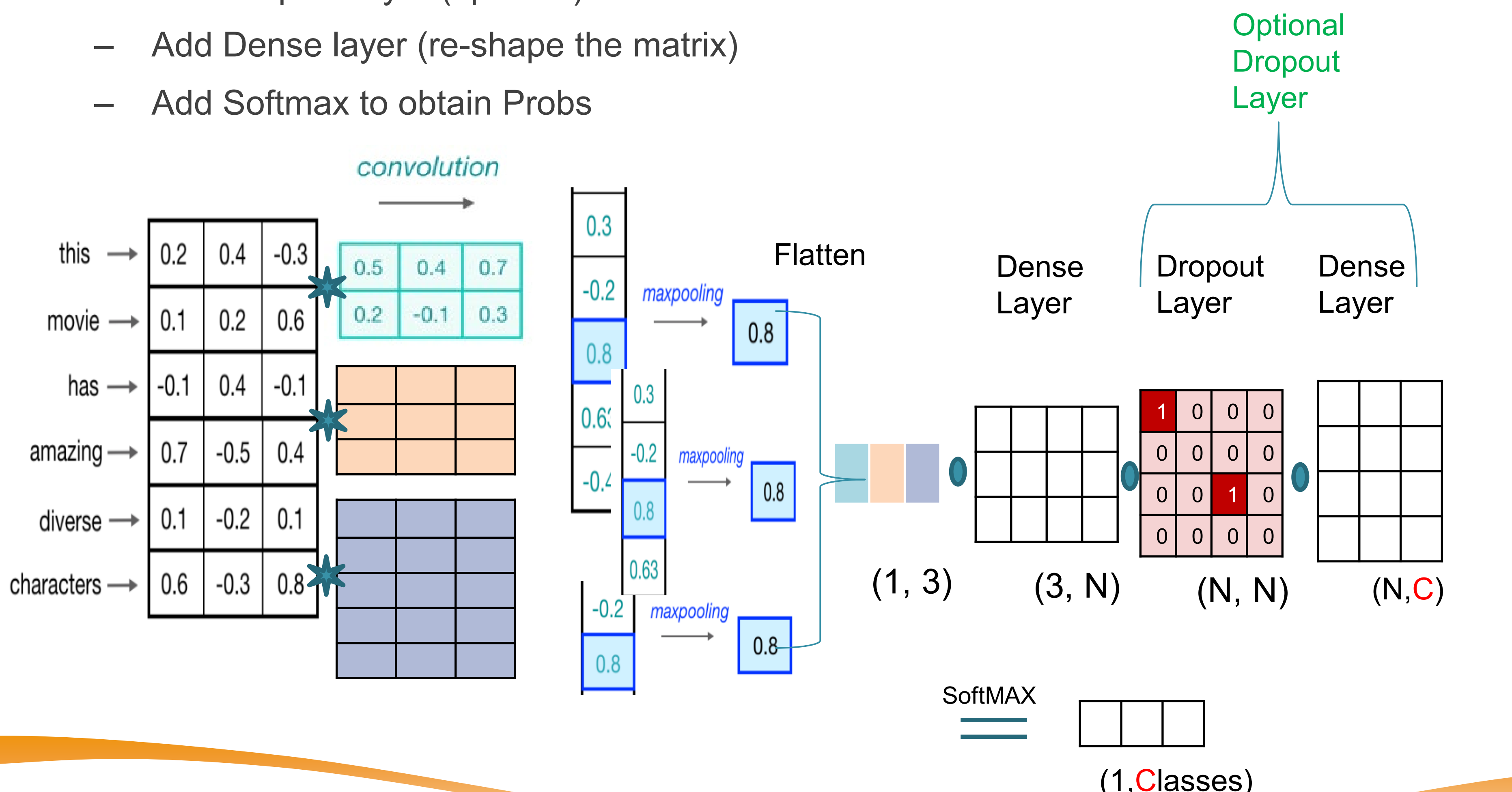
CNN for Text Classification

- **Multi-Kernels (Ngram feature extractors)**
  - Sliding window towards the (**1-D**) direction of Text
  - **Weights** to be learned and shared
  - **MaxPooling** the results (re-shaping the rows)
  - Concatenate (Flatten) results from different (Ngram) Kernels



CNN for Text Classification

- **Multi-Kernels (Ngram feature extractors)**
  - Concatenate (Flatten) results from different (Ngram) Kernels
  - Add Dropout layer (optional)
  - Add Dense layer (re-shape the matrix)
  - Add Softmax to obtain Probs



Workshop

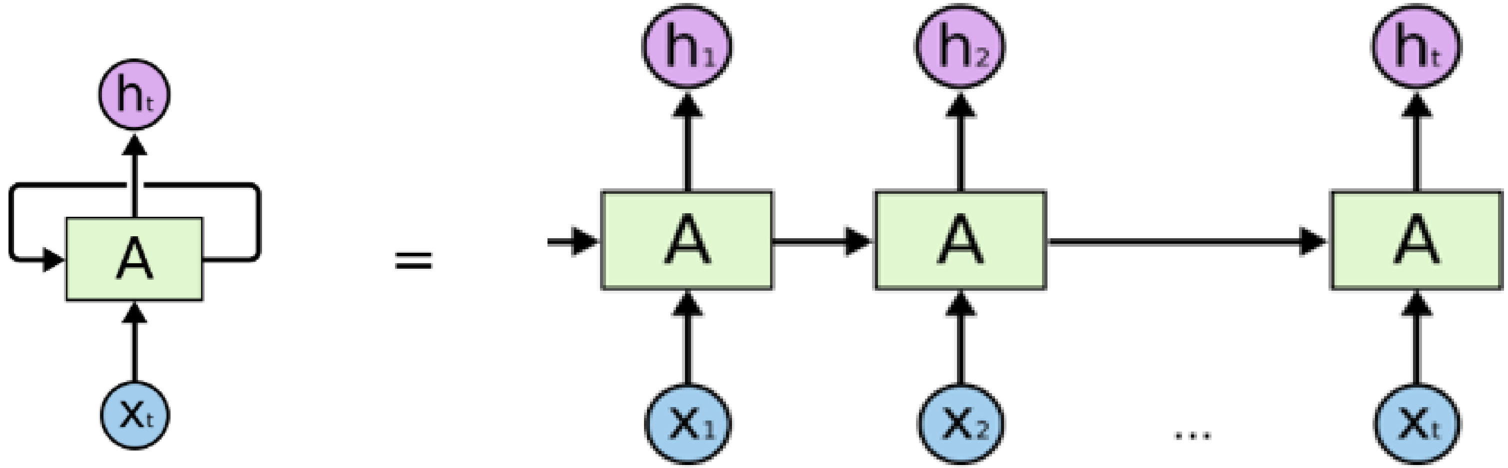
OPTIMIZATION AND CNN

Agenda

- Data Splits and Evaluation
- Evaluation and Optimization
- CNN for Text Classification
- Workshop
- **RNN and LSTM**
  - **RNN text Encoder**
  - **LSTM for Text Processing**
- SeqtoSeq model
  - Encoder Decoder Model
- Workshop

Recurrent Neural Networks

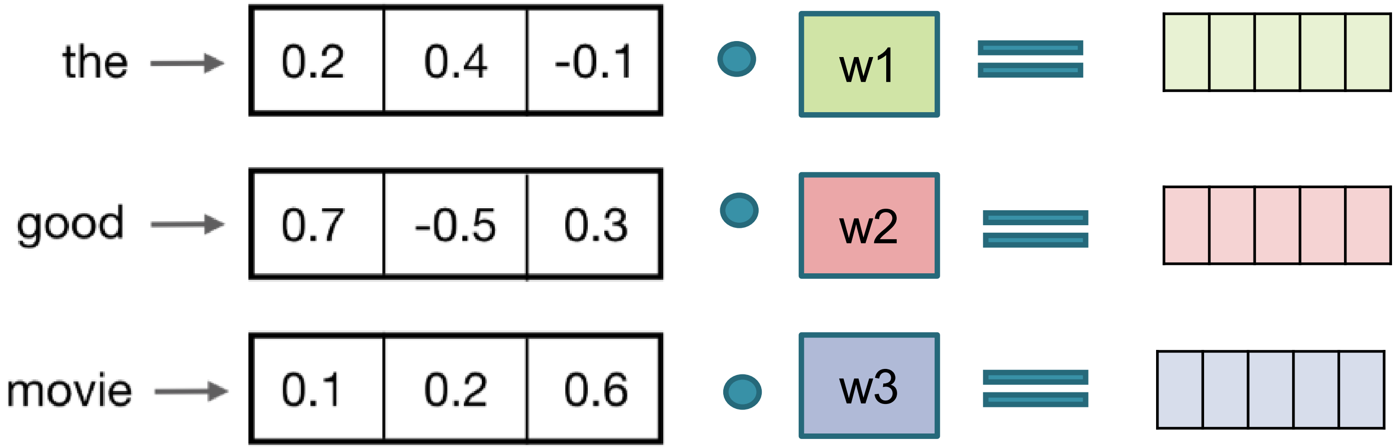
- Texts are always in Sequence. But CNN is not
- Where are the neurals and layers ?
- What's inside A?
- Why Loop? Why Equal ?



An unrolled recurrent neural network.

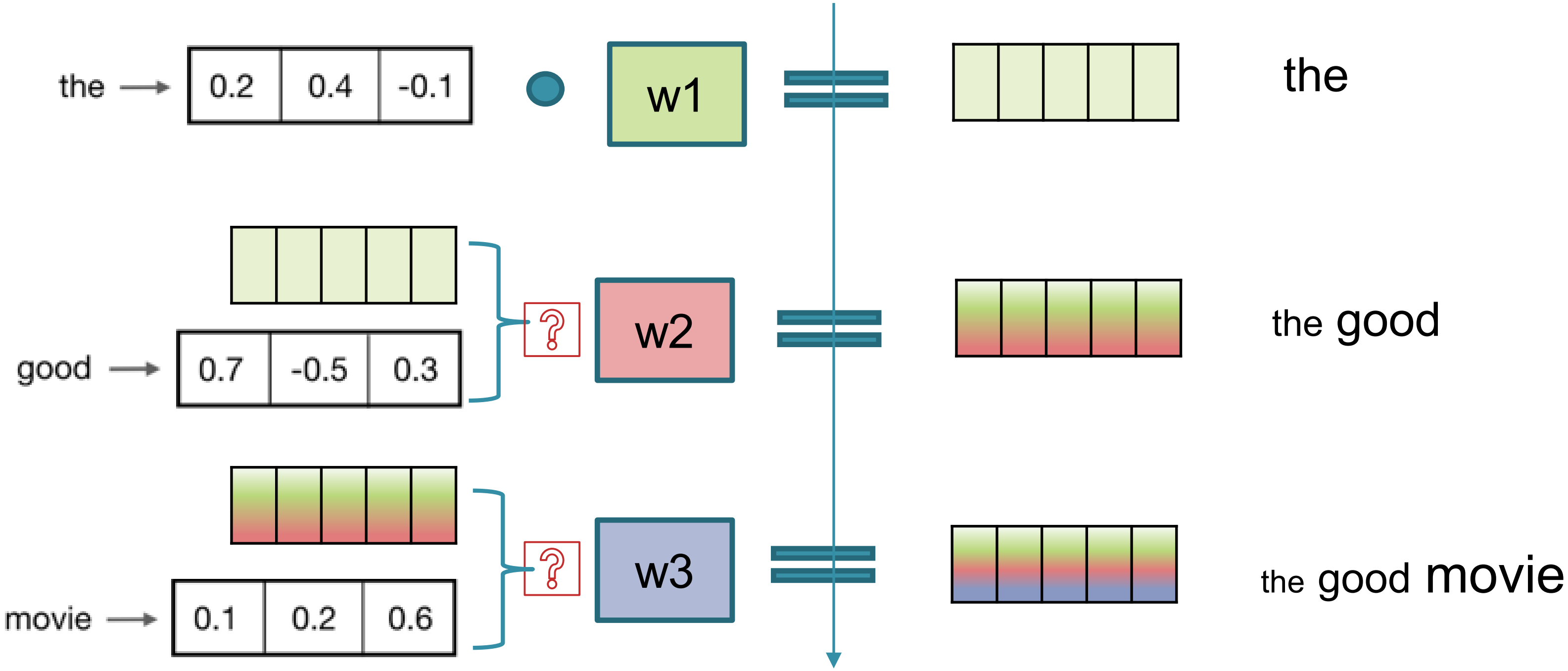
RNN Encoder

- A better (than Ngram) way to represent word/sentence
- Encode the sequence of word tokens (which CNN lacks of)



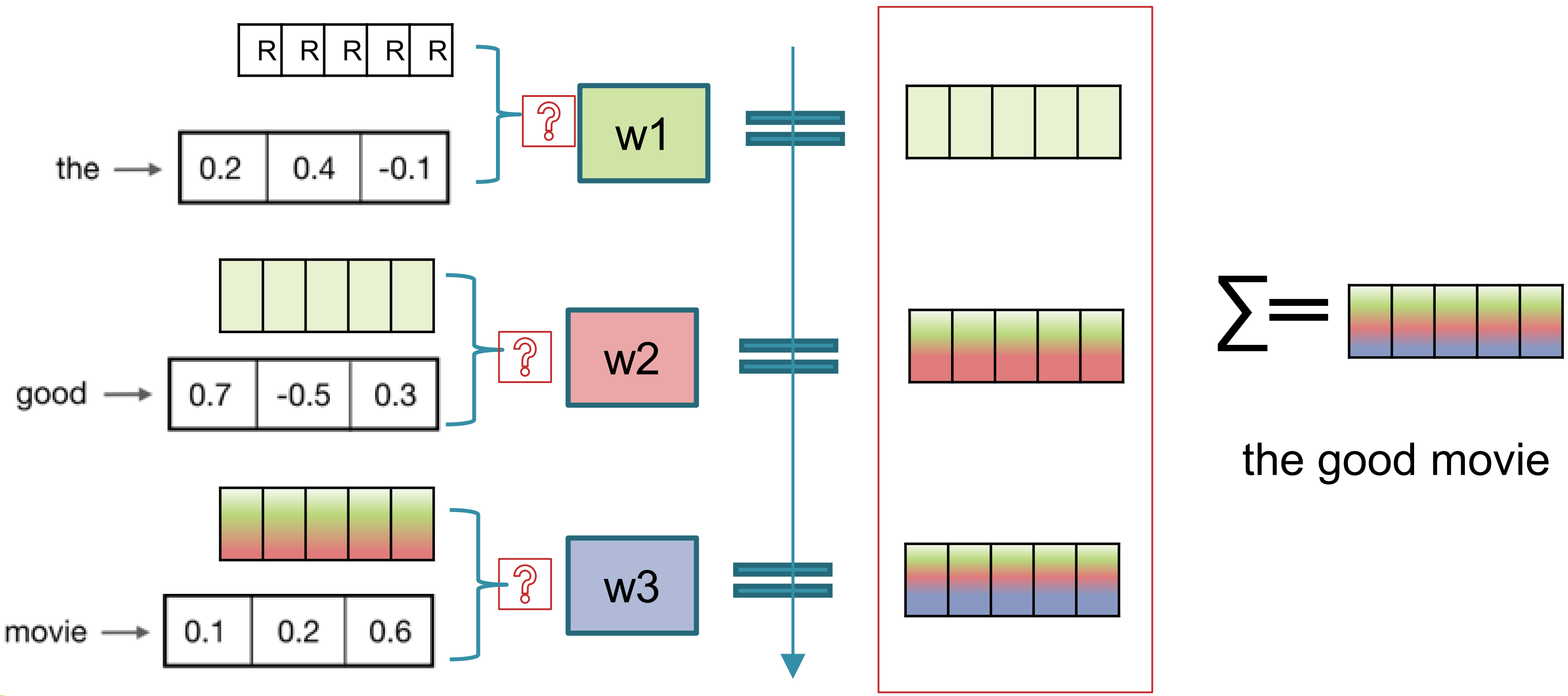
RNN Encoder

- A better (than Ngram) way to represent word/sentence
- Encode the sequence of word tokens (which CNN lacks of)



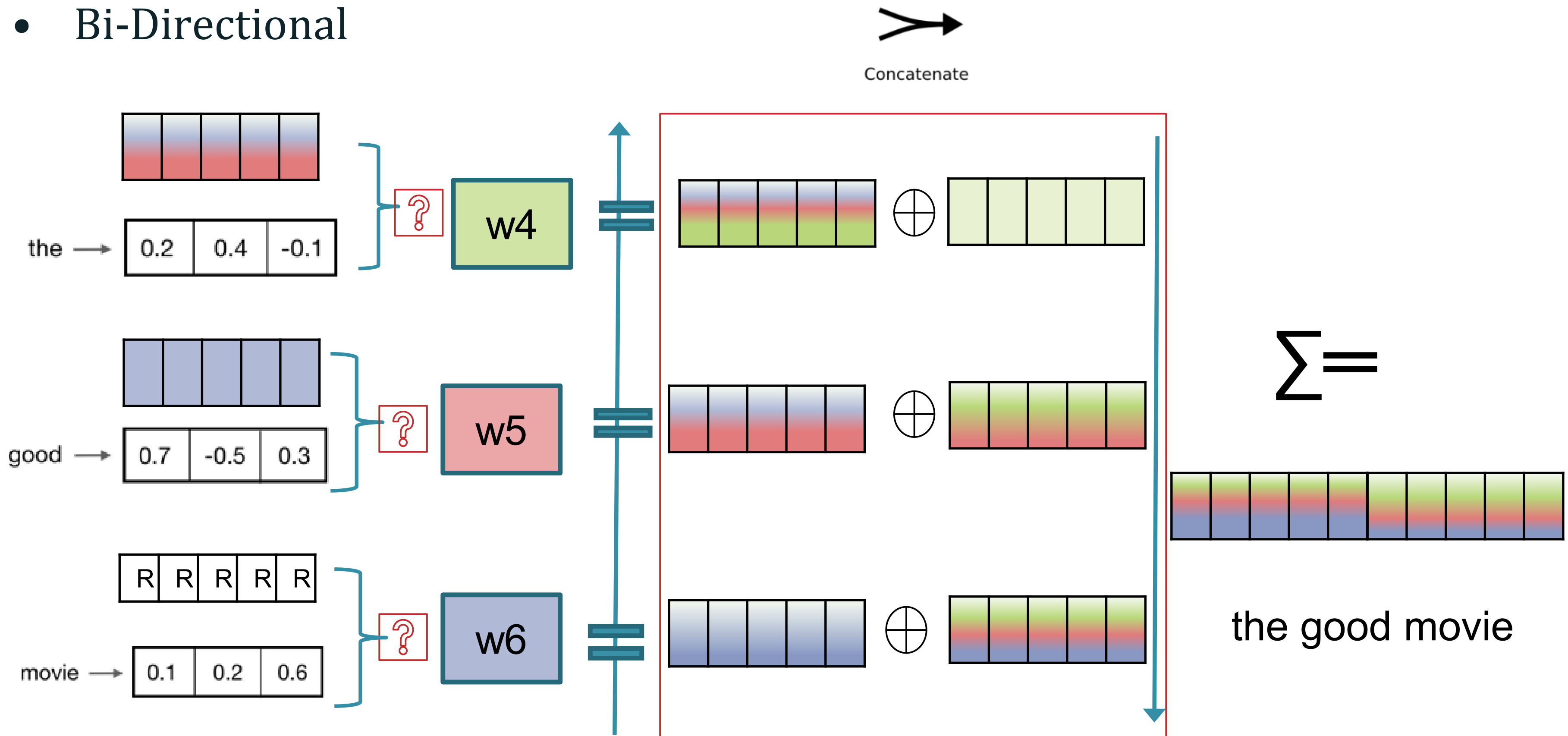
RNN Encoder

- A better (than Ngram) way to represent word/sentence
- Encode the sequence of word tokens (which CNN lacks of)



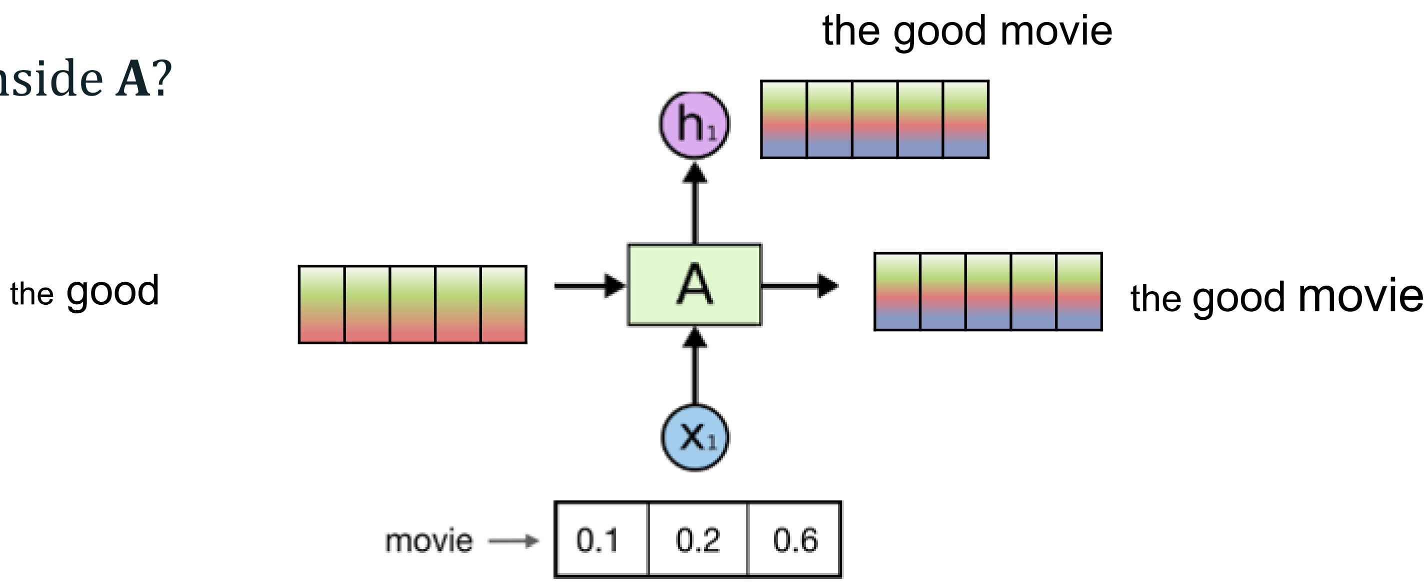
Bi-RNN Encoder

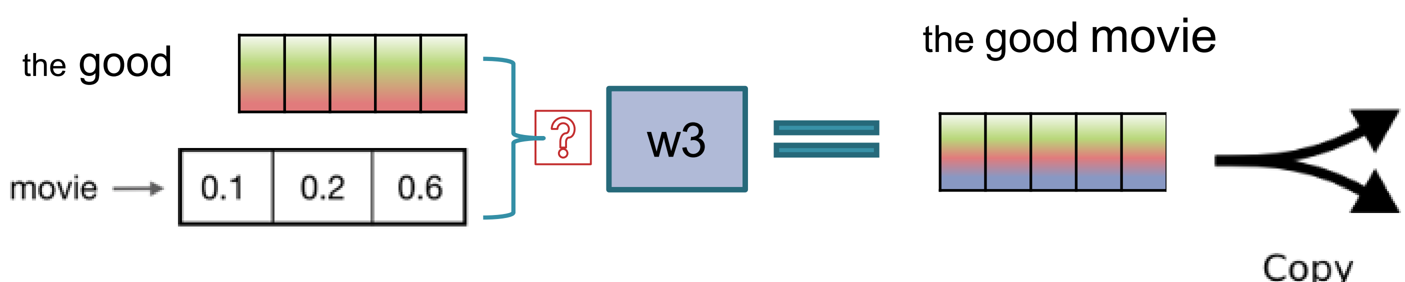
- A better (than Ngram) way to represent word/sentence
- Encode the sequence of word tokens (which CNN lacks of)
- Bi-Directional



Vanilla RNN

- What's inside **A**?

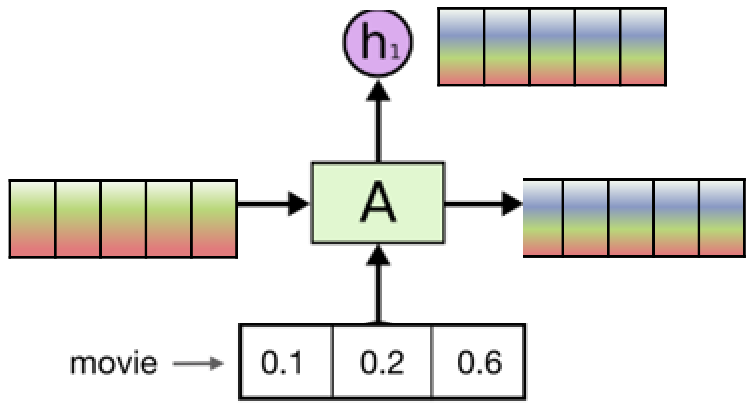


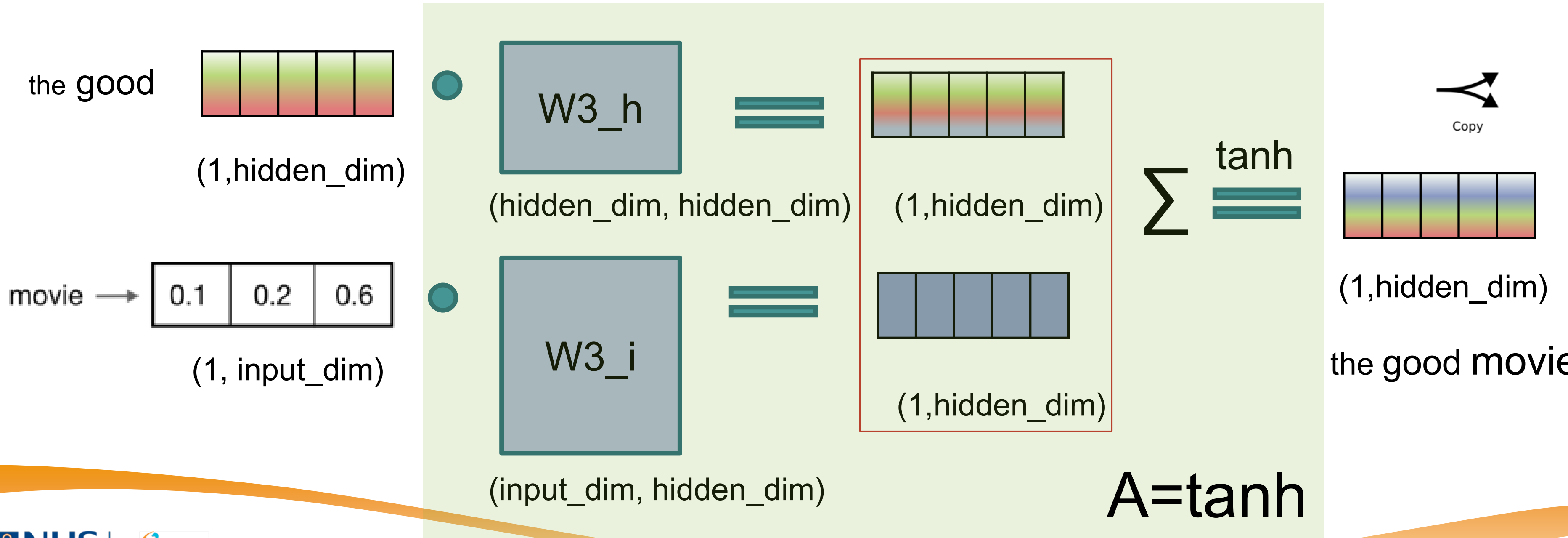


Vanilla RNN

- What's inside **A**?

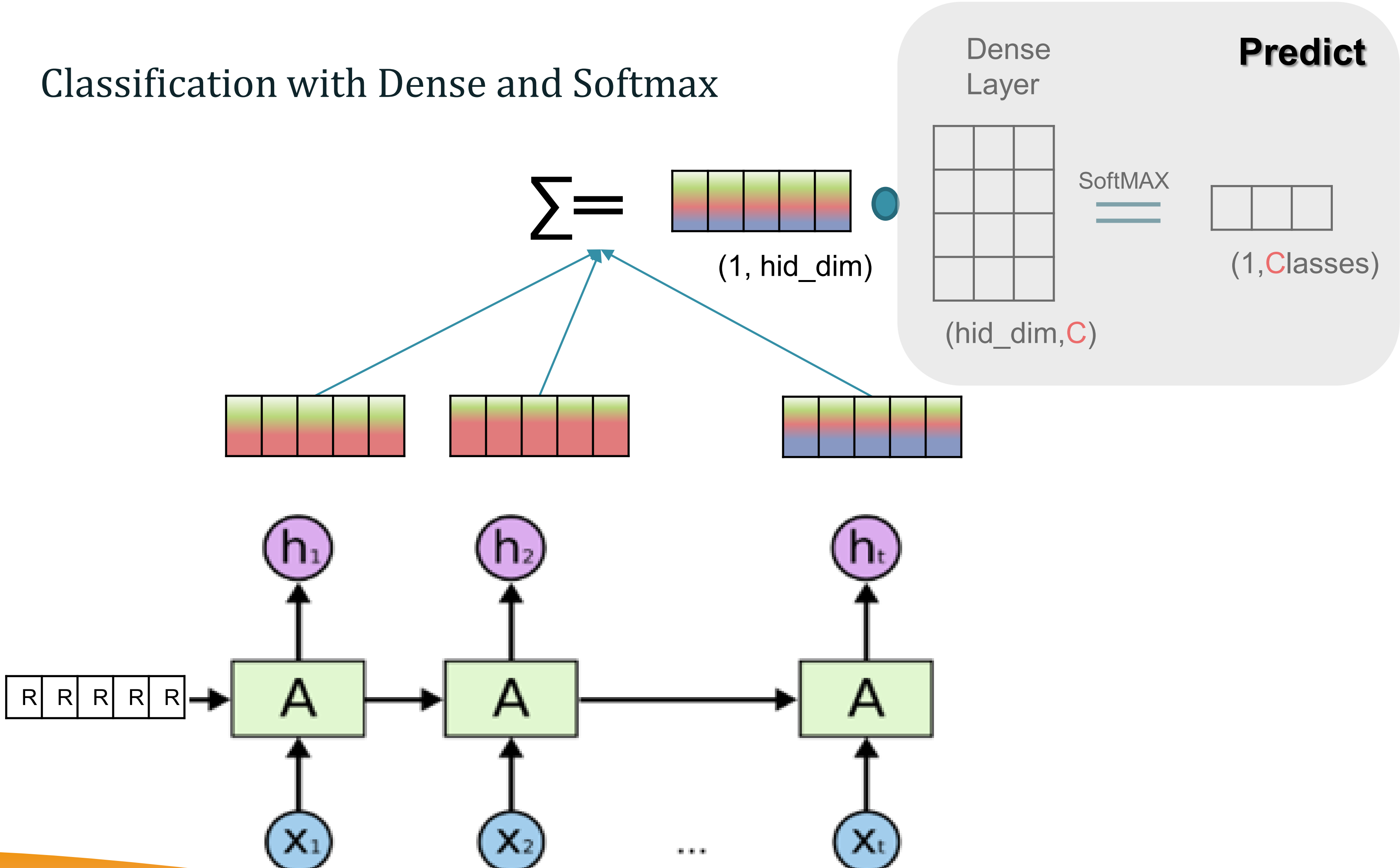
$$h' = \tanh(w_{ih}x + b_{ih} + w_{hh}h + b_{hh})$$





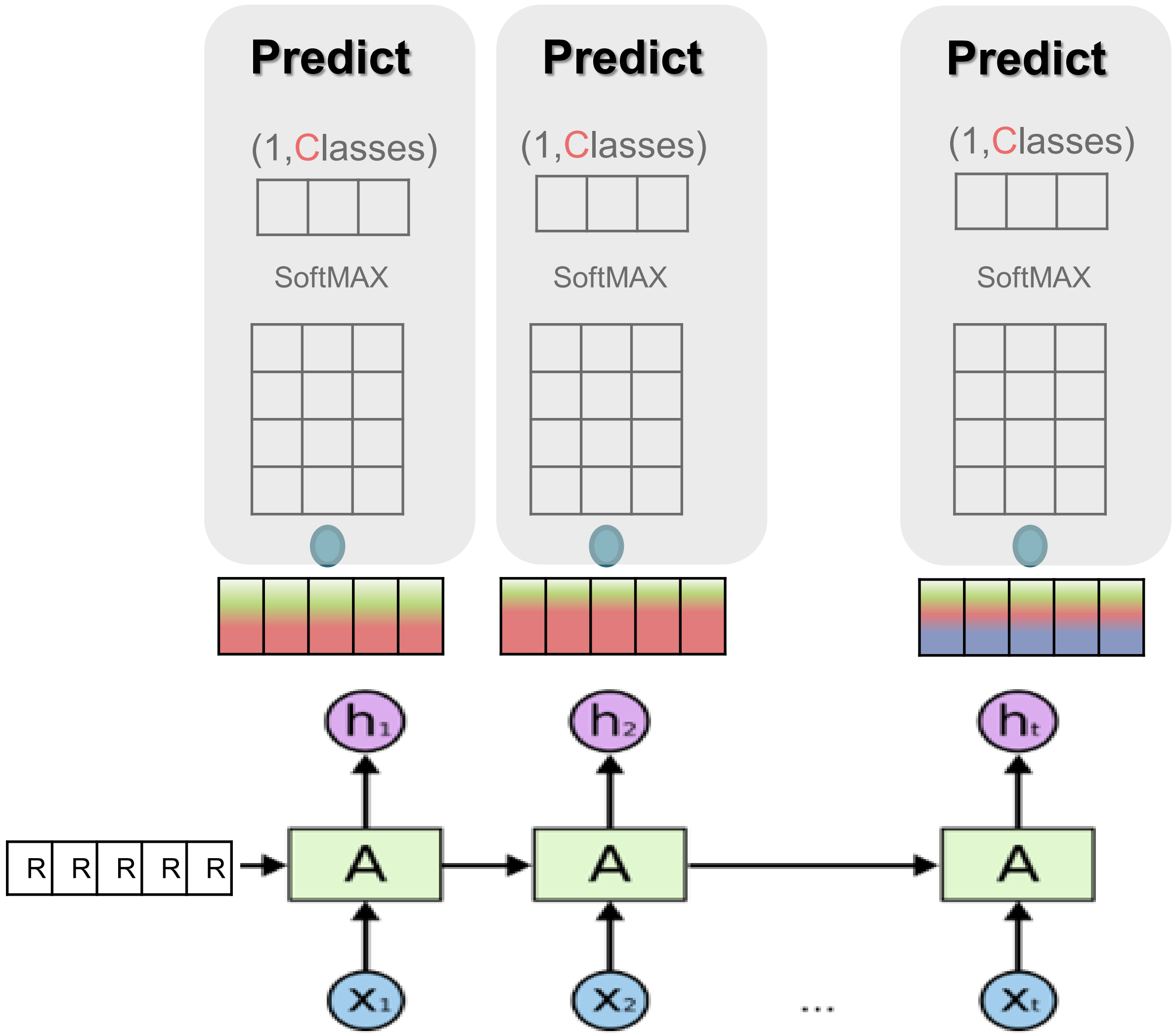
Recurrent Neural Networks

- Classification with Dense and Softmax



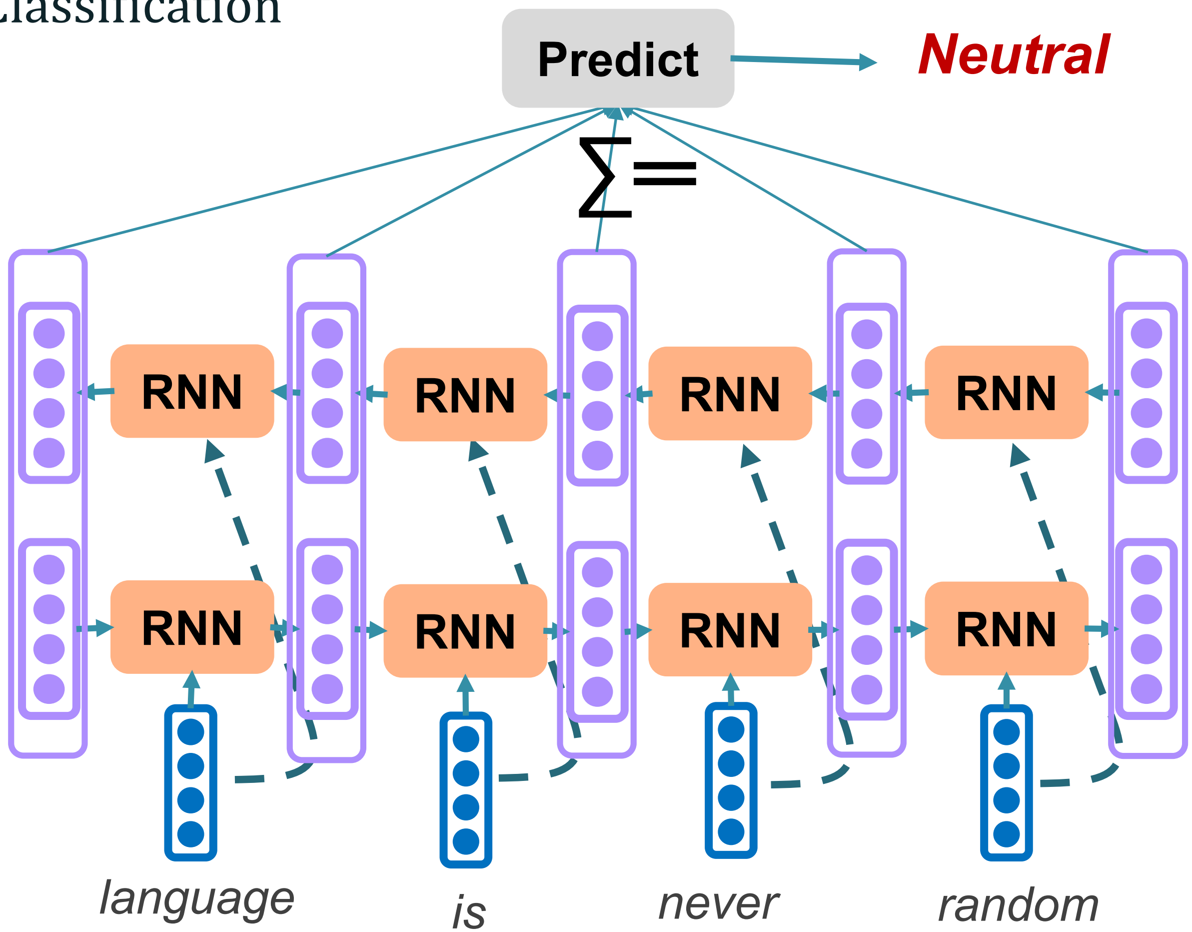
Recurrent Neural Networks

- Sequence Labelling



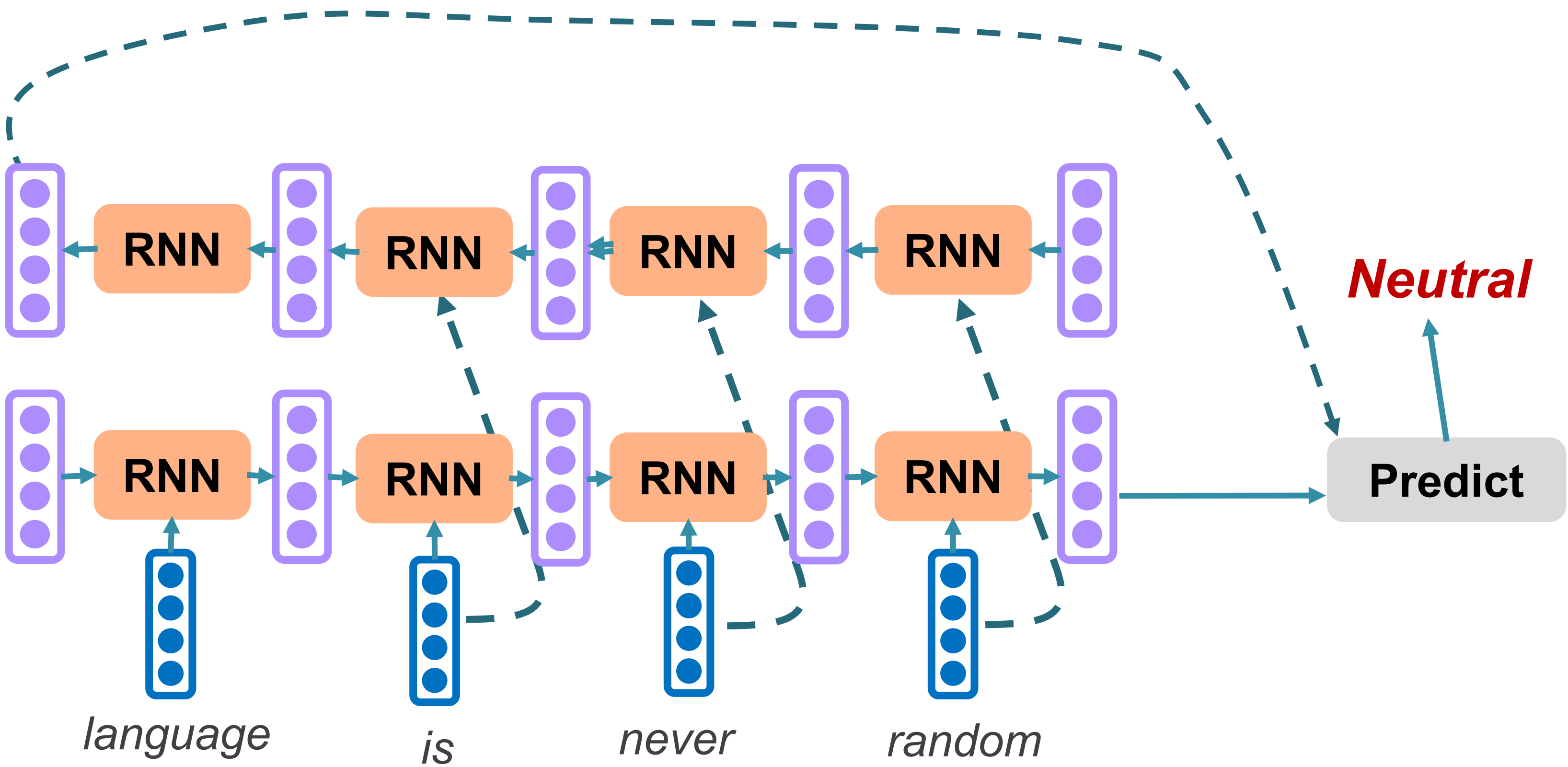
Bi-Directional RNN

- Text Classification



Bi-Directional RNN

- Text Classification



Agenda

- Data Splits and Evaluation
- Evaluation and Optimization
- CNN for Text Classification
- Workshop
- **RNN and LSTM**
  - RNN text Encoder
  - **LSTM for Text Processing**
- SeqtoSeq model
  - Encoder Decoder Model
- Workshop

Long Short Term Memory

- **RNN Vanishing Gradients**

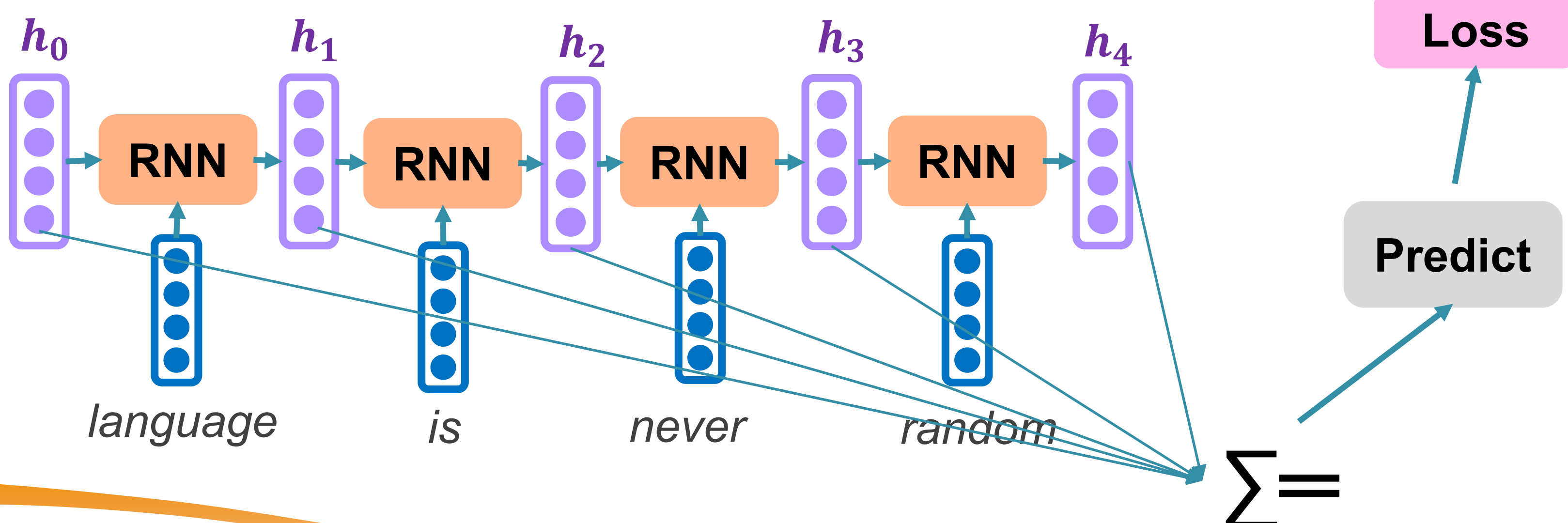
$$\frac{d(loss)}{dh_0} = \text{“poof”}$$

$$\frac{d(loss)}{dh_1} = \textit{tiny}$$

$$\frac{d(loss)}{dh_2} = \textit{small}$$

$$\frac{d(loss)}{dh_3} = \textit{Okay}$$

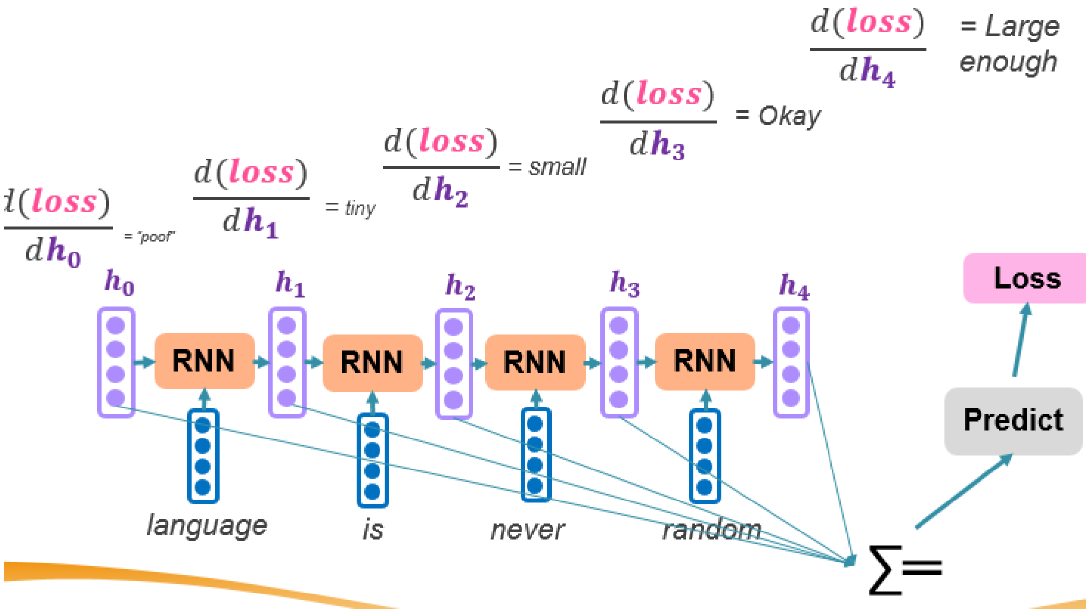
$$\frac{d(loss)}{dh_4} = \textit{Large enough}$$



Long Short Term Memory

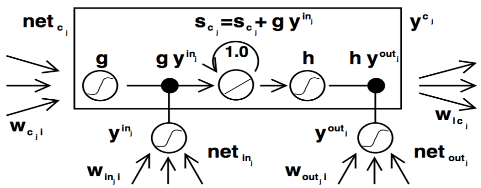
- **RNN Vanishing Gradients**

historical words are less influential to represent the future words

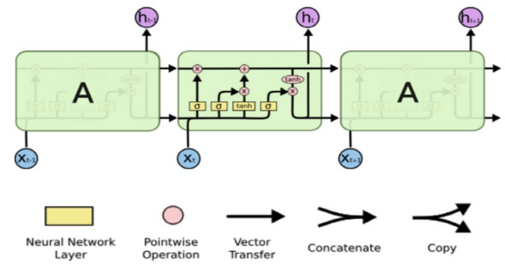


- *Keeping a **memory** of past time steps*
- ***Add** the memory to the current step*
- *Gates control the **strengths** of the memory*

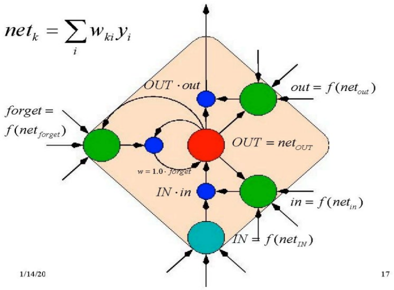
LSTM Gallery



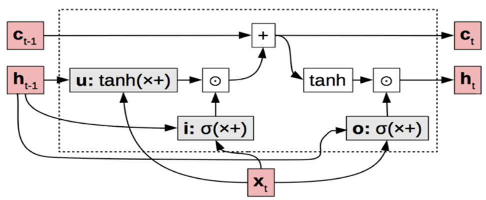
From Hochreiter and Schmidhuber (1997)



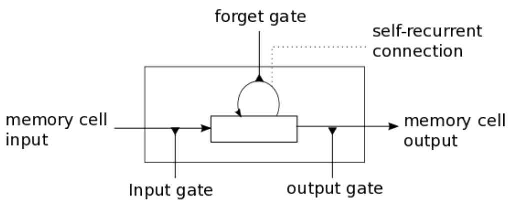
From Olah (2015) blogpost



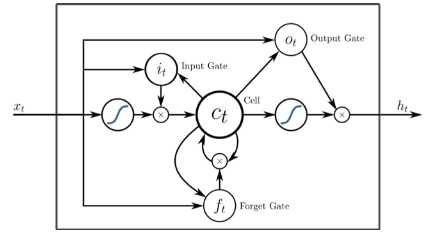
From Schmidhuber (2017) page



From Neubig (2019) CMU NN4NLP Course



From http://deeplearning.net



From Wikipedia

"*Absorb* ***what is useful***, ***discard*** ***what is not***, ***add what is uniquely your own***."
– Bruce Lee

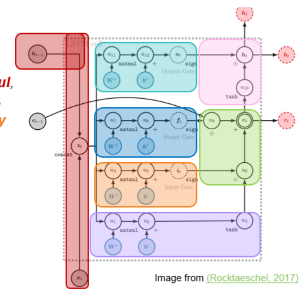
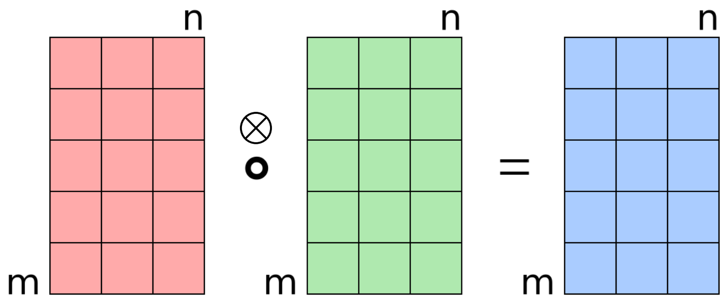


Image from (Rocktaeschel, 2017)

What ‘s inside LSTM

- Some preparations:
  - Element Wise Product $\circ$ or $\otimes$
  - **Different** from dot Product
  - Similar to ***Conv*** without summing up
  - **Shape** of Matrix **remains**

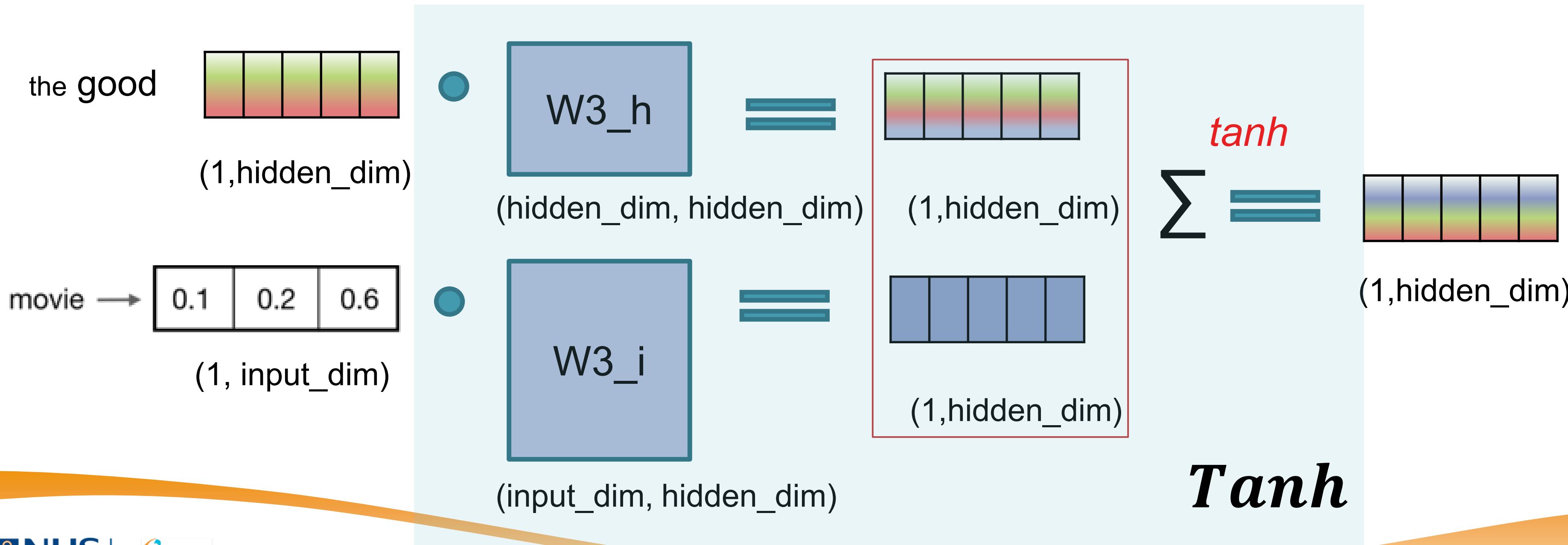


$$A \circ B = C$$

$$\overset{G}{\begin{bmatrix}3 & 5 & 7\\ 4 & 9 & 8\end{bmatrix}} \circ \overset{H}{\begin{bmatrix}1 & 6 & 3\\ 0 & 2 & 9\end{bmatrix}} = \overset{N}{\begin{bmatrix}3\times1 & 5\times6 & 7\times3\\ 4\times0 & 9\times2 & 8\times9\end{bmatrix}}$$

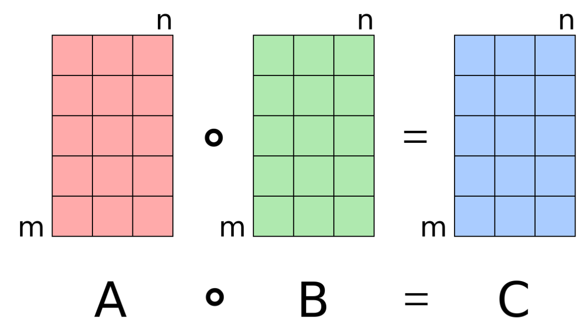
What ‘s inside LSTM

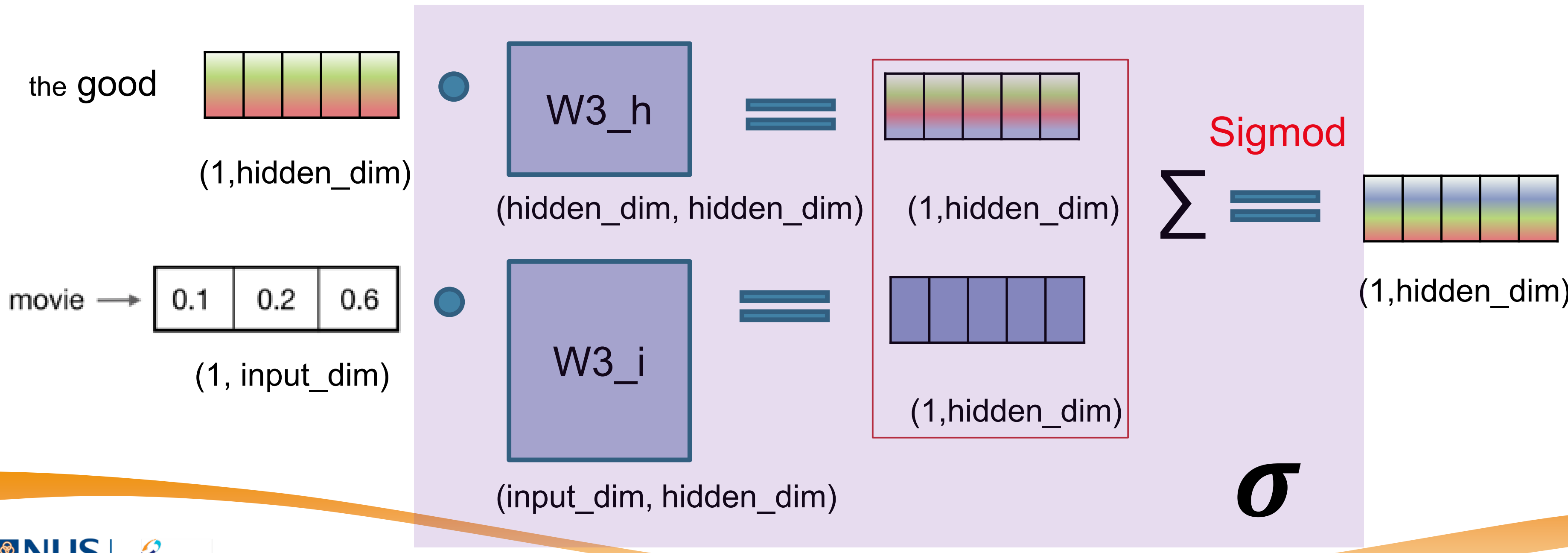
- Some preparations:
  - Element Wise Product
  - $\boldsymbol{\sigma}$ annotation
  - $\boldsymbol{Tanh}$ annotation



What ‘s inside LSTM

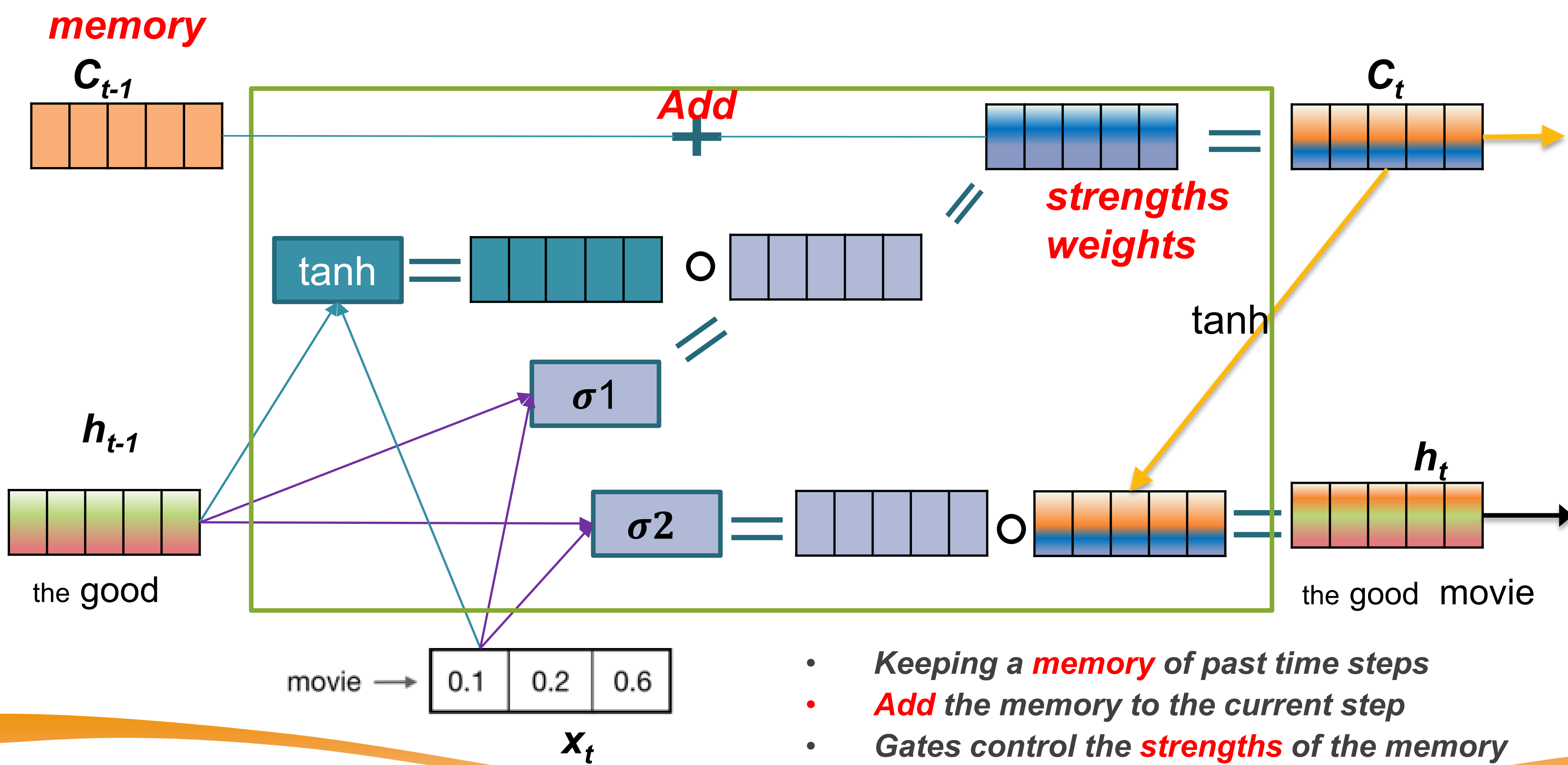
- Some preparations:
  - Element Wise Product
  - $\boldsymbol{\sigma}$ annotation





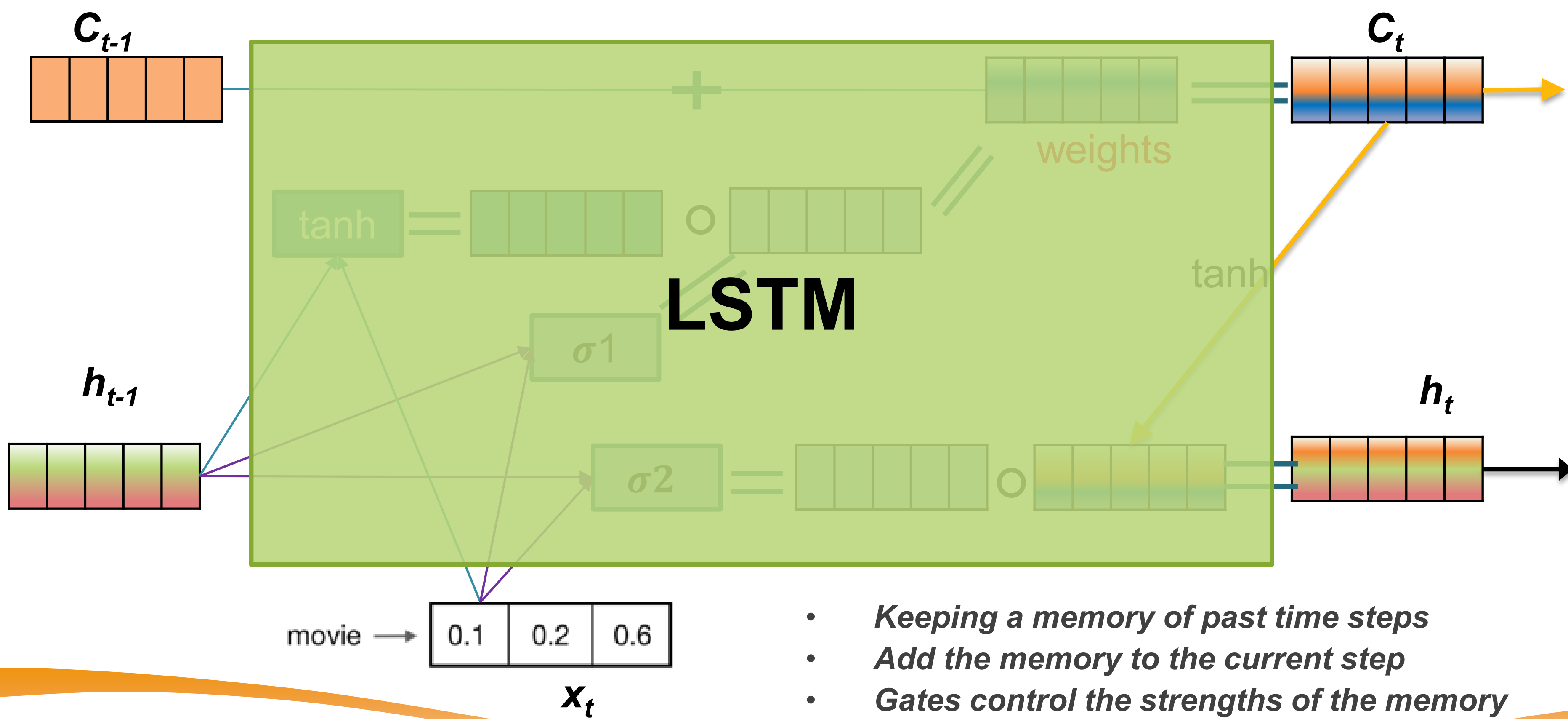
LSTM

- What's inside **A for LSTM**?
  - Introduce one **Empty/Random** Matrix ***C*** to hold **memory**
  - Calculate the **strengths/weights** of the current memory of ***C***



- ***Keeping a memory of past time steps***
- ***Add the memory to the current step***
- ***Gates control the strengths of the memory***

- What's inside **A for LSTM**?
  - Introduce one **Empty/Random** Matrix ***C*** to hold **memory**
  - Calculate the **strengths/weights** of the current memory of ***C***



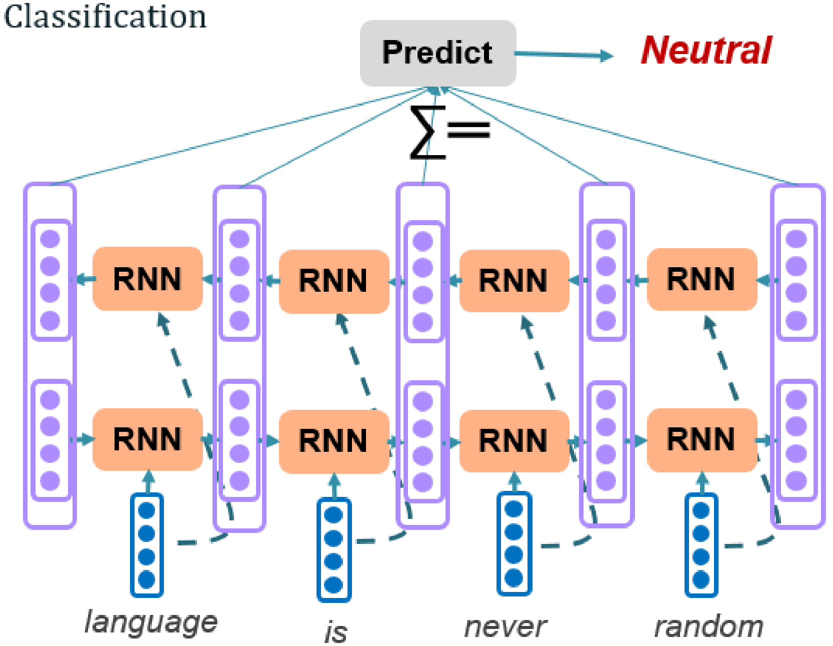
- ***Keeping a memory of past time steps***
- ***Add the memory to the current step***
- ***Gates control the strengths of the memory***

LSTM

- Is vanishing gradient just an RNN problem?
  - As long as there are many layers nested , the functions and gradients gets multiplied in a nested manner
  - It is a **DNN problem**

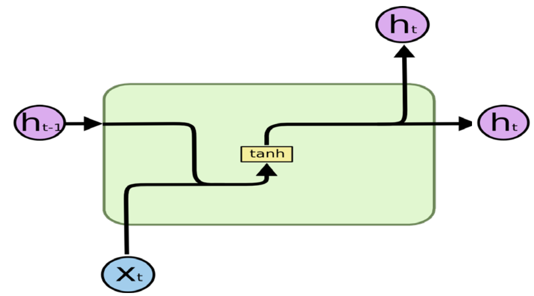
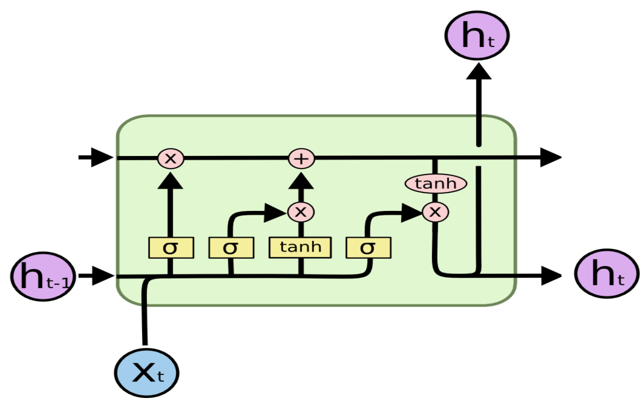
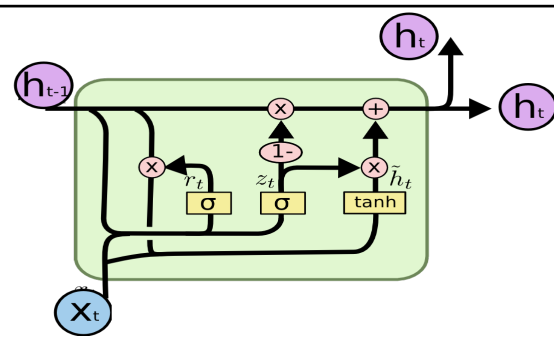
Bi-Directional RNN

- Text Classification



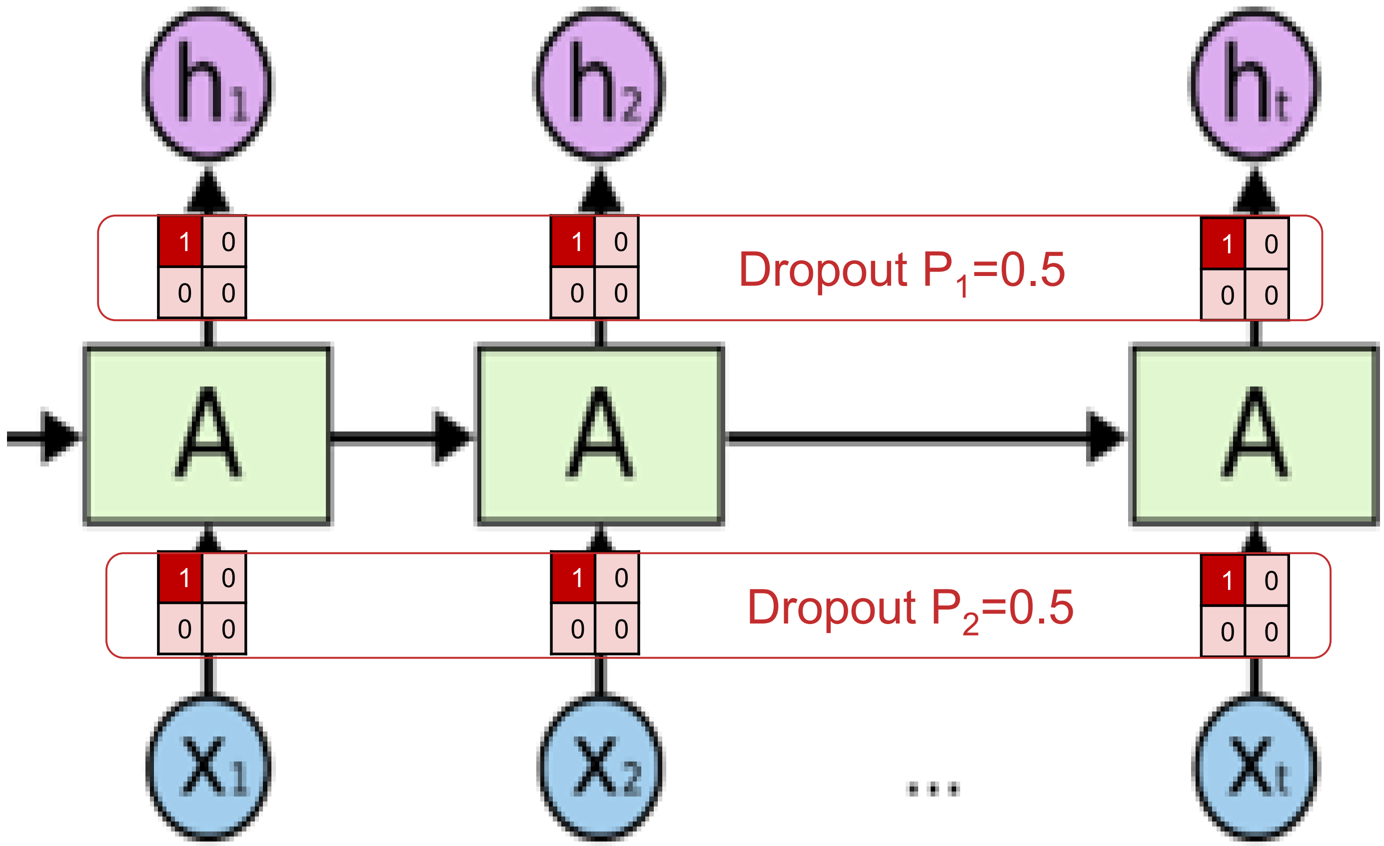
LSTM

- As your RNN nodes
  - Vanilla
  - LSTM
  - GRU
- **LSTM/GRU doesn't guarantee no gradient vanishing**
- It's just better than the vanilla RNN

	Vanilla Not in used anymore
	LSTM Most popular Bi-directional
	GRU Simplified LSTM Faster but weaker

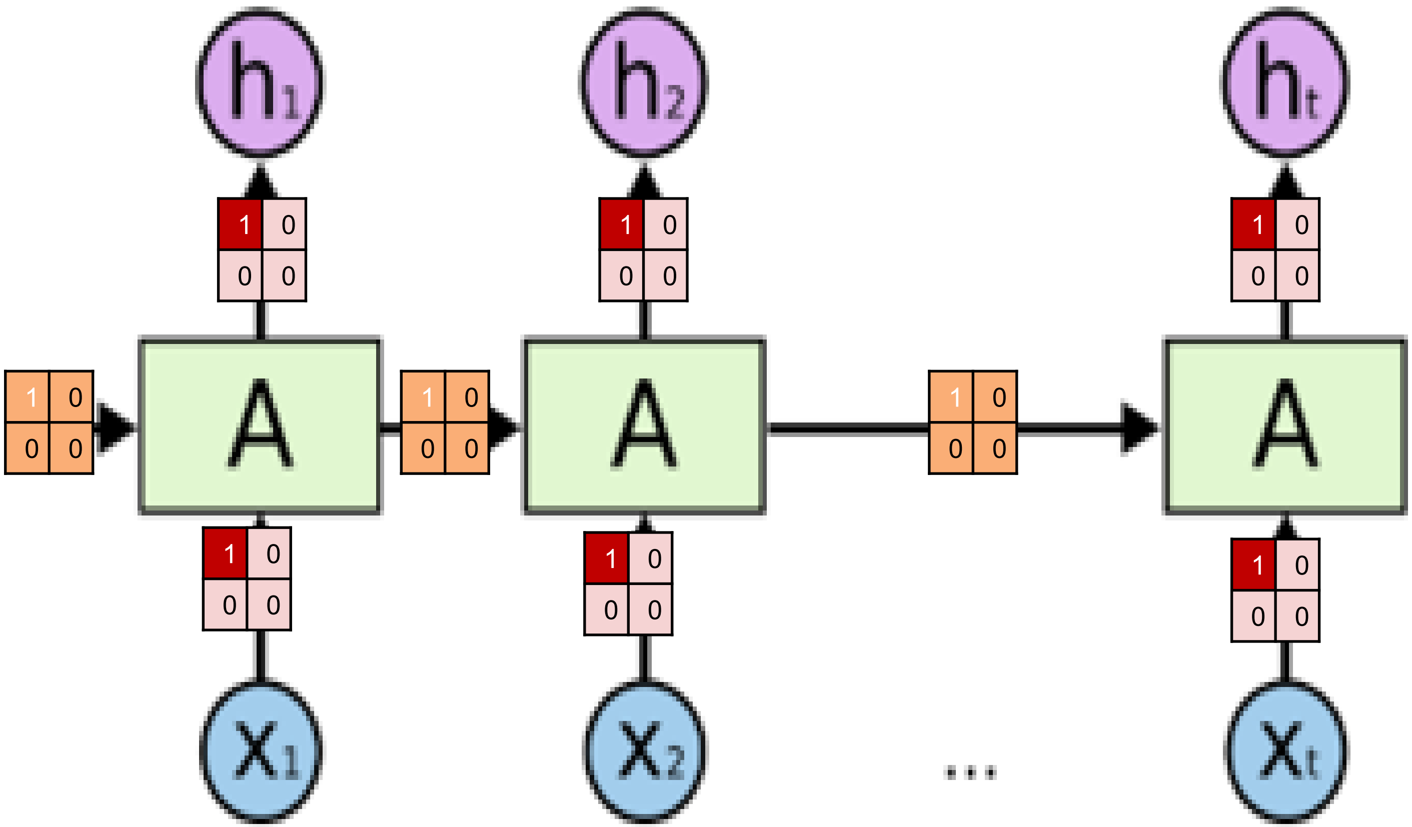
LSTM with Dropout

- Naïve Dropout RNN



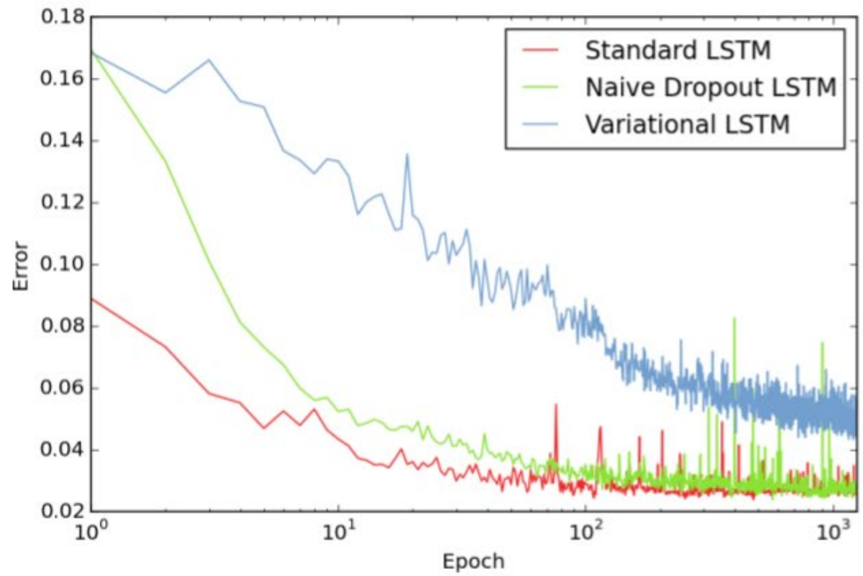
LSTM with Dropout

- Variational RNN

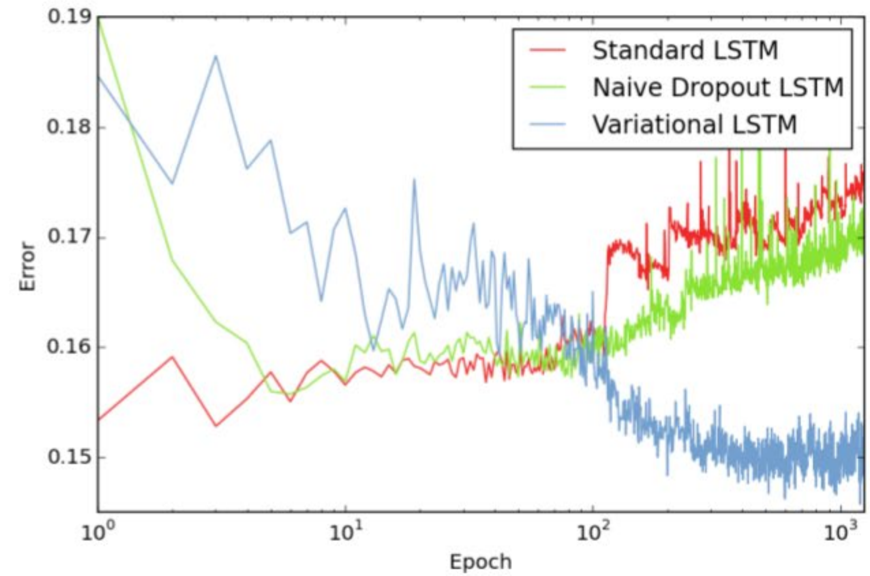


LSTM with Dropout

- Variational RNN



(a) LSTM train error: variational, naive dropout, and standard LSTM.



(b) LSTM test error: variational, naive dropout, and standard LSTM.

https://arxiv.org/pdf/1512.05287.pdf

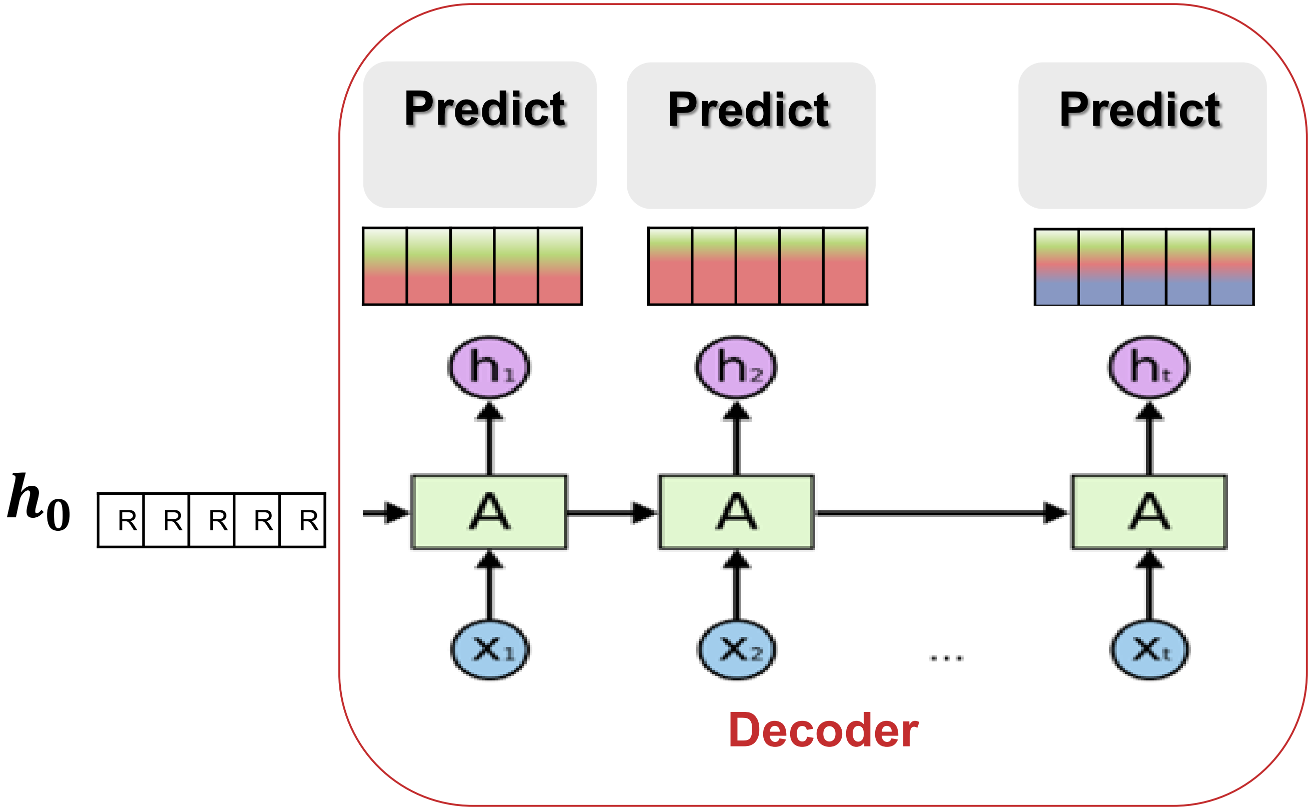
Agenda

- Data Splits and Evaluation
- Evaluation and Optimization
- CNN for Text Classification
- Workshop
- RNN and LSTM
- **SeqtoSeq model**
  - **Encoder Decoder Model**
- Workshop

Sequence to Sequence

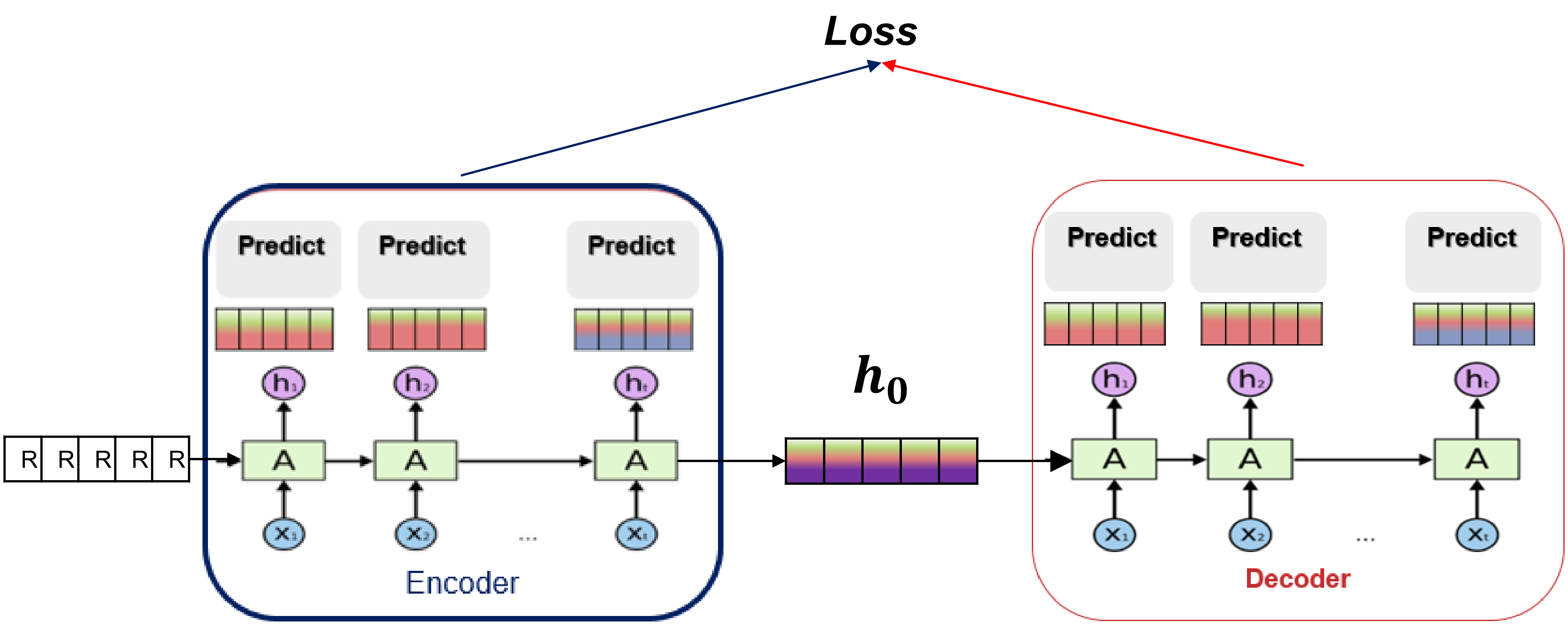
Sequence to Sequence

- RNN starts with a **random state (vector)** $\boldsymbol{h_0}$
- What if $\boldsymbol{h_0}$ is something non-random?



Sequence to Sequence

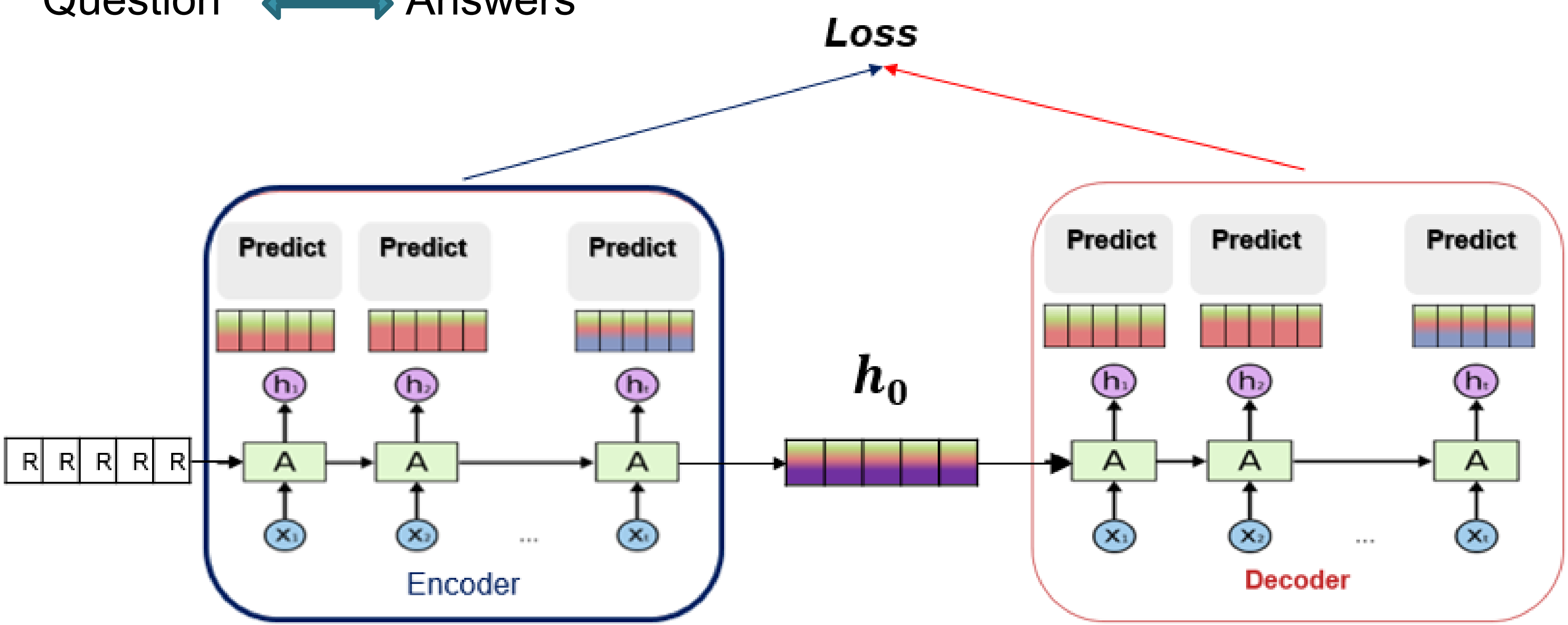
- RNN starts with a **random state (vector)** $\boldsymbol{h_0}$
- What if $\boldsymbol{h_0}$ is something non-random?
- What if $\boldsymbol{h_0}$ is generated by another RNN?



Sequence to Sequence

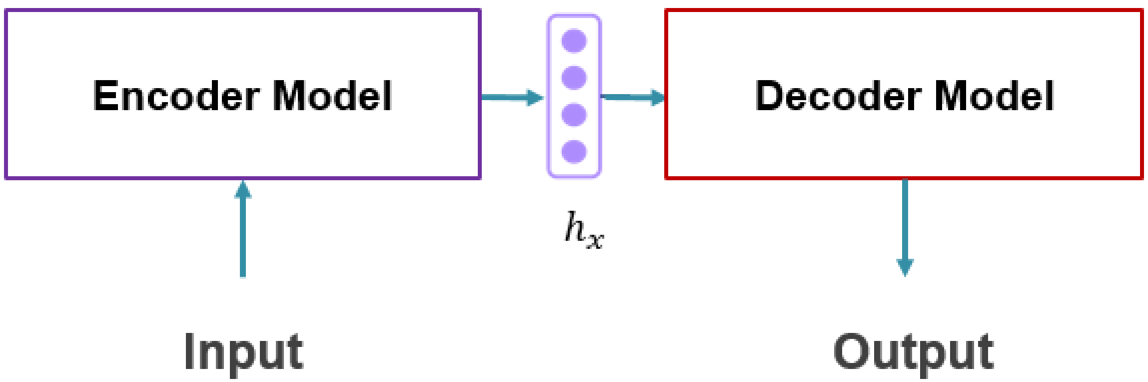
- “Translate” (Ideally) any object A to any object B

French ⟷ English
Image ⟷ Text
Audio ⟷ Text
Document ⟷ Summary
Question ⟷ Answers



Sequence to Sequence

- Generalized Encoder-Decoder Framework



- **Endless possibilities** of what to condition on and what to generate
- But training requires the **paired condition and target generation**
- Relatively **huge amount of data is needed** for model to train well

What's Next

WELCOME TO THE LEGO LAND

The “ImageNet” Moment for NLP

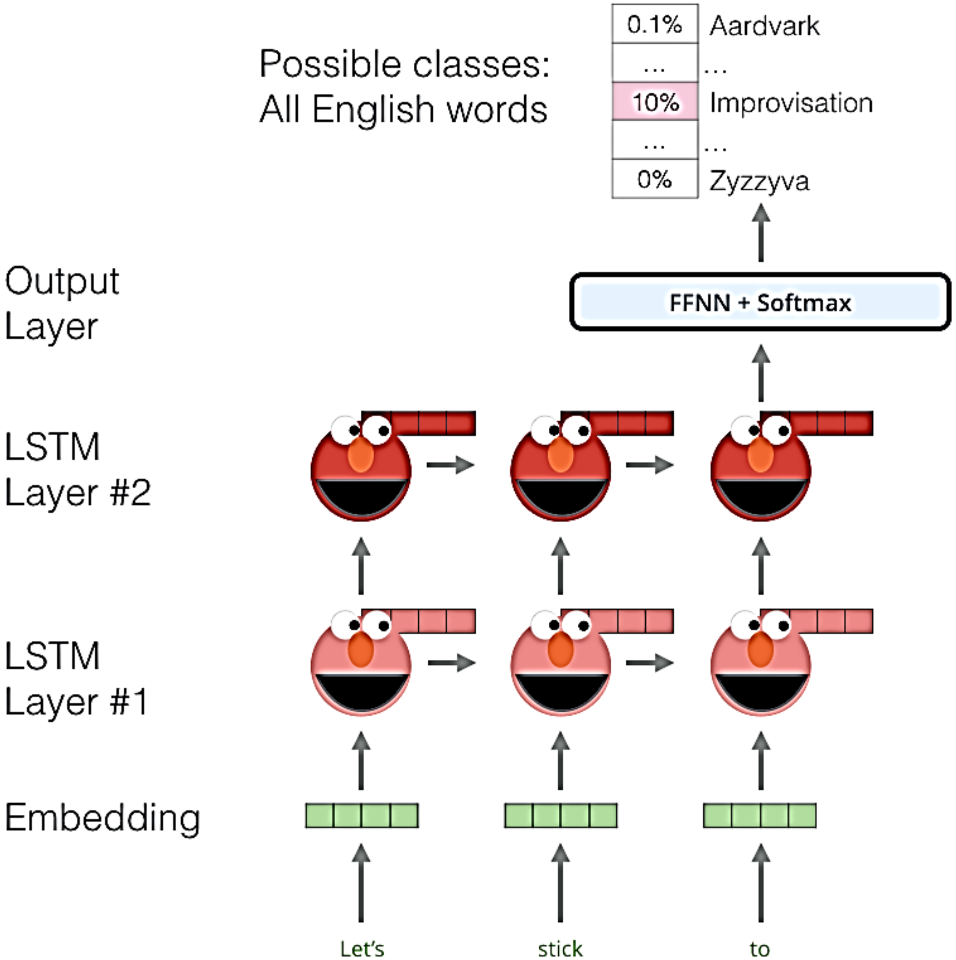
- **NLP counterparts of “ImageNet”**
  - an ImageNet-like dataset should be *sufficiently large* **with labels**
  - should be *representative of the problem space* of the discipline

"The service was poor, but the food was ________".

- **Language Modeling is capturing**
  - long-term dependencies
  - hierarchical relations
  - sentiment
- **Plain Text are everywhere**

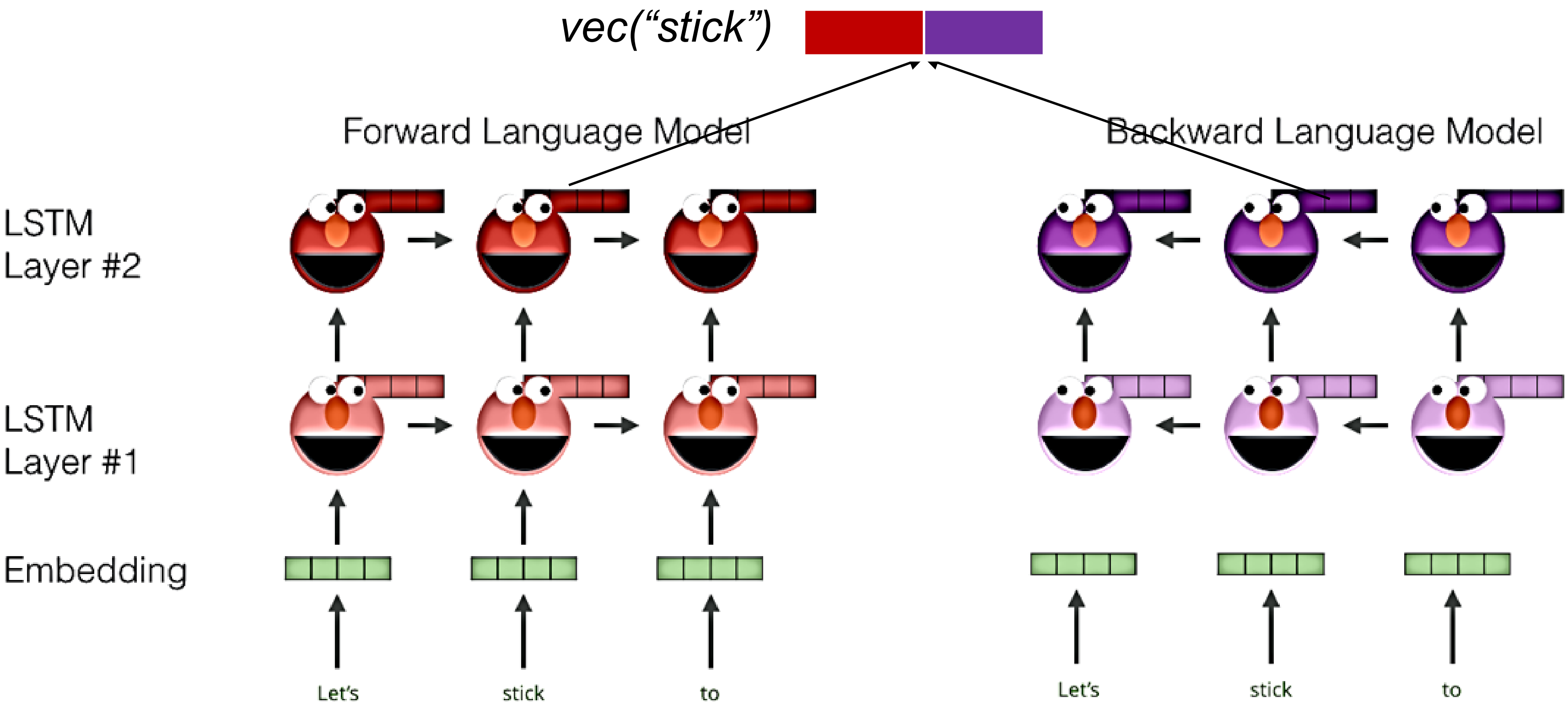
ELMo (2017)

- LSTM for Language Modelling
  - Starting from **GLOVE**
  - Given "*Let's stick to*"
  - Predict "*improvisation*"
  - Two layers of LSTMs **stacked**
  - Single direction?

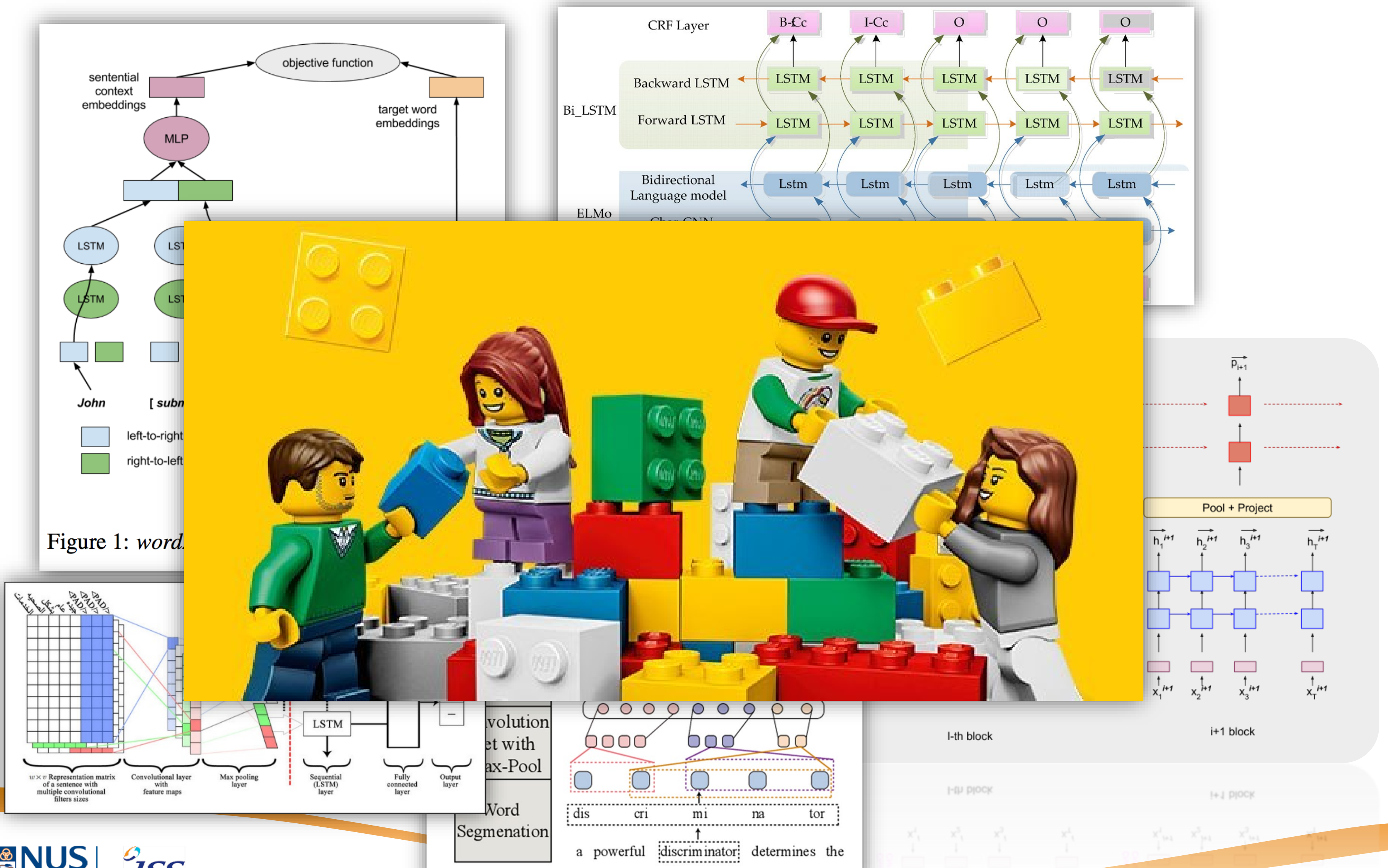


ELMo

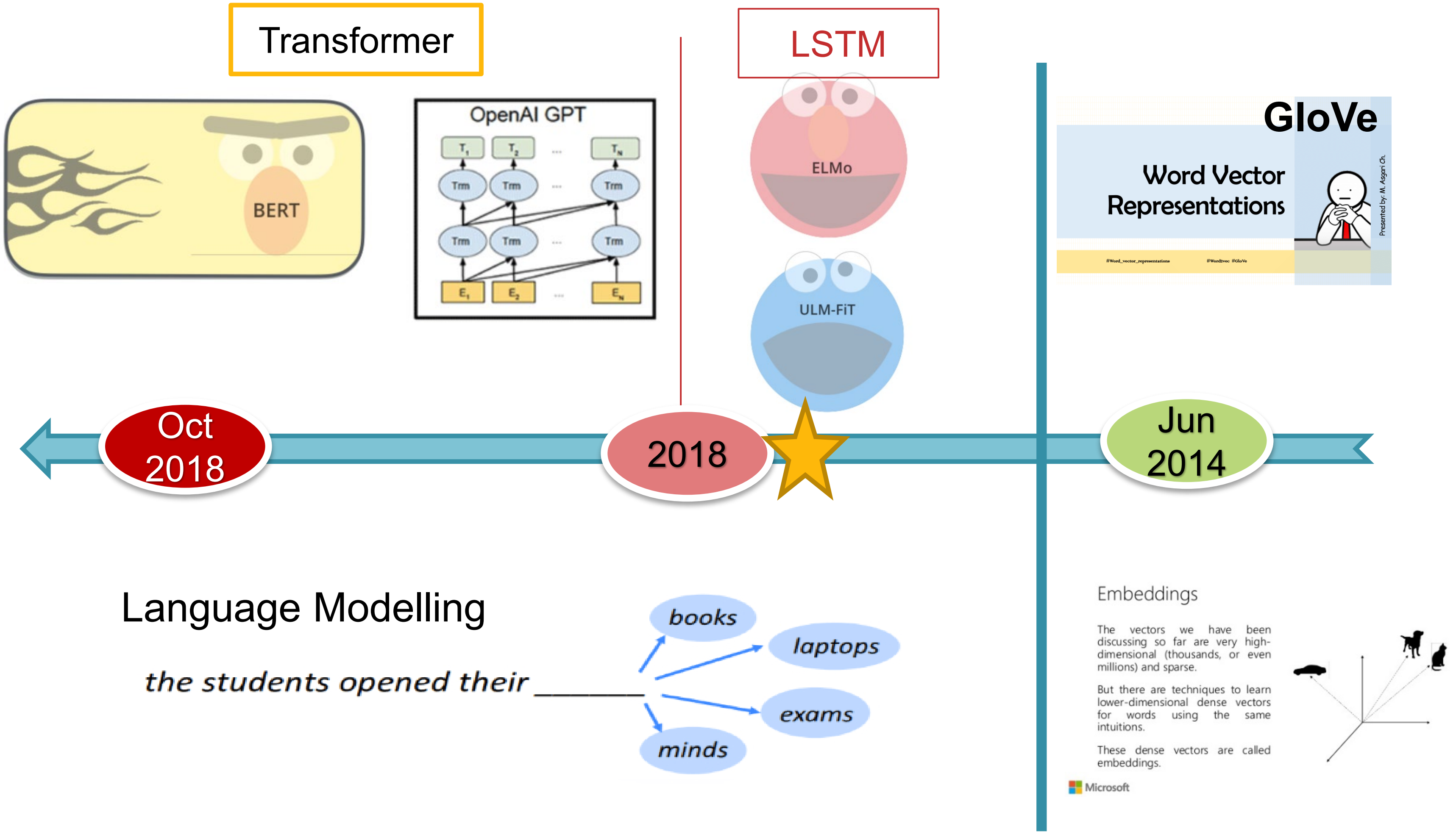
- Always Bi-LSTM



LEGO Arts - Have Fun with it....

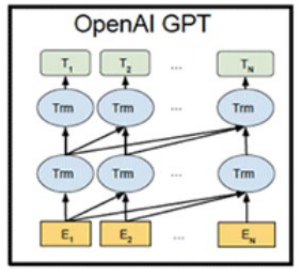


Before and After the Moment



Turing Award to "NVIDIA"

Transformer



LSTM

Oct 2018

2018

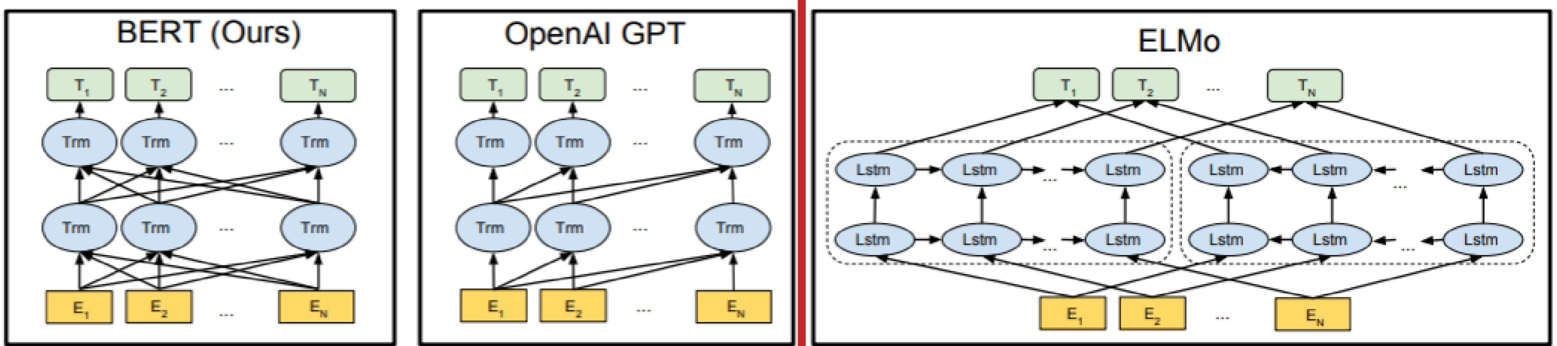
2012

1997

LSTM

Attention

The "ImageNet" Moment for NLP



Oct 2018

??? 2018

Apr 2017

24 Transformer blocks
1024 hidden layers, and
340M parameters
16 TPU pods for
training

12 blocks
37-layer
1 month on
8 GPUs

4096 units
512 dimensions
Single GPU

(*Image from Jay Alammar's blog)

Workshop

LSTM AND SEQ2SEQ