SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Neural Networks – Sequence Models

PROF LIM KWAN HUI

50.021 Artificial Intelligence

The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.

Outline & Objectives

- Be able to use neural networks for generating word representations, e.g., Word2Vec
- Have a general understanding of how sequence models work, including RNNs and LSTMs

This week

- Understand how convolution neural networks work
- Be able to apply the various convolution-related operations in a simple example

Next week

Sequence modelling

- Given an image of a ball, where will it roll to?
- Music -> what is the next note?
- Audio -> what is the expected next sound?
- Text -> what is the next expected word?
- Text -> translation of an entire sentence?

"This morning I took my cat for a walk."

given these words

How to represent text?

- Bag of words (BOW)

	I	Went	to	the	movies	yesterday	it	listening	to	music	is	great	swimming	and	ran
D1	1	1	1	1	1	1									

- TD-IDF: Term Frequency - Inverse Document Frequency (Rare words carry more meaning)

	Mary	Loves	Movies	Cinema	Art	John	Went	to	the	Delicatessen	Robert	Football	Game	and	for
d1	0.3779	0.3779	0.3779	0.3779	0.3779									0.0001	

How to represent text?

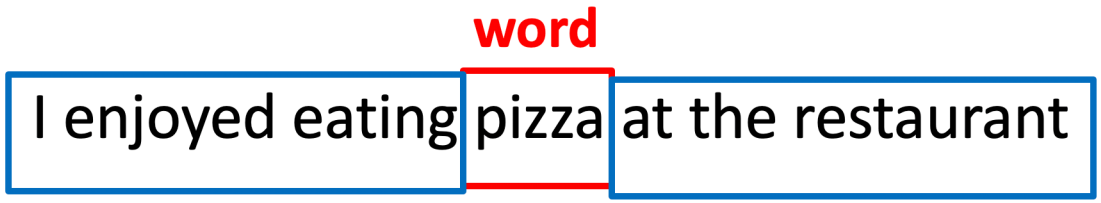
- Term frequency (TF):
  - Number of times the term appears in the doc/total number of words in the document
- Inverse Document Frequency (IDF):
  - log(number of docs/number docs the term appears in)
- TF-IDF = TF *IDF
- BOW AND TF-IDF are sparse representations
  - Less sparse models: co-occurence models (Singular value decomposition) and vector embeddings models (GLOVE, word2vec)

Word vectors or embeddings

- Based on **Distributional Semantics Hypothesis**
- *"You shall know a word by the company it keeps"*

  – John Rupert Firth 1957
- Words that occur in a similar context tend to have similar meaning (Harris, 1954)

Word embeddings



The company it keeps

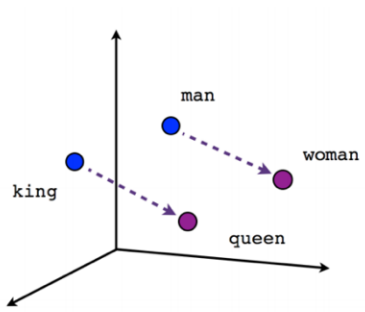
I enjoyed eating pizza at the restaurant

I enjoyed eating pineapple at the restaurant

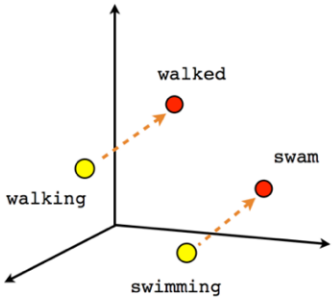
Same context, same meaning?

Word embeddings

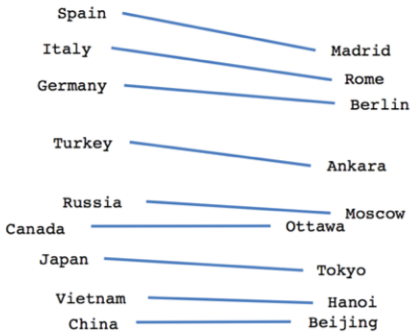
- Multidimentional representation for each word (e.g. coordinates in 128-dimensional space)
- Here represented via t-SNE projection in 2-dimensions



Male-Female



Verb tense



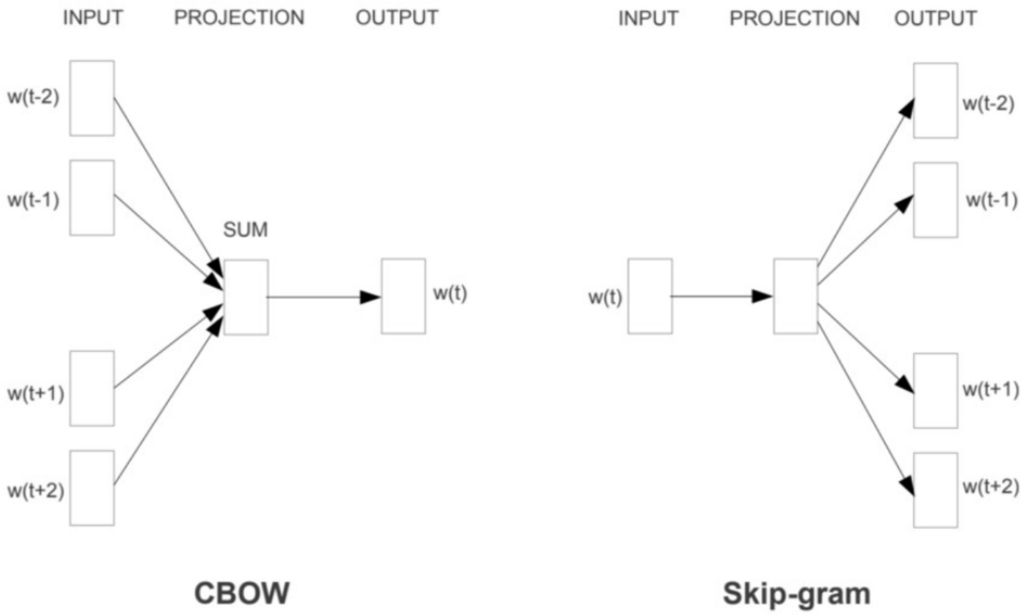
Country-Capital

Advantages

Traditional Method - Bag of Words Model	Word Embeddings
• **one hot** encoding • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . . 0 0 0 0 • Context information is not utilized	• Each word is a **point in *n*-dimensional space**, represented by a **vector of length *n***, (*n* ~ 100 - 300) • Trained by using big text dataset • For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are basically projections along different axes, more of a mathematical concept.

Two popular architectures

- 2 architecture options (both 1 layer): CBOW and Skip-gram (the latter more popular)
  - Continuous Bag of Words (CBOW) Predicting a center word form the surrounding context.
  - Skip-grams(SG) Predicting surrounding context words given a center word.

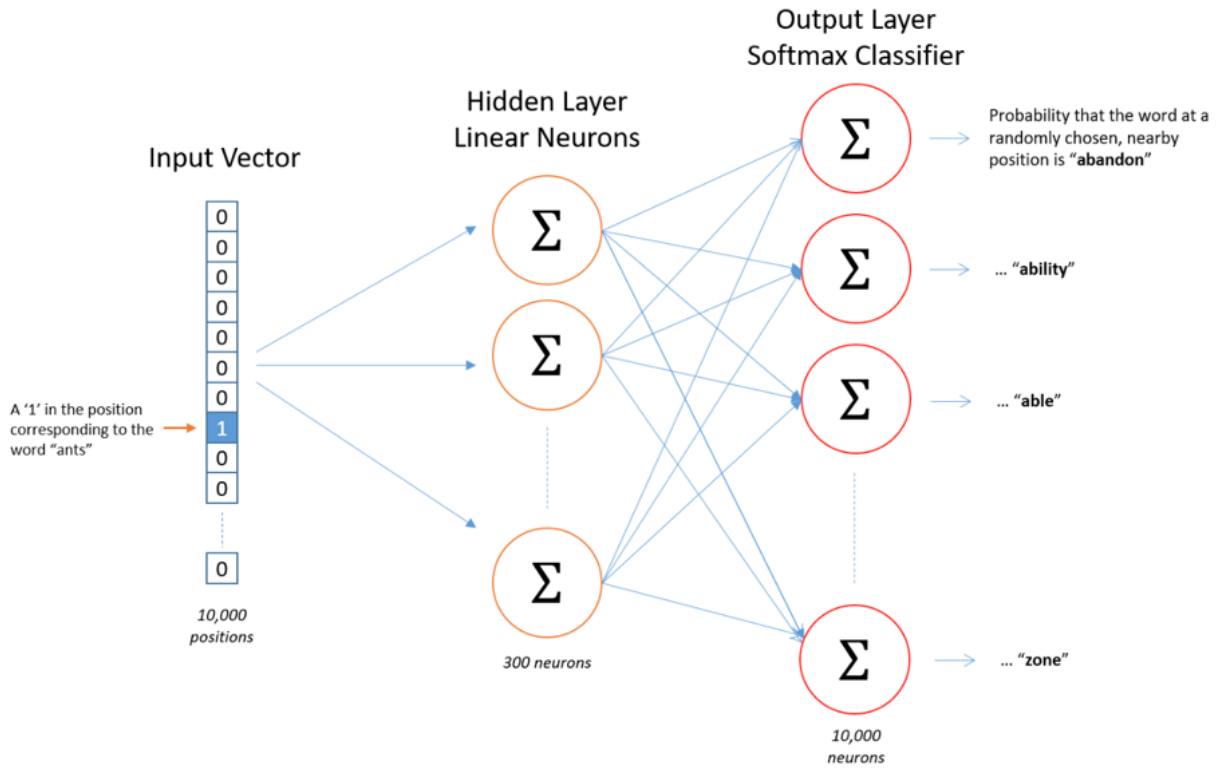


Word2vec - Skip Gram

- Instead of counting co-occurrences
  - Predict surrounding words for every word
- Training objective:
- ***Maximize the likelihood of the context, given the focus word***
  - P (I | pizza)
  - P (enjoyed | pizza)
  - P (restaurant | pizza)

Skip-gram

- Feed-forward neural network – 1 hidden layer



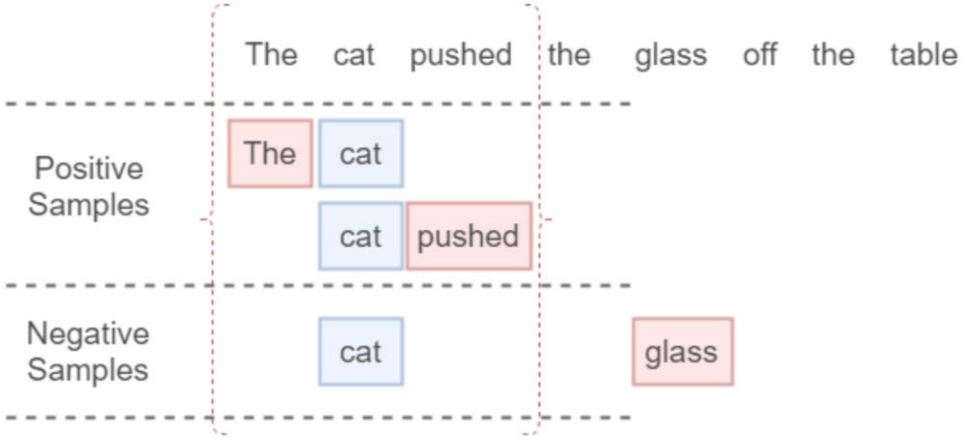
Word2vec

- To train this neural network (i.e., set the weights), we need:
  - Decide window size – dependent on your dataset size / quality
  - Decide the embedding size (i.e., number of nodes in the network)
  - Size of the vocabulary: typically words that occur only once or words like ‘the’ are omitted.

- Training objective? Softmax cross-entropy? => hard to compute
  - Better solution: negative sampling

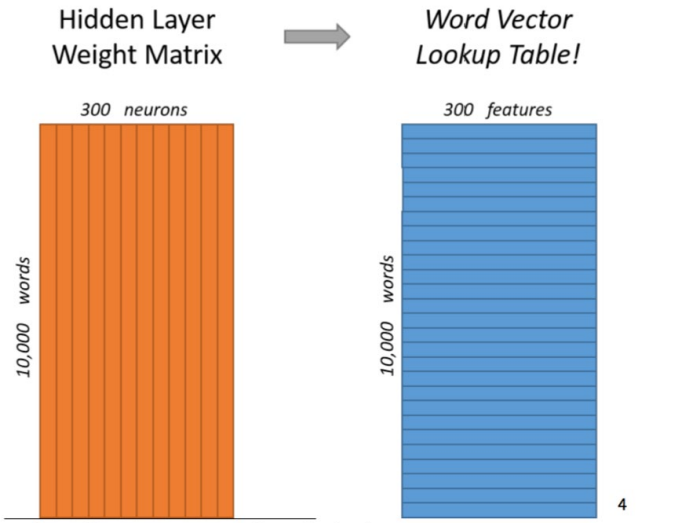
Negative sampling as objective

- The original training objective is thus approximated by a new, more efficient, formulation that implements a binary logistic regression to classify between data and noise samples.



Resulting model

- Fast and accurate predictive model
- Captures semantic content
- Very popular
- Pretrained models available
- Super fast to train
- Easy to use with Python Gensim library



[4] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Don’t reinvent the wheel

- Load pretrained embeddings:

```
import gensim
from torch import nn

model = gensim.models.KeyedVectors.load_word2vec_format('path/to/file')
# weights to use in from_pretrained()
weights = torch.FloatTensor(model.vectors)
# getting index from word
model.vocab['banana'].index
```

Some issues:

- How long do we look back?
  - Distant past

"**France** is where I grew up, but I now live in Boston. I speak fluent ___."

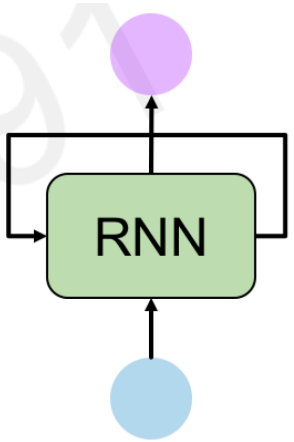
- Counts (BOW) don't preserve order:
  - That food was good, not bad at all
  - That food was bad, not good at all
- We need to be able to assign weights to each of the words during training

[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

this morning took the cat

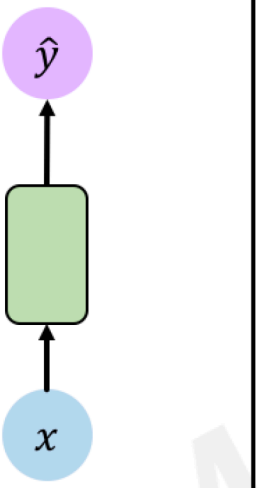
Sequence modeling

- Variable length sequences
- Long-term dependencies
- Maintain info about order
- Share parameters across the sequence

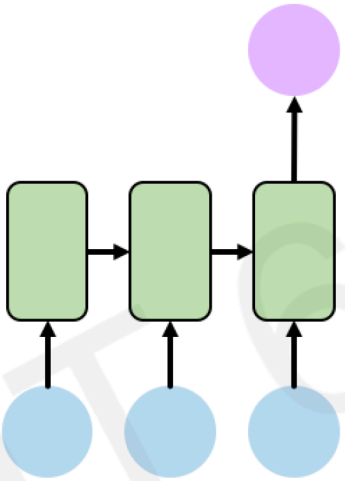


- Recurrent neural networks

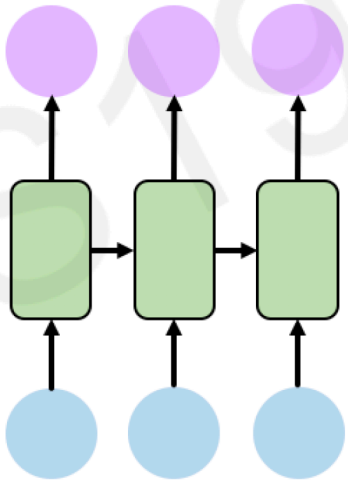
RNNs



One to One
"Vanilla" neural network



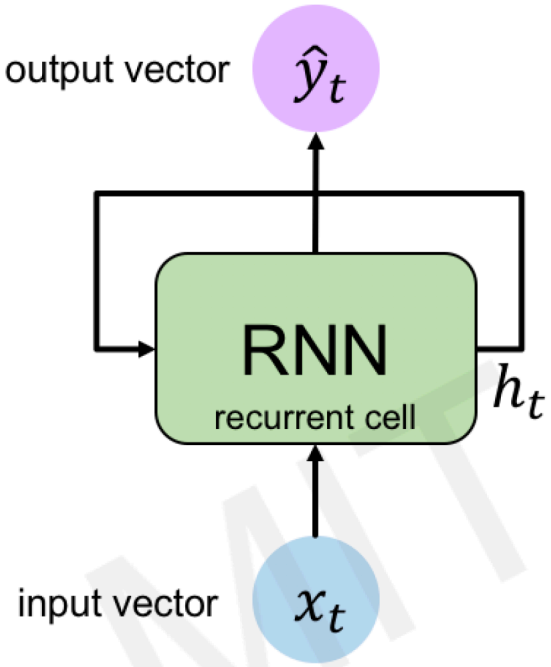
Many to One
Sentiment Classification



Many to Many
Music Generation

… and many other architectures and applications

RNN



Apply a **recurrence relation** at every time step to process a sequence:

$$\boxed{h_t} = \boxed{f_W}\left(\boxed{h_{t-1}}, \boxed{x_t}\right)$$

cell state

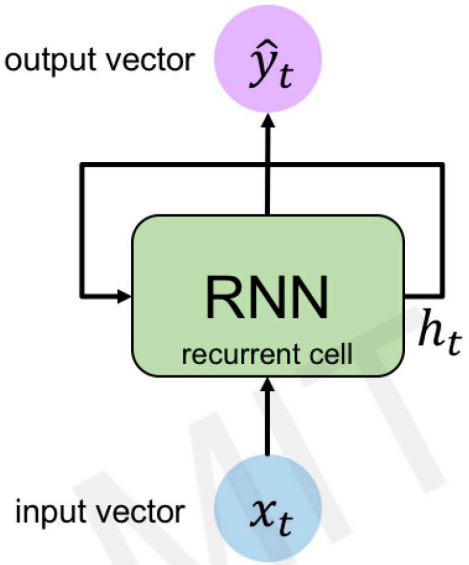
function parameterized by W

old state

input vector at time step *t*

Note: the same function and set of parameters are used at every time step

RNN state update



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{\boldsymbol{hy}}^{\boldsymbol{T}} h_t$$

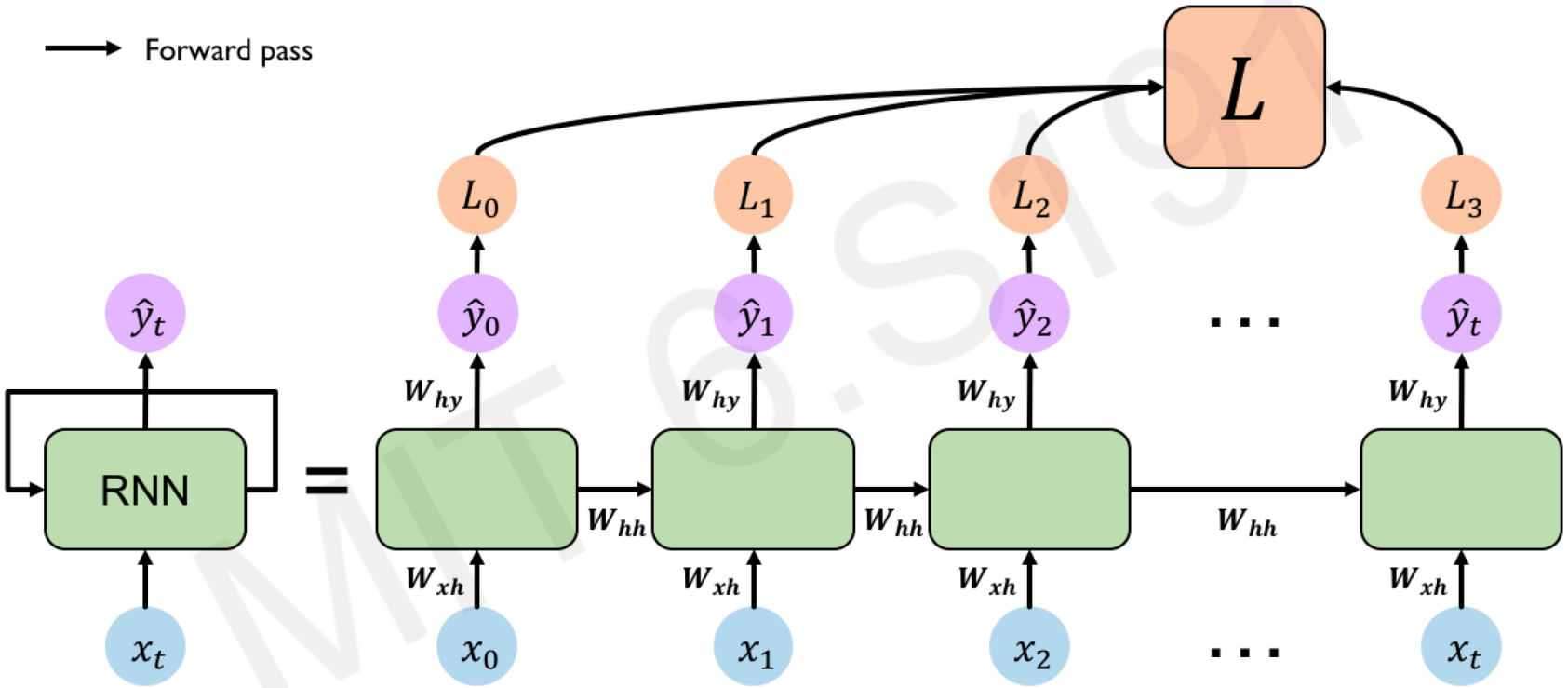
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

Input Vector

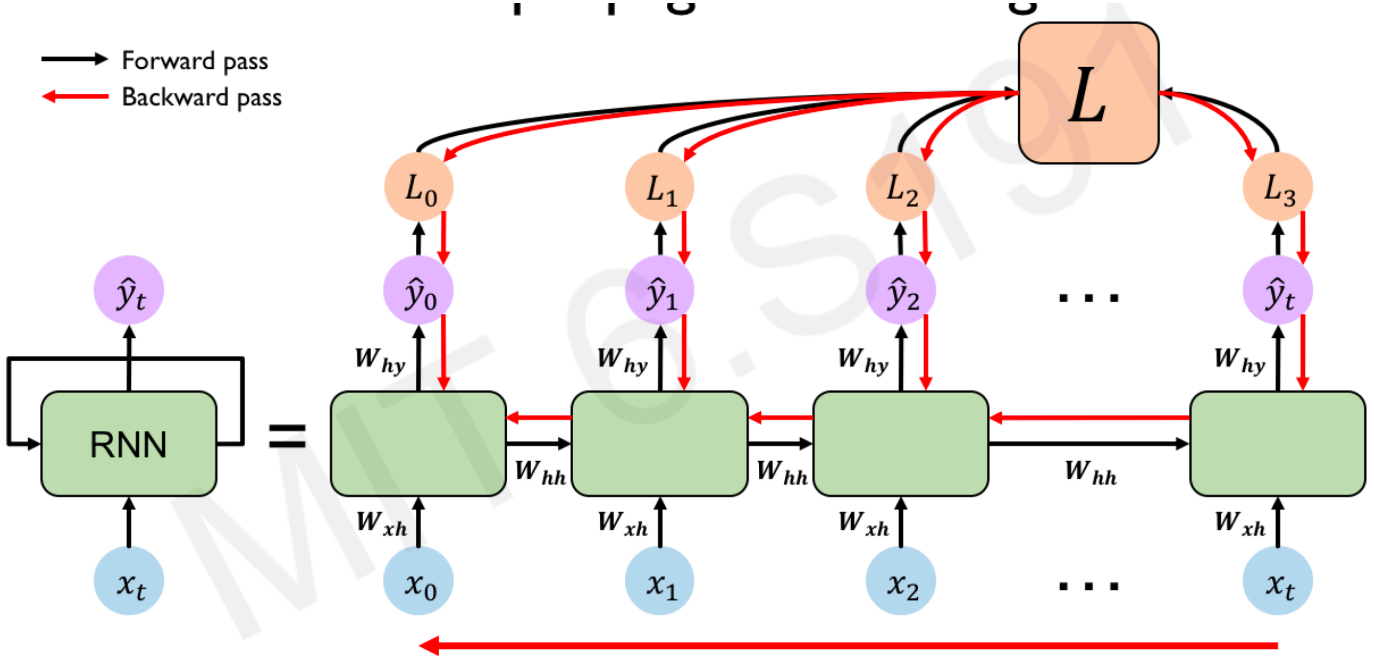
$$x_t$$

RNNs - computational graph over time

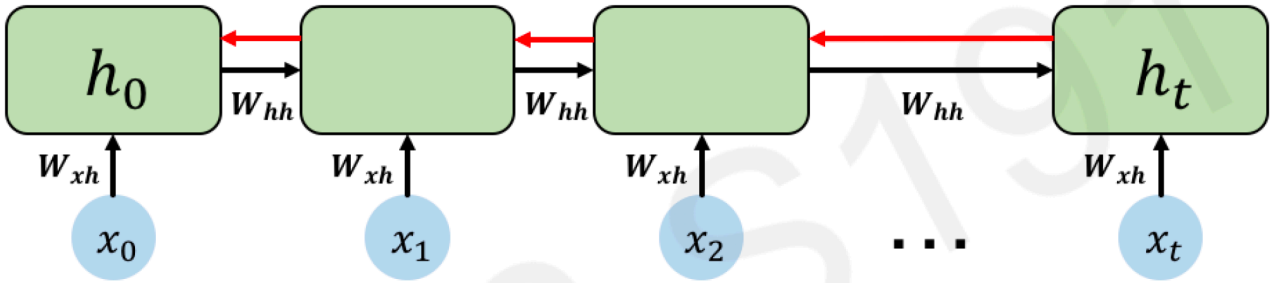



```
rnn = nn.RNN(input_size, hidden_size, out_size)
```

Backpropagation through time



Vanishing gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation!**

Many values > 1:
exploding gradients

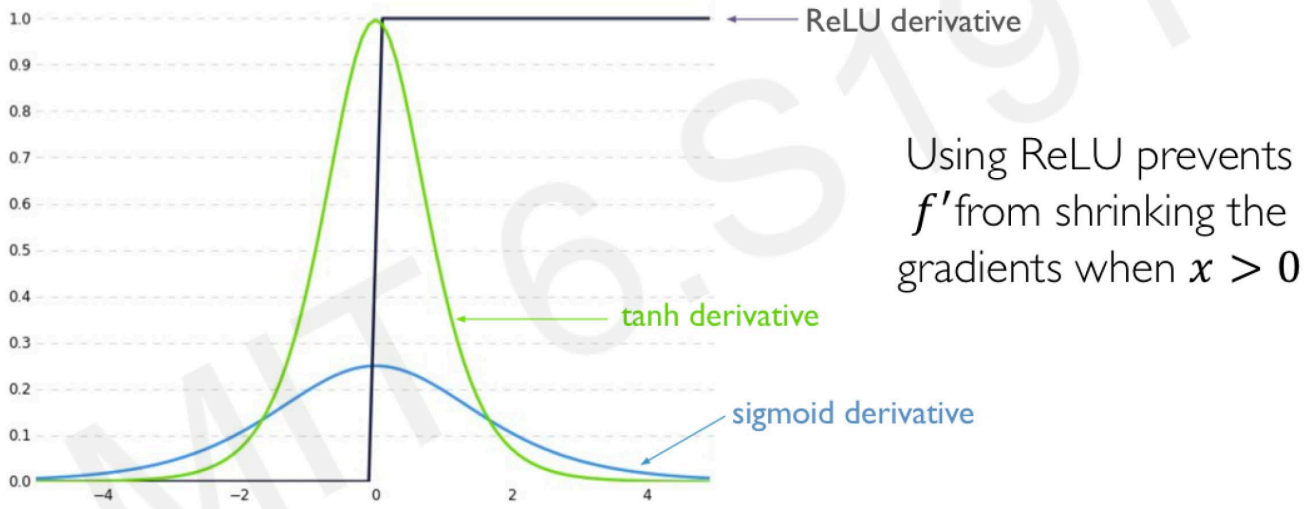
Gradient clipping to scale big gradients

Many values < 1:
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

Trick 1 - Activations

- Activation functions



Using ReLU prevents f' from shrinking the gradients when $x > 0$

Trick 2 - initialise parameters

Initialize **weights** to identity matrix

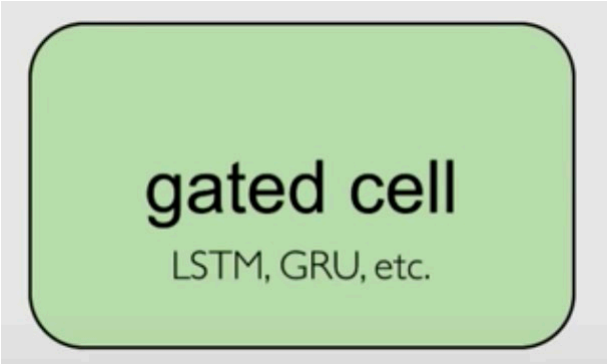
Initialize **biases** to zero

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

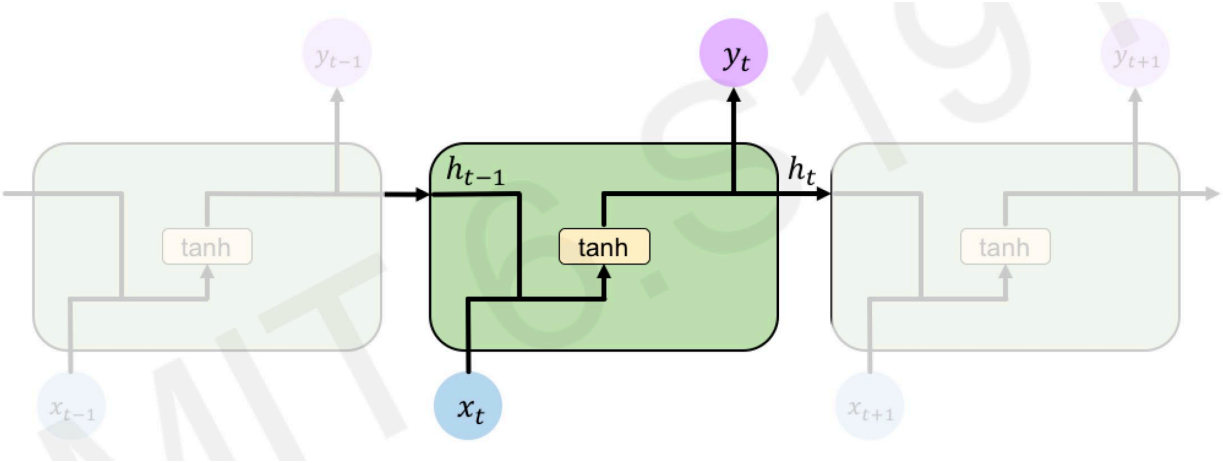
Trick 3 - Gated cells

- Use a more complex **recurrent unit with gates** to control what information is passed through.
- Long-short term memory (LSTM) networks rely on a gated cell to track information throughout many time steps.



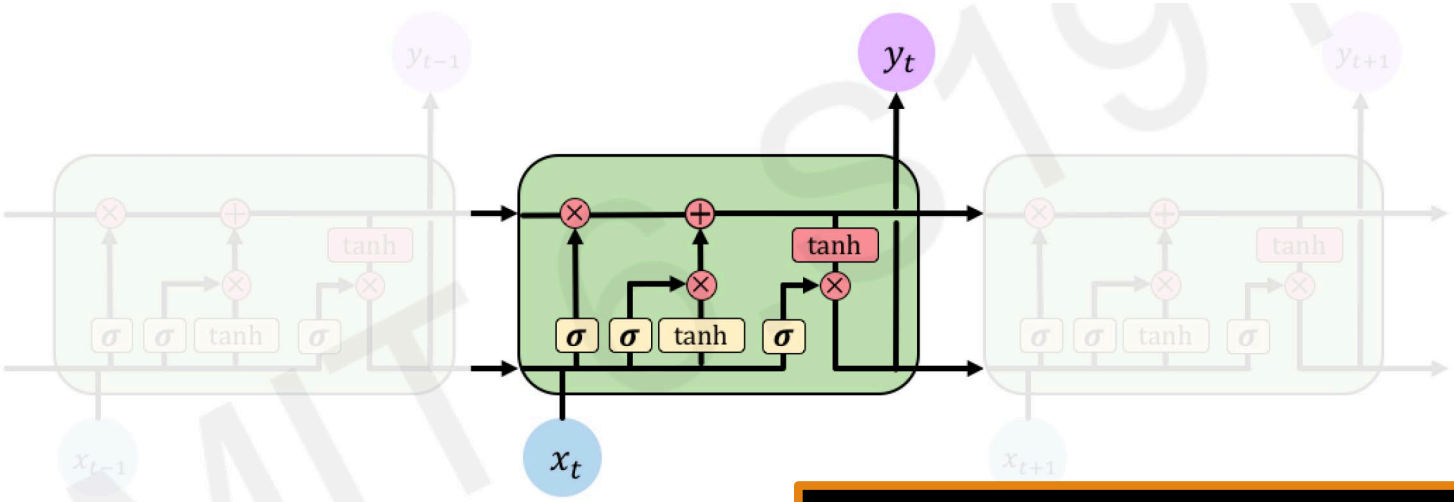
Standard RNN

- Simple computation node



Long-Short term memory cells

- Contain computational blocks that **control information flow**
  - i.e., Gates: Forget, Update, Store, Output
- Able to **track information throughout many time steps**

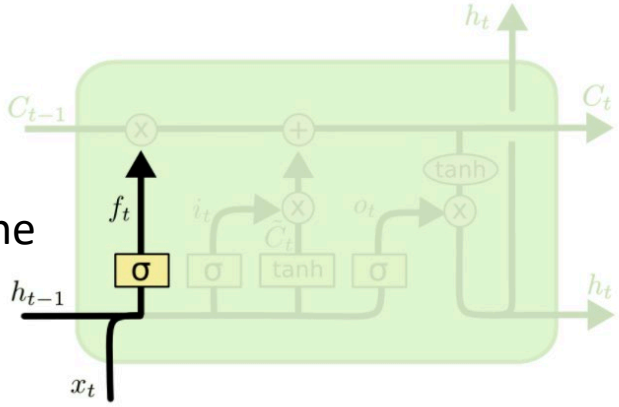



```
rnn = nn.LSTMCell(h_0, c_0)
nn.LSTM(hidden_size, hidden_size, out_size)
```

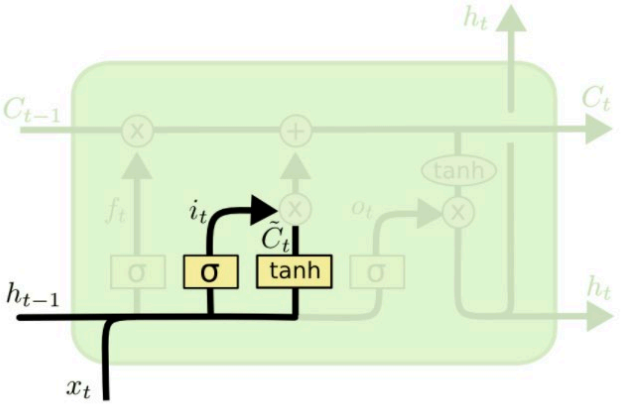
LSTM

- Info is added or removed through structures called ‘Gates’

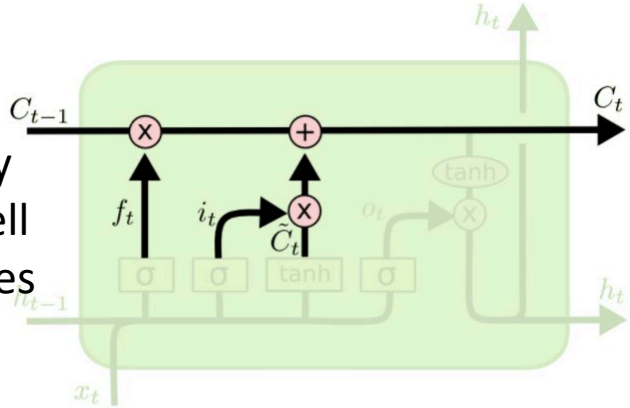
Forget irrelevant parts of the previous state



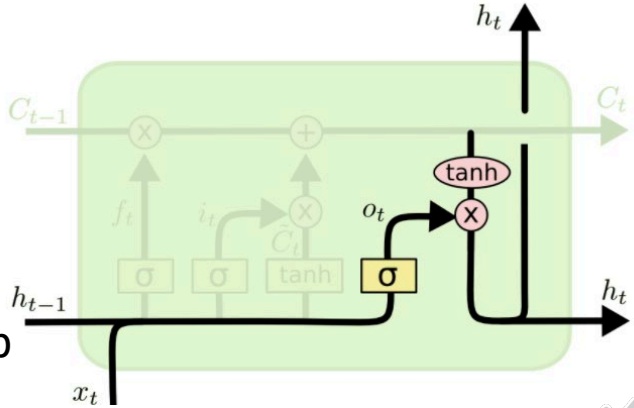
Store relevant new info into cell state



Selectively **Update** cell state values

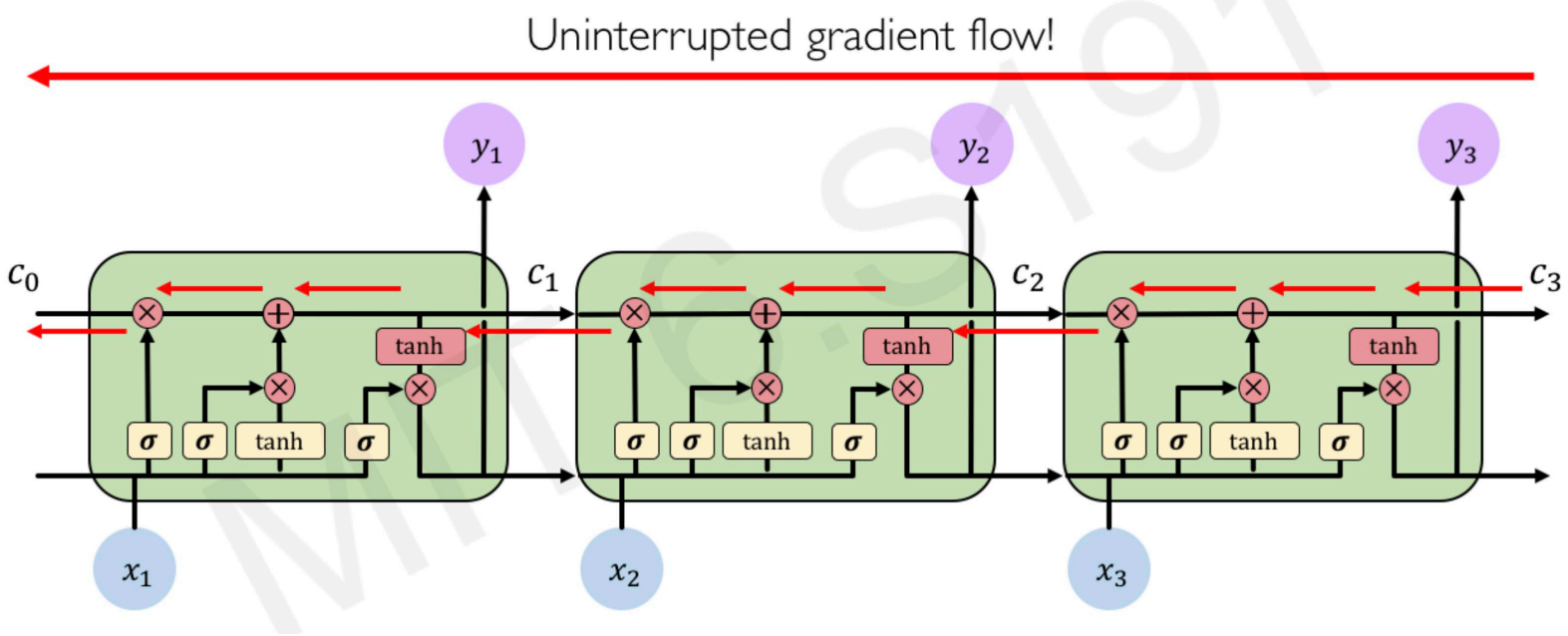


Output controls info sent to next time step



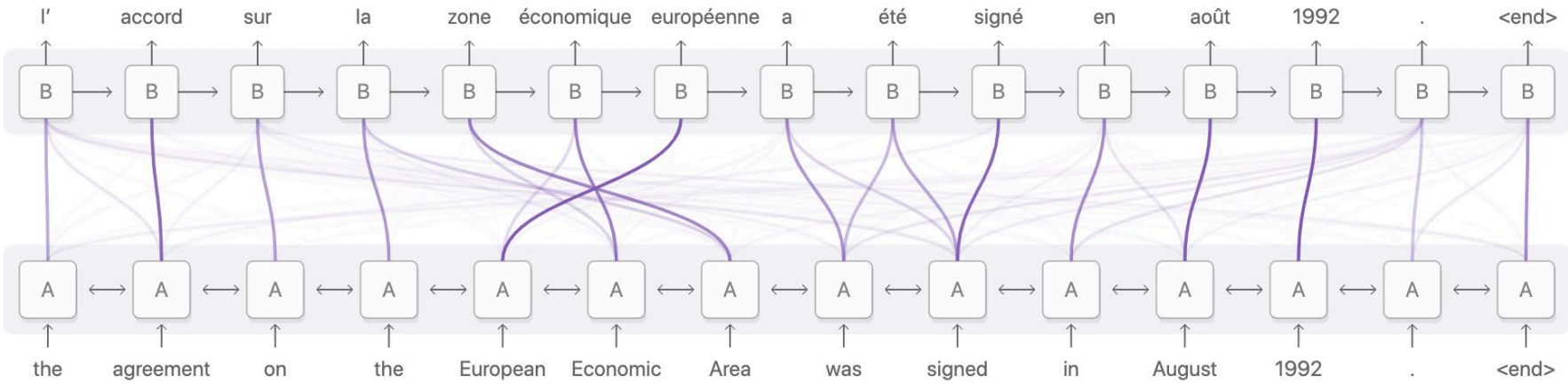
LSTM Gradient Flow

- The internal cell state causes an:



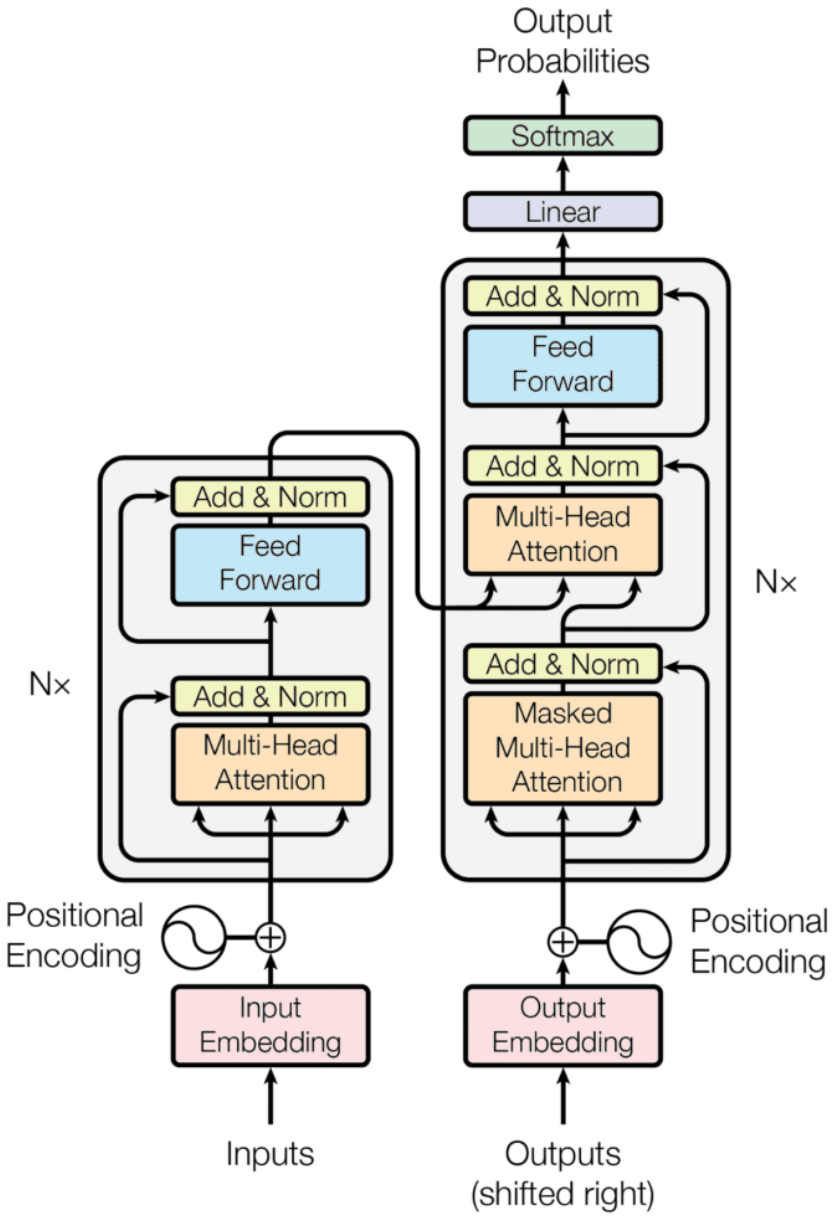
Other memory structures

- Gated recurrent units (GRUs): added “peephole connections.” This means that we let the gate layers look at the cell state.
- Attention networks (transformers)



Transformer

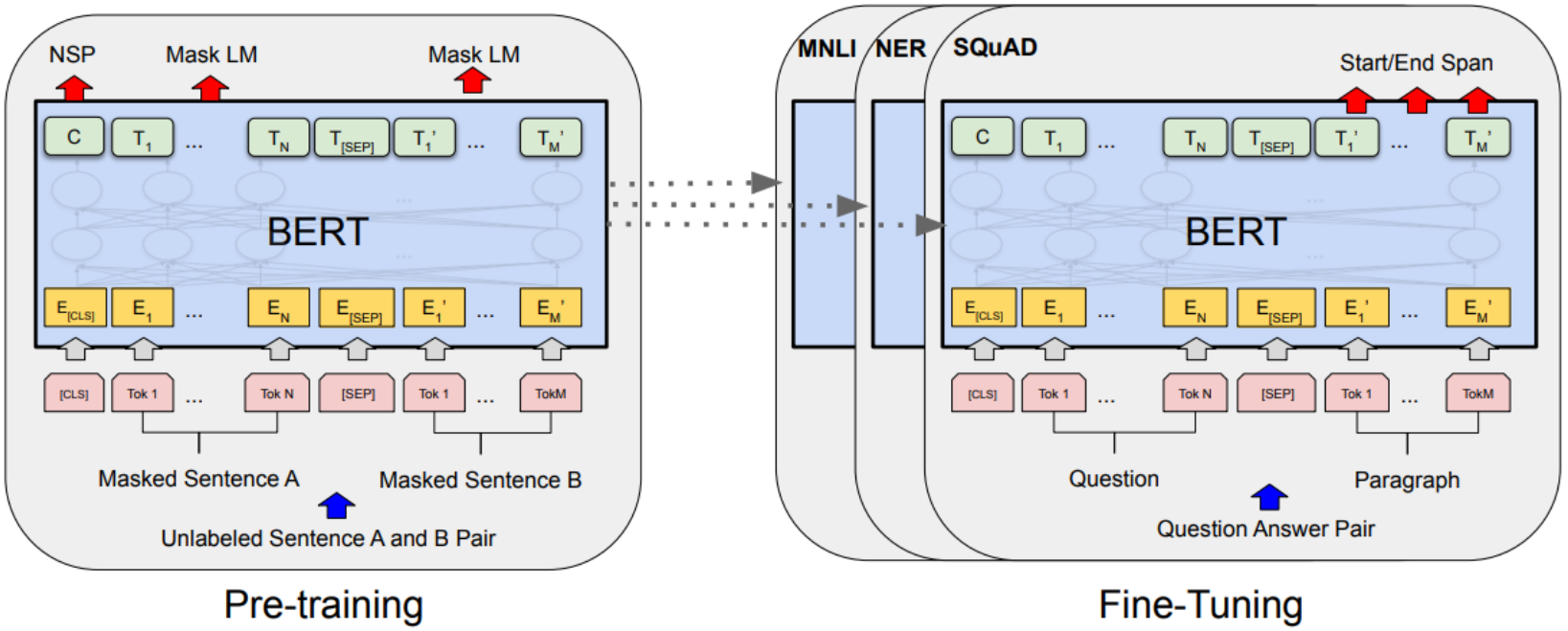
- Encoder (left)
  - 6 layers of Self-attention + Feed-forward
- Decoder (right)
  - 6 layers of Self-attention, Encoder-Decoder Attention and Self-attention
- Self-attention allow for capture of contextual information and dependencies, and parallelization



Source: Vaswani, A., Shazeer, N., Parmar, N., et al., 2017. Attention is all you need. *NeurIPS*.

BERT

- BERT: Bidirectional Encoder Representations from Transformers
  - Multi-layer bidirectional Transformer encoder (BERT Base=12, Large=24)
  - Pre-training by Masked Language Model (MLM) and Next Sentence Prediction (NSP)



Credits

- Images thanks to:
  - MIT 6.S191
  - https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba
  - https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
  - https://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - https://colah.github.io/
  - https://www.deeplearningbook.org/contents/convnets.html
  - https://github.com/BlackBindy/MNIST-invert-color
  - https://www.slideshare.net/GauravMittal68/convolutional-neural-networks-cnn
  - http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf
  - https://www.jeremyjordan.me/convnet-architectures/#resnext
  - https://www.superdatascience.com/ppt-the-ultimate-guide-to-convolutional-neural-networks-cnn/