

Deep Neural Network for NLP

Sequential NLP Data

Sequential Data

- Characters in words
- Words in sentences
- Sentences in paragraphs
- Paragraphs in documents

NLP is full of **sequential** data

Dependencies in Language

Dependencies: the relationship between two words (it can be semantic or syntax)

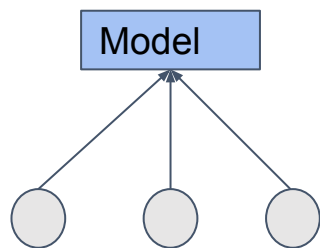
- It is equal to the sequential information contained in the sequence data.

Long-distance Dependencies in Languages

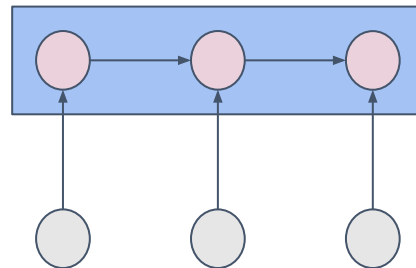
Examples

- He does not have very much confidence in himself
- The rain has lasted as long as the life of clouds
- The *trophy* would not fit in the brown *suitcase* because it was too small

Sequential NLP Data



Sequence Model

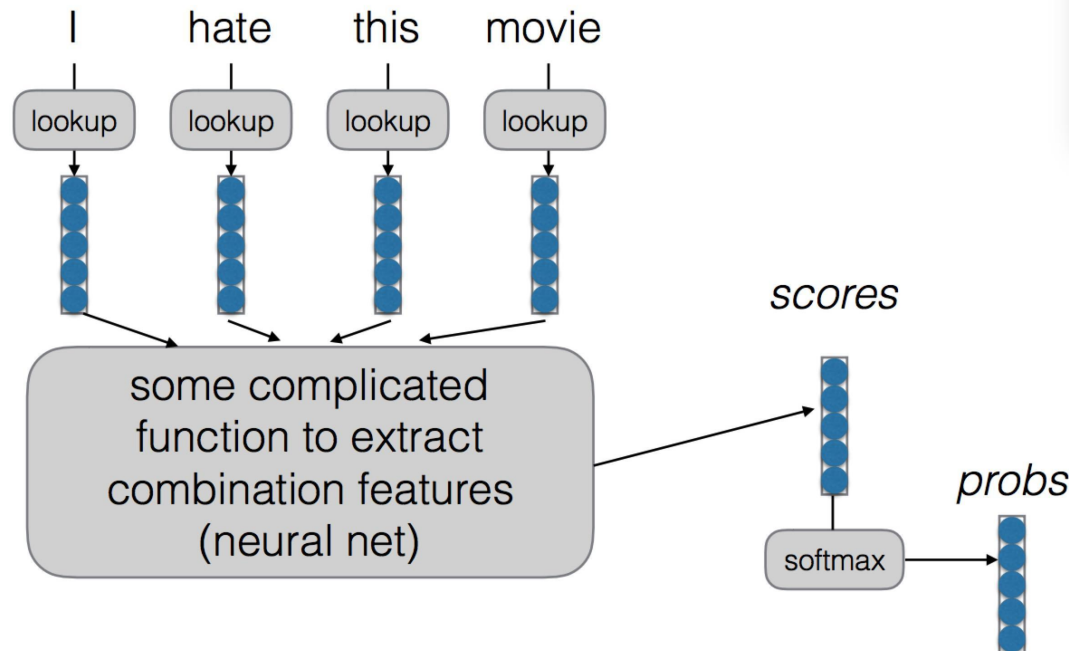


Machine learning models should capture this kind of sequential information in NLP data.

Complex Semantic

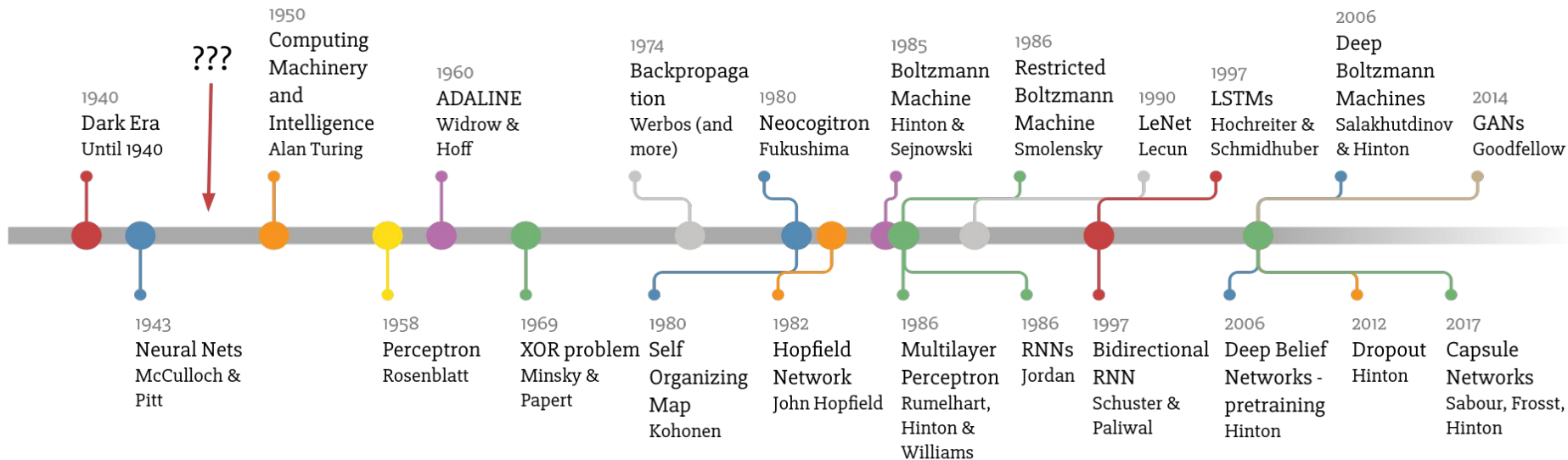
1. **Input Text:** a sequence of words;
2. **Through Word Embedding Look-up:** a sequence of word vectors;
3. Neural networks is applied upon the vector sequences to learn semantic **composition** for final prediction;

→ Human understand the word meaning firstly, then get the whole sentence meaning by composing these words' meaning together.



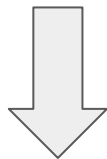
Intro to RNN

Deep Learning Timeline



Examples

- Tagging Task: Assuming we have a predefined set of tags, we assign a tag to each word in input sentences.
- Given an input sentence: I would like to arrive at **Singapore** on **Mar. 29**.



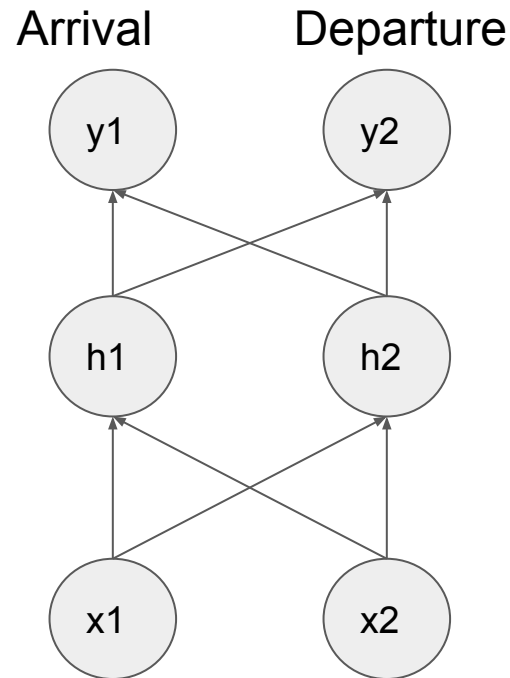
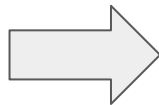
Singapore -> Dest. Place

Mar. 29 -> Time of Arrival

Feed-forward Network

- Input: Each word (word vector)
- Output: Prob. Scores that the input word belong to the tag

Singapore



Confusing Case

- Input Sentence 1:

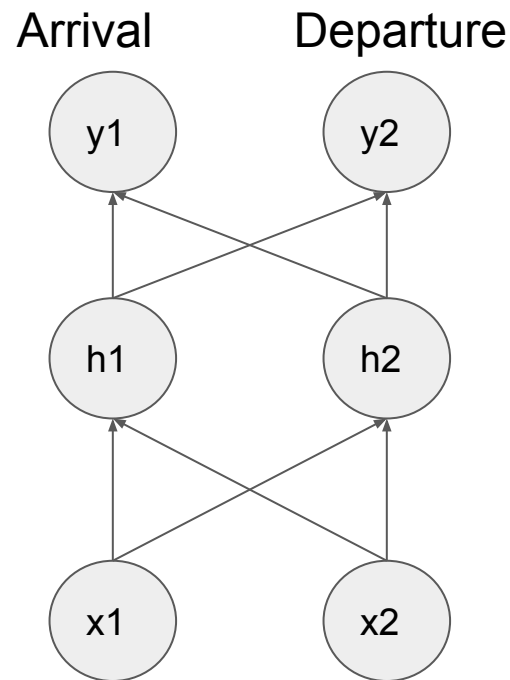
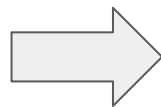
I would like to arrive at **Singapore** on **Mar. 29**.

- Input Sentence 2:

I would like to leave **Singapore** on **Mar. 29**.

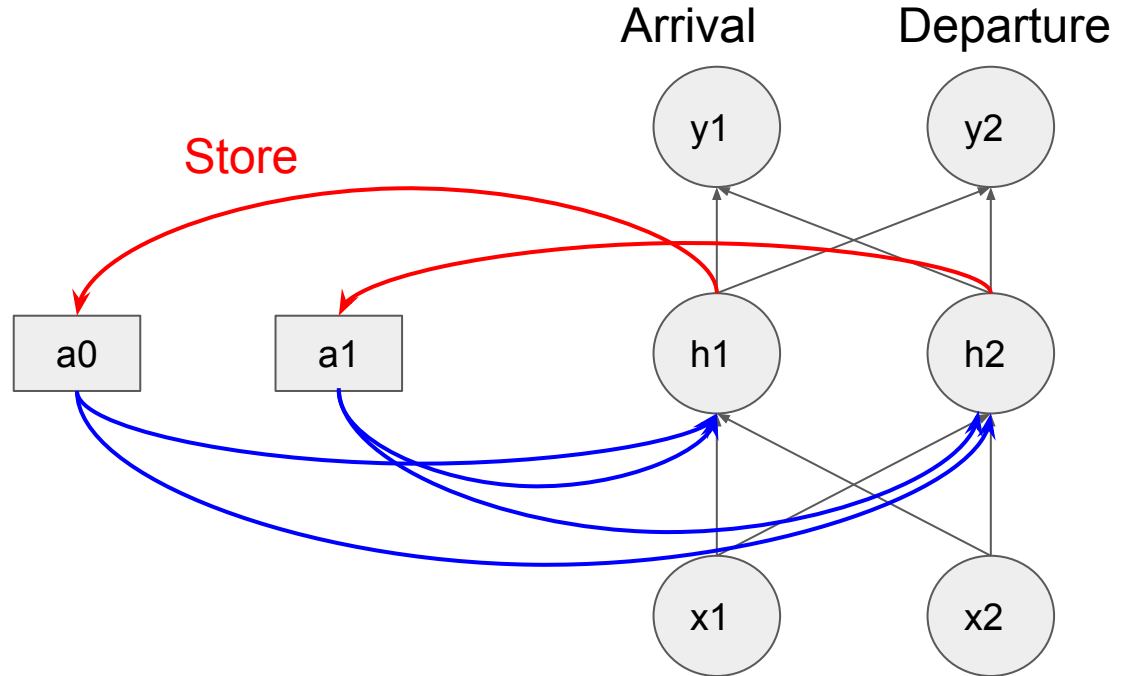
**Neural Network
needs Memory**

Singapore



Neural Network with Memory

The outputs of hidden layer are stored in the memory.



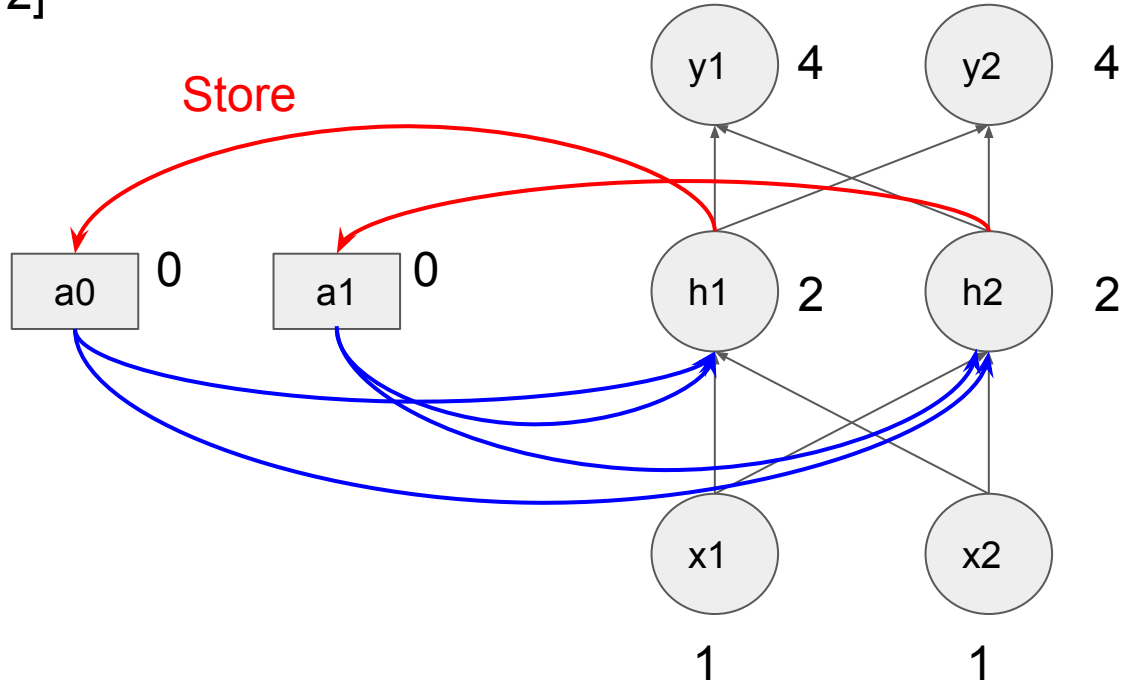
Memory can be considered as another input

Neural Network with Memory

Input Seq : [1, 1] [1, 1] [2, 2]

Output Seq: [4, 4]

Given initial values



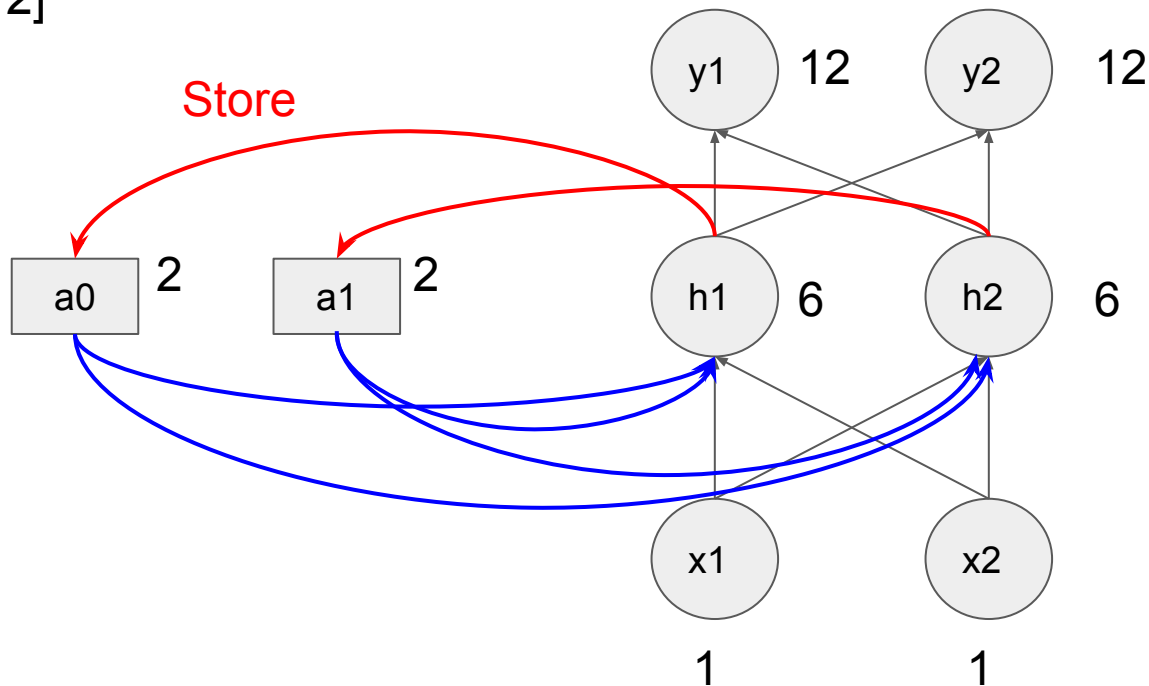
All weights are 1, no bias,
linear activation

Neural Network with Memory

Input Seq : [1, 1] [1, 1] [2, 2]

Output Seq: [4, 4] [12, 12]

Using previous
hidden output

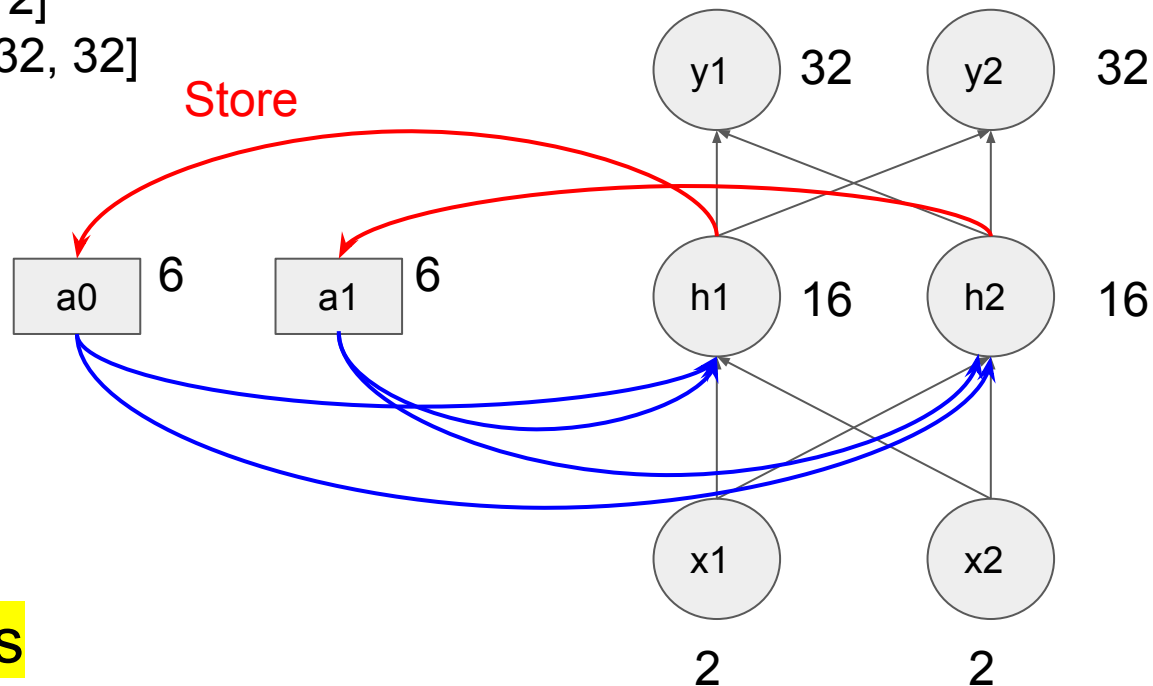


All weights are 1, no bias,
linear activation

Neural Network with Memory

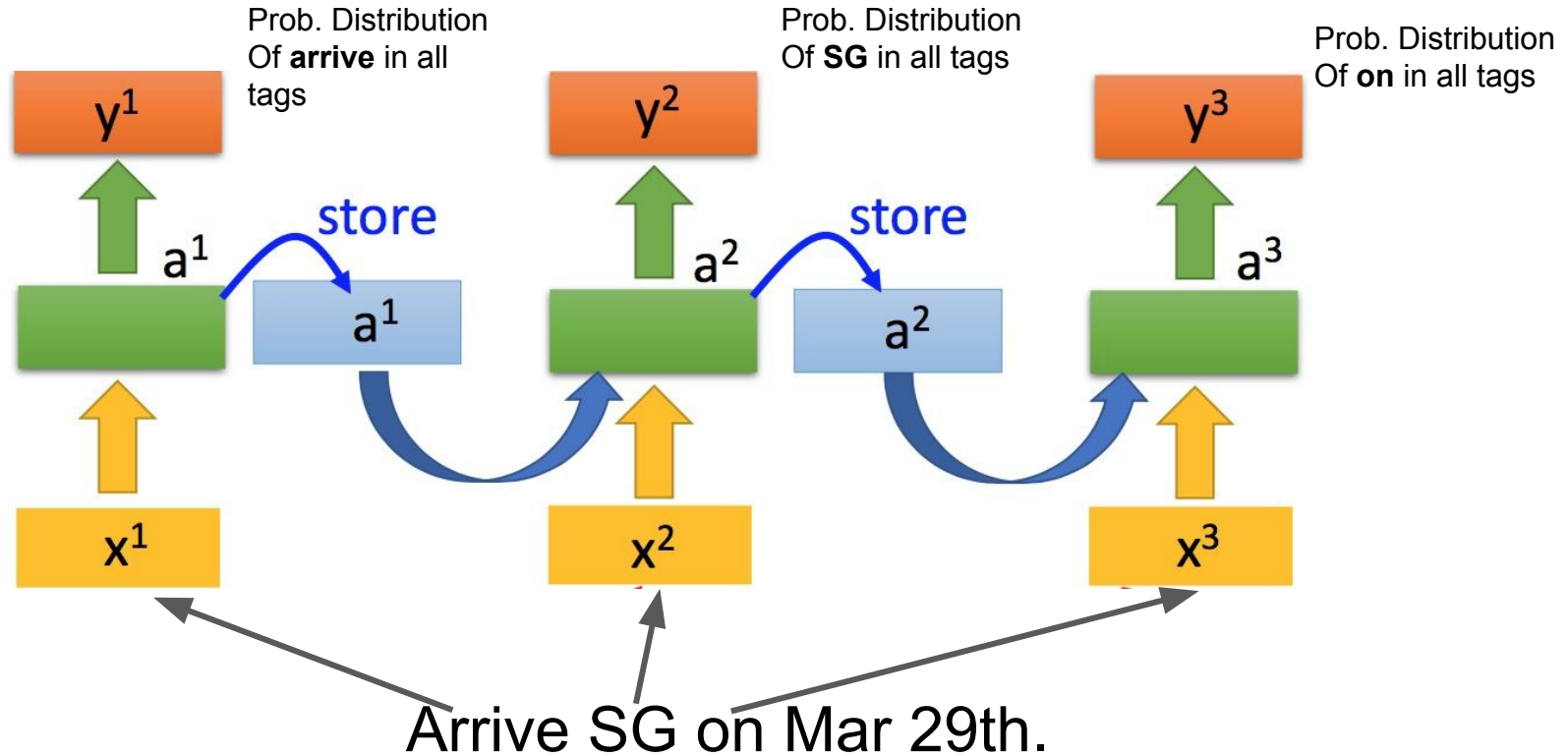
Input Seq : [1, 1] [1, 1] [2, 2]
Output Seq: [4, 4] [12, 12] [32, 32]

Try [1, 1] [2, 2], [1,1]



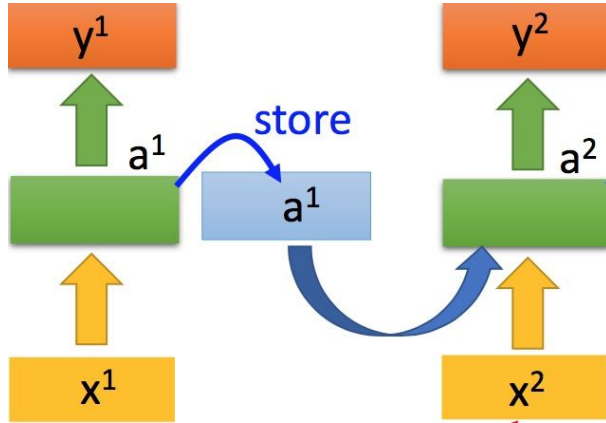
Sequence Order Matters

Back to Our Case



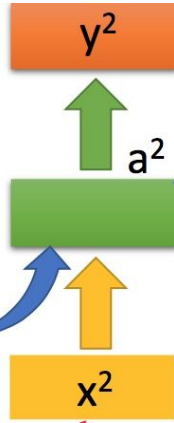
Back to Our Case

Prob. Distribution
Of **arrive** in all
tags



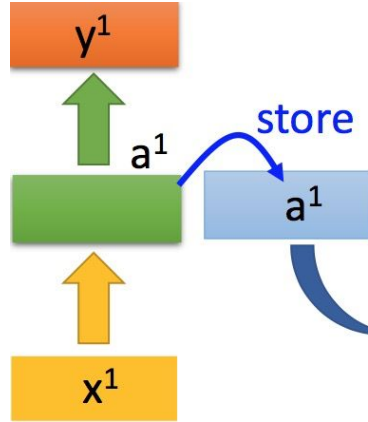
Arrive

Prob. Distribution
Of **SG** in all tags



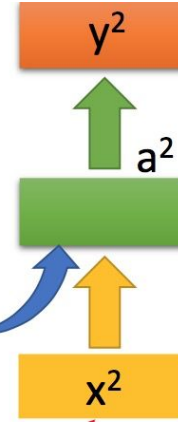
SG

Prob. Distribution
Of **leave** in all
tags



Leave

Prob. Distribution
Of **SG** in all tags



SG

Are these two probs. distribution of SG same ? And Why?

Recurrent Neural Network (Elman 1990)

- Recurrent neural network is proposed to utilize information from **previous** time steps and **current** information to make reasoning at the current step

- A is the neural network that take x_t in and generate h_t

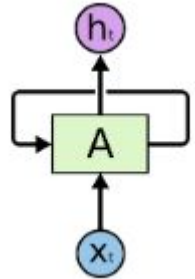


Image credits to Colah.

- **A loop** allows information to be passed from one step of the network to the next.

Allows information to persist

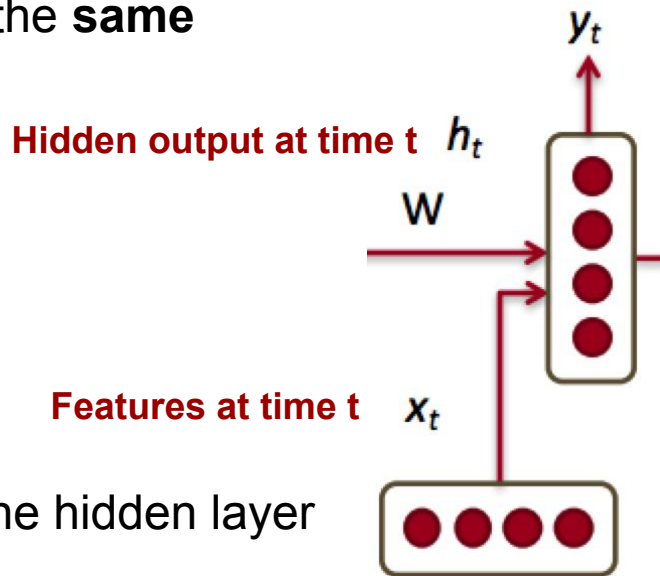
Neuron computation of RNN

- Model parameters are tied across all time steps (run the **same** RNN cell)

$$h_t = \sigma \left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]} \right)$$

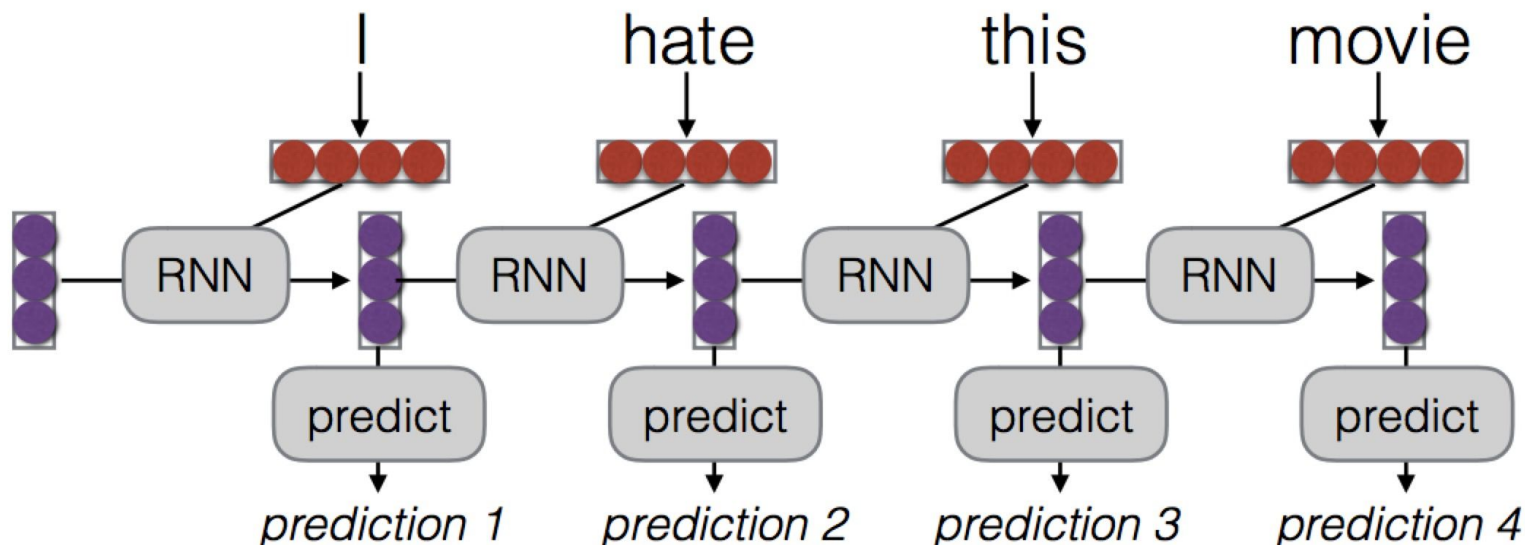
$$\hat{y}_t = \text{softmax} \left(W^{(S)} h_t \right)$$

- We need $h_0 \in \mathbb{R}^{D_h}$ as the initialization vector for the hidden layer at time step **0**.
- Inputs enter and move forward at each time step



Unrolling in Time

- Process the NLP sequence data



RNN's Bottleneck

- RNN is not suitable for **parallel** computation.
- RNN's training is not easy
 - Gradient Vanishing
 - Gradient Exploding
 - Reminder on gradient descent algorithms:

$$W = W - \alpha dL/dW$$

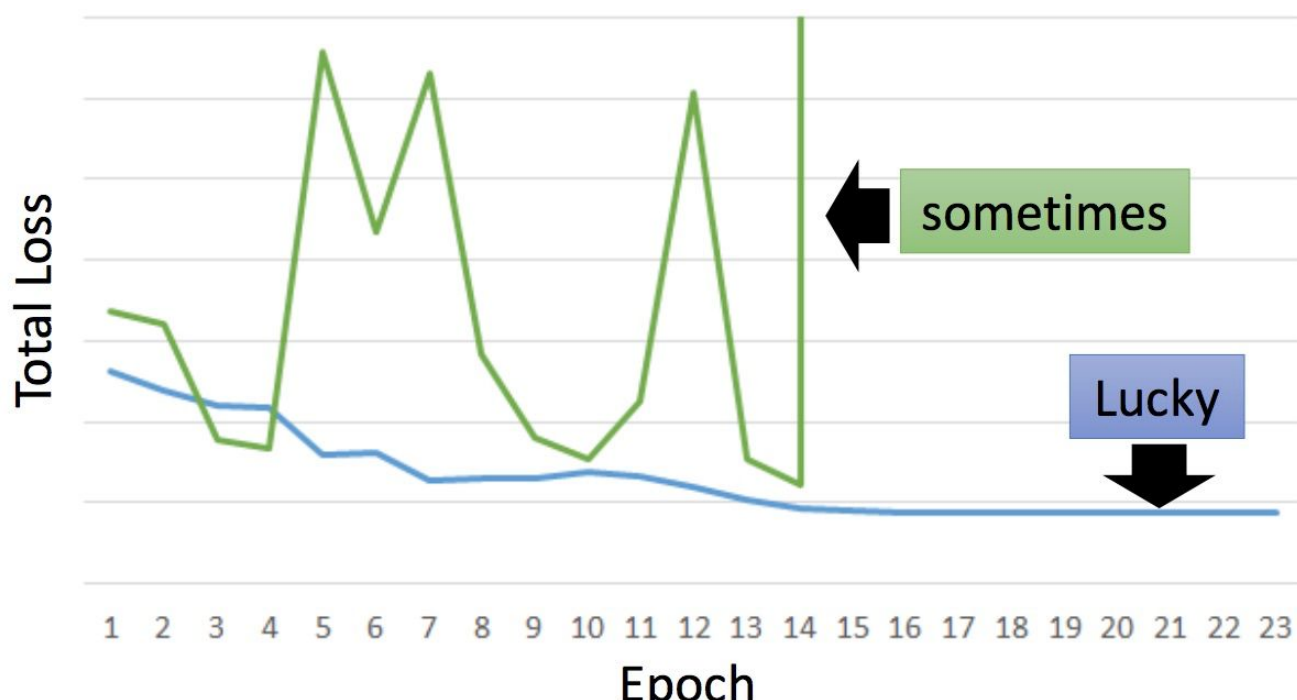
W is the layer weights;

L is the loss function;

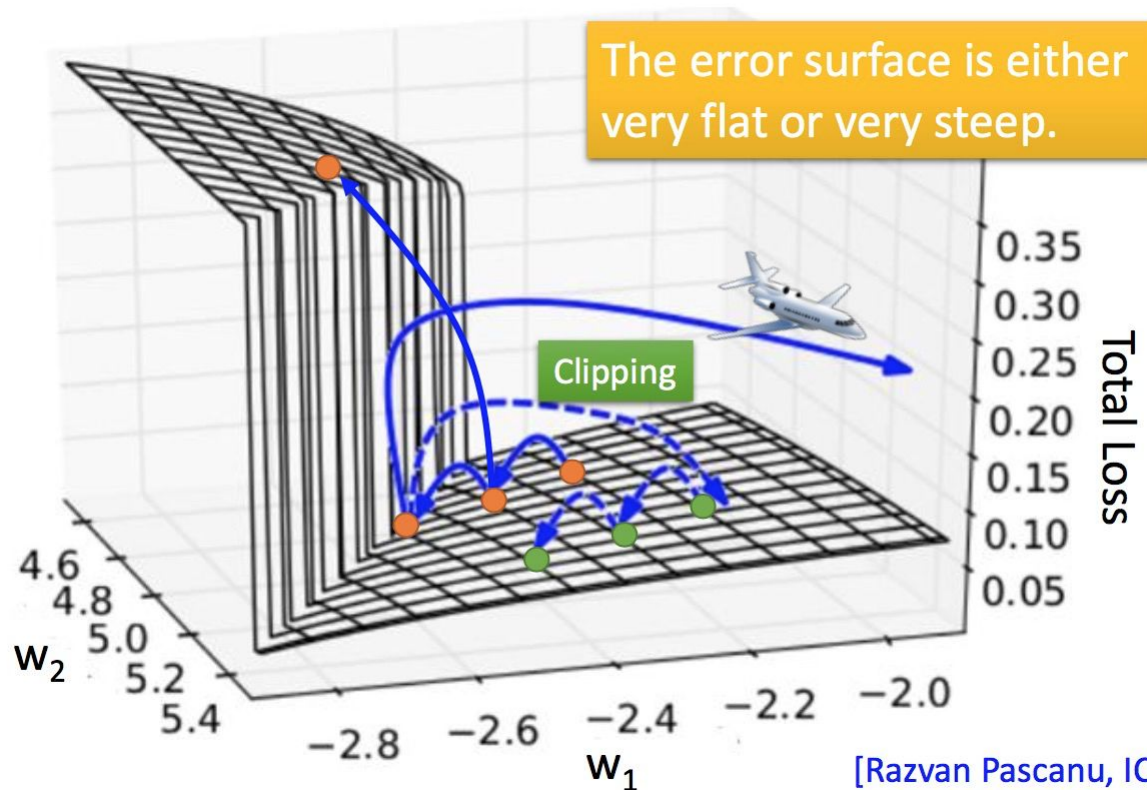
Alpha is the learning rate, which represents how aggressive that our model parameters are updated.

RNN Training is Hard

- Real experiments on Language Models



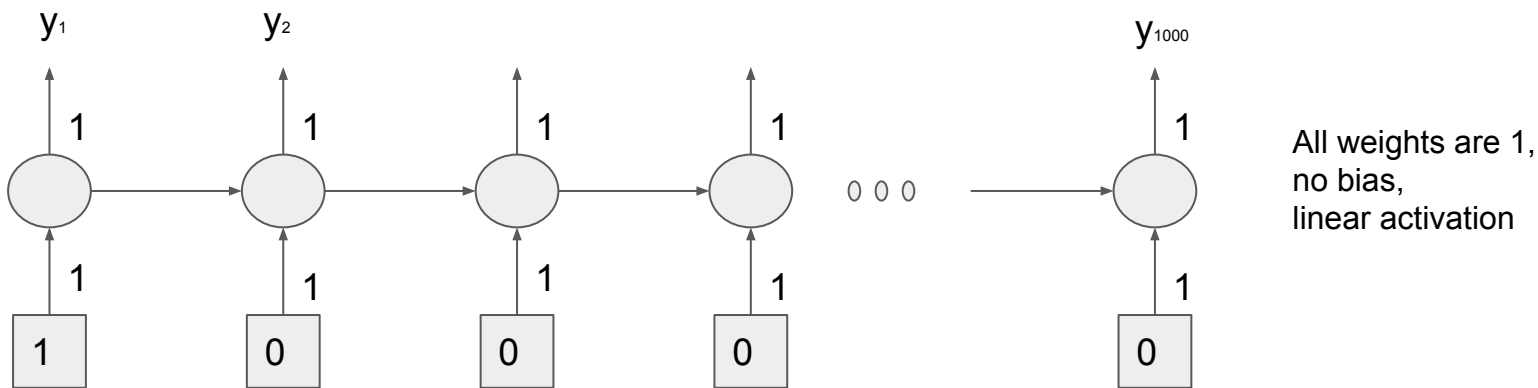
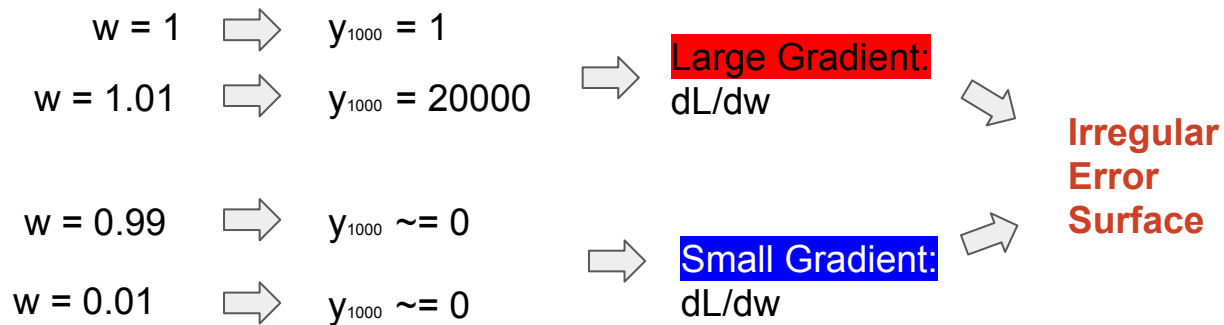
Rough Error Surface of RNN



**Exploding/Vanishing
Gradient**

[Razvan Pascanu, ICML'13]

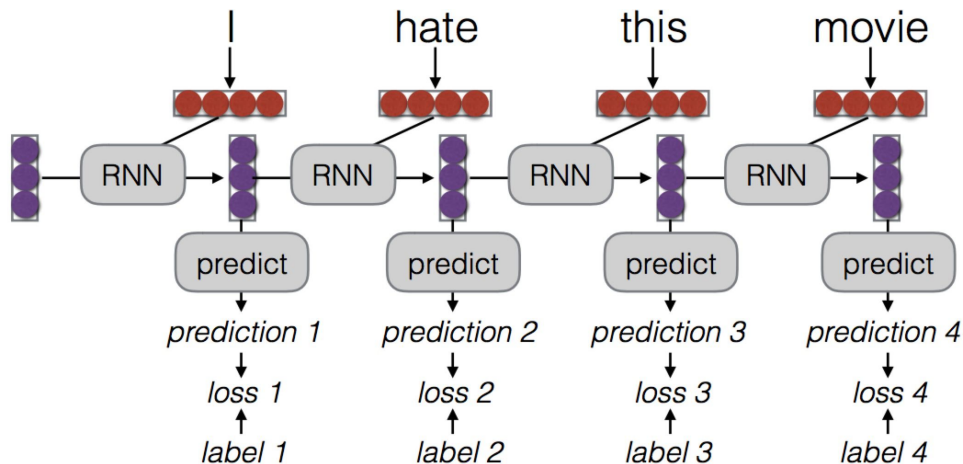
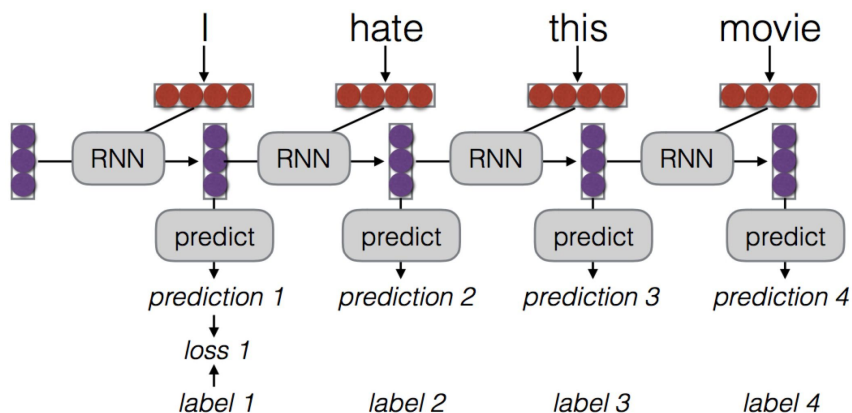
Toy Example



Loss Computation

- Total Loss

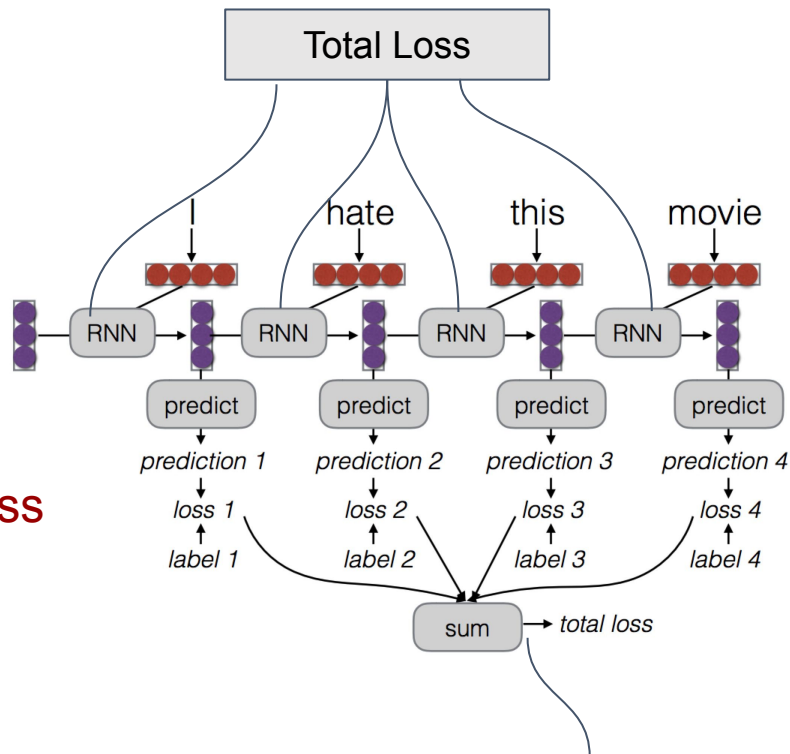
- Total loss is the sum of loss computed on each time step.



BPTT

- For RNN gradient computation, Backpropagation Through Time
- Weight is the same for all the time steps
- Inputs from many time step ago can modify output

1. Shared Model Parameters across time steps
2. Accumulated Derivatives



BPTT Equation

- RNN formulation

$$h_t = \sigma \left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]} \right)$$

$$\hat{y}_t = \text{softmax} \left(W^{(S)} h_t \right)$$

- **Total error** is the sum of each error at time steps **t**

$$\frac{\partial E}{\partial W} = \sum_{t=1}^T \frac{\partial E_t}{\partial W}$$

- Apply **chain rule** for error at certain time step **t**

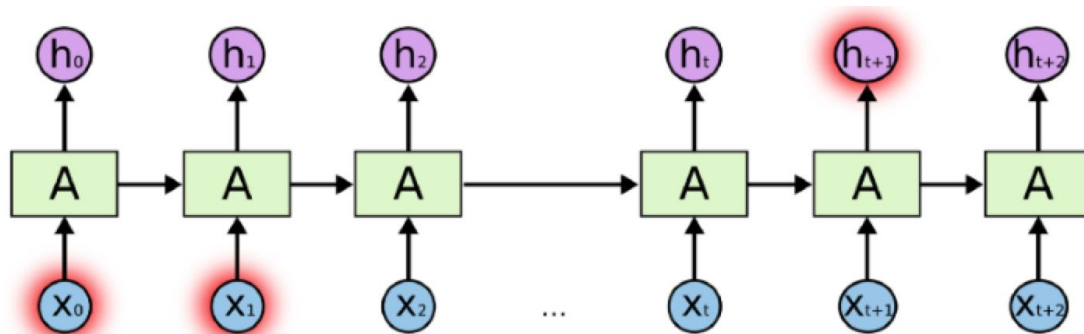
$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Exploding Gradient Solutions

- Truncated BPTT
- Clip gradients at threshold
- RMSprop to adjust learning rate

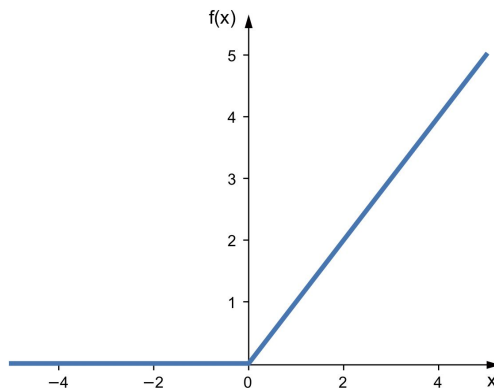
Vanishing Gradient Problem

- The error at a time step **ideally** can tell a previous time step from many steps away to change during backprop
- **Can not capture long-term dependency**
- The representation from time steps 0 and t can not travel to influence the time step $t+1$
- **Harder to detect**



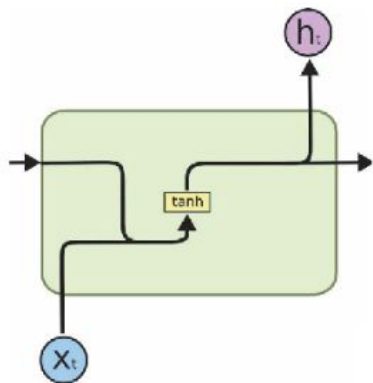
Vanishing Gradient Solutions

- RMSprop
- LSTM, GRUs (gated RNN)
- ReLu activation functions

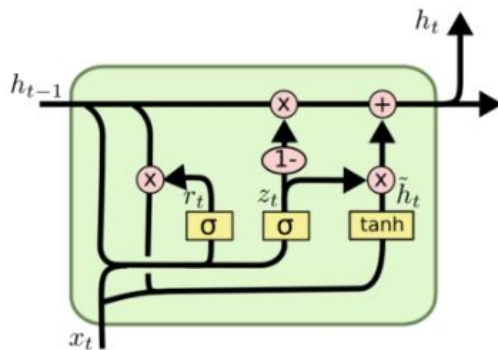


LSTM/GRU

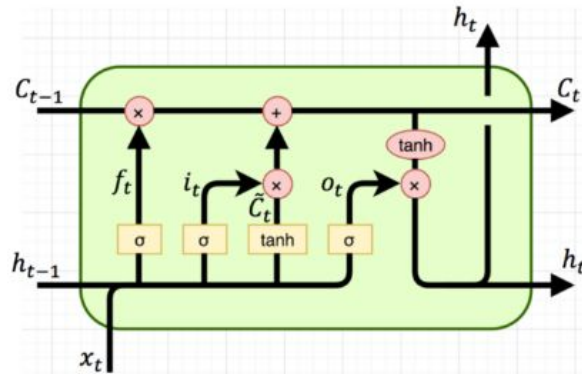
- LSTM/GRU are using gates in cell computation to control information flow



RNN



GRU



LSTM

RNN for Language Model

Recurrent Neural Network

- Recurrent neural network works in a chain way
- The method is in naturally suitable for processing sequences data
- A broad applications:
 - Speech Recognition
 - Time series Prediction
 - Language Modeling
 - Machine Translation

Language Models

- A language model computes **a probability for a sequence of words**:
 - $P(w_1, \dots, w_T)$
- Useful for machine translation/Question Answering
 - Word Ordering
 - $p(\text{the cat is small}) > p(\text{small is the cat})$
 - Word Choice
 - $p(\text{walking home after school}) > p(\text{walking house after school})$

Traditional Methods

- Probability is usually conditioned on **window of n** previous words
- An incorrect but practical **Markov** Assumption
- To estimate probabilities, compute for unigram and bigrams

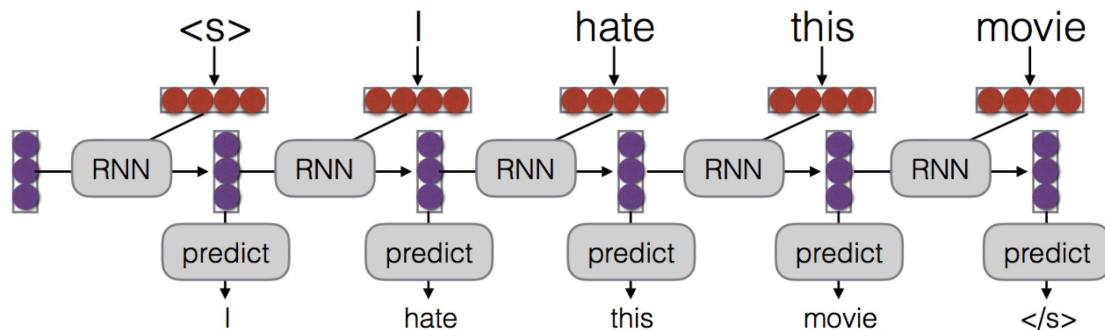
$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \qquad p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

Traditional Methods

- A lot of n-grams and extremely large combinations
 - Requires large RAM requirements
- Use one machine with 140GB RAM for 2.8days to built a model on 126 billion tokens.

RNN-based Language Model

- Language Modelling here is one of **tagging** task
- Each label/tag is the next word!



At a single time step:

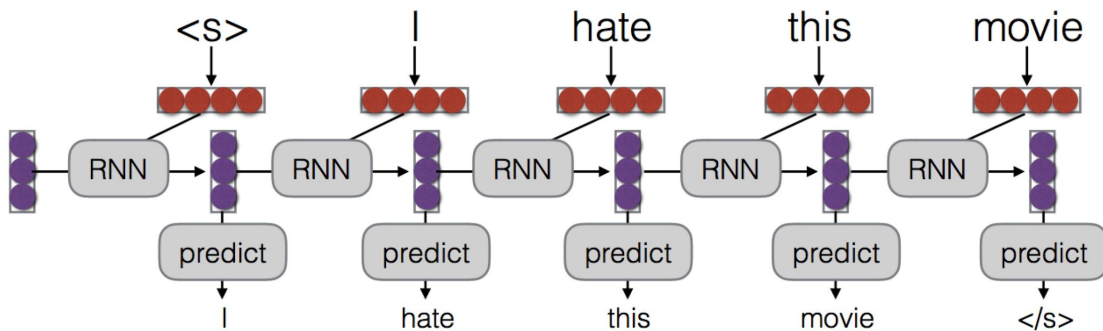
$$h_t = \sigma \left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]} \right)$$

$$\hat{y}_t = \text{softmax} \left(W^{(S)} h_t \right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \dots, x_1) = \hat{y}_{t,j}$$

RNN-based Language Model

- This task is for sequential labelling, i.e., each time step has its own target value
- At each time step, the learned hidden representation are fed into two layers:
 - Predict the target value (softmax or linear regression)
 - Compute the hidden features at the next time step



Intro to CNN

Convolutional Neural Network

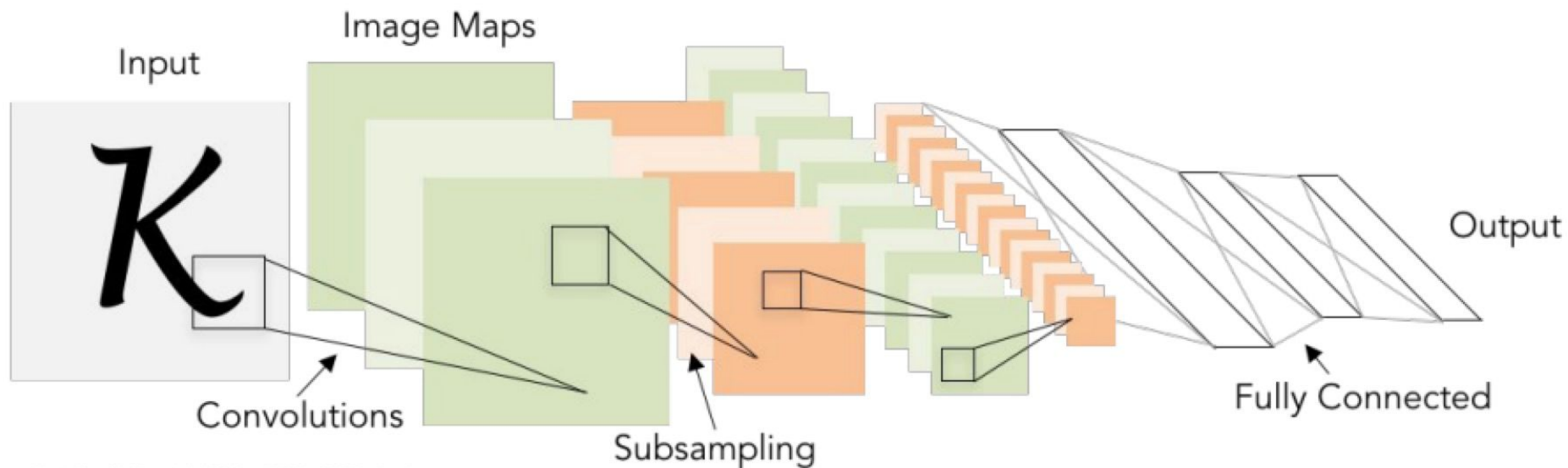
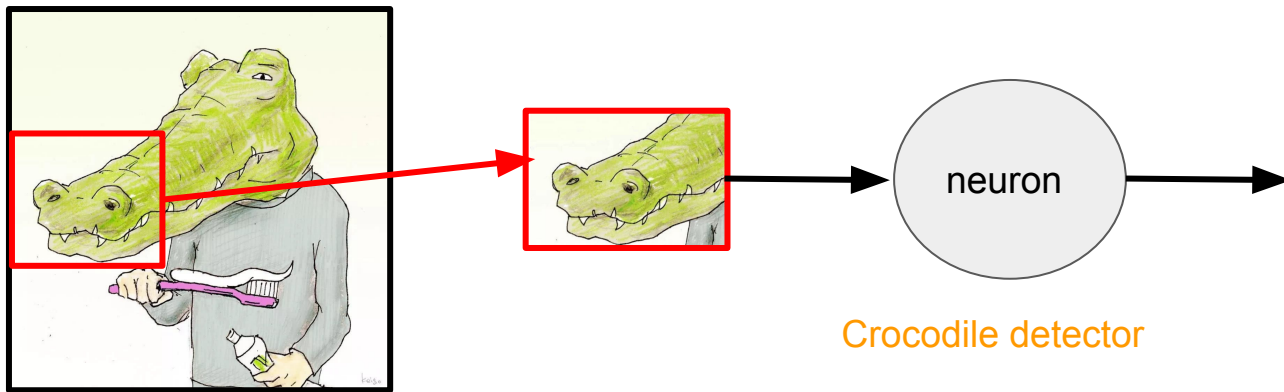


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

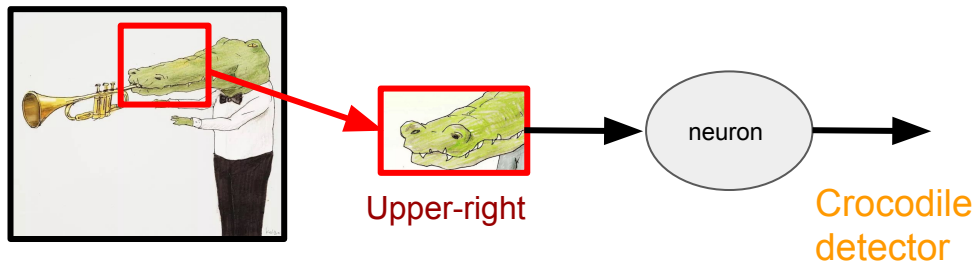
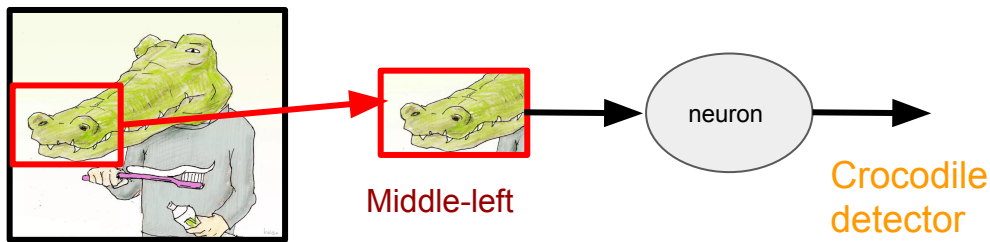
Local Features Matter

- Discriminative patterns are much smaller than the whole image
- A neuron does not have to see the whole image
- Less parameters required



Location Insensitive

- The same patterns appear in different regions
- A neuron should be location insensitive.



Subsampling Works

- Subsampling the pixels will not change the object
- We can subsample the pixels to make images smaller -> less parameters required

Crocodile



subsampling

Crocodile



CNN for Images

CNN:

1. **Convolutional Layer:** from local regions in images to feature map
2. **Pooling Layer:** reduce the dimensionality of feature maps
3. 2d example ->

Yellow shows filter weights

Green shows input

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

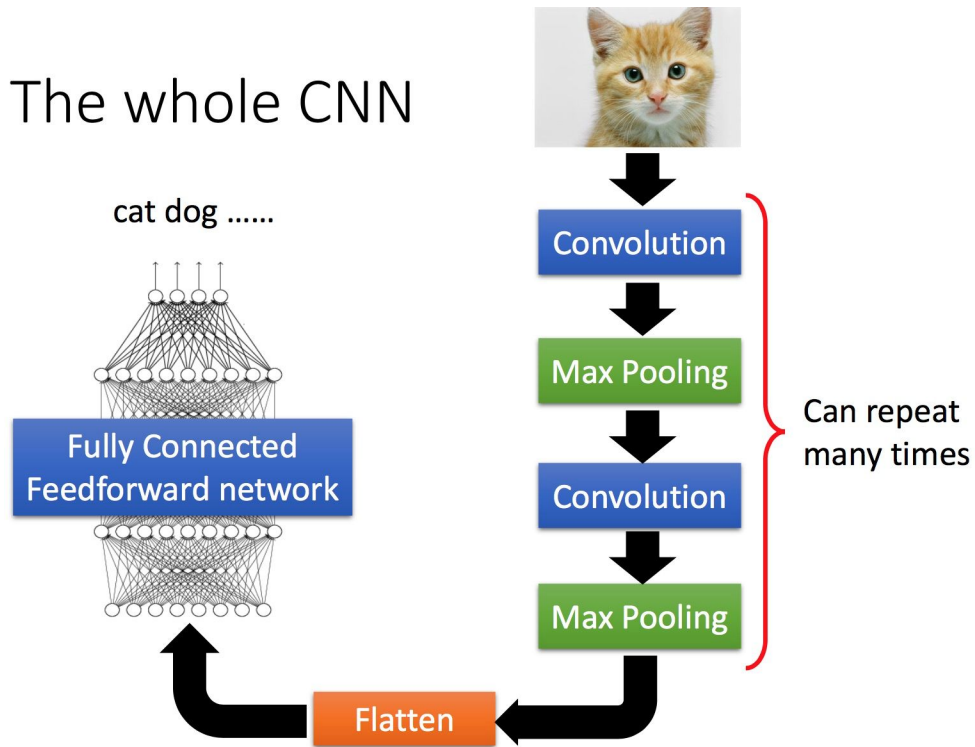
4		

Convolved
Feature

CNN for Images

- Convolution Layer:
 - Local features matters
 - Location Insensitive
- Max Pooling Layer:
 - Subsampling works

The whole CNN



CNN for Text

CNN works for Text

Images

- Local Features Matter
- Locations Insensitive
- Subsampling Works

Texts

- Key n-grams define semantics
*Pulp fiction's director is Quentin. I **am obsessed of** it.*
- Locations of key n-grams Insensitive?
*I **am obsessed of** Pulp fiction, whose director is Quentin.*
*Pulp fiction's director is Quentin. I **am obsessed of** it.*

*I owe **you** ten dollars*
***You** owe **me** ten dollars.*
- Doc. Summarization

Combinations

E.g., I hate this movie

- Compute vectors for every possible phrase
 - *I hate this movie* ----> I hate; hate this; this movie
- Compute these vectors for these phrases

Convolution Operation

Word Vectors

I
like
this
movie
very
much
!

0.6	0.5	0.2	-0.1	0.4
0.8	0.9	0.1	0.5	0.1
0.4	0.6	0.1	-0.1	0.7
...
...
...
...

0.2	0.1	0.2	0.1	0.1
0.1	0.1	0.4	0.1	0.1

Filters updated
during training

0.51

I
like
this
movie
very
much
!

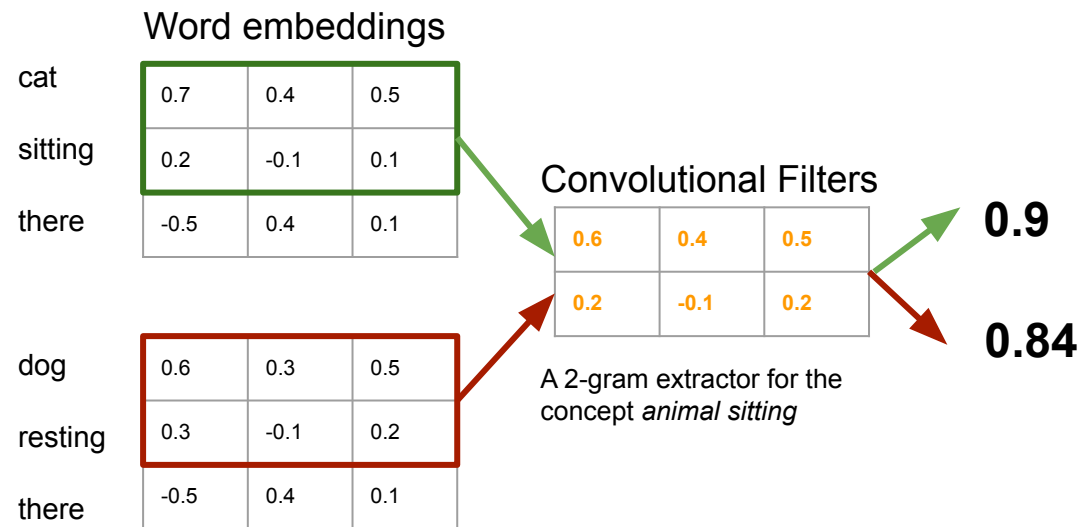
0.6	0.5	0.2	-0.1	0.4
0.8	0.9	0.1	0.5	0.1
0.4	0.6	0.1	-0.1	0.7
...
...
...
...

0.2	0.1	0.2	0.1	0.1
0.1	0.1	0.4	0.1	0.1

Feature Maps

0.51
0.53

Toy Example



- This convolution provides high activations for 2-grams with certain meaning
- Can be extended to 3-grams, 4-grams, etc.
- Can have various filters, need to track many n-grams.
- They are called 1D since we only slice the windows only in one direction

Why is it better than BoW?

Convolution Operation

$$p = \tanh \left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right)$$

Word Vector: $c \in R^k$

[]: concatenation operation

W: linear matrix $W \in R^{d \times nk}$

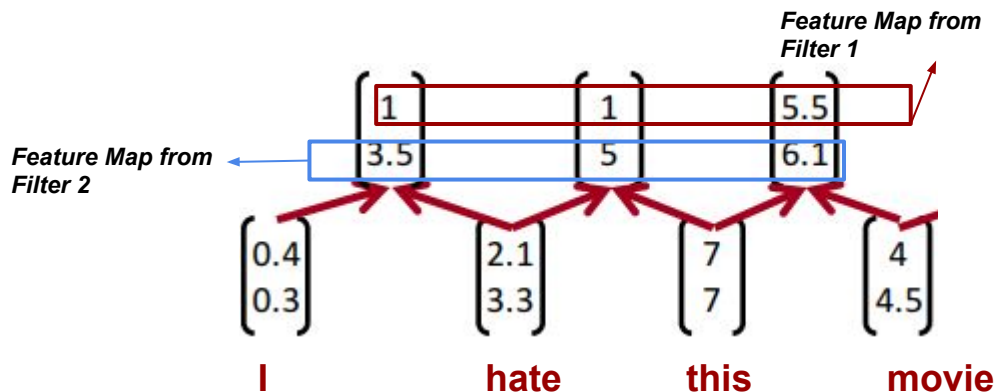
b: bias vector $b \in R^d$

W and b are the network parameters to be learned.

Convolution Layer Weights

W: linear matrix $W \in R^{d \times nk}$

- Each **row vector** in linear matrix can be regarded as one **n-gram extractor**, i.e., filter.
- Number of row d can be regarded as number of extractors.
- The output is the feature value for the input n-words.
- They are called 1D because we slide the window only in one direction.

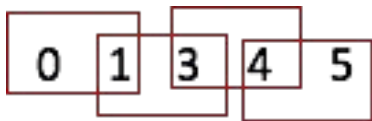


Padding

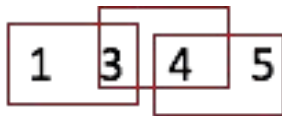
Padding: After convolution, the lengths of feature maps depends on padding

Toy Sequence: 1, 3, 4, 5

- a. To be the same as the input length **(same)**

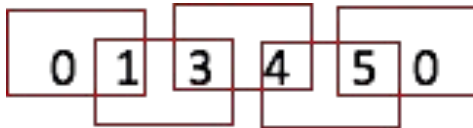


- b. Input length - window size + 1 **(valid or narrow)**



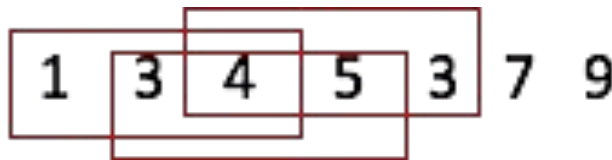
**Variable Feature
Dimension after
Convolution**

- c. Input length + window size - 1 **(wide)**

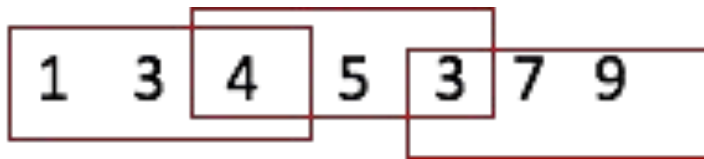


Stride

- Control how the filters move along the input sequence
- **N-stride** means that the filter convolve around the input sequence by **shifting n units every time**.



Stride = 1



Stride = 2

Pooling Operation

1. Calculates some reduction function feature-wise
2. Pooling is conducted over the sequence direction
 - a. **K-range Max pooling:** *Did u see this feature anywhere in the k window*

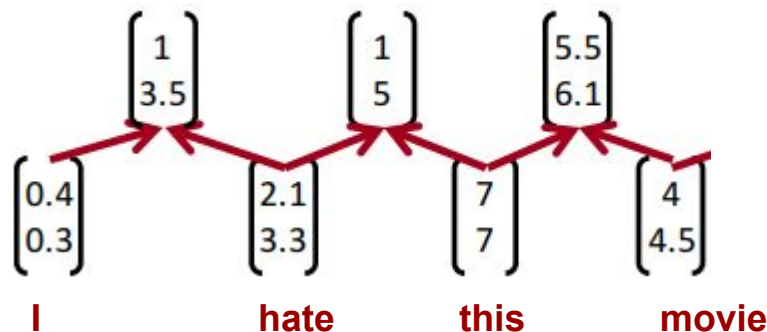
$$[5.5 ; 6.1]^T$$

- b. **Average pooling:** *How prevalent is this feature over the entire range?*

$$[2.9 ; 4.5]^T$$

Lose the order information.

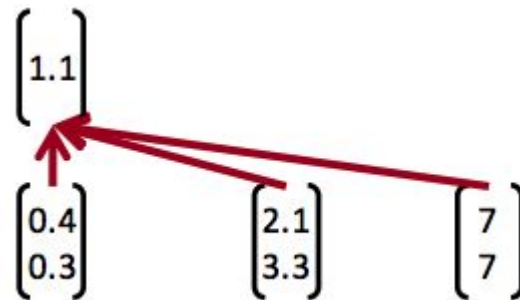
We do not care the position of the key n-grams in the sentence.
Whether the key n-grams is in the sentence or not is important.



Convolution Operation Hyper-parameters

Windows size n: how many words are considered together?

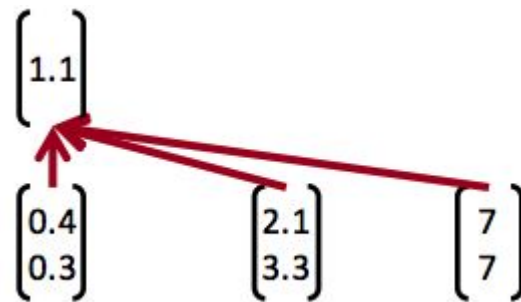
Output dimension d: how many filters are used for word windows?



Windows Size

Windows size n: capture the n-gram features in convolutional layer

- CNN **automatically** learns the values of its filters
- Compared to n-grams model, CNN is **efficient** and **compact** in representation without representing all vocabulary



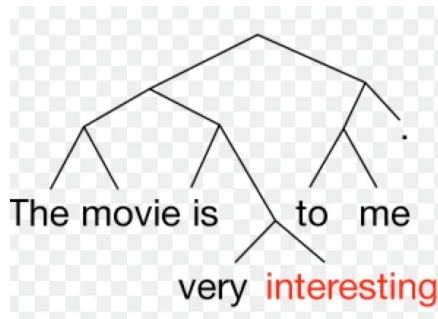
Stride Size

Stride size: how much you want to shift your filter at each step

- It is usually set to be one
- If large stride size, build a model that behaves somewhat similarly to a tree



stride size=2



Multiple Filters

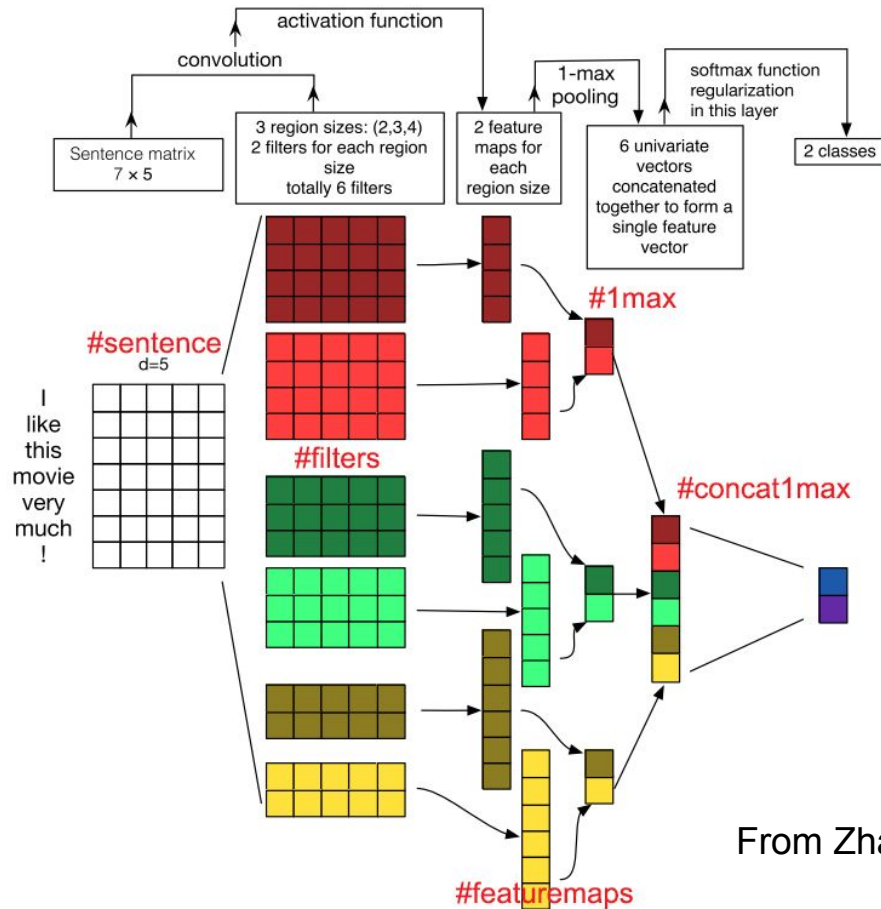
- Use multiple filter weights w (initialize randomly)
- Use different window sizes n
- Then, we can have features for bigrams, tri-grams, 4-grams

Classification after one CNN layer

1. First one convolution, followed by one max-pooling
2. Obtain final features vectors: $z = [c_1, \dots, c_m]$ where m is the number of filters
3. Apply softmax layers on final vector

$$y = \textit{softmax} \left(W^{(S)} z + b \right)$$

CNN Framework



From Zhang 2015

Multiple Channels

- Like image, CNN is applied on R-G-B channels
- For NLP, different word embeddings can be regarded as different channels

CNN for NLP

1. n-grams features are important (window size)
2. Location of **key** n-grams are trivial (pooling)
3. Stack of Convolutional layer or large window size can also capture long-range information