Sequence models for NLP

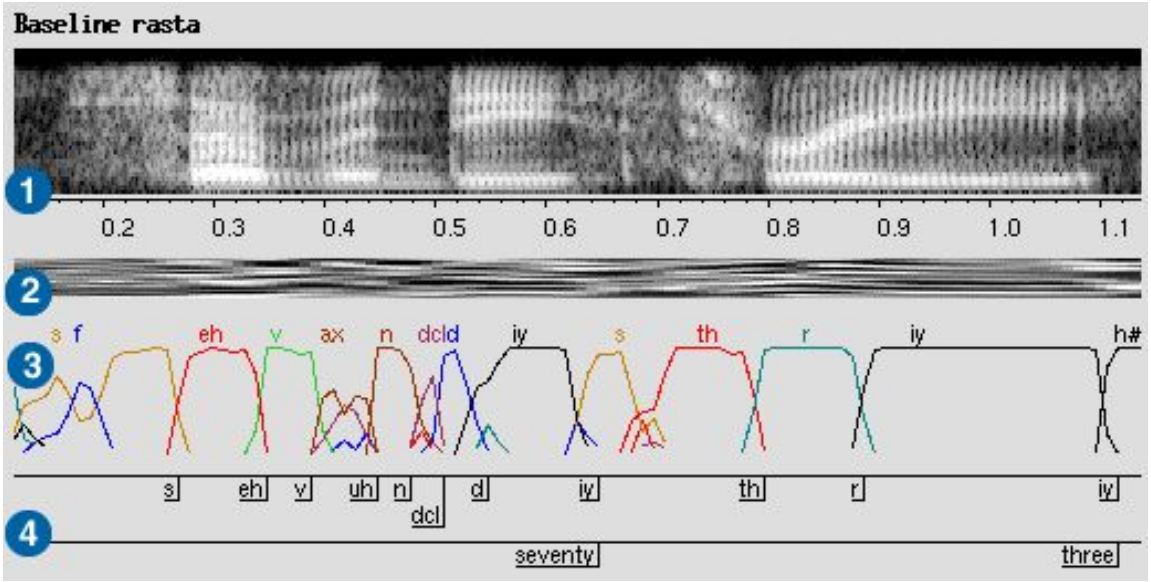
Dr. Ahmad El Sallab
AI Senior Expert

Agenda

- Why sequence models?
- Statistical Language Models
- Recurrent NN
- Neural Language Models
- RNN for text classification
- What’s wrong with RNN?
- Other sequence models: CNN1D
- CNN-LSTM models

Why Sequence models?

Speech

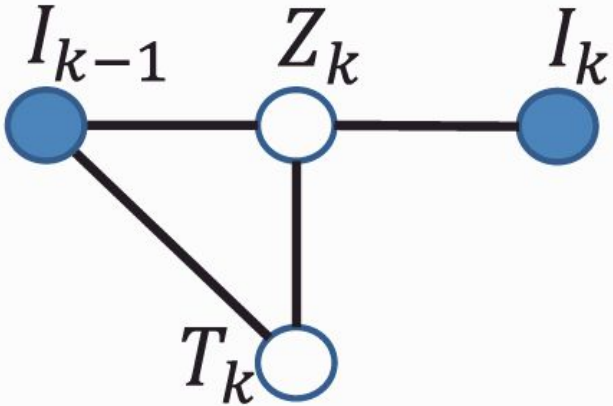

Baseline rasta
1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
2
3
s f
eh
v
ax
n
dcld
iy
s
th
r
iy
h#
s
eh
v
uh
n
dcl
d
iy
th
r
iy
4
seventy
three

Text

the clouds are in the

Ground
Garage
Sky
Airport
Oven

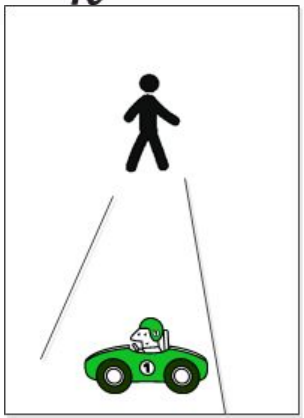
Image/Video

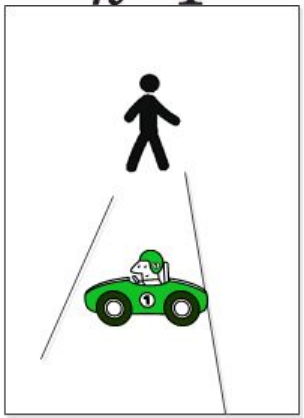


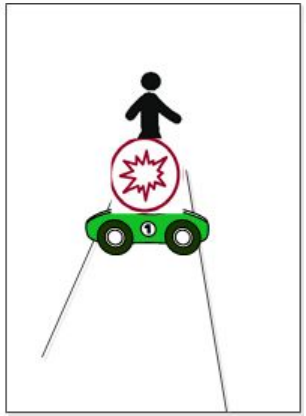
Frame I_k

Frame I_{k-1}

Frame I_k







Sequences are everywhere!

Context is all what matters!

Statistical Language Models (SLM)

Language Modeling

- **Language Modeling** is the task of predicting what word comes next.

- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$

- A system that does this is called a **Language Model**.

Language Modeling

- You can also think of a Language Model as a system that assigns probability to a piece of text.

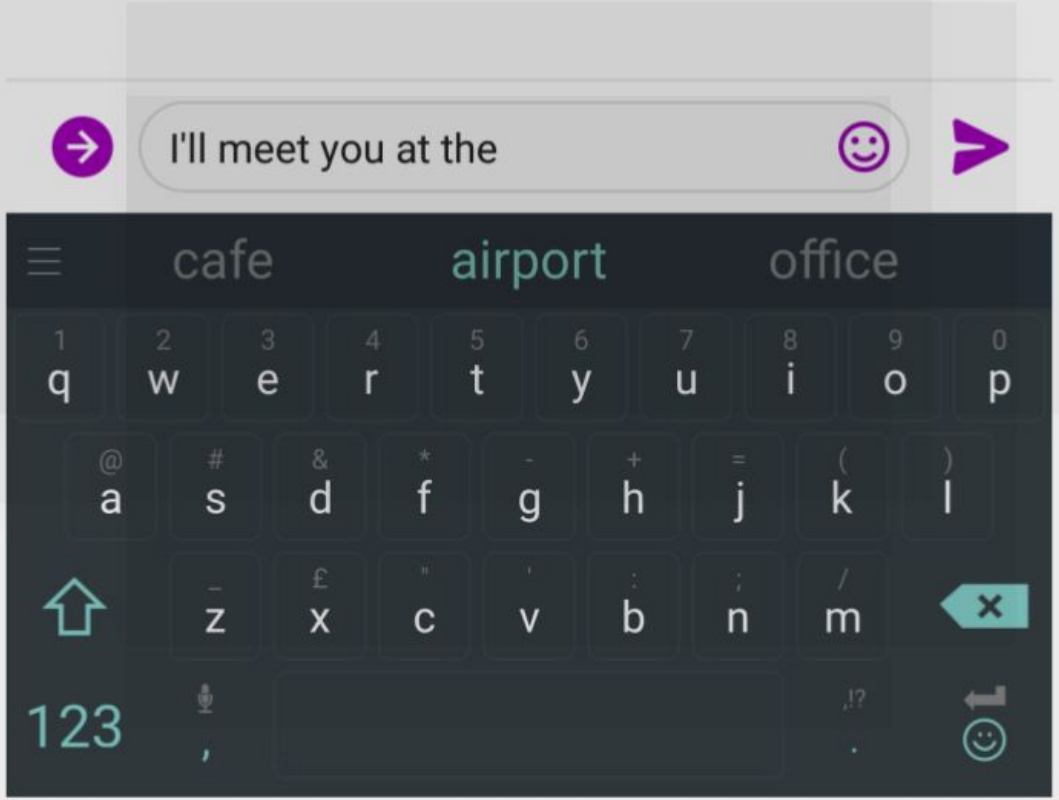
- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) = P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)})$$

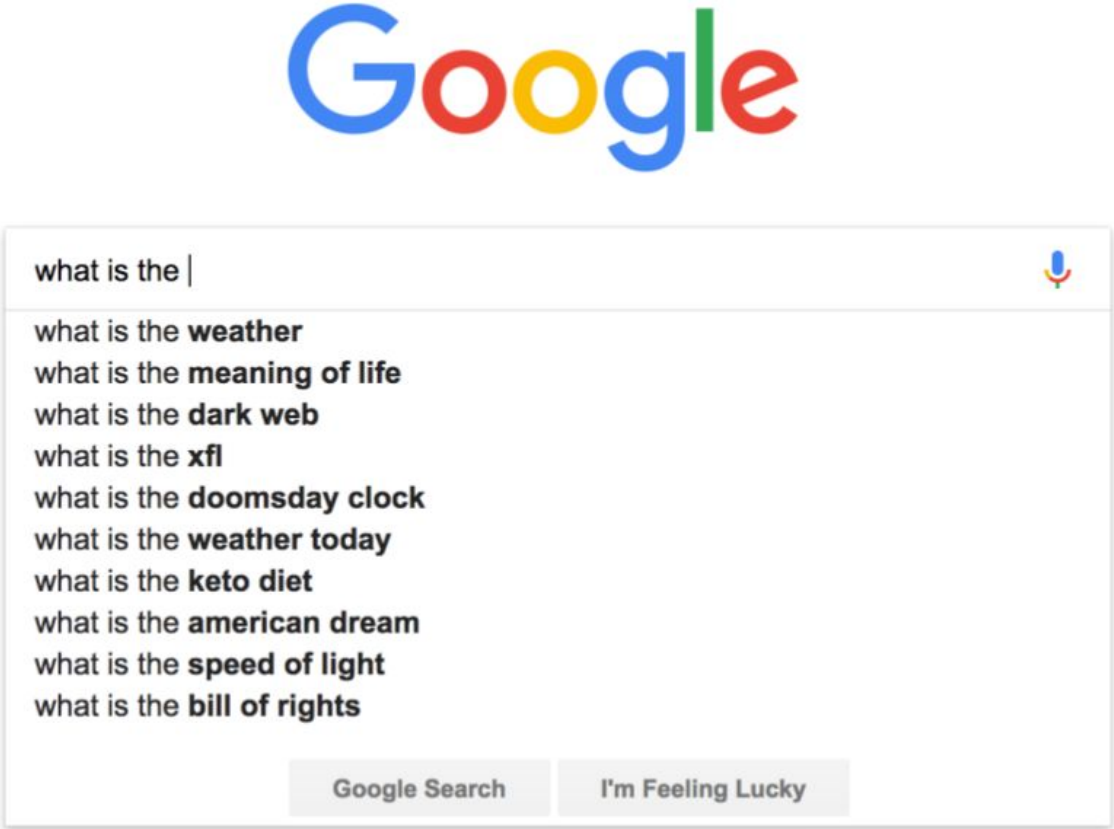
$$= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

This is what our LM provides

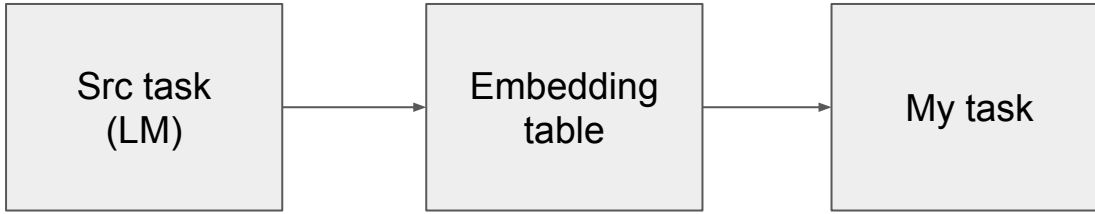
You use Language Models every day!



You use Language Models every day!



LM to train embeddings



Language Modeling

the clouds are in the

Ground
Garage
Sky
Airport
Oven

Sequential by nature!

n-gram Language Models

the students opened their _____

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn a *n*-gram Language Model!

- Definition: A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: "the", "students", "opened", "their"
  - bigrams: "the students", "students opened", "opened their"
  - trigrams: "the students opened", "students opened their"
  - 4-grams: "the students opened their"

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.

n-gram Language Models

- First we make a simplifying assumption: $\boldsymbol{x}^{(t+1)}$ depends only on the preceding *n-1* words.

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

(assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

(definition of conditional prob)

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

(statistical approximation)

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their* _____

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Sparsity Problems with n-gram Language Models

Sparsity Problem 1

Problem: What if *"students opened their* $\boldsymbol{w}$*"* never occurred in data? Then $\boldsymbol{w}$ has probability 0!

(Partial) Solution: Add small δ to the count for every $\boldsymbol{w} \in V$. This is called *smoothing*.

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

Problem: What if *"students opened their"* never occurred in data? Then we can't calculate probability for *any* $\boldsymbol{w}$!

(Partial) Solution: Just condition on *"opened their"* instead. This is called *backoff*.

Note: Increasing *n* makes sparsity problems *worse*. Typically we can't have *n* bigger than 5.

Storage Problems with n-gram Language Models

Storage: Need to store count for all *n*-grams you saw in the corpus.

As the sequence length increases, it will be more "unique" leading to more entries in the n-gram Counter table.
If n is small, then it's more "repeated", so less unique entries

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

Increasing *n* or increasing corpus increases model size!

n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop*

Business and financial news

today the _____

get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
…	

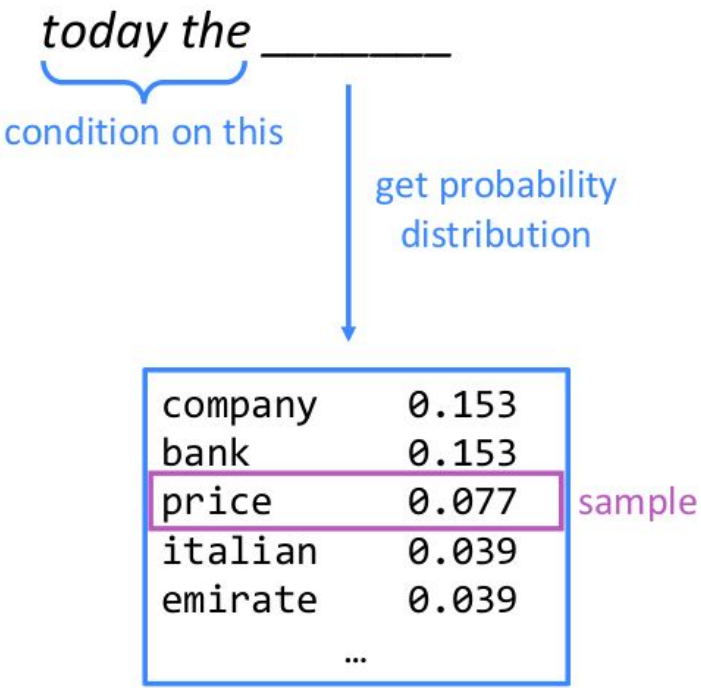
Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

* **Try for yourself:** https://nlpforhackers.io/language-models/

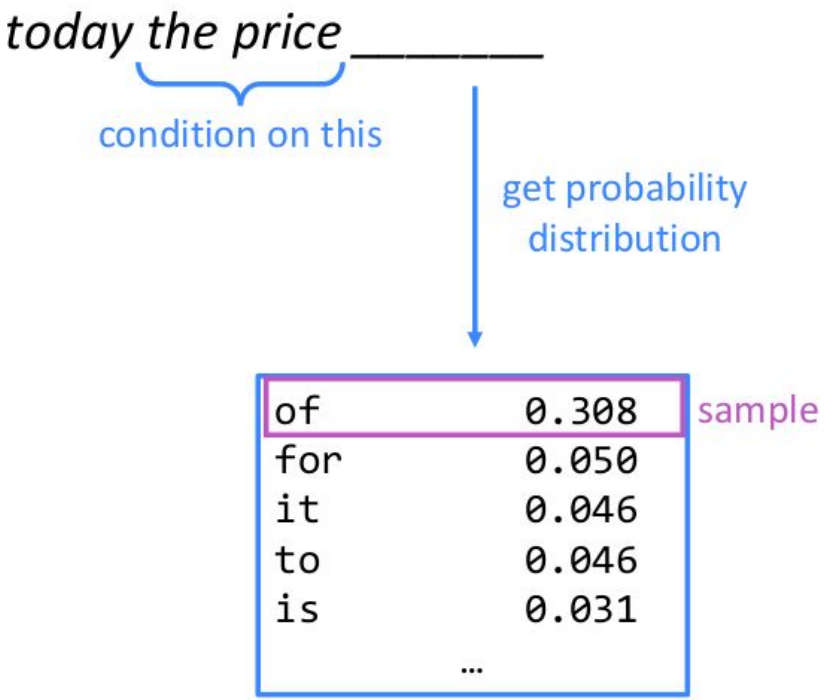
Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.



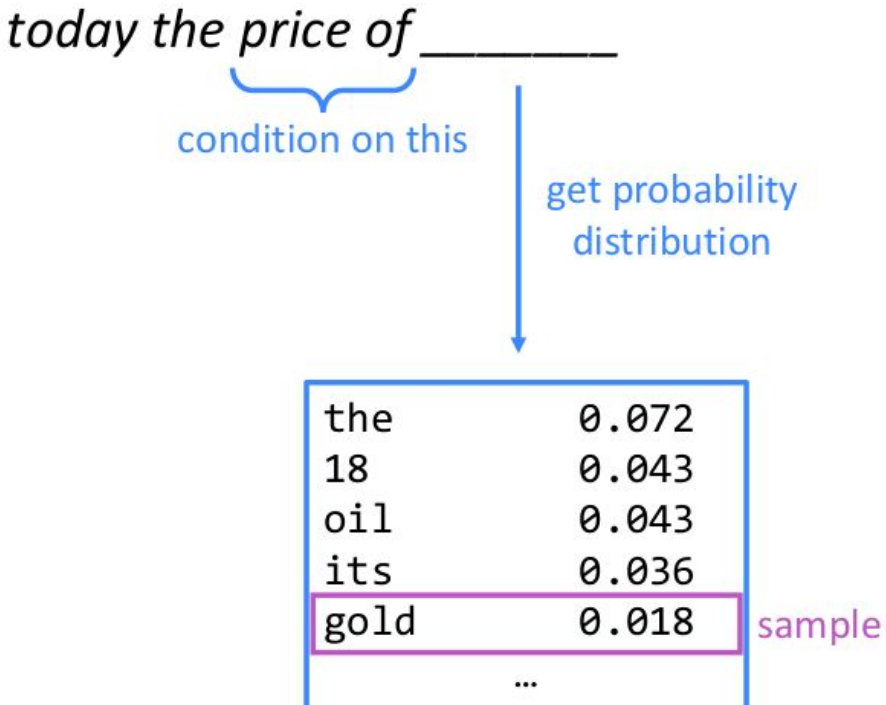
Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.



Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.



Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

But increasing *n* worsens sparsity problem, and increases model size…

Let’s code

N-gram SLM

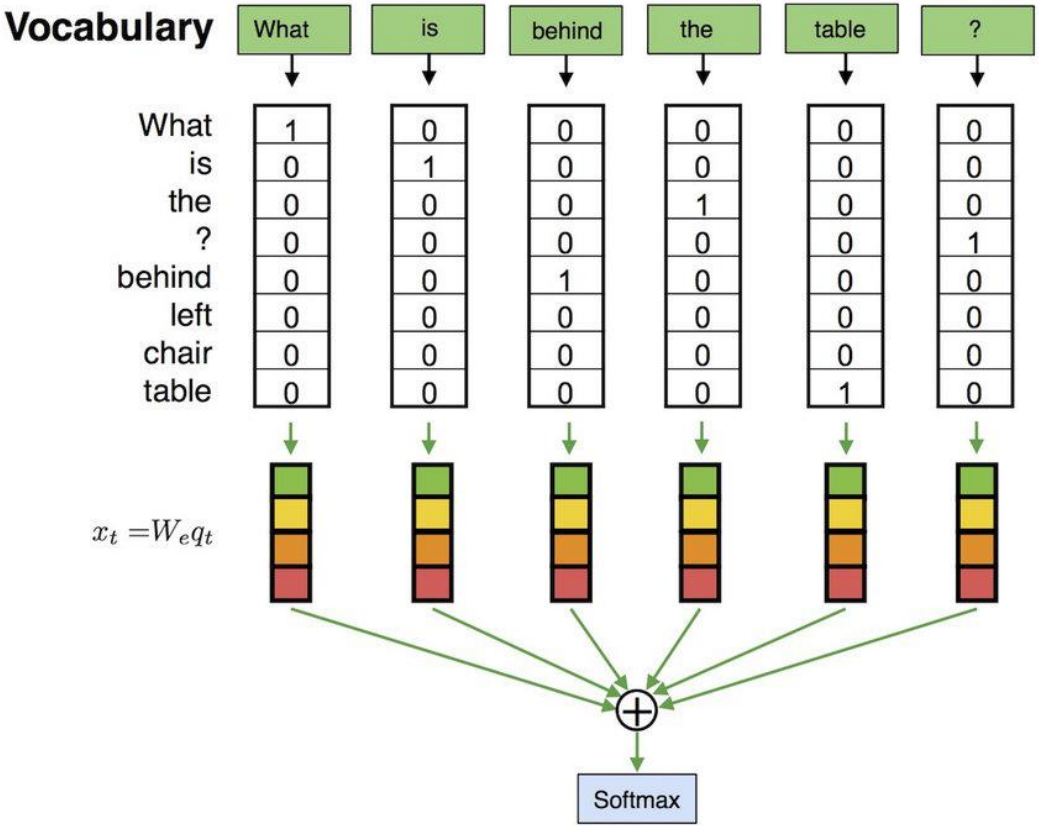
https://colab.research.google.com/drive/1cq-FKtD8tJpaoEODzZM6dm6hWaLUEyGk?usp=sharing

Neural Language Models (NLM)

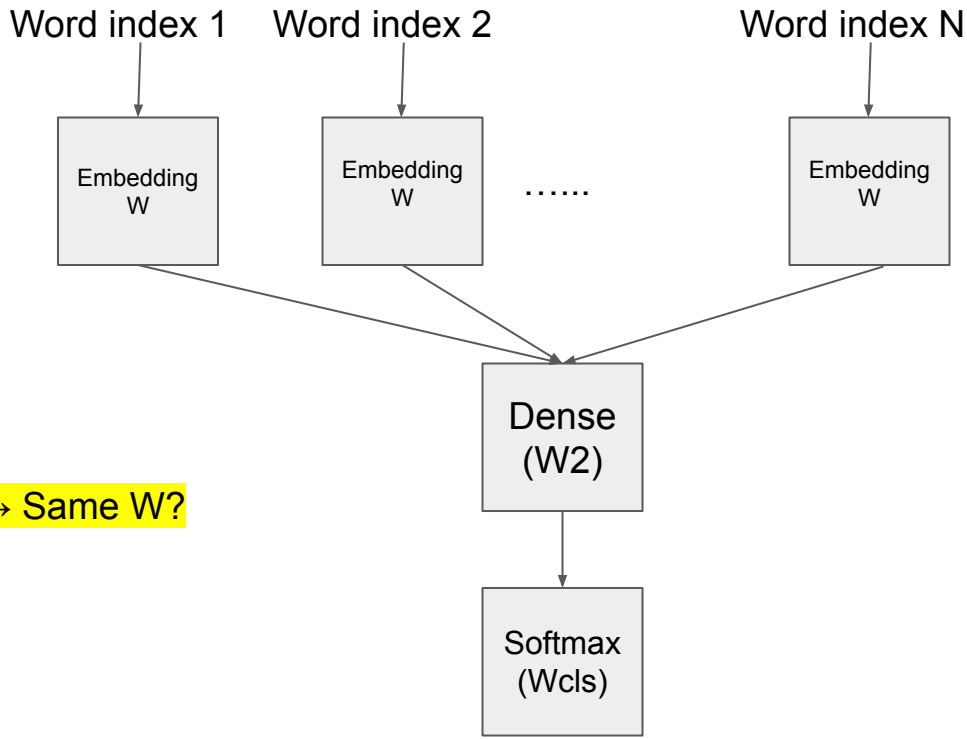
NLM with BoW vectors

How to represent words to NN?

Bag-of-Words model

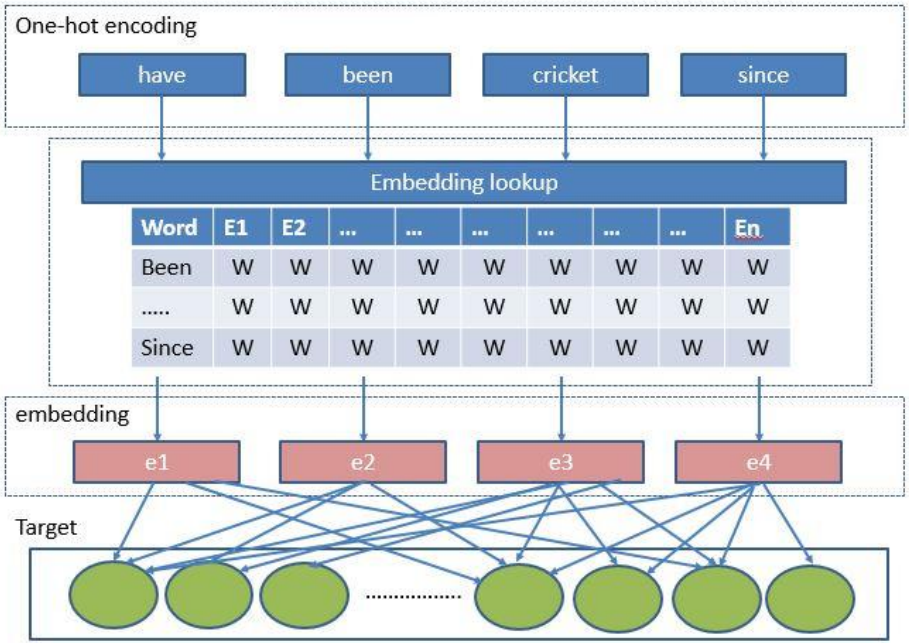


Word Embeddings



ALL Embedding layers → Same W?

BoW vectors model



A fixed-window neural Language Model

A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

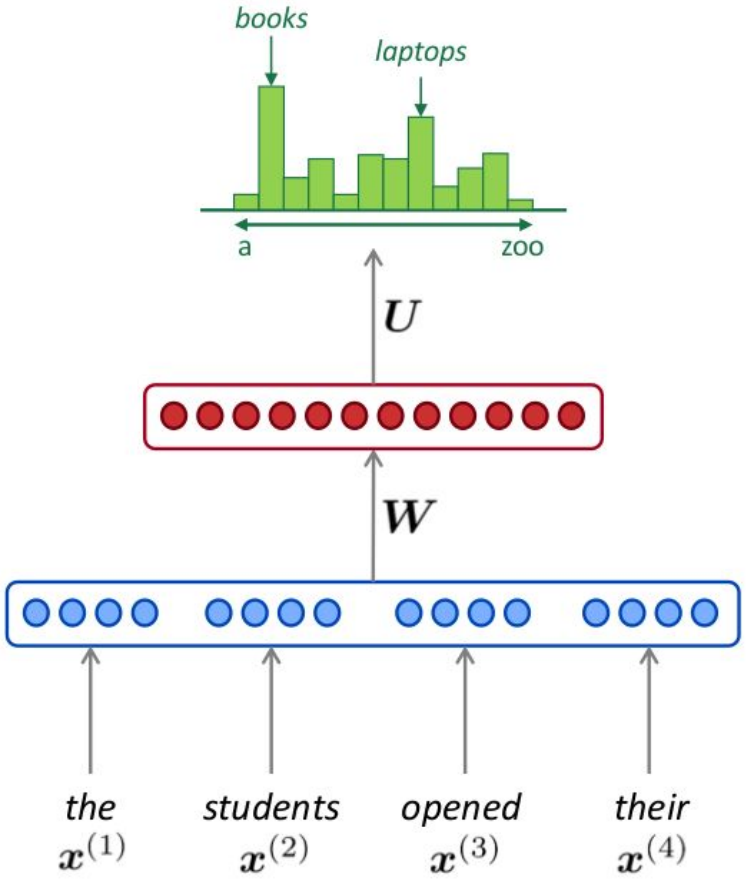
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



A fixed-window neural Language Model

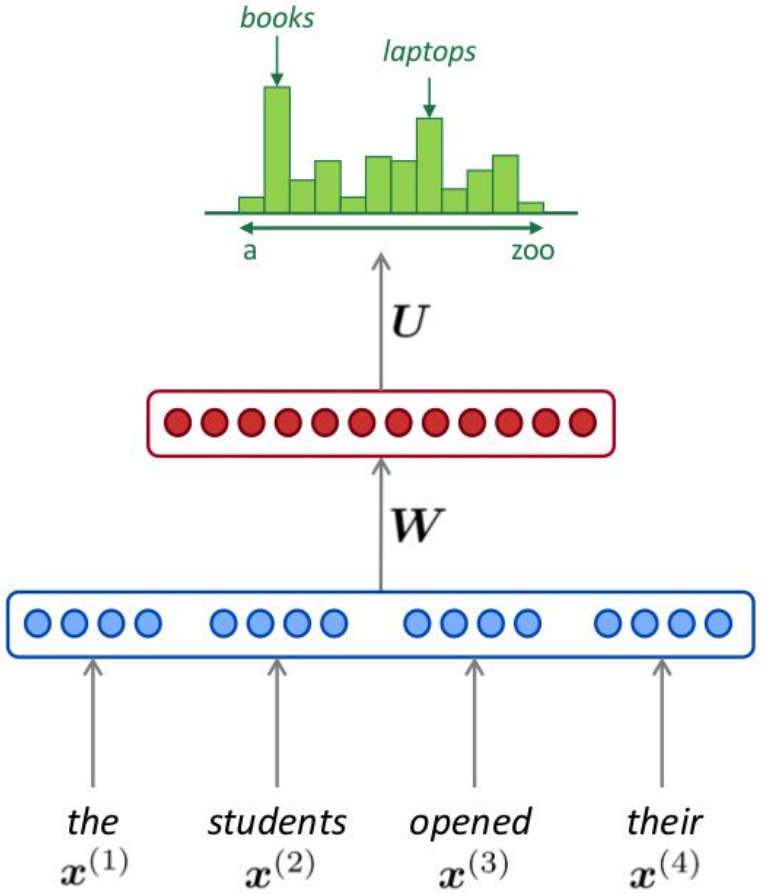
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



What’s wrong with BoW?

Soup of words vectors → Sequence info is lost!

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

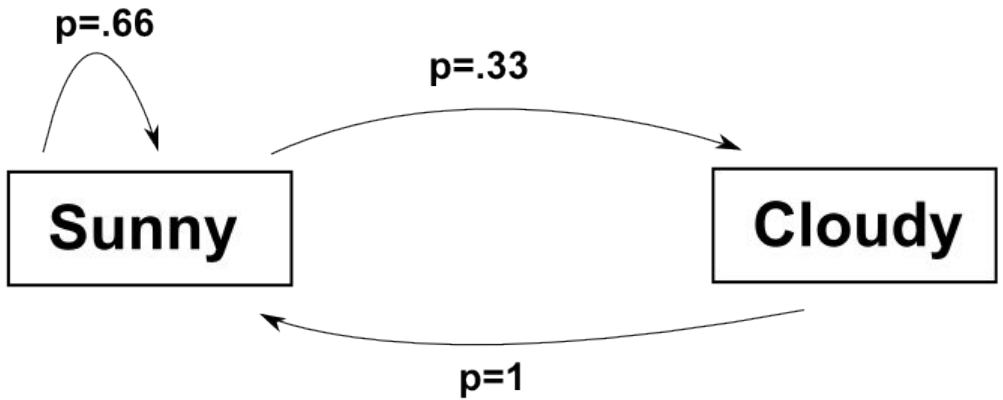
$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

Recurrent Neural Networks

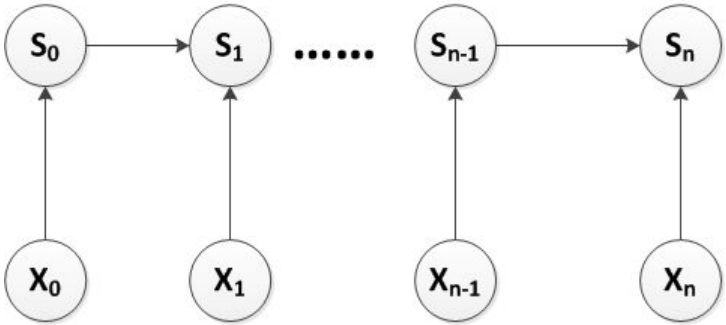
Natural Selection for sequence models

Sequential process prediction



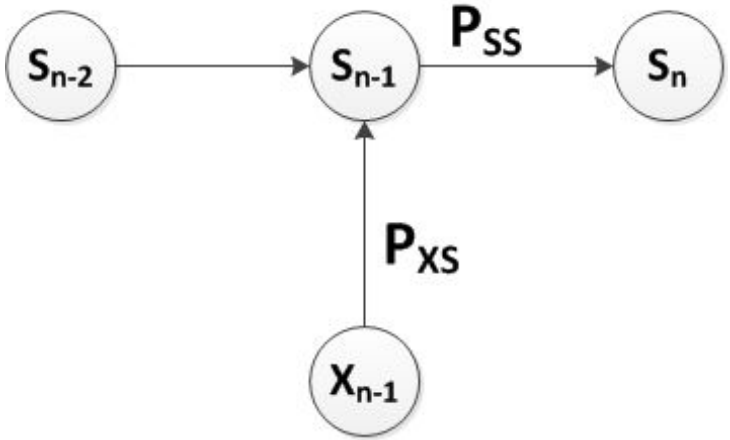
History factor

What if we introduce time?
Do we need to worry about old times?



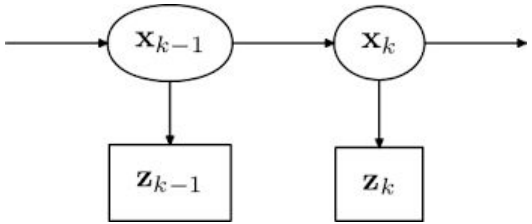
The Markov Assumption and Markov Decision Process (MDP)

- In Chess, you only need to worry about the current state of the game to take the next action



Bayes filters

- True state is hidden
- Only partial states are observable
- “Filter” the noise in the observed measurement to obtain the “de-noised” state



Algorithm Bayes_filter$(bel(x_{t-1}), u_t, z_t)$**:**

for all x_t do

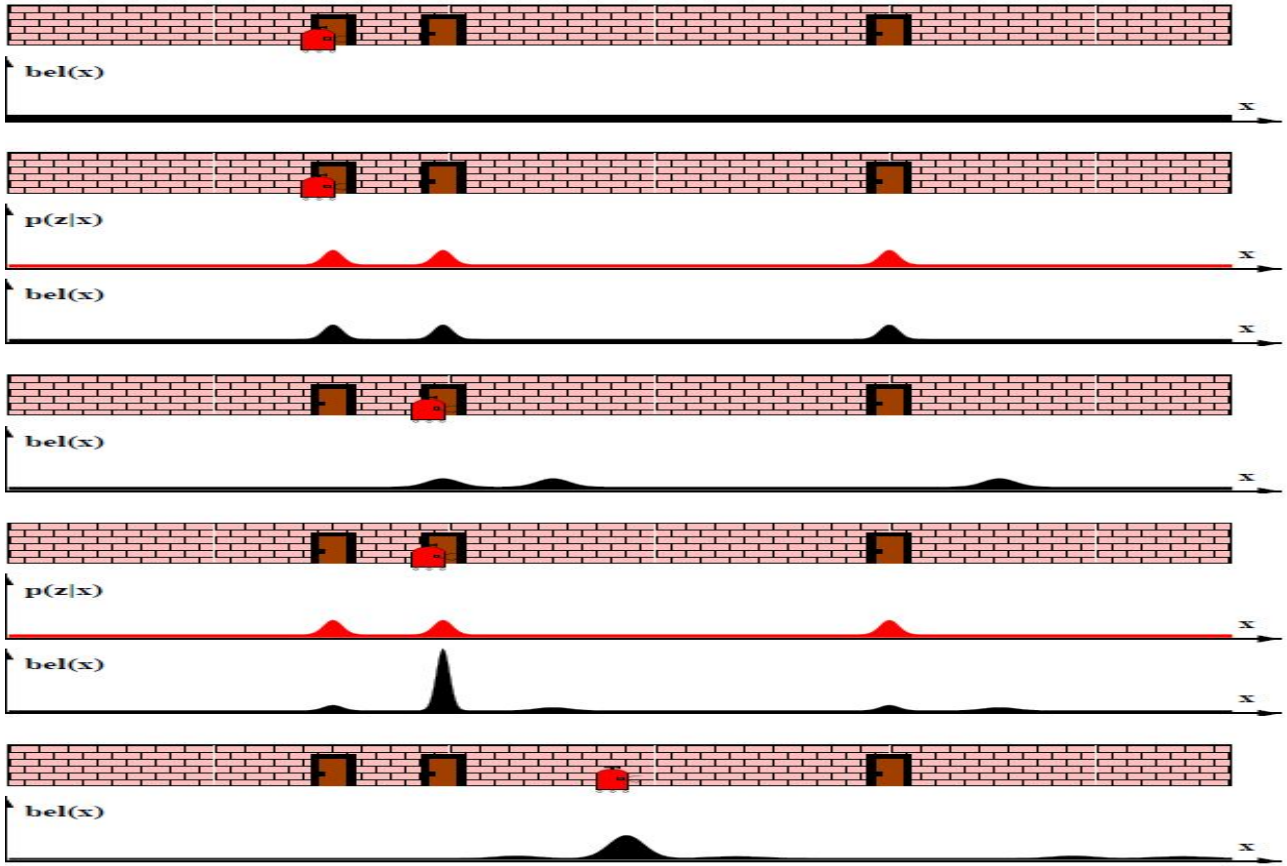
$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx$

$bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$

endfor

return $bel(x_t)$

Robot Localization



Recursive State estimation

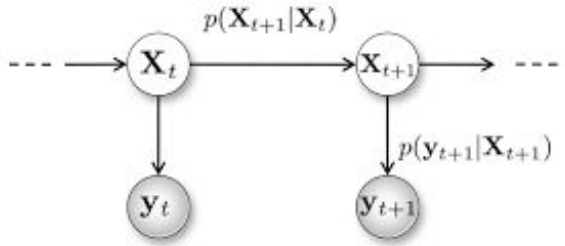
- State space equations in control
  - Continuous states
- Probabilistic State Estimation
  - Bayes
  - Gaussian
  - Kalman=Gaussian + Linear (State space equations + randomness to be probabilistic)

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$
$$\text{where}$$
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix}$$

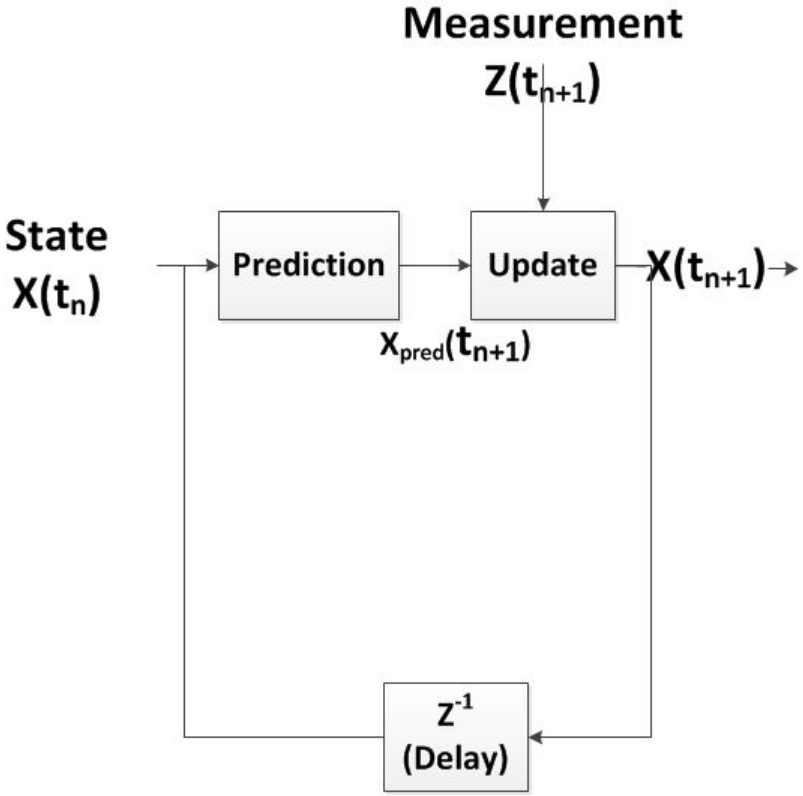
$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{w}$$
$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}$$
$$\tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{v}$$

Hidden Markov Model

- Well known in speech
- Discrete state

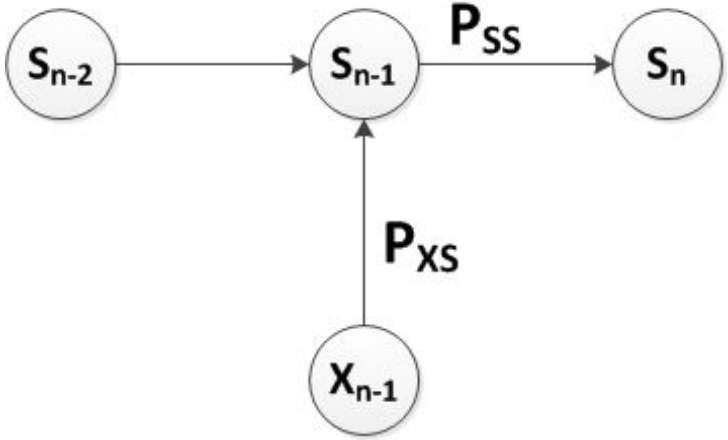


Kalman filter



Exploration vs. Exploitation

- All forms depend on two terms
  - Exploitation: use of known information = history = old state term = forget gate
  - Exploitation: use of new information = inclusion of input = input gate
- All comes from the MDP formulation



Is Kalman/Bayes Filter Enough?

- How to set the motion model?
- How to set the measurement/sensor model?
- It has to be learnt!
- But computationally effective so far
- Bad in remembering history
  - Due to MDP assumption

Emission and Transition pdf's are "assumptions"

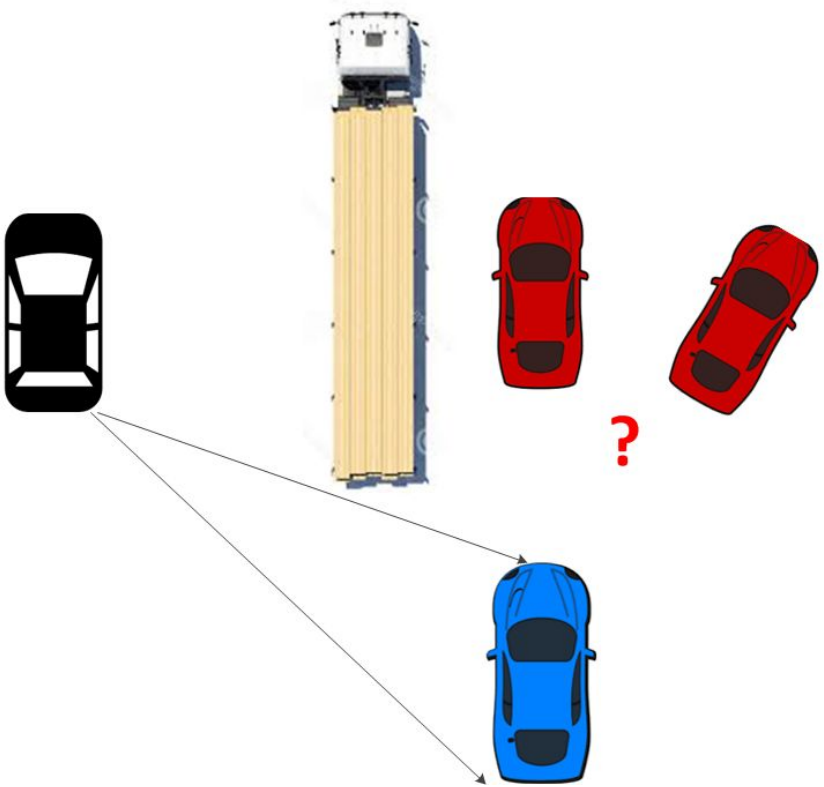
Partially observable MDP (POMDP)

- In Chess, you only need to worry about the current state of the game to take the next action

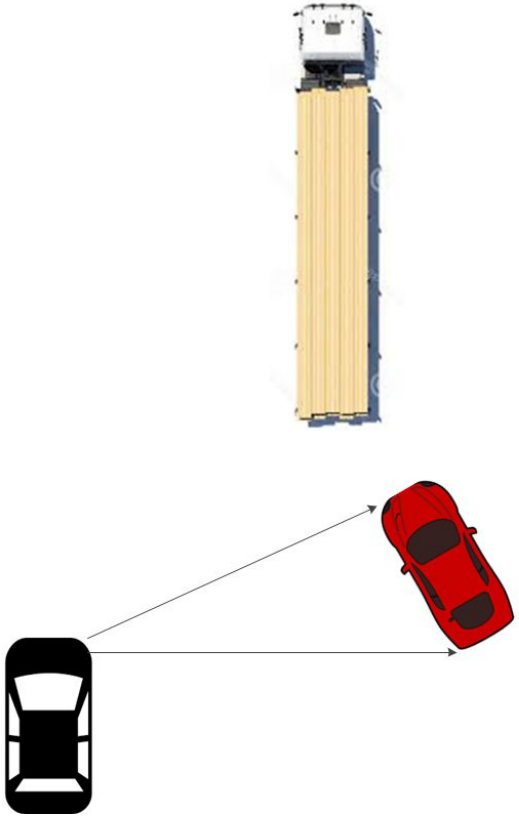
- In Card games (Estimation, Poker,…etc), the current state of the game is not sufficient

  - You need to be aware what were the previous cards played to take the correct actions (or you estimate ;)

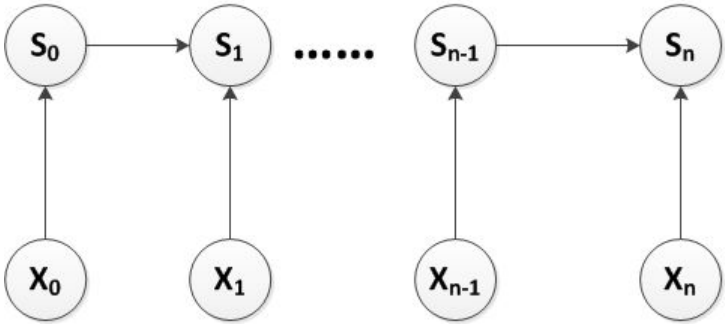
Occlusion in Automotives



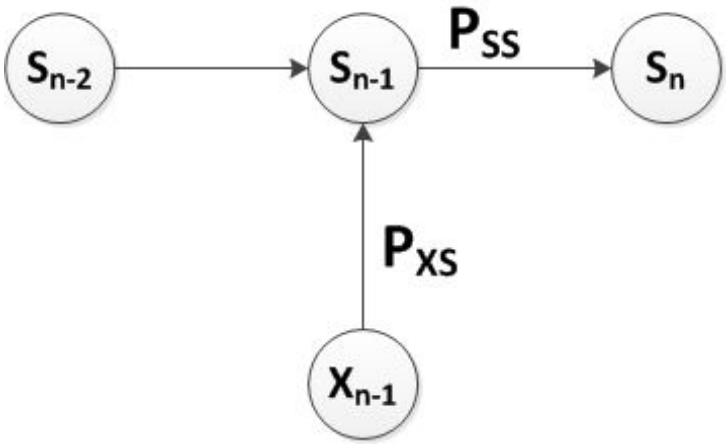
Occlusion in Automotives



Is Kalman/Bayes Filter Enough?



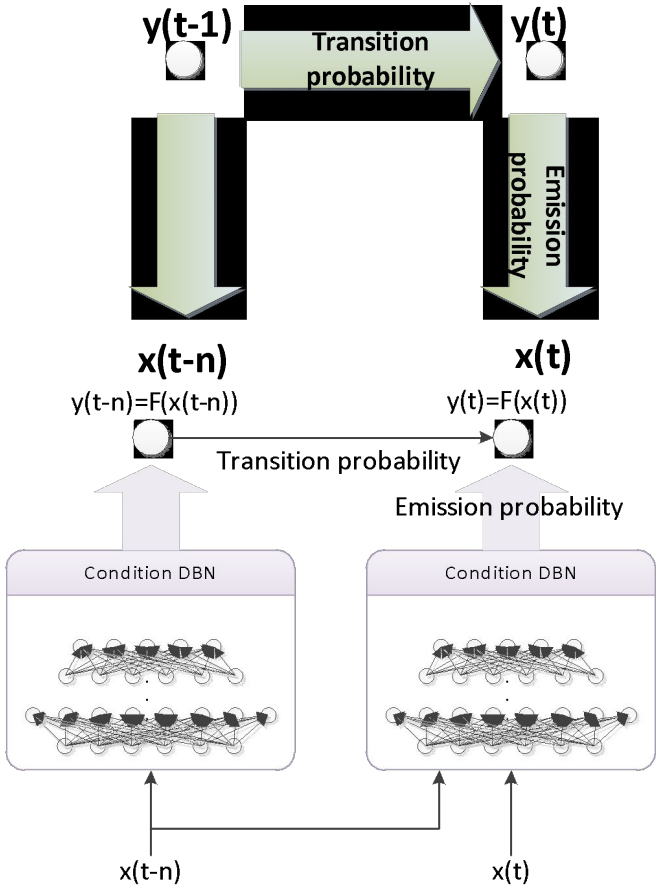
MDP is an assumption
Dependency could be on longer history



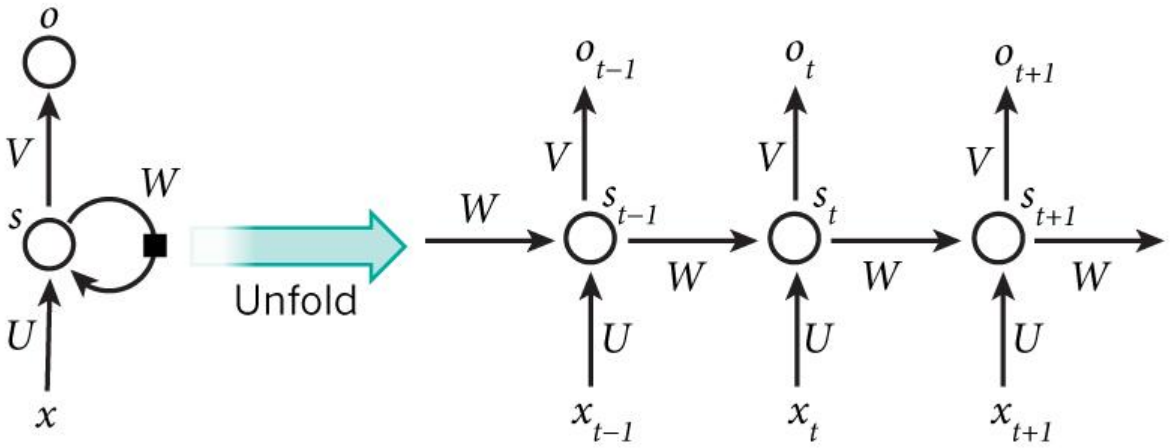
Emission and Transition pdf's are "assumptions"

Sequence modeling in DL

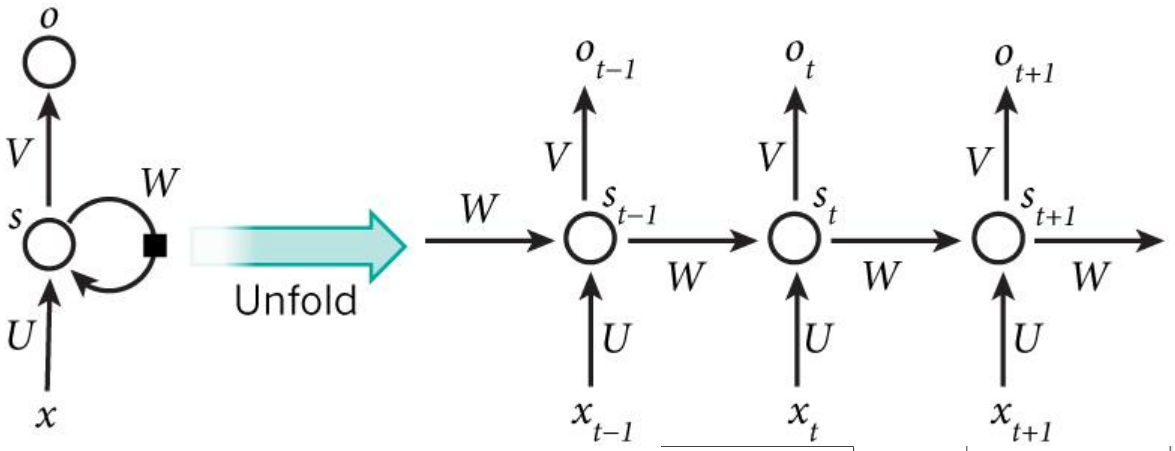
- Generative
  - HMM
  - Recursive auto encoders
- Discriminative
  - CRF
  - Recurrent neural nets
- Used in context modeling

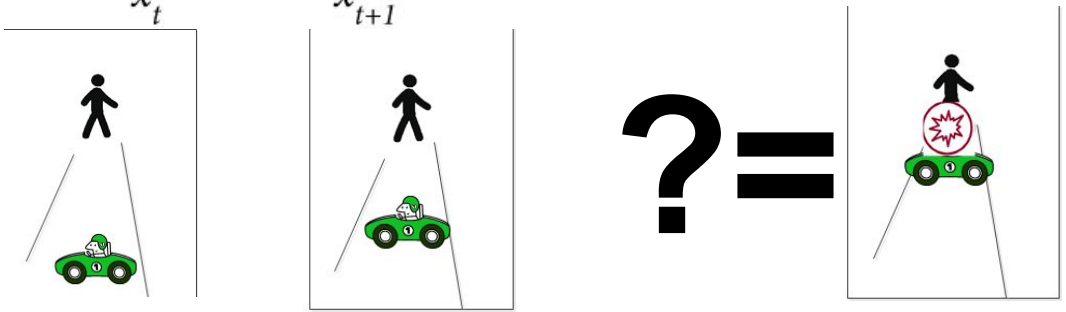


Recurrent Neural Nets

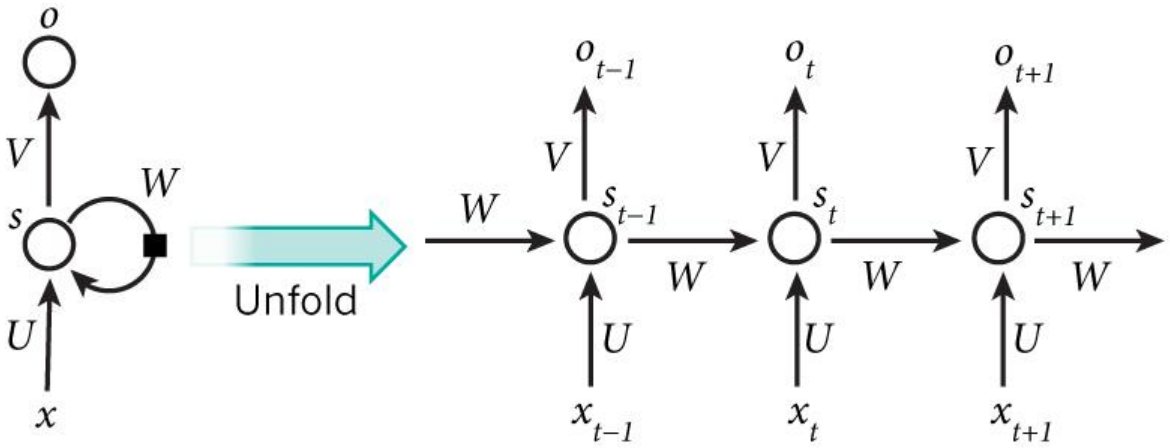


Recurrent Neural Nets





Recurrent Neural Nets



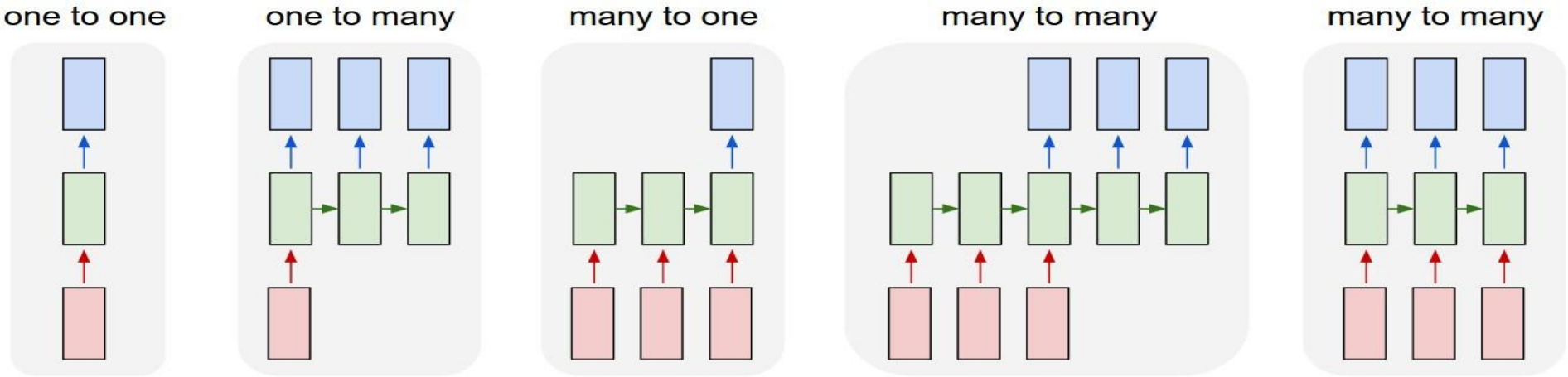
What makes RNN special vs. DNN/CNN?

- A glaring limitation of Vanilla Neural Networks (and also Convolutional Networks) is that their API is too constrained: they accept a **fixed-sized vector** as **input** (e.g. an image) and produce a **fixed-sized vector** as **output** (e.g. probabilities of different classes).

- Not only that: These models perform this mapping using a **fixed amount of computational steps** (e.g. the number of layers in the model).

- The core reason that recurrent nets are more exciting is that they allow us to operate over ***sequences* of vectors: Sequences in the input, the output, or in the most general case both.**

What makes RNN special vs. DNN/CNN?



What makes RNN special vs. DNN/CNN?

- One-One: Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).
- One-many: Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
- Many-one: Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).
- Many-Many: Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
  - Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like

DNN=Function approximation
RNN=Program approximation

If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs.

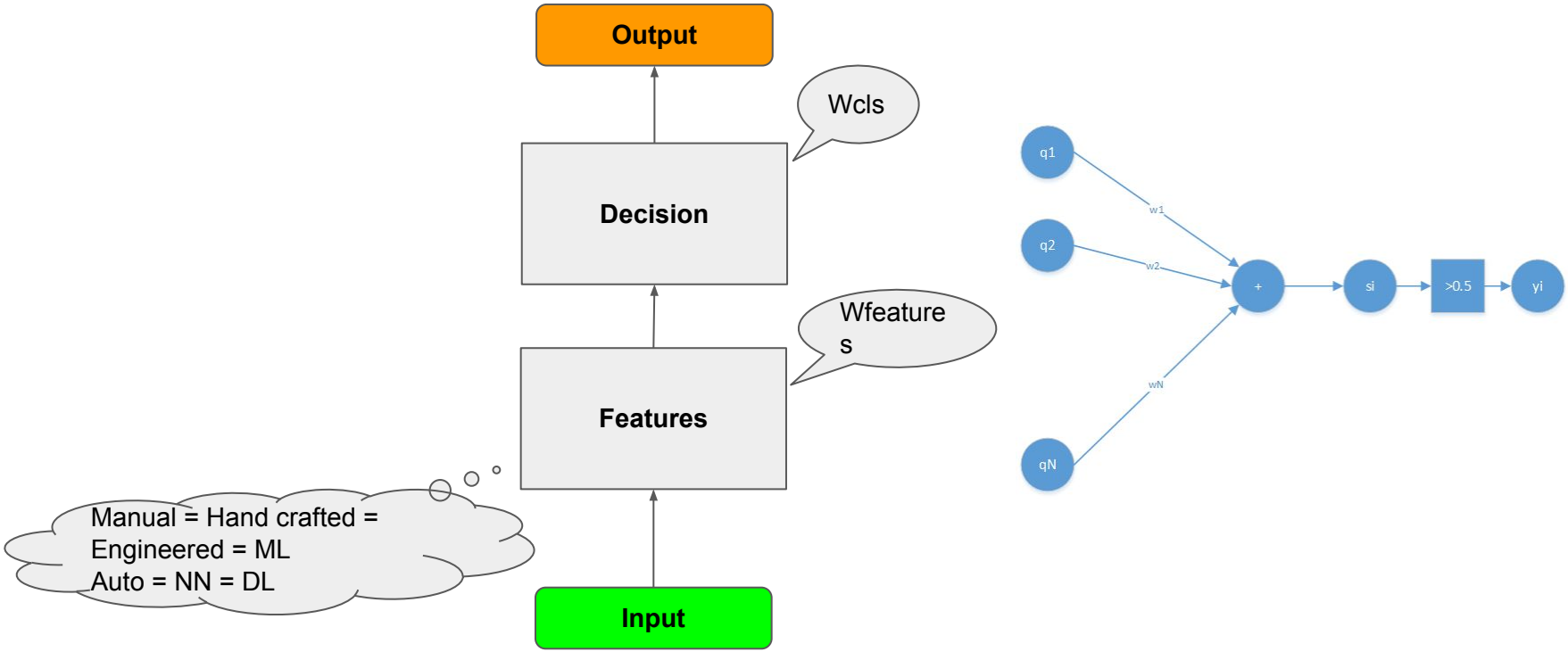
What is a program anyway?
A sequence of operations
Performed in order
Result of one affecting the next
==Recurrence!

Problems the RNN solves = The Oracle

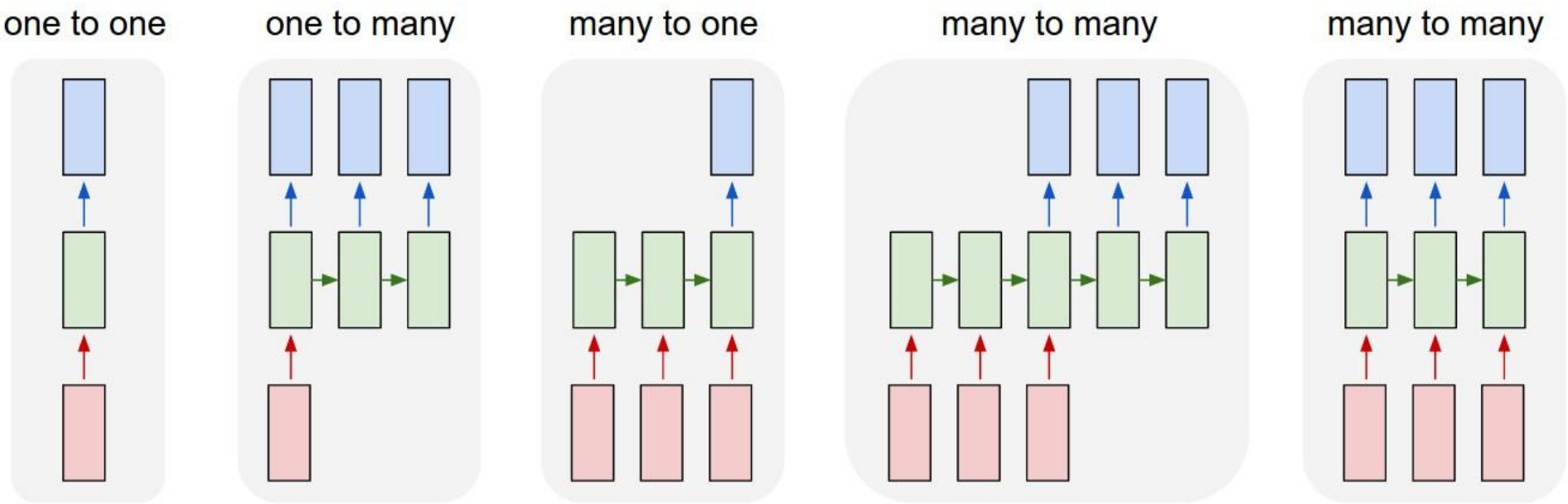
- Classification problems
  - DNN = Associative Mapping of Input/Output
  - =Human recognition (develops in youngest age)□ Easiest
  - In automotive = Environment awareness and object detection
  - As hard as a human understands the traffic system
- Prediction problems
  - RNN = Information aggregation/integration overtime to predict next state
  - =Human anticipation (develops in older age) □ Harder
  - In automotive = Mapping and Environment state estimation and object tracking
- Planning problems
  - RL = Evaluate the utility of state-action pair and take action that maximizes the expected total future payoff (reward) from source to goal
  - =Human planning (develops with maturity, requires too much experience)□ Hardest

Sentence Embedding with Sequence models

Supervised Learning Model Design Pattern



Encoder-Decoder pattern



What is the best representation of language?

This question summarizes all NLP efforts!

For what?

- Classification
  - Many-to-one: Seq2Class
  - Sentiment analysis, Toxicity detection (JIGSAW), Real or not? Disaster tweets
- Dialogue:
  - Many-to-many: Seq2Seq → Unaligned case → More on that later
  - MT, Spelling correction, Speech, OCR,...etc

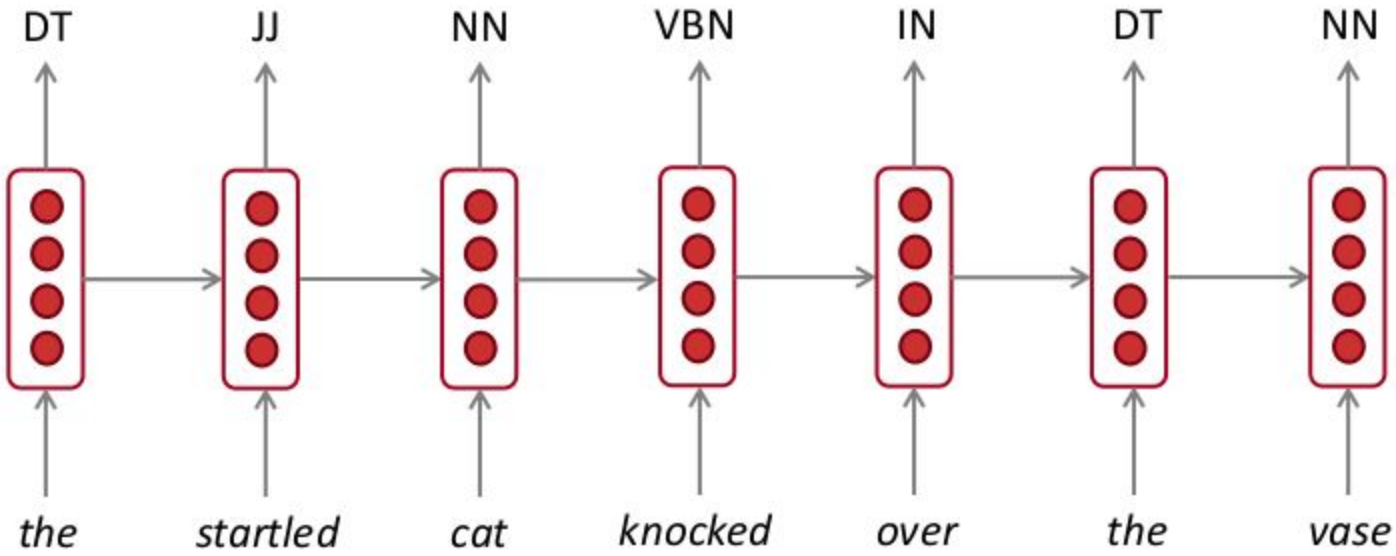
NLP models meta-architectures

Just like in CV: **Encoder-Decoder**

- **Seq2Class:**
  - Encoder = words vectors aggregation **(How?)**
  - Decoder = None (just classifer=softmax)
  - Analogy to CV: Encoder-Softmax (AlexNet, VGG,...etc)
- **Seq2Seq:**
  - Encoder = words vectors aggregation **(How?)**
  - Decoder = multiple words generation **(How?)**
  - Analogy to CV: Encoder-Decoder in semantic segmentation. But in SS, we have aligned many2many, while in NLP, we have unaligned sequences→ challenge in annotation, model, when to stop, position encoding...etc
- **Word vectors are the input to all the above meta-architectures:**
  - Unlike in CV, where pixels are already digitized
  - Also, in NLP **order matters! = Context**

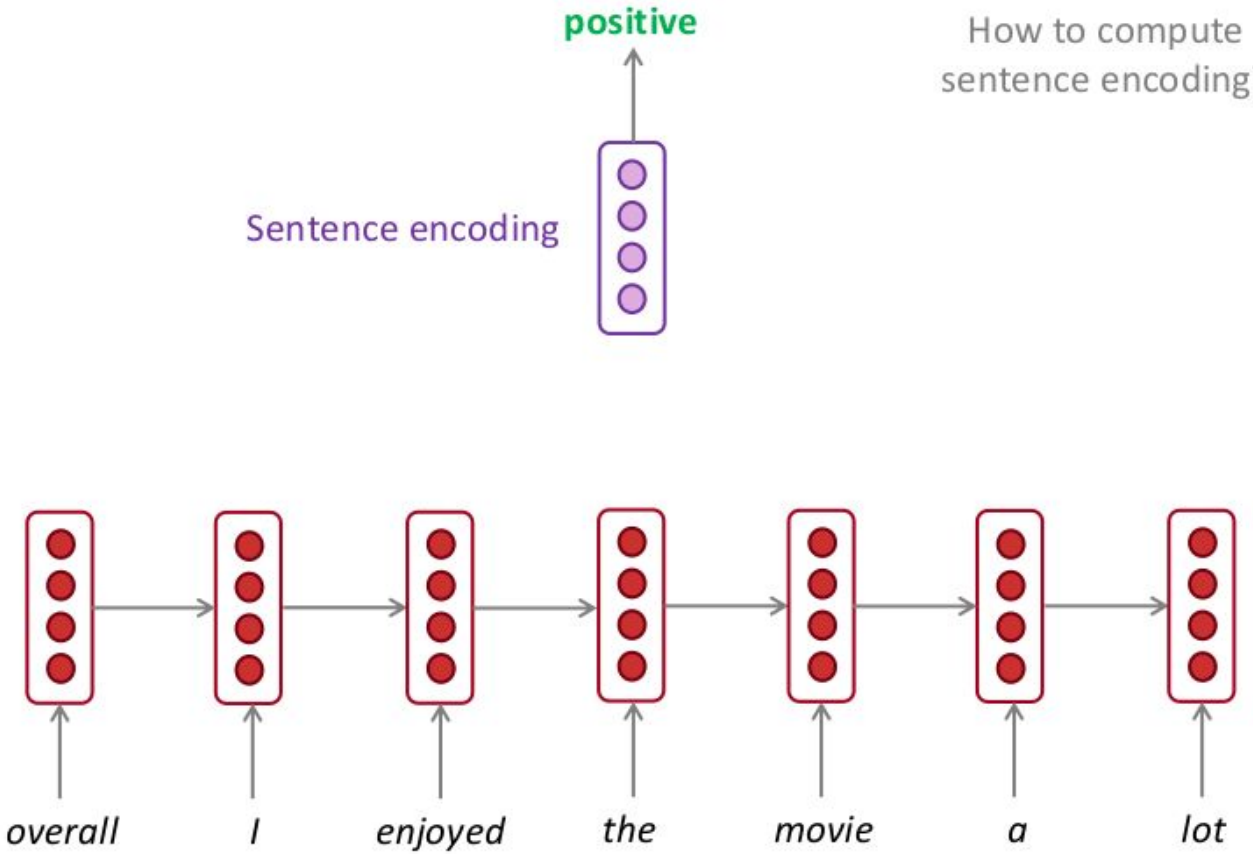
RNNs can be used for tagging

e.g. part-of-speech tagging, named entity recognition



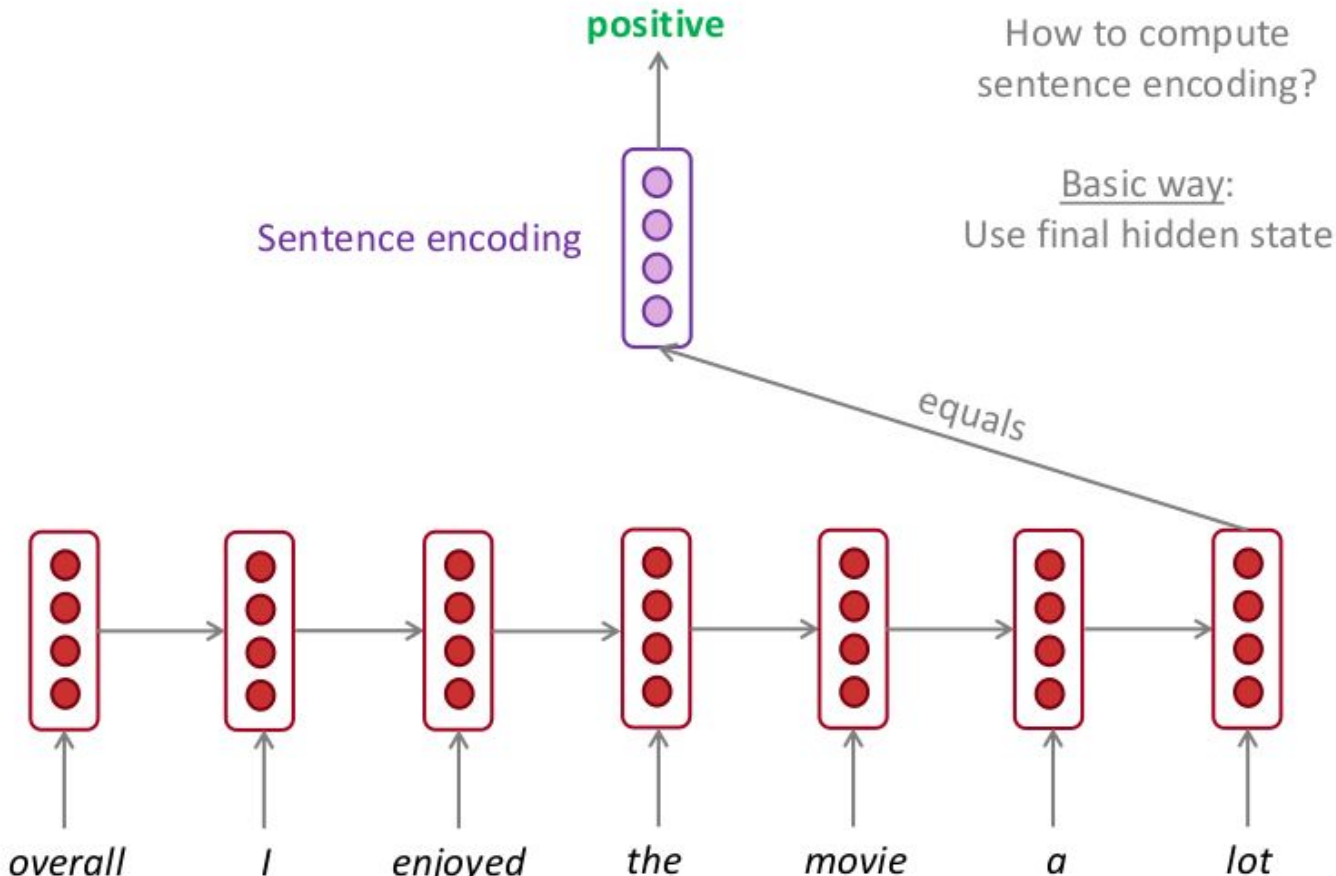
RNNs can be used for sentence classification

e.g. sentiment classification



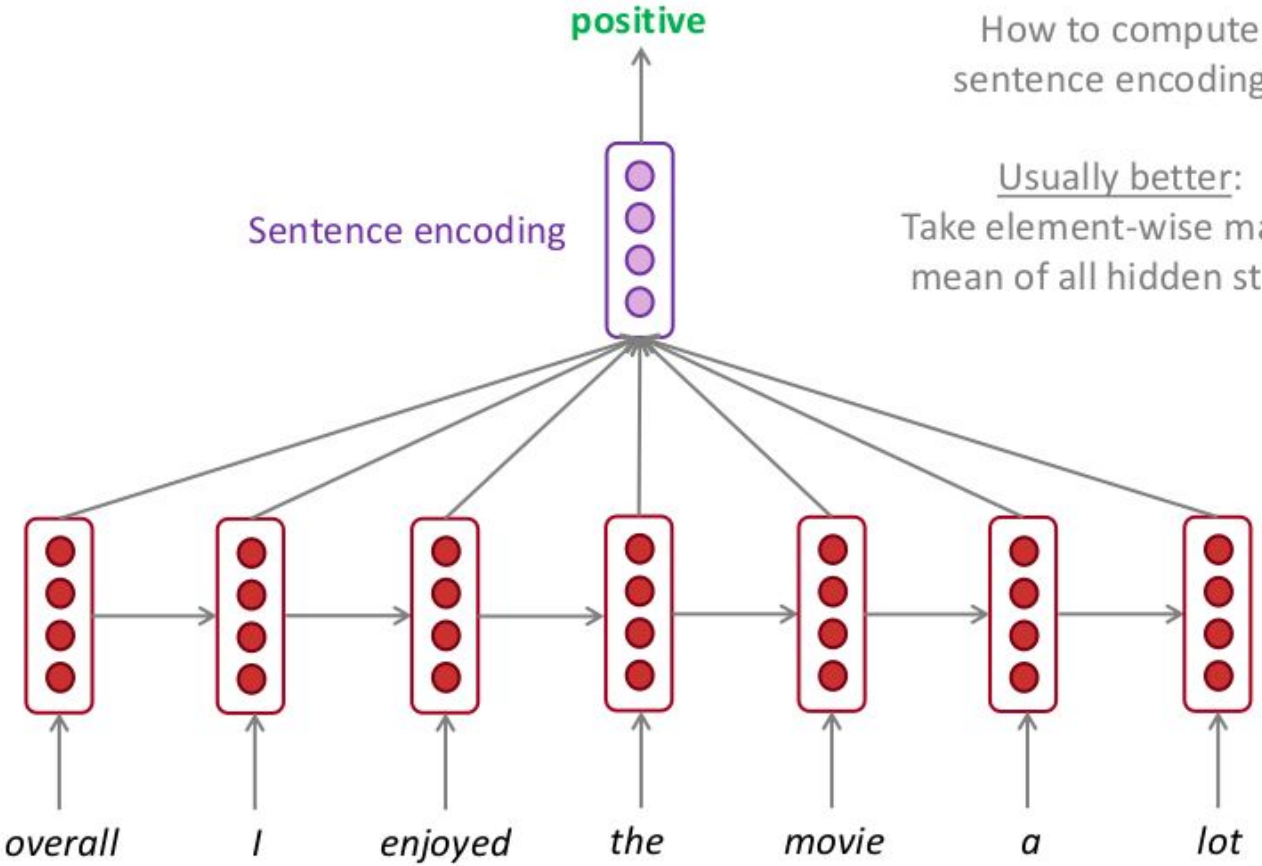
RNNs can be used for sentence classification

e.g. sentiment classification



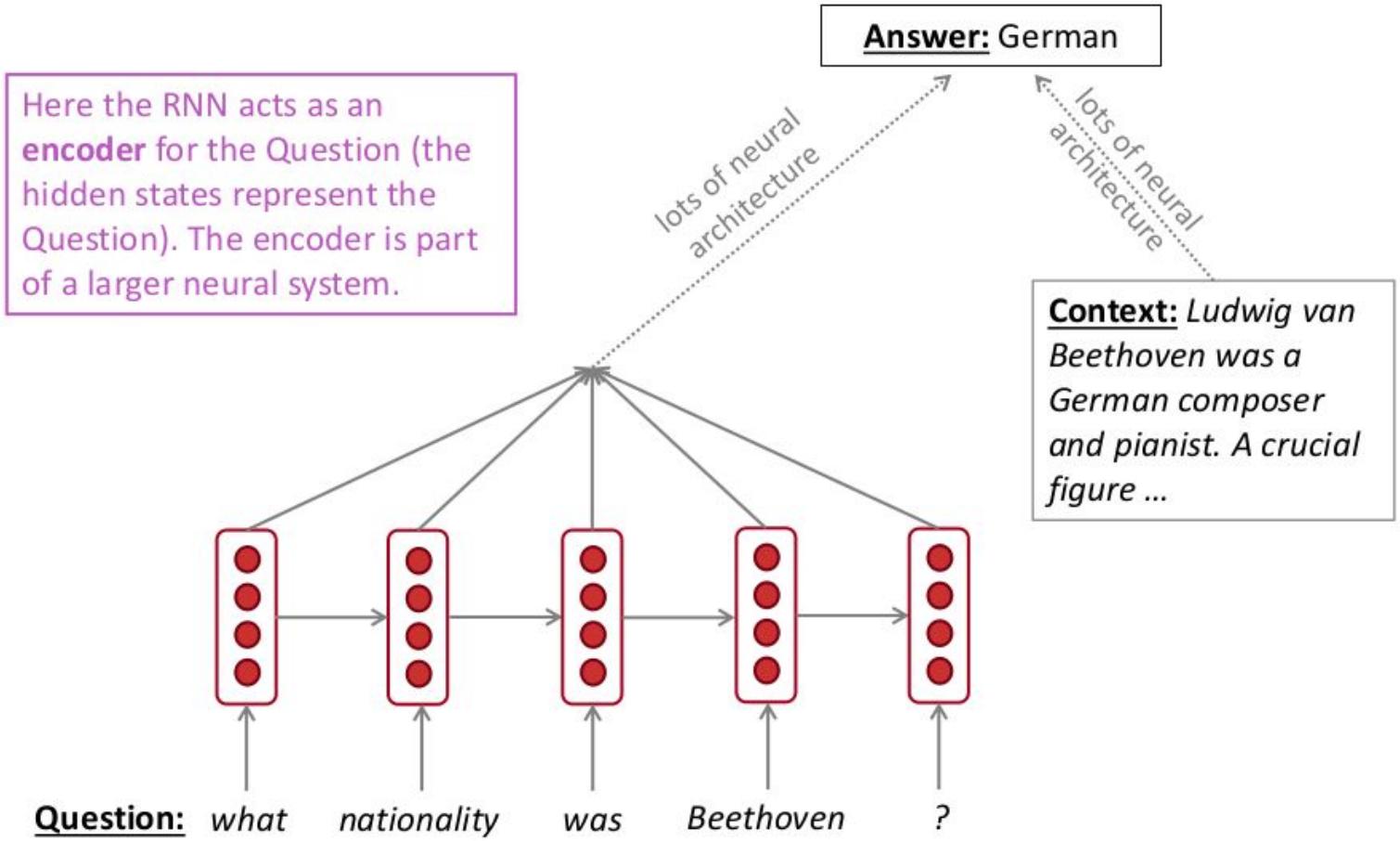
RNNs can be used for sentence classification

e.g. sentiment classification



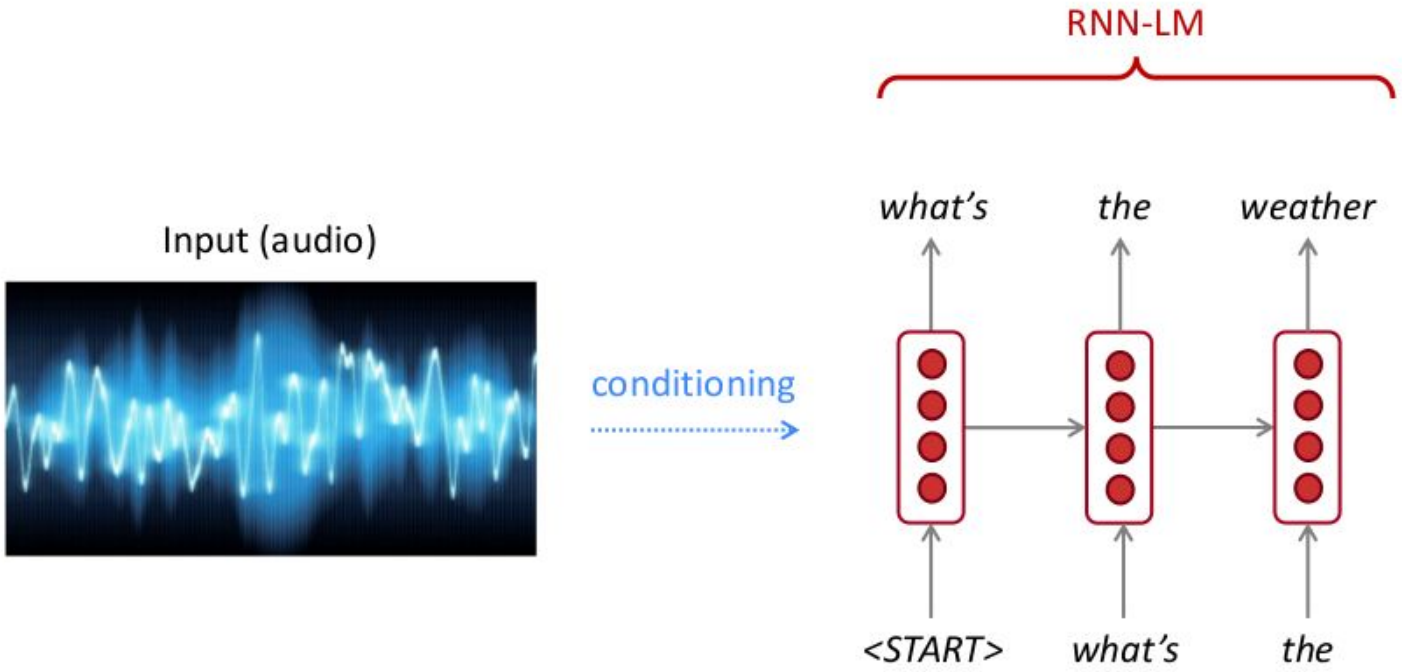
RNNs can be used as an encoder module

e.g. question answering, machine translation, *many other tasks!*



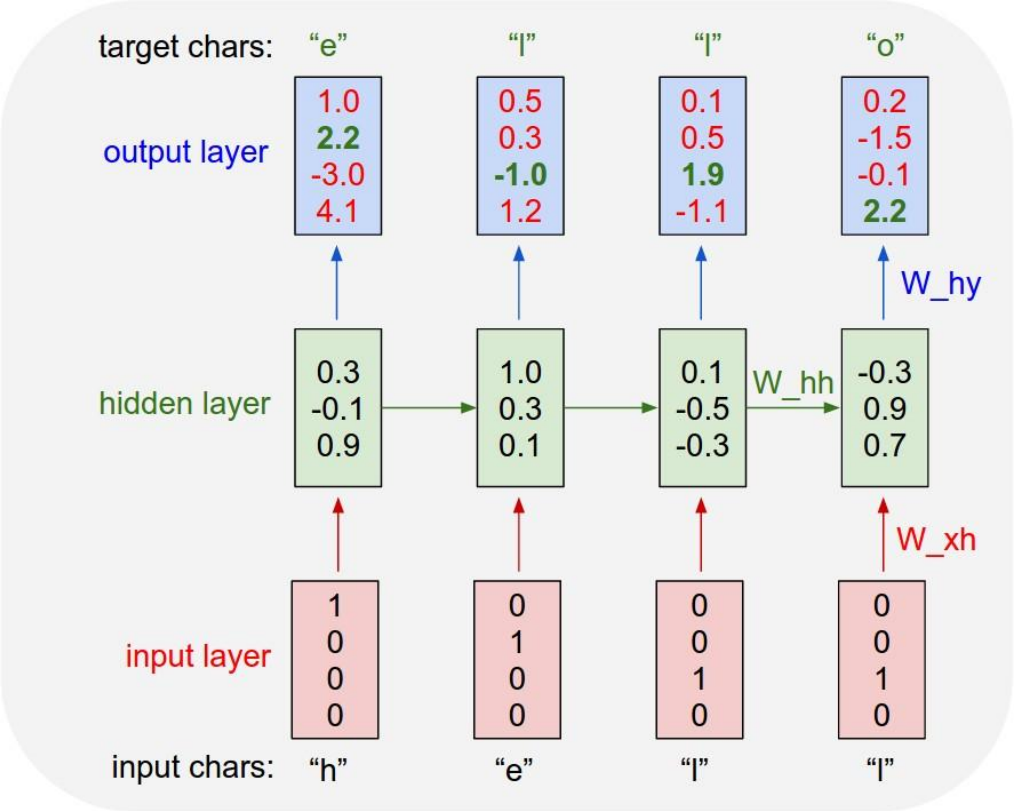
RNN-LMs can be used to generate text

e.g. speech recognition, machine translation, summarization



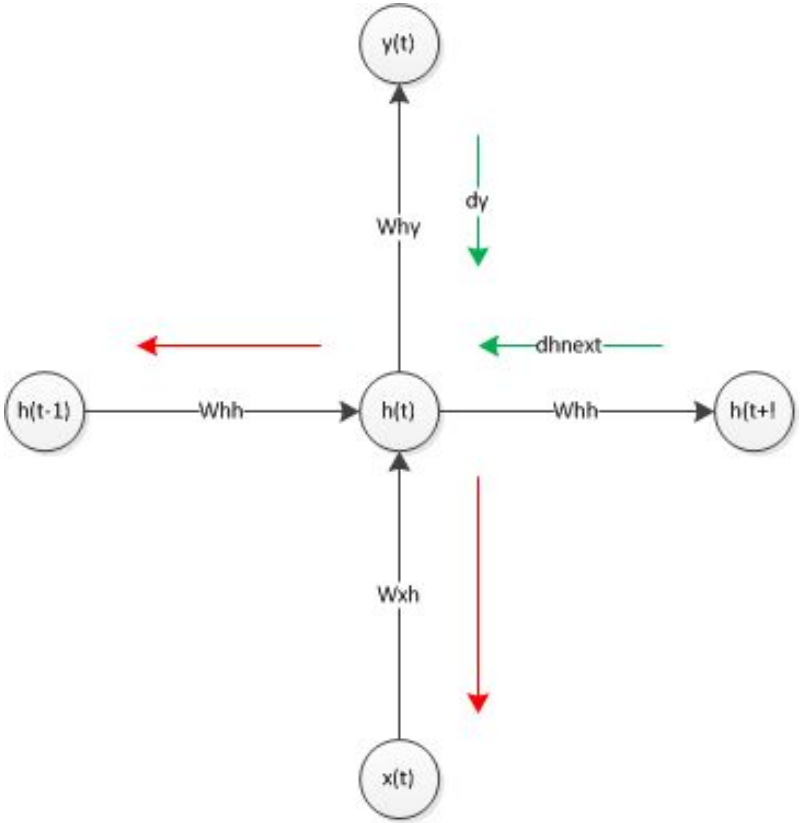
This is an example of a *conditional language model*.
We'll see Machine Translation in much more detail later.

Characters generator



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Characters generator



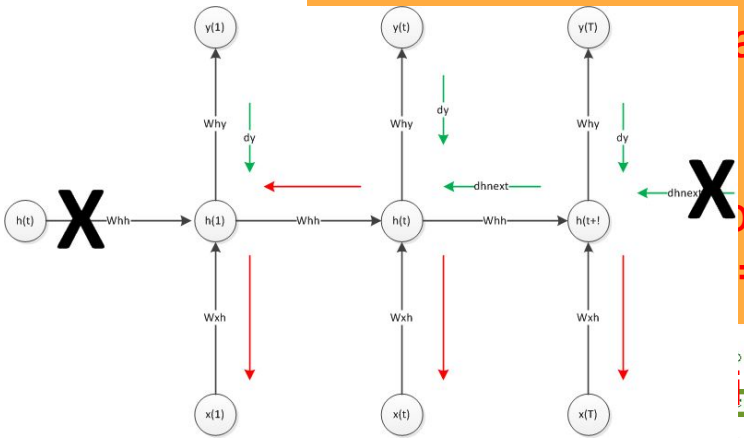
Backpro

Error_Softmax = y_i – t_i (no f' of activation, but still y_i-t_i, review the intro and workshop)

and 0

s[t][target[t]]-1

=ps[t][otherwis



robabilities for next

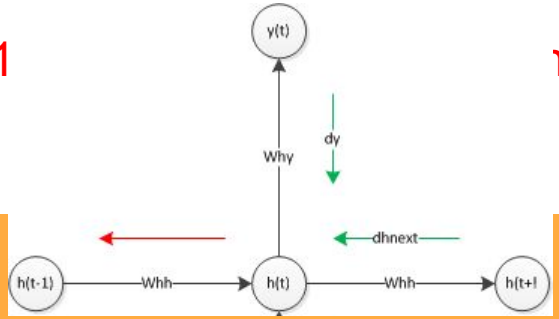
input

tropy loss)

```
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh),
np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

The error backpropagated to Wxx and Wxh is coming from the output + next time step (dhnext)

1 ners 0's



Notice that we start from the end of the sequence and backpropagate the error from this point back. At the final node, no dhnext. At the first node, no hs[t-1]

Notice after the softmax layer, all backprop of errors shall include the non-linearity f'

Essays generation

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

100 epochs: At least it is starting to get an idea about words separated by spaces.
Except sometimes it inserts two spaces. It also doesn't know that comma is amost always followed by a space.

"Tmont thithey" fomesscerliund Keushey.

300 epochs: Quotes and periods are learnt

we counter. He stutn co des. His stanted out one ofler that concossions and
was to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn

500 epochs: The model has now learned to spell the shortest and most common words such as "we", "He", "His", "Which", "and", etc.

Aftair fall unsuch that the hall for Prince Velzonski's that me of her hearly,
and behs to so arwage fiving were to it beloge, pavu say falling misfort how,
and Gogition is so overelical and ofter.

700 epochs: English-Like !!

Essays generation

"Kite vouch!" he repeated by her door. "But I would be done and quarts, feeling, then, son is people...."

1200 epochs: we're now seeing use of quotations and question/exclamation marks. Longer words have now been learned as well

"Why do what that day," replied Natasha, and wishing to himself the fact the princess, Princess Mary was easier, fed in had oftened him. Pierre aking his soul came to the packs and drove up his father-in-law women.

2000 epochs: we start to get properly spelled words, quotations, names, and so on

The picture that emerges is that the model first discovers the general word-space structure and then rapidly starts to learn the words; First starting with the short words and then eventually the longer ones. Topics and themes that span multiple words (and in general longer-term dependencies) start to emerge only much later.

Wikipedia

Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by **[[John Clair]], [[An Imperial Japanese Revolt]],** associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to **[[Antioch, Perth, October 25|21]]** to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PJS)[**http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm** Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule, was starting to signing a major tripad of aid exile.]]

Markdown

```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2002-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT [[Christianity]]</text>
  </revision>
</page>
```

Algebra Lemmas (Latex)

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. $\square$

Valid Latex Syntax!!

Linux Source Code (Input=Linux Repo 400MB)

```
/*
 * Increment the size file of the new incorrect UI_FILTER group
information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] |
PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

Compiling!!

The RNN inserts comments every now and then!

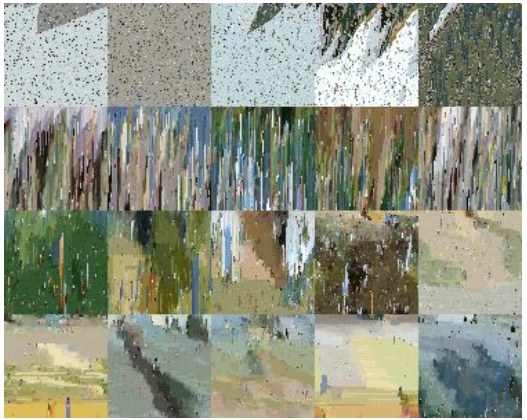
Music generator

- Input is a GuitarPro Notes in ASCII
- Output are ASCII too Play
- Nice music!!
  - https://soundcloud.com/optometrist-prime/recurrence-music-written-by-a-recurrent-neural-network

Images generator

- Input Monet drawings (50 images)

- Output: Monet style drawings!!
  - Output evolution every 100 iterations



Issues with Vanilla RNN

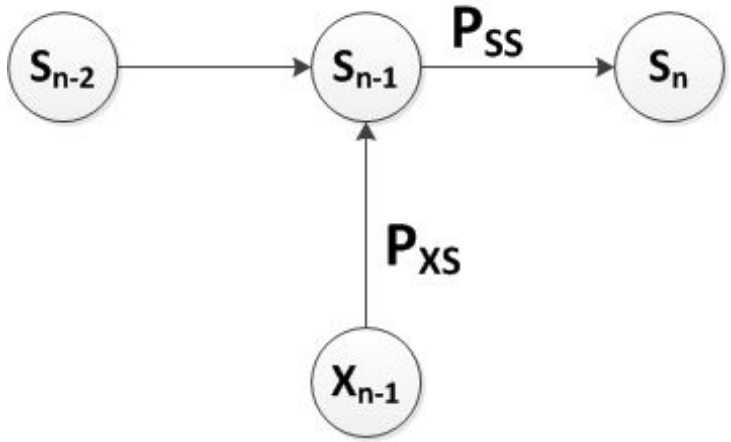
Exploration vs. Exploitation

When to act quickly based on current input?

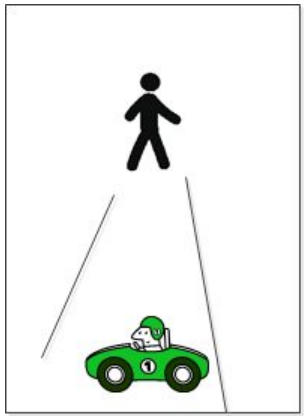
When to consider long history?

Can we have access over longer history and control what to use and what to drop?

- All forms depend on two terms
  - Exploitation: use of known information = history = old state term = forget gate
  - Exploitation: use of new information = inclusion of input = input gate
- All comes from the MDP formulation



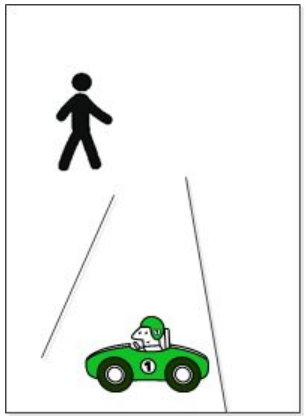
Long term dependency



.....100 frames

V=120 KPH

Short term dependency



V=60 KPH

.....10 frames

Partially observable MDP (POMDP)

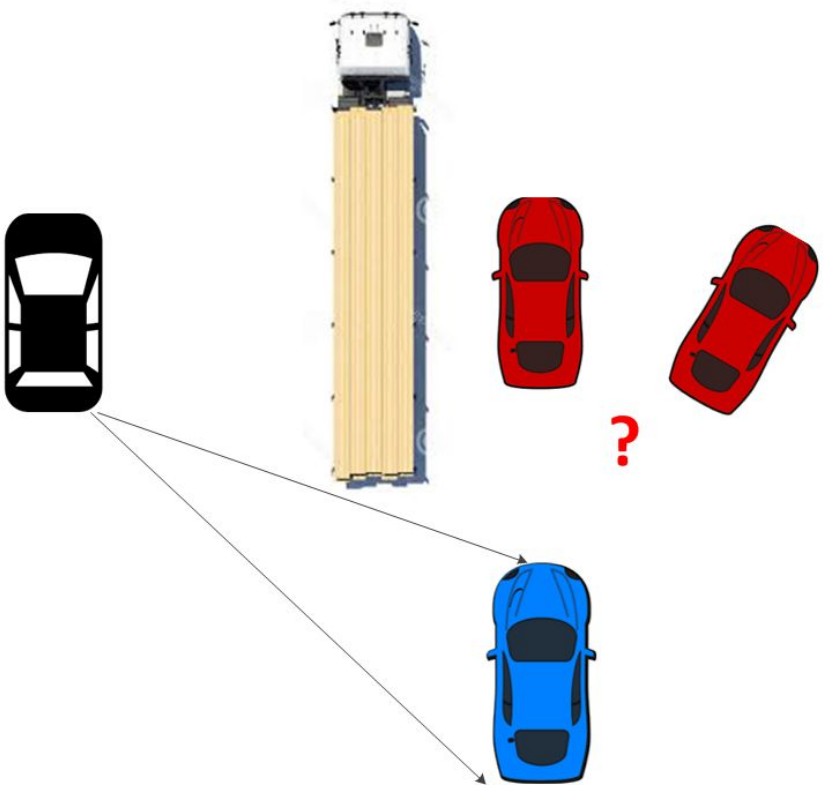
- In Chess, you only need to worry about the current state of the game to take the next action

- In Card games (Estimation, Poker,…etc), the current state of the game is not sufficient

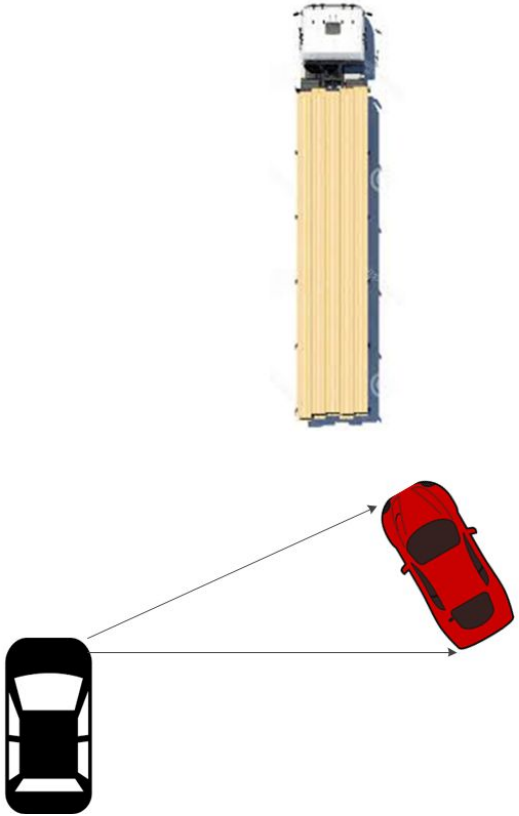
  - You need to be aware what were the previous cards played to take the correct actions (or you estimate ;)

History dependency still controlled by the “summarizing” last hidden state

Occlusion in Automotives



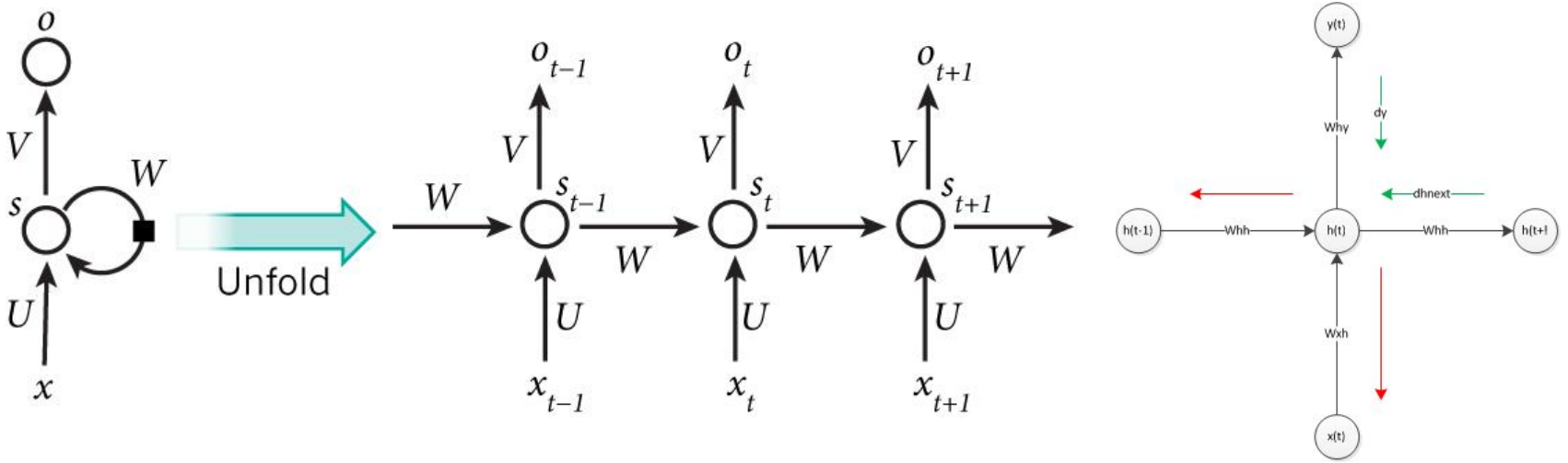
Occlusion in Automotives



Can we have access over longer history and control what to use and what to drop?

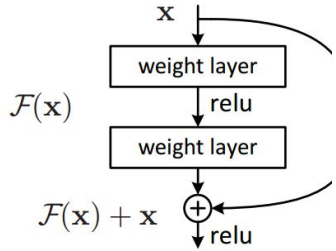
Vanishing gradients in RNN

BPTT “dilutes” the gradient in very “deep” architecture in time



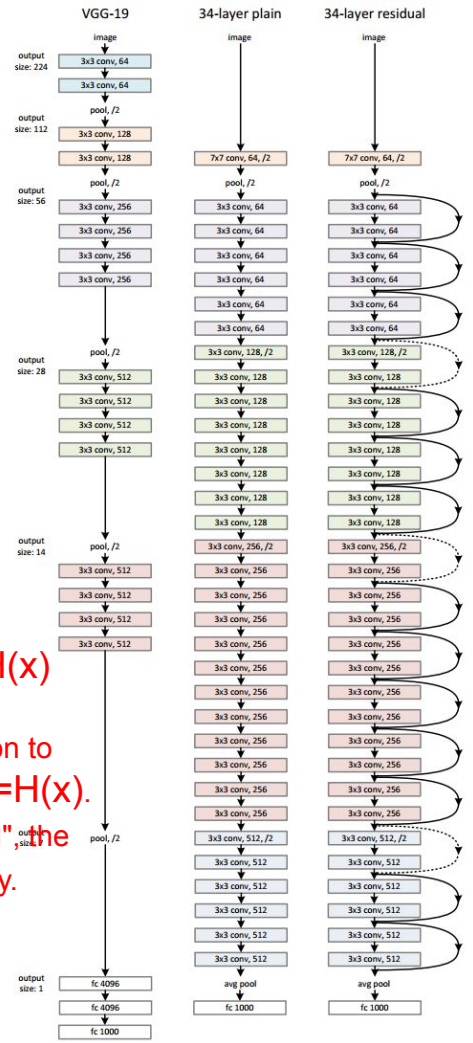
Skip connections to Very Deep Arch

- Very deep networks vanishing gradients
- Remember that the back-propagation of a sum node will replicate the input gradient with no degradation.



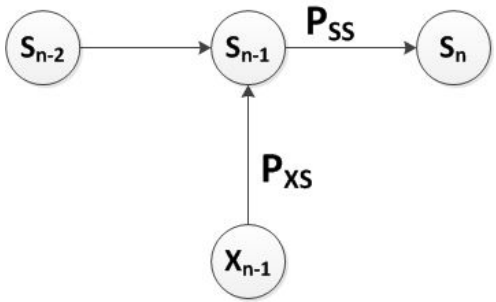
Can we do the same in time?
Skip connection through time!

So they view the map F(x):=H(x) −x as some residual map.
They use a skip layer connection to cast this mapping into F(x)+x=H(x).
So if the residual F(x) is "small", the map H(x) is roughly the identity.



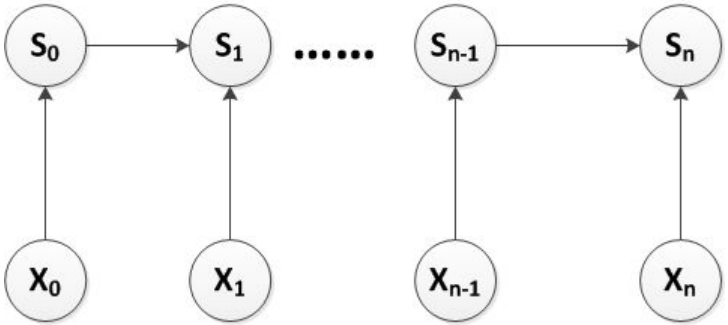
Long-Short Term Memory (LSTM)

the clouds are in the
Ground
Garage
Sky
Airport
Oven



I grew up in France…**[LONG_DOC]** I speak fluent
Arabic
Chineses
French
English
German

How to pay “attention” to words from very OLD context?



Long-Short Term Memory (LSTM)

- Sometimes, we only need to look at recent information to perform the present task.

- For example, consider a language model trying to predict the next word based on the previous ones.

- If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need any further context – it’s pretty obvious the next word is going to be sky.

- In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.

Long-Short Term Memory (LSTM)

- But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.”

- Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back.

- It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

- Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

Long-Short Term Memory (LSTM)

- In theory, RNNs are absolutely capable of handling such “long-term dependencies.”
- A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them.
- The problem was explored in depth by Hochreiter (1991) [German] and Bengio, et al. (1994), who found some pretty fundamental reasons why it might be difficult.
- Thankfully, LSTMs don’t have this problem!

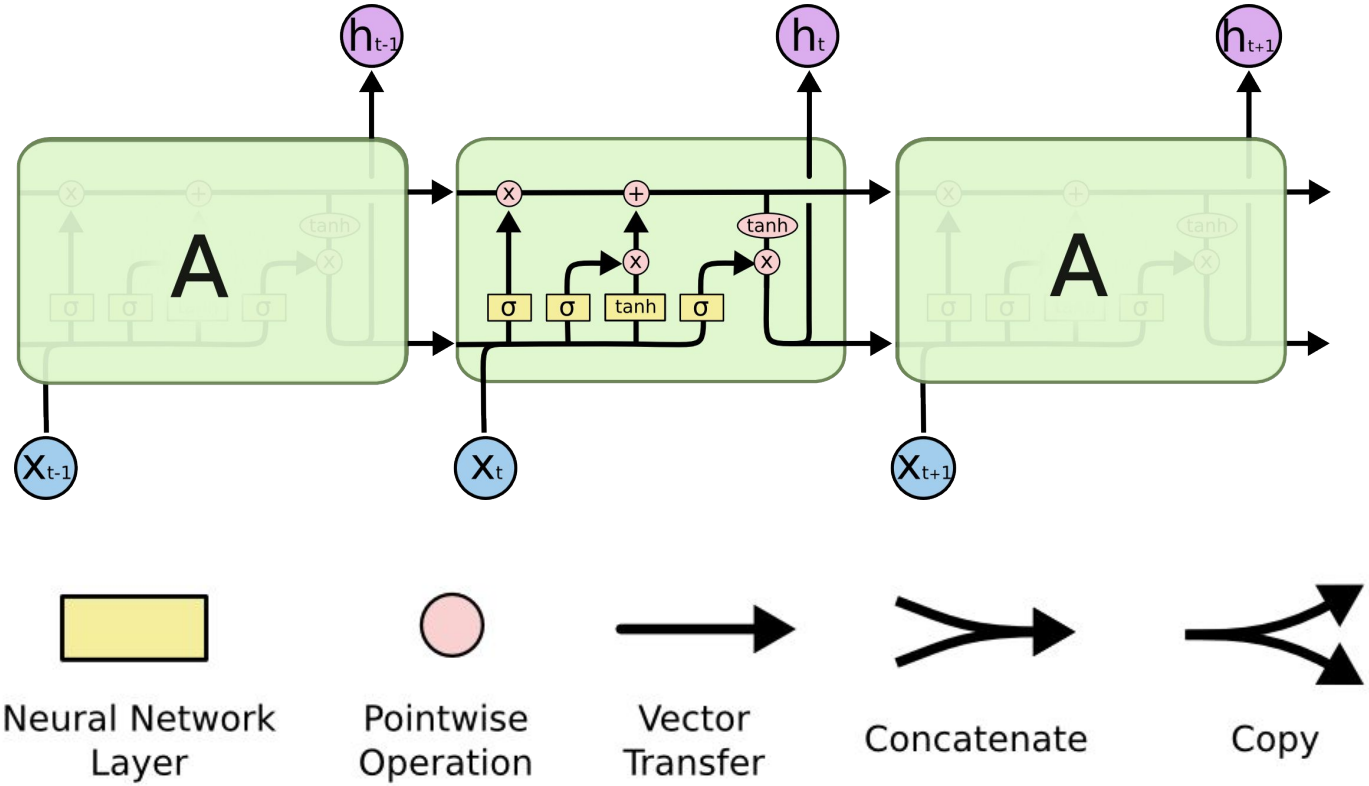
Long-Short Term Memory (LSTM)

- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

- They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.

- LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

Long-Short Term Memory (LSTM)

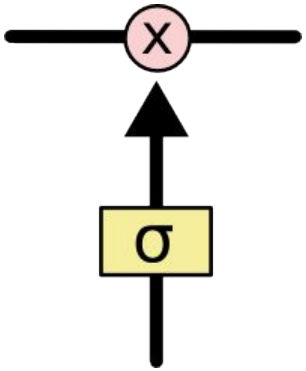


LSTM Step By Step: Gates

- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,

- An LSTM has three of these gates, to protect and control the cell state.” while a value of one means “let everything through!”

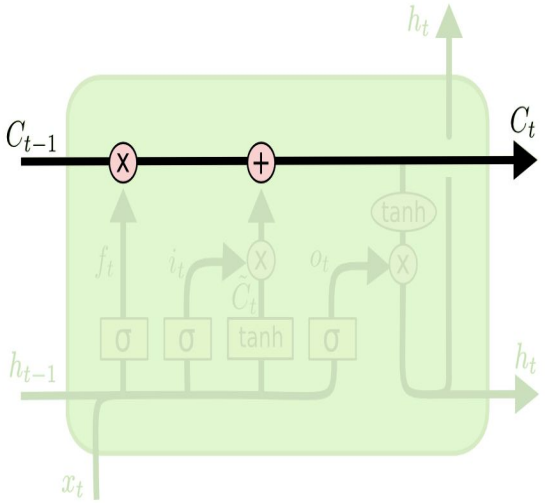


LSTM Step By Step: Cell State

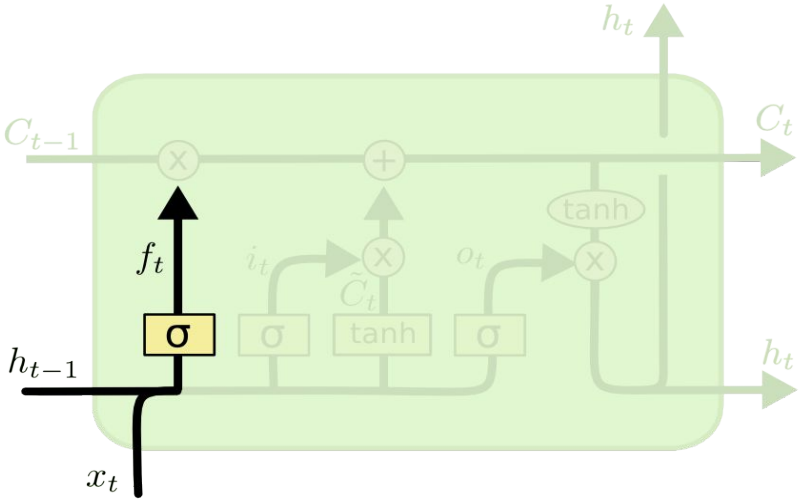
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.



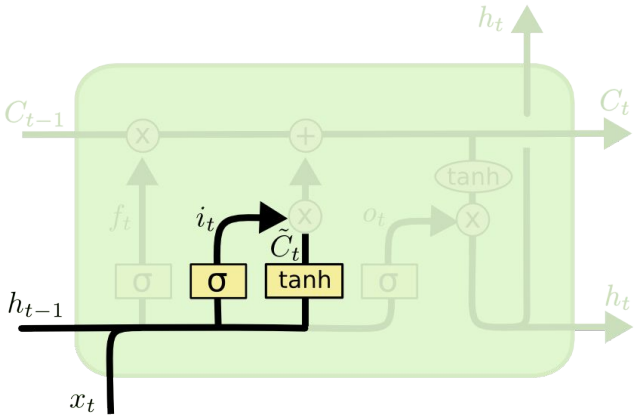
LSTM Step By Step: Forget gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Which information to forget/keep/propagate (1/0*Ct-1) from cell state Ct-1 to next cell state Ct

LSTM Step By Step: Input gate



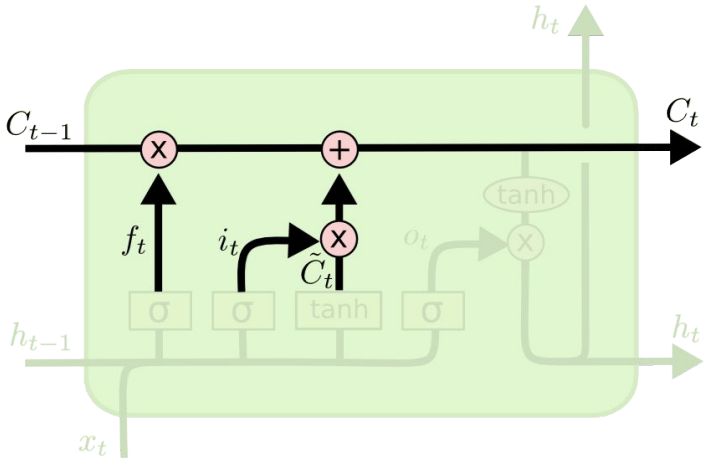
The tanh combination acts as a “regularizer” on the input
It acts as P(xt|ht-1), so that we don’t keep any info from the input independent from what the prev. state was

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

What new information to ADD (+Ct-1) to next cell state Ct
The tanh combines previous hidden state and new input
The sigmoid input gate blocks and let pass the info to add from the prev. hidden + input

LSTM Step By Step: Cell State Update

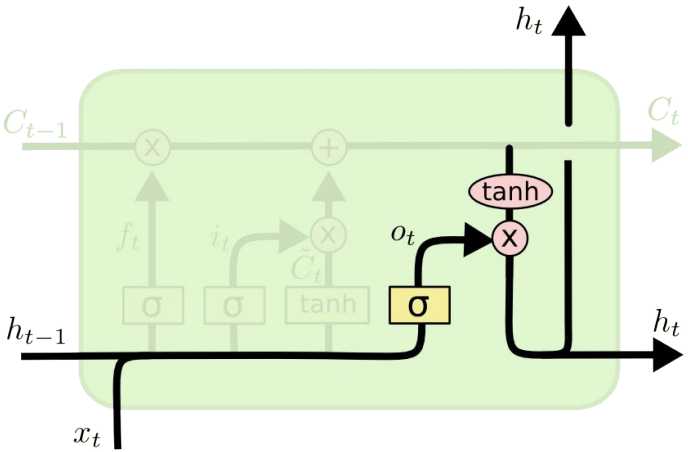


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Exploitation From prev. Cell state

Exploration from new input (regulated by the prev. hidden)

LSTM Step By Step: Output gate



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

What to propagate from input to output (Redundant to be treated in GRNN)

LSTM Matrix Equations

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

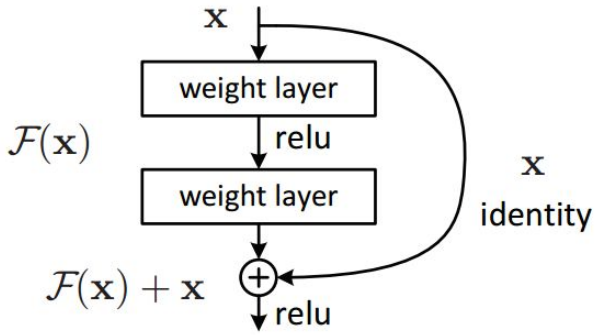
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

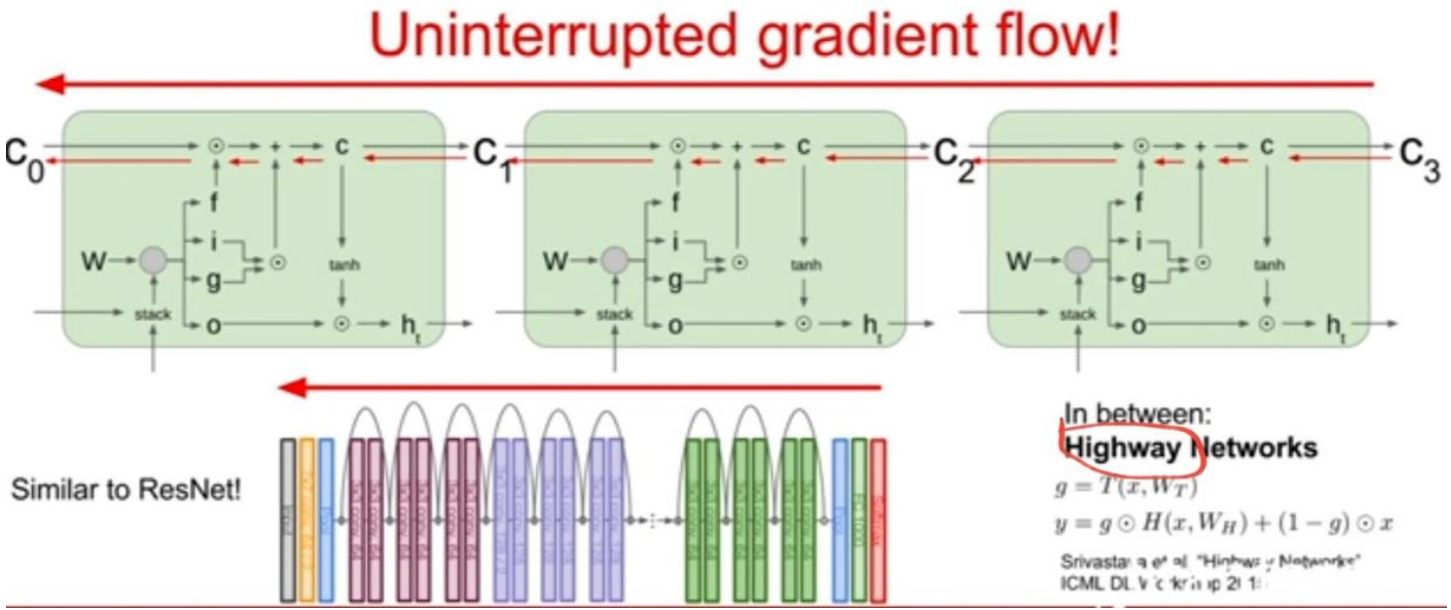
$$h_t = o \odot \tanh(c_t)$$

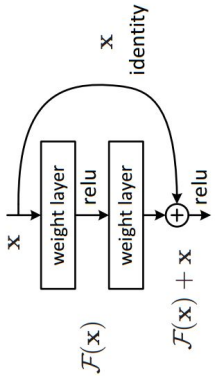
LSTM Gates = Skip connections

- Skip connections to the rescue
- LSTM gates are skip connections ove

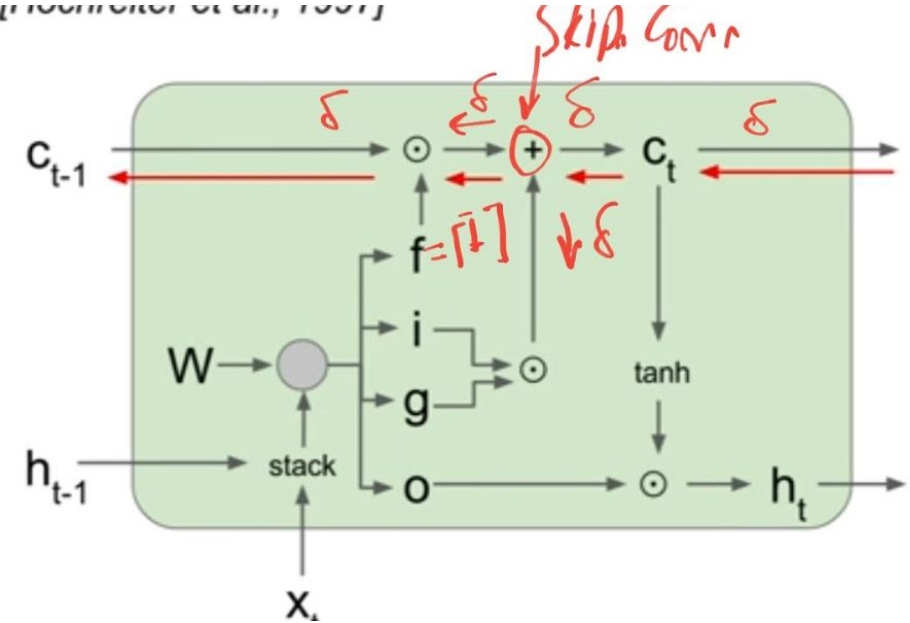


LSTM as skip connection (ResNet)





LSTM as skip connection (ResNet)



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

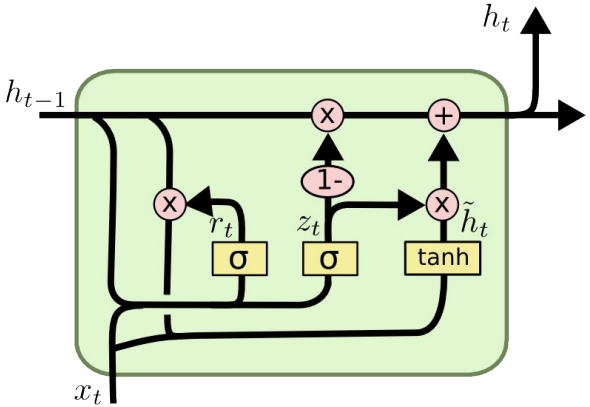
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

In BWD → Grads are just copied at the "+" → no vanishing = Reset!

Gated Recurrent Neural Networks



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

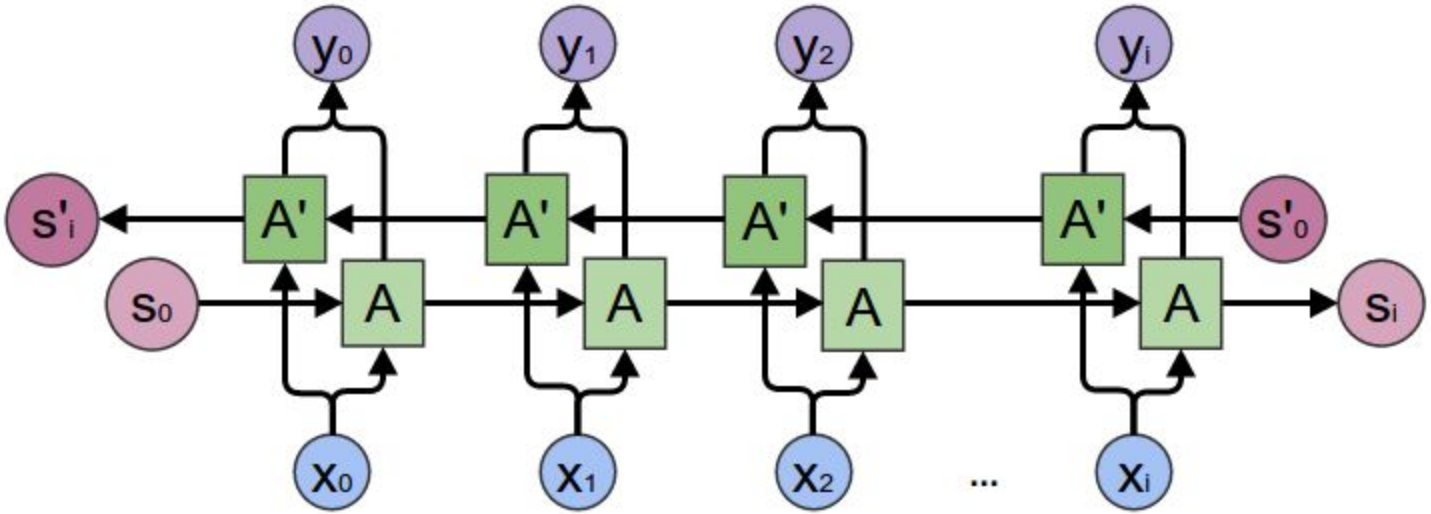
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

What to forget/keep/propagate

What to add from new input

Bi-directional LSTM



Let’s code

LSTM

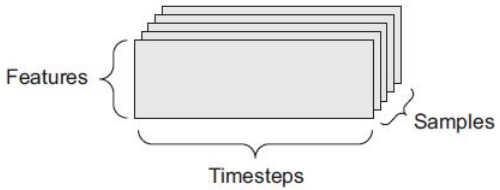
https://colab.research.google.com/drive/1dXeClcTIaFqG3UGmZrdDjPktte6TZ5si#scrollTo=Y9xddpbZJrjm

https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/

Conventions Time series Tensors

3D tensors of shape **(samples, timesteps, features)** samples could be the batch (size=batch_size)

Whenever time matters in your data (or the notion of sequence order), it makes sense to store it in a 3D tensor with an explicit time axis. Each sample can be encoded as a sequence of vectors (a 2D tensor), and thus a batch of data will be encoded as a 3D tensor



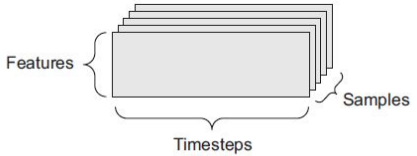
LSTM return sequences

A MUST for stacked LSTM!
Otherwise, use TimeDistributed (NOT preferred)

```
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(n_steps, n_features)))
model.add(LSTM(50, activation='relu'))
```

3D tensors of shape **(samples, timesteps, features)** samples could be the batch (size=batch_size)

Whenever time matters in your data (or the notion of sequence order), it makes sense to store it in a 3D tensor with an explicit time axis. Each sample can be encoded as a sequence of vectors (a 2D tensor), and thus a batch of data will be encoded as a 3D tensor



(samples, timesteps, features)

LSTM/RNN always expects 3D tensor⇒
- hidden_dim
- return_sequences

True → **(samples, timesteps, features)**

False → **(samples, features)**

https://colab.research.google.com/drive/1dXeClcTIaFqG3UGmZrdDjPktte6TZ5si#scrollTo=7LRuYaCqILOn

LSTM return_state

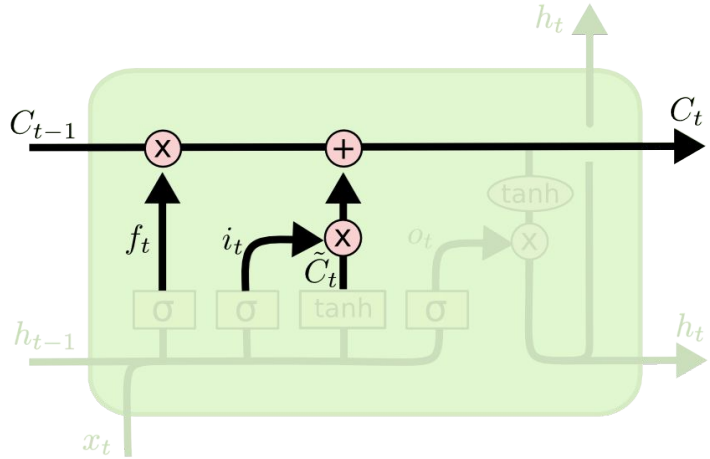
https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/

LSTM has 2 states: hidden and cell (h,c):

- h = output
- c = internal state

In keras, if you set "return_states=True", then 3 things are returned:

- h = lstm1
- h = state_h (again!) → why?
- c = state_c




```
lstm1, state_h, state_c = LSTM(1, return_state=True)
```

LSTM return_state and return_sequences

https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/

In keras, if you set “return_states=True”,
then 3 things are returned:

- h = lstm1 → ALL h’s of all states!
- h = state_h→ Last h
- c = state_c → Last c

```
lstm1, state_h, state_c = LSTM(1, return_sequences=True, return_state=True)(inputs
```

Verify: Last of lstm1=h

```
[array([[[-0.02145359],
         [-0.0540871 ],
         [-0.09228823]]], dtype=float32),
 array([[-0.09228823]], dtype=float32),
 array([[-0.19803026]], dtype=float32)]
```

NLM with RNN

Let’s code

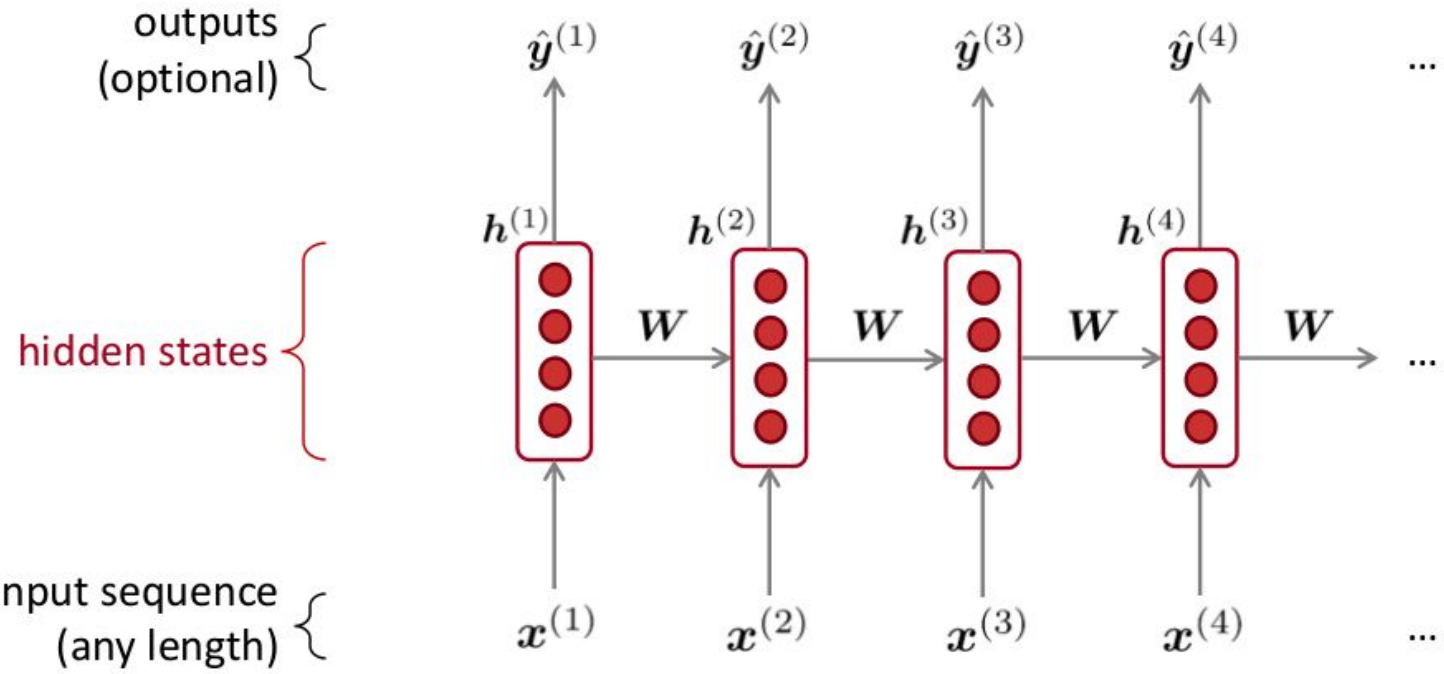
LSTM Text Generation

https://colab.research.google.com/drive/16XcUKh2eWIqIwC3vnnbKUdqkpHqY99LB?usp=sharing

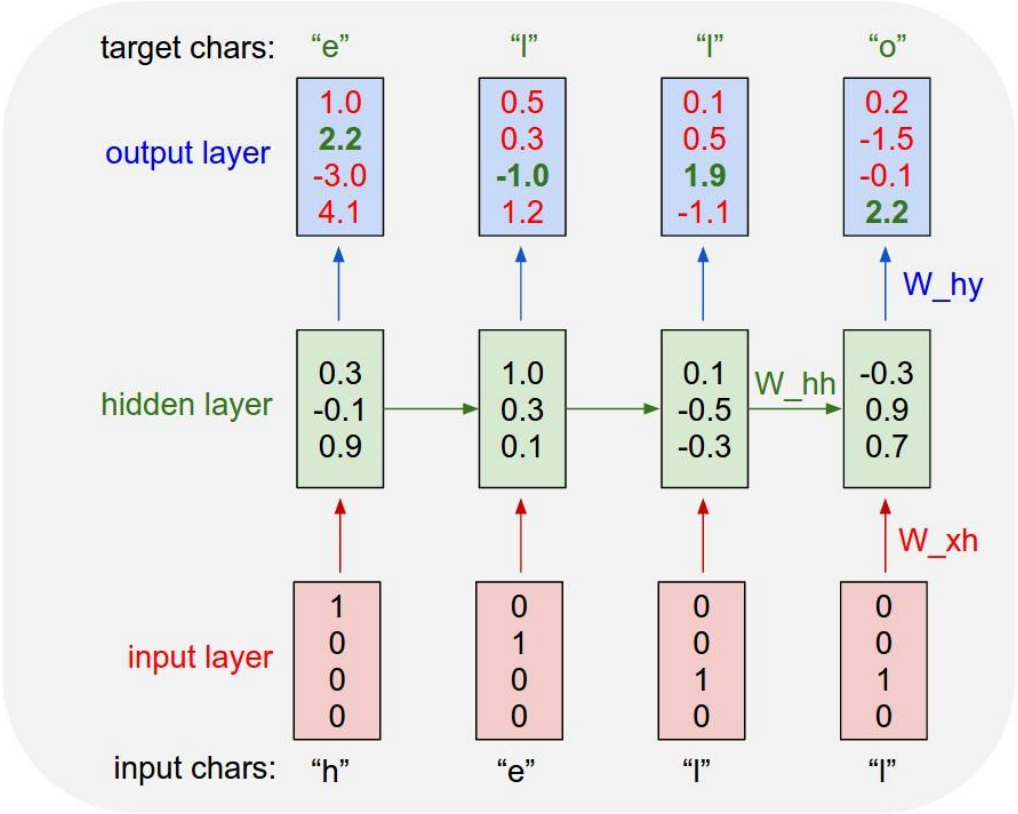
Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights $\boldsymbol{W}$ *repeatedly*

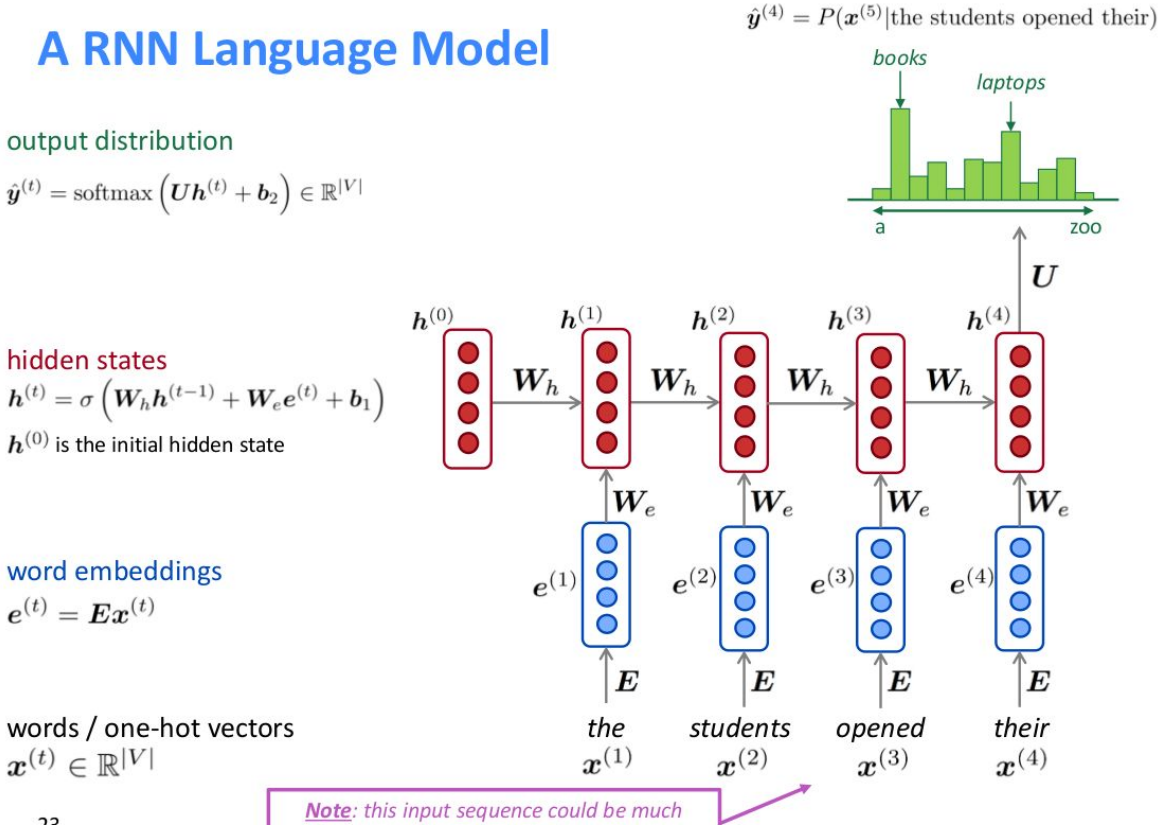


Characters generator = Char level NLM



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Word level NLM



A RNN Language Model

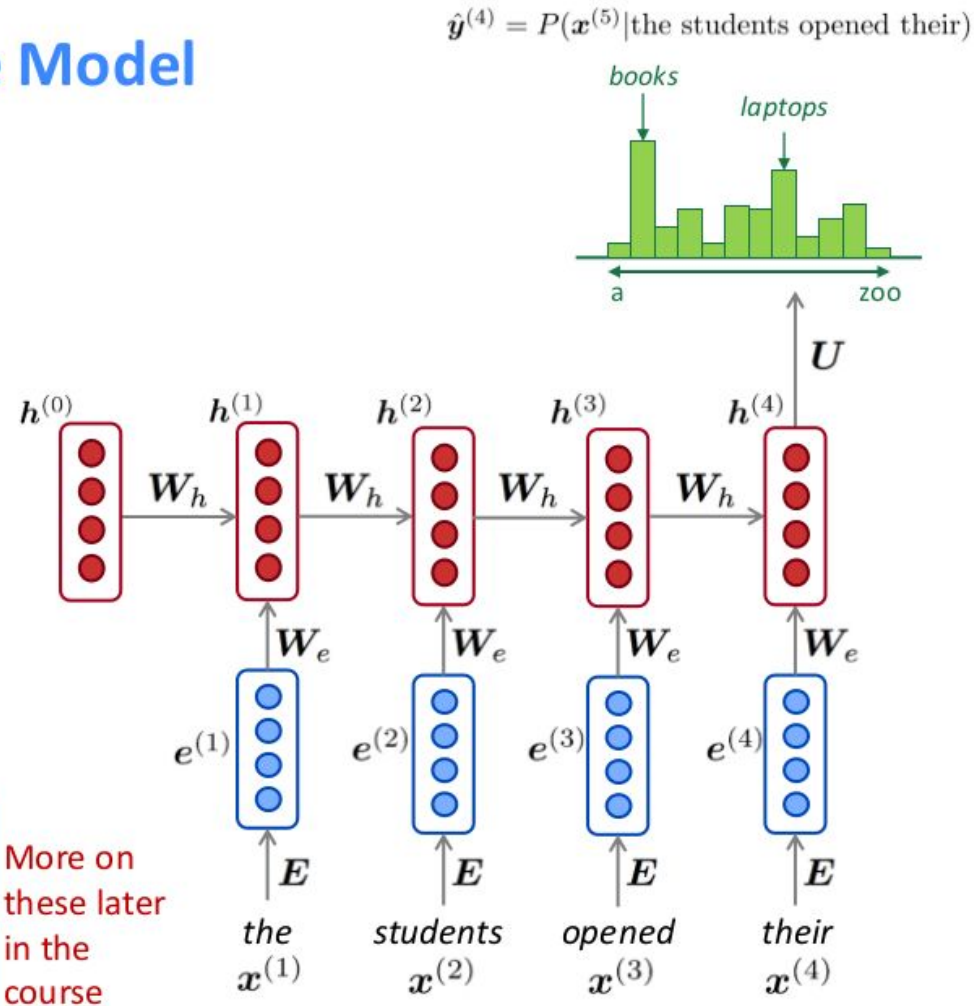
RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

More on these later in the course



Training a RNN Language Model

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ for *every step t*.
  - i.e. predict probability dist of *every word*, given words so far

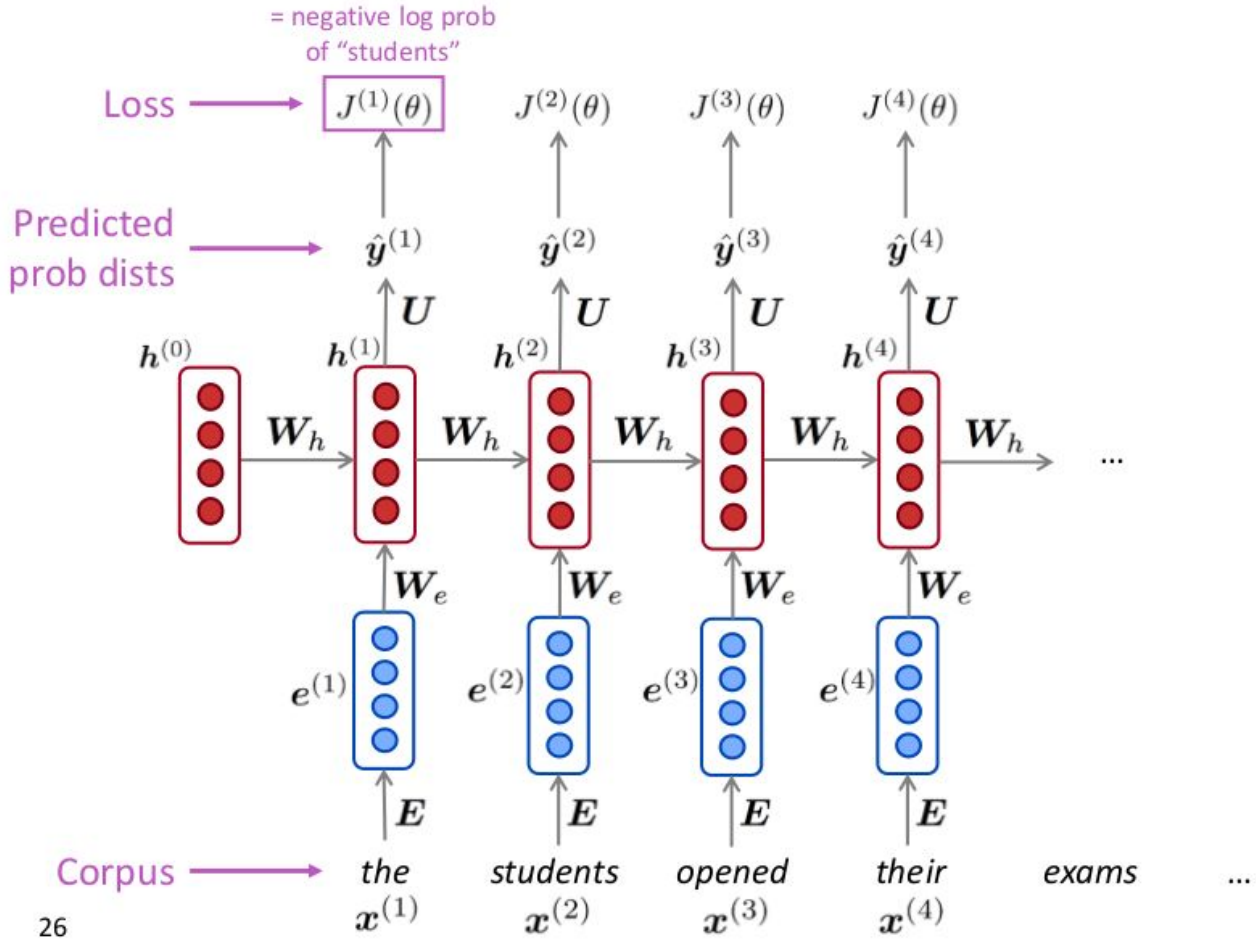
- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

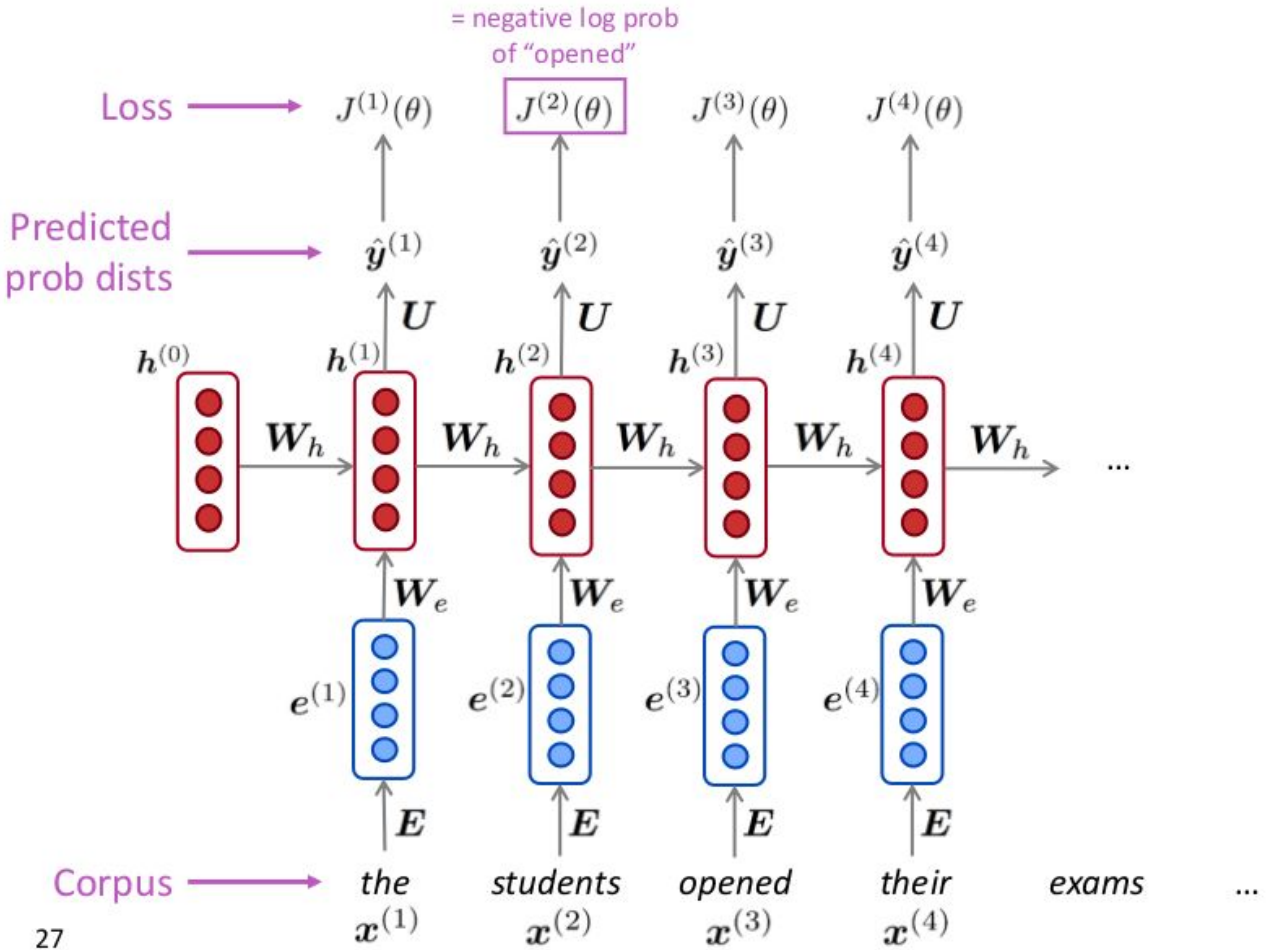
- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

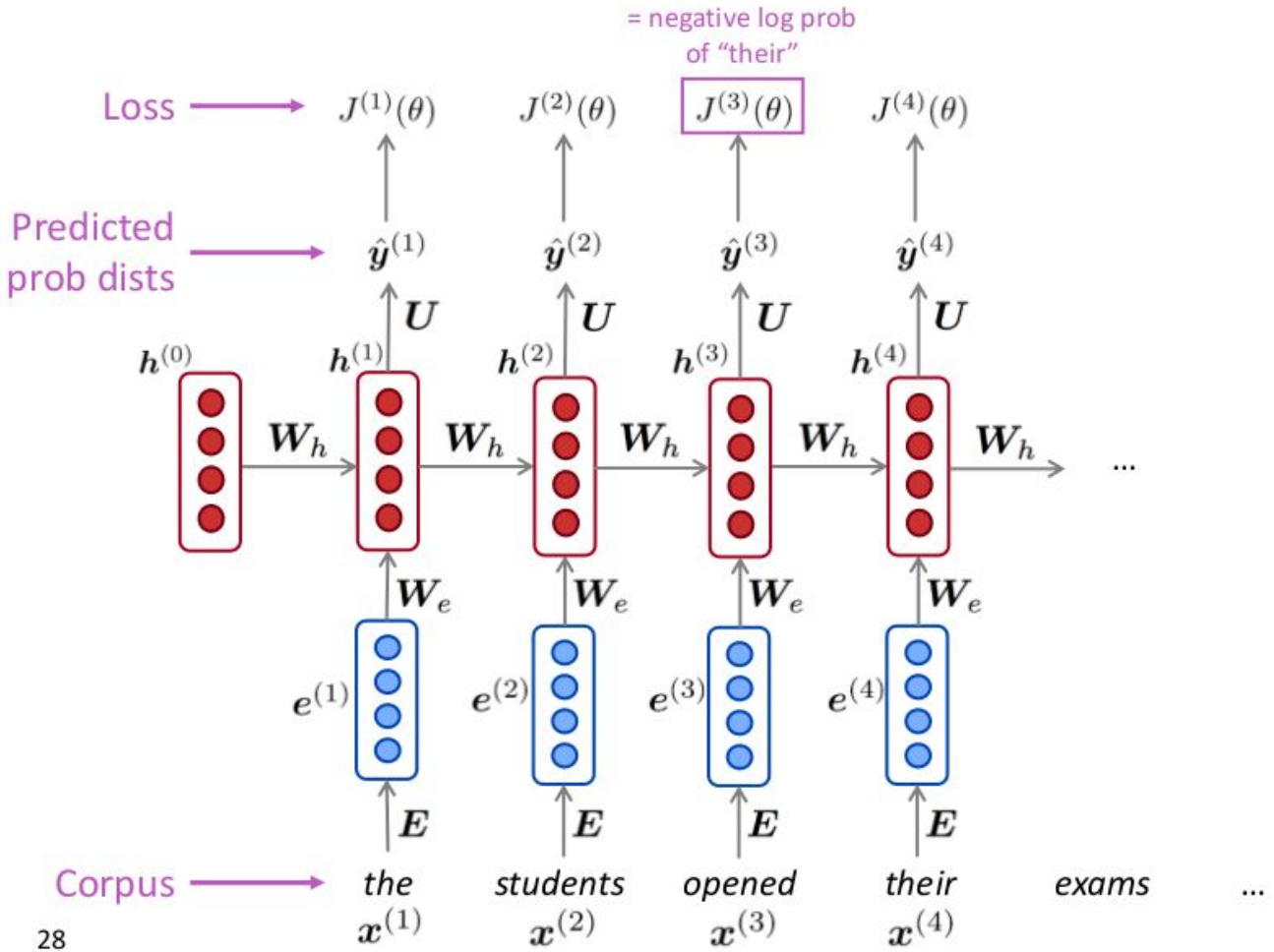
Training a RNN Language Model



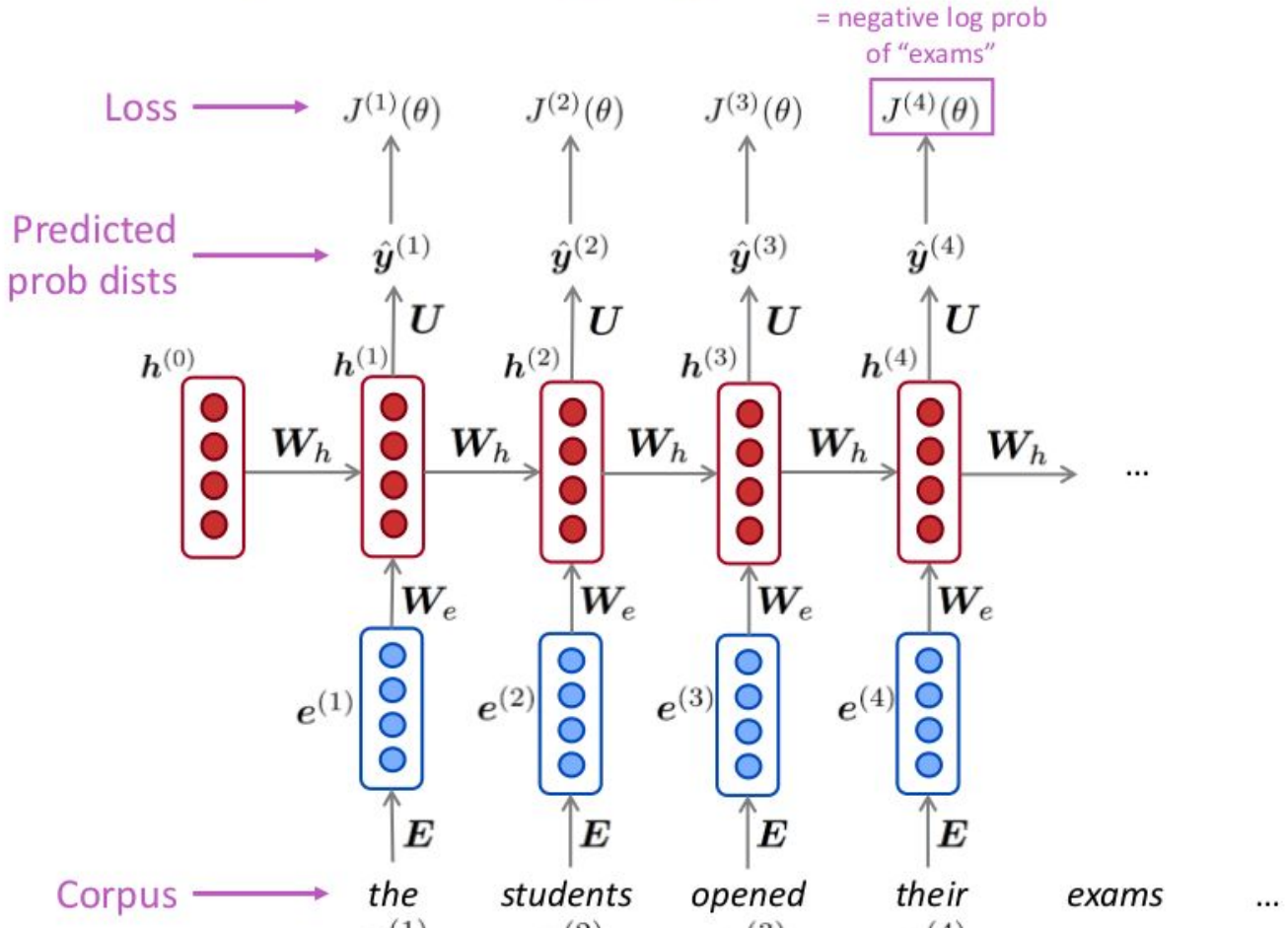
Training a RNN Language Model



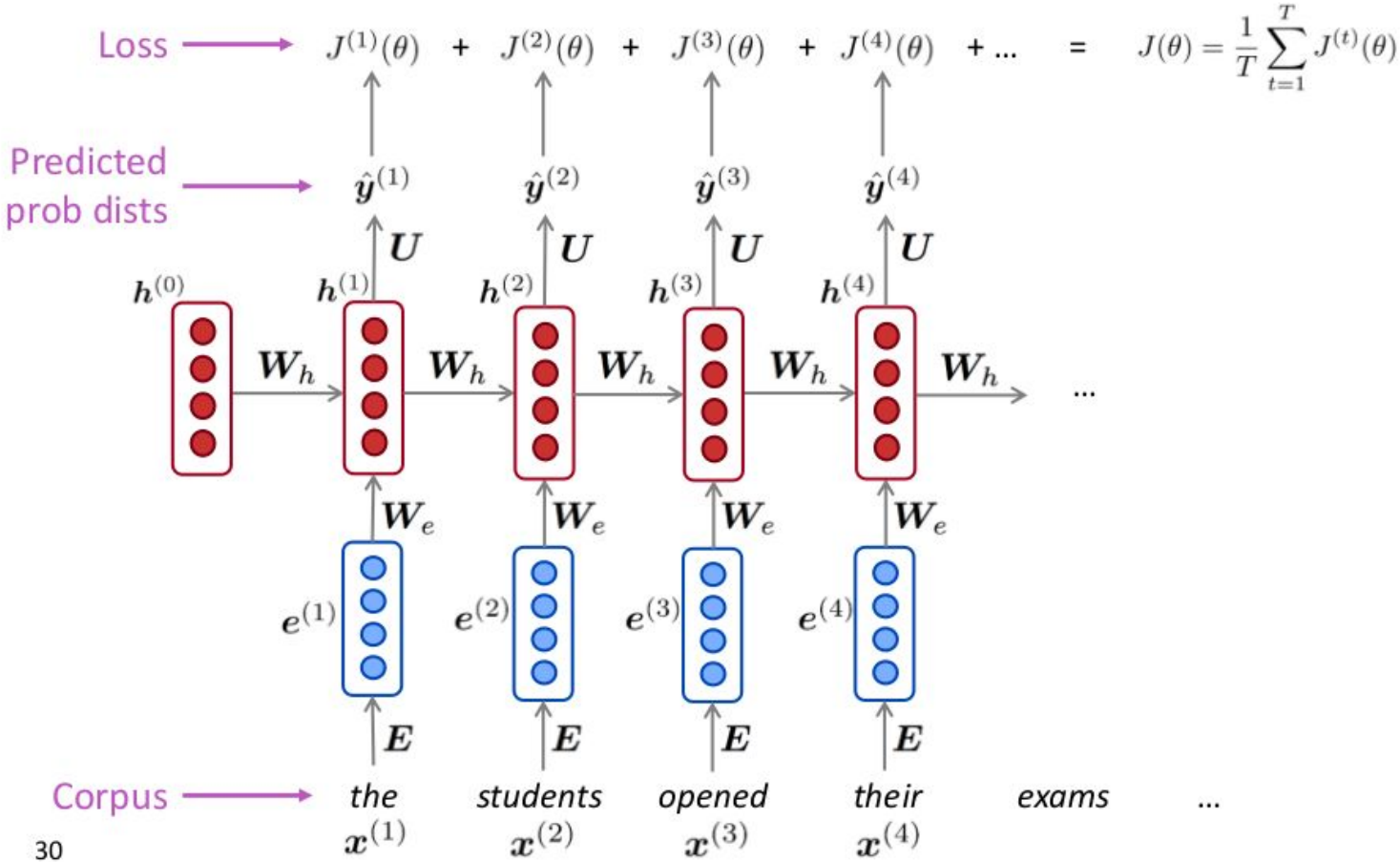
Training a RNN Language Model



Training a RNN Language Model



Training a RNN Language Model



Training a RNN Language Model

- However: Computing loss and gradients across entire corpus $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ is too expensive!

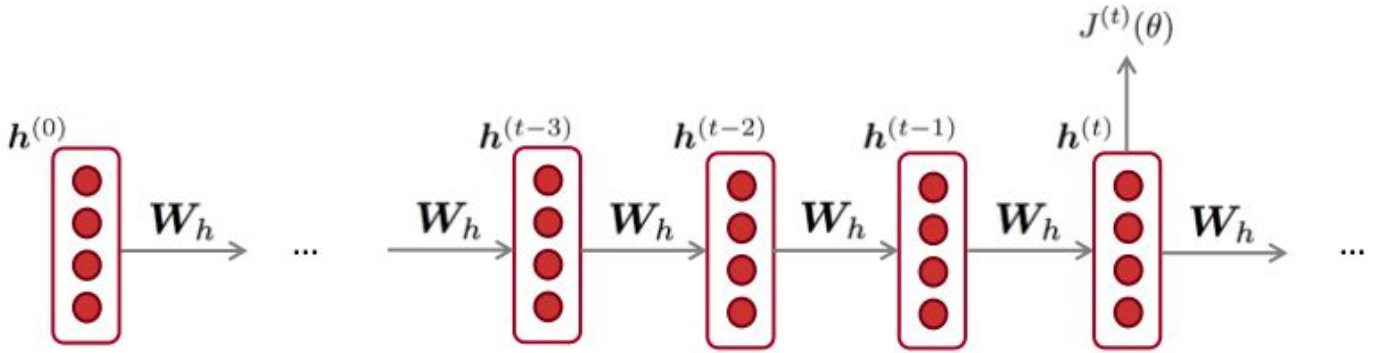
$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

- In practice, consider $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$ as a sentence (or a document)

- Recall: Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.

- Compute loss $J(\theta)$ for a sentence (actually a batch of sentences), compute gradients and update weights. Repeat.

Backpropagation for RNNs



Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W_h}$?

Answer: $$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}$$

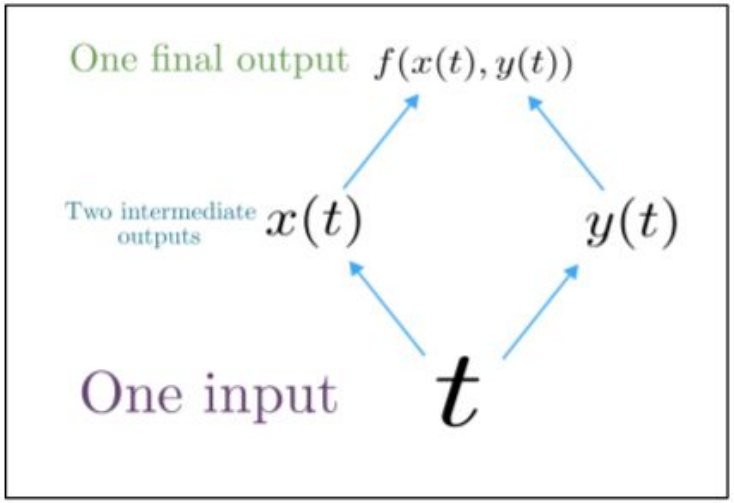
"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Why?

Multivariable Chain Rule

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$



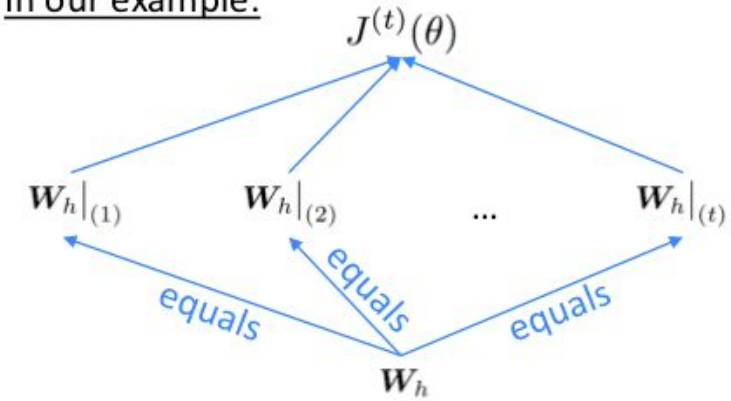
Source:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

Backpropagation for RNNs: Proof sketch

- Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

In our example:



Apply the multivariable chain rule:

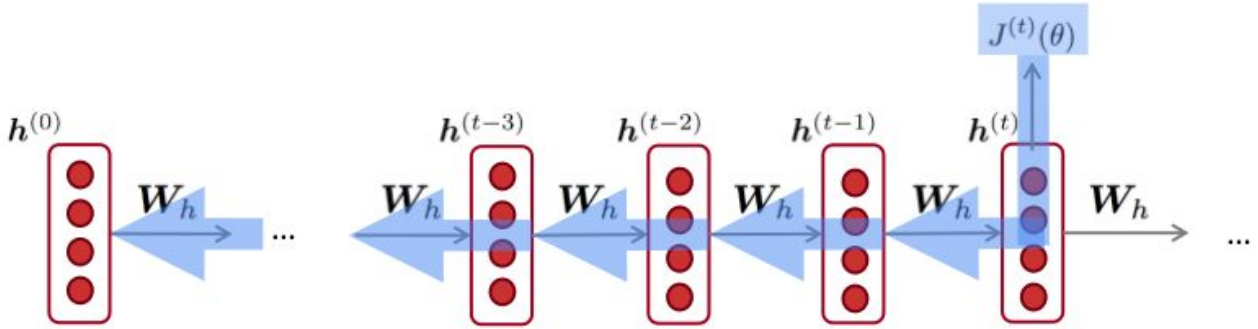
$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\bigg|_{(i)} \frac{\partial W_h|_{(i)}}{\partial W_h}$$

$= 1$

$$= \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial W_h}\bigg|_{(i)}$$

Source:
https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/differentiating-vector-valued-functions/a/multivariable-chain-rule-simple-version

Backpropagation for RNNs



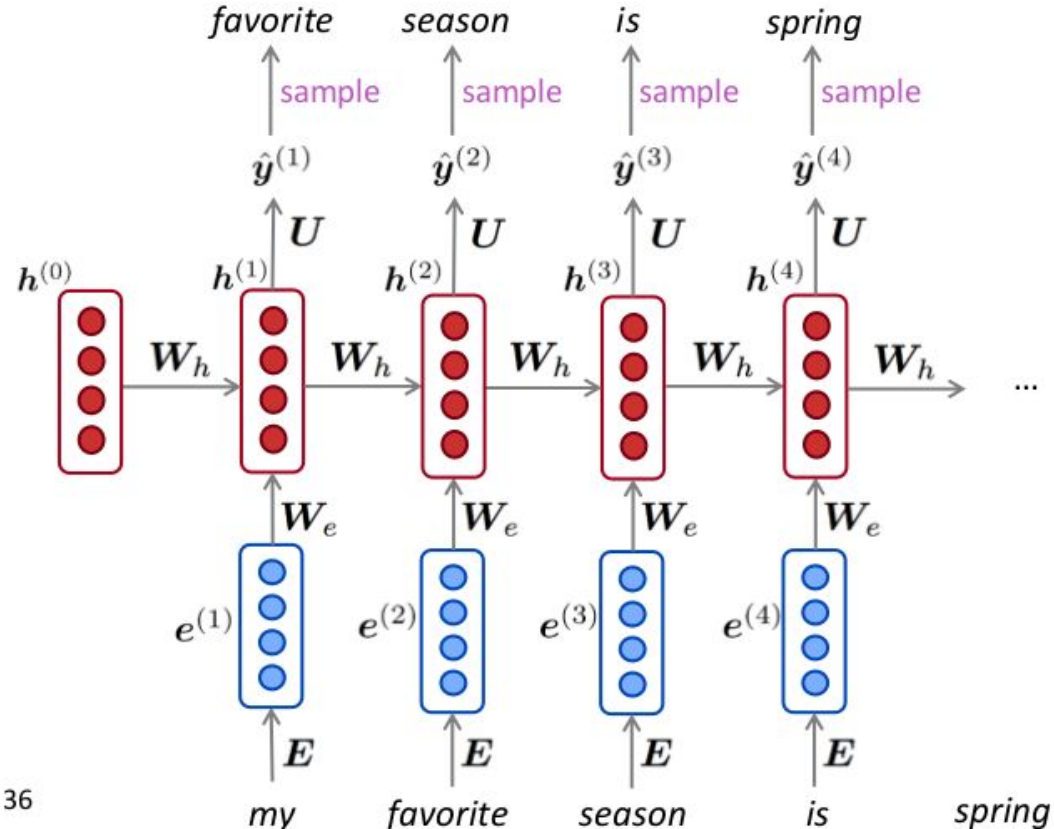
$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \boxed{\sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\Big|_{(i)}}$$

Question: How do we calculate this?

Answer: Backpropagate over timesteps *i*=*t*,…,0, summing gradients as you go.
This algorithm is called **"backpropagation through time"**

Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output is next step's input.



Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on recipes:

```
      Title: CHOCOLATE RANCH BARBECUE
Categories: Game, Casseroles, Cookies, Cookies
      Yield: 6 Servings

      2 tb Parmesan cheese -- chopped
      1 c  Coconut milk
      3    Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer
until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients
and stir in the chocolate and pepper.
```

Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

Let’s code

Word level NLM

https://colab.research.google.com/drive/1WRG86sVhXxI3bADagQOy6DdNVlog_COo?usp=sharing

Char level NLM

https://colab.research.google.com/drive/1qQAqtDK6b9E27Dzu38hBB4cV6QVtuNg0?usp=sharing

RNNs have greatly improved perplexity

n-gram model →

Increasingly complex RNNs ↓

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

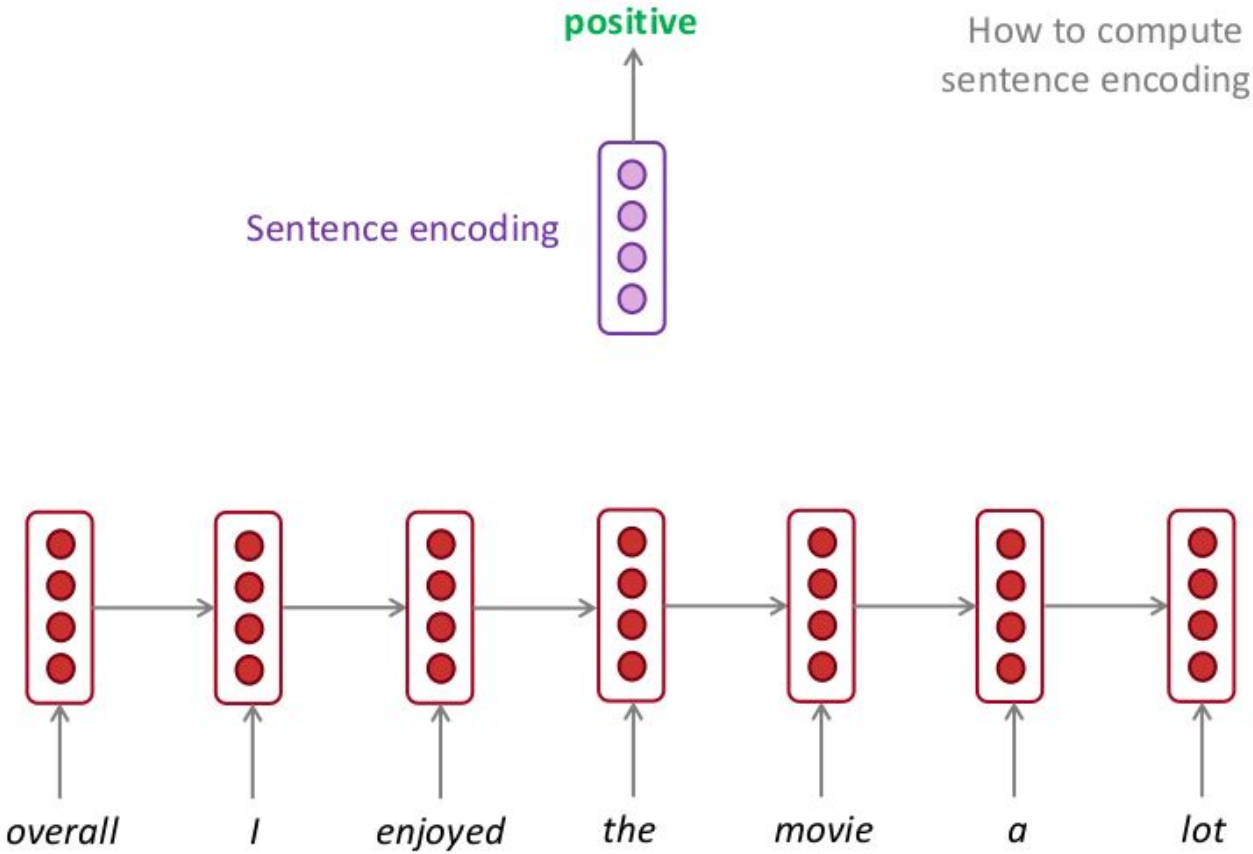
↓ Perplexity improves (lower is better)

Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

RNN for text classification

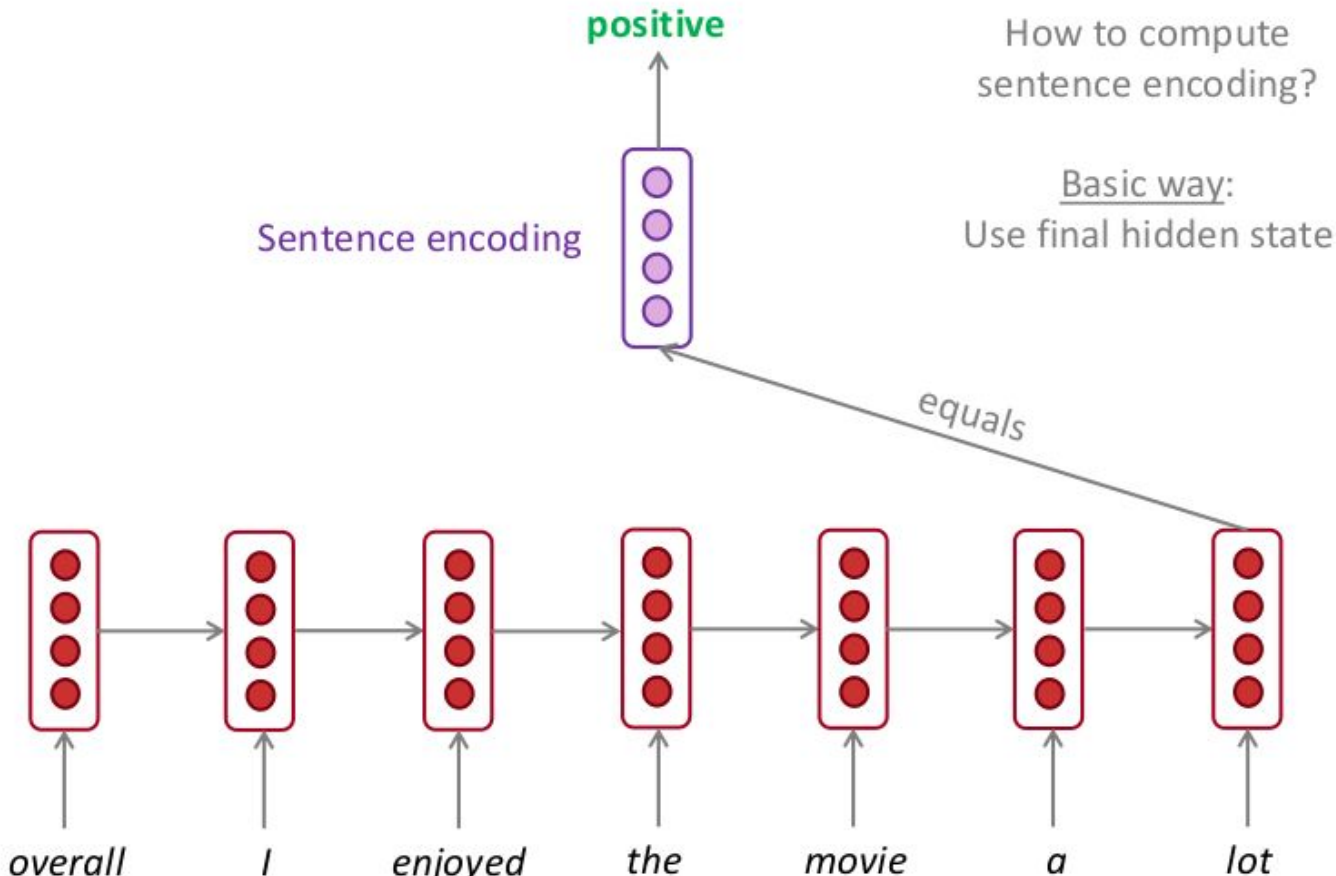
RNNs can be used for sentence classification

e.g. sentiment classification



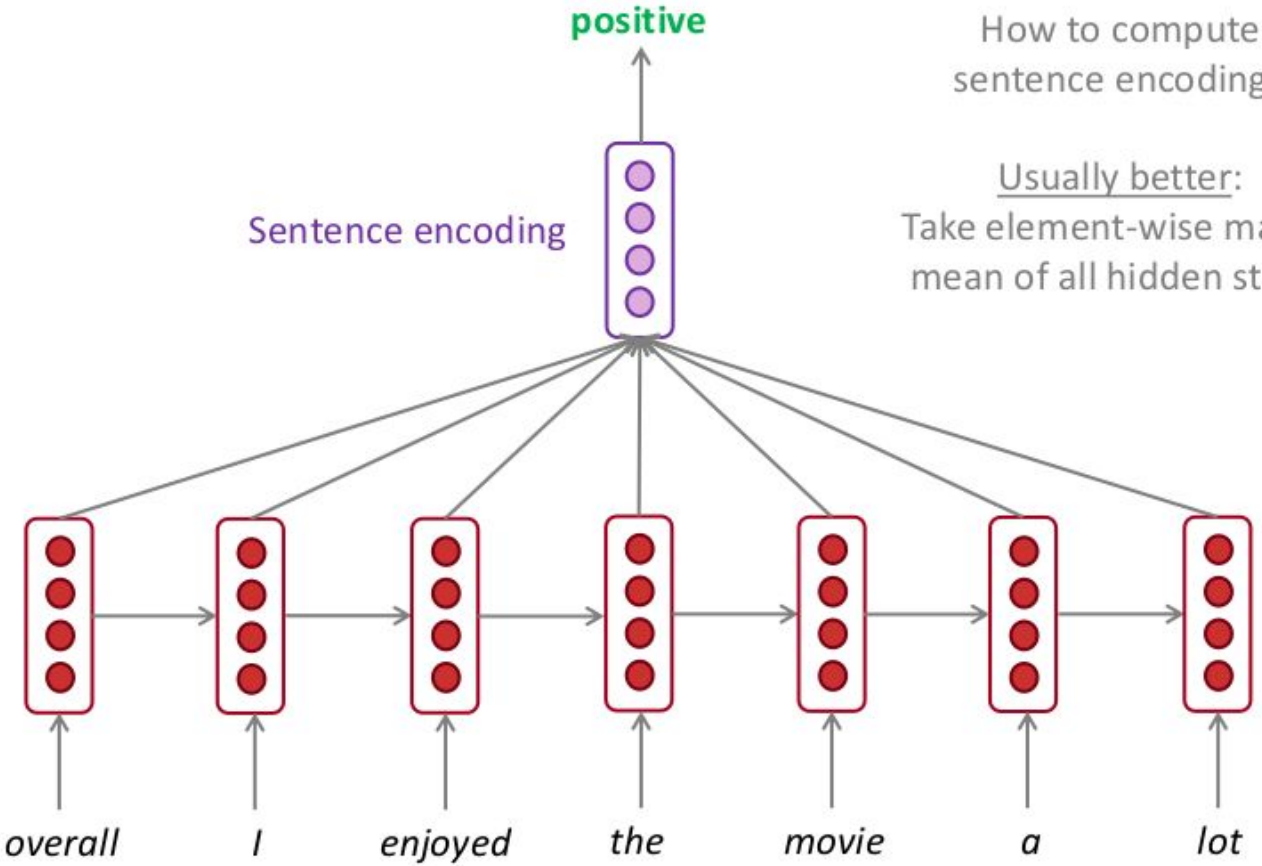
RNNs can be used for sentence classification

e.g. sentiment classification



RNNs can be used for sentence classification

e.g. sentiment classification



Let’s code

LSTM-IMDB Keras dataset

https://colab.research.google.com/drive/1Qllb_h_kuwv8PaIHPZDr2eC4Zo5ljr5b?usp=sharing

LSTM-IMDB Raw dataset

https://colab.research.google.com/drive/1WOUuqsfPZXurupMxlExvScB3ivSiVaV9?usp=sharing

Conv1D for sequence models

What's wrong with RNN?

It's sequential!

- We cannot make use of parallelism of modern computer architectures or GPU!

What's good about it?

- It captures sequence information and summarizes it in a **state**
- Use that state for (condition output on that state):
  - Classification
  - Sequence generation

State = Feature of the input

NLP = Context

Image → Spatial context → Conv2D

Text → Sequence context → RNN ⇒ CNN1D (this lecture)

How to make sequence parsing fast?

Employ parallelism!

Matrix operations → Transformers (Advanced: More on that later)

Convolution operations → CNNs

ConvNets in NLP

CNNs are good to summarize image into one feature vector → How?

It takes advantage of the **spatial** structure of the image

And it's very fast → **Parallel** (conv formula)

Can we use it for the **sequential** structure summarization?

ConvNets in NLP

- Main CNN idea: What if we compute multiple vectors for every possible phrase in parallel?

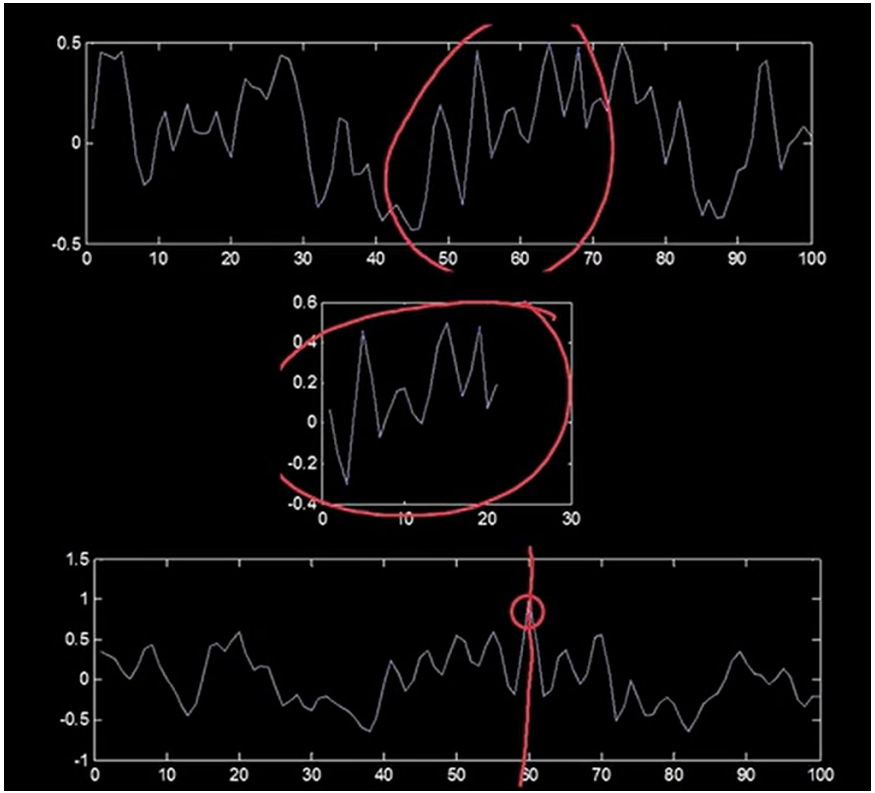
- Regardless of whether phrase is grammatical
- Example: “the country of my birth” computes vectors for:
  - the country, country of, of my, my birth, the country of, country of my, of my birth, the country of my, country of my birth

- Then group them afterwards (more soon)

- Not very linguistically or cognitively plausible but very fast!

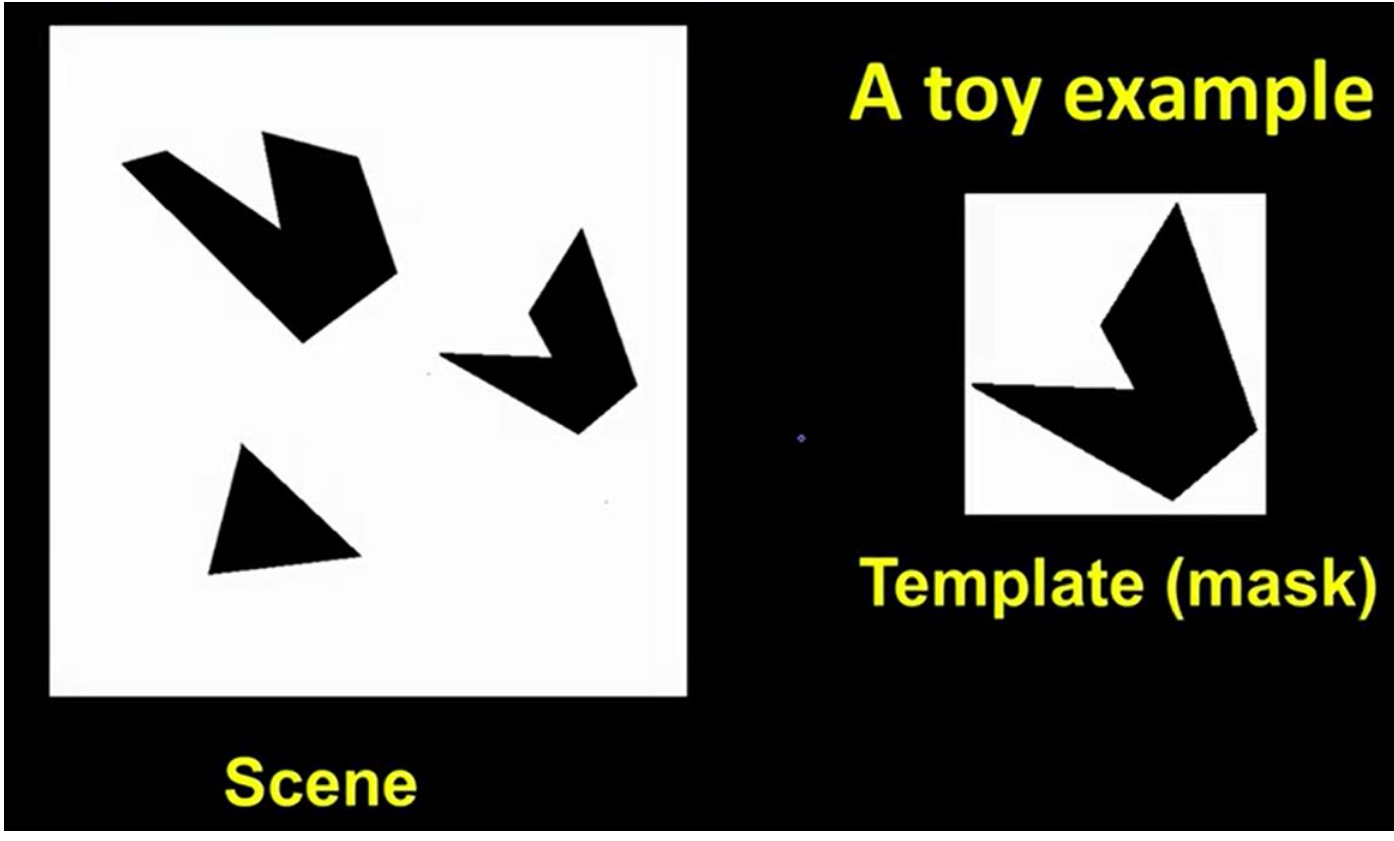
Filters as template matching

1D

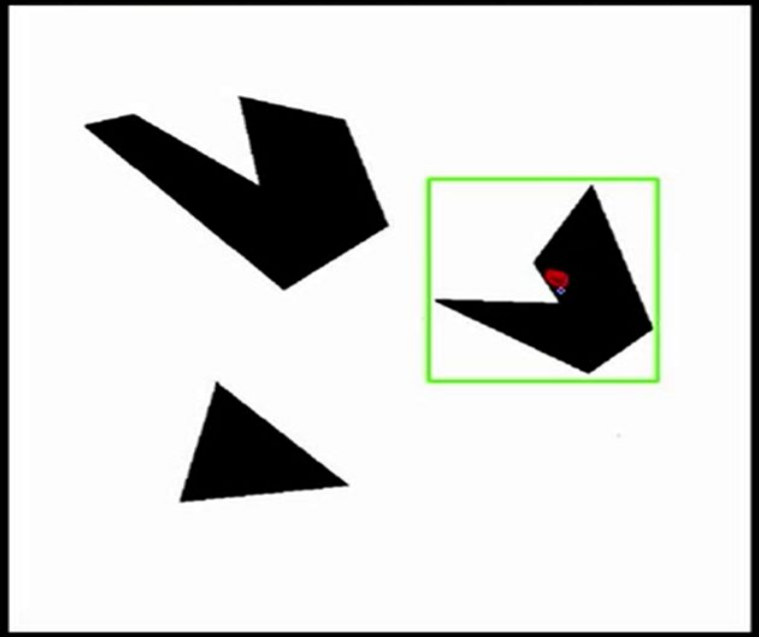


Filters as template matching

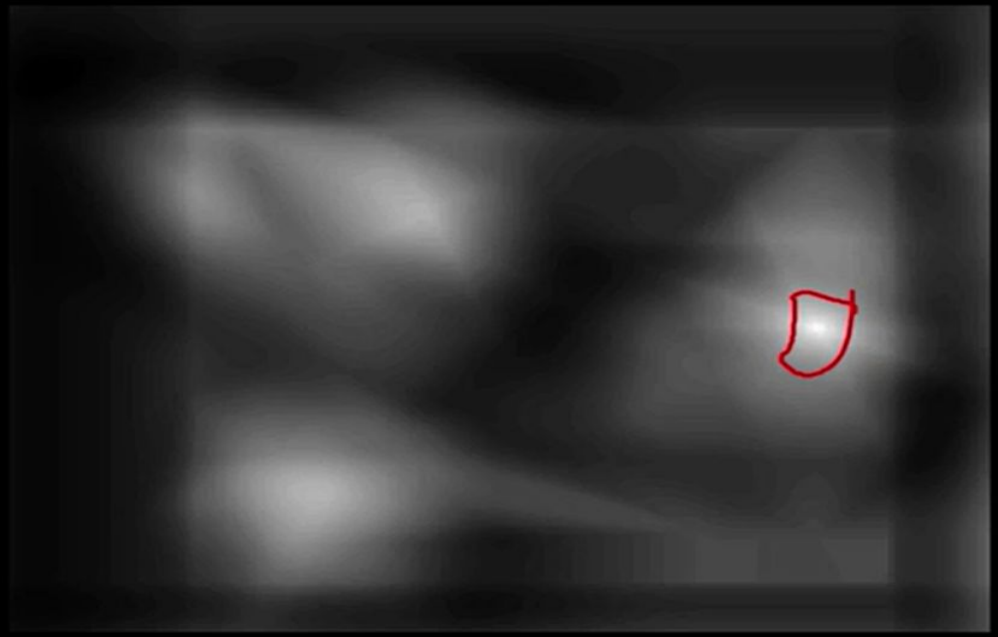
2D



Filters as template matching



Detected template



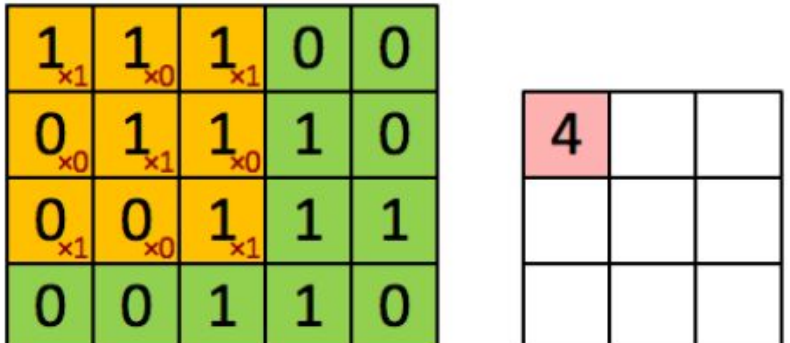
Correlation map

Convolution for feature extraction

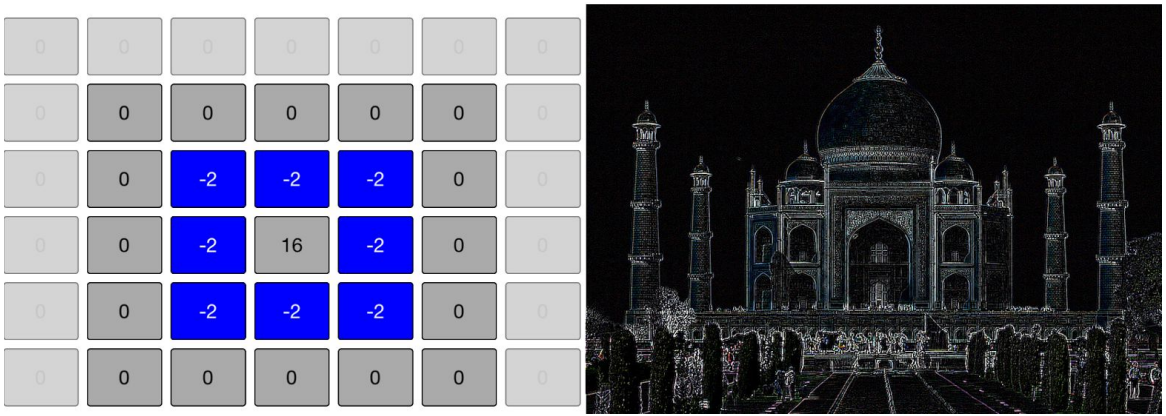
- 1d discrete convolution generally: $(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m].$

- Convolution is great to extract features from images

- 2d example →
- Yellow and red numbers show filter weights
- Green shows input



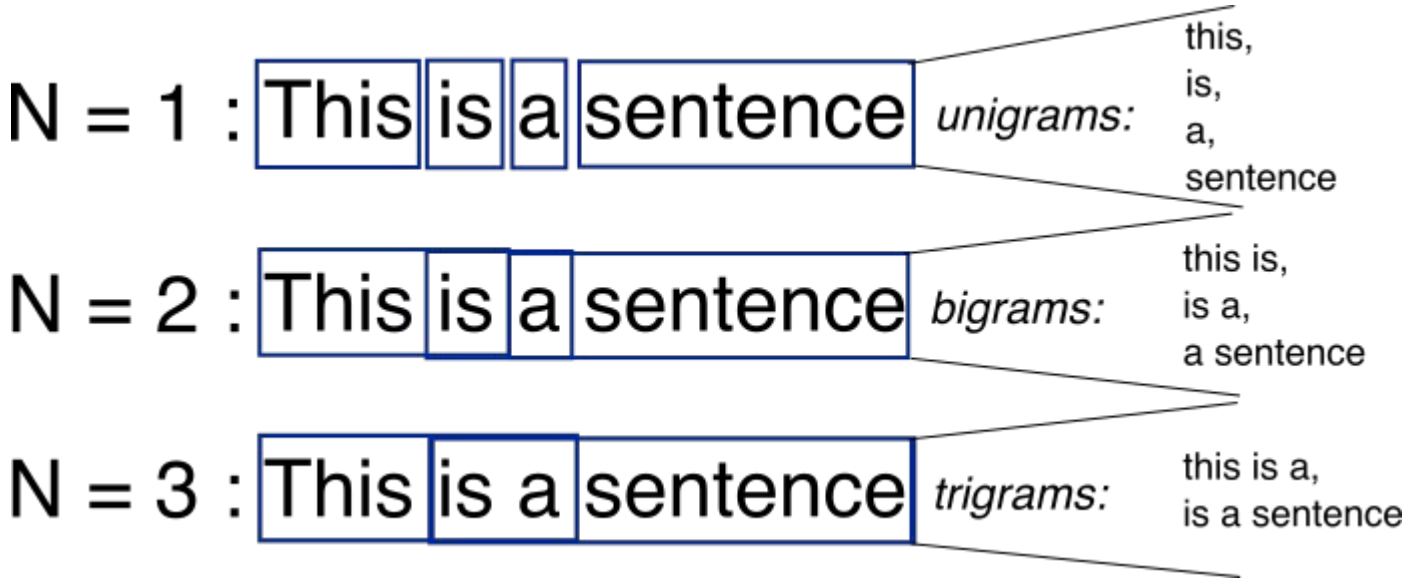
Template matching = features extraction

- Feature = template
- Extraction = matching
- Example of features are edge detectors
- 

How to decide on the kernel weights?

- Kernel weights = features
- Method 1 = Hand crafted like vertical or horizontal edges
- Method 2 = Learning! **ConvNets**

N-grams

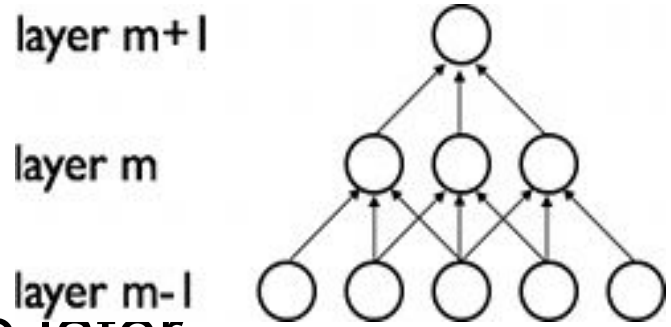

N = 1 :
This
is
a
sentence
unigrams:
this,
is,
a,
sentence
N = 2 :
This
is
a
sentence
bigrams:
this is,
is a,
a sentence
N = 3 :
This
is a
sentence
trigrams:
this is a,
is a sentence

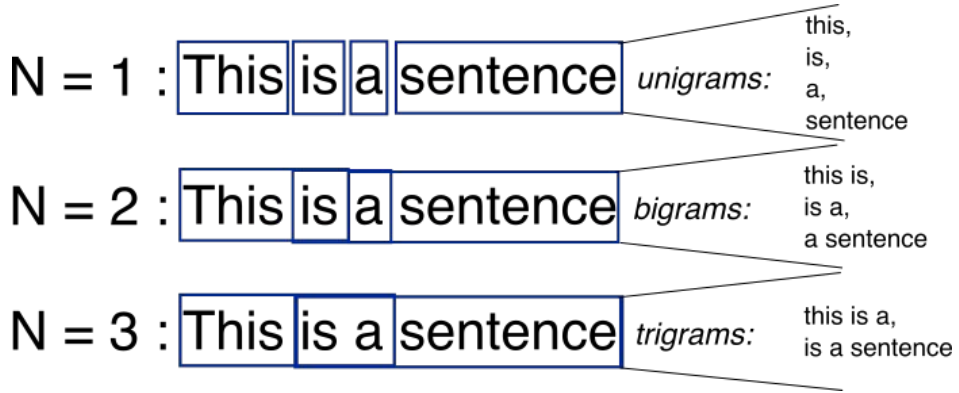
Filters as template matching

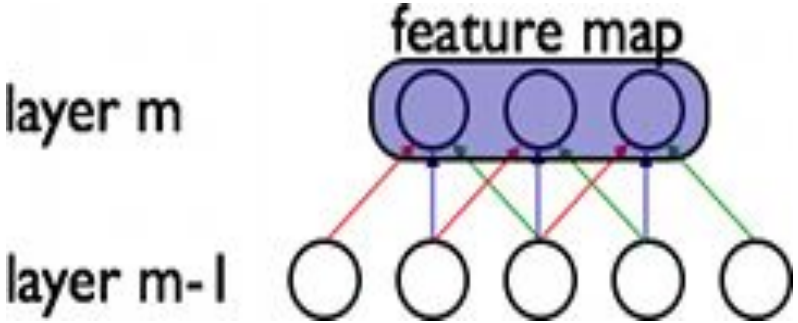
1D
Template=kernel=n-gram

Shared weights = Efficiency

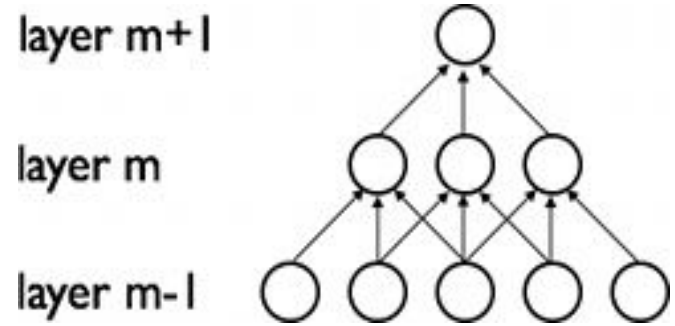
Every n-gram = Channel → See later

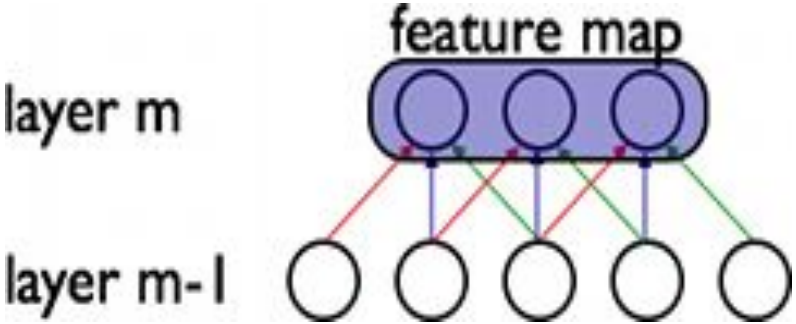


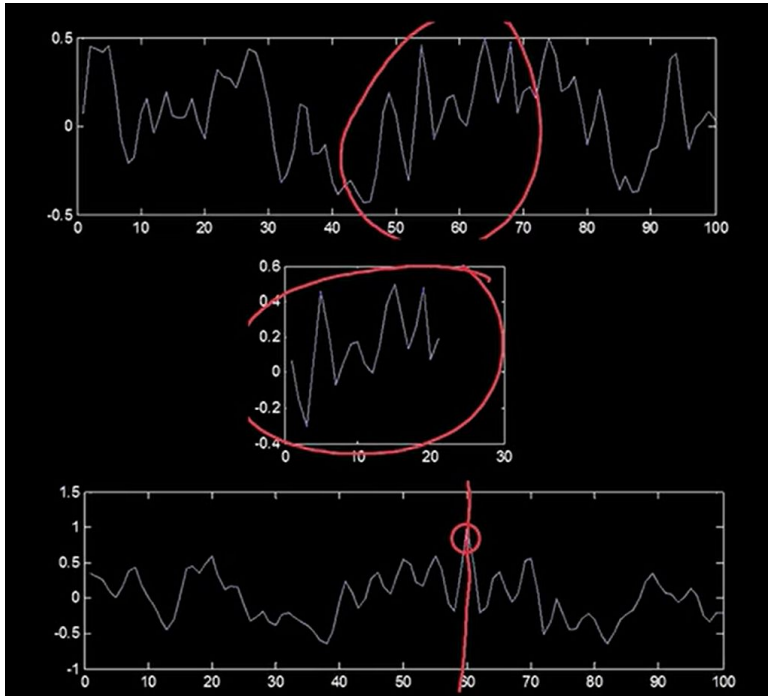




Conv1D

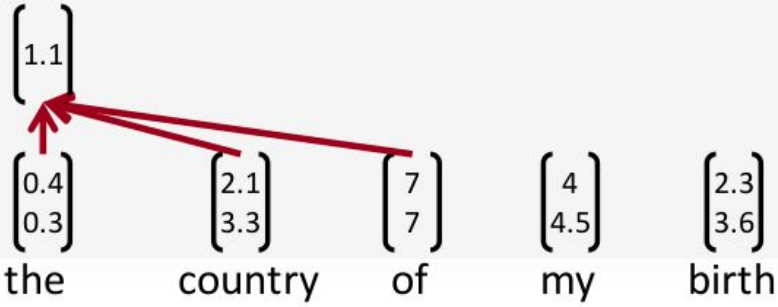

layer m+1
layer m
layer m-1


feature map
layer m
layer m-1


0.5
0
-0.5
0 10 20 30 40 50 60 70 80 90 100
0.6
0.4
0.2
0
-0.2
-0.4
0 10 20 30
1.5
1
0.5
0
-0.5
-1
0 10 20 30 40 50 60 70 80 90 100

Single Layer CNN

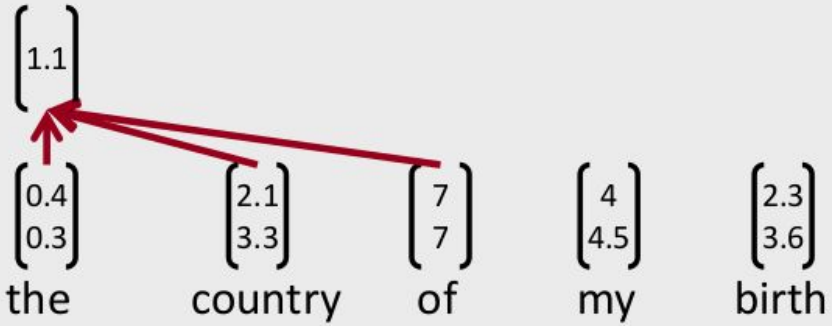
- A simple variant using one convolutional layer and **pooling**
- Based on Collobert and Weston (2011) and Kim (2014) "Convolutional Neural Networks for Sentence Classification"
- Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$ (vectors concatenated)
- Concatenation of words in range: $\mathbf{x}_{i:i+j}$
- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$ (goes over window of h words)
- Could be 2 (as before) higher, e.g. 3:



Single layer CNN

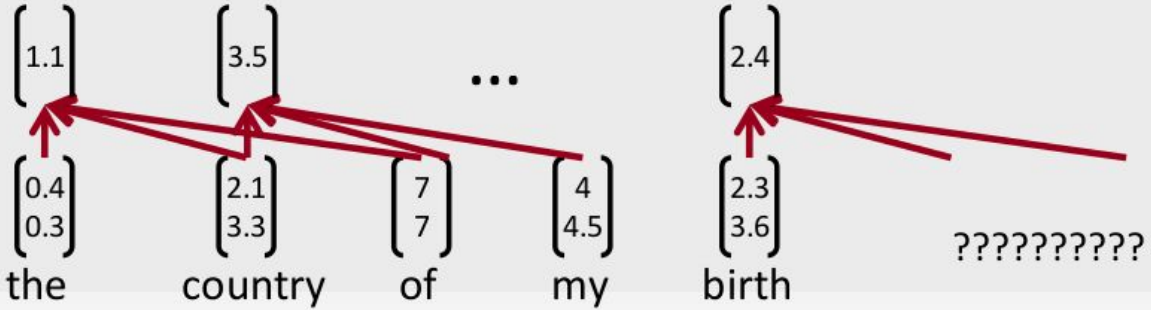
- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$ (goes over window of h words)
- Note, filter is vector!
- Window size h could be 2 (as before) or higher, e.g. 3:
- To compute feature for CNN layer:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



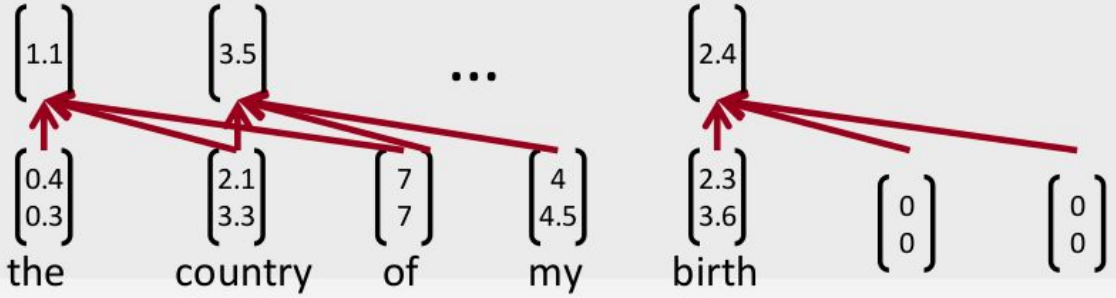
Single layer CNN

- Filter w is applied to all possible windows (concatenated vectors)
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length h: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Single layer CNN

- Filter w is applied to all possible windows (concatenated vectors)
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length h: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Single layer CNN: Pooling layer

- New building block: Pooling
- In particular: max-over-time pooling layer
- Idea: capture most important activation (maximum over time)
- From feature map $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooled single number: $\hat{c} = \max\{\mathbf{c}\}$
- But we want more features!

Solution: Multiple filters

- Use multiple filter weights w
- Useful to have different window sizes h
- Because of max pooling $\hat{c} = \max\{\mathbf{c}\}$, length of **c** irrelevant

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

- So we can have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.

Multi-channel idea

- Initialize with pre-trained word vectors (word2vec or Glove)
- Start with two copies
- Backprop into only one set, keep other “static”
- Both channels are added to c_i before max-pooling

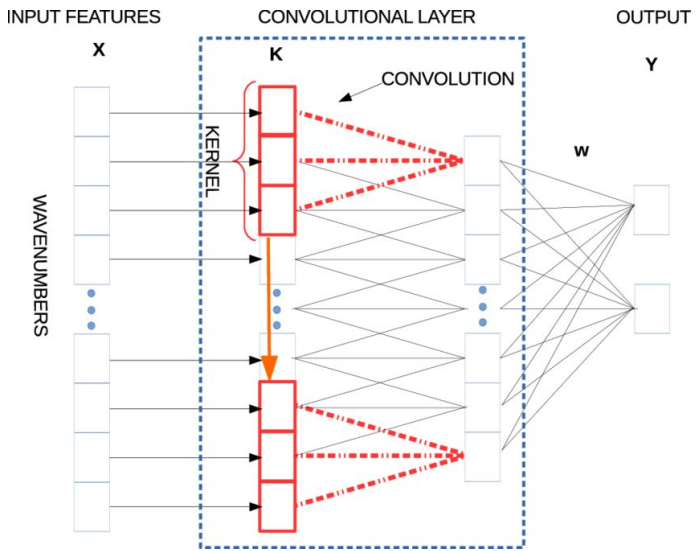
Conv2D

- Conv2D → (N, OUT_CH (n_kernels), L, W, C) → 5D tensor
- Concatenation model:
  - If we concatenate([t0,t1,..tT],axis=-1): t0=(N,L,W,3), t1=(N,L,W,3)...tT=(N,L,W,3) → x = (N,L,W,3xT)
  - Next layers will treat the concatenated frames as just channels
- Spatio-temporal:
  - Spatial model:
    - Spatio model = Conv2D(out_maps, kernel)(ti=(N,L,W,3)) → (N,L,W,out_maps)
    - If we concatenate([t0,t1,..tT],axis=-1): t0=(N,L,W,out_maps), t1=(N,L,W,out_maps)...tT=(N,L,W,out_maps) → x = (N,L,W,out_mapsxT)
  - Temporal model:
    - Conv2D(final_maps, kernel)(x => (N,L,W,out_mapsxT)) → (N,L,W,final_maps)
    - Flatten
    - Classification → Softmax

Conv1D

Works same as Conv2D, except the input is (N, OUT_CH (n_kernels), T, C=Features) → 4D tensor

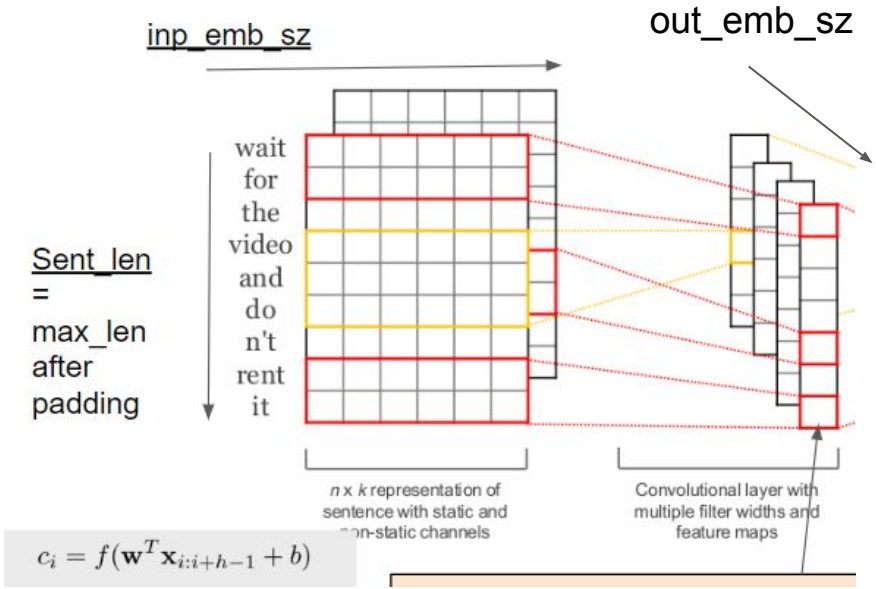
- T is the samples (scalars in time)=#words for example
- C (channels) is the dimension of feature vector at time t
- OUT_CH is the output dimension of the mapped vector (**out_emb**)
- Exampe C=1 feature per time step, OUT_CH=1
- Conv1D(1, kernel=3)(x=>(1,F)--> y=>(1,F-3+1)
- If we have more than 1 frame, concatenate in T not channels



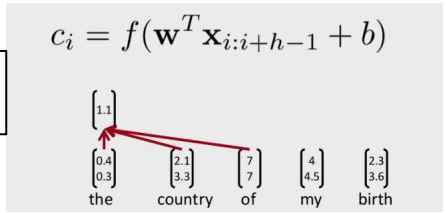
Multi-channel convolution illustration

Conv1D: (kernel, out_channels, inp_channels, sent_len):

- Inp_ch = inp_emb_sz
- Out_ch = out_emb_sz
- Each output kernel will operate on the input (sent_lenxemb_sz) ⇒ 2D (N-M+1 x emb_sz)
- we treat inp_emb_sz as input channels and sum ⇒ learnable with W⇒ **2D-1D**
- we we concat on the channel dim for each 1D output filter ⇒ **2D**



Crush the row dim (emb_sz) using learnable sum

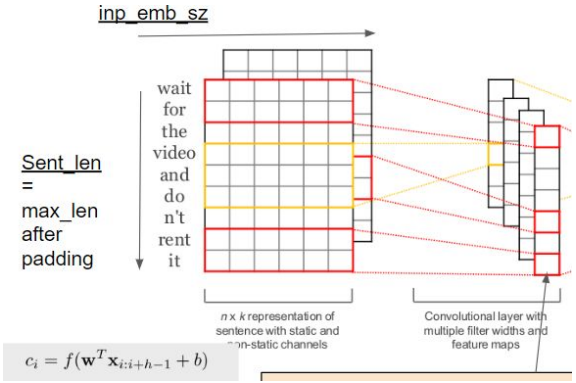


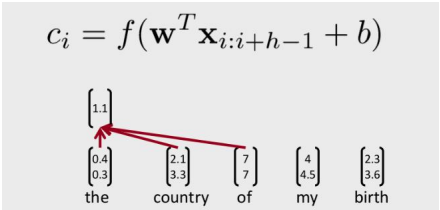
Conv1D example

Input seq max_len = 400 words
Each 50 emb_dim

Conv over 400 with kernel=3→ 398 token
Each should have 50 dim
BUT, we summarize them with w → so each is 1 scalar → 398x1
We have 250 output kernels (each 3) → so we have 398x250 outputs

embedding_2 (Embedding)	(None, 400, 50)	250000
dropout_3 (Dropout)	(None, 400, 50)	0
conv1d_2 (Conv1D)	(None, 398, 250)	37750
max_pooling1d_2 (MaxPooling1	(None, 199, 250)	0



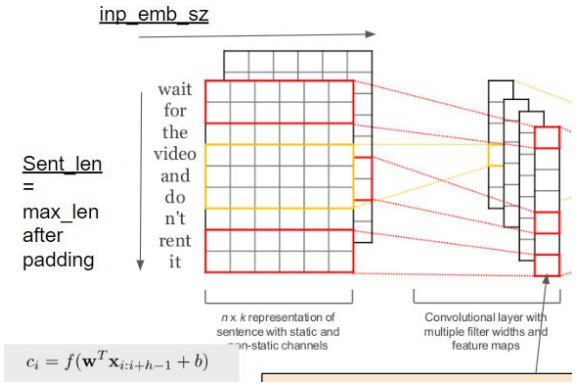


Multi-channel convolution illustration: Conv2D vs Conv1D

In Conv2D: we sum over input feature maps ⇒ **3D-2D**
In Conv1D: we treat emb_sz as input channels and sum ⇒ learnable with W⇒ **2D-1D**

Following:
In Conv2D: we concat on the channel dim for each output 2D filter ⇒ 3D

In Conv1D: we we concat on the channel dim for each 1D output filter ⇒ 2D



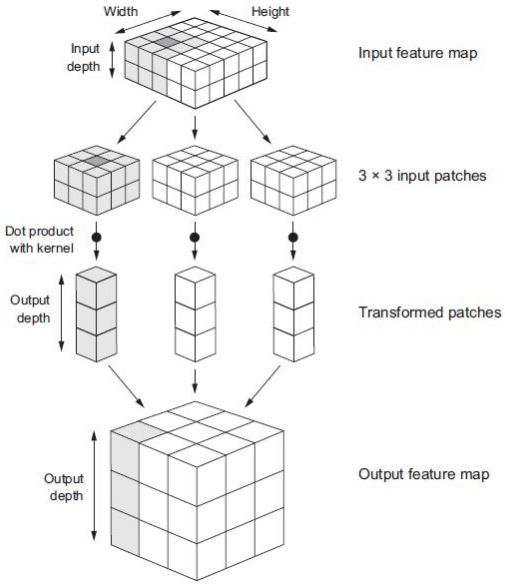


Figure from Kim (2014)

Conv1D + GlobalMaxPool1D example

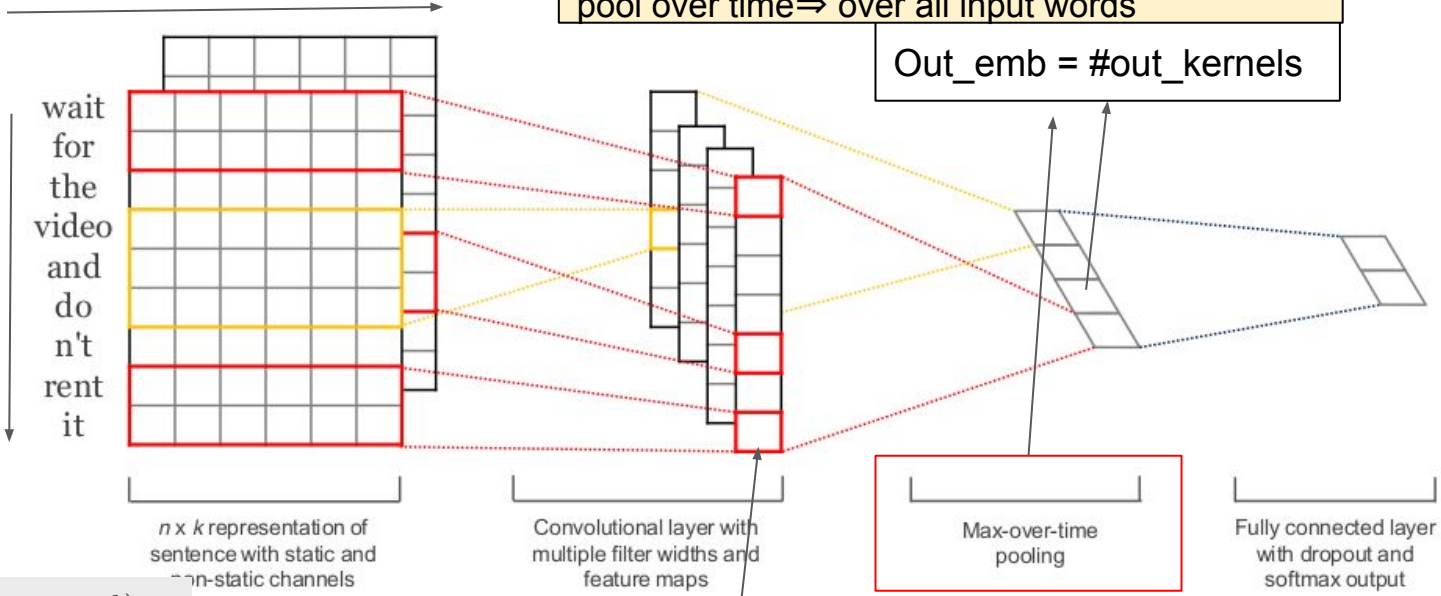
How to get 1D vector per sentence?

Every kernel conv gives one output due to max pool over time⇒ over all input words

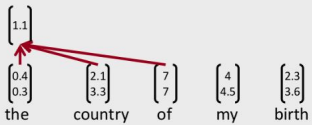
Out_emb = #out_kernels

inp_emb_sz

Sent_len = max_len after padding



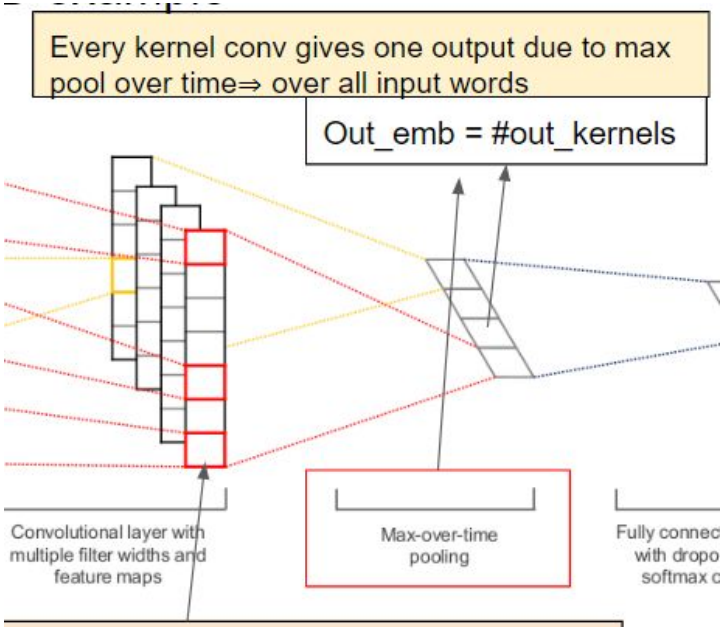
$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



For a filter operating on input sequence each of emb_sz, we concat them, and summarize with eight vector, for every kernel **slide.**, so one local dot = produces 1 scalar

n words (possibly zero padded) and each word vector has k dimensions

Conv1D + GlobalMaxPool1D example



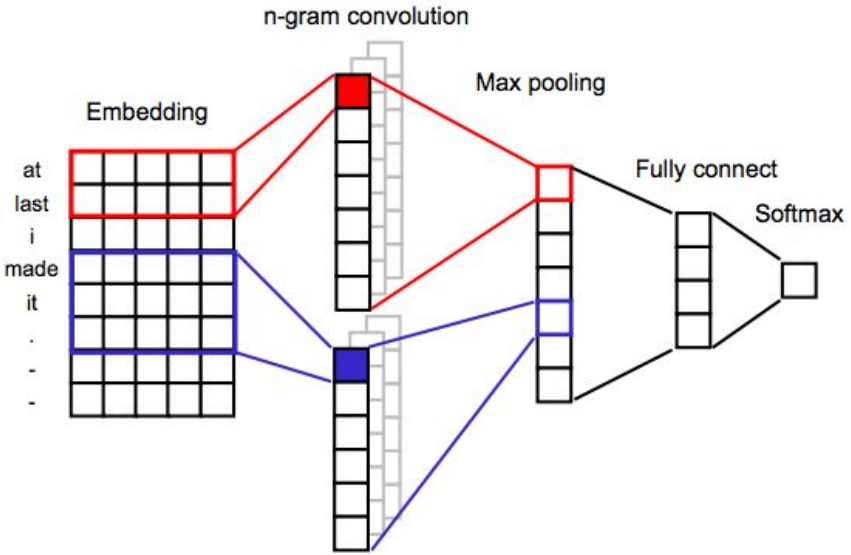
Layer (type)	Output Shape
embedding_3 (Embedding)	(None, 400, 50)
dropout_5 (Dropout)	(None, 400, 50)
conv1d_3 (Conv1D)	(None, 398, 250)
global_max_pooling1d_1 (Glob	(None, 250)

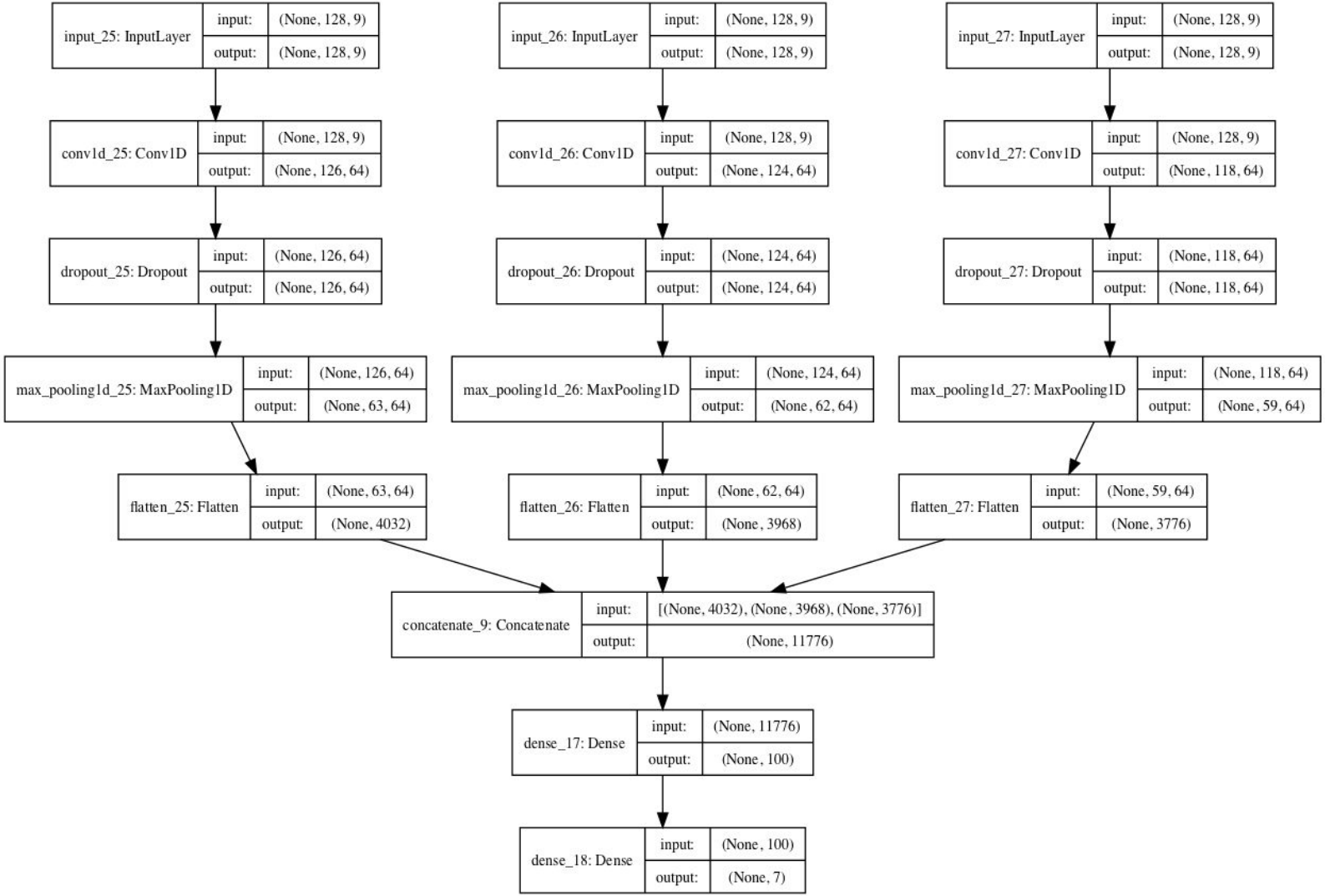
Multi-headed CNN

But what if we want different kernels ⇒ n-grams (2-3-...etc)?

Another popular approach with CNNs is to have a multi-headed model, where each head of the model reads the input time steps using a different sized kernel.

For example, a three-headed model may have three different kernel sizes of 3, 5, 11, allowing the model to read and interpret the sequence data at three different resolutions. The interpretations from all three heads are then concatenated within the model and interpreted by a fully-connected layer before a prediction is made.



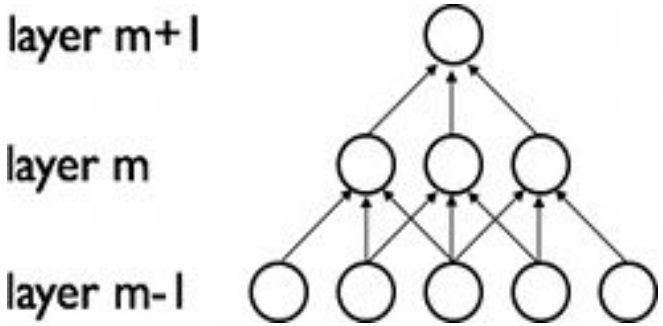

input_25: InputLayer
input: (None, 128, 9)
output: (None, 128, 9)
input_26: InputLayer
input: (None, 128, 9)
output: (None, 128, 9)
input_27: InputLayer
input: (None, 128, 9)
output: (None, 128, 9)
conv1d_25: Conv1D
input: (None, 128, 9)
output: (None, 126, 64)
conv1d_26: Conv1D
input: (None, 128, 9)
output: (None, 124, 64)
conv1d_27: Conv1D
input: (None, 128, 9)
output: (None, 118, 64)
dropout_25: Dropout
input: (None, 126, 64)
output: (None, 126, 64)
dropout_26: Dropout
input: (None, 124, 64)
output: (None, 124, 64)
dropout_27: Dropout
input: (None, 118, 64)
output: (None, 118, 64)
max_pooling1d_25: MaxPooling1D
input: (None, 126, 64)
output: (None, 63, 64)
max_pooling1d_26: MaxPooling1D
input: (None, 124, 64)
output: (None, 62, 64)
max_pooling1d_27: MaxPooling1D
input: (None, 118, 64)
output: (None, 59, 64)
flatten_25: Flatten
input: (None, 63, 64)
output: (None, 4032)
flatten_26: Flatten
input: (None, 62, 64)
output: (None, 3968)
flatten_27: Flatten
input: (None, 59, 64)
output: (None, 3776)
concatenate_9: Concatenate
input: [(None, 4032), (None, 3968), (None, 3776)]
output: (None, 11776)
dense_17: Dense
input: (None, 11776)
output: (None, 100)
dense_18: Dense
input: (None, 100)
output: (None, 7)

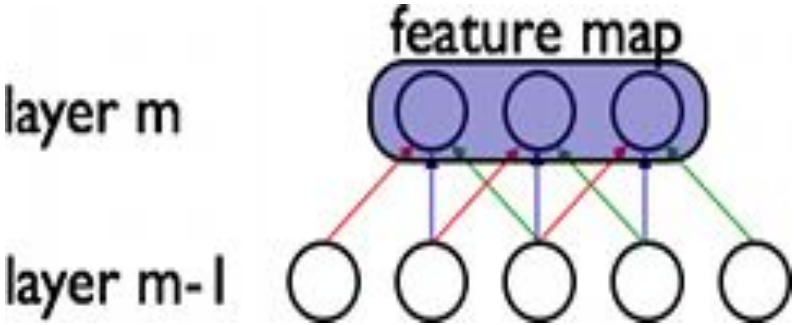
Classification after one CNN layer

- First one convolution, followed by one max-pooling
- To obtain final feature vector: $\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$ (assuming m filters w)
- Simple final softmax layer $y = softmax\left(W^{(S)}z + b\right)$

Disadv: Depth scales with seq length!

- ConvNet captures sequence information and summarizes it in a **state**
- Use that state for (condition output on that state):
  - Classification
  - Sequence generation
- Network depth scales with max seq (sentence) length!
  - sequence length requires deeper models, it makes it difficult to learn dependencies between distant words
- Inefficient for variable length! Needs padding





Let’s code

Conv1D-IMDB Keras dataset

https://colab.research.google.com/drive/1zazDWpg6JeWUe8lxXp4dY61k5QzfuKMA?usp=sharing

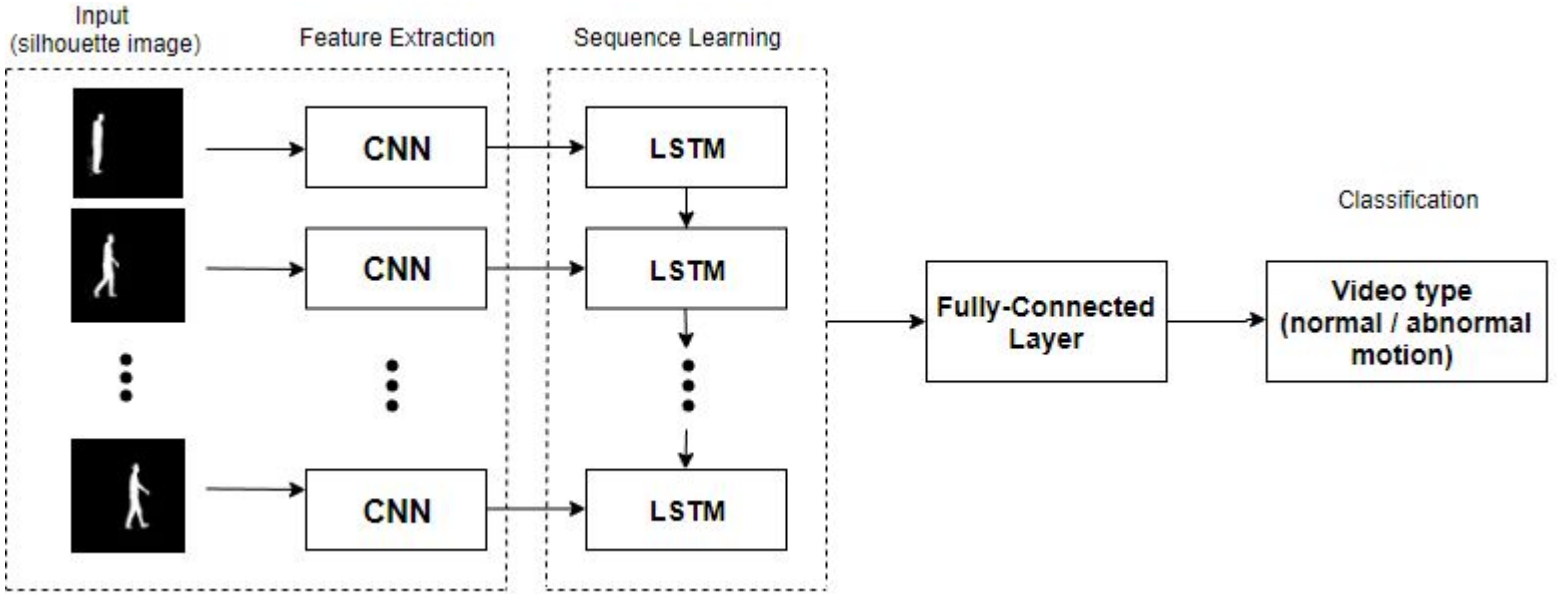
Conv1D-IMDB Raw dataset

https://colab.research.google.com/drive/17iOxgMW-MT36RCLm7N31smxylk9buAGK?usp=sharing

CNN-LSTM

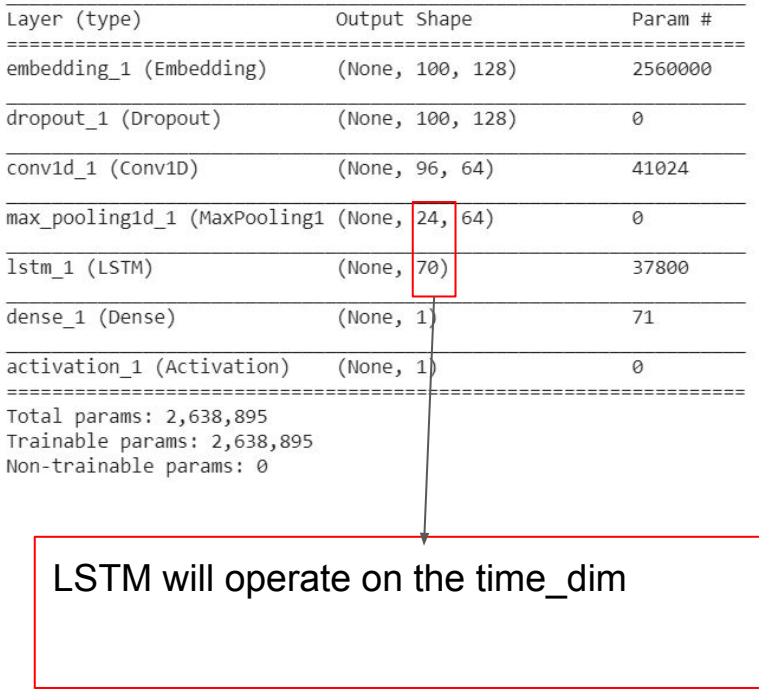
Spatio-temporal

CNN-LSTM: internal hidden state flattened as normal vector dot



CNN-LSTM - Just take the raw CNN output without flatten

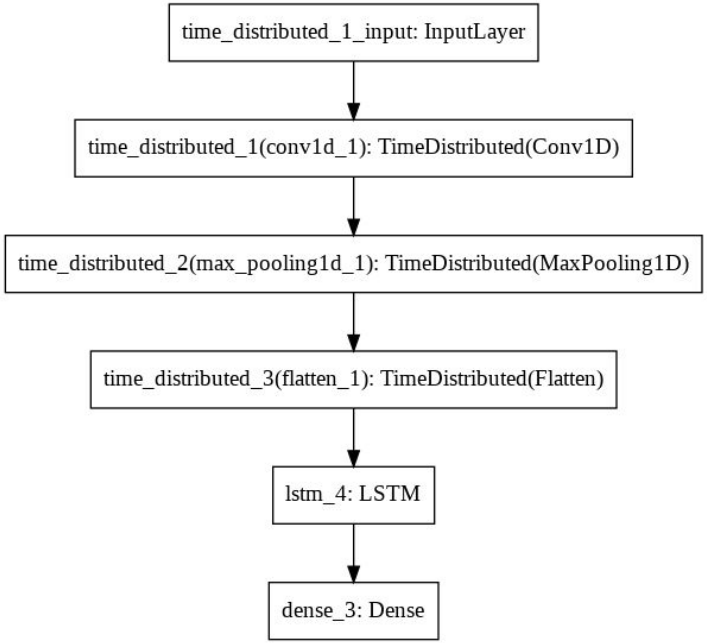
```
model = Sequential()
model.add(Embedding(max_features, embedding_size, input_length=maxlen))
model.add(Dropout(0.25))
model.add(Conv1D(filters,
                 kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1))
model.add(MaxPooling1D(pool_size=pool_size))
model.add(LSTM(lstm_output_size))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```



CNN-LSTM - TimeDistributed (Like VideoCNN)

```
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), inp
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
```

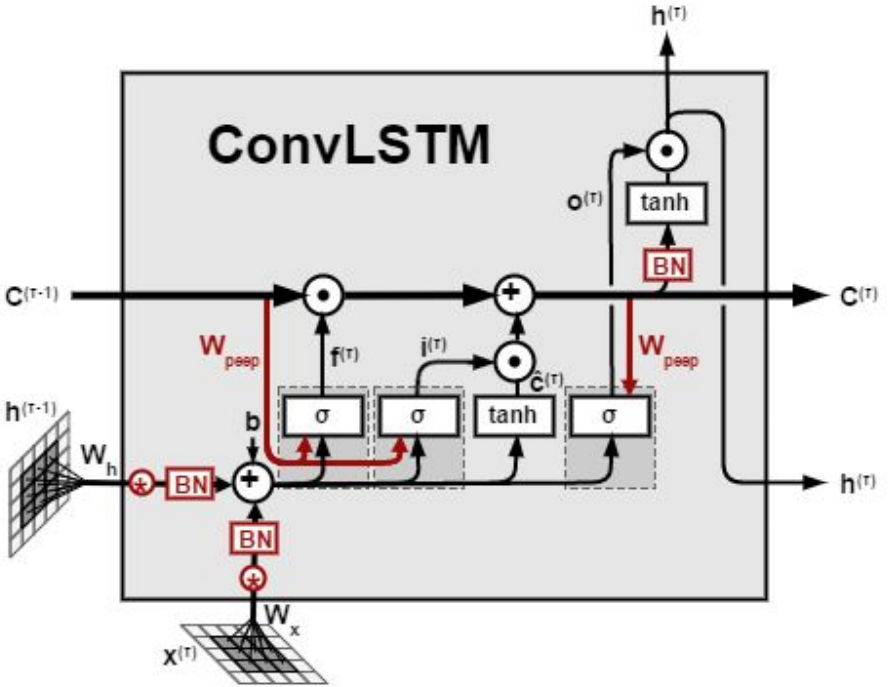
```
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
```



Spatio-temporal

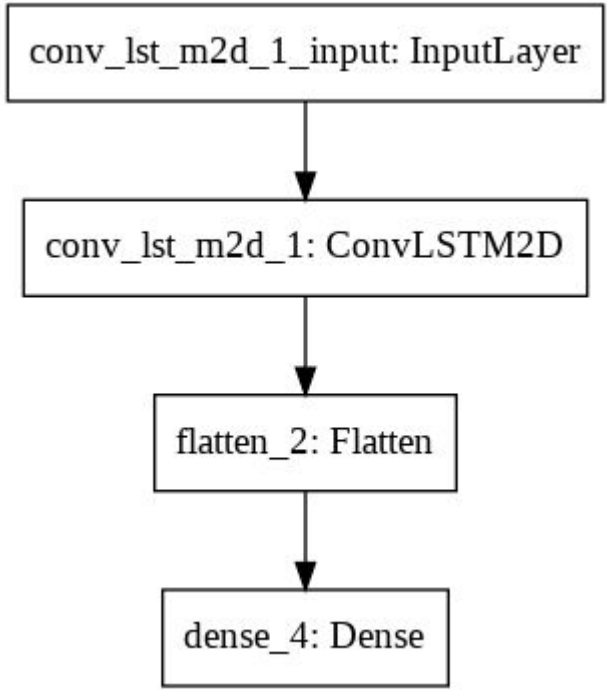
ConvLSTM

Internal hidden state a convolution



ConvLSTM - No TimeDistributed (states already temporal)

```
model.add(ConvLSTM2D(filters=64, kernel_size=(1,2), activation='relu', input_shape=
model.add(Flatten())
```



Built-in
Spatio-temporal

Let’s code

CNN-LSTM-IMDB Keras dataset

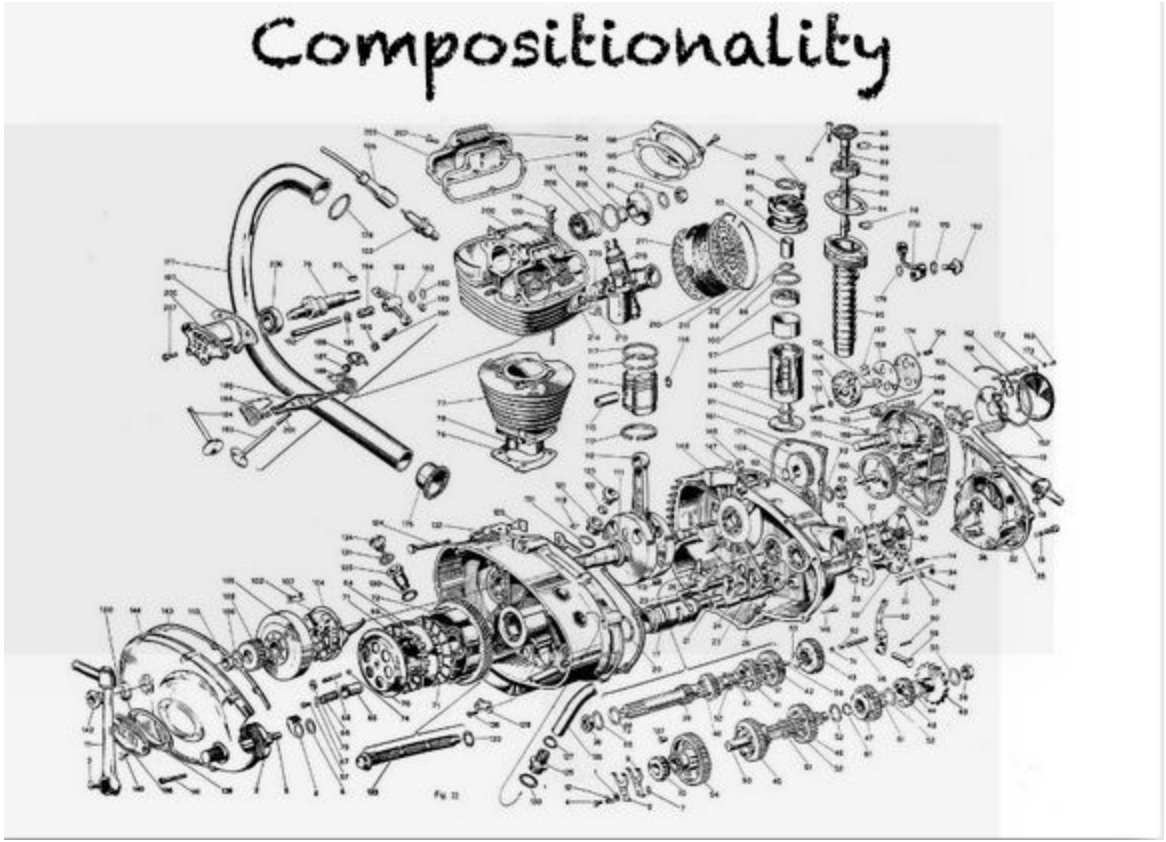
https://colab.research.google.com/drive/1zazDWpg6JeWUe8lxXp4dY61k5QzfuKMA?usp=sharing

CNN-LSTM-IMDB Raw dataset

https://colab.research.google.com/drive/17iOxgMW-MT36RCLm7N31smxylk9buAGK?usp=sharing

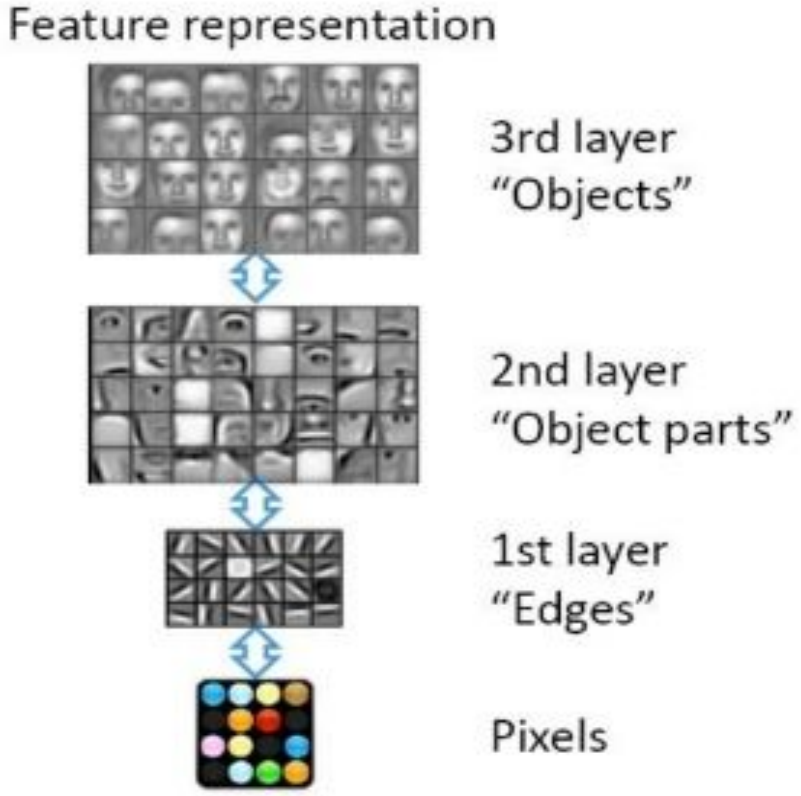
Recursive Auto Encoders (optional)

Compositionality



Compositionality modeling

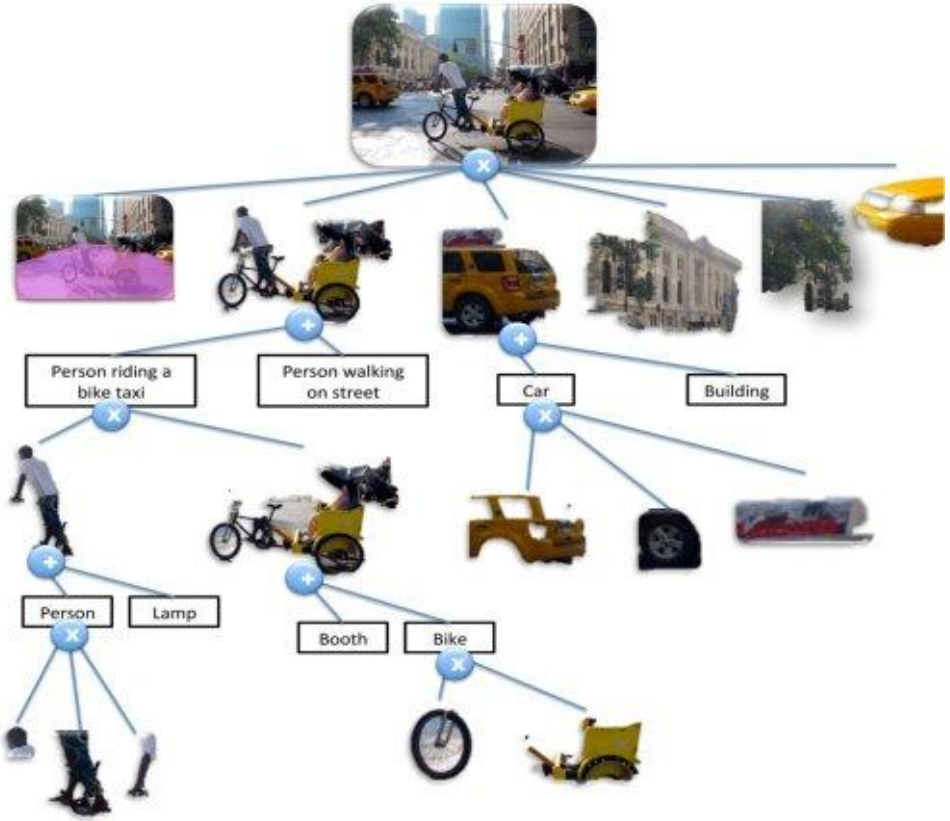
- Spatial
  - Image
    - CNN
  - Distributed representations
  - No control
  - No prior templates for context. Ex: parse trees
  - The only template is the 2D map spatial relation: every pixel is affected by its neighbors



http://www.slideshare.net/roelofp/2014-1021-sicsdlnlpg

Compositionality modeling

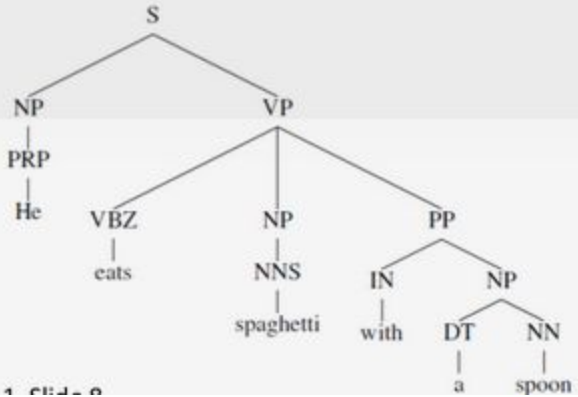
- Hierarchical
  - Recursive
  - RNN
  - RNTN
  - LTSM NN
    - Prior templates for context can be imp

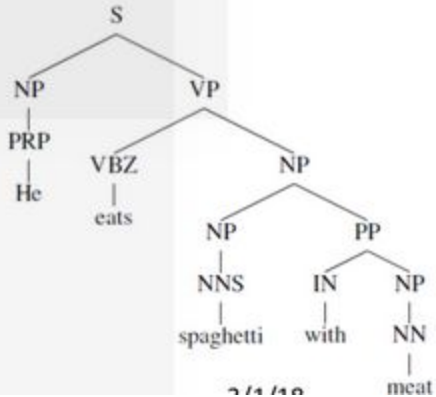


http://www.kdnuggets.com/2014/08/interview-pedro-domingos-master-algorithm-new-deep-learning.html

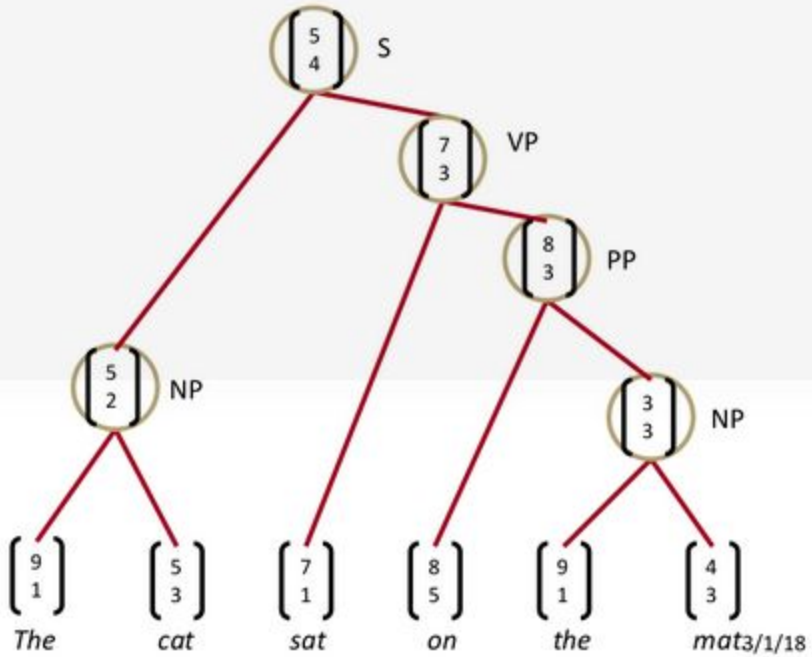
Are languages recursive?

- Cognitively somewhat debatable
- But: recursion is natural for describing language
- *[The man from [the company that you spoke with about [the project] yesterday]]*
- noun phrase containing a noun phrase containing a noun phrase
- Arguments for now: 1) Helpful in disambiguation:





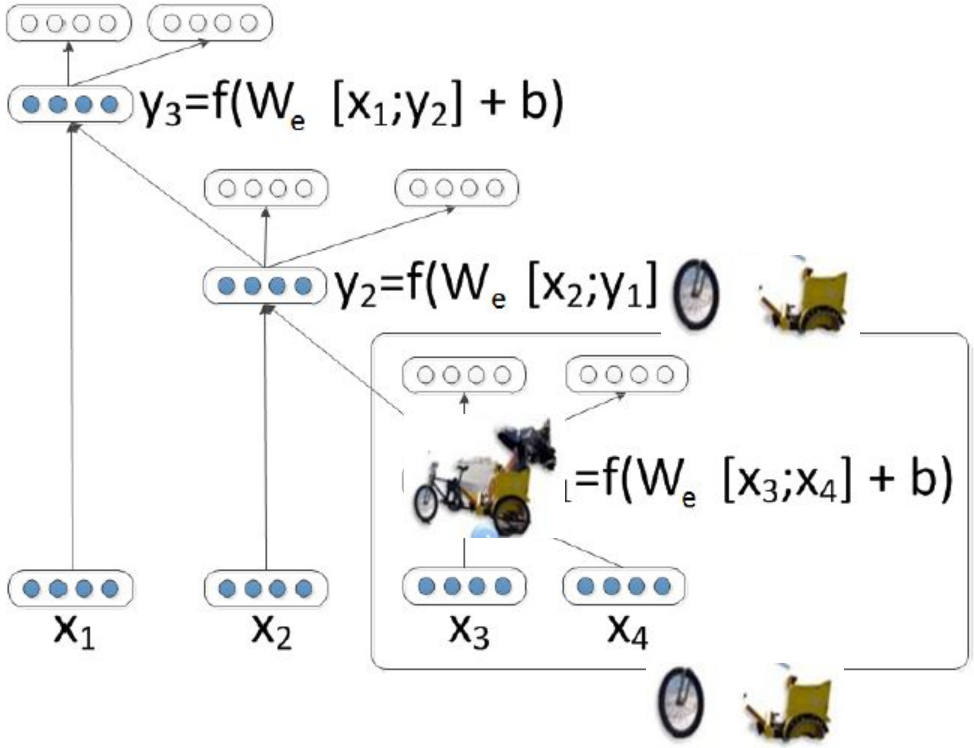
Learn Structure and Representation



How to get the parsing order?

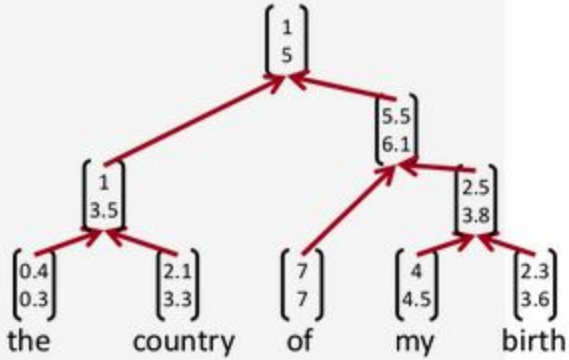
- Parser
  - Stanford
  - Expensive Language resource (Low-Language-Resources (LLR) = Arabic)
- RAE
  - Unsupervised
  - Sub-optimal
  - Expensive computations

Recursive Auto Encoder

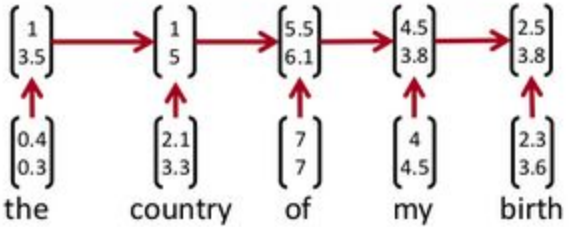


Recursive vs. recurrent neural networks

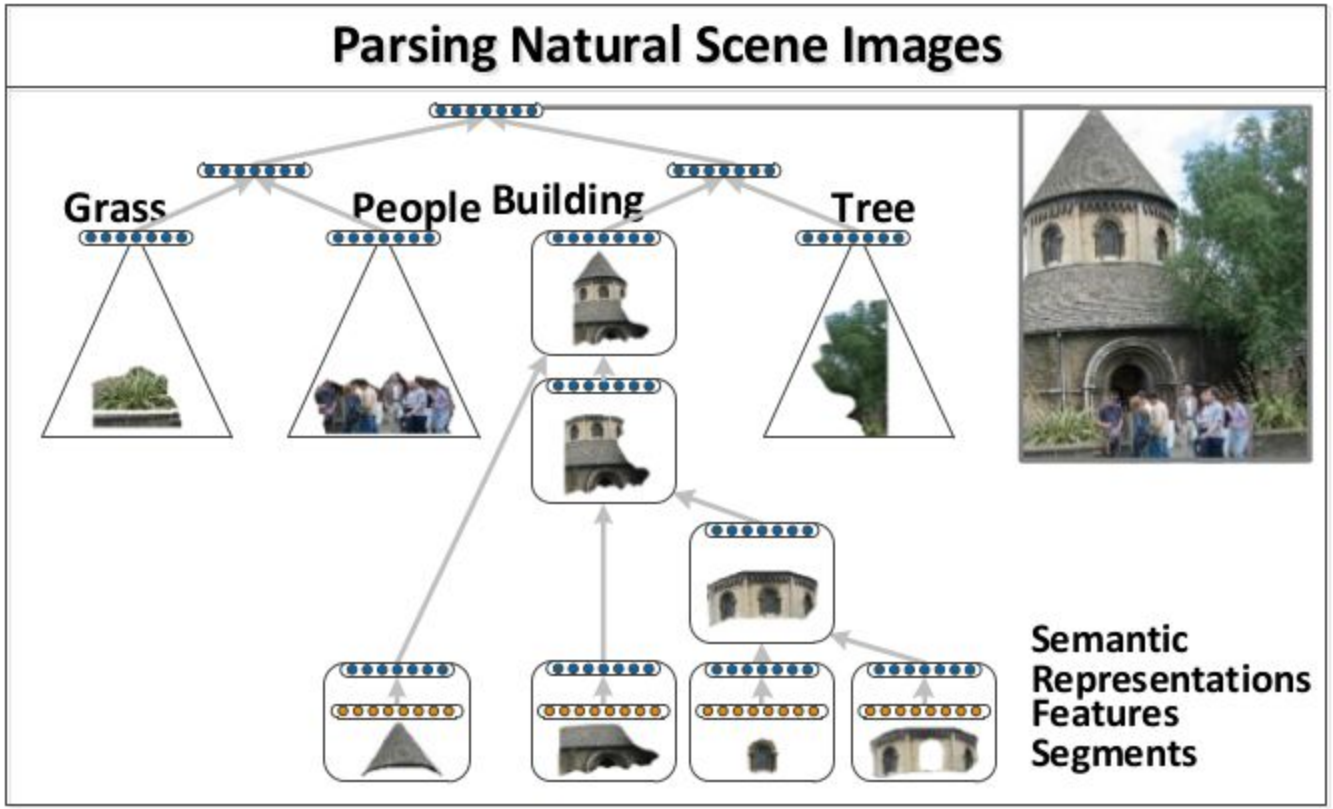
- Recursive neural nets require a tree structure



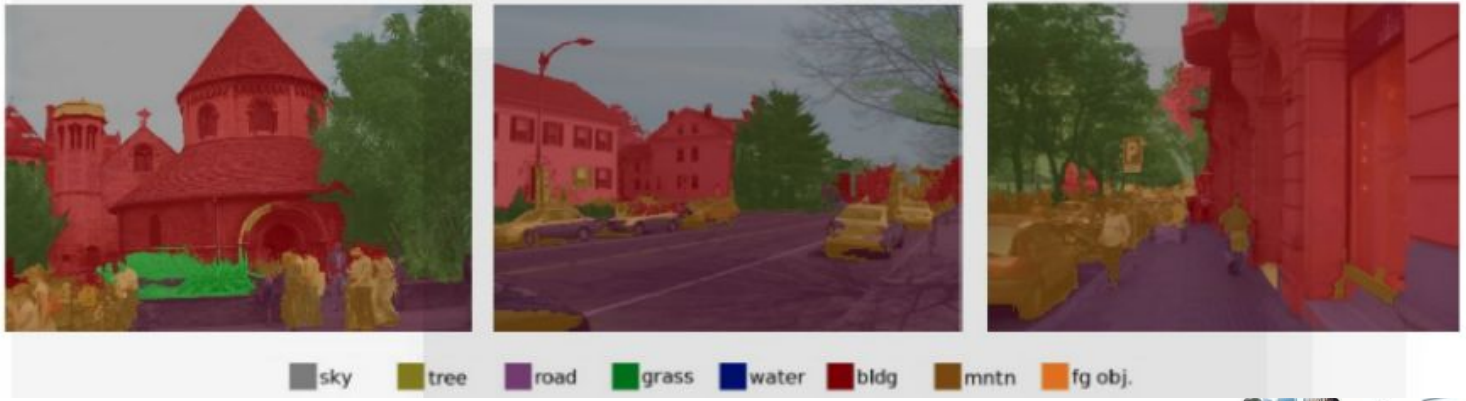
- Recurrent neural nets cannot capture phrases without prefix context and often capture too much of last words in final vector

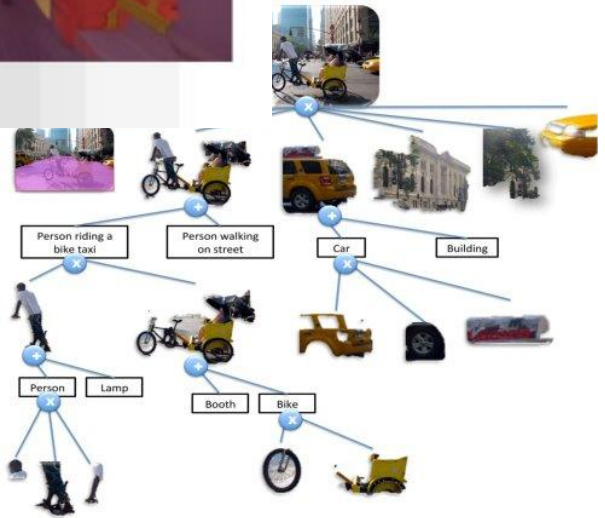


Same Recursive Neural Network as for natural language parsing!
(Socher et al. ICML 2011)



Multi-class segmentation





Recursive models

Adv

- Parsing language structure

Disadv

- Require parse trees
- Expensive computations