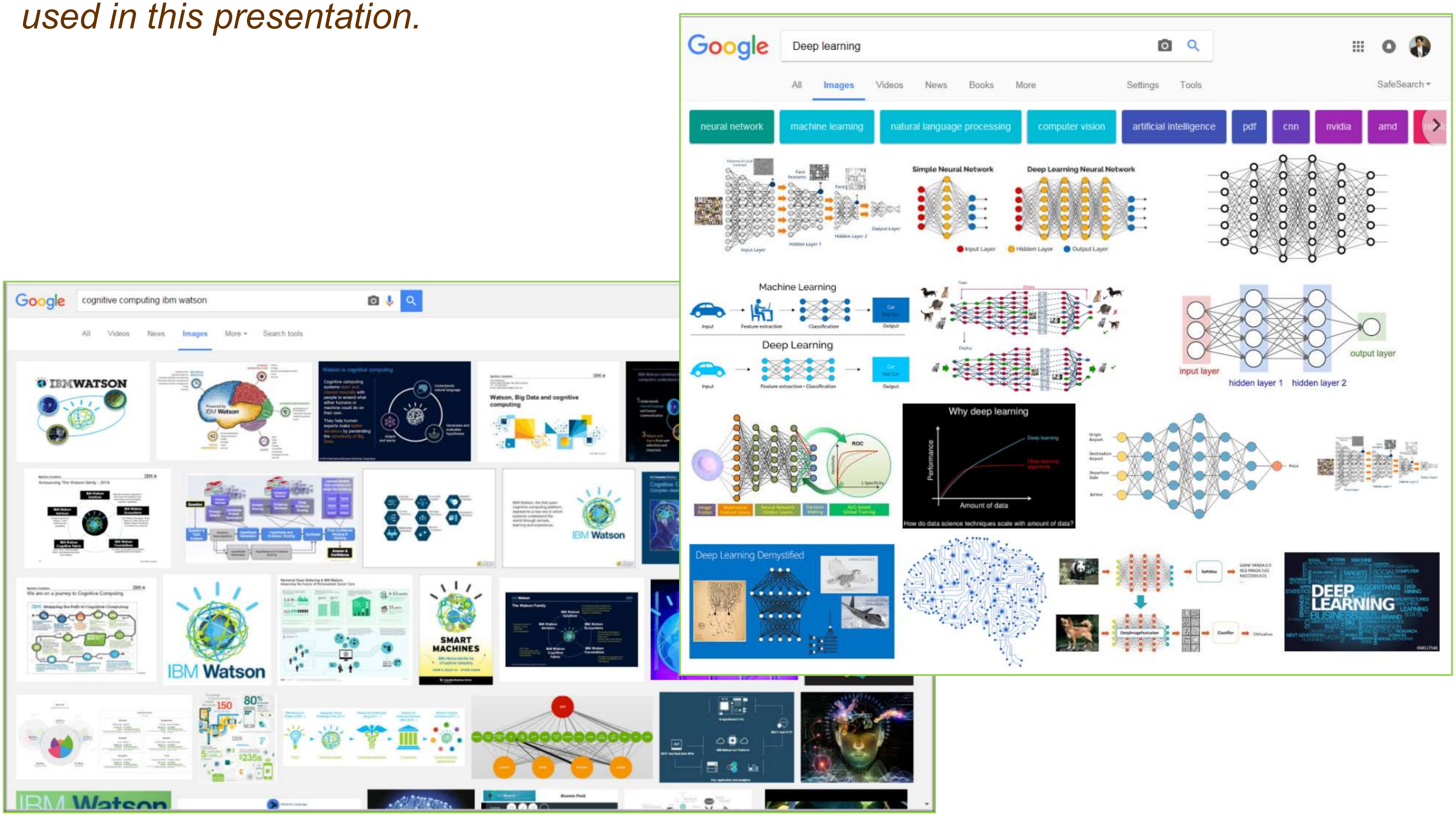
Lecture 7: Recurrent Neural Networks (RNNs) and Transformers

TIES4911 Deep-Learning for Cognitive Computing for Developers
Spring 2024

by:
Dr. Oleksiy Khriyenko
IT Faculty
University of Jyväskylä

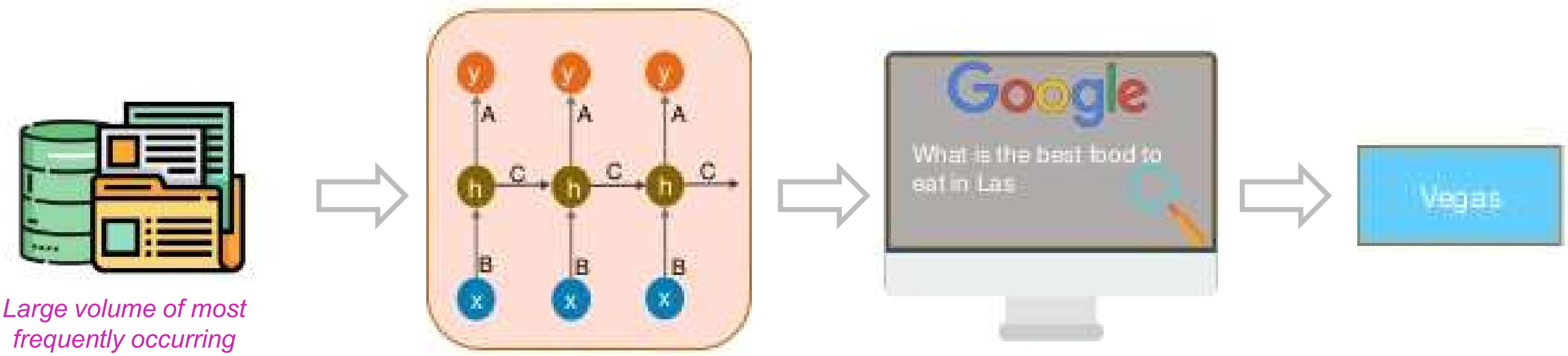
Acknowledgement

I am grateful to all the creators/owners of the images that I found from Google and have used in this presentation.



Autocomplete feature

Google's autocomplete feature predicts the rest of the words a user is typing…



Large volume of most frequently occurring consecutive words

Recurrent Neural Network (RNN)

Google Search

Auto-completion

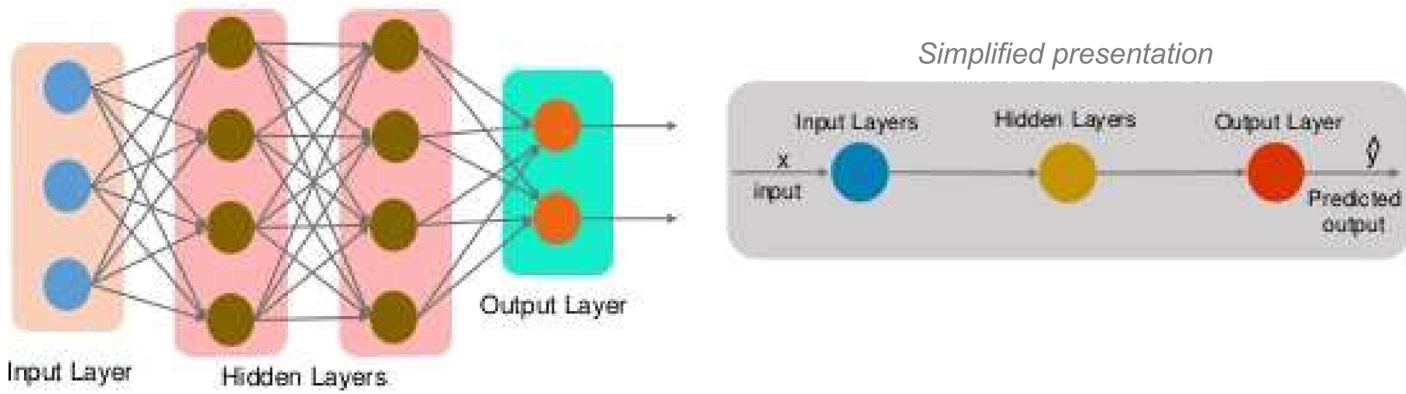
Analyses the data by finding the sequence of words occurring frequently and builds a model to predict the next word in the sentence

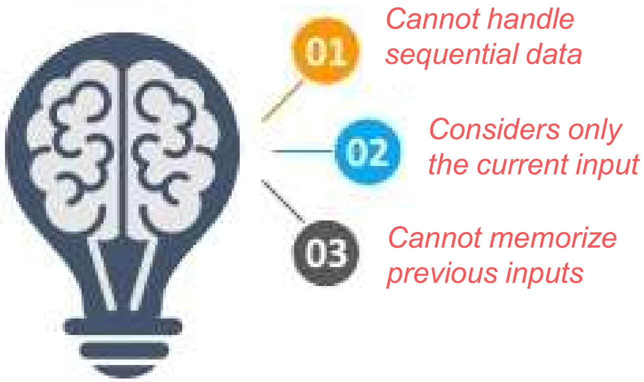
Relevant links:
https://www.slideshare.net/Simplilearn/recurrent-neural-network-rnn-tutorial-rnn-lstm-tutorial-deep-learning-tutorial-simplilearn
https://www.youtube.com/watch?v=IWkFhVq9-nc

Why RNNs?..

In a Feed-Forward Neural Network:

- *information flows in forward direction from input to output through the hidden layers (if any)*
- *decisions are based on current input with no memory about the past and future scope*

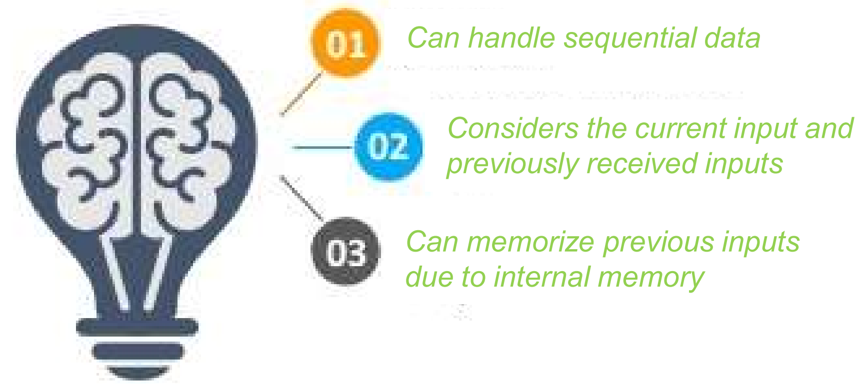


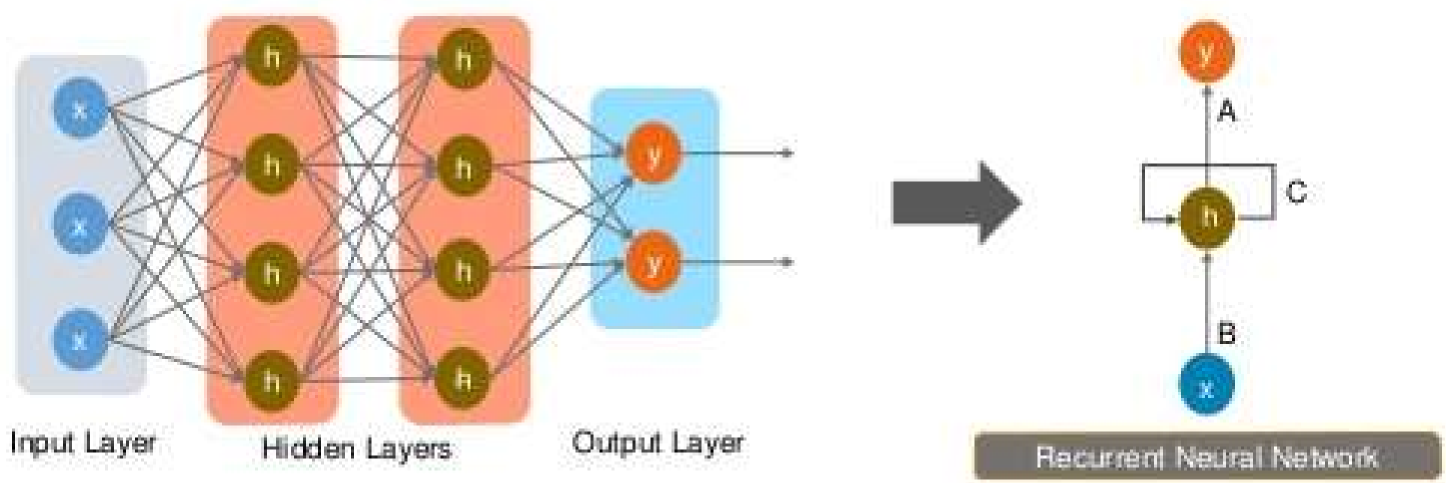


Relevant links:
https://www.slideshare.net/Simplilearn/recurrent-neural-network-rnn-tutorial-rnn-lstm-tutorial-deep-learning-tutorial-simplilearn
https://www.youtube.com/watch?v=lWkFhVq9-nc

Why RNNs?..

Recurrent Neural Network handles sequential data. Introducing a loop in the hidden layer(s), RNN saves the output of a layer and feeds this back to the input in order to predict the next one...





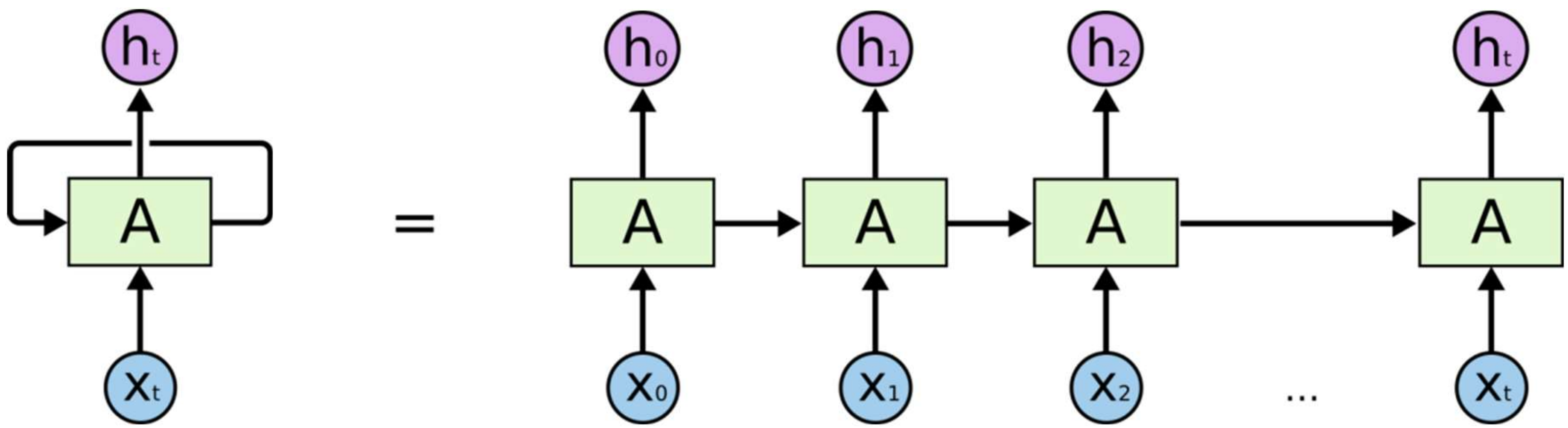
Relevant links:
https://www.slideshare.net/Simplilearn/recurrent-neural-network-rnn-tutorial-rnn-lstm-tutorial-deep-learning-tutorial-simplilearn
https://www.youtube.com/watch?v=lWkFhVq9-nc

RNNs

Sequence Modeling Design Criteria:

- *Handle variable-length sequences*
- *Track long-term dependencies*
- *Maintain information about order*
- *Share parameters across the sequence*

RNNs have loops allowing information to persist:

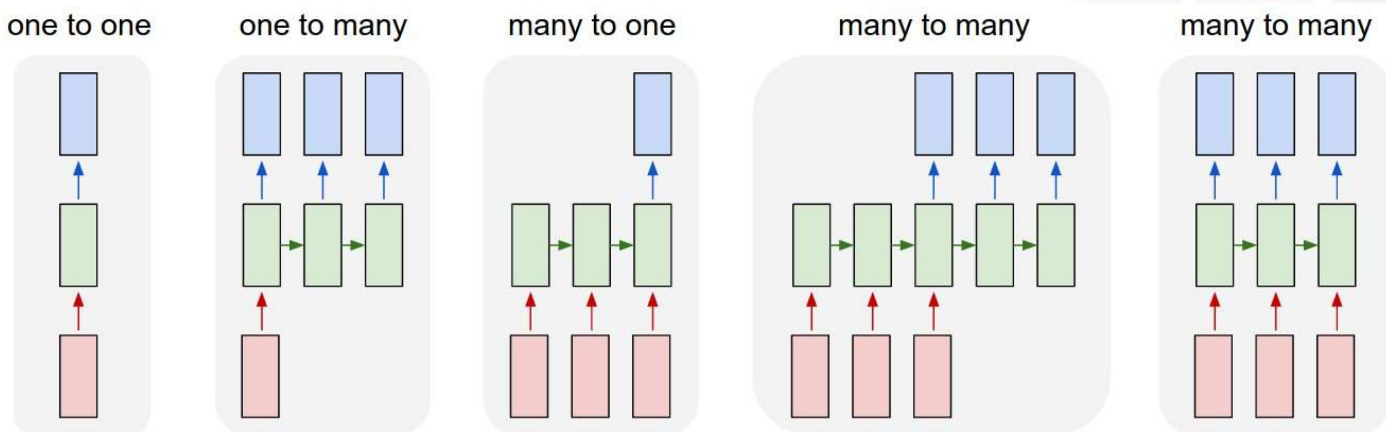


RNNs

Recurrent Neural Networks (RNNs) *is a neural network that is used when you deal with sequential data, where the particular order of the data-points matter (e.g. predict event that is happening at every point in a movie based on its previous events).*

Variety of problems related to sequential data:

- *speech recognition*
- *language modeling, text and code generation*
- *(multilingual) machine translation*
- *handwriting generation*
- *question answering*
- *time series prediction (e.g. stock market trend prediction)*
- *image captioning*
- *control of autonomous vehicles and robots*
- *…*



Vanilla mode RNN, e.g. image classification

e.g. image captioning

e.g. sentiment classification, video based event detection

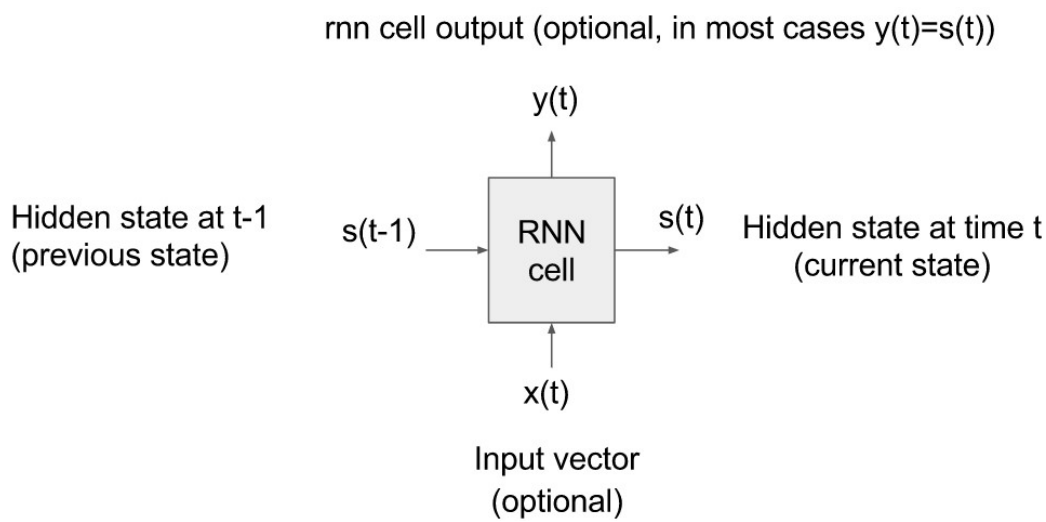
e.g. machine translation

e.g. language model (next word prediction), video frames classification

Relevant links:
http://karpathy.github.io/2015/05/21/rnn-effectiveness
https://www.youtube.com/watch?v=UNmqTiOnRfg

RNNs

RNN Cell…

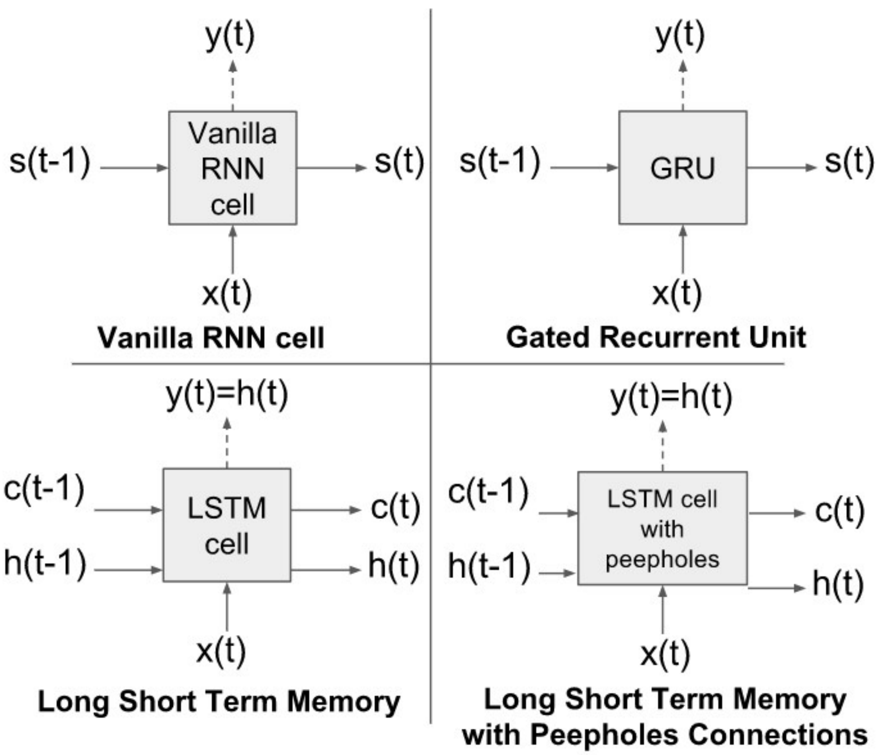


Relevant links:
https://www.youtube.com/watch?v=UNmqTiOnRfg
https://deepsystems.ai/

RNNs

The most common ***RNN Cells****…*

- *Vanilla*
- *Gated Recurrent Units (GRN)*
- *Long Short Term Memory (LSTM)*
- *LSTM with Peepholes Connections*



Vanilla RNN

Late 1980s - backpropagation through time to train Vanilla RNN

LSTM

1997 - Long Short-Term Memory (S.Hochreiter, J.Schmidhuber)

LSTM with Peepholes

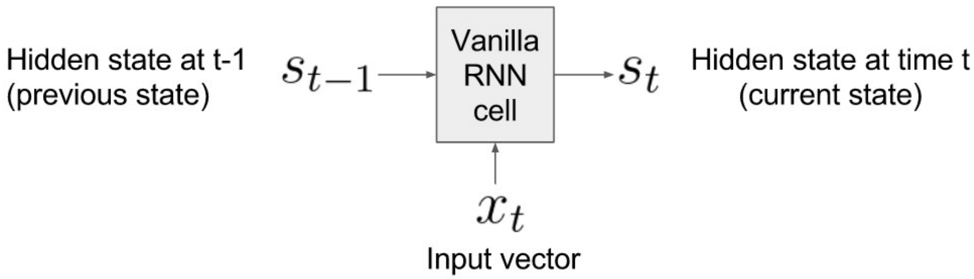
2000 - Recurrent nets that time and count (F.A. Gers ; J. Schmidhuber)

GRU

2014 - Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation (Kyunghyun Cho, Yoshua Bengio, and others)

Relevant links:
https://deepsystems.ai/

Vanilla RNN



$$\underset{(n\times 1)}{s_t} = \varphi(\underset{(n\times n)}{W}\underset{(n\times 1)}{s_{t-1}} + \underset{(n\times m)}{U}\underset{(m\times 1)}{x_t} + \underset{(n\times 1)}{b})$$

$n = 2$ (state size),
$m = 3$ (input size)

$$\begin{pmatrix} s_t^1 \\ s_t^2 \end{pmatrix} = \varphi(\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}\begin{pmatrix} s_{t-1}^1 \\ s_{t-1}^2 \end{pmatrix} + \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \end{pmatrix}\begin{pmatrix} x_t^1 \\ x_t^2 \\ x_t^3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix})$$

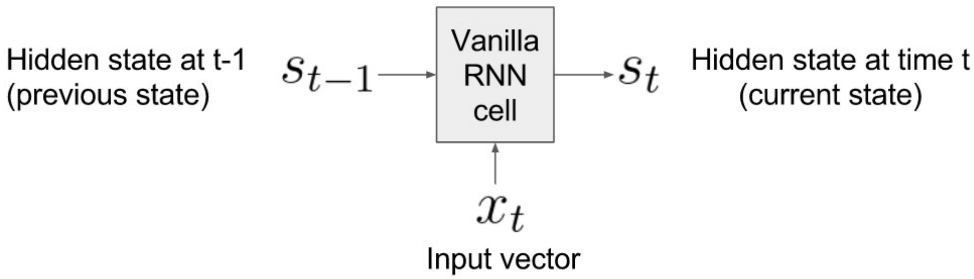
$$= \varphi(\begin{pmatrix} w_{11}s_{t-1}^1 + w_{12}s_{t-1}^2 \\ w_{21}s_{t-1}^1 + w_{22}s_{t-1}^2 \end{pmatrix} + \begin{pmatrix} u_{11}x_t^1 + u_{12}x_t^2 + u_{13}x_t^3 \\ u_{21}x_t^1 + u_{22}x_t^2 + u_{23}x_t^3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix})$$

$$= \varphi(\begin{pmatrix} w_{11}s_{t-1}^1 + w_{12}s_{t-1}^2 + u_{11}x_t^1 + u_{12}x_t^2 + u_{13}x_t^3 \\ w_{21}s_{t-1}^1 + w_{22}s_{t-1}^2 + u_{21}x_t^1 + u_{22}x_t^2 + u_{23}x_t^3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix})$$

$$= \varphi(\begin{pmatrix} w_{11} & w_{12} & u_{11} & u_{12} & u_{13} \\ w_{21} & w_{22} & u_{21} & u_{22} & u_{23} \end{pmatrix}\begin{pmatrix} s_{t-1}^1 \\ s_{t-1}^2 \\ x_t^1 \\ x_t^2 \\ x_t^3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix})$$ - single layer network

Relevant links:
https://deepsystems.ai/

RNNs



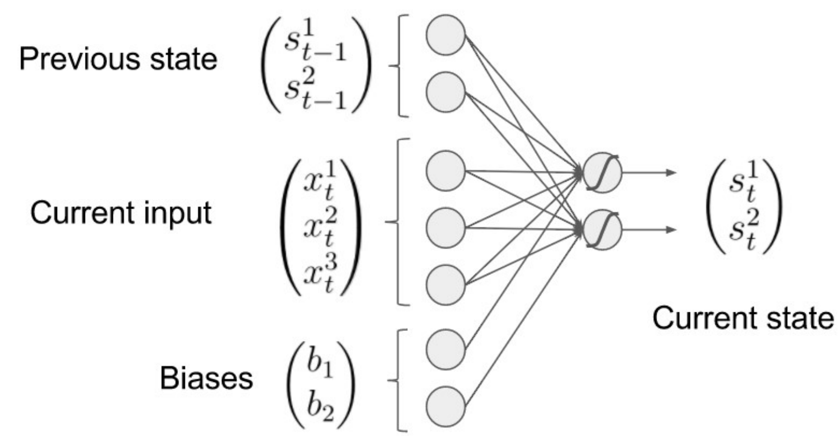
$$\begin{pmatrix} s_t^1 \\ s_t^2 \end{pmatrix} = \varphi\left(\begin{pmatrix} w_{11} & w_{12} & u_{11} & u_{12} & u_{13} \\ w_{21} & w_{22} & u_{21} & u_{22} & u_{23} \end{pmatrix} \begin{pmatrix} s_{t-1}^1 \\ s_{t-1}^2 \\ x_t^1 \\ x_t^2 \\ x_t^3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)$$

$$\begin{pmatrix} s_t^1 \\ s_t^2 \end{pmatrix} = \varphi\left(\begin{pmatrix} w_{11} & w_{12} & u_{11} & u_{12} & u_{13} & b_1 \\ w_{21} & w_{22} & u_{21} & u_{22} & u_{23} & b_2 \end{pmatrix} \begin{pmatrix} s_{t-1}^1 \\ s_{t-1}^2 \\ x_t^1 \\ x_t^2 \\ x_t^3 \\ 1 \end{pmatrix}\right)$$

$$s_t = \varphi(W_b[s_{t-1}, x_t, 1])$$

$$s_t = \varphi(W_b[s_{t-1}, x_t])$$

"1" is usually omitted…



$$s_t = \varphi(W s_{t-1} + U x_t + b)$$

sometimes written as

$$s_t = \varphi(W_c[s_{t-1}, x_t] + b)$$

sometimes as

$$s_t = \varphi(W_b[s_{t-1}, x_t])$$

Relevant links:
https://deepsystems.ai/

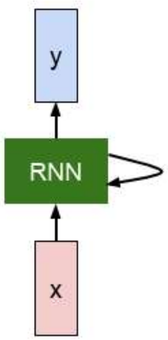
RNNs

RNN computation…

*At the core, RNNs have a deceptively simple API: They accept an input vector **x** and output vector **y**. However, this output vector is influenced not only by the input you fed in, but also by the entire history of inputs fed in previously.*

```
# x is an input vector
# y is the RNN's output vector
rnn = RNN()
y = rnn.step(x)
```

- *RNN's parameters are the three matrices W_hh, W_xh, W_hy*
- *hidden state self.h is initialized with the zero vector*
- *np.tanh function implements a non-linearity*



$$h_t = f_W(h_{t-1}, x_t)$$

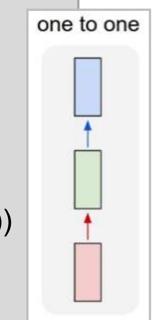
$$\downarrow$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

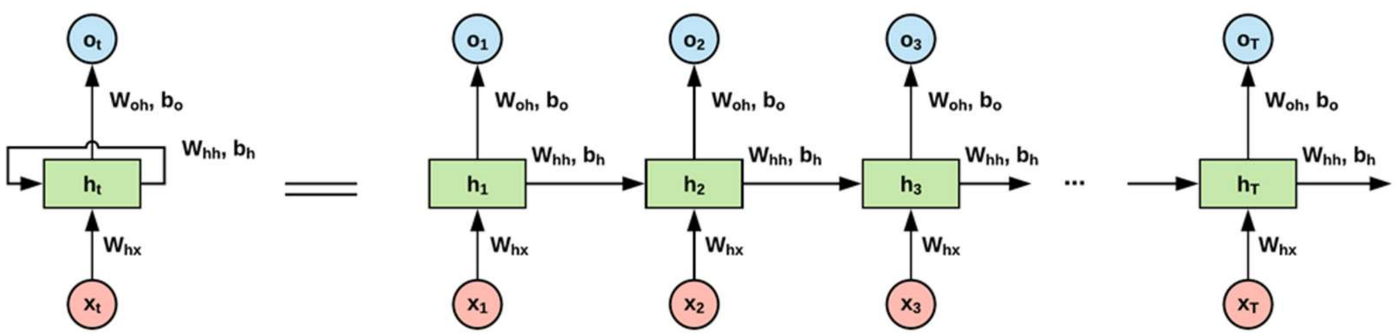
Vanilla RNN

```
# the RNN class has some internal state (in this case a simple hidden
vector h) that it gets to update every time step is called.
…
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```



a 2-layer recurrent network

```
y1 = rnn1.step(x)
y = rnn2.step(y1)
```



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

RNNs

RNN implementation in TensorFlow…

```
class MyRNNCell(tf.keras.layers.Layer):
    def __init__(self, rnn_units, input_dim, output_dim):
        super(MyRNNCell, self).__init__()
        # initialize weight matrices
        self.W_xh = self.add_weight([rnn_units, input_dim])
        self.W_hh = self.add_weight([rnn_units, rnn_units])
        self.W_hy = self.add_weight([output_dim, rnn_units])
        # initialize hidden state to zeros
        self.h = tf.zeros([rnn_units, 1])

    def call(self, x):
        # update the hidden state
        self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )
        # compute the output
        output = self.W_hy * self.h
        # return the current output and hidden state
        return output, self.h
```

```
…
tf.keras.layers.SimpleRNN(rnn_units)
```

RNNs

toy *Character-Level Language Models…(by Andrej Karpathy)*

The idea is to give the RNN a huge chunk of text and ask it to model the probability distribution of the next character in the sequence given a sequence of previous characters. This will allow generation of new text one character at a time.

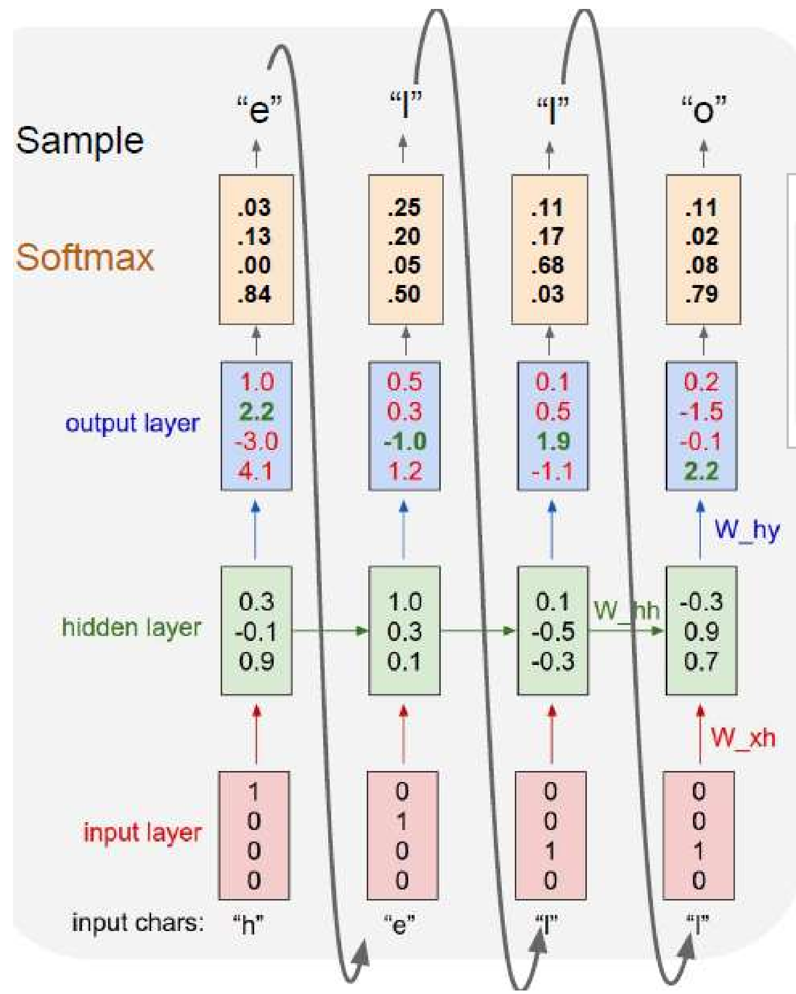
Pre-conditions:

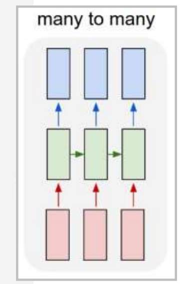
- *a vocabulary of four possible letters "**h, e, l, o**"*
- *a training data - the string "hello"*

The goal of the training is to increase the confidence of a desired target character (green) at every one of the 4 time steps and decrease the confidences of all other letters (red).

Implementations:

- *Minimal character-level language model (mini-char-rnn) with a Vanilla Recurrent Neural Network (in Python/numpy):*
  - https://gist.github.com/karpathy/d4dee566867f8291f086
  - https://www.tensorflow.org/tutorials/text/text_generation
- *Caracter-level language model (char-rnn):* https://github.com/karpathy/char-rnn





Relevant links:
http://karpathy.github.io/2015/05/21/rnn-effectiveness
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNNs

```
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
```

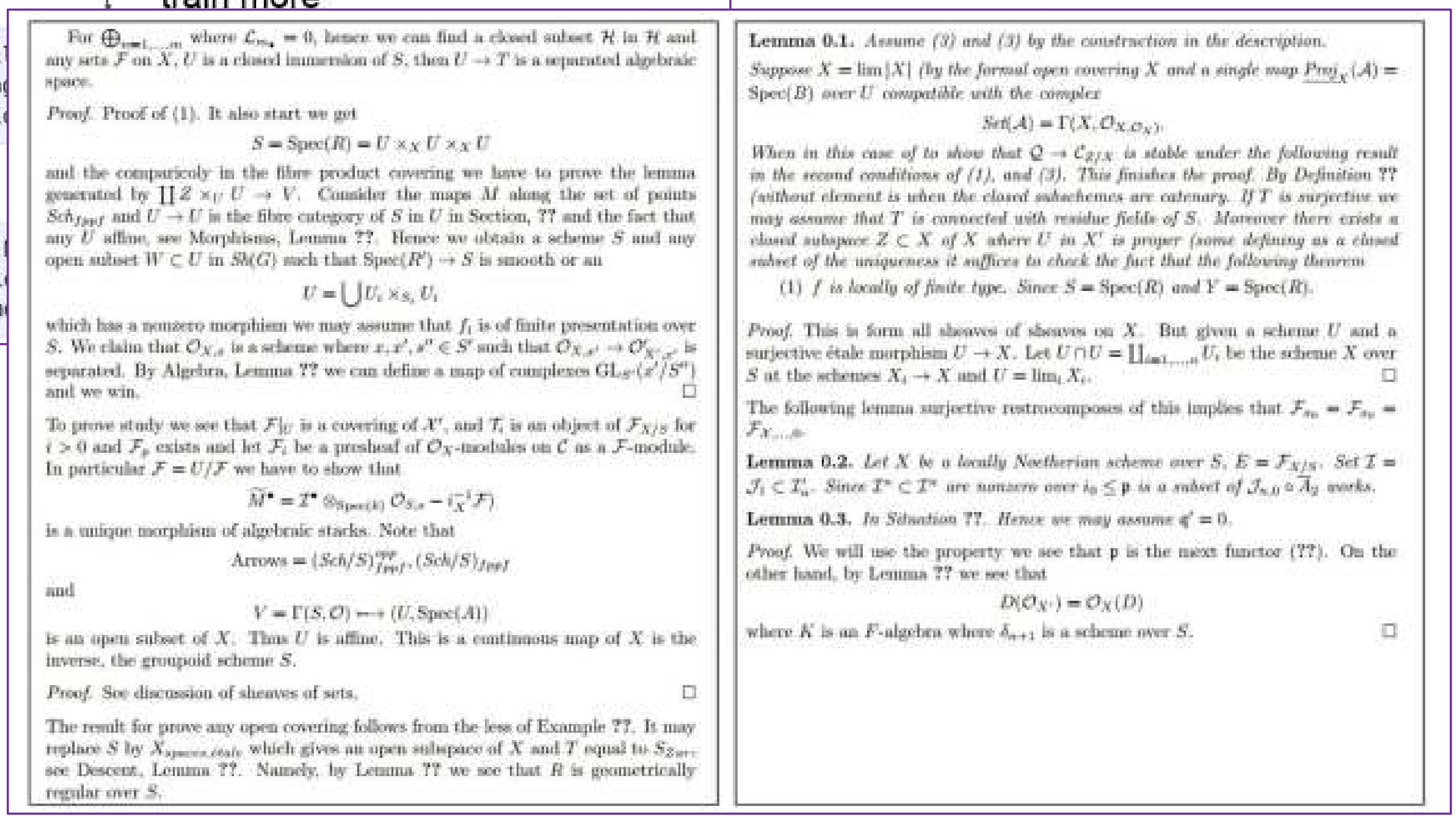
train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hal
her hearly, and behs to so arwa
how, and Gogition is so overeli
```

```
"Why do what that day," replied
princess, Princess Mary was easi
Pierre aking his soul came to th
```



For $\bigoplus_{n=1,\dots,m}$ 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. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longrightarrow (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) f is locally of finite type. Since $S = \mathrm{Spec}(R)$ and $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\dots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\dots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation ??. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

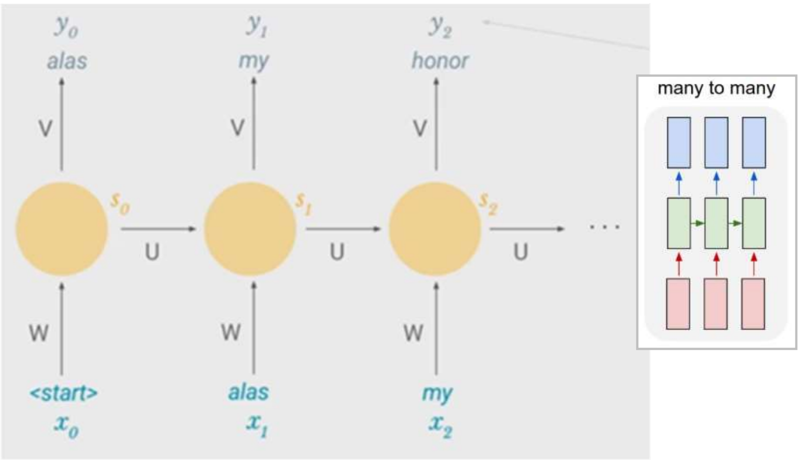
where K is an F-algebra where δ_{n+1} is a scheme over S. □

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

RNNs

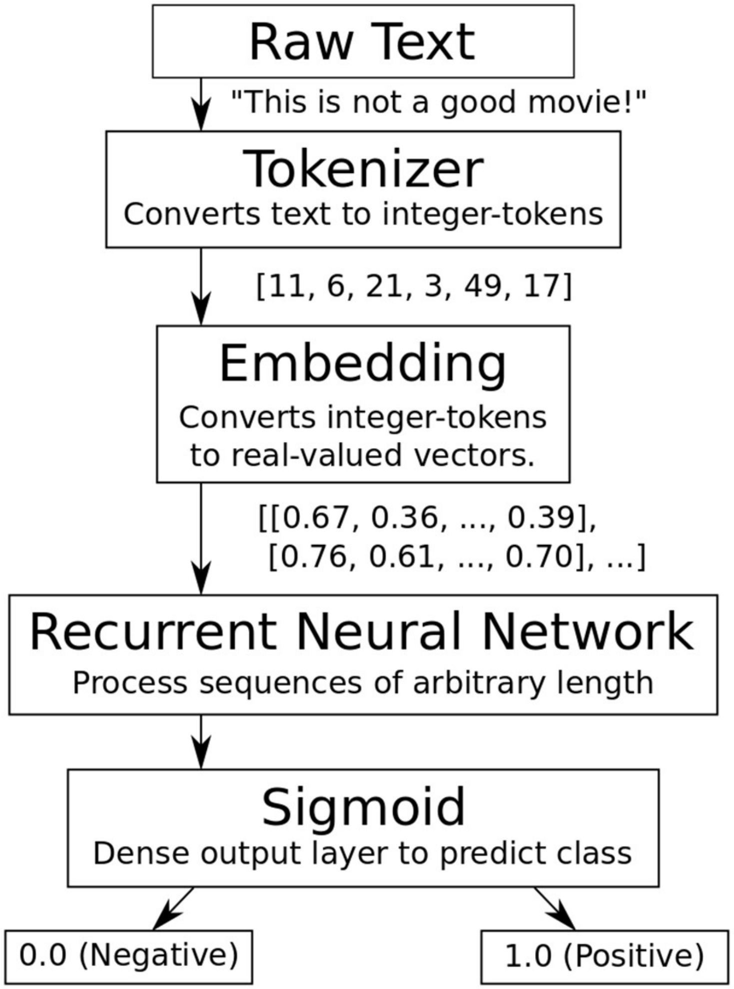
Word-Level Language Models could be implemented similarly to a *Character-Level models*.
There are some papers regarding Language Modeling and Generating Text:

- http://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.pdf
- http://www.fit.vutbr.cz/research/groups/speech/publi/2011/mikolov_icassp2011_5528.pdf
- http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Sutskever_524.pdf



Relevant links:
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Natural Language Processing with RNN



Relevant links:

https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/20_Natural_Language_Processing.ipynb

https://www.tensorflow.org/tutorials/representation/word2vec

https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

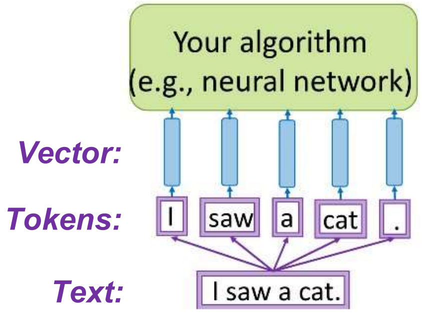
https://www.youtube.com/watch?v=wNBaNhvL4pg

https://www.youtube.com/watch?v=5PL0TmQhItY

https://platform.openai.com/tokenizer

arxiv.org/abs/2301.10472

Word embeddings



One-hot encoding

	cat	mat	on	sat	the
the =>	0	0	0	0	1
cat =>	1	0	0	0	0
sat =>	0	0	0	1	0
...		...			

cat = $\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$

from 8-dimensional (small datasets)

…

up to 1024-dimensions (large datasets)

*"He went to the prison **cell** with his **cell** phone to extract blood **cell** samples from inmates"*

Word2Vec ***GloVe*** ***ELMo*** ***BERT***

Have a good day vs. *Have a great day*

{*have*, *a*, *good*, *great*, *day*}

have = [1,0,0,0,0]

a = [0,1,0,0,0]

good = [0,0,1,0,0]

great = [0,0,0,1,0]

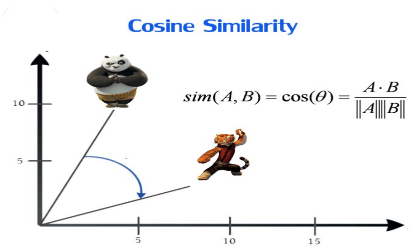
day = [0,0,0,0,1]

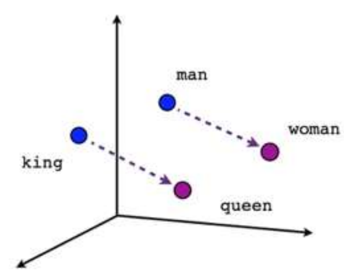
no projection along the other dimensions…

'good' and 'great' *are as different as* 'day' and 'have'

Objective: *to have words with similar context occupy close spatial positions.*

Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.





Male-Female

Relevant links:

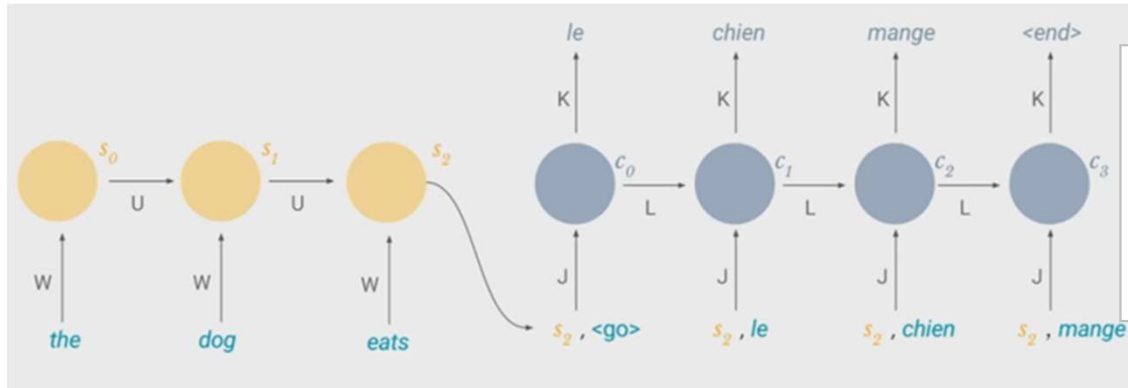
https://www.tensorflow.org/tutorials/text/word_embeddings
https://www.tensorflow.org/tutorials/representation/word2vec
https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
https://arxiv.org/pdf/1411.2738.pdf
https://medium.com/@jonathan_hui/nlp-word-embedding-glove-5e7f523999f6
https://code.google.com/archive/p/word2vec/
https://heartbeat.fritz.ai/the-7-nlp-techniques-that-will-change-how-you-communicate-in-the-future-part-i-f0114b2f0497
https://nlp.stanford.edu/projects/glove/ , *https://machinelearningmastery.com/what-are-word-embeddings/*
https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b
https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/
https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/
http://jalammar.github.io/illustrated-bert/

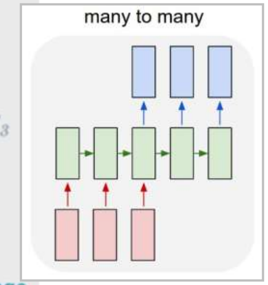
RNNs

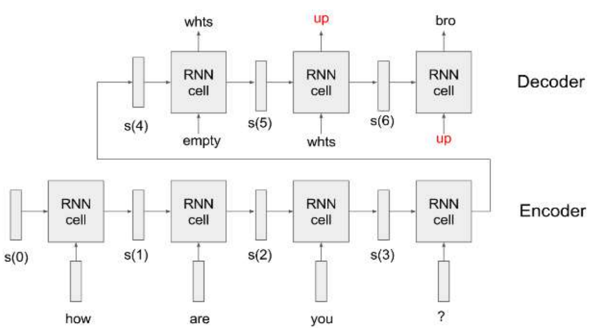
Machine Translation *is similar to language modeling in that our input is a sequence of words in source language (e.g. English), and the output is a sequence of words in target language (e.g. French).*

*It could be considered as combination of two architectures: many-to-one (**Encoder**) and one-to-many (**Decoder**)…*

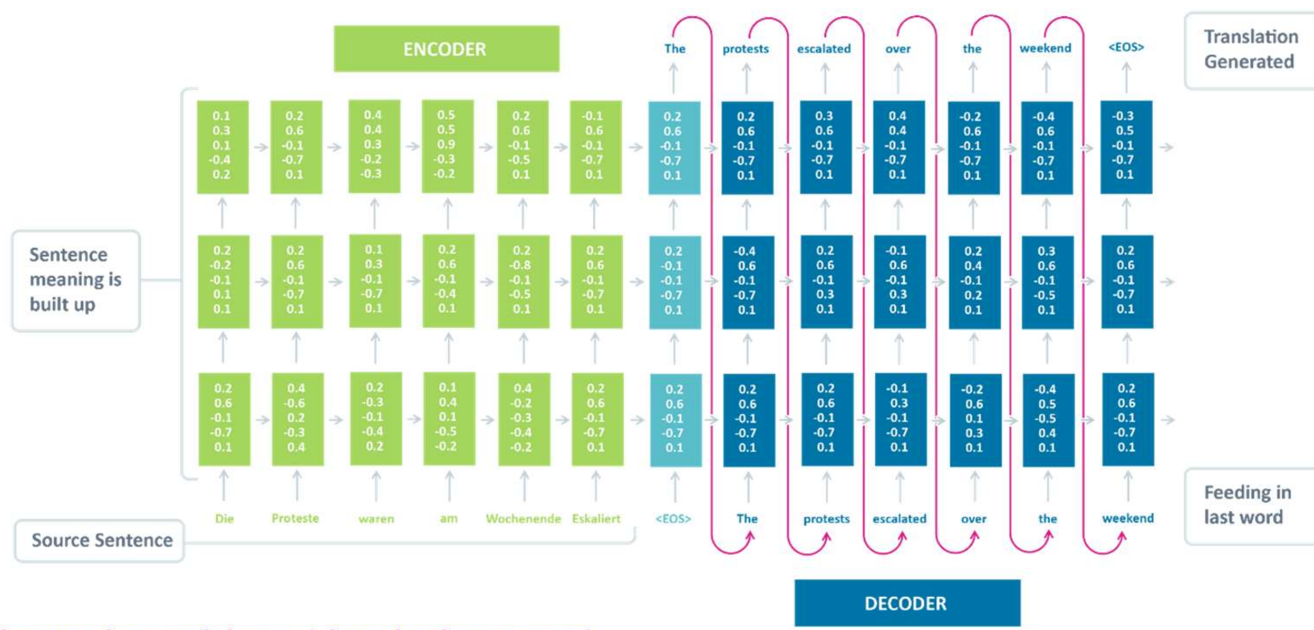
or English to slang English…







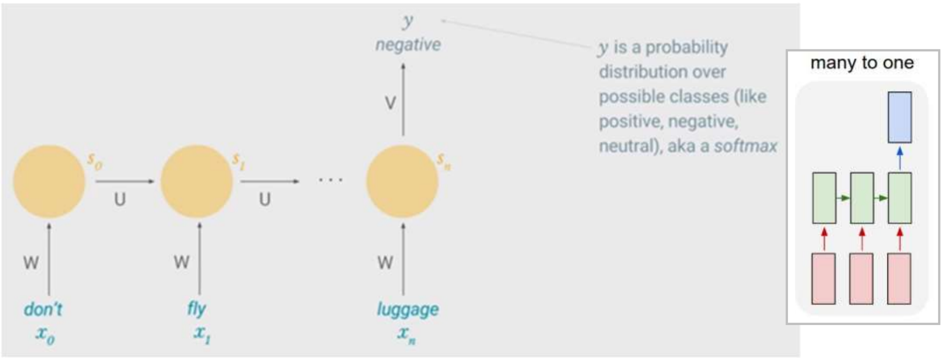
A Recurrent Neural Network for Machine Translation

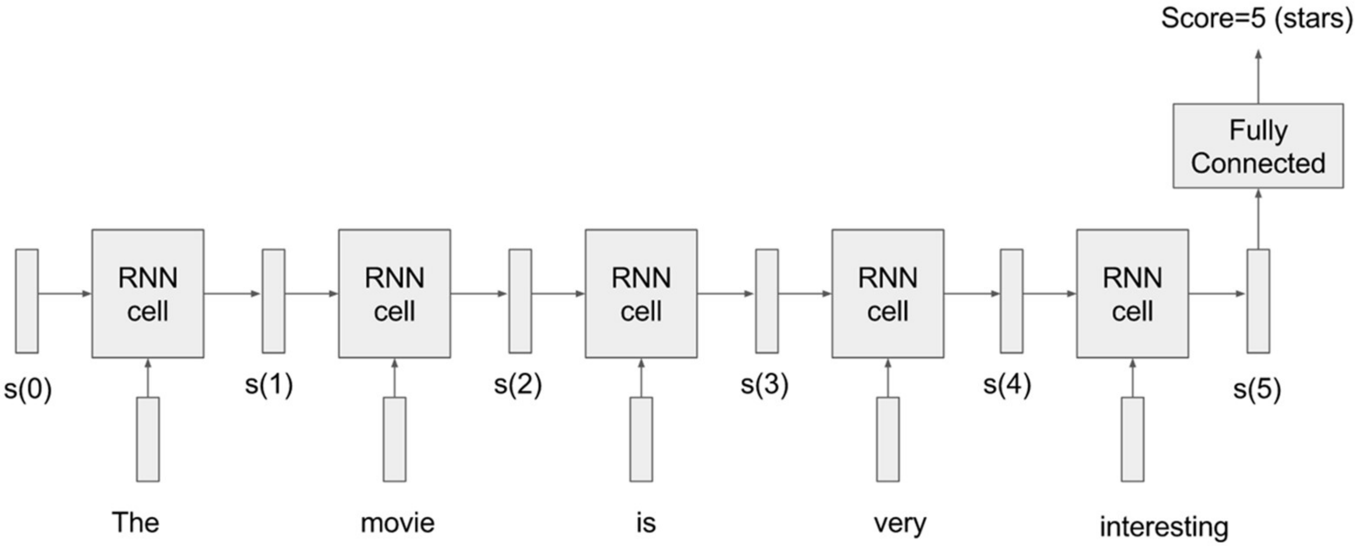


Relevant links:
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
https://deepsystems.ai/

RNNs

***Classificaltion** of the text for sentiment analysis.*





Relevant links:
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
https://deepsystems.ai/

Training RNNs

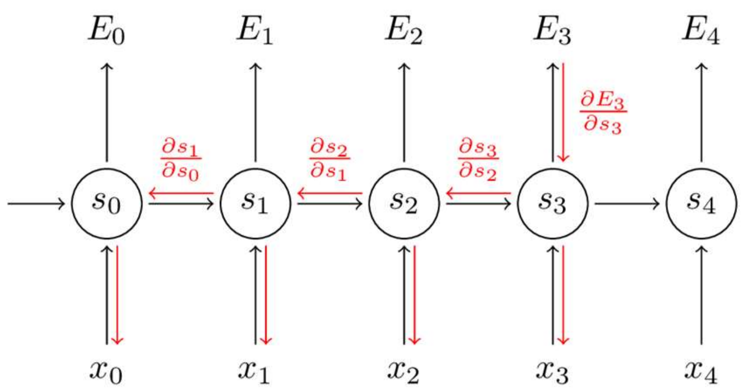
Similarly to a traditional Neural Network, *training a RNN* is done using the backpropagation algorithm with a little twist - ***Backpropagation Through Time (BPTT)***. Since parameters are shared among all time steps in the network, the gradient at each output depends not only on the calculations of the current, but also the previous time steps (in order to calculate the gradient at *t=4* we would need to backpropagate 3 steps and sum up the gradients.

A naïve implementation of BPTT:

```
def bptt(self, x, y):
    T = len(y)
    # Perform forward propagation
    o, s = self.forward_propagation(x)
    # We accumulate the gradients in these variables
    dLdU = np.zeros(self.U.shape)
    dLdV = np.zeros(self.V.shape)
    dLdW = np.zeros(self.W.shape)
    delta_o = o
    delta_o[np.arange(len(y)), y] -= 1.
    # For each output backwards...
    for t in np.arange(T)[::-1]:
        dLdV += np.outer(delta_o[t], s[t].T)
        # Initial delta calculation: dL/dz
        delta_t = self.V.T.dot(delta_o[t]) * (1 - (s[t] ** 2))
        # Backpropagation through time (for at most self.bptt_truncate steps)
        for bptt_step in np.arange(max(0, t-self.bptt_truncate), t+1)[::-1]:
            # print "Backpropagation step t=%d bptt step=%d " % (t, bptt_step)
            # Add to gradients at each previous step
            dLdW += np.outer(delta_t, s[bptt_step-1])
            dLdU[:,x[bptt_step]] += delta_t
            # Update delta for next step dL/dz at t-1
            delta_t = self.W.T.dot(delta_t) * (1 - s[bptt_step-1] ** 2)
    return [dLdU, dLdV, dLdW]
```

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\frac{\partial s_3}{\partial s_1} = \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1}$$



Relevant links:
http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

What is wrong with Vanilla RNN?

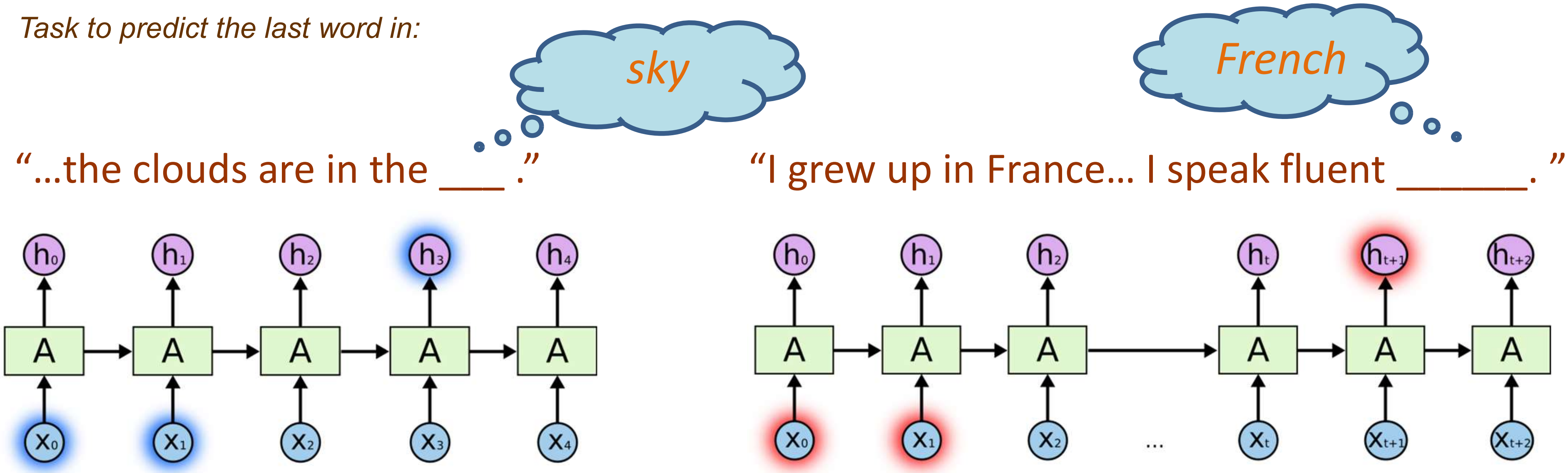
$$s_t = \varphi(W s_{t-1} + U x_t + b)$$

- ***Non-linearity*** ***is bad for long term memory…***
The RNN state (memory) should be protected using only ”+” or ”-” operations to write to it.

- ***No selectivity*** ***(reade all, overwrite all)…***
It should be possible to choose what to read, write and forget.

Long-term dependencies

Task to predict the last word in:



Unfortunately… *in practice, ordinary RNNs are not really capable to solve such "long-term dependencies" problem. The problem and reasons why it might be difficult were explored in depth in following works:*
http://people.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf
http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf

Fortunately… *Hochreiter and Schmidhuber, as well as many other researchers who refined and popularized them, introduced* ***Long Short Term Memory networks (LSTMs)*** *that show tremendously great performance on a large variety of problems..*
http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf

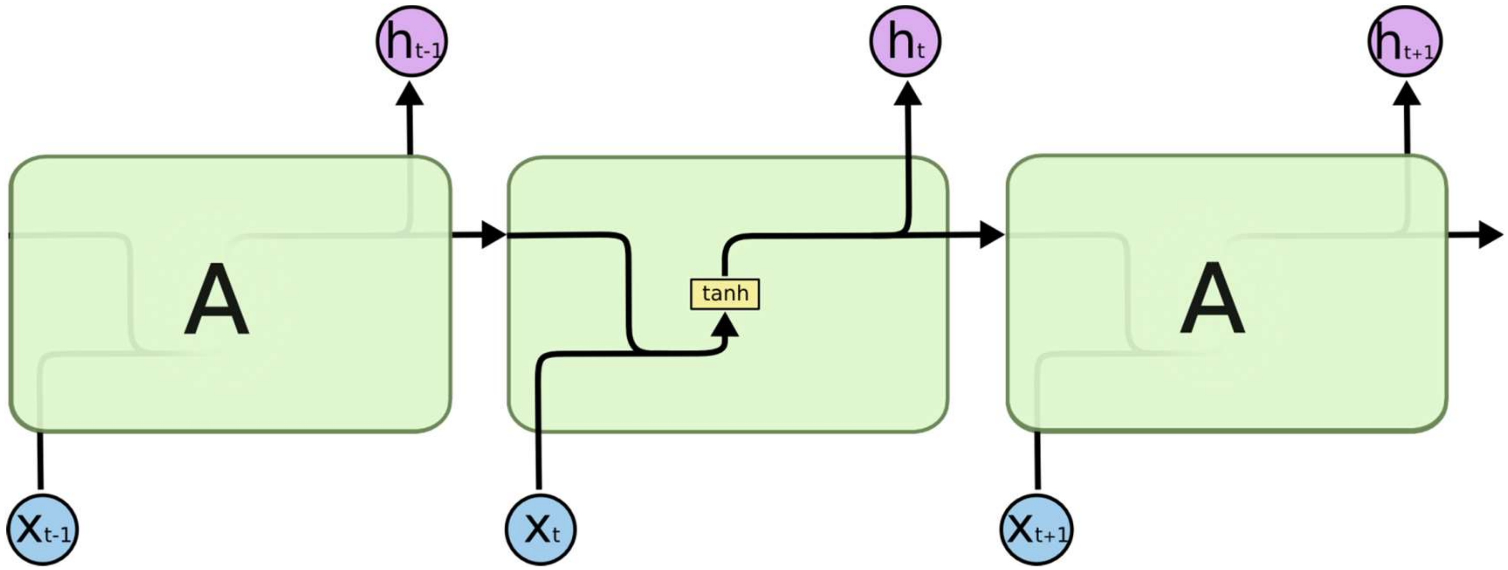
Relevant links:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs

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

RNNs have the form of a chain of repeating modules of neural network…

The repeating module in a standard RNN contains a single layer:



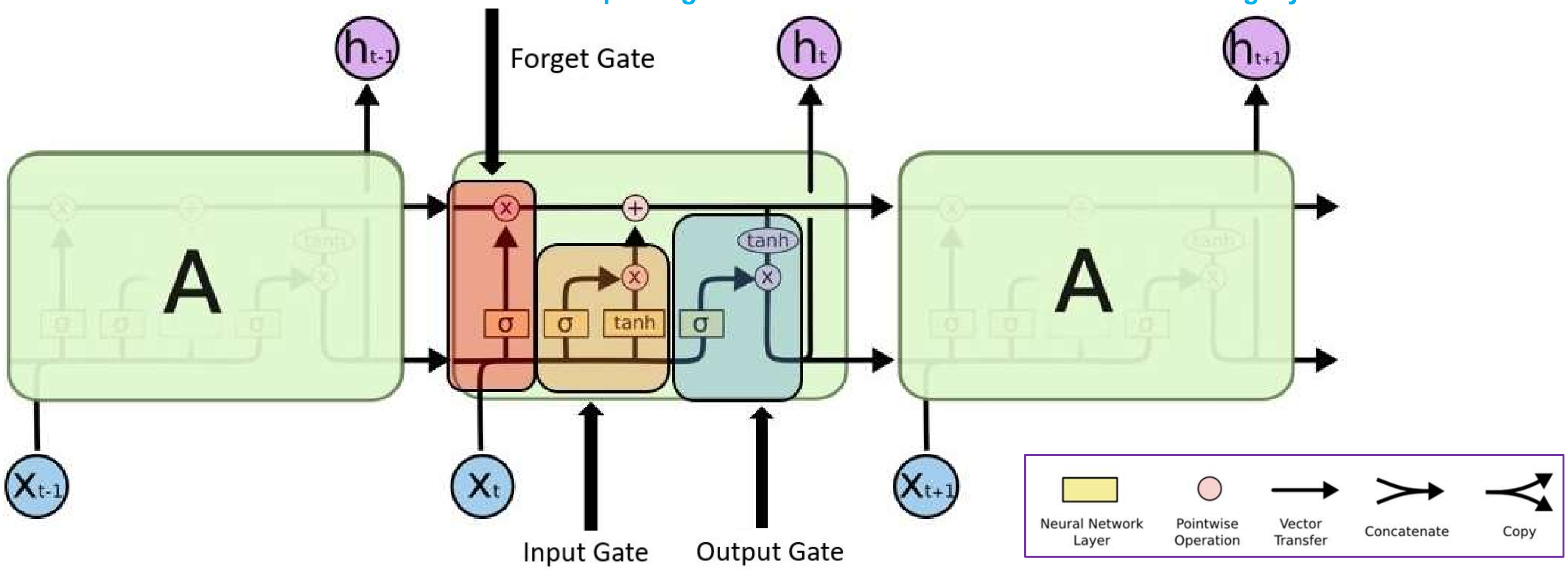
Relevant links:
http://karpathy.github.io/2015/05/21/rnn-effectiveness
https://www.youtube.com/watch?v=WCUNPb-5EYI

LSTMs

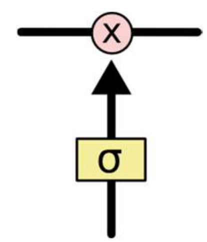
Long Short Term Memory networks *– usually just called* ***"LSTMs"*** *– are a special kind of RNN, capable of learning long-term dependencies.*

RNNs have the form of a chain of repeating modules of neural network…

The repeating module in an LSTM contains four interacting layers:



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called *gates*. Gates are a way to optionally let information through. They are composed out of a *sigmoid neural net layer* (describing how much of each component should be let through) and *a pointwise multiplication operation*.



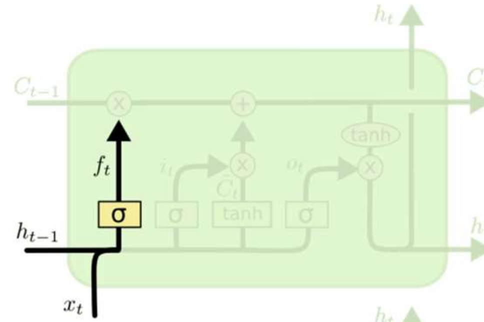
Relevant links:

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

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

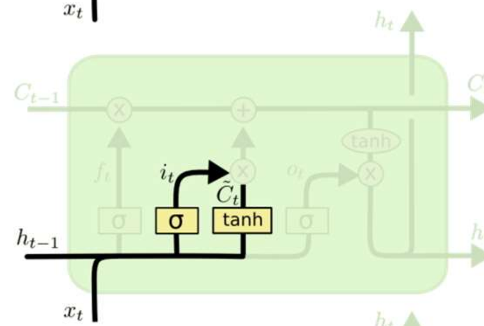
LSTMs

"forget gate layer" decides what information we are going to throw away from the cell state.
e.g. when we see a new subject, we want to forget the gender of the old subject…



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

… what new information we're going to store in the cell state? *Sigmoid* ***"input gate layer"*** decides which values to update and a *tanh layer* creates a vector of new candidate values that could be added to the state. Combination of those updates the state.
e.g. add the gender of the new subject to the cell state, to replace the old one we are forgetting…

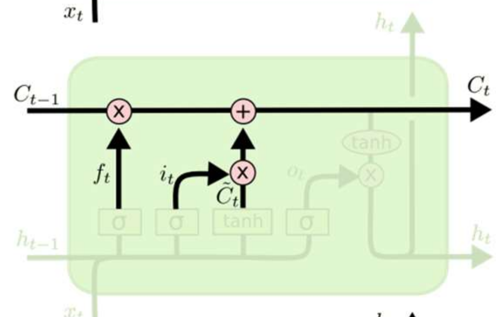


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Update the old cell state C(t-1) into the new cell state C(t):

- multiply the old state by output of "forget gate layer" forgetting the things we decided to forget earlier.
- add the new candidate values, scaled by how much we decided to update each state value.

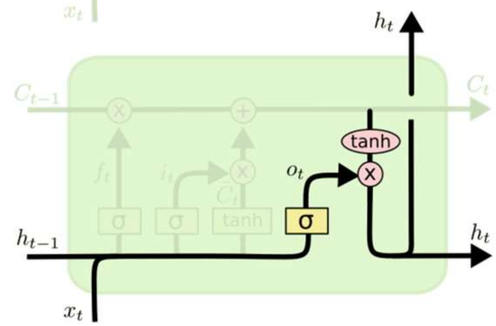
e.g. drop the information about the old subject's gender and add the new information, as were decided in the previous steps…



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

…***"output gate"*** decides what to output. It will be a filtered version of the cell state.

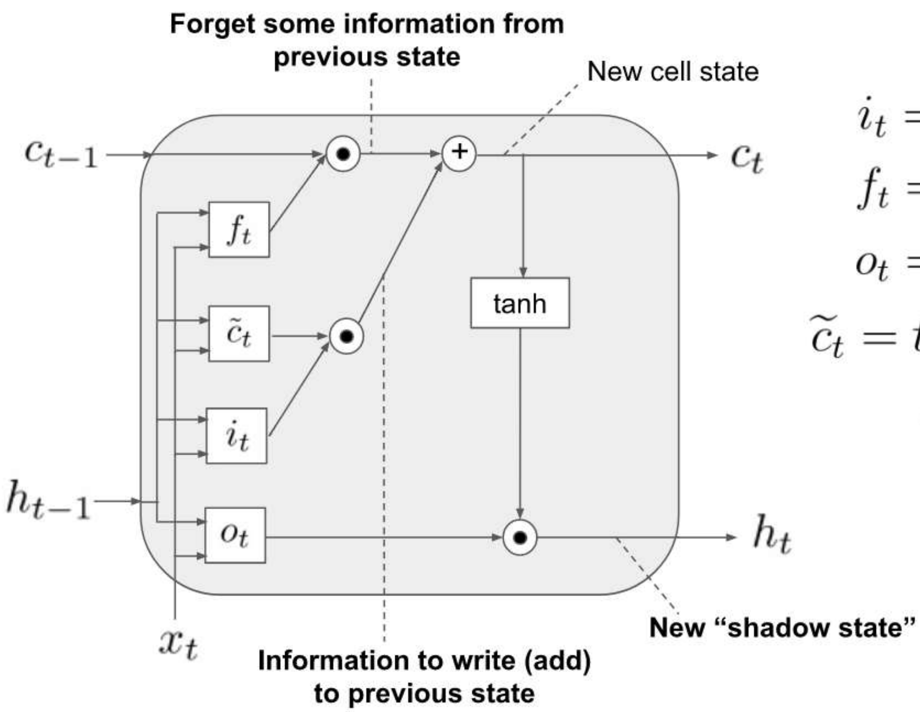
- a *sigmoid layer* decides what parts of the cell state we're going to output.
- *tanh* pushes the values to be between [−1; 1] and multiplies it by the output of the *sigmoid gate*.



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

Relevant links:
http://karpathy.github.io/2015/05/21/rnn-effectiveness
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

What is the bottleneck of LSTMs?



LSTM equations

$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$	Input gate	Write gate
$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$	Forget gate	
$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$	Output gate	Read gate
$\widetilde{c}_t = tanh(W h_{t-1} + U x_t + b)$	Memory cell candidate	
$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$	Memory cell	
$h_t = o_t \circ tanh(c_t)$	Shadow state	
$y_t = h_t$	Cell Output	

Conceptually we loose some information since calculate our gates based on *"filtered" output* of the previous state (*h*), and do not consider actual cell mamory (*c*).

Relevant links:
https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html#information-morphing-and-vanishing-and-exploding-sensitivity
http://www.bioinf.jku.at/publications/older/2604.pdf
https://deepsystems.ai/

LSTM with Peephole connections

Peephole connections (introduced by Gers & Schmidhuber, 2000) let the gate layers look at the cell state. ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf

LSTM equations

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$\widetilde{c}_t = tanh(W h_{t-1} + U x_t + b)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$

$$h_t = o_t \circ tanh(c_t)$$
$$y_t = h_t$$

LSTM with peephole connections

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + \underline{P_i c_{t-1}} + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + \underline{P_f c_{t-1}} + b_f)$$

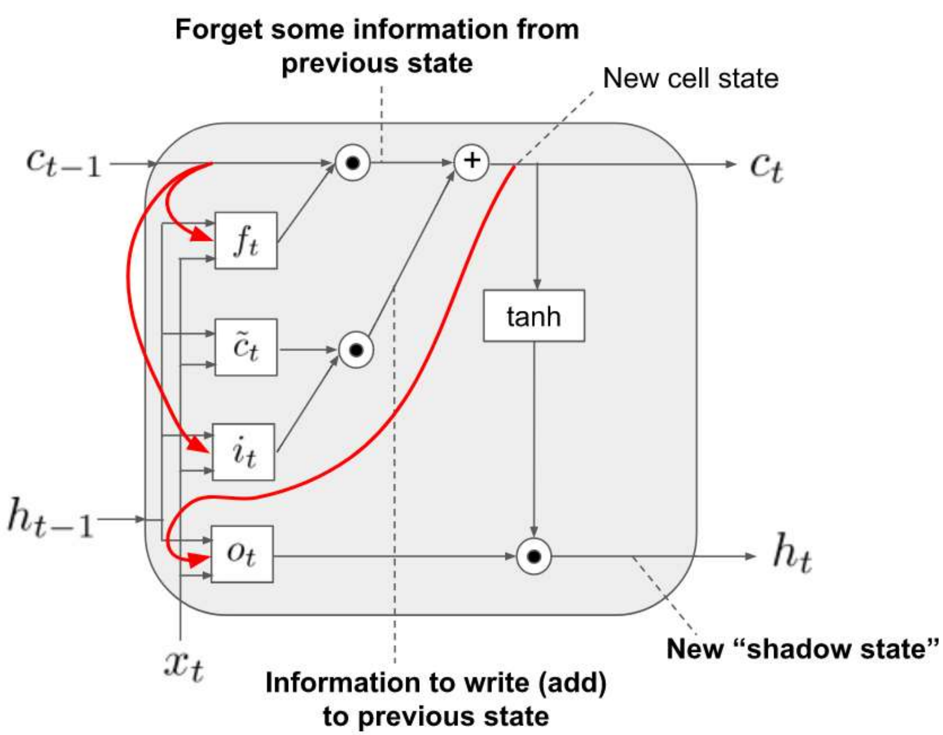
$$\widetilde{c}_t = tanh(W h_{t-1} + U x_t + b)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + \underline{P_o c_t} + b_o)$$
$$h_t = o_t \circ tanh(c_t)$$
$$y_t = h_t$$

Note that each ***Px*** *is an n×n matrix (a peephole matrix), much like each* ***Wx****…*

Relevant links:
https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html#information-morphing-and-vanishing-and-exploding-sensitivity
http://ieeexplore.ieee.org/document/861302/?reload=true
https://arxiv.org/pdf/1308.0850v5.pdf
https://deepsystems.ai/

LSTM with Peephole connections

Peephole connections (introduced by Gers & Schmidhuber, 2000) let the gate layers look at the cell state. ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf



LSTM with peephole connections

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + \underline{P_i c_{t-1}} + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + \underline{P_f c_{t-1}} + b_f)$$

$$\widetilde{c}_t = tanh(W h_{t-1} + U x_t + b)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + \underline{P_o c_t} + b_o)$$

$$h_t = o_t \circ tanh(c_t)$$
$$y_t = h_t$$

*Note that each **Px** is an n×n matrix (a peephole matrix), much like each **Wx**…*

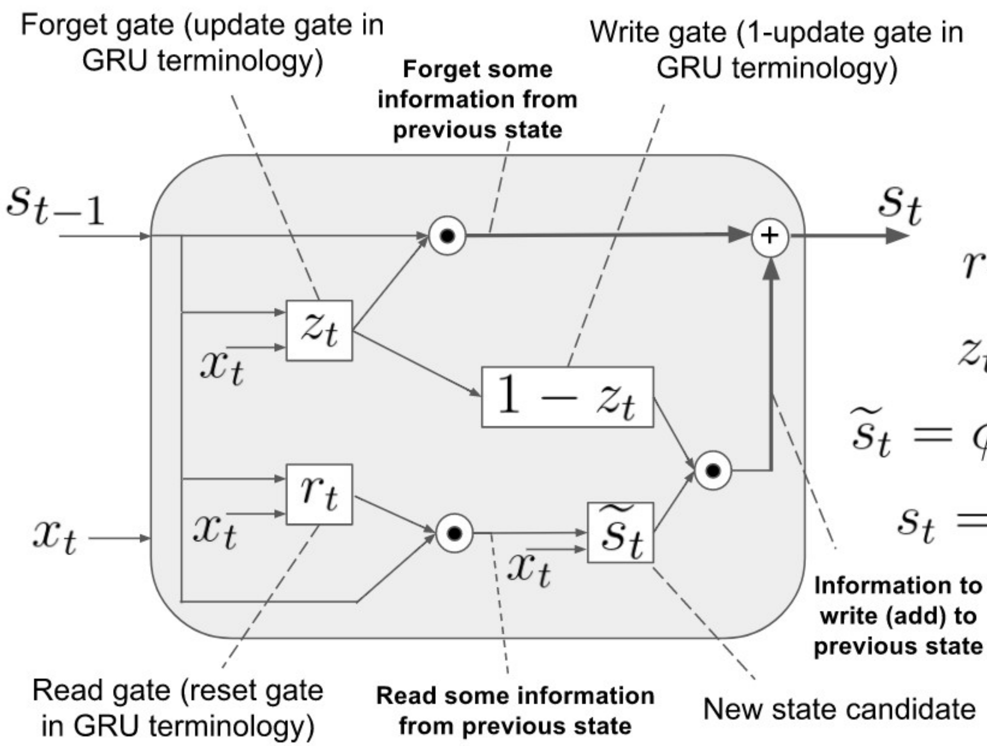
Relevant links:
https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html#information-morphing-and-vanishing-and-exploding-sensitivity
http://ieeexplore.ieee.org/document/861302/?reload=true
https://arxiv.org/pdf/1308.0850v5.pdf
https://deepsystems.ai/

GRU

Gated Recurrent Unit (GRU) variation of LSTM (introduced by Cho et al., 2014):

- combines the forget and input gates into a single "*update gate*"
- merges the cell state and hidden state
- does some other changes

The resulting model is simpler than *standard LSTM* models, and has been growing increasingly popular…



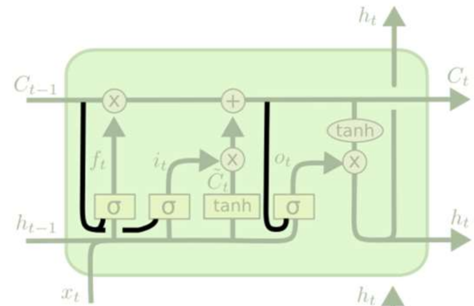
GRU equations

		In GRU terminology
$r_t = \sigma(W_r s_{t-1} + U_r x_t + b_r)$	Read gate	Reset gate
$z_t = \sigma(W_z s_{t-1} + U_z x_t + b_z)$	Forget gate	Update gate
$\widetilde{s}_t = \phi(W\ (r_t \circ s_{t-1}) + U x_t + b)$	State candidate	State candidate
$s_t = z_t \circ s_{t-1} + \underbrace{(1 - z_t)}_{\text{Write gate}} \circ \widetilde{s}_t$	Current State	Current State

Relevant links:
https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html#information-morphing-and-vanishing-and-exploding-sensitivity
http://emnlp2014.org/papers/pdf/EMNLP2014179.pdf
https://arxiv.org/pdf/1412.3555.pdf
https://deepsystems.ai/

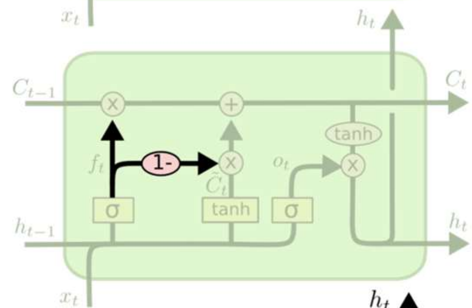
Variations on LSTMs…

LSTM variant with "***peephole connections***" (introduced by Gers & Schmidhuber, 2000) lets the gate layers look at the cell state. ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Another variation uses *coupled forget and input gates*. Here we only forget when we are going to input something in its place, and input new values to the state only when we forget something older.

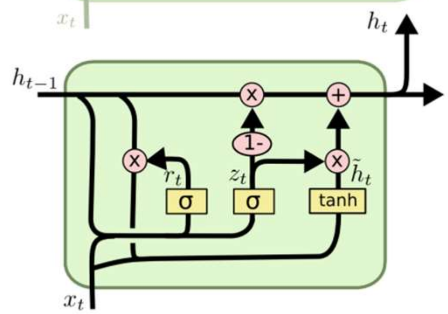


$$C_t = f_t * C_{t-1} + \boldsymbol{(1 - f_t)} * \tilde{C}_t$$

Gated Recurrent Unit (GRU) variation (introduced by Cho et al., 2014):

- combines the forget and input gates into a single "*update gate*"
- merges the cell state and hidden state
- does some other changes

The resulting model is simpler than *standard LSTM* models, and has been growing increasingly popular…

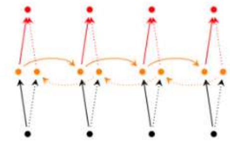


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Depth Gated RNNs (by Yao et al., 2015): http://arxiv.org/pdf/1508.03790v2.pdf

Clockwork RNNs (by Koutnik et al., 2014) is completely different approach to tackling long-term dependencies: http://arxiv.org/pdf/1402.3511v1.pdf

Relevant links:
http://karpathy.github.io/2015/05/21/rnn-effectiveness
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://arxiv.org/pdf/1503.04069.pdf
http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf

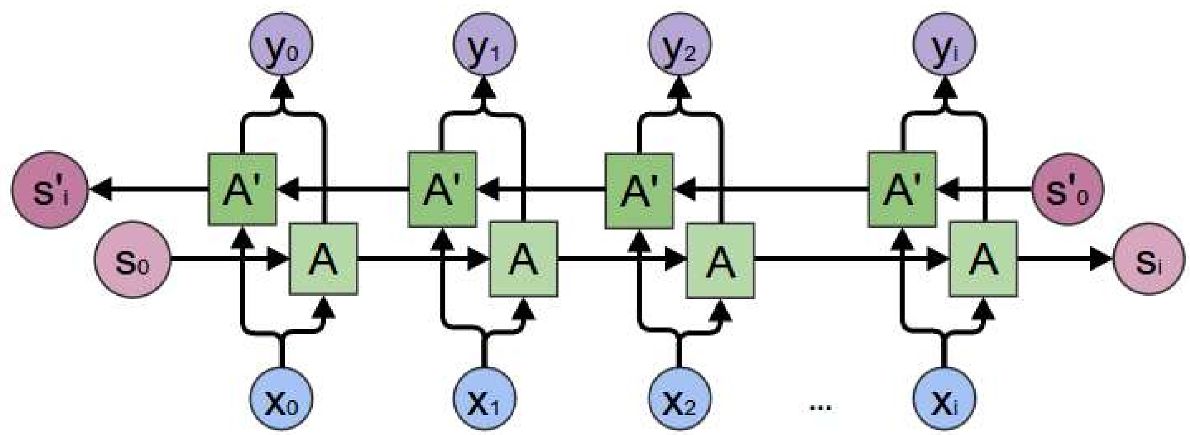


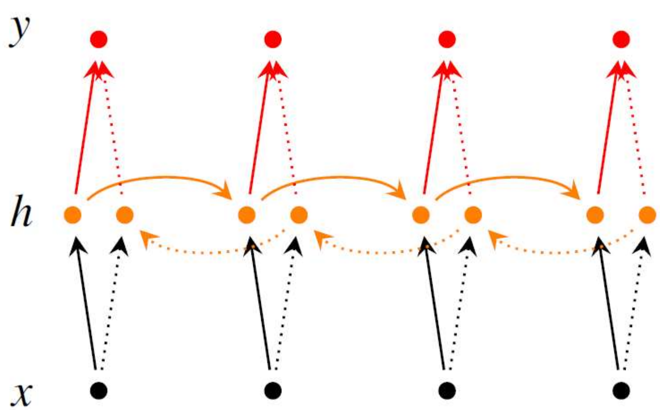
BiRNNs

Bi-directional RNNs (BiRNNs) *a special kind of RNNs that also traverse in the reverse direction, to understand context not only from the past, but from the future as well...*

This type of nets are based on the idea that the output at time ***t*** may not only depend on the previous elements in the sequence, but also future elements. Therefore, to predict a missing word in a sequence you want to look at both the left and the right context.

BiRNNs are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs.

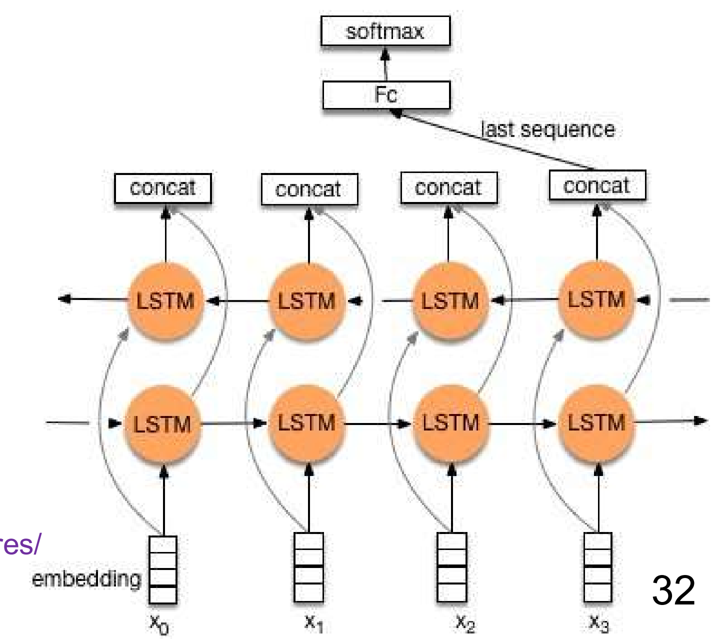




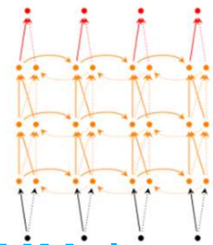
$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

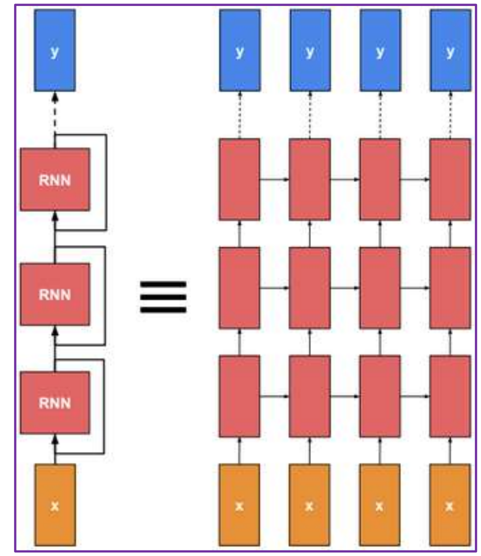
$$y_t = g(U[\vec{h}_t;\overleftarrow{h}_t] + c)$$



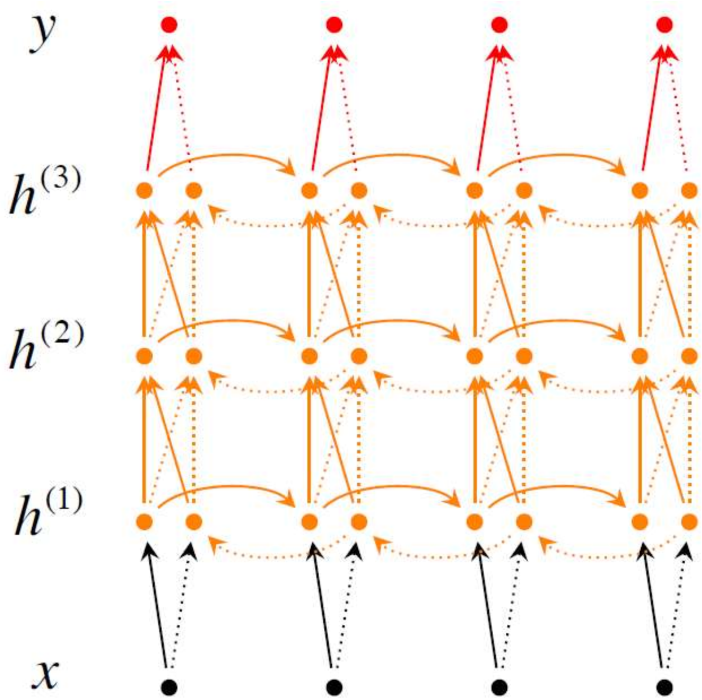
Relevant links:
https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66
http://www.wildml.com/2016/08/rnns-in-tensorflow-a-practical-guide-and-undocumented-features/



Deep BiRNNs



Deep Bi-directional RNNs (Deep BiRNNs) *are similar to Bi-directional RNNs, only that it has multiple layers per time step. In practice this gives a higher learning capacity (but also requires a lot of training data).*



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

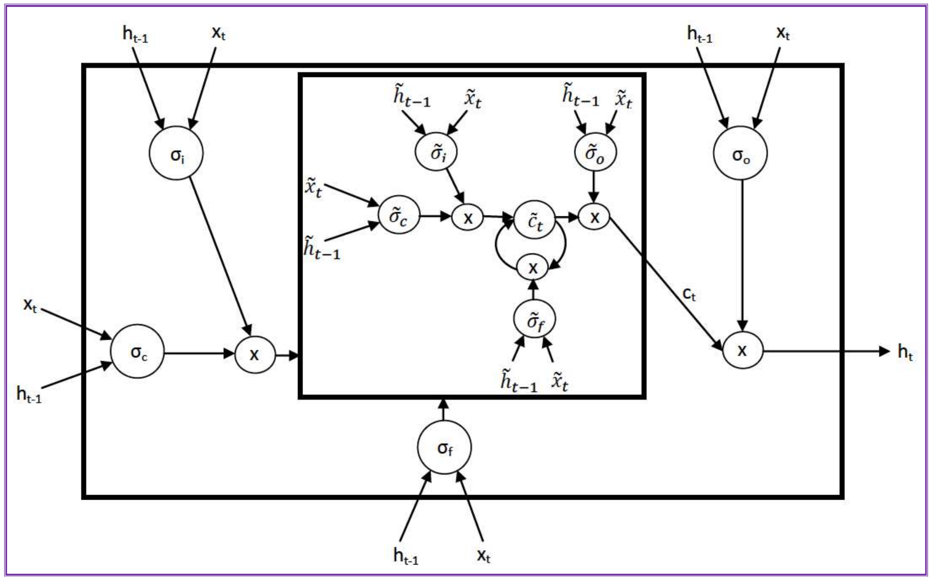
$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

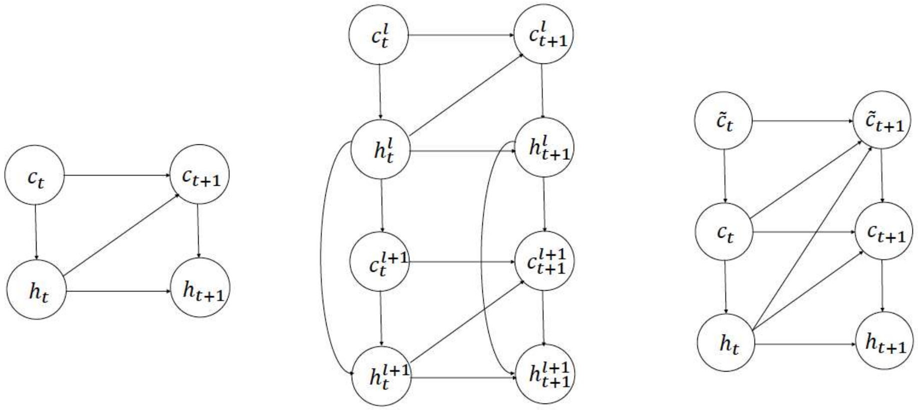
Relevant links:
https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66
http://www.wildml.com/2016/08/rnns-in-tensorflow-a-practical-guide-and-undocumented-features/

Nested LSTM

Nested LSTMs (NLSTM) *a novel RNN architecture with multiple levels of memory.*
(by Moniz and Krueger, submitted 2018)

Nested LSTMs add depth to LSTMs via nesting as opposed to stacking. The value of a memory cell in an NLSTM is computed by an LSTM cell, which has its own inner memory cell. Nested LSTMs outperform both stacked and single-layer LSTMs with similar numbers of parameters in author's experiments on various character-level language modeling tasks, and the inner memories of an LSTM learn longer term dependencies compared with the higher-level units of a stacked LSTM.





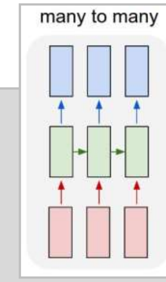
(a) LSTM (b) Stacked LSTM (c) Nested LSTM

Relevant links:
https://arxiv.org/abs/1801.10308
https://github.com/hannw/nlstm

LSTMs

Implementations…

2-layer LSTM based Language Modeling to predict the next word (TF v1):




```
import time
import collections
import random
import tensorflow as tf
import numpy as np
from tensorflow.contrib import rnn

start_time = time.time()
def elapsed(sec):
    if sec<60:
        return str(sec)+ " sec"
    elif sec<(60*60):
        return str(sec/60)+ " min"
# Training source file with words
training_file = 'data.txt'
def read_data(fname):
    with open(fname) as f:
        content = f.readlines()
    content = [x.strip() for x in content]
    content = [content[i].split() for i in range(len(content))]
    content = np.array(content)
    content = np.reshape(content, [-1,])
    return content
training_data = read_data(training_file)
print("Loaded training data...")

def build_dataset(words):
    count = collections.Counter(words).most_common()
    dictionary = dict()
    for word, _ in count:
        dictionary[word] = len(dictionary)
    reverse_dictionary = dict(zip(dictionary.values(), dictionary.keys()))
    return dictionary, reverse_dictionary
dictionary, reverse_dictionary = build_dataset(training_data)
vocab_size = len(dictionary)
```

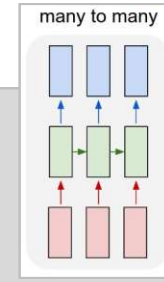
```
# Parameters
learning_rate = 0.001
training_iters = 50000
display_step = 1000
n_input = 3
# Number of units in RNN cell
n_hidden = 512
# tf Graph input
x = tf.placeholder("float", [None, n_input, 1])
y = tf.placeholder("float", [None, vocab_size])
# RNN output node weights and biases
weights = {
    'out': tf.Variable(tf.random_normal([n_hidden, vocab_size]))
}
biases = {
    'out': tf.Variable(tf.random_normal([vocab_size]))
}
def RNN(x, weights, biases):
    # reshape x for compatibility
    x = tf.reshape(x, [-1, n_input])
    # Convert input words to sequence of inputs
    # e.g. [Company] [size] [is] -> [650] [30] [45]
    x = tf.split(x, n_input, 1)
    #2-layer LSTM, each layer contains n_hidden units
    # Avarage Accuracy is 95% at 50K iterations. With 1-layer LSTM, accuracy is 90%...
    rnn_cell = rnn.MultiRNNCell([rnn.BasicLSTMCell(n_hidden), rnn.BasicLSTMCell(n_hidden)])
    #generate prediction
    outputs, states = rnn.static_rnn(rnn_cell, x, dtype=tf.float32)
    # there are n_inputs outputs, but we need only the last one
    return tf.matmul(outputs[-1], weights['out']) + biases['out']
pred = RNN(x, weights, biases)
# Loss and Optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
optimizer = tf.train.RMSPropOptimizer(learning_rate=learning_rate).minimize(cost)
#Evaluation of the Model
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

Relevant links:
https://www.youtube.com/watch?v=y7qrilE-Zlc

LSTMs

Implementations…

2-layer LSTM based Language Modeling to predict the next word (TF v1):




```
# Variable initialization
init = tf.global_variables_initializer()
# Launching the graph
with tf.Session() as session:
    session.run(init)
    step = 0
    offset = random.randint(0, n_input+1)
    end_offset = n_input + 1
    acc_total = 0
    loss_total = 0
    while step < training_iters:
        # Generate a minibatch with some randomness on selection process
        if offset > (len(training_data)-end_offset):
            offset = random.randint(0, n_input+1)
        symbols_in_keys = [[dictionary[str(training_data[i])]] for i in range(offset,
                                offset+n_input)]
        symbols_in_keys = np.reshape(np.array(symbols_in_keys), [-1, n_input, 1])
        symbols_out_onehot = np.zeros([vocab_size], dtype=float)
        symbols_out_onehot[dictionary[str(training_data[offset+n_input])]] = 1.0
        symbols_out_onehot = np.reshape(symbols_out_onehot, [-1, vocab_size])
        _, acc, lass, onehot_pred = session.run([optimizer, accuracy, cost, pred],
                                   feed_dict={x: symbols_in_keys, y: symbols_out_onehot})
        loss_total += lass
        acc_total += acc
        if (step+1) % display_step == 0:
            print("iter= " + str(step+1) + ", avarage loss= {:.6f}".format(loss_total/display_step)
                           + ", avarage accuracy= {:.2f}".format(100*acc_total/display_step))
            acc_total = 0
            loss_total = 0
            symbols_in = [training_data[i] for i in range(offset, offset+n_input)]
            symbols_out = training_data[offset+n_input]
            symbols_out_pred = reverse_dictionary[int(tf.argmax(onehot_pred, 1).eval())]
            print("%s - [%s] vs [%s]" % (symbols_in, symbols_out, symbols_out_pred))
        step += 1
        offset += (n_input+1)
    print("Optimization is finished...")
    print("Elapsed time: ", elapsed(time.time() - start_time))
    print("Run on command line.")
```

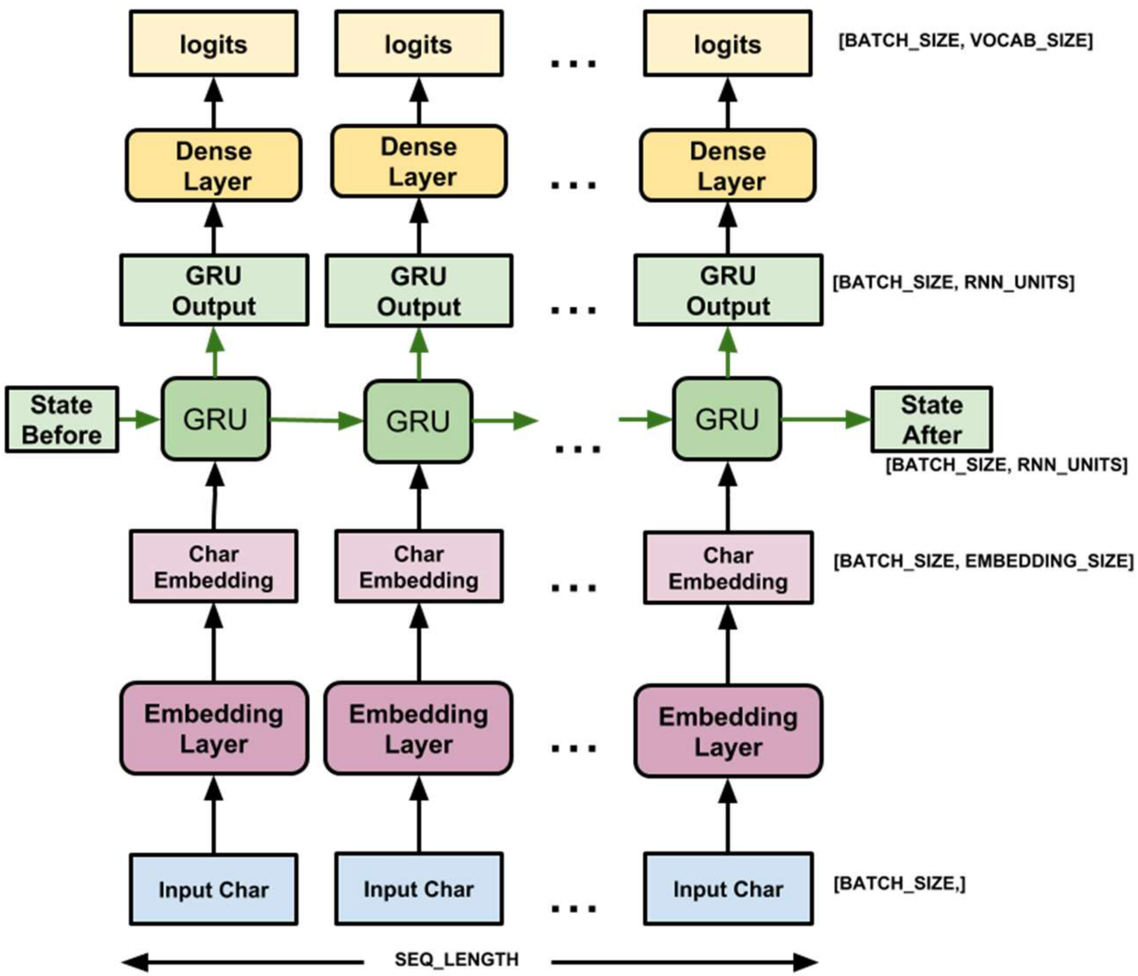
```
    while True:
        prompt = "%s words: " % n_input
        sentance = input(prompt)
        sentance = sentance.strip()
        words= sentance.split(' ')
        if len(words) != n_input:
            continue
        try:
            symbols_in_keys = [dictionary[str(words[i])] for i in range(len(words))]
            for i in range(32):
                keys = np.reshape(np.array(symbols_in_keys), [-1, n_input, 1])
                onehot_pred = session.run(pred, feed_dict={x: keys})
                onehot_pred_index = int(tf.argmax(onehot_pred, 1).eval())
                sentance = "%s %s" % (sentance, reverse_dictionary[onehot_pred_index])
                symbols_in_keys = symbols_in_keys[1:]
                symbols_in_keys.append(onehot_pred_index)
            print(sentance)
        except:
            print("Word is not in dictionary")
```

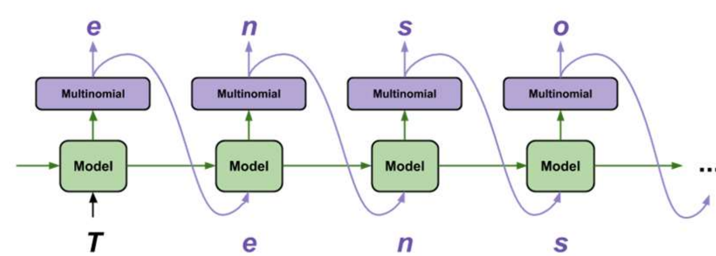
```
iter= 48000, avarage loss= 0.204543, avarage accuracy= 94.00
['nobody', 'spoke', '.'] - [then] vs [then]
iter= 49000, avarage loss= 0.393655, avarage accuracy= 91.70
['.', 'but', 'who'] - [is] vs [is]
iter= 50000, avarage loss= 0.364555, avarage accuracy= 91.30
['.', 'this', 'proposal'] - [met] vs [met]
Optimization is finished...
Elapsed time:  28.85817804733912 min
Run on command line.
3 words: nobody escape to
nobody escape to young their common enemy . the cat . some said this . t
he enemy approaches us . now .
3 words: go to cat
```

GRU

Implementations…

GRU based Text Generation (TF v2): https://www.tensorflow.org/text/tutorials/text_generation





LSTMs

Implementations…

Predict MNIST using an RNN with Keras:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM #, CuDNNLSTM


mnist = tf.keras.datasets.mnist  # mnist is a dataset of 28x28 images of handwritten digits and their labels
(x_train, y_train),(x_test, y_test) = mnist.load_data()  # unpacks images to x_train/x_test and labels to y_train/y_test

x_train = x_train/255.0
x_test = x_test/255.0

print(x_train.shape)
print(x_train[0].shape)

model = Sequential()

# If you are running with a GPU, try out the CuDNNLSTM layer type instead
# (don't pass an activation, tanh is required)
model.add(LSTM(128, input_shape=(x_train.shape[1:]), activation='relu', return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(128, activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax'))
```

```
opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)

# Compile model
model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer=opt,
    metrics=['accuracy'],
)

model.fit(x_train,
          y_train,
          epochs=3,
          verbose=1,
          validation_data=(x_test, y_test))
```

with LSTM and Relu activation function:

```
Epoch 1/3 60000/60000 [==========] - 235s 4ms/step - loss: 0.6488 - acc: 0.7853 - val_loss: 0.1433 - val_acc: 0.9556
Epoch 2/3 60000/60000 [==========] - 228s 4ms/step - loss: 0.1672 - acc: 0.9539 - val_loss: 0.0906 - val_acc: 0.9740
Epoch 3/3 60000/60000 [==========] - 229s 4ms/step - loss: 0.1137 - acc: 0.9701 - val_loss: 0.0722 - val_acc: 0.9773
```

with LSTM and Tanh activation function:

```
Epoch 1/3 60000/60000 [==========] - 233s 4ms/step - loss: 0.3888 - acc: 0.8819 - val_loss: 0.1181 - val_acc: 0.9655
Epoch 2/3 60000/60000 [==========] - 229s 4ms/step - loss: 0.1172 - acc: 0.9688 - val_loss: 0.0846 - val_acc: 0.9749
Epoch 3/3 60000/60000 [==========] - 234s 4ms/step - loss: 0.0847 - acc: 0.9772 - val_loss: 0.0676 - val_acc: 0.9809
```

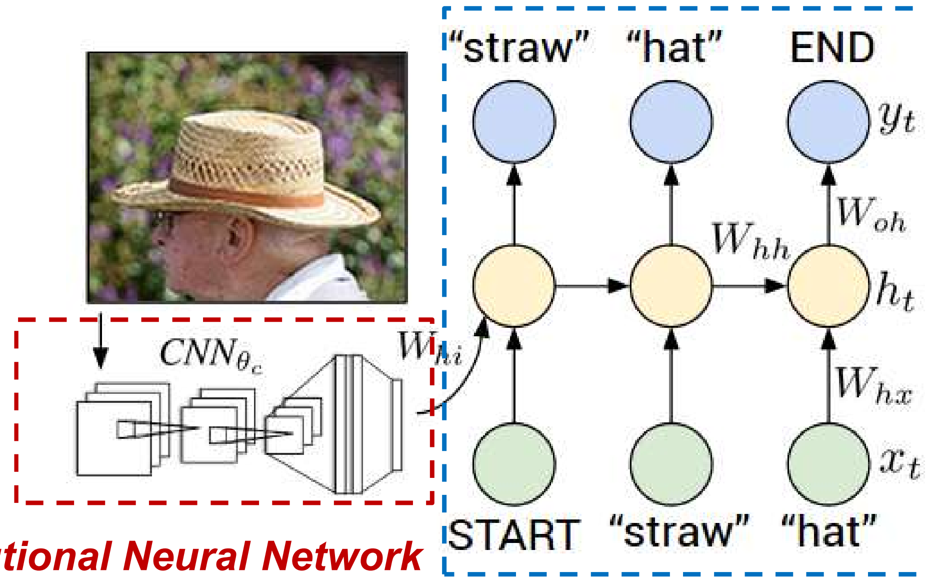
with CuDNNLSTM and Tanh activation function:

```
Epoch 1/3 60000/60000 [=========] - 27s 445us/step - loss: 0.3742 - acc: 0.8854 - val_loss: 0.1280 - val_acc: 0.9632
Epoch 2/3 60000/60000 [=========] - 25s 419us/step - loss: 0.1159 - acc: 0.9693 - val_loss: 0.0731 - val_acc: 0.9790
Epoch 3/3 60000/60000 [=========] - 25s 421us/step - loss: 0.0843 - acc: 0.9785 - val_loss: 0.0661 - val_acc: 0.9813
…
Epoch 10/10 60000/60000 [=======] - 25s 416us/step - loss: 0.0284 - acc: 0.9924 - val_loss: 0.0510 - val_acc: 0.9871
```

Relevant links:
https://pythonprogramming.net/recurrent-neural-network-deep-learning-python-tensorflow-keras/
https://www.tensorflow.org/guide/keras/working_with_rnns

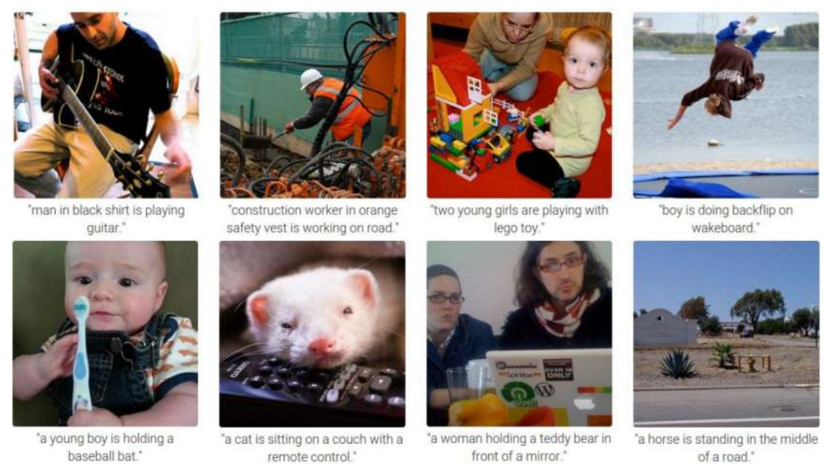
Image Captioning

Connecting Images and Natural Language
Andrej Karpathy
PhD Thesis, 2016
https://cs.stanford.edu/people/karpathy/main.pdf



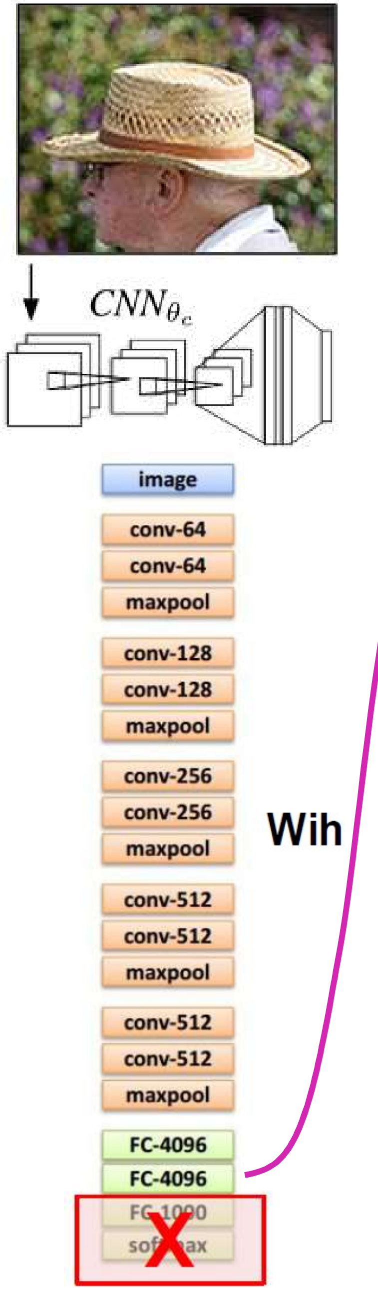
Recurrent Neural Network

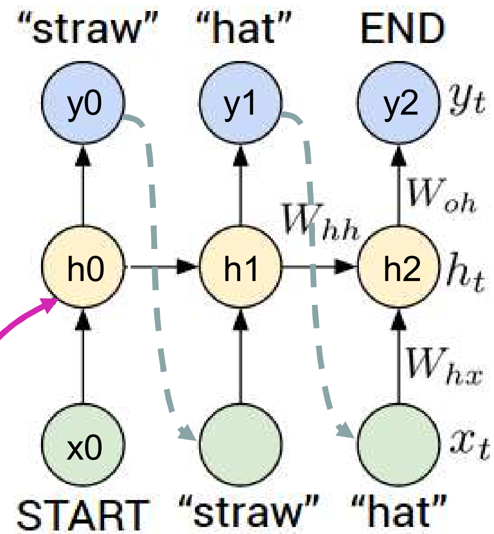
Convolutional Neural Network



Relevant links:
https://www.tensorflow.org/text/tutorials/image_captioning
https://arxiv.org/pdf/1412.2306v2.pdf
https://arxiv.org/pdf/1406.5679v1.pdf
https://arxiv.org/abs/1410.1090
https://arxiv.org/abs/1411.4555
https://arxiv.org/abs/1411.4389
https://arxiv.org/abs/1411.5654
https://cs.stanford.edu/people/karpathy/
https://cs.stanford.edu/people/karpathy/deepimagesent/

Image Captioning





before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih** * v)

Image Sentence Datasets (http://cocodataset.org/)

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.

Relevant links:
http://cs231n.stanford.edu/slides/2016/winter1516_lecture10.pdf

Image Captioning

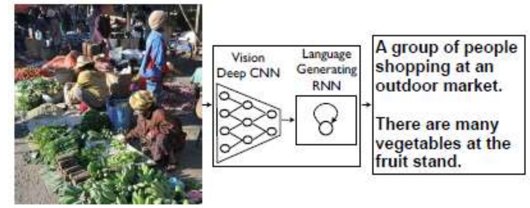
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

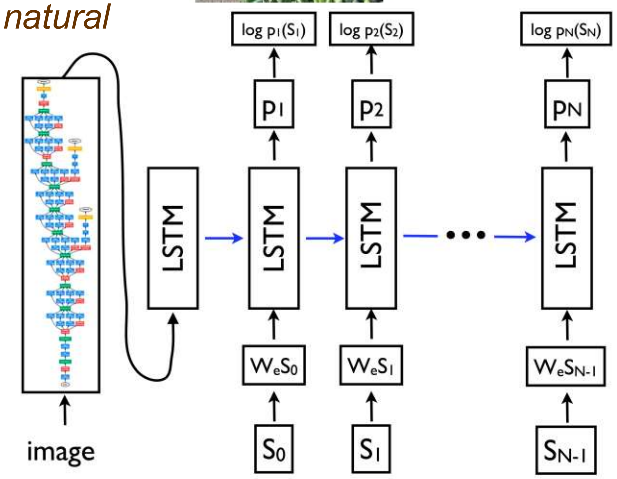
Computer Vision & NLP

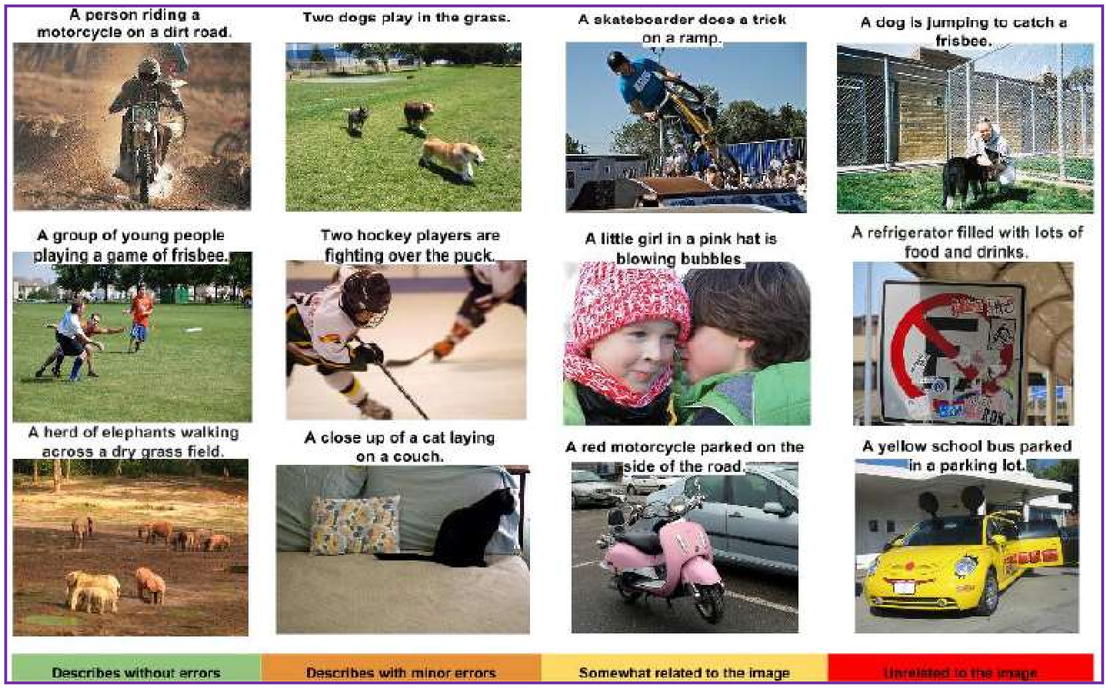
Show and Tell: A Neural Image Caption Generator

(by Oriol Vinyals et al., 2015) https://arxiv.org/abs/1411.4555

A generative model based on a deep recurrent architecture that combines recent advances in computer vision and machine translation to generate natural sentences describing an image.

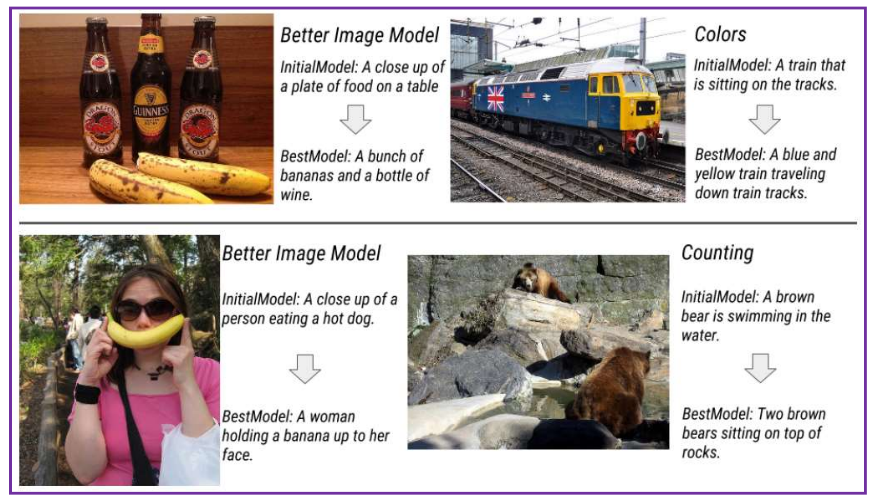






***Improved model:* Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge**

(by Oriol Vinyals et al., 2016) https://arxiv.org/abs/1609.06647



Computer Vision & NLP

Deep Visual-Semantic Alignments for Generating Image Descriptions

(by Andrej Karpathy and Fei-Fei Li, 2015)

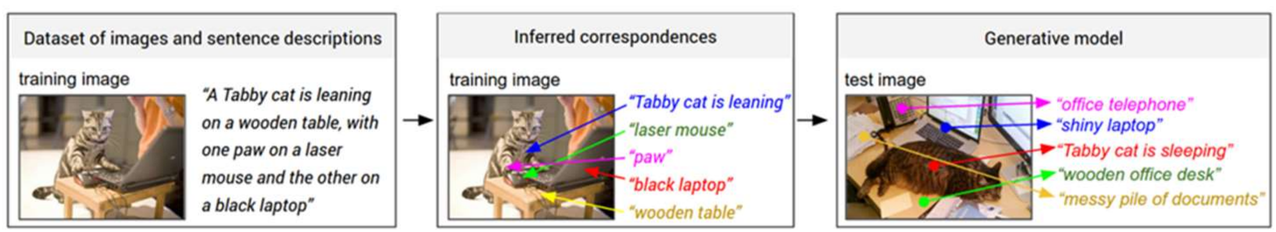
The paper looks into a combination of CNNs and bidirectional RNNs (Recurrent Neural Networks) to generate natural language descriptions of different image regions…

In contrast to traditional CNNs (where a single label associated with each image), the offered model has training examples that have a sentence (a weak label, where segments of the sentence refer to unknown parts of the image) associated with each image. Using this training data, a deep neural network "***infers the latent alignment between segments of the sentences and the region that they describe***"

Alignment The model is trained on compatible and incompatible image-sentence pairs, by accepting an image and a sentence as input, where the output is a score for how well they match.

Generation Having a dataset - a set of image regions (found by the RCNN) and corresponding text (thanks to the BRNN) as an output of Alignment step, the generation model is going to learn from that dataset in order to generate descriptions given an image. The model takes in an image and feeds it through a CNN. The softmax layer is disregarded as the outputs of the fully connected layer become the inputs to another RNN that forms probability distributions on the different words in a sentence.

Example output of the model



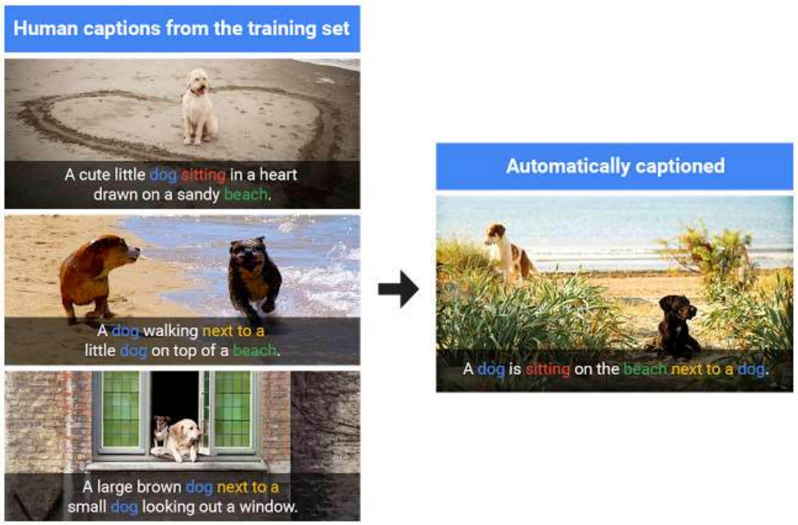
Workflow of alignment and generative model

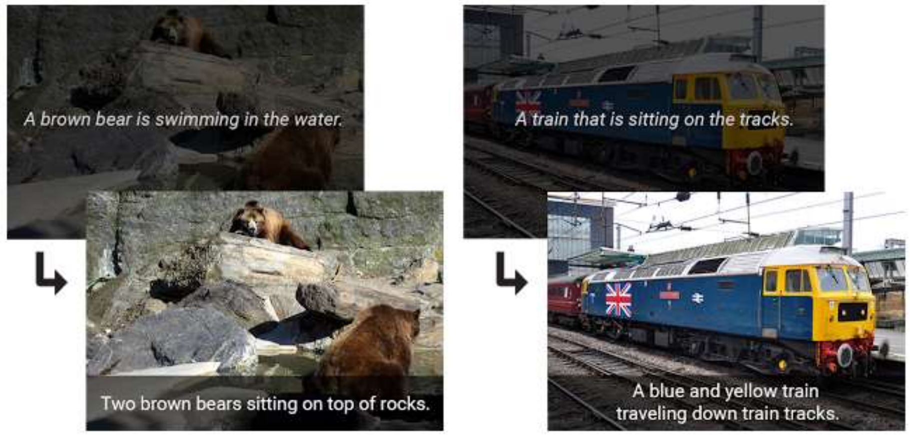
Relevant links:
https://arxiv.org/pdf/1412.2306v2.pdf
https://arxiv.org/pdf/1406.5679v1.pdf
https://cs.stanford.edu/people/karpathy/
https://cs.stanford.edu/people/karpathy/deepimagesent/

Computer Vision & NLP

Image Caption ***Implementation…***

- **NeuralTalk** *(Deprecated) https://github.com/karpathy/neuraltalk*
- **NeuralTalk2:** *is written in Torch and is SIGNIFICANTLY (~100x) faster because it is batched and runs on the GPU. It also supports CNN finetuning, which helps a lot with performance. But overall speed is slowed down because we also have to forward a VGGNet. However, overall very good models can be trained in 2-3 days, and they show a much better performance. https://github.com/karpathy/neuraltalk2*
- ***Show and Tell: A Neural Image Caption Generator*** *is a TensorFlow implementation of the image-to-text model described in the paper (http://arxiv.org/abs/1609.06647): https://github.com/tensorflow/models/tree/master/research/im2txt*
  Examples with pre-trained models: https://github.com/jmrf/im2txt-demo
  https://github.com/KranthiGV/Pretrained-Show-and-Tell-model
- ***Image captioning with visual attention*** *is a TensorFlow implementation inspired by previous architecture and been updated to use a 2-layer Transformer-decoder. https://www.tensorflow.org/tutorials/text/image_captioning*





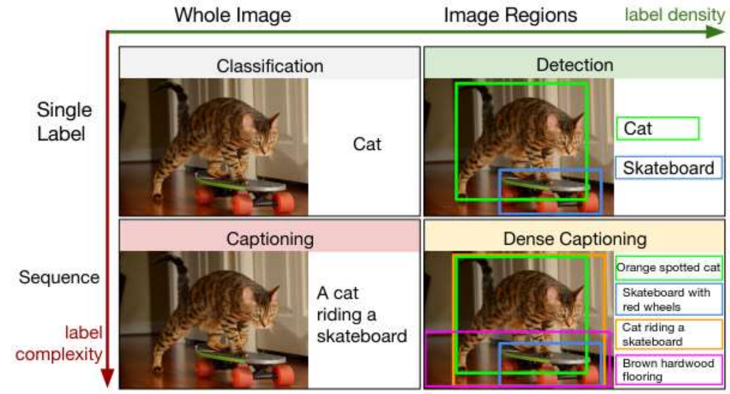
Left: the better image model allows the captioning model to generate more detailed and accurate descriptions. Right: after fine-tuning the image model, the image captioning system is more likely to describe the colors of objects correctly.

Relevant links:
https://research.googleblog.com/2016/09/show-and-tell-image-captioning-open.html

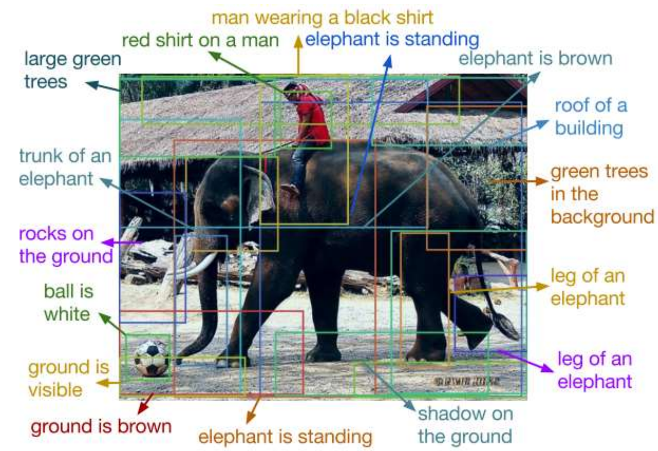
Computer Vision & NLP

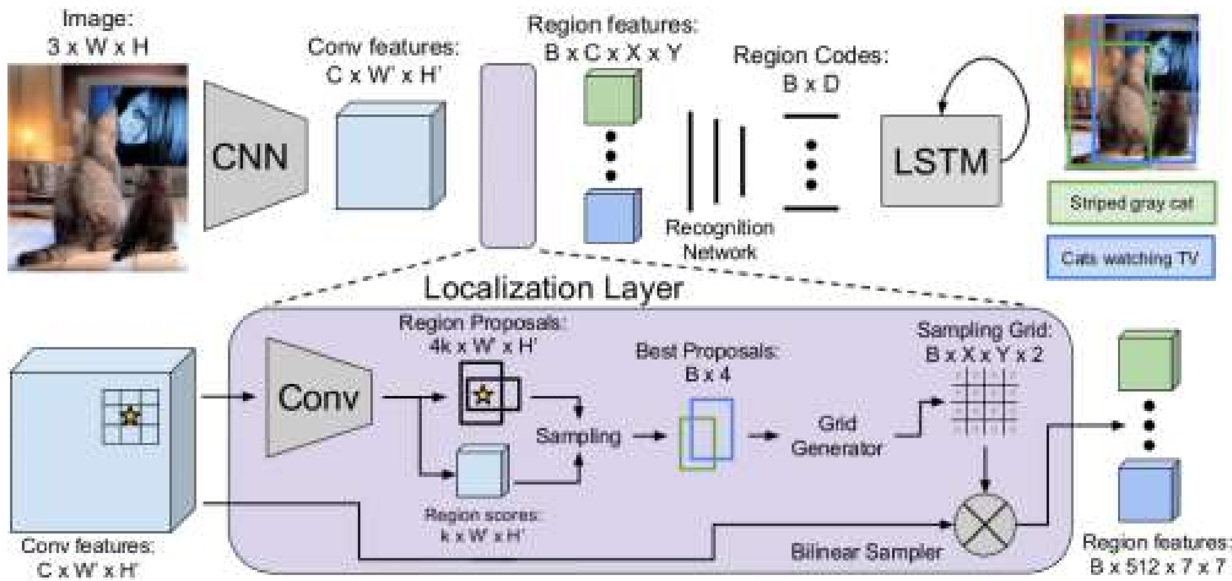
DenseCap: Fully Convolutional Localization Networks for Dense Captioning

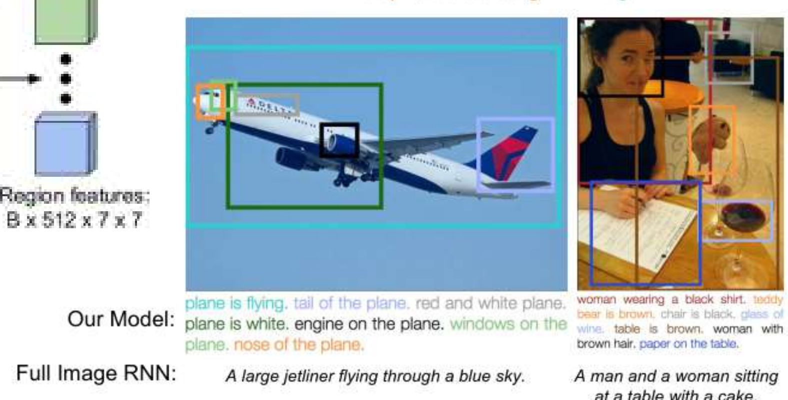
(by Justin Johnson, Andrej Karpathy and Fei-Fei Li, 2016)



The model efficiently identifies and captions all the things in an image with a single forward pass of a network. It is fully differentiable and trained end-to-end without any pipelines. The model is capable to process a 720x600 image in only 240ms, and evaluation on a large-scale dataset of 94,000 images and 4,100,000 region captions shows that it outperforms baselines based on previous approaches. Model is successfully applied for *image retrieval* as well as *region search* .





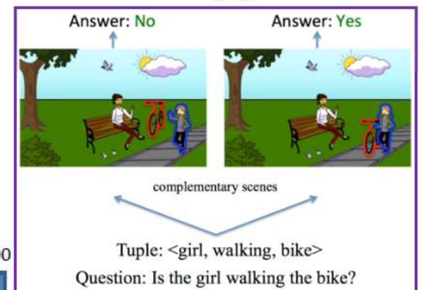


Relevant links:
https://cs.stanford.edu/people/karpathy/
https://cs.stanford.edu/people/karpathy/densecap/
https://cs.stanford.edu/people/karpathy/densecap.pdf

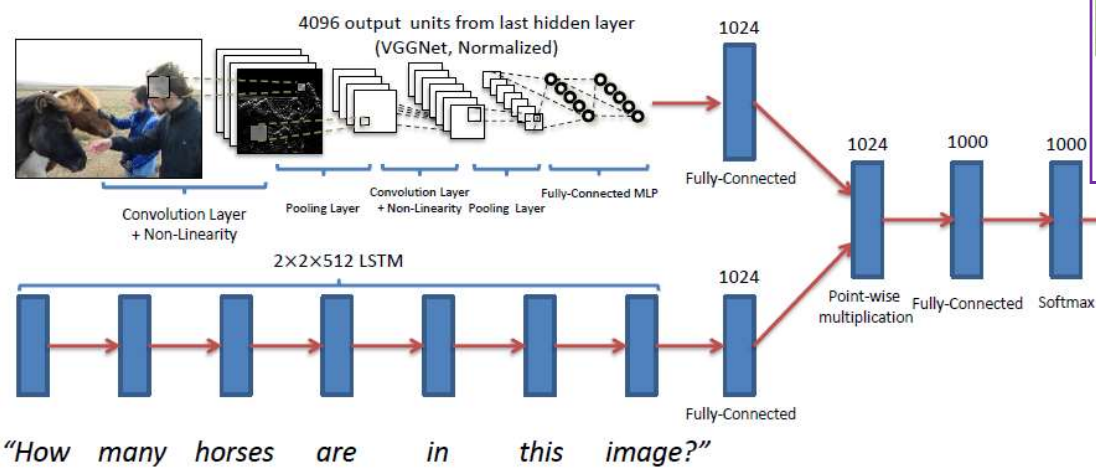
Computer Vision & NLP

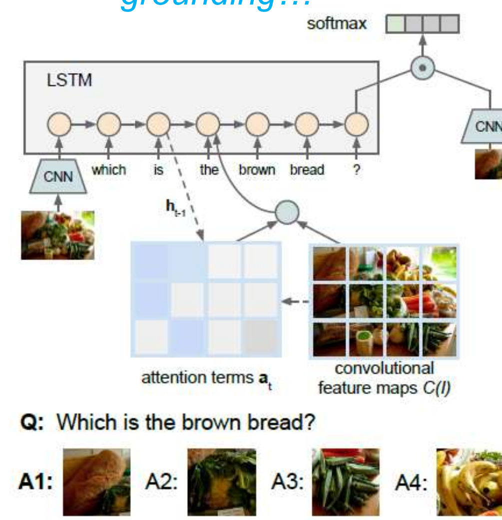
Visual Question Answering with deep image understanding

- **VQA: Visual Question Answering** *(Agrawal et al.,2015) http://arxiv.org/pdf/1505.00468v6.pdf*
- **Visual 7W: Grounded Question Answering in Images** *(Zhu et al., 2016) https://arxiv.org/pdf/1511.03416.pdf*
- **Balancing and Answering Binary Visual Questions** *(Yin and Yang, 2016) https://arxiv.org/pdf/1511.05099.pdf*
- **Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering** *(Goyal et al., 2017) https://arxiv.org/pdf/1612.00837.pdf*
- …



*From global association between QA sentences and images towards a **semantic link between textual descriptions and image regions** by object-level grounding…*

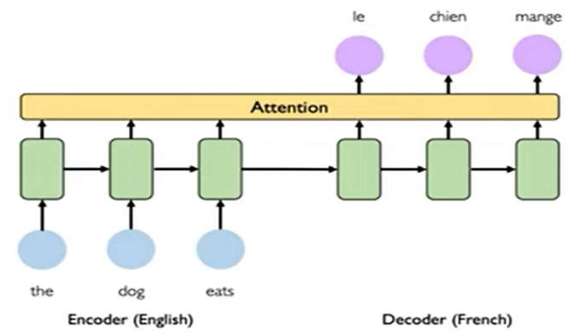


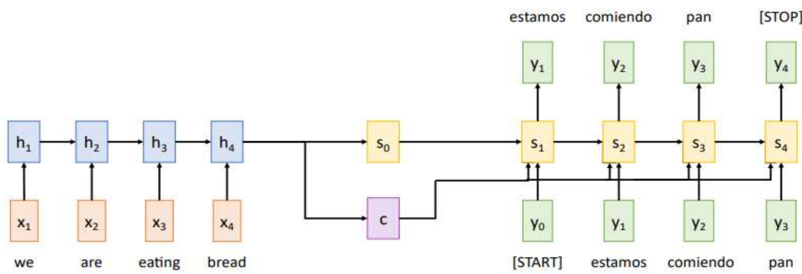


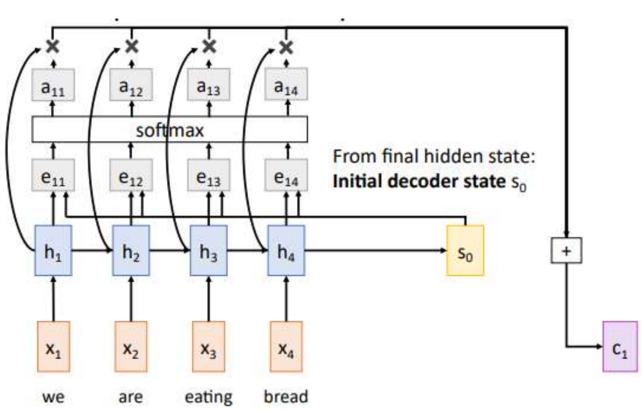
VQA dataset , link to VQA Challenge and other related materials could be found following the link http://www.visualqa.org/

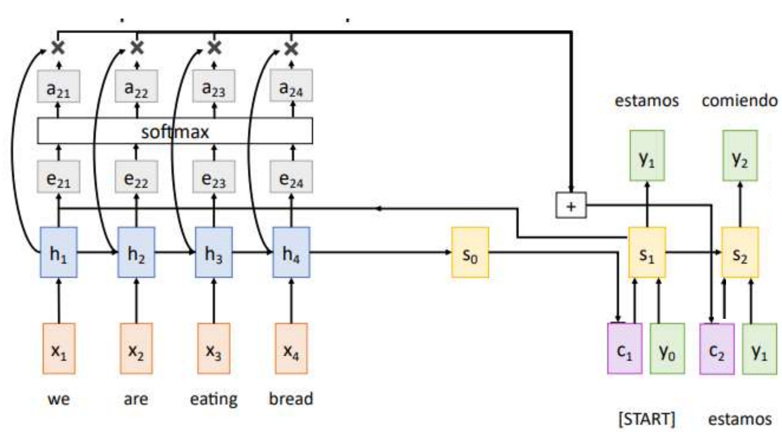
Attention mechanism in RNN

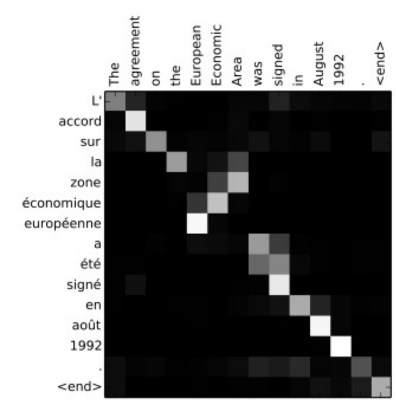
To improve memory in RNN, ***attention mechanisms*** *is used as a* ***learnable memory access****…*











Relevant links:
https://www.youtube.com/watch?v=qjrad0V0uJE
https://www.youtube.com/watch?v=YAgjfMR9R_M
https://tianguoguo.fun/2019/09/15/3-Neural-Machine-Translation-by-Jointly-Learning-to-Align-and-Translate%E8%AE%BA%E6%96%87%E5%A4%8D%E7%8E%B0%E4%BB%A3%E7%A0%81/

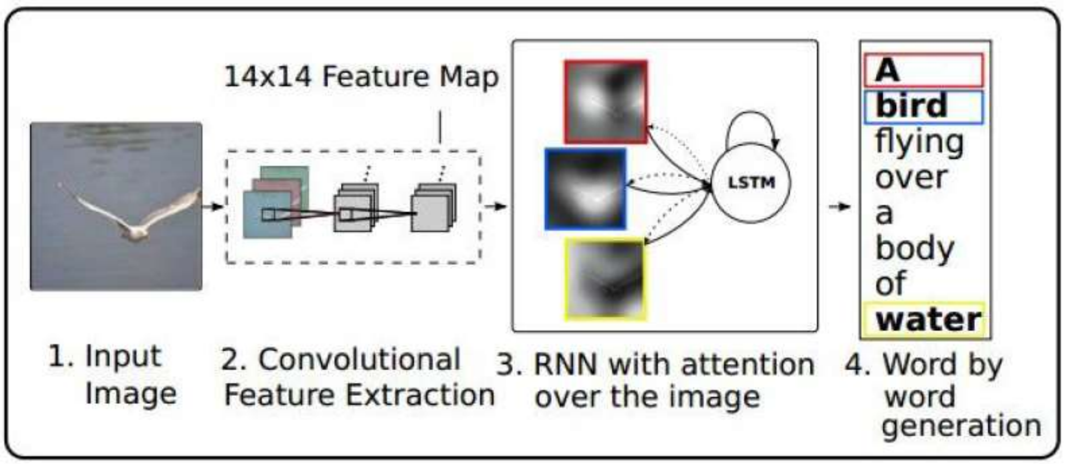
Computer Vision & NLP

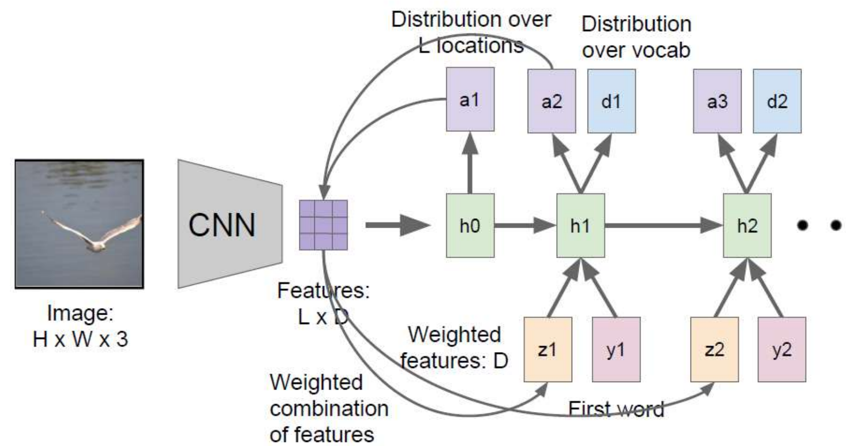
Show, Attend and Tell: *Neural Image Caption Generation with Visual Attention*

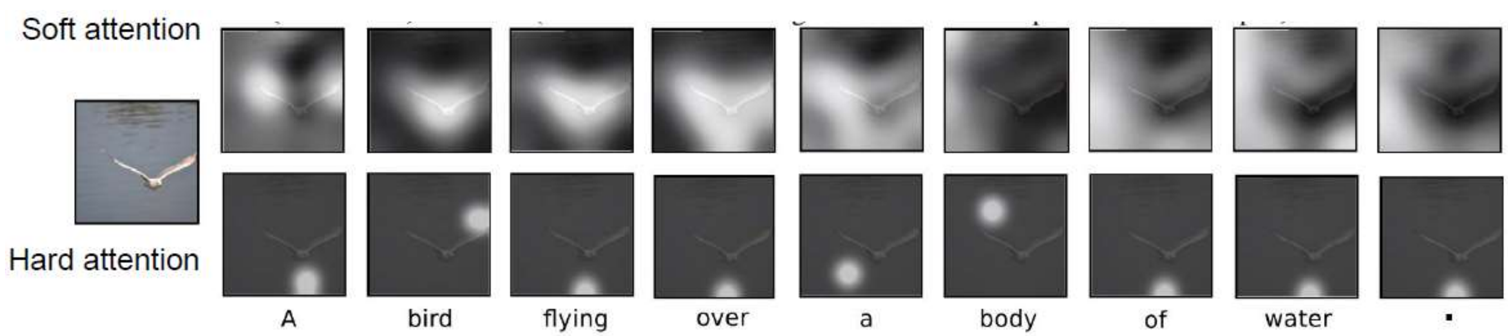
(by Xu et at., 2015)

Authors apply attention mechanisms to the problem of generating image descriptions. They use a Convolutional Neural Network to "encode" the image, and a Recurrent Neural Network with attention mechanisms to generate a description.

RNN attends spatially to different parts of images while generating each word of the sentence:





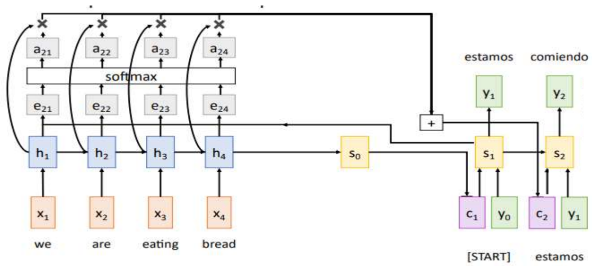


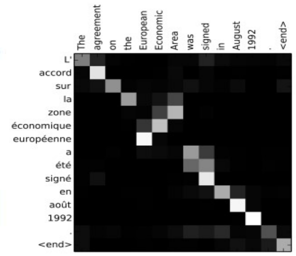
Relevant links:

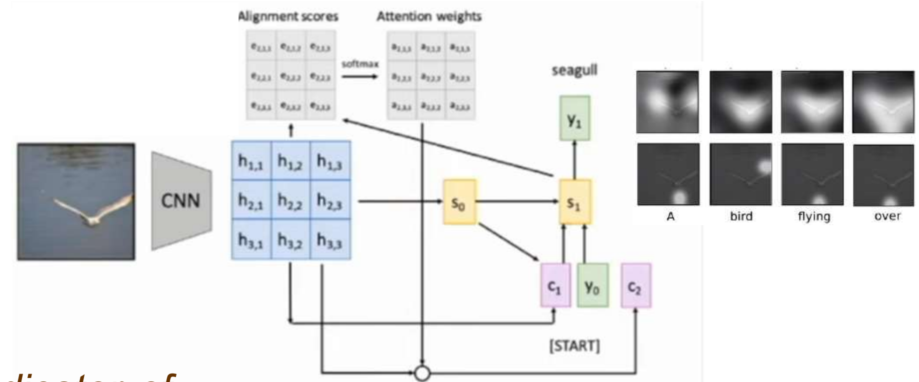
https://arxiv.org/abs/1502.03044

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

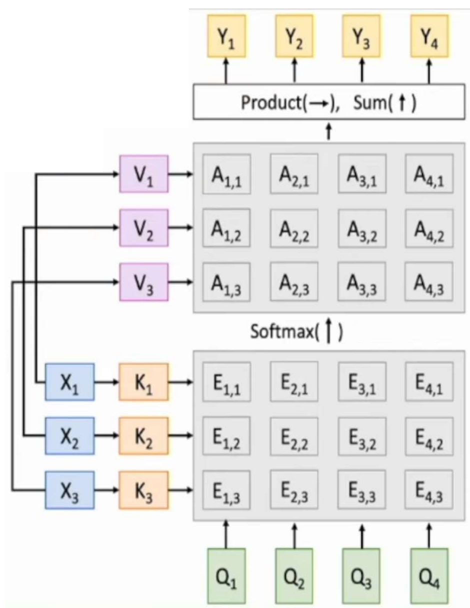
Generalization of Attention Mechanism

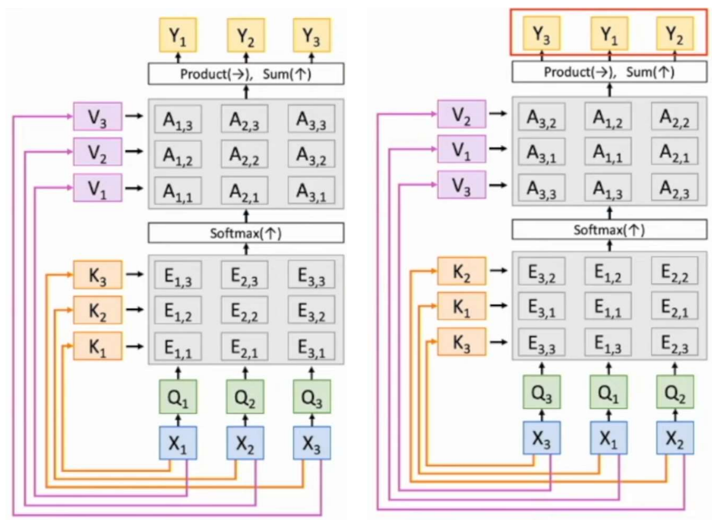




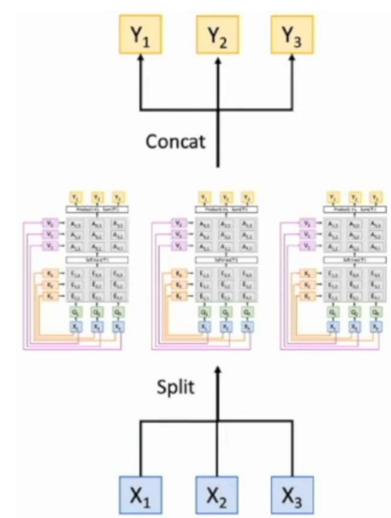


Sequence-2-Sequence ***attention layer*** *as an indicator of relevance of their elements…*





Self-attention layer *(one Query per Input vector)*
It is Permutation Equivariant type of network layers that operates on sets of vectors regardless of their order...

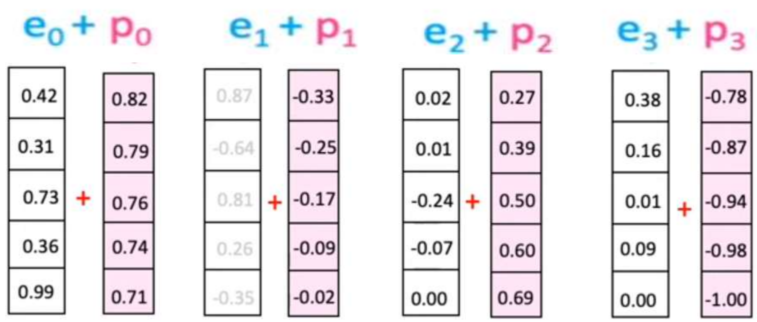


Multihead Self-attention layer *allows to catch several contexts in parallel and aggregate them in the output…*

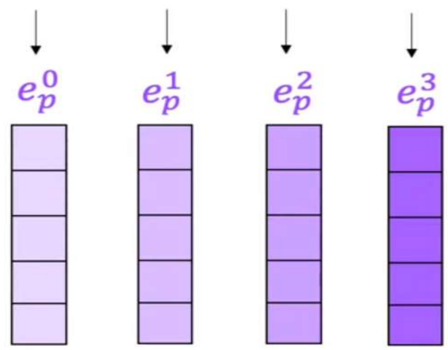
Relevant links:
https://www.youtube.com/watch?v=YAgjfMR9R_M

Generalization of Attention Mechanism

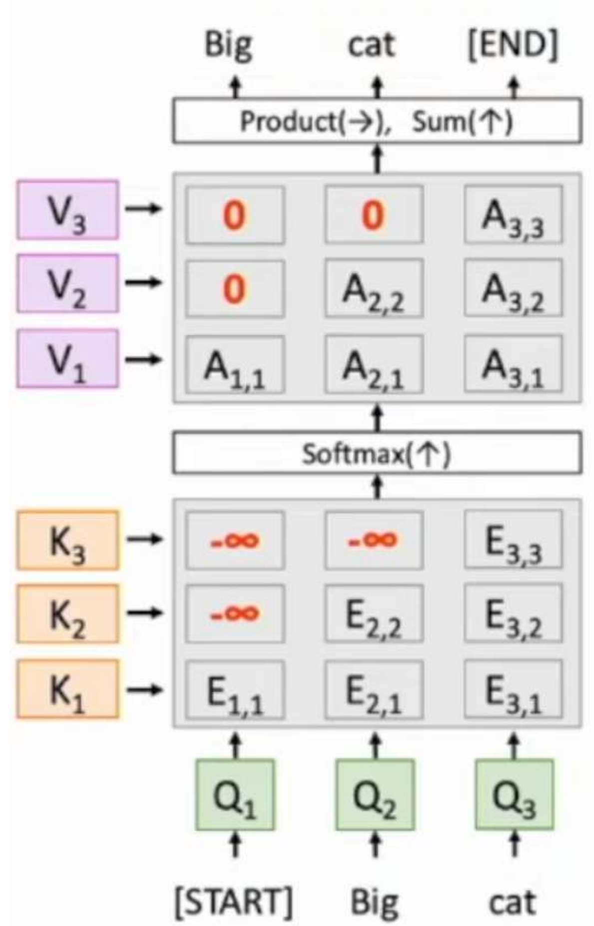
Positional Embedding *modifies input matrix in a way to encode the order of the elements by adding the position matrix of the same size as an input…*



$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \qquad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$



Masked Self-Attention Layer *considers an order of the elements in a sequence and does not let vectors "look ahead", using only information from the past…*



Relevant links:
https://www.youtube.com/watch?v=YAgjfMR9R_M
https://www.youtube.com/watch?v=dichIcUZfOw
https://arxiv.org/abs/1706.03762

Neural Attention Mechanism

Attention *(or neural attention) mechanism equips a neural network with the ability to focus on a subset of its inputs (or features).*

It is modeled by analogy to the way humans focus on a particular subset of their sensory input, and tune-out the rest.

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

Neural Machine Translation by Jointly Learning to Align and Translate. *Authors conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of the basic encoder-decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. (https://arxiv.org/abs/1409.0473)*

Recurrent Models of Visual Attention. *Authors present a novel recurrent neural network model that is capable of extracting information from an image or video by adaptively selecting a sequence of regions or locations and only processing the selected regions at high resolution. (https://arxiv.org/abs/1406.6247)*

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *Authors apply attention mechanisms to the problem of generating image descriptions. They use a Convolutional Neural Network to "encode" the image, and a Recurrent Neural Network with attention mechanisms to generate a description. (https://arxiv.org/abs/1502.03044)*

Teaching Machines to Read and Comprehend. *Authors use a RNN to read a text, read a (synthetically generated) question, and then produce an answer. By visualizing the attention matrix we can see where the networks "looks" while it tries to find the answer to the question. (https://arxiv.org/abs/1506.03340)*

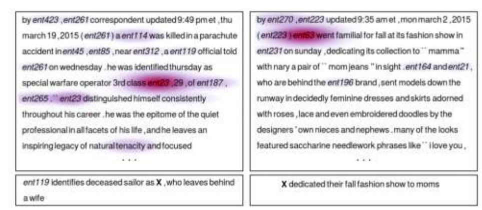
Reasoning about Entailment with Neural Attention. *Authors propose a neural model that reads two sentences to determine entailment using long short-term memory units. They extend this model with a word-by-word neural attention mechanism that encourages reasoning over entailments of pairs of words and phrases. (https://arxiv.org/abs/1509.06664)*

Attention-Based Models for Speech Recognition. *Authors extend the attention-mechanism with features needed for speech recognition (https://arxiv.org/abs/1506.07503)*

A Neural Attention Model for Abstractive Sentence Summarization. *Authors propose a fully data-driven approach to abstractive sentence summarization. The method utilizes a local attention-based model that generates each word of the summary conditioned on the input sentence. (https://arxiv.org/abs/1509.00685)*

Attention in:

- *Text Translation*
- *Image Descriptions*
- *Entailment (logical consequence)*
- *Speech Recognition*
- *Text Summarization*



Relevant links:

http://akosiorek.github.io/ml/2017/10/14/visual-attention.html
http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/
https://machinelearningmastery.com/attention-long-short-term-memory-recurrent-neural-networks/
https://distill.pub/2016/augmented-rnns/
https://www.youtube.com/watch?v=QuvRWevJMZ4

Intelligent Robots

*Microsoft and Alibaba **AI** programs **beat humans in** Stanford **reading** comprehension test for 1st time*

Machines can already outplay us in chess, poker and other games, and now they are becoming better readers as well.

*AI programs from both Microsoft and Alibaba outperformed humans in the beginning of January 2018 on a reading comprehension data set developed at Stanford - **The Stanford Question Answering Dataset (SQuAD)**. "Crowdworkers" scraped more than 500 Wikipedia articles to produce more than 100,000 question-and-answer sets for the test.*

Here's a sample question: "What year did Genghis Khan die?" (Spoiler alert: It's 1227.)

"This is the first time that a machine has outperformed humans on such a test," Alibaba said in a statement.

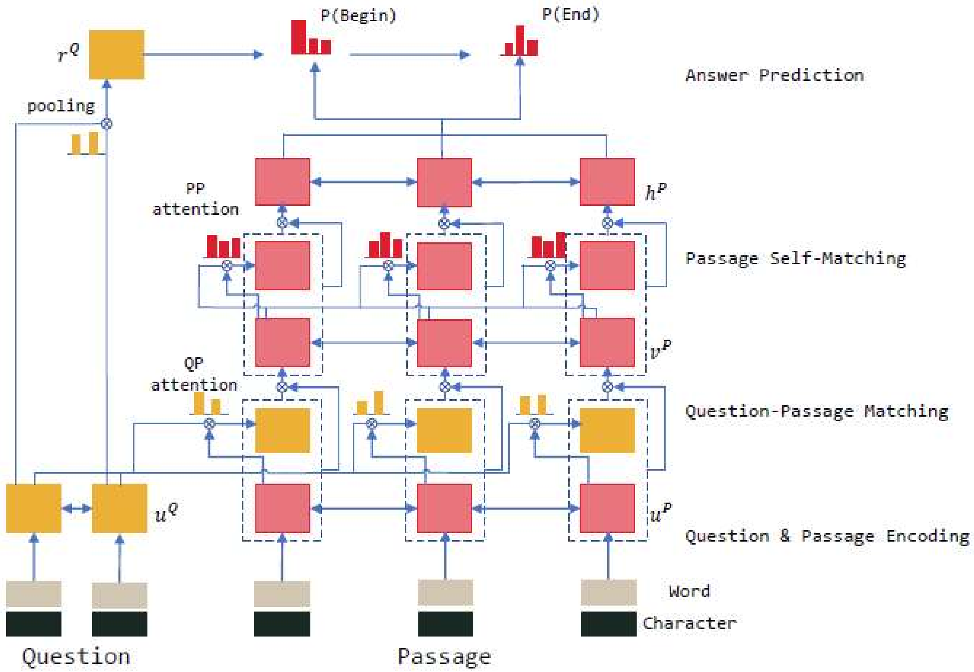
*Microsoft's score of **82.6** and Alibaba's grade of **82.4** beat out the human standard of **82.3**. Other notable AI programs participating in the test and closing in on beating human scores come from the Allen Institute for Artificial Intelligence, Tencent, Salesforce and others.*

Relevant links:
https://www.microsoft.com/en-us/research/publication/mrc/
https://codeburst.io/understanding-r-net-microsofts-superhuman-reading-ai-23ff7ededd96
https://www.geekwire.com/2018/microsoft-alibaba-ai-programs-beat-humans-stanford-reading-test-1st-time/

R-NET

R-NET: *Machine Reading Comprehension with Self-matching Networks*

- *R-Net mainly utilizes **RNNs** (more specifically, Gated Recurrent Units (GRN)) to simulate the action of 'reading' a passage of text…*
- *R-Net utilizes **Attention** to highlight some part of the text, under the context of another*



Given a passage P:

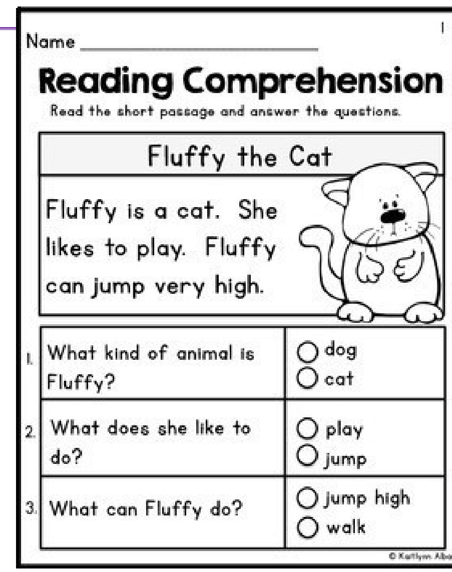
"Tesla was born on 10 July [O.S. 28 June] 1856 into a Serb family in the village of Smiljan, Austrian Empire (modern-day Croatia). His father, Milutin Tesla, was a Serbian Orthodox priest. Tesla's mother, Đuka Tesla (née Mandić), whose father was also an Orthodox priest, had a talent for making home craft tools, mechanical appliances, and the ability to memorize Serbian epic poems. Đuka had never received a formal education. Nikola credited his eidetic memory and creative abilities to his mother's genetics and influence. Tesla's progenitors were from western Serbia, near Montenegro."

Question Q:

"What were Tesla's mother's special abilities?"

Continuous 'span' of text as the answer A:

"making home craft tools, mechanical appliances, and the ability to memorize Serbian epic poems"


Name ______________________

Reading Comprehension

Read the short passage and answer the questions.

Fluffy the Cat

Fluffy is a cat. She likes to play. Fluffy can jump very high.

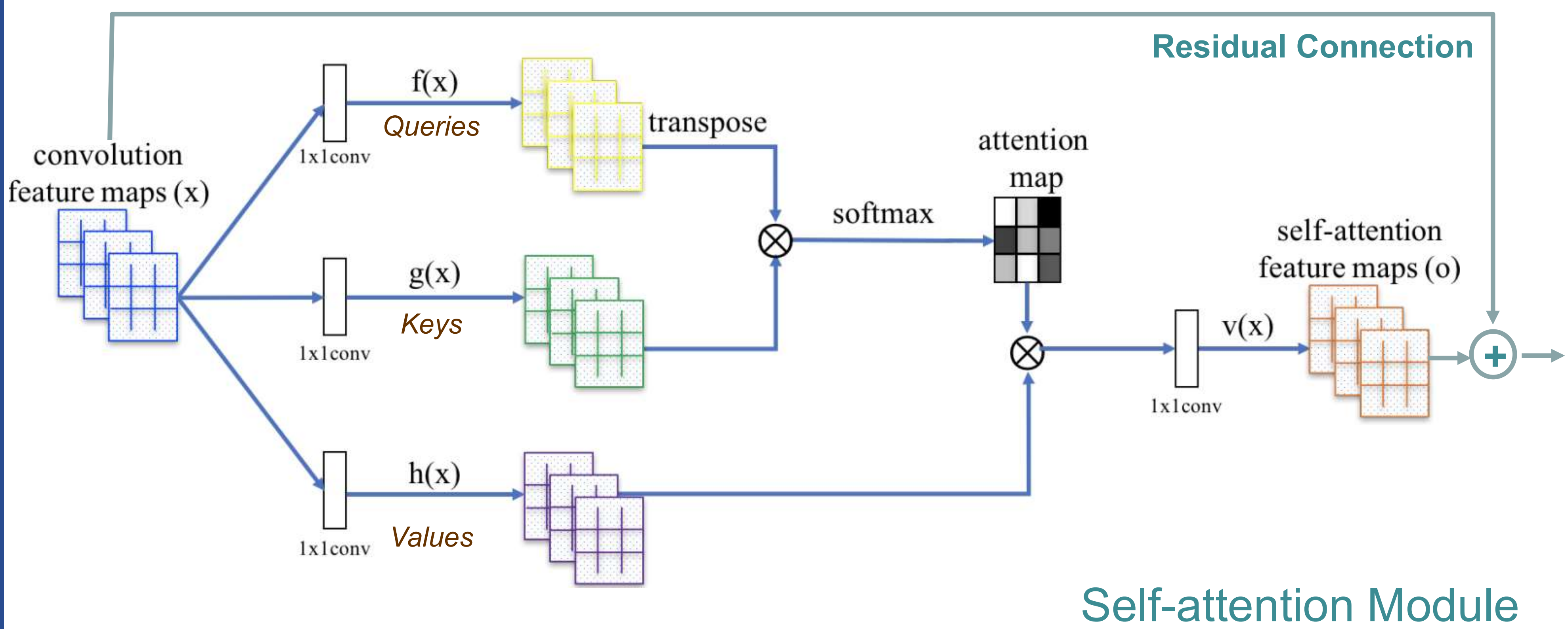
1.	What kind of animal is Fluffy?	○ dog ○ cat
2.	What does she like to do?	○ play ○ jump
3.	What can Fluffy do?	○ jump high ○ walk

Relevant links:

https://www.microsoft.com/en-us/research/publication/mrc/
https://codeburst.io/understanding-r-net-microsofts-superhuman-reading-ai-23ff7ededd96
https://www.geekwire.com/2018/microsoft-alibaba-ai-programs-beat-humans-stanford-reading-test-1st-time/

Generalization of Attention Mechanism

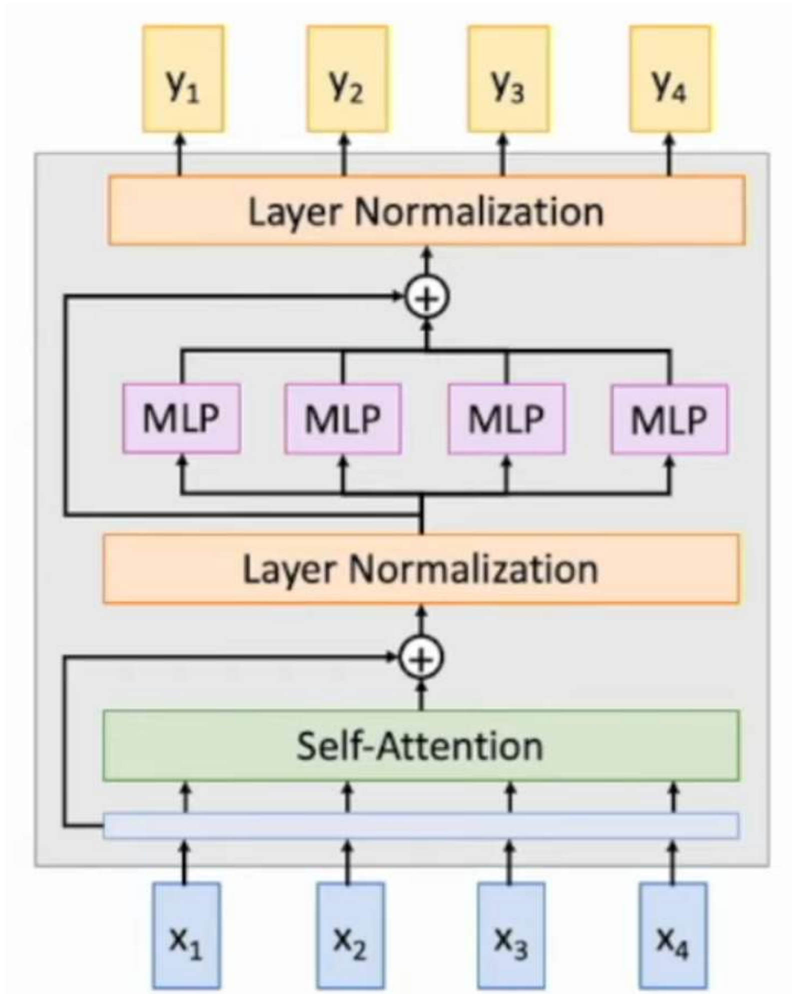
CNN with self-attention…



Relevant links:
https://arxiv.org/abs/1805.08318
https://paperswithcode.com/method/sagan

Generalization of Attention Mechanism

Transformer block…



Computation

- *independently*
- *in parallel*

Communication

- *process all-at-once*
- *capture relationships*

Relevant links:
https://arxiv.org/abs/1706.03762
https://www.youtube.com/watch?v=YAgjfMR9R_M
arxiv.org/abs/2012.14913

Transformer

The core idea behind the **Transformer model** is self-attention — the ability to attend to different positions of the input sequence to compute a representation of that sequence. Transformer creates stacks of self-attention layers via *Scaled Dot Product Attention* and *Multi-Head Attention*. *https://arxiv.org/abs/1706.03762*

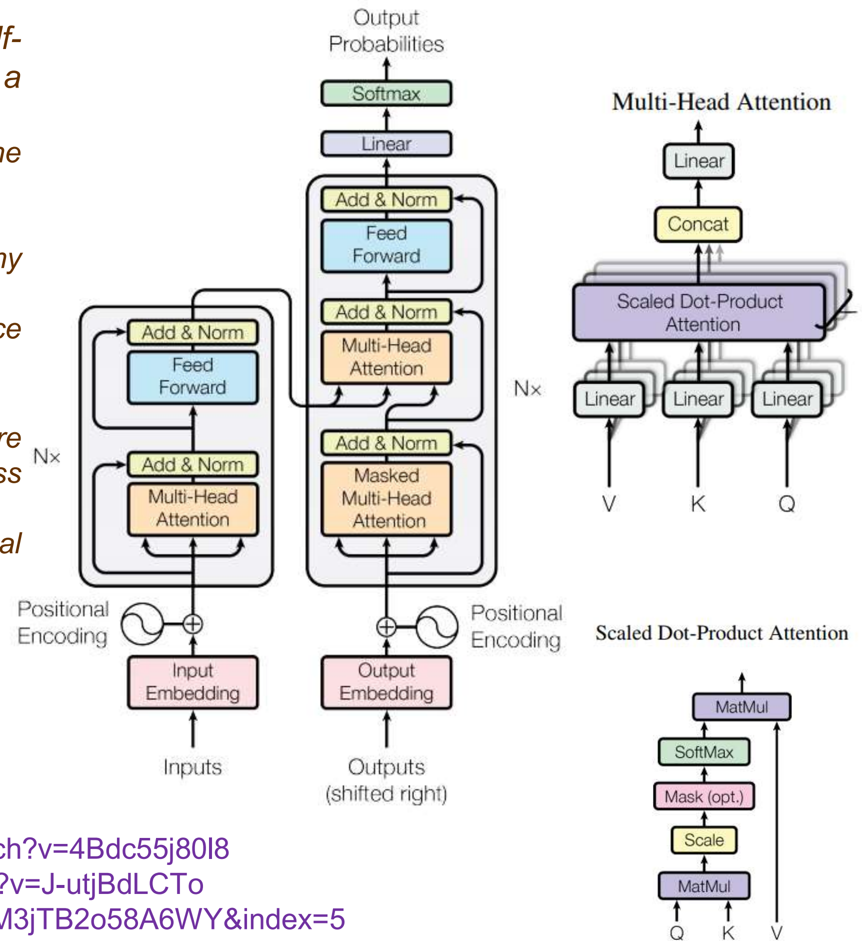
A transformer model handles variable-sized input using stacks of self-attention layers instead of RNNs or CNNs. This general architecture has a number of advantages:

- *It makes no assumptions about the temporal/spatial relationships across the data. This is ideal for processing a set of objects.*
- *Layer outputs can be calculated in parallel, instead of a series like an RNN.*
- *Distant items can affect each other's output without passing through many RNN-steps, or convolution layers.*
- *It can learn long-range dependencies. This is a challenge in many sequence tasks.*

The downsides of this architecture are:

- *For a time-series, the output for a time-step is calculated from the entire history instead of only the inputs and current hidden-state. This may be less efficient.*
- *If the input does have a temporal/spatial relationship, like text, some positional encoding must be added, or the model will effectively see a bag of words.*

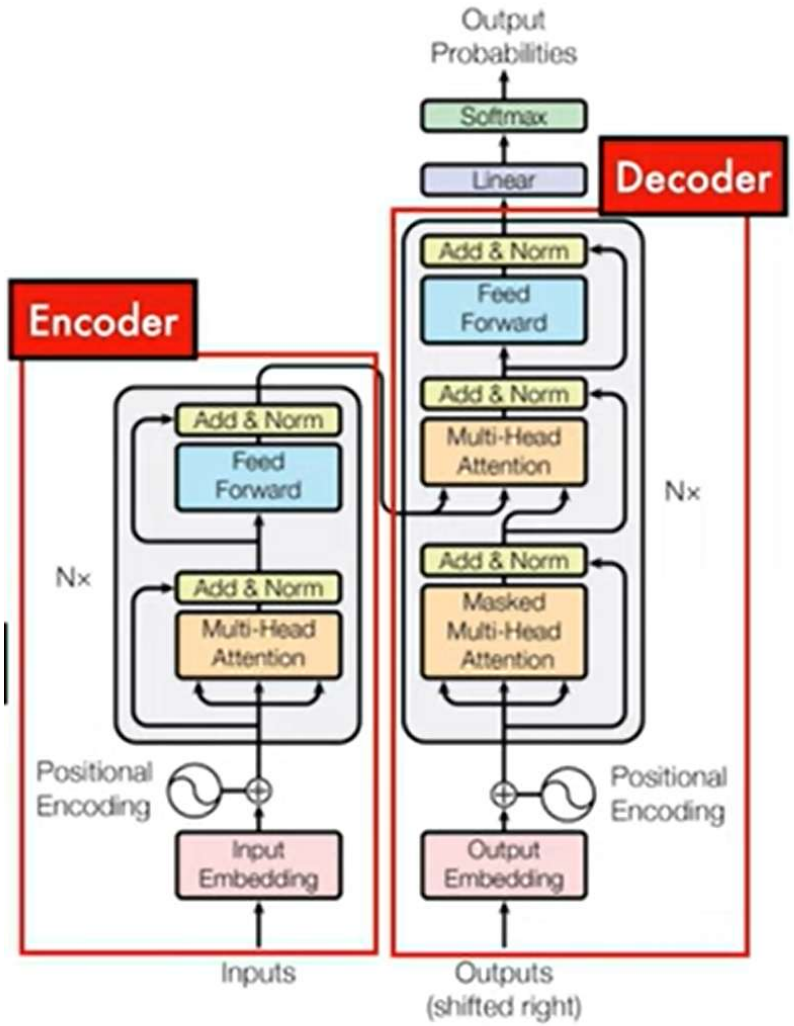
Transformers *provides thousands of pretrained models (e.g. BERT, GPT-2, GPT-3, ELMo, T5, etc.) to perform tasks on texts such as classification, information extraction, question answering, summarization, translation, text generation, etc. in 100+ languages. Its aim is to make cutting-edge NLP easier to use for everyone.* https://transformer.huggingface.co/ , https://github.com/huggingface/transformers



Relevant links:

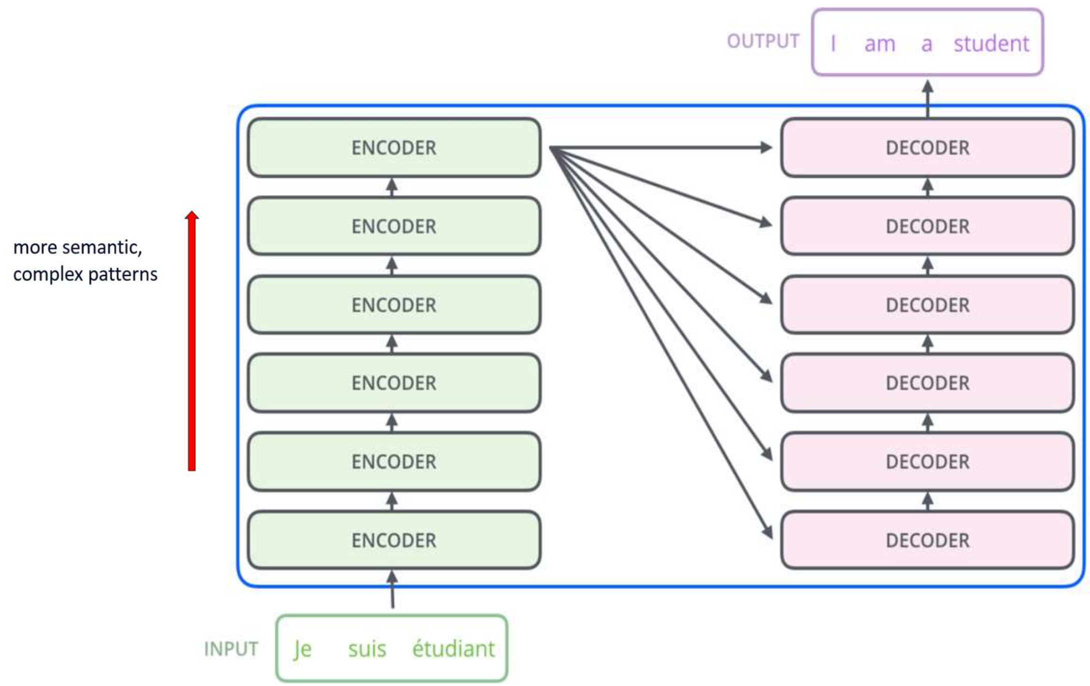
https://www.tensorflow.org/text/tutorials/transformer
https://www.youtube.com/watch?v=S27pHKBEp30 , https://www.youtube.com/watch?v=4Bdc55j80l8
https://www.youtube.com/watch?v=dichIcUZfOw , https://www.youtube.com/watch?v=J-utjBdLCTo
https://www.youtube.com/watch?v=6tzn5-XlhwU&list=PLaJCKi8Nk1hwaMUYxJMiM3jTB2o58A6WY&index=5
https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04
http://jalammar.github.io/illustrated-transformer/

Transformer

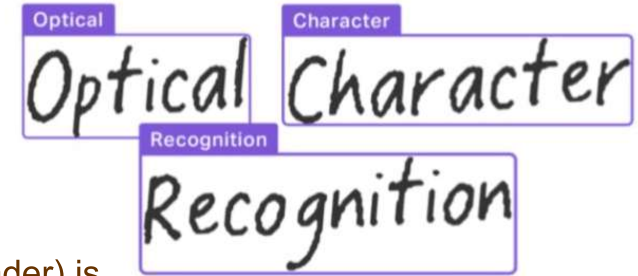


Encoder: *learns useful representations of input*
Decoder: *decodes encoded representation and combines with other input to predict output*

Encoder-Only*: used for learning representation (e.g. BERT)*
Decoder-Only*: used for generation tasks (e.g. GPT)*
Encoder-Decoder*: used for sequence-to-sequence*

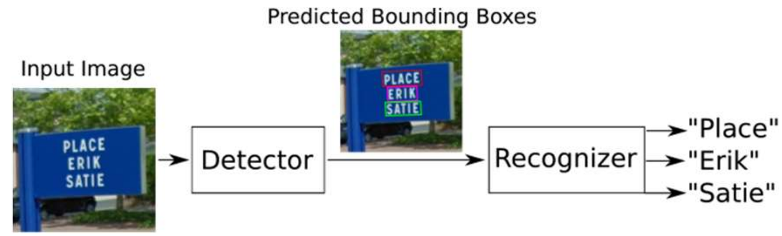


Relevant links:
arxiv.org/abs/2012.14913



OCR

OCR (optical character recognition or optical character reader) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image.

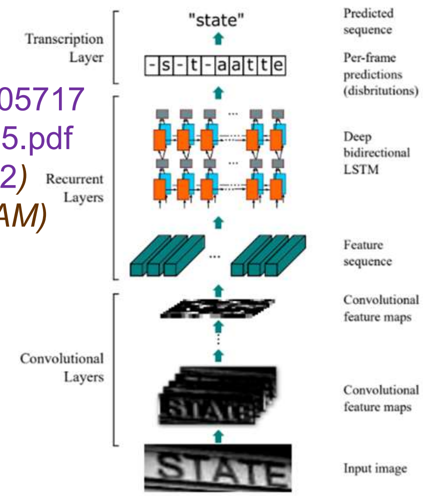


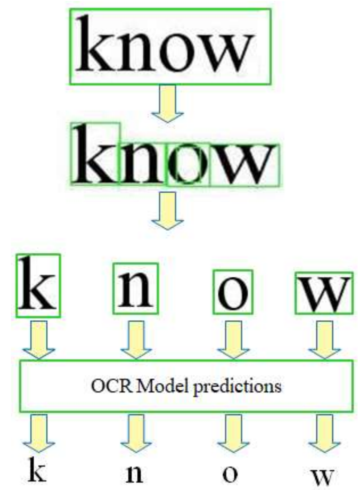
Text Detection:

- *Faster R-CNN, Mask R-CNN, R-FCN, SSD, YOLO, etc.*

Text Recognition:

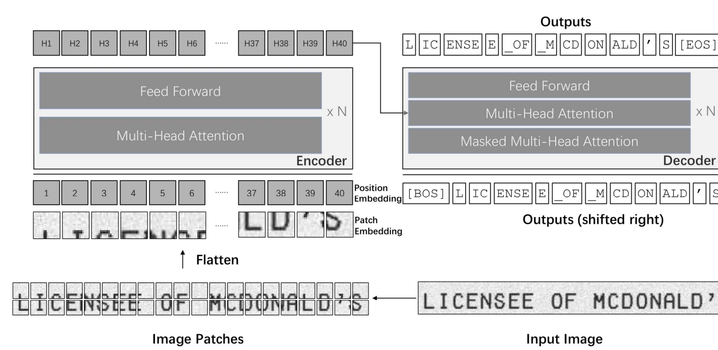
- *Convolutional Recurrent Neural Network (CRNN)* https://arxiv.org/abs/1507.05717
- *EAST (Efficient accurate scene text detector)* https://arxiv.org/pdf/1704.03155.pdf
- *Transformer based approaches (e.g. TrOCR* https://arxiv.org/abs/2109.10282*)*
- *Recurrent Attention Model (RAM) and Deep Recurrent Attention Model (DRAM)*
- *Attention OCR (Tensorflow)*
- *Tesseract OCR* https://github.com/tesseract-ocr/tesseract
- *Variety of online platforms*





Keras-OCR: https://keras-ocr.readthedocs.io/en/latest/
https://github.com/faustomorales/keras-ocr

Relevant links:
https://labelyourdata.com/articles/ocr-with-deep-learning/
https://nanonets.com/blog/deep-learning-ocr/
https://medium.com/saarthi-ai/how-to-build-your-own-ocr-a5bb91b622ba
https://nanonets.com/blog/attention-ocr-for-text-recogntion/
https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa
https://www.pyimagesearch.com/2020/08/17/ocr-with-keras-tensorflow-and-deep-learning/
https://keras.io/examples/vision/captcha_ocr/
https://github.com/microsoft/unilm/tree/master/trocr
https://huggingface.co/docs/transformers/model_doc/trocr



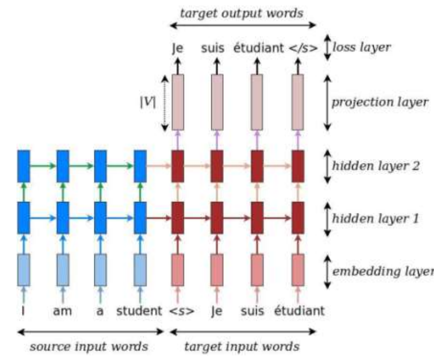
Implementation Use-Cases…

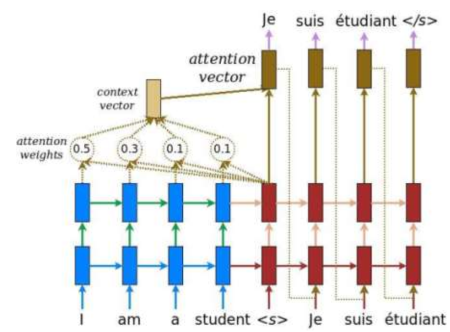
RNNs

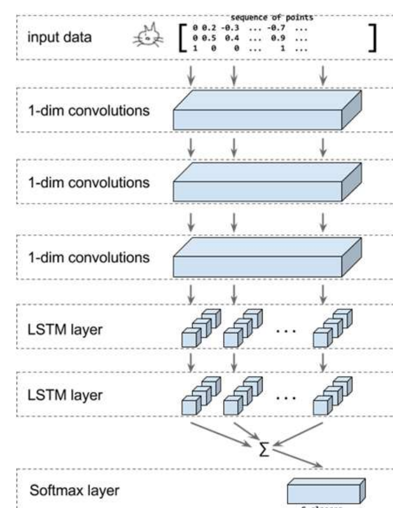
- ***Language Modeling***:
  - *https://www.tensorflow.org/text/tutorials/text_generation*
  - *https://adventuresinmachinelearning.com/keras-lstm-tutorial/*
- ***Machine Translation***:
  - *https://www.tensorflow.org/text/tutorials/nmt_with_attention*
  - *https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/21_Machine_Translation.ipynb*
  - *https://github.com/tensorflow/nmt/*
- ***Transformer based model***:

  NLP:
  - *https://www.tensorflow.org/text/tutorials/transformer*
  - *https://platform.openai.com/docs/guides/fine-tuning*
  - *As an option you may search and study some particular Transformer model with practical examples*

  Vision (e.g. image classification or object detection / segmentation):
  - *https://keras.io/examples/vision/image_classification_with_vision_transformer/*
  - *https://keras.io/examples/vision/vivit/*
- ***Audio Recognition***:
  - *https://www.tensorflow.org/tutorials/audio/simple_audio*
  - *https://www.tensorflow.org/tutorials/audio/transfer_learning_audio*
  - *https://www.tensorflow.org/tutorials/audio/music_generation*
- ***Time Series Prediction***:
  - *https://www.tensorflow.org/tutorials/structured_data/time_series*
  - *https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/*
  - *https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/*
  - *https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-networks-python/*
  - *https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/23_Time-Series-Prediction.ipynb*
- ***Text Classification*** and ***Sentiment Analysis***:
  - *https://www.tensorflow.org/tutorials/text/text_classification_rnn*
  - *https://www.tensorflow.org/tutorials/text/classify_text_with_bert*
  - *https://www.tensorflow.org/tfmodels/nlp/fine_tune_bert*
  - *https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/20_Natural_Language_Processing.ipynb*
- ***Image Captioning***:
  - *https://www.tensorflow.org/tutorials/text/image_captioning*
  - *https://keras.io/examples/vision/image_captioning/*
  - *https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/22_Image_Captioning.ipynb*







RNNs

Relevant materials…

- *A friendly introduction to Recurrent Neural Networks:*
  - https://www.youtube.com/watch?v=UNmqTiOnRfg
  - https://www.youtube.com/watch?v=WCUNPb-5EYI
- *Collection of RNN related publications:* http://people.idsia.ch/~juergen/rnn.html
- *Sequence Modeling: Recurrent and Recursive Nets (Book Chapter):* http://www.deeplearningbook.org/contents/rnn.html
- *Text handling with TensorFlow:* https://www.tensorflow.org/tutorials/load_data/text
- *Word Vector Representations (embeddings) :*
  - https://www.youtube.com/watch?v=ERibwqs9p3
  - https://www.youtube.com/watch?v=ASn7ExxLZws
  - https://www.youtube.com/watch?v=QyrUentbkvw
- *Image Captioning:*
  - https://blog.paperspace.com/image-captioning-with-tensorflow/
  - https://towardsdatascience.com/image-captions-with-attention-in-tensorflow-step-by-step-927dad3569fa
- *Dissecting BERT:*
  - https://medium.com/dissecting-bert
  - https://towardsdatascience.com/bert-to-the-rescue-17671379687f
  - https://medium.com/swlh/simple-transformers-multi-class-text-classification-with-bert-roberta-xlnet-xlm-and-8b585000ce3a
- *Fine-tuning GPT-3:*
  - *https://platform.openai.com/docs/guides/fine-tuning*
  - https://towardsdatascience.com/unleashing-the-power-of-gpt-how-to-fine-tune-your-model-da35c90766c4