BBM 495
RECURRENT NEURAL NETWORKS (RNNs)

LECTURER: BURCU CAN

2019-2020 SPRING

REFRESHER FROM LAST WEEK

- Computation graph
- Expressions (nodes in the graph)
- Parameters, LookupParameters
- Model (a collection of parameters)
- Trainers
- Create a graph for each example, then compute loss, backprop, update.

DyNET

LANGUAGE MODELS

A language model computes a probability for a sequence of words: $P(w_1, \ldots, w_T)$

- Useful for machine translation
  - Word ordering:
    p(the cat is small) > p(small the is cat)
  - Word choice:
    p(walking home after school) > p(walking house after school)

TRADITIONAL LANGUAGE MODELS

- Probability is usually conditioned on window of n previous words
- An incorrect but necessary Markov assumption!

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

- To estimate probabilities, compute for unigrams and bigrams (conditioning on one/two previous word(s):

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

What is the problem with this model?

TRADITIONAL LANGUAGE MODELS

- Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff (e.g. if 4-gram not found, try 3-gram, etc)
- There are A LOT of n-grams!
  → Gigantic RAM requirements!
- Recent state of the art: *Scalable Modified Kneser-Ney Language Model Estimation* by Heafield et al.:
  "Using one machine with 140 GB RAM for 2.8 days, we built an unpruned model on 126 billion tokens"

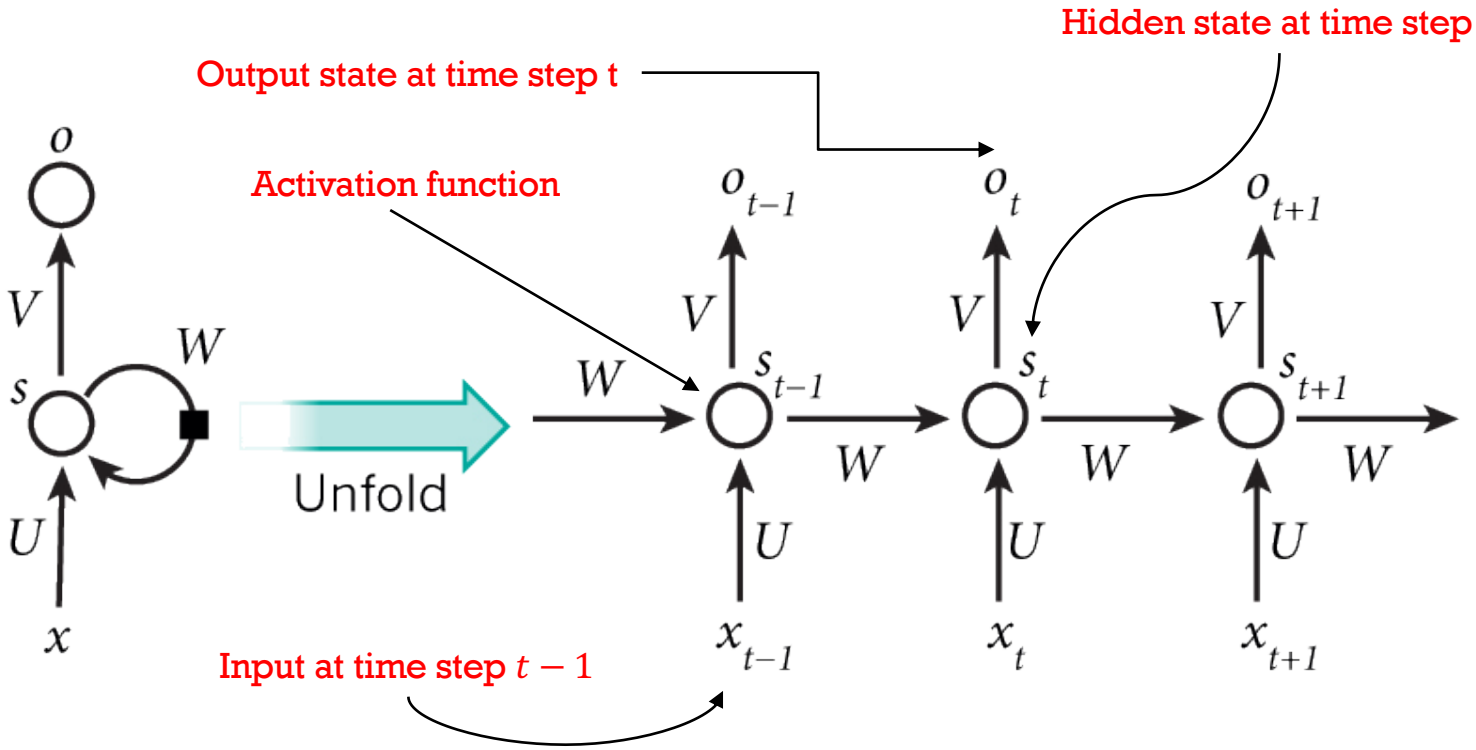
MOTIVATION

- Humans don't start their thinking from scratch every second
  - Thoughts have persistence
- Traditional neural networks can't characterize this phenomena
  - Ex: classify what is happening at every point in a movie
  - How a neural network can inform later events about the previous ones
- Recurrent neural networks address this issue
- How?

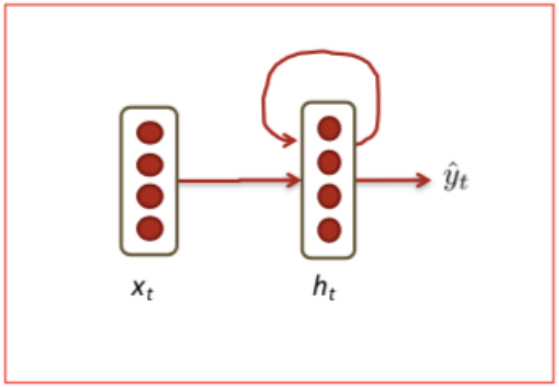
WHAT ARE RNNS?

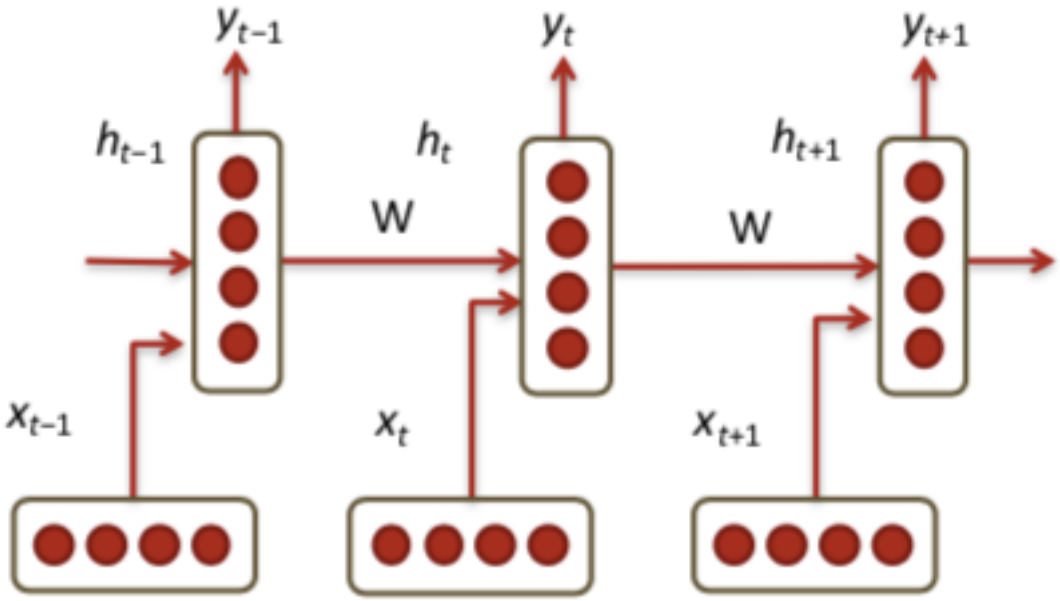
- Main idea is to make use of sequential information
- How RNN is different from neural network?
  - Vanilla neural networks assume all inputs and outputs are independent of each other
  - But for many tasks, that's a very bad idea
- What RNN does?
  - Perform the same task for every element of a sequence (that's what recurrent stands for)
  - Output depends on the previous computations!
- Another way of interpretation – RNNs have a "memory"
  - To store previous computations

RECURRENT NEURAL NETWORKS



RECURRENT NEURAL NETWORKS

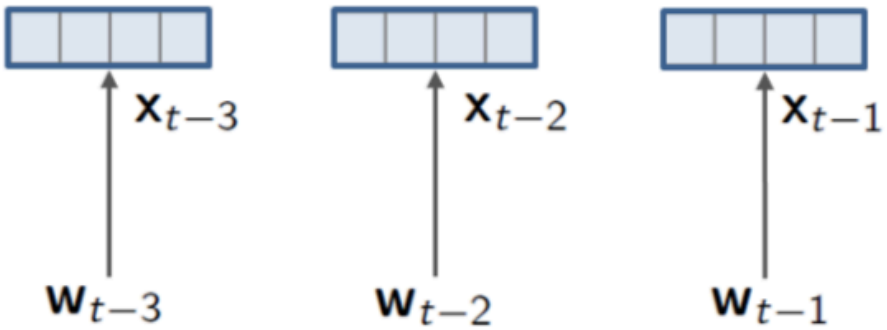




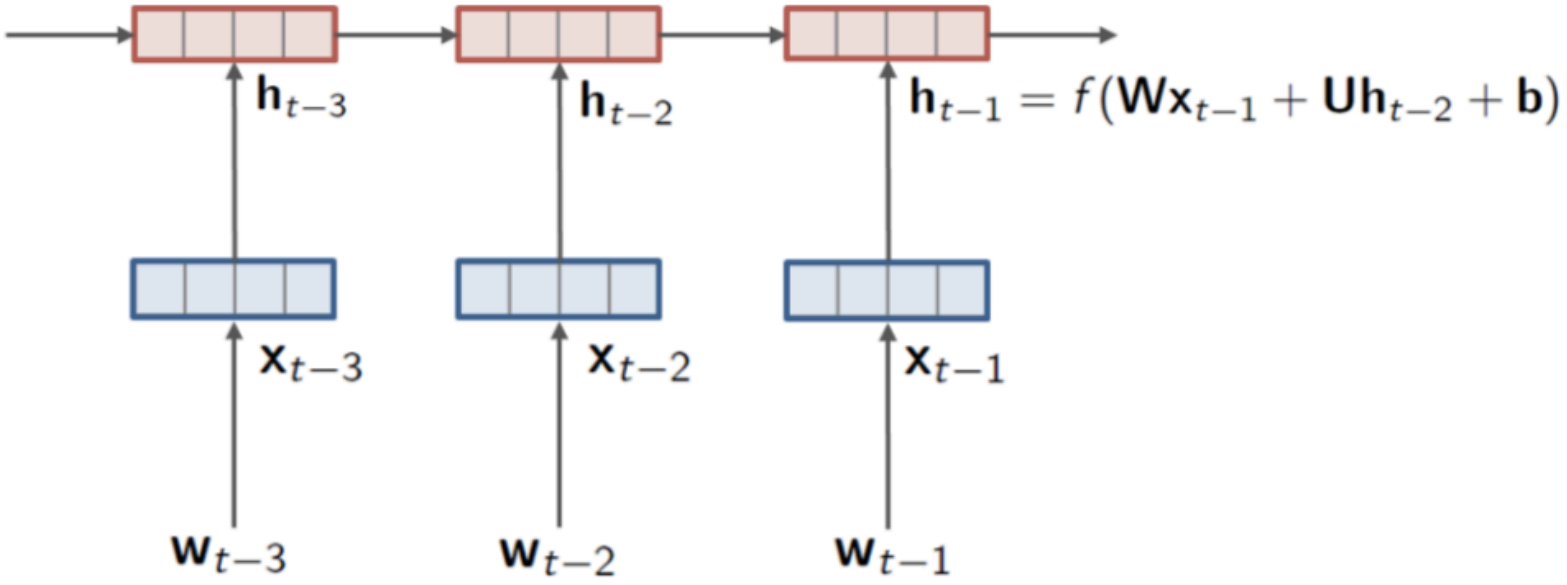
RNN Language Model

$\mathbf{w}_{t-3}$ $\mathbf{w}_{t-2}$ $\mathbf{w}_{t-1}$

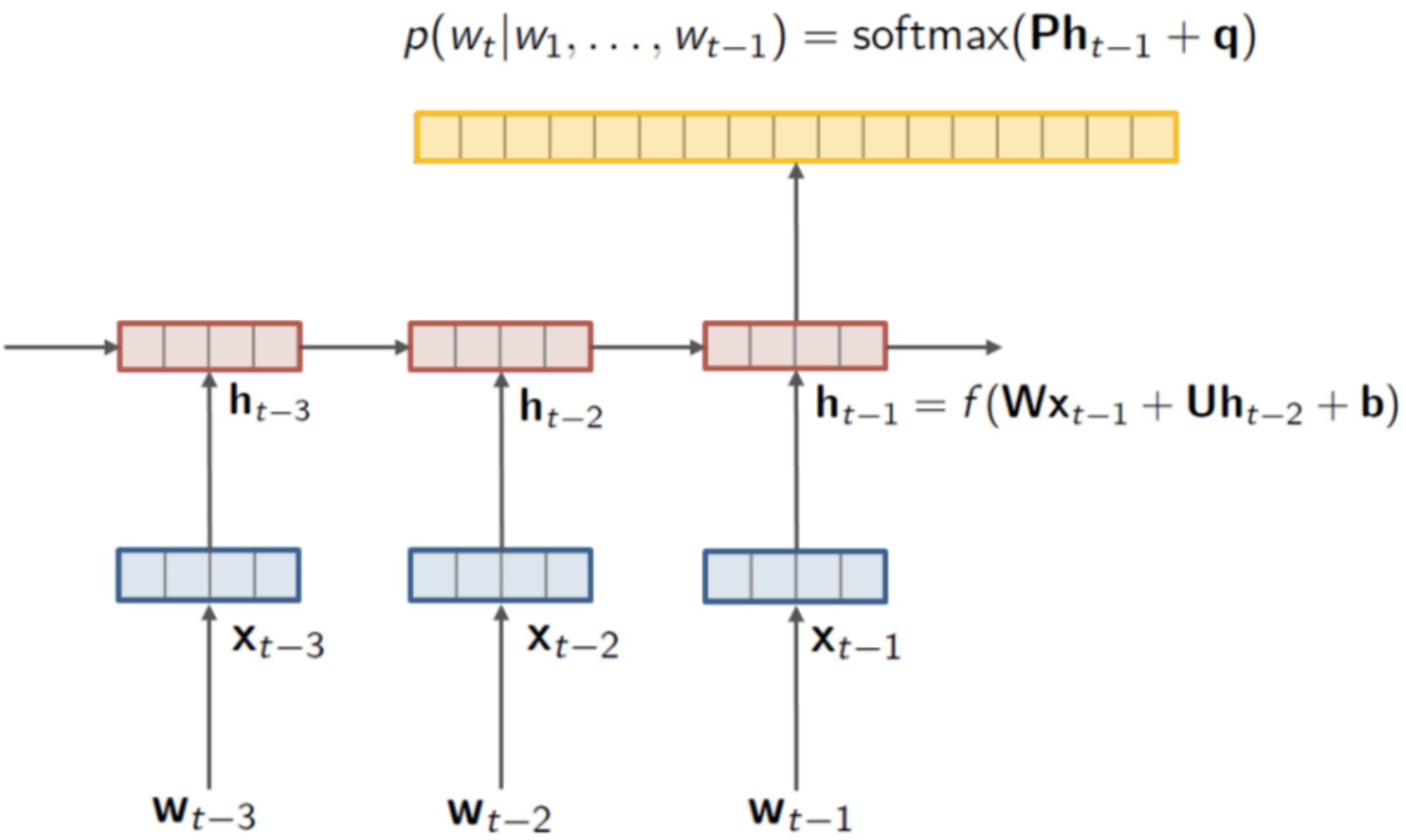
RNN Language Model



RNN Language Model

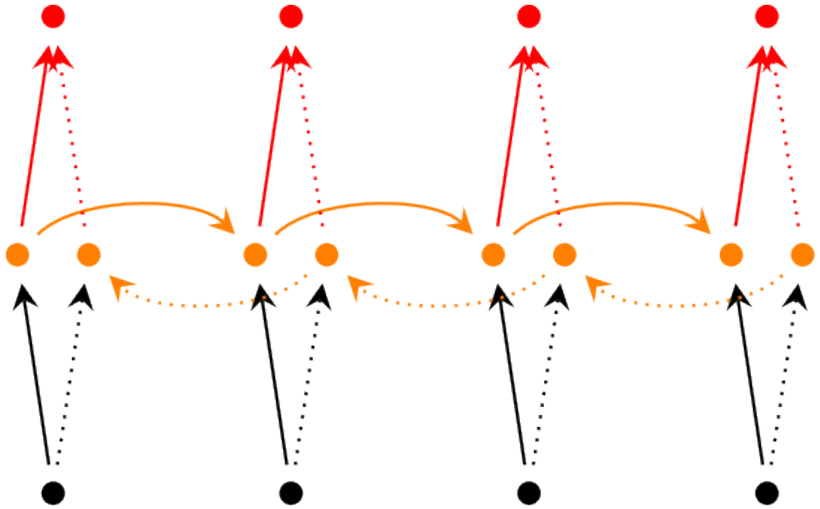


RNN Language Model



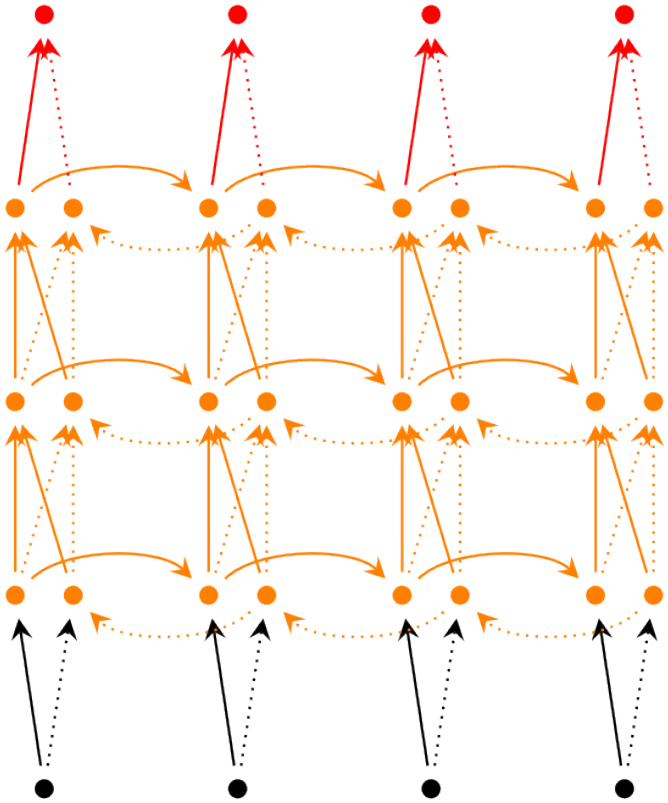
RNN EXTENSIONS

- Bidirectional RNNs



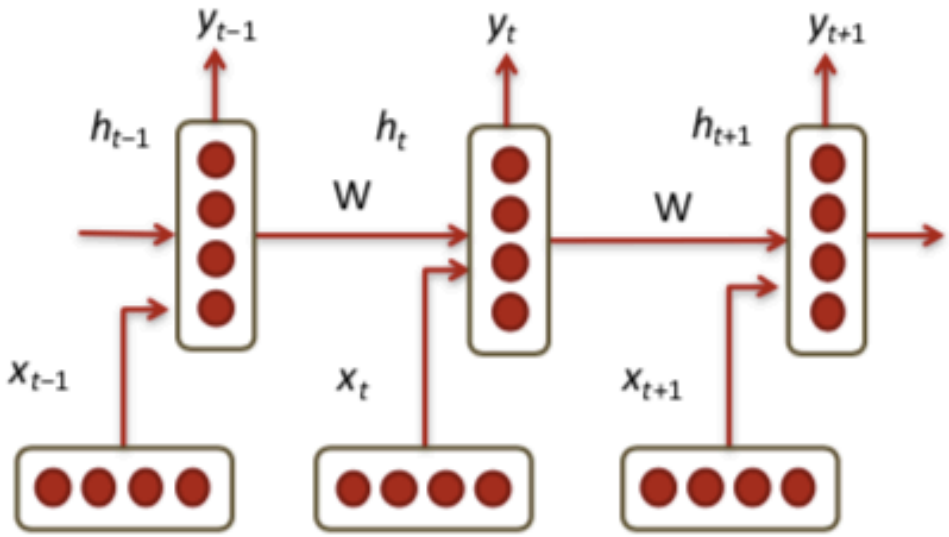
RNN EXTENSIONS

- Deep Bidirectional RNNs



TRAINING RNNS IS HARD

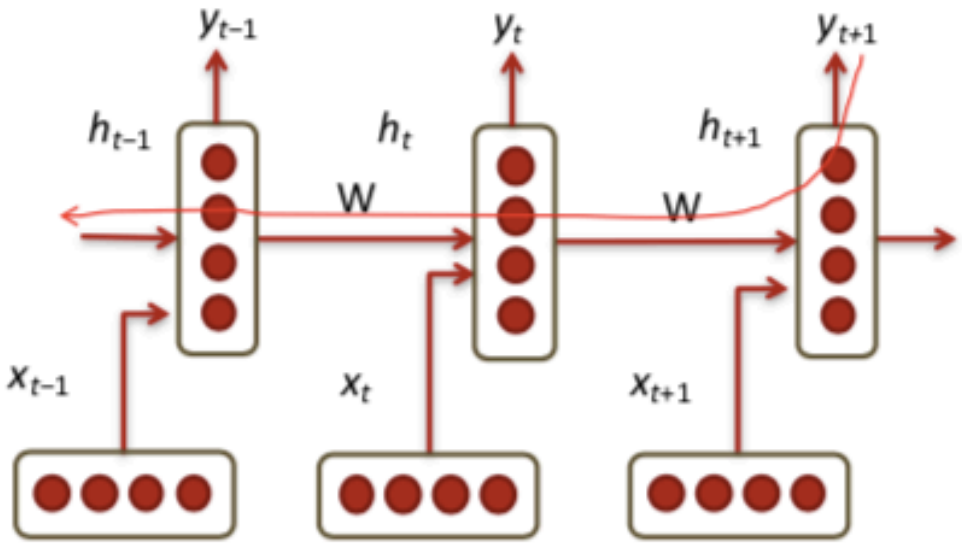
- Multiply the same matrix at each time step during forward prop



- Ideally inputs from many time steps ago can modify output y
- Take $\frac{\partial E_2}{\partial W}$ for an example RNN with 2 time steps! Insightful!

WHY IS THE VANISHING GRADIENT A PROBLEM

- The error at a time step ideally can tell a previous time step from many steps away to change during backprop



THE VANISHING GRADIENT PROBLEM FOR LANGUAGE MODELS

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word

- Example:

  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____

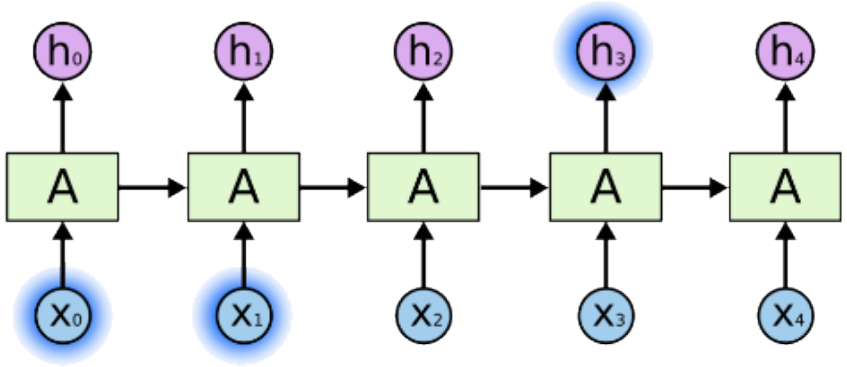
RNN EXTENSIONS

- LSTM networks
  - Not fundamentally different from RNN
  - Use different functions to compute hidden state
  - Memory of LSTMs are called cells
  - Cells decide what to keep in memory
- Very effective in capturing long-term dependencies

LONG-SHORT TERM MEMORY NETWORKS (LSTMS)

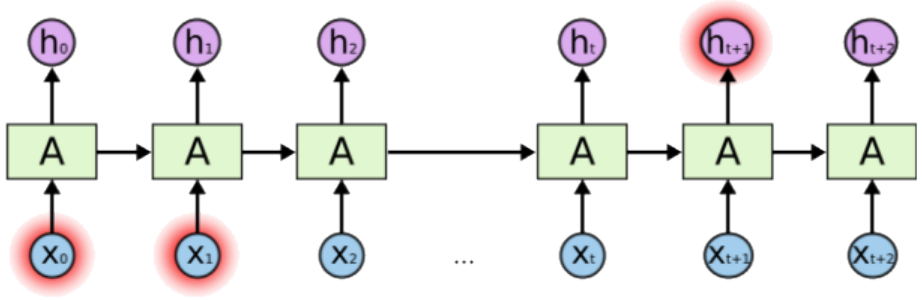
LONG TERM DEPENDENCIES

- Is RNN capable of capturing long-term dependencies?
- Why long-term dependencies?
  - Sometimes we only need to look at recent information to perform present task
- Consider an example
  - Predict next word based on the previous words



PROBLEM OF LONG TERM DEPENDENCIES

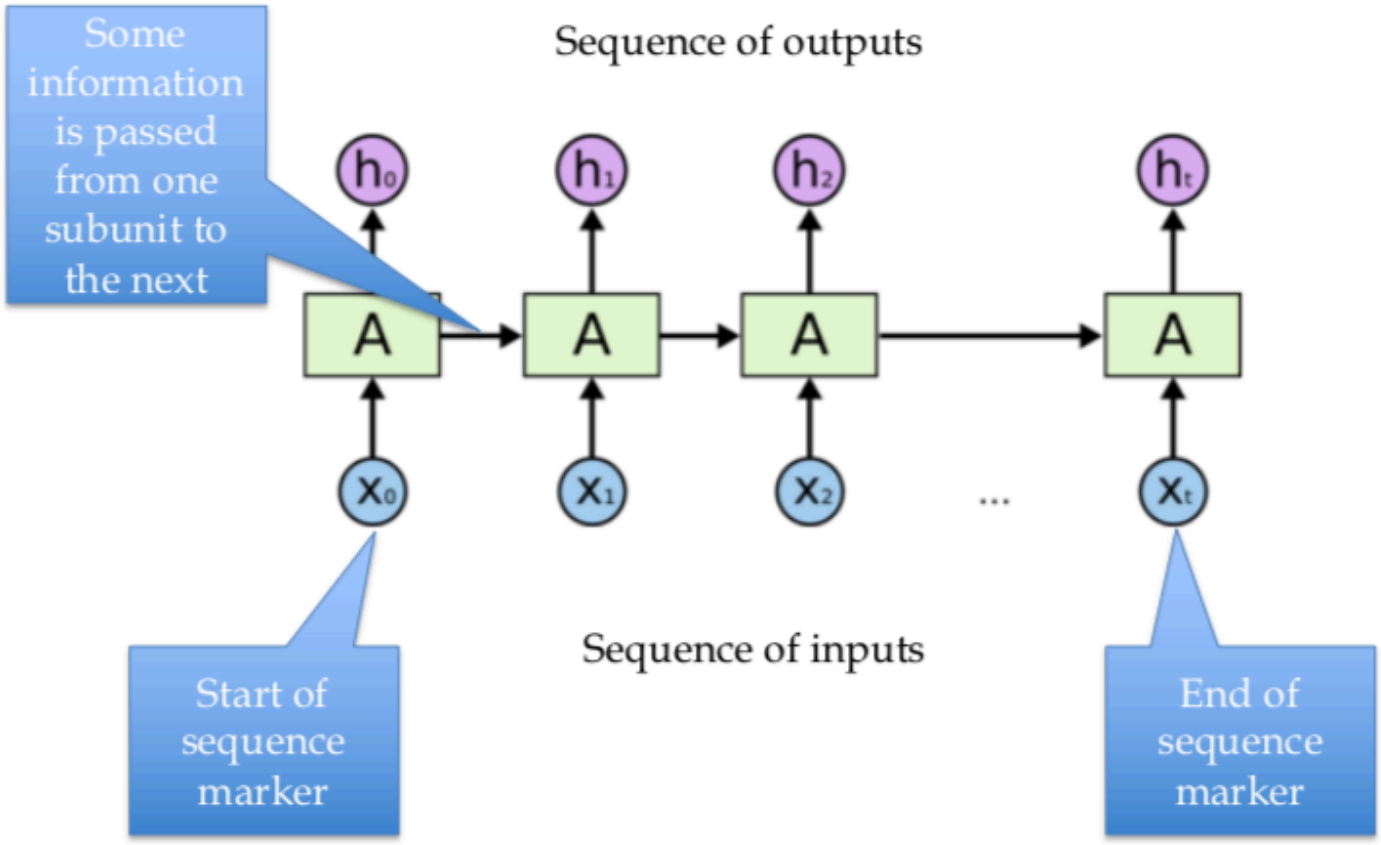
- What if we want to predict the next word in a long sentence?
- Do we know which past information is helpful to predict the next word?
- In theory, RNNs are capable of handling long-term dependencies.
- But in practice, they are not!



LONG SHORT TERM MEMORY (LSTM)

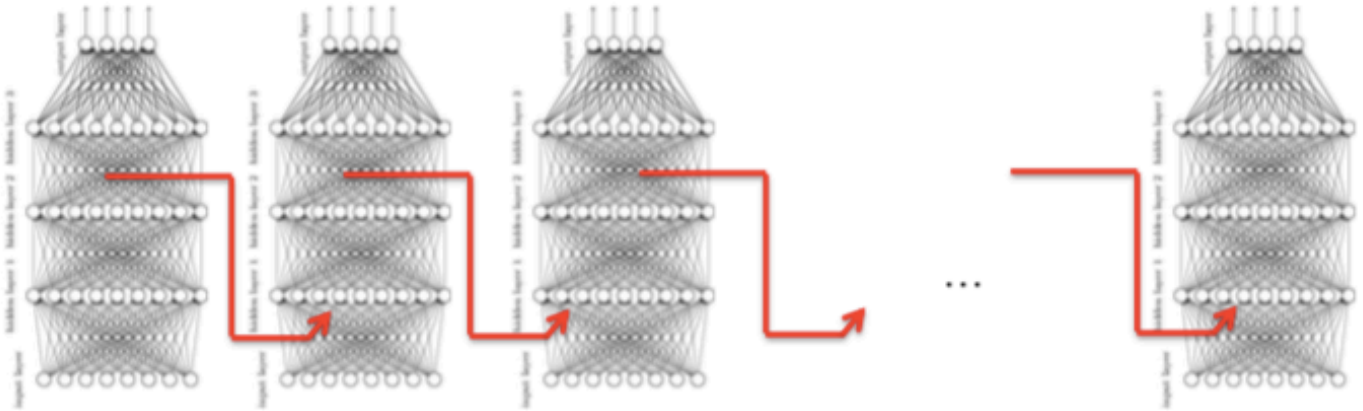
- Special kind of recurrent neural network
- Works well in many problems and now widely used
- Explicitly designed to avoid the long-term dependency problem
- Remembering information for long periods of time is their default behavior
  - Not something they struggle to learn
- So, what is the structural difference between RNN and LSTM?

Architecture for an RNN

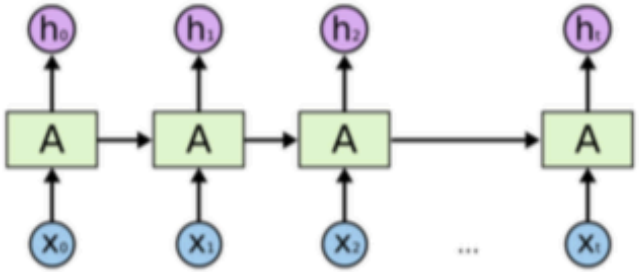


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

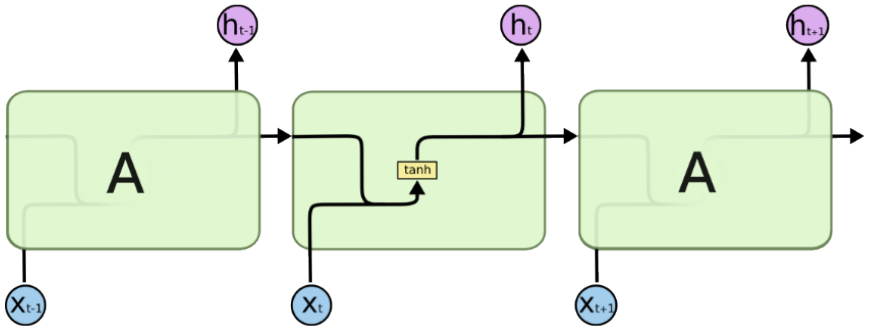
Architecture for an 1980's RNN



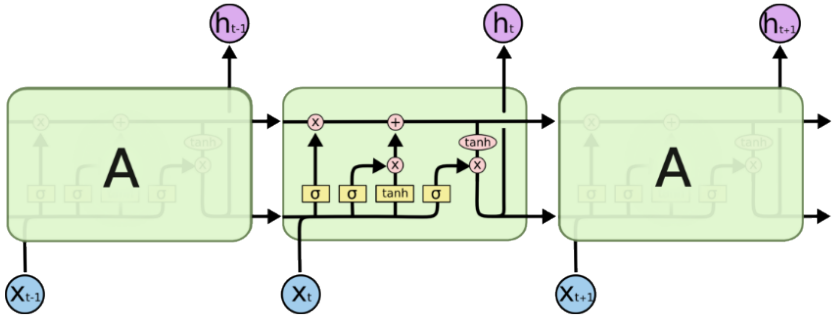
Problem with this: it's extremely deep and very hard to train



DIFFERENCE BETWEEN RNN AND LSTM

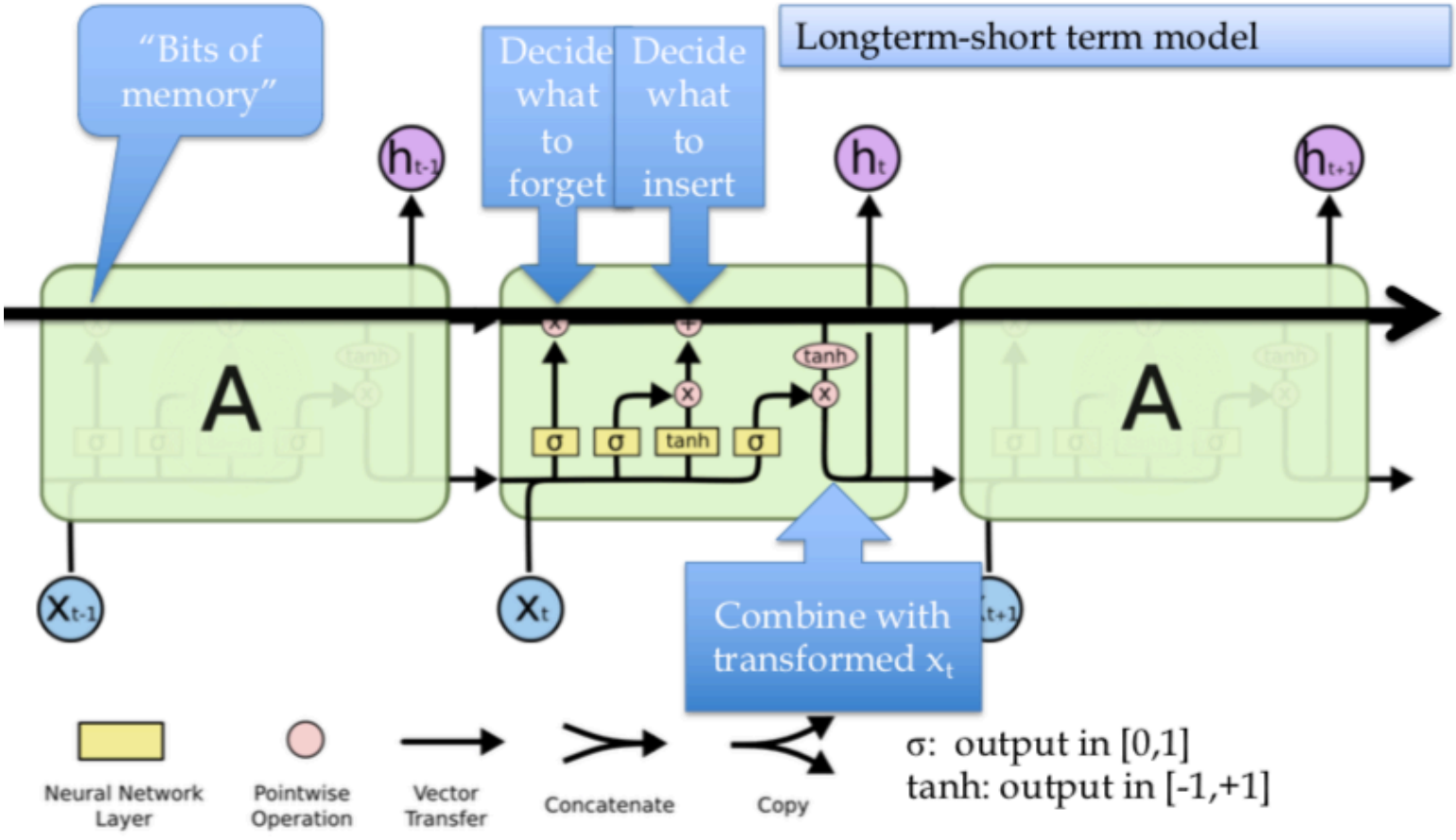


The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

Architecture for an LSTM



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

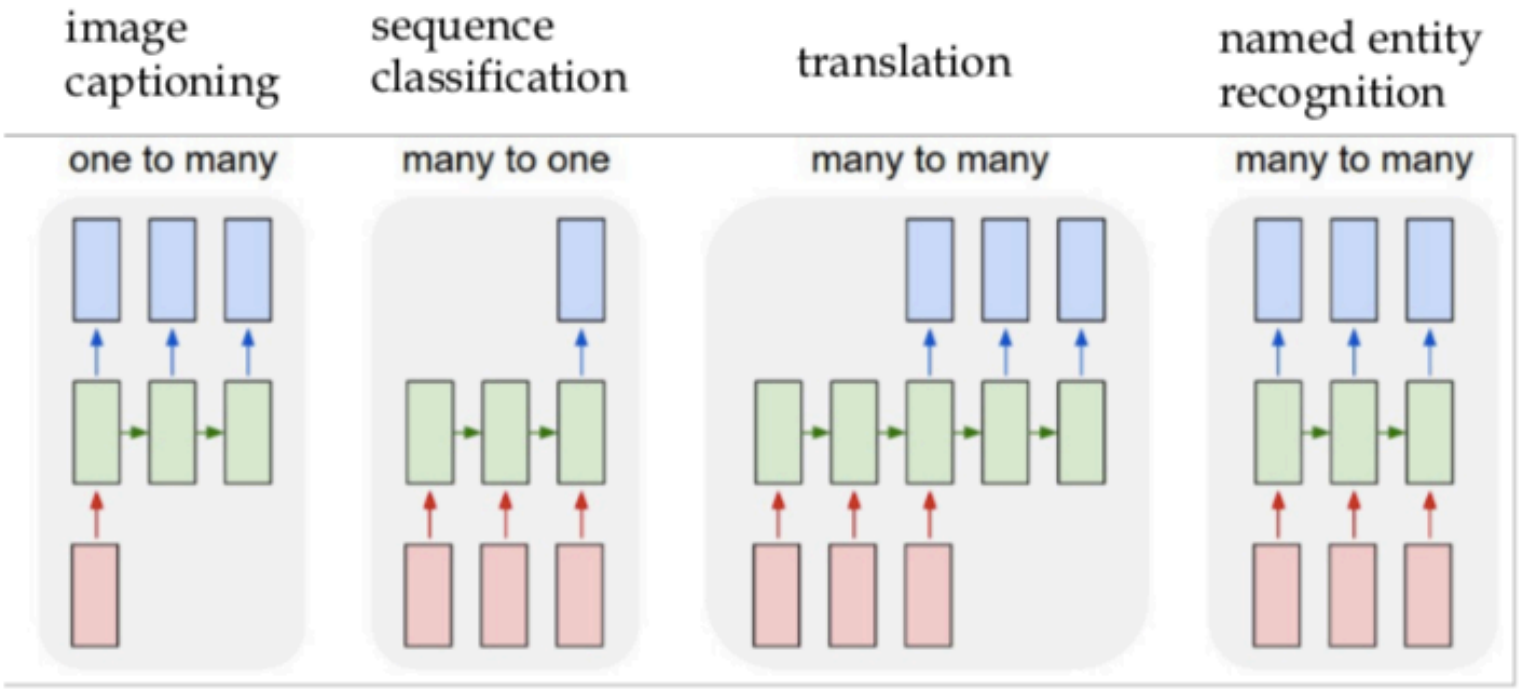
RNN Language Model Performance

Difficult/expensive to train, but performs well.

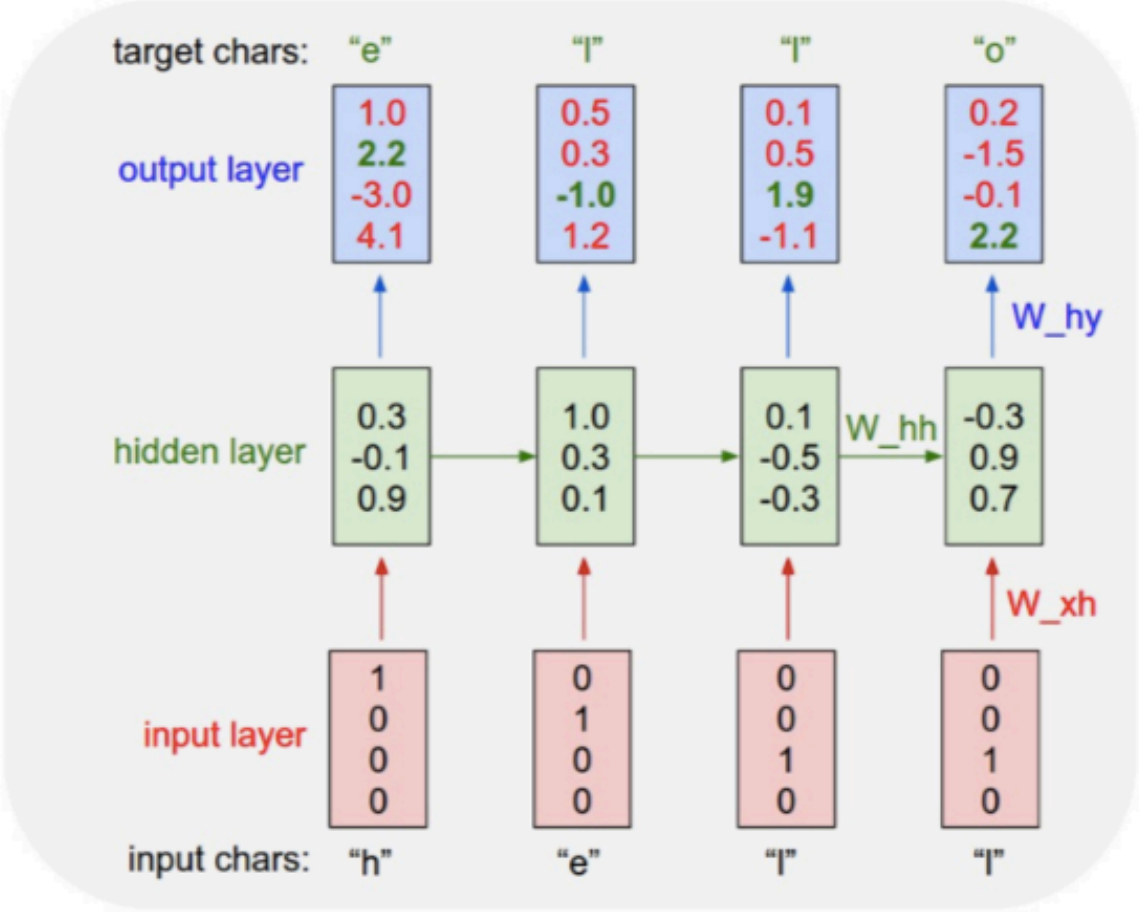
Language Model	Perplexity
5-gram count-based (Mikolov and Zweig 2012)	141.2
RNN (Mikolov and Zweig 2012)	124.7
Deep RNN (Pascanu et al. 2013)	107.5
LSTM (Zaremba, Sutskever, and Vinyals 2014)	78.4

Renewed interest in language modeling.

LSTMs can be used for other sequence tasks…



Character-level Language Model



Test time:

- pick a seed character sequence
- generate the next character
- then the next
- then the next …

Character-level LM

```
First Citizen:
Nay, then, that was hers,
It speaks against your other service:
But since the
youth of the circumstance be spoken:
Your uncle and one Baptista's daughter.

SEBASTIAN:
Do I stand till the break off.

BIRON:
Hide thy head.

VENTIDIUS:
He purposeth to Athens: whither, with the vow
I made to handle you.

FALSTAFF:
My good knave.
```

Yoav Goldberg: order-10 unsmoothed character n-grams

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

Character-level LM

```
MMMM----- Recipe via Meal-Master (tm) v8.05

      Title: BARBECUE RIBS
 Categories: Chinese, Appetizers
      Yield: 4 Servings

      1 pk Seasoned rice
      1    Beer -- cut into
           -cubes
      1 ts Sugar
    3/4 c  Water
           Chopped finels,
           -up to 4 tblsp of chopped
      2 pk Yeast Bread/over

MMMM-------------------------FILLINC
      2 c  Pineapple, chopped
    1/3 c  Milk
    1/2 c  Pecans
           Cream of each
      2 tb Balsamic cocoa
      2 tb Flour
      2 ts Lemon juice
           Granulated sugar
      2 tb Orange juice
      1 c  Sherry wheated curdup
      1    Onion; sliced
      1 ts Salt
      2 c  Sugar
    1/4 ts Salt
    1/2 ts White pepper, freshly ground
           Sesame seeds
      1 c  Sugar
    1/4 c  Shredded coconut
    1/4 ts Cumin seeds

Preheat oven to 350.  In a medium bowl, combine milk,
flour and water and then cornstarch. add tomatoes, or
nutmeg; serve.
```

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

Character-level LM

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. $\square$

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

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

LaTeX "almost compiles"

RNNS IN DYNET

- Based on ”Builder” class (=SimpleRNN/LSTM)
- Add parameters to model (once):

```
#LSTM (layers=1, input=64, hidden=128, model)
RNN = dy.LSTMBuilder(1,64,128,model)
```

- Add parameters to CG and get initial state (per sentence)

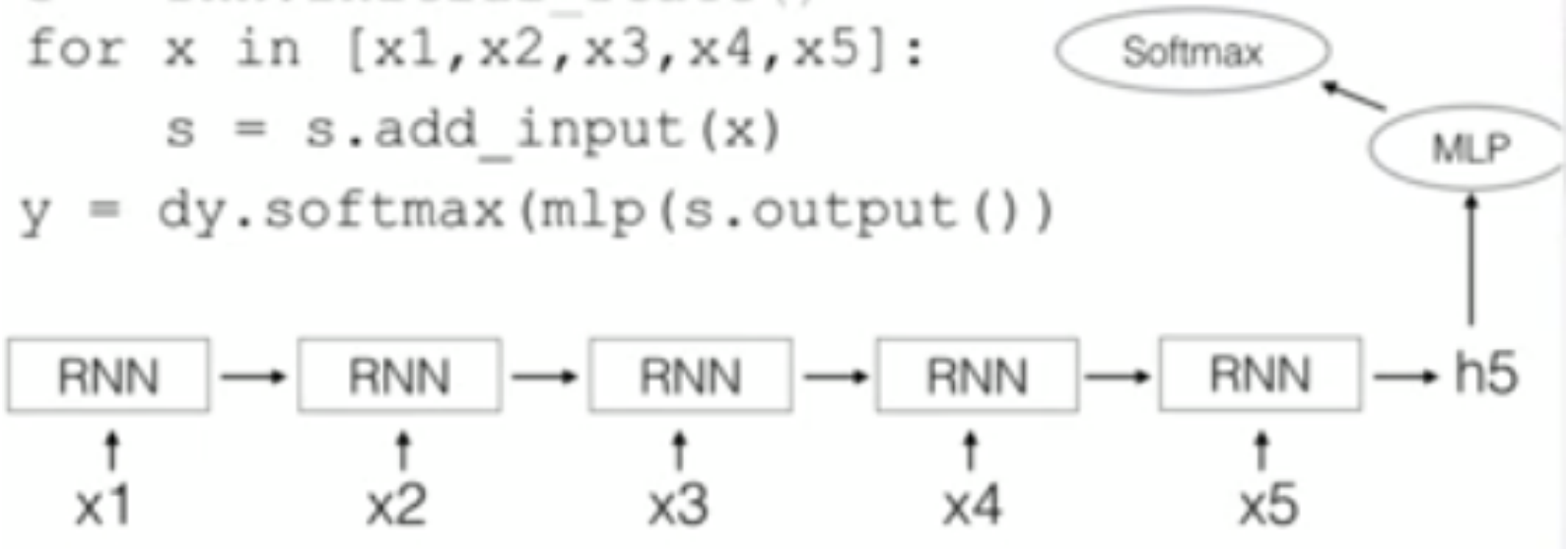
```
s = RNN.initial_state()
```

- Update state and access (per input word/character)

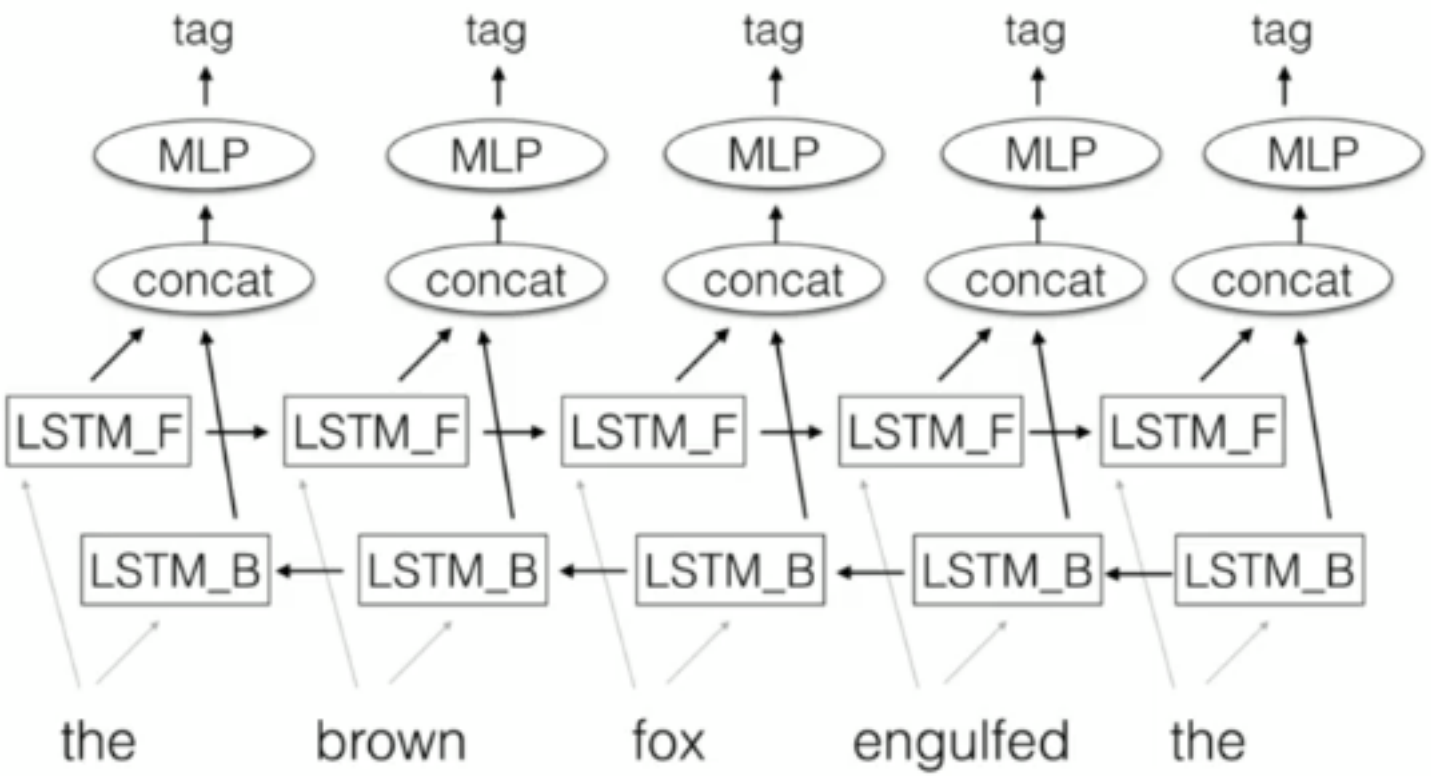
```
s = s.add_input(x_t)
h_t = s.output()
```

RNNS IN DYNET

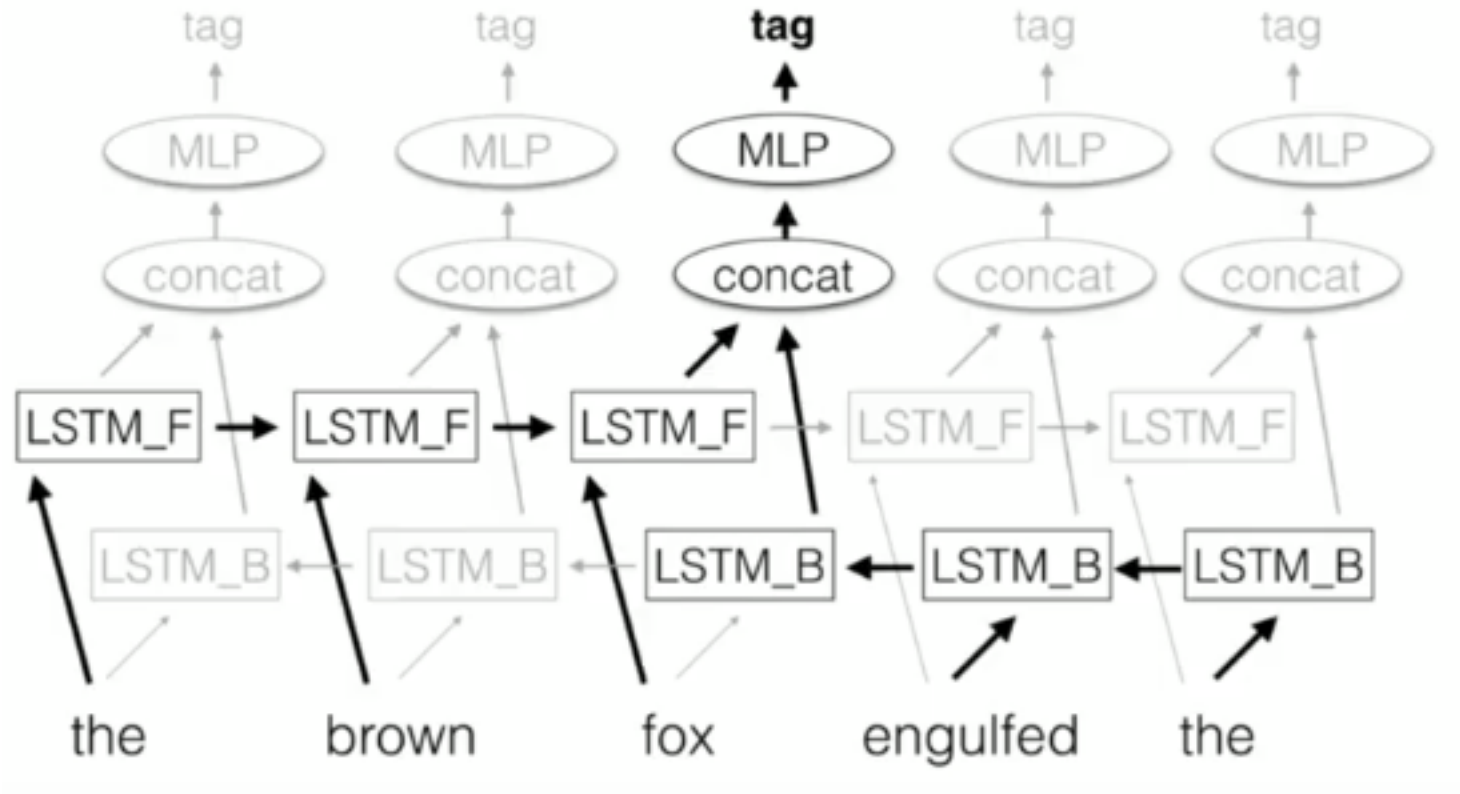
```
rnn = dy.LSTMBuilder(1, 64, 128,model)
s = rnn.initial_state()
for x in [x1,x2,x3,x4,x5]:
    s = s.add_input(x)
y = dy.softmax(mlp(s.output()))
```



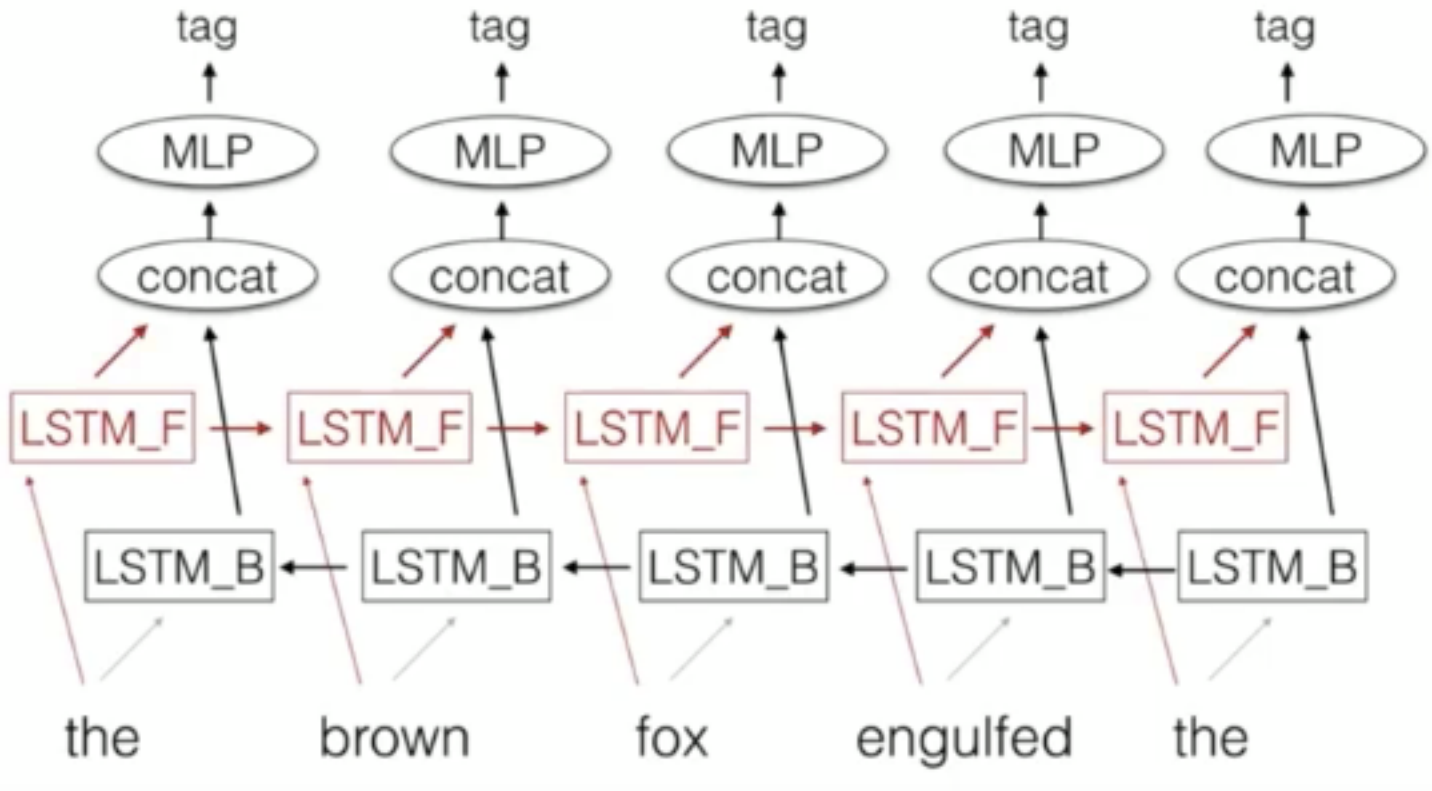
BILSTM TAGGER



BILSTM TAGGER



BILSTM TAGGER



```
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 128))
fwdRNN = dy.LSTMBuilder(1,  128,  50, model)
```

layers in-dim out-dim

```
dy.renew_cg()
# initialize the RNNs
f_init = fwdRNN.initial_state()

wembs = [word_rep(w) for w in words]

fw_exps = []
s = f_init
for we in wembs:
    s = s.add_input(we)
    fw_exps.append(s.output())
```

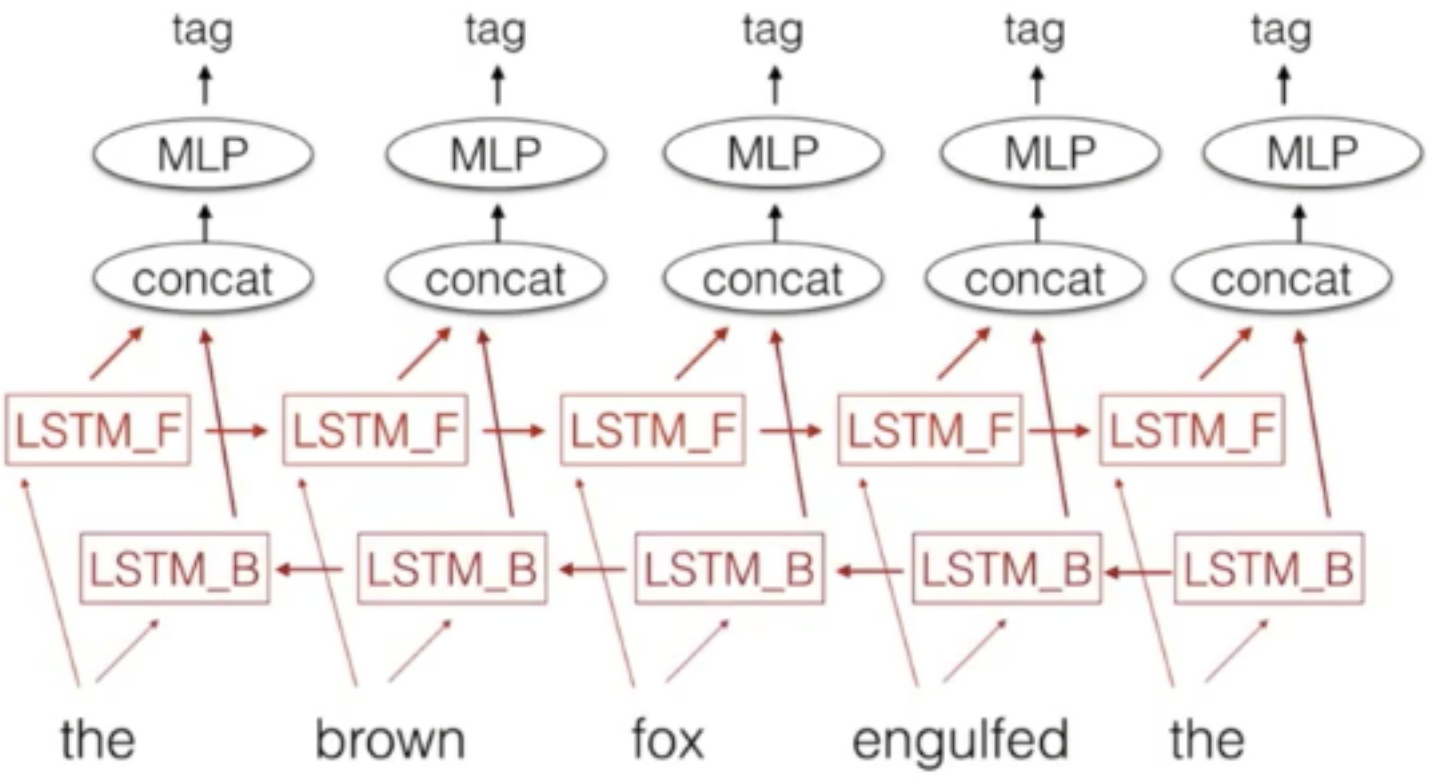
```
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 128))
fwdRNN = dy.LSTMBuilder(1,  128,  50, model)
                        layers in-dim out-dim

dy.renew_cg(
# initialize
f_init = fwd

wembs = [word_rep(w) for w in words]

fw_exps = []
s = f_init
for we in wembs:
    s = s.add_input(we)
    fw_exps.append(s.output())
```

```
def word_rep(w):
    w_index = vw.w2i[w]
    return WORDS_LOOKUP[w_index]
```

```
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 128))
fwdRNN = dy.LSTMBuilder(1,  128,  50, model)
```

layers in-dim out-dim

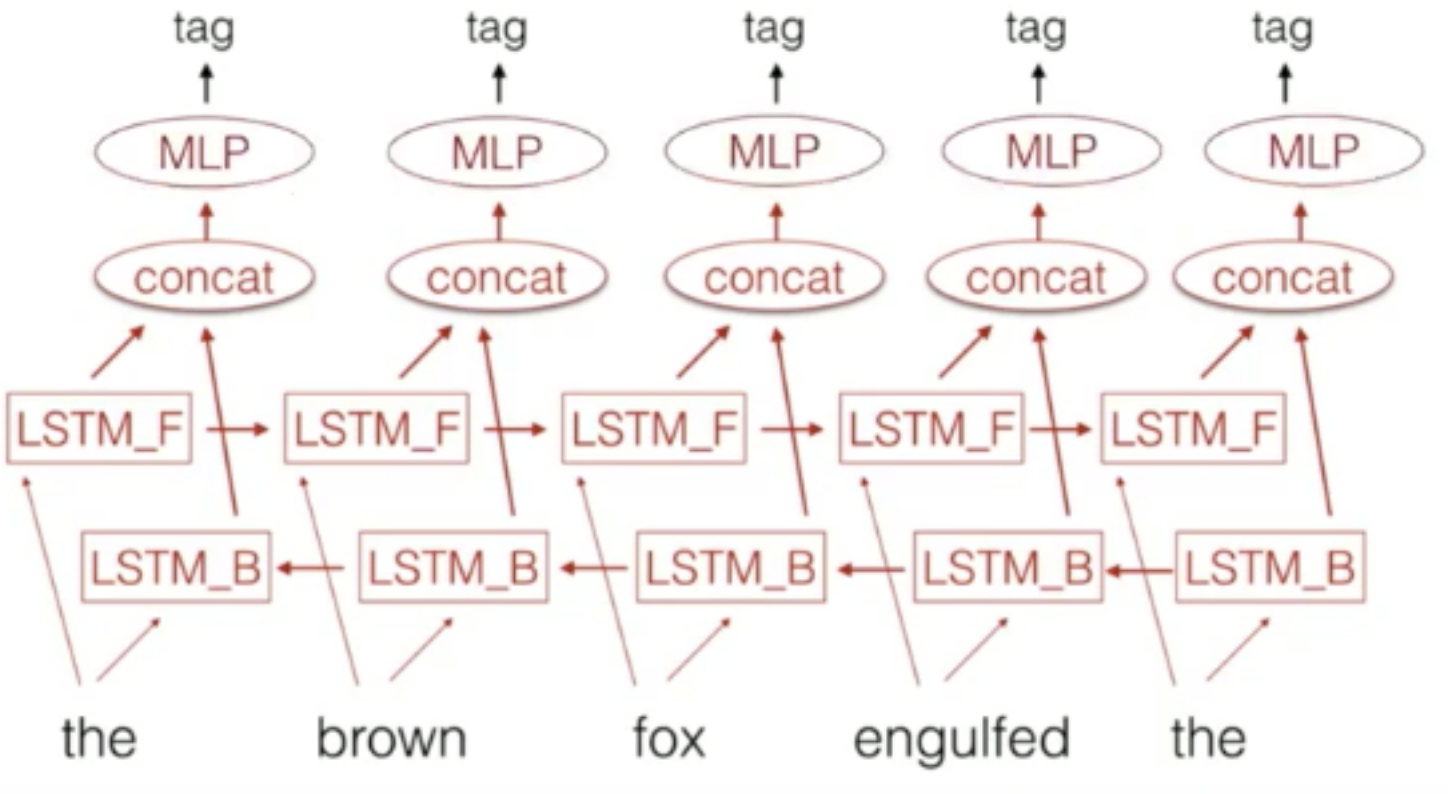
```
dy.renew_cg()
# initialize the RNNs
f_init = fwdRNN.initial_state()

wembs = [word_rep(w) for w in words]

fw_exps = f_init.transduce(wembs)
```


tag
tag
tag
tag
tag
MLP
MLP
MLP
MLP
MLP
concat
concat
concat
concat
concat
LSTM_F
LSTM_F
LSTM_F
LSTM_F
LSTM_F
LSTM_B
LSTM_B
LSTM_B
LSTM_B
LSTM_B
the
brown
fox
engulfed
the

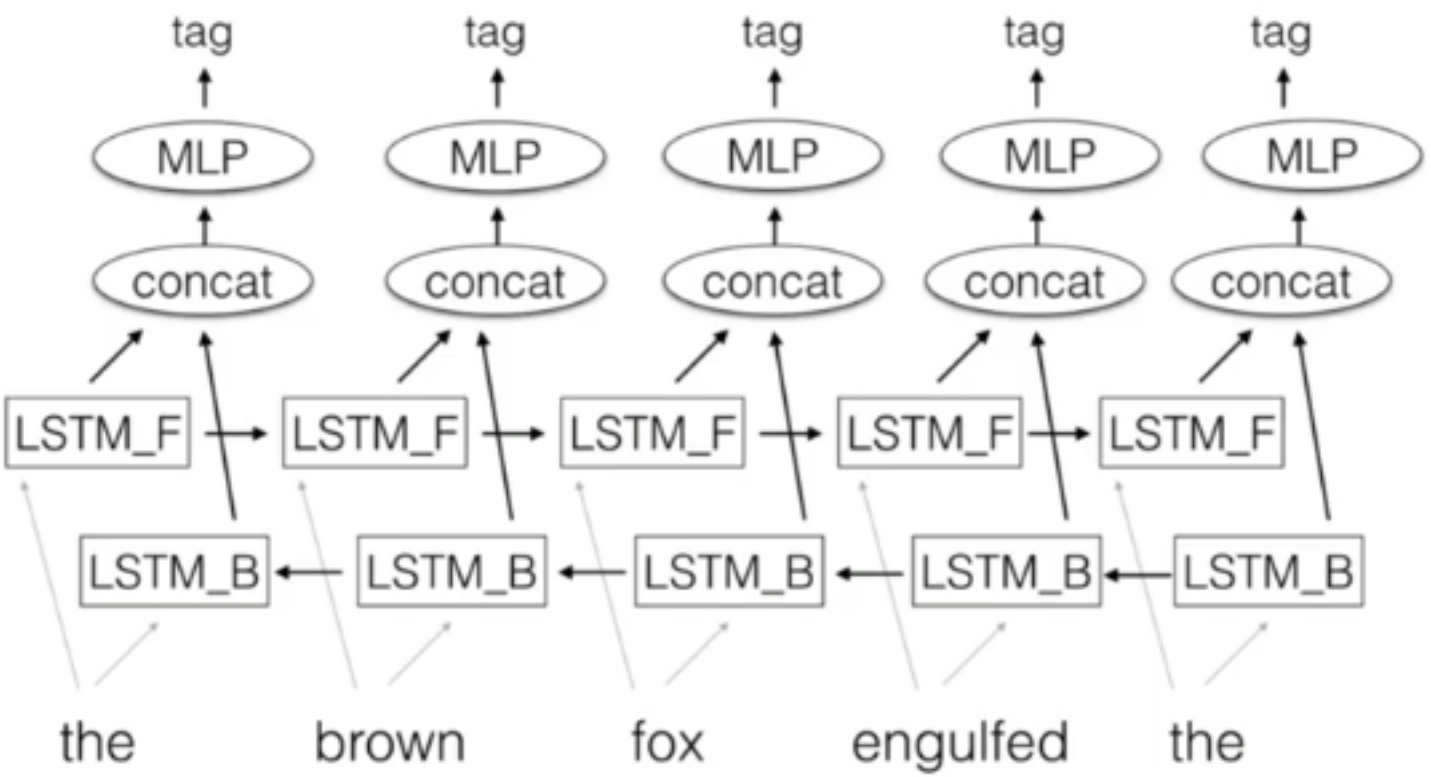
```
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 128))
fwdRNN = dy.LSTMBuilder(1, 128, 50, model)
bwdRNN = dy.LSTMBuilder(1, 128, 50, model)
```

```
dy.renew_cg()
# initialize the RNNs
f_init = fwdRNN.initial_state()
b_init = bwdRNN.initial_state()

wembs = [word_rep(w) for w in words]

fw_exps = f_init.transduce(wembs)
bw_exps = b_init.transduce(reversed(wembs))
```

```
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 128))
fwdRNN = dy.LSTMBuilder(1, 128, 50, model)
bwdRNN = dy.LSTMBuilder(1, 128, 50, model)



dy.renew_cg()
# initialize the RNNs
f_init = fwdRNN.initial_state()
b_init = bwdRNN.initial_state()

wembs = [word_rep(w) for w in words]

fw_exps = f_init.transduce(wembs)
bw_exps = b_init.transduce(reversed(wembs))

# biLSTM states
bi = [dy.concatenate([f,b]) for f,b in zip(fw_exps,
                                           reversed(bw_exps))]
```


tag
tag
tag
tag
tag
MLP
MLP
MLP
MLP
MLP
concat
concat
concat
concat
concat
LSTM_F
LSTM_F
LSTM_F
LSTM_F
LSTM_F
LSTM_B
LSTM_B
LSTM_B
LSTM_B
LSTM_B
the
brown
fox
engulfed
the

```
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 128))
fwdRNN = dy.LSTMBuilder(1, 128, 50, model)
bwdRNN = dy.LSTMBuilder(1, 128, 50, model)
pH = model.add_parameters((32, 50*2))
pO = model.add_parameters((ntags, 32))


dy.renew_cg()
# initialize the RNNs
f_init = fwdRNN.initial_state()
b_init = bwdRNN.initial_state()
wembs = [word_rep(w) for w in words]
fw_exps = f_init.transduce(wembs)
bw_exps = b_init.transduce(reversed(wembs)

# biLSTM states
bi = [dy.concatenate([f,b]) for f,b in zip(fw_exps,
                                           reversed(bw_exps))]

# MLPs
H = dy.parameter(pH)
O = dy.parameter(pO)
outs = [O*(dy.tanh(H * x)) for x in bi]
```


tag
tag
tag
tag
tag
MLP
MLP
MLP
MLP
MLP
concat
concat
concat
concat
concat
LSTM_F
LSTM_F
LSTM_F
LSTM_F
LSTM_F
LSTM_B
LSTM_B
LSTM_B
LSTM_B
LSTM_B
the
brown
fox
engulfed
the

REFERENCES

- Introduction to Neural Networks, Philipp Koehn, 3 October 2017
- Artificial Neural Networks and Deep Learning, Christian Borgelt, University of Konstanz, Germany
- Natural Language Processing with Deep Learning, Richard Socher, Kevin Clark, Stanford University, 2017
- Richard Socher, Neural Networks, 2018