Part of Speech Tagging

Natural Language Processing: Jordan Boyd-Graber
University of Maryland

HIDDEN MARKOV MODELS

Adapted from material by Ray Mooney

Roadmap

- Identify common classes of part of speech tags
- Understand why pos tags can help
- How to add features to improve classification
- Joint labeling: Hidden Markov Models (high level)
- Hidden Markov Model (rigorous definition)
- Estimating HMM

POS Tagging: Task Definition

- Annotate each word in a sentence with a part-of-speech marker.
- Lowest level of syntactic analysis.

John	saw	the	saw	and	decided	to	take	it	to	the	table
NNP	VBD	DT	NN	CC	VBD	TO	VB	PRP	IN	DT	NN

- Useful for subsequent syntactic parsing and word sense disambiguation.

What are POS Tags?

- Original Brown corpus used a large set of 87 POS tags.
- Most common in NLP today is the Penn Treebank set of 45 tags. Tagset used in these slides for “real” examples. Reduced from the Brown set for use in the context of a parsed corpus (i.e. treebank).
- The C5 tagset used for the British National Corpus (BNC) has 61 tags.

Tag Examples

- Noun (person, place or thing)
  - Singular (NN): dog, fork
  - Plural (NNS): dogs, forks
  - Proper (NNP, NNPS): John, Springfields
- Personal pronoun (PRP): I, you, he, she, it
- Wh-pronoun (WP): who, what
- Verb (actions and processes)
  - Base, infinitive (VB): eat
  - Past tense (VBD): ate
  - Gerund (VBG): eating
  - Past participle (VBN): eaten
  - Non 3rd person singular present tense (VBP): eat
  - 3rd person singular present tense: (VBZ): eats
  - Modal (MD): should, can
  - To (TO): to (to eat)

Tag Examples (cont.)

- Adjective (modify nouns)
  - Basic (JJ): red, tall
  - Comparative (JJR): redder, taller
  - Superlative (JJS): reddest, tallest
- Adverb (modify verbs)
  - Basic (RB): quickly
  - Comparative (RBR): quicker
  - Superlative (RBS): quickest
- Preposition (IN): on, in, by, to, with
- Determiner:
  - Basic (DT) a, an, the
  - WH-determiner (WDT): which, that
- Coordinating Conjunction (CC): and, but, or,
- Particle (RP): off (took off), up (put up)

Open vs. Closed Class

- Closed class categories are composed of a small, fixed set of grammatical function words for a given language.
  - Pronouns, Prepositions, Modals, Determiners, Particles, Conjunctions
- Open class categories have large number of words and new ones are easily invented.
  - Nouns (Googler, textlish), Verbs (Google), Adjectives (geeky), Abverb (chompingly)

Ambiguity

“Like" can be a verb or a preposition

- I like/VBP candy.
- Time flies like/IN an arrow.

“Around” can be a preposition, particle, or adverb

- I bought it at the shop around/IN the corner.
- I never got around/RP to getting a car.
- A new Prius costs around/RB $25K.

How hard is it?

- Usually assume a separate initial tokenization process that separates and/or disambiguates punctuation, including detecting sentence boundaries.
- Degree of ambiguity in English (based on Brown corpus)
  - 11.5% of word types are ambiguous.
  - 40% of word tokens are ambiguous.
- Average POS tagging disagreement amongst expert human judges for the Penn treebank was 3.5%
- Based on correcting the output of an initial automated tagger, which was deemed to be more accurate than tagging from scratch.
- Baseline: Picking the most frequent tag for each specific word type gives about 90% accuracy 93.7% if use model for unknown words for Penn Treebank tagset.

What about classification / feature engineering?

- Let’s view the context as input
- pos tag is the label
- How can we select better features?

Baseline

- Just predict the most frequent class
- 0.38 accuracy

Prefix and Suffixes

- Take what characters start a word (`un`, `re`, `in`)
- Take what characters end a word (`ly`, `ing`)
- Use as features

Prefix and Suffixes

- Take what characters start a word (`un`, `re`, `in`)
- Take what characters end a word (`ly`, `ing`)
- Use as features (Accuracy: 0.55)

Prefix and Suffixes

- Take what characters start a word (`un`, `re`, `in`)
- Take what characters end a word (`ly`, `ing`)
- Use as features (Accuracy: 0.55)
- What can you do to improve the set of features?

Error Analysis

- Look at predictions of the models
- Look for patterns in frequent errors

Errors from prefix / suffix model

said (372), back (189), get (153), then (147), know (144), Mr. (87), Mike (78)

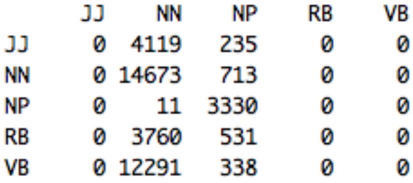
Error Analysis

- Look at predictions of the models
- Look for patterns in frequent errors

Errors from prefix / suffix model

said (372), back (189), get (153), then (147), know (144), Mr. (87), Mike (78)

Confusion Matrix: Only Capitalization

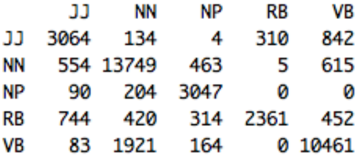


```
        JJ     NN     NP     RB     VB
JJ       0   4119    235      0      0
NN       0  14673    713      0      0
NP       0     11   3330      0      0
RB       0   3760    531      0      0
VB       0  12291    338      0      0
```

Accuracy: 0.45

Incorporating Knowledge

- Use WordNet, an electronic dictionary in nltk
- (We'll talk more about it later)
- Now getting 0.82 accuracy



```
       JJ    NN    NP    RB    VB
JJ   3064   134     4   310   842
NN    554 13749   463     5   615
NP     90   204  3047     0     0
RB    744   420   314  2361   452
VB     83  1921   164     0 10461
```

Error Analysis

back	then	now	there	here	still	long	thought	want	even
223	145	140	116	115	100	99	88	79	67

A more fundamental problem . . .

- Each classification is independent . . .
- This isn't right!
- If you have a noun, it's more likely to be preceeded by an adjective
- Determiners are followed by either a noun or an adjective
- Determiners don't follow each other

Approaches

- Rule-Based: Human crafted rules based on lexical and other linguistic knowledge.
- Learning-Based: Trained on human annotated corpora like the Penn Treebank.
    - Statistical models: Hidden Markov Model (HMM), Maximum Entropy Markov Model (MEMM), Conditional Random Field (CRF)
    - Rule learning: Transformation Based Learning (TBL)
    - Deep learning: RNN / LSTM
- Generally, learning-based approaches have been found to be more effective overall, taking into account the total amount of human expertise and effort involved.

Approaches

- Rule-Based: Human crafted rules based on lexical and other linguistic knowledge.
- Learning-Based: Trained on human annotated corpora like the Penn Treebank.
    - Statistical models: Hidden Markov Model (HMM), Maximum Entropy Markov Model (MEMM), Conditional Random Field (CRF)
    - Rule learning: Transformation Based Learning (TBL)
    - Deep learning: RNN / LSTM
- Generally, learning-based approaches have been found to be more effective overall, taking into account the total amount of human expertise and effort involved.

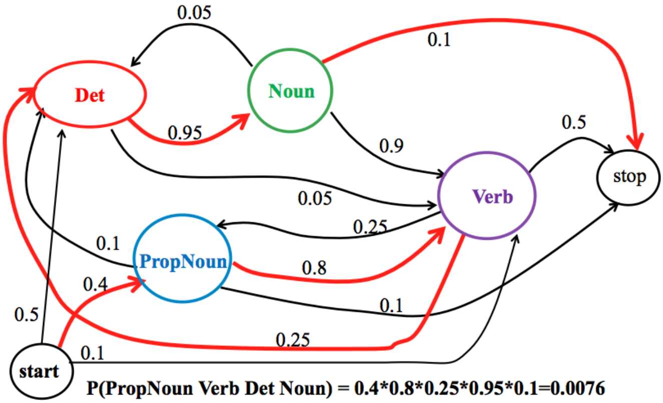
HMM Definition

- A finite state machine with probabilistic state transitions.
- Makes Markov assumption that next state only depends on the current state and independent of previous history.

Generative Model

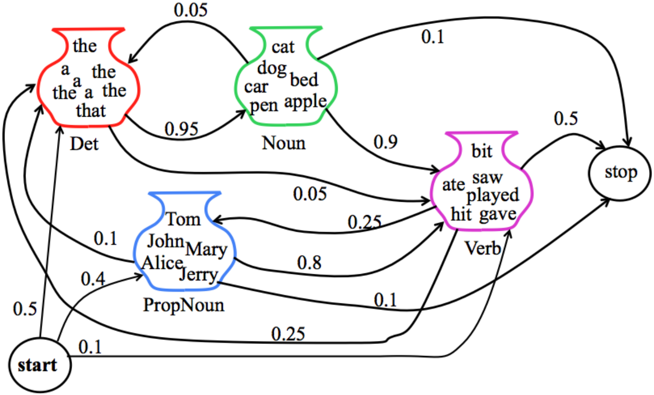
- Probabilistic generative model for sequences.
- Assume an underlying set of hidden (unobserved) states in which the model can be (e.g. parts of speech).
- Assume probabilistic transitions between states over time (e.g. transition from POS to another POS as sequence is generated).
- Assume a probabilistic generation of tokens from states (e.g. words generated for each POS).

Cartoon



P(PropNoun Verb Det Noun) = 0.4*0.8*0.25*0.95*0.1=0.0076

Cartoon


0.05
0.1
the
a
a
the
the
a
the
that
Det
cat
dog
bed
car
pen
apple
Noun
0.95
0.9
0.5
bit
ate
saw
played
hit
gave
Verb
stop
0.05
Tom
John
Mary
Alice
Jerry
PropNoun
0.25
0.1
0.8
0.4
0.1
0.5
0.25
0.1
start

HMM Definition

Assume K parts of speech, a lexicon size of V, a series of observations $\{x_1, \dots, x_N\}$, and a series of unobserved states $\{z_1, \dots, z_N\}$.

- π A distribution over start states (vector of length K): $\pi_i = p(z_1 = i)$
- θ Transition matrix (matrix of size K by K): $\theta_{i,j} = p(z_n = j | z_{n-1} = i)$
- β An emission matrix (matrix of size K by V): $\beta_{j,w} = p(x_n = w | z_n = j)$

HMM Definition

Assume K parts of speech, a lexicon size of V, a series of observations $\{x_1, \dots, x_N\}$, and a series of unobserved states $\{z_1, \dots, z_N\}$.

- π A distribution over start states (vector of length K): $\pi_i = p(z_1 = i)$
- θ Transition matrix (matrix of size K by K): $\theta_{i,j} = p(z_n = j | z_{n-1} = i)$
- β An emission matrix (matrix of size K by V): $\beta_{j,w} = p(x_n = w | z_n = j)$

Two problems: How do we move from data to a model? (Estimation) How do we move from a model and unlabled data to labeled data? (Inference)

Reminder: How do we estimate a probability?

- For a multinomial distribution (i.e. a discrete distribution, like over words):

$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \tag{1}$$

- α_i is called a smoothing factor, a pseudocount, etc.

Reminder: How do we estimate a probability?

- For a multinomial distribution (i.e. a discrete distribution, like over words):

$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \tag{1}$$

- α_i is called a smoothing factor, a pseudocount, etc.
- When $\alpha_i = 1$ for all i, it's called “Laplace smoothing” and corresponds to a uniform prior over all multinomial distributions.

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	V

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

x	here	come	old	flattop
	MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	V

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

x	here	come	old	flattop
z	MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	V

and	I	love	her
CONJ	PRO	V	PRO

Initial Probability π

POS	Frequency	Probability
MOD	1.1	0.234
DET	1.1	0.234
CONJ	1.1	0.234
N	0.1	0.021
PREP	0.1	0.021
PRO	0.1	0.021
V	1.1	0.234

Remember, we're taking MAP estimates, so we add 0.1 (arbitrarily chosen) to each of the counts before normalizing to create a probability distribution. This is easy; one sentence starts with an adjective, one with a determiner, one with a verb, and one with a conjunction.

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Transition Probability θ

- We can ignore the words; just look at the parts of speech. Let's compute one row, the row for verbs.
- We see the following transitions: V → MOD, V → CONJ, V → V, V → PRO, and V → PRO

POS	Frequency	Probability
MOD	1.1	0.193
DET	0.1	0.018
CONJ	1.1	0.193
N	0.1	0.018
PREP	0.1	0.018
PRO	2.1	0.368
V	1.1	0.193

- And do the same for each part of speech ...

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Emission Probability β

Let's look at verbs . . .

Word	a	and	come	crowd	flattop
Frequency	0.1	0.1	1.1	0.1	0.1
Probability	0.0125	0.0125	0.1375	0.0125	0.0125

Word	get	gotta	her	here	i
Frequency	1.1	1.1	0.1	0.1	0.1
Probability	0.1375	0.1375	0.0125	0.0125	0.0125

Word	into	it	life	love	my
Frequency	0.1	0.1	0.1	1.1	0.1
Probability	0.0125	0.0125	0.0125	0.1375	0.0125

Word	of	old	people	stared	stopped
Frequency	0.1	0.1	0.1	1.1	1.1
Probability	0.0125	0.0125	0.0125	0.1375	0.1375

Next time . . .

- Viterbi algorithm: dynamic algorithm discovering the most likely pos sequence given a sentence
- em algorithm: what if we don't have labeled data?

Sequence Models

Natural Language Processing: Jordan Boyd-Graber
University of Maryland

RNNS

Slides adapted from Richard Socher

N-Gram Language Models

- Given: a string of English words $W = w_1, w_2, w_3, ..., w_n$
- Question: what is $p(W)$?
- Sparse data: Many good English sentences will not have been seen before

⟶ Decomposing $p(W)$ using the chain rule:

$$p(w_1, w_2, w_3, ..., w_n) = \\ p(w_1)\, p(w_2|w_1)\, p(w_3|w_1, w_2) \dots p(w_n|w_1, w_2, ...w_{n-1})$$

(not much gained yet, $p(w_n|w_1, w_2, ...w_{n-1})$ is equally sparse)

Markov Chain

- **Markov independence assumption**:
    - □ only previous history matters
    - □ limited memory: only last k words are included in history (older words less relevant)

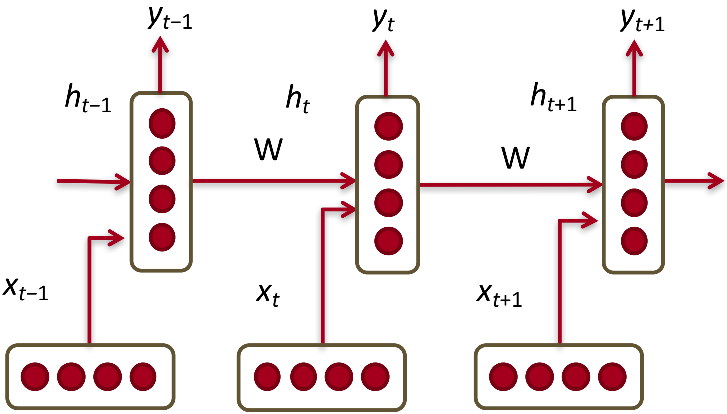
  ⟶ k**th order Markov model**

- For instance 2-gram language model:

$$p(w_1, w_2, w_3, ..., w_n) \simeq p(w_1)\, p(w_2|w_1)\, p(w_3|w_2)...p(w_n|w_{n-1})$$

- What is conditioned on, here w_{i-1} is called the **history**. Estimated from counts.

Recurrent Neural Networks



- Condition on all previous words
- Hidden state at each time step

RNN parameters

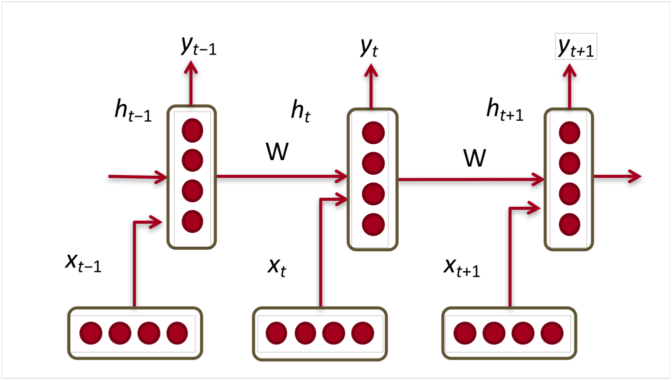
$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t) \tag{1}$$

$$\hat{y}_t = \text{softmax}(W^{(S)} h_t) \tag{2}$$

$$P(x_{t+1} = v_j \mid x_t, \dots x_1) = \hat{y}_{t,j} \tag{3}$$

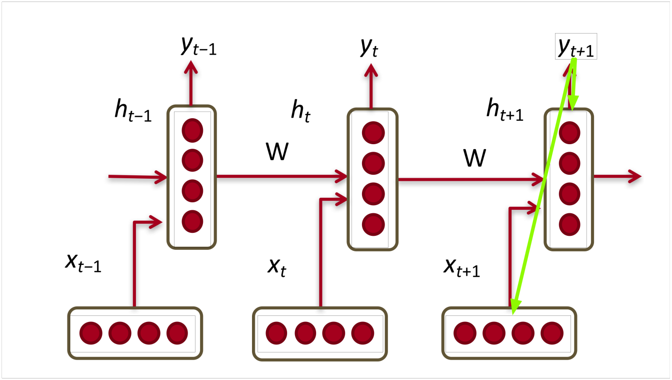
- Learn parameter h_0 to initialize hidden layer
- x_t is representation of input (e.g., word embedding)
- $\hat{y}$ is probability distribution over vocabulary

Training Woes



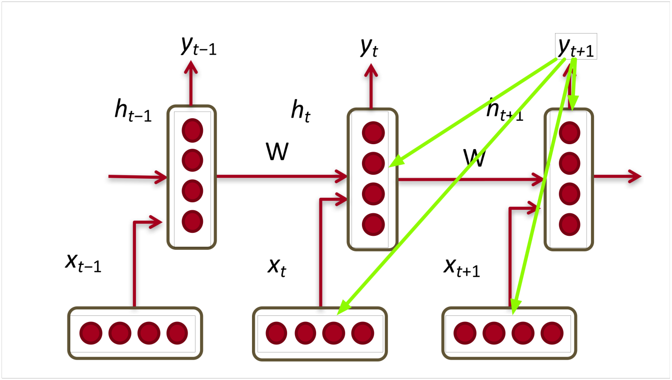
Multiplying same matrix over and over

Training Woes



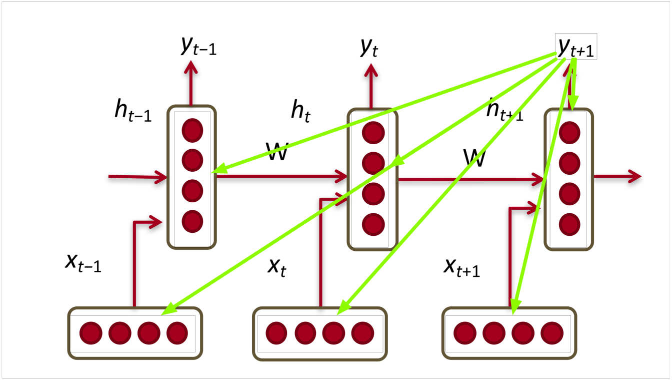
Multiplying same matrix over and over

Training Woes



Multiplying same matrix over and over

Training Woes

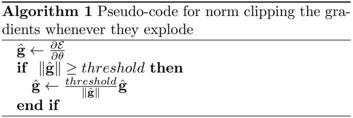


Multiplying same matrix over and over

Vanishing / Exploding Gradient

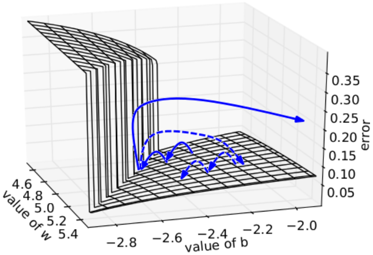
- Work out the math:
  - Define β_W / β_h as upper bound of norms of W, h
  - Bengio et al 1994: Partial derivative is $(\beta_W \beta_h)^{t-k}$
  - This can be very small or very big
- If it's big, SGD jumps too far
- If it's small, we don't learn what we need: "Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____"

Gradient Clipping



Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

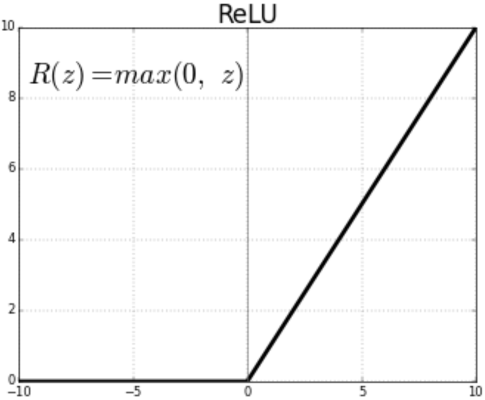
```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if
```



From Pascanu et al. 2013

- If they get too big, stop at boundary
- Prevents (dashed) values from jumping around (solid)

Fixing Vanishing Gradients



- ReLU activation
- Initialize W to identity matrix

RNN Recap

- Simple model
- Complicated training (but good toolkits available)
- Do we need to remember everything?

Sequence Models

Natural Language Processing: Jordan Boyd-Graber
University of Maryland

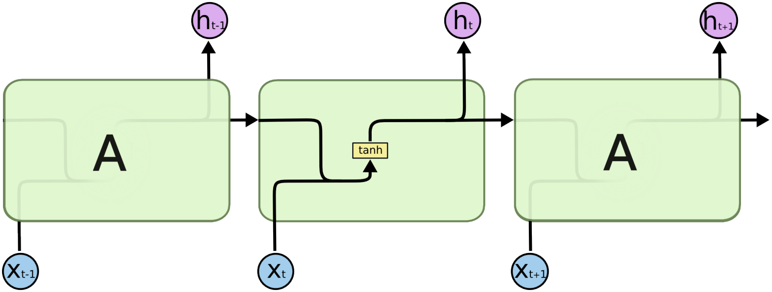
LSTMS

Slides adapted from Christopher Olah

The Model of Laughter and Forgetting

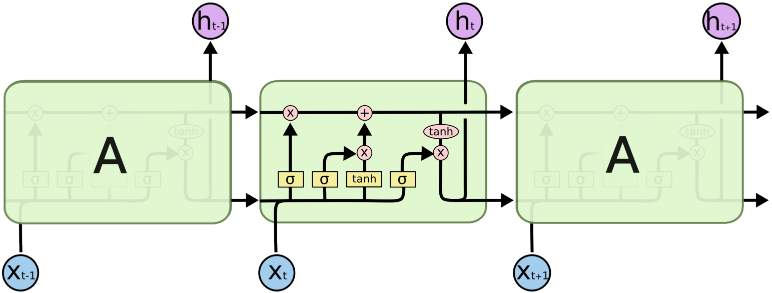
- RNN is great: can remember anything
- RNN stinks: remembers everything
- Sometimes important to forget: LSTM

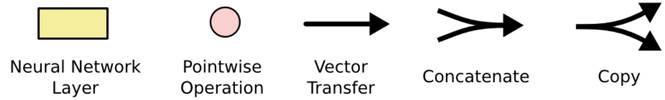
RNN transforms Input into Hidden



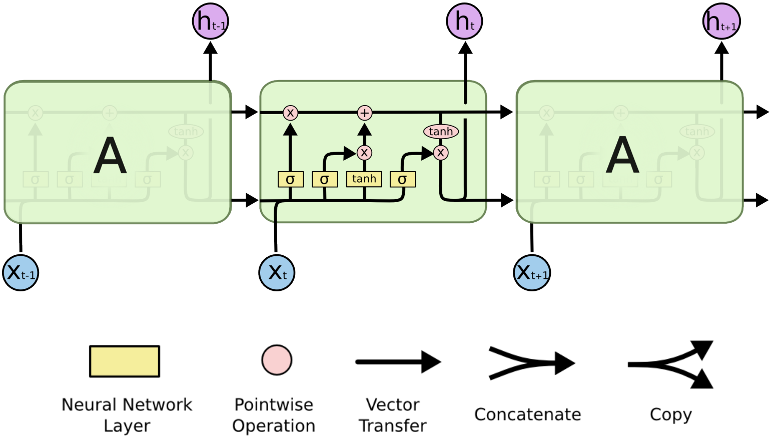
(Can be other nonlinearities)

LSTM has more complicated innards



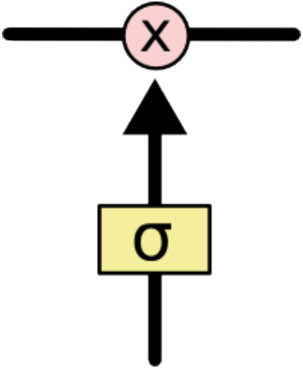


LSTM has more complicated innards



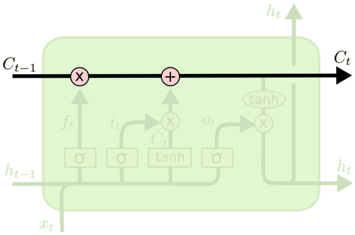
Built on gates!

Gates



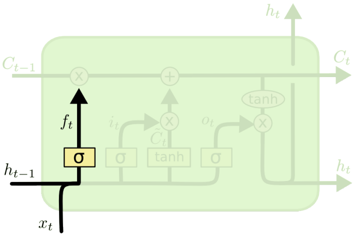
- Multiply vector dimension by value in $[0, 1]$
- Zero means: forget everything
- One means: carry through unchanged
- LSTM has three different gates

Cell State



Can pass through (memory)

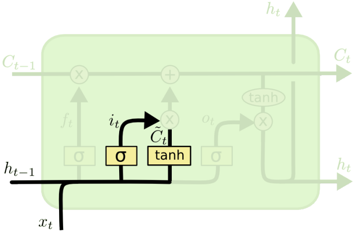
Deciding When to Forget



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

Based on previous hidden state h_{t-1}, can decide to forget past cell state

Updating representation

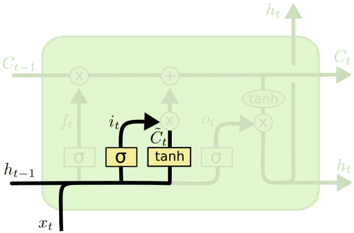


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Compute new contribution to cell state based on hidden state h_{t-1} and input x_t

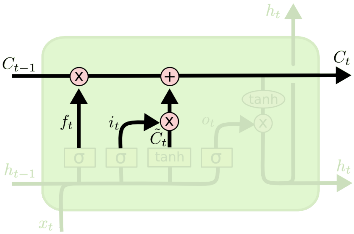
Updating representation



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

Compute new contribution to cell state based on hidden state h_{t-1} and input x_t. Strength of contribution is i_t

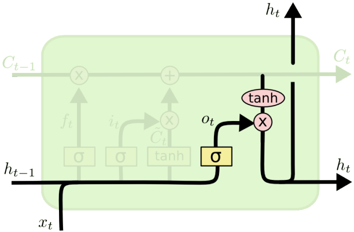
Updating representation



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

Interpolate new cell value

Output hidden



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

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

Hidden layer is function of cell C_t, not h_{t-1}