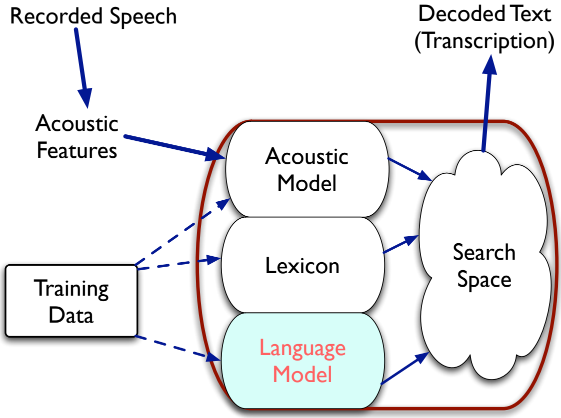
Language Modelling

Steve Renals

Automatic Speech Recognition – ASR Lecture 11
6 March 2017

HMM Speech Recognition



Language modelling

- Basic idea The language model is the prior probability of the word sequence $P(W)$
- Use a language model to disambiguate between similar acoustics when combining linguistic and acoustic evidence
  recognize speech / *wreck a nice beach*
- Use hand constructed networks in limited domains
- Statistical language models: cover "ungrammatical" utterances, computationally efficient, trainable from huge amounts of data, can assign a probability to a sentence fragment as well as a whole sentence

Statistical language models

- For use in speech recognition a language model must be: statistical, have wide coverage, and be compatible with left-to-right search algorithms
- Only a few grammar-based models have met this requirement (eg Chelba and Jelinek, 2000), and do not yet scale as well as simple statistical models
- Until very recently **n-grams** were the state-of-the-art language model for ASR
  - Unsophisticated, linguistically implausible
  - Short, finite context
  - Model solely at the shallow word level
  - But: wide coverage, able to deal with "ungrammatical" strings, statistical and scaleable
- Probability of a word depends only on the identity of that word and of the preceding n-1 words. These short sequences of n words are called n-grams.

Bigram language model

- Word sequence $\mathbf{W} = w_1, w_2, \ldots w_M$

$$P(\mathbf{W}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \\ \ldots P(w_M \mid w_1, w_2, \ldots w_{M-1})$$

- Bigram approximation—consider only one word of context:

$$P(\mathbf{W}) \simeq P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2) \ldots P(w_M \mid w_{M-1})$$

Bigram language model

- Word sequence $\mathbf{W} = w_1, w_2, \ldots w_M$

$$P(\mathbf{W}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \\ \ldots P(w_M \mid w_1, w_2, \ldots w_{M-1})$$

- Bigram approximation—consider only one word of context:

$$P(\mathbf{W}) \simeq P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2) \ldots P(w_M \mid w_{M-1})$$

- Parameters of a bigram are the conditional probabilities $P(w_i \mid w_j)$
- Maximum likelihood estimates by counting:

$$P(w_i|w_j) \sim \frac{c(w_j, w_i)}{c(w_j)}$$

where $c(w_j, w_i)$ is the number of observations of w_j followed by w_i, and $c(w_j)$ is the number of observations of w_j (irrespective of what follows)

The zero probability problem

- Maximum likelihood estimation is based on counts of words in the training data
- If a n-gram is not observed, it will have a count of 0—and the maximum likelihood probability estimate will be 0
- The zero probability problem: just because something does not occur in the training data does not mean that it will not occur
- As n grows larger, so the data grow sparser, and the more zero counts there will be

The zero probability problem

- Maximum likelihood estimation is based on counts of words in the training data
- If a n-gram is not observed, it will have a count of 0—and the maximum likelihood probability estimate will be 0
- The zero probability problem: just because something does not occur in the training data does not mean that it will not occur
- As n grows larger, so the data grow sparser, and the more zero counts there will be
- Solution: smooth the probability estimates so that unobserved events do not have a zero probability
- Since probabilities sum to 1, this means that some probability is redistributed from observed to unobserved n-grams

Smoothing language models

- What is the probability of an unseen n-gram?

Smoothing language models

- What is the probability of an unseen n-gram?
- Add-one smoothing: add one to all counts and renormalize.
  - "Discounts" non-zero counts and redistributes to zero counts
  - Since most n-grams are unseen (for large n more types than tokens!) this gives too much probability to unseen n-grams (discussed in Manning and Schütze)
- Absolute discounting: subtract a constant from the observed (non-zero count) n-grams, and redistribute this subtracted probability over the unseen n-grams (zero counts)
- Kneser-Ney smoothing: family of smoothing methods based on absolute discounting that are at the state of the art (Goodman, 2001)

Backing off

- How is the probability distributed over unseen events?
- Basic idea: estimate the probability of an unseen n-gram using the (n-1)-gram estimate
- Use successively less context: trigram $\rightarrow$ bigram $\rightarrow$ unigram
- Back-off models redistribute the probability "freed" by discounting the n-gram counts

Backing off

- How is the probability distributed over unseen events?
- Basic idea: estimate the probability of an unseen n-gram using the (n-1)-gram estimate
- Use successively less context: trigram $\rightarrow$ bigram $\rightarrow$ unigram
- Back-off models redistribute the probability "freed" by discounting the n-gram counts
- For a bigram

$$
\begin{aligned}
P(w_i \mid w_j) &= \frac{c(w_j, w_i) - D}{c(w_j)} \quad \text{if } c(w_j, w_i) > c \\
&= P(w_i) b_{w_j} \qquad \textit{otherwise}
\end{aligned}
$$

c is the count threshold, and D is the discount. b_{w_j} is the backoff weight required for normalization

Interpolation

- Basic idea: Mix the probability estimates from all the estimators: estimate the trigram probability by mixing together trigram, bigram, unigram estimates
- Simple interpolation

$$\hat{P}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_3 P(w_n \mid w_{n-2}, w_{n-1}) + \lambda_2 P(w_n \mid w_{n-1}) + \lambda_1 P(w_n)$$

  With $\sum_i \lambda_i = 1$
- Interpolation with coefficients conditioned on the context

$$\hat{P}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_3(w_{n-2}, w_{n-1}) P(w_n \mid w_{n-2}, w_{n-1}) + \lambda_2(w_{n-2}, w_{n-1}) P(w_n \mid w_{n-1}) + \lambda_1(w_{n-2}, w_{n-1}) P(w_n)$$

- Set λ values to maximise the likelihood of the interpolated language model generating a *held-out* corpus (possible to use EM to do this)

Perplexity

- Measure the quality of a language model by how well it predicts a test set W (i.e. estimated probability of word sequence)
- Perplexity ($PP(W)$) – inverse probability of the test set W, normalized by the number of words N

$$PP(W) = P(W)^{\frac{-1}{N}} = P(w_1 w_2 \ldots w_N)^{\frac{-1}{N}}$$

- Perplexity of a bigram LM

$$PP(W) = (P(w_1)P(w_2|w_1)P(w_3|w_2) \ldots P(w_N|w_{N-1}))^{\frac{-1}{N}}$$

- Example perplexities for different n-gram LMs trained on Wall St Journal (38M words)
  - Unigram – 962
  - Bigram – 170
  - Trigram – 109

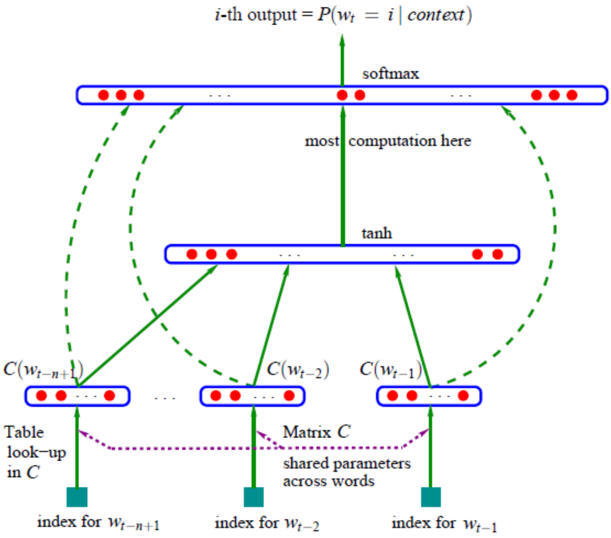
Practical language modelling

- Work in log probabilities
- The ARPA language model format is commonly used to store n-gram language models (unless they are very big)
- Many toolkits: SRILM, IRSTLM, KenLM, Cambridge-CMU toolkit, ...
- Some research issues:
  - Advanced smoothing
  - Adaptation to new domains
  - Incorporating topic information
  - Long-distance dependencies
  - Distributed representations

Distributed representation for language modelling

- Each word is associated with a learned *distributed representation* (feature vector)
- Use a neural network to estimate the conditional probability of the next word given the the distributed representations of the context words
- Learn the distributed representations and the weights of the conditional probability estimate jointly by maximising the log likelihood of the training data
- Similar words (distributionally) will have similar feature vectors — small change in feature vector will result in small change in probability estimate (since the NN is a smooth function)

Neural Probabilistic Language Model



Bengio et al (2006)

Neural Probabilistic Language Model

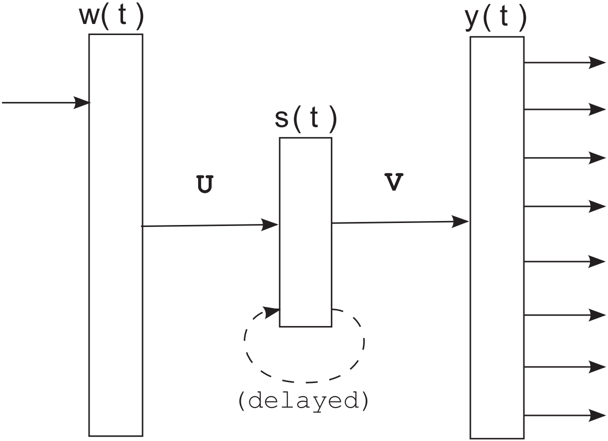
- Train using stochastic gradient ascent to maximise log likelihood
- Number of free parameters (weights) scales
  - Linearly with vocabulary size
  - Linearly with context size
- Can be (linearly) interpolated with n-gram model
- Perplexity results on AP News (14M words training). $|V| = 18k$

model	**n**	**perplexity**
NPLM(100,60)	6	109
n-gram (KN)	3	127
n-gram (KN)	4	119
n-gram (KN)	5	117

Recurrent Neural Network (RNN) LM

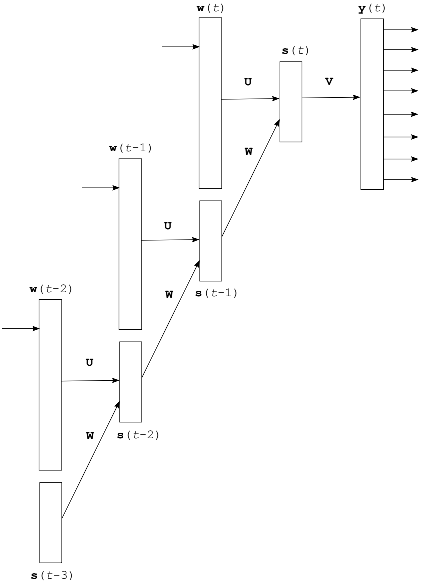
- Rather than fixed input context, *recurrently connected* hidden units provide memory
- Model learns "how to remember" from the data
- Recurrent hidden layer allows clustering of variable length histories

RNN LM



Mikolov (2011)

RNN training: back-propagation through time



Reducing computation at the output layer

Majority of the weights (hence majority of the computation) is in the output layer – potentially V units wide, where V is vocabulary size

1. Model fewer words
   - **Shortlist**: use the NN to model only the most frequent words
2. Structure the output layer
   - **Factorization of the output layer**: first estimate the probability over word classes then over words within the selected class
   - **Hiearchical softmax**: structure the output layer as a binary tree
3. Efficiently estimate the normalised outputs
   - **Noise contrastive estimation**: train each output unit as an independent binary classifier

Shortlists

- Reduce computation by only including the s most frequent words at the output — the *shortlist* (S) (full vocabulary still used for context)
- Use an n-gram model to estimate probabilities of words not in the shortlist
- Neural network thus redistributes probability for the words in the shortlist

$$P_S(h_t) = \sum_{w \in S} P(w|h_t)$$

$$P(w_t|h_t) = \begin{cases} P_{NN}(w_t|h_t)P_S(h_t) & \text{if} w_t \in S \\ P_{KN}(w_t|h_t) & \text{else} \end{cases}$$

- In a $|V| = 50k$ task a 1024 word shortlist covers 89% of 4-grams, 4096 words covers 97%

NPLM — ASR results

Speech recognition results on Switchboard

7M / 12M / 27M words in domain data.

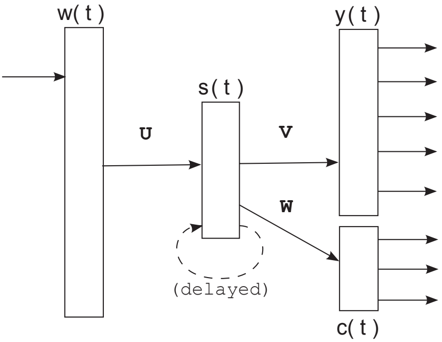
500M words background data (broadcast news)

Vocab size $|V| = 51k$, Shortlist size $|S| = 12k$

WER/%			
in-domain words	7M	12M	27M
KN (in-domain)	25.3	23.0	20.0
NN (in-domain)	24.5	22.2	19.1
KN (+b/g)	24.1	22.3	19.3
NN (+b/g)	23.7	21.8	18.9

Factorised RNN LM

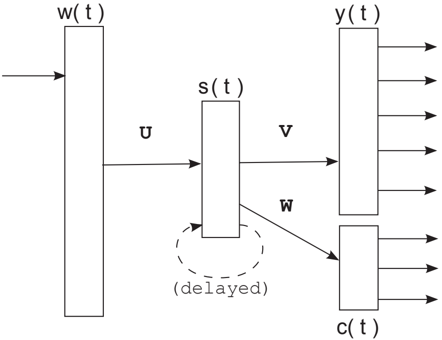
Mikolov 2011



$$P(w_i|hist) = P(c_i|s(t))P(w_i|c_i, s(t))$$

1. Compute a probability distribution over C classes
2. Compute a probability distribution over $V' \leq V$ words in the class

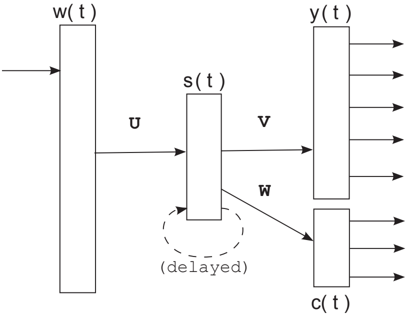
Factorised RNN LM



$$P(w_i|hist) = P(c_i|s(t))P(w_i|c_i, s(t))$$

- Instead of doing softmax over V elements, only $C + V'$ outputs have to be computed, and the softmax function is applied separately to the classes and the words in that class.
- C is constant; V can be variable.

Classes for a factorised RNN LM



$$P(w_i|hist) = P(c_i|s(t))P(w_i|c_i, s(t))$$

- Each word is assigned to a single class based on unigram probabilities – "frequency binning"
- Most frequent words in class 1: V' is small for that class
- Rarest words in class C: V' is large for that class but the words are infrequent

Perplexity Results

Table 2. *Comparison of different neural network architectures on Penn Corpus (1M words) and Switchboard (4M words).*

	Penn Corpus		Switchboard	
Model	NN	NN+KN	NN	NN+KN
KN5 (baseline)	-	141	-	92.9
feedforward NN	141	118	85.1	77.5
RNN trained by BP	137	113	81.3	75.4
RNN trained by BPTT	123	106	77.5	72.5

Factorised output layer an speedup training by 25x for 100k vocabulary, with minimal effect on perplexity

Hierarchical Softmax

Bengio 2006

- Push the class-based factorization idea to the limit
- Class-based factorization is 1-level structuring
- "Classes of classes" – 2 level structuring
- Balanced binary tree – n level structuring ($n \sim \log_2 V$), each leaf is a word
- Each node of the tree is a 2-class classifier – make probabilistic binary decisions
- Need only consider the $\log_2 V$ nodes on the path from the root to the leaf for each word

$$P(w|hist) = \prod_{j=1}^{n} P(b_j(v)|b_1(v), \ldots, b_{j-1}(v), hist)$$

Noise contrastive estimation (NCE)

Chen et al (2015) (not the original source, but a clear application to ASR)

- Aim: avoid directly computing the normalisation term (denominator) in softmax (involves summing over V units)
- Method: treat each output unit separately, as sigmoid classifier between the observed data (for that word) and a "noise" distribution
- Assume that data for a history h generated by a mixture of an RNNLM distribution $P_{RNN}(\cdot|h)$ and a noise distribution $P_n(\cdot|h)$ – typically, P_n is a unigram
- Each node computes the posterior probability of whether a word sample w comes from the RNNLM or the noise:

$$P(C_w^{RNN} = 1|w, h) = \frac{P_{RNN}(w|h)}{P_{RNN}(w|h) + kP_n(w|h)}$$

$$P(C_w^n = 1|w, h) = 1 - P(C_w^{RNN} = 1|w, h)$$

Noise contrastive estimation (NCE)

- NCE training minimises this cost function:

$$E = -\frac{1}{N_w}\sum_{i=1}^{N_w}\left(\ln P(C_w^{RNN} = 1|w,h) + \sum_{j=1}^{k}\ln P(C_w^{n} = 1|w,h)\right)$$

- k samples drawn from the unigram noise distribution for the current word; typically $k \sim 10$
- The RNNLM distribution is given by

$$P_{RNN}(w_i|h) = \frac{\exp(\mathbf{v}\,\mathbf{s}(t))}{Z}$$

The normalisation term Z is learned; in practice it may be set to a constant for all contexts

State-of-the-art (2016)

Jozefowicz et al (2016), "Exploring the Limits of Language Modeling", http://arxiv.org/abs/1602.02410. (Google)

- Experiments on One Billion Word Benchmark data set, with 800k vocabulary
- Large-scale language modeling experiments, comparing
  - 5-gram model (interpolated Kneser-Ney smoothing)
  - RNN with sigmoid transfer functions
  - Various LSTM recurrent network models
    - "Size matters... The best models are the largest we were able to fit into a GPU memory."
    - Best performing model had two LSTM recurrent layers with 8192 and 1024 units ($\sim$ 1.8 billion parameters)
  - RNN models used a variant of NCE at the output layer
  - Also obtained more compact and slightly better performing models using convolutional layers over characters at input and output

Results: Single models

Model	Test Perplexity
Sigmoid-RNN-2048 (Ji et al., 2015a)	68.3
Interpolated KN 5-gram, 1.1B n-grams (Chelba et al., 2013)	67.6
Sparse Non-Negative Matrix LM (Shazeer et al., 2015)	52.9
RNN-1024 + MaxEnt 9-gram features (Chelba et al., 2013)	51.3
LSTM-512-512	54.1
LSTM-1024-512	48.2
LSTM-2048-512	43.7
LSTM-8192-2048 (No Dropout)	37.9
LSTM-8192-2048 (50% Dropout)	32.2
2-Layer LSTM-8192-1024 (BIG LSTM)	30.6
BIG LSTM+CNN Inputs	**30.0**
BIG LSTM+CNN Inputs + CNN Softmax	39.8
BIG LSTM+CNN Inputs + CNN Softmax + 128-dim correction	35.8
BIG LSTM+CNN Inputs + Char LSTM predictions	47.9

Results: Ensembles of models

Model	Test Perplexity
Large Ensemble (Chelba et al., 2013)	43.8
RNN+KN-5 (Williams et al., 2015)	42.4
RNN+KN-5 (Ji et al., 2015a)	42.0
RNN+SNM10-skip (Shazeer et al., 2015)	41.3
Large Ensemble (Shazeer et al., 2015)	41.0
Our 10 best LSTM models (equal weights)	26.3
Our 10 best LSTM models (optimal weights)	26.1
10 LSTMs + KN-5 (equal weights)	25.3
10 LSTMs + KN-5 (optimal weights)	25.1
10 LSTMs + SNM10-skip (Shazeer et al., 2015)	**23.7**

Reading

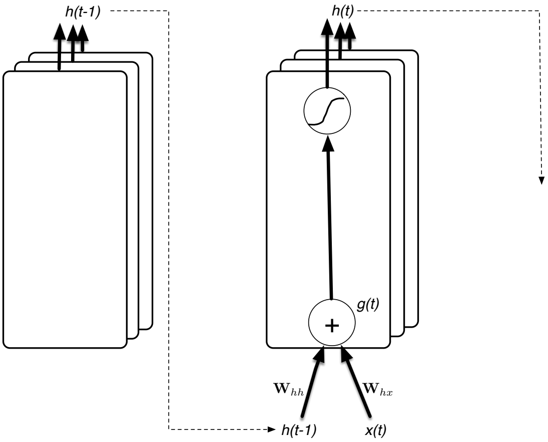
- Jurafsky and Martin, chapter 4
- Y Bengio et al (2006), "Neural probabilistic language models" (sections 6.1, 6.2, 6.3, 6.6, 6.7, 6.8), Studies in Fuzziness and Soft Computing Volume 194, Springer, chapter 6. http://link.springer.com/chapter/10.1007/3-540-33486-6_6
- T Mikolov et al (2011), "Extensions of recurrent neural network language model", ICASSP–2011. http://ieeexplore.ieee.org/document/5947611
- X Chen et al (2015), "Recurrent neural network language model training with noise contrastive estimation for speech recognition", ICASSP-2015. http://mi.eng.cam.ac.uk/~xc257/papers/ICASSP2015-rnnlm-nce.pdf
- R Jozefowicz et al (2016), "Exploring the Limits of Language Modeling", http://arxiv.org/abs/1602.02410.

Annex: LSTM Recurrent Networks

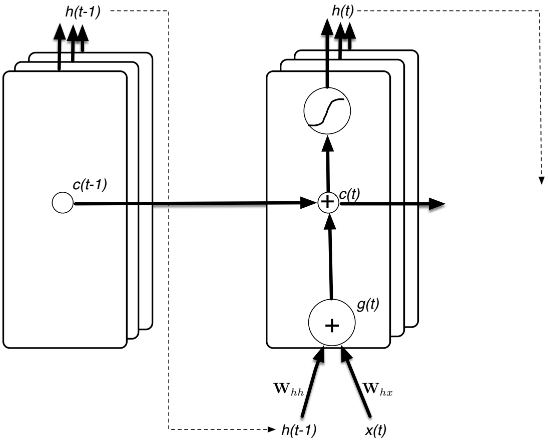
LSTM

- **Internal recurrent state** ("cell") $c(t)$ combines previous state $c(t-1)$ and LSTM input $g(t)$

LSTM



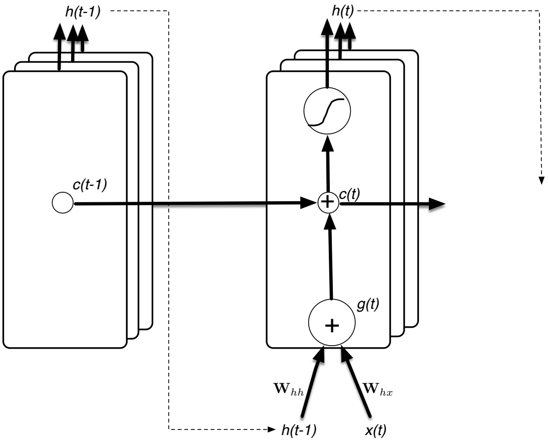
LSTM – Internal recurrent state



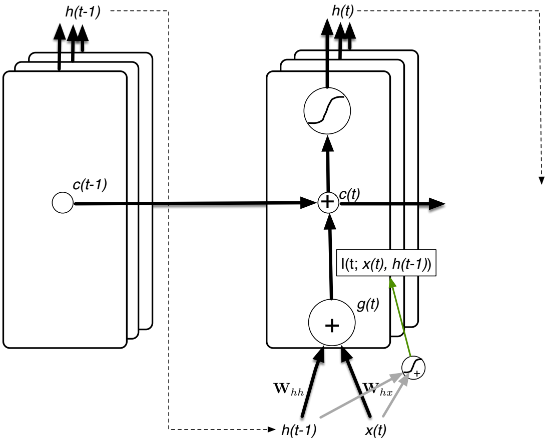
LSTM

- **Internal recurrent state** ("cell") $c(t)$ combines previous state $c(t-1)$ and LSTM input $g(t)$
- Gates - weights dependent on the current input and the previous state
- **Input gate**: controls how much input to the unit $g(t)$ is written to the internal state $c(t)$
- **Forget gate**: controls how much of the previous internal state $c(t-1)$ is written to the internal state $c(t)$
    - Input and forget gates together allow the network to control what information is stored and overwritten at each step

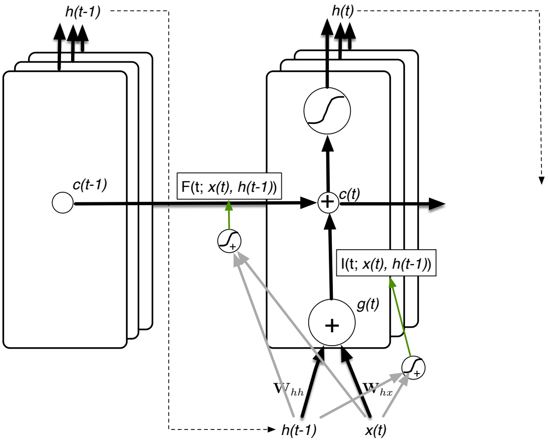
LSTM



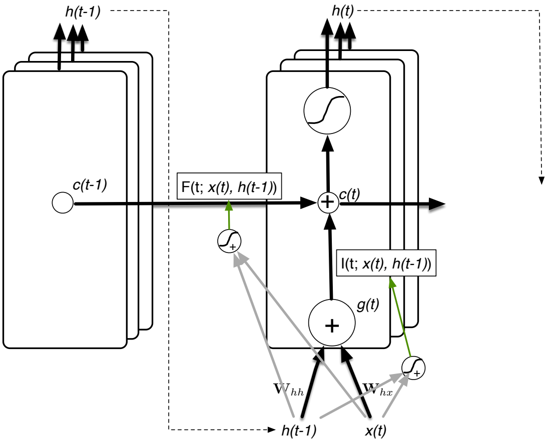
LSTM – Input Gate



LSTM – Forget Gate



LSTM – Input and Forget Gates



$$\mathbf{I}(t) = \sigma\left(\mathbf{W}_{ix}\mathbf{x}(t) + \mathbf{W}_{ih}\mathbf{h}(t-1) + \mathbf{b}_i\right)$$

$$\mathbf{F}(t) = \sigma\left(\mathbf{W}_{fx}\mathbf{x}(t) + \mathbf{W}_{fh}\mathbf{h}(t-1) + \mathbf{b}_f\right)$$

σ is the sigmoid function

$$\mathbf{g}(t) = \mathbf{W}_{hx}\mathbf{x}(t) + \mathbf{W}_{hh}\mathbf{h}(t-1) + \mathbf{b}_h$$

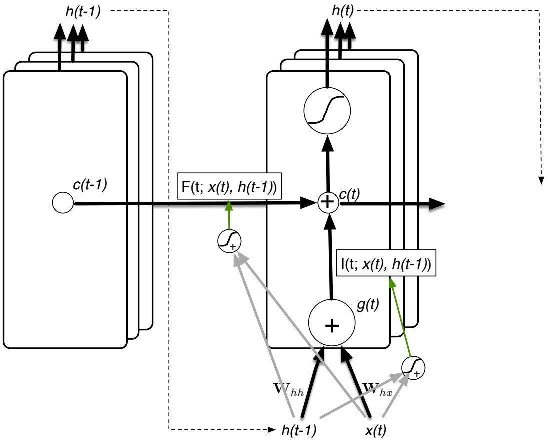
$$\mathbf{c}(t) = \mathbf{F}(t) \circ \mathbf{c}(t-1) + \mathbf{I}(t) \circ \mathbf{g}(t)$$

$\circ$ is element-wise vector multiply

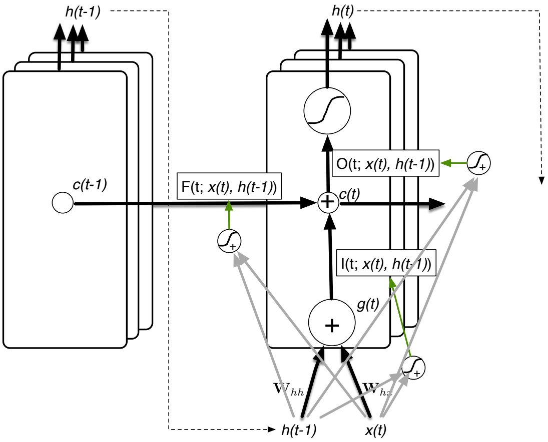
LSTM

- **Internal recurrent state** ("cell") $c(t)$ combines previous state $c(t-1)$ and LSTM input $g(t)$
- Gates - weights dependent on the current input and the previous state
- **Input gate**: controls how much input to the unit $g(t)$ is written to the internal state $c(t)$
- **Forget gate**: controls how much of the previous internal state $c(t-1)$ is written to the internal state $c(t)$
  - Input and forget gates together allow the network to control what information is stored and overwritten at each step
- **Output gate**: controls how much of each unit's activation is output by the hidden state – it allows the LSTM cell to kepp information that is not relevant at the current time, but may be relevant later

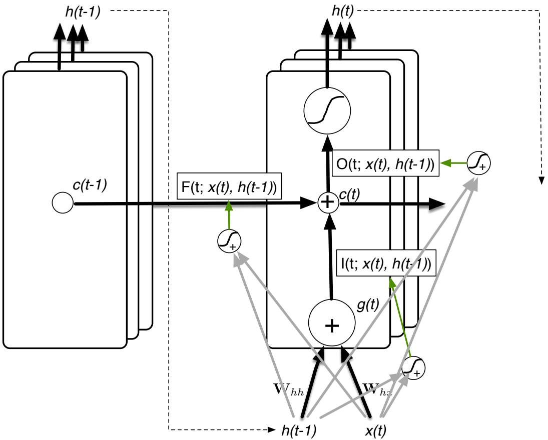
LSTM – Input and Forget Gates



LSTM – Output Gate

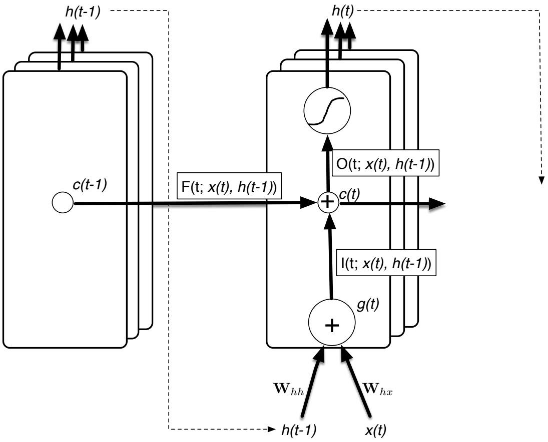


LSTM – Output Gate



$$\mathbf{O}(t) = \sigma\left(\mathbf{W}_{ox}\mathbf{x}(t) + \mathbf{W}_{oh}\mathbf{h}(t-1) + \mathbf{b}_o\right) \qquad \mathbf{h}(t) = \tanh\left(\mathbf{O}(t) \circ \mathbf{c}(t)\right)$$

LSTM



$$\mathbf{I}(t) = \sigma\left(\mathbf{W}_{ix}\mathbf{x}(t) + \mathbf{W}_{ih}\mathbf{h}(t-1) + \mathbf{b}_i\right) \quad \mathbf{g}(t) = \mathbf{W}_{hx}\mathbf{x}(t) + \mathbf{W}_{hh}\mathbf{h}(t-1) + \mathbf{b}_h$$

$$\mathbf{F}(t) = \sigma\left(\mathbf{W}_{fx}\mathbf{x}(t) + \mathbf{W}_{fh}\mathbf{h}t-1) + \mathbf{b}_f\right) \quad \mathbf{c}(t) = \mathbf{F}(t) \circ \mathbf{c}(t-1) + \mathbf{I}(t) \circ \mathbf{g}(t)$$

$$\mathbf{O}(t) = \sigma\left(\mathbf{W}_{ox}\mathbf{x}(t) + \mathbf{W}_{oh}\mathbf{h}(t-1) + \mathbf{b}_o\right) \quad \mathbf{h}(t) = \tanh\left(\mathbf{O}(t) \circ \mathbf{c}(t)\right)$$