AdvNLP/E Lecture 4

Language Modelling 2

Dr Julie Weeds, Spring 2024

Language Models

PREVIOUSLY

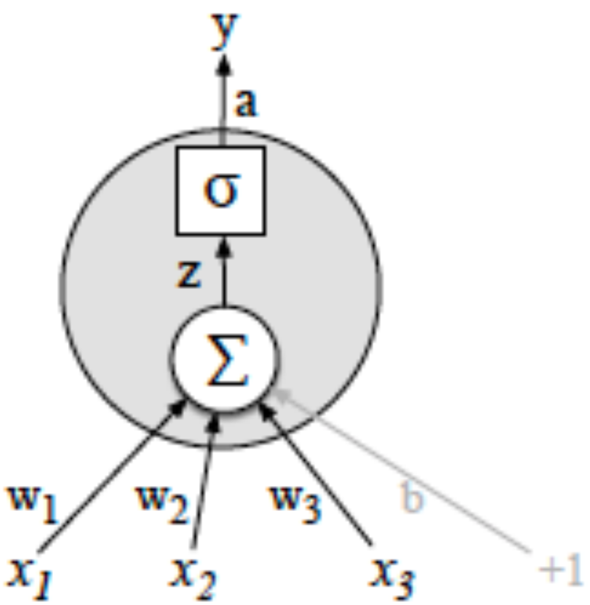
- N-gram language modelling
  - Markov assumptions
  - perplexity and evaluation
  - smoothing

THIS TIME

- Neural language models
  - feed forward
  - recurrent
  - long-short term memory
  - convolutional
  - word-based vs character-based

Neural Networks

A Neural Unit



y is the output and is equal to the activation of the unit

$$y = f(z)$$

z is a weighted sum of inputs

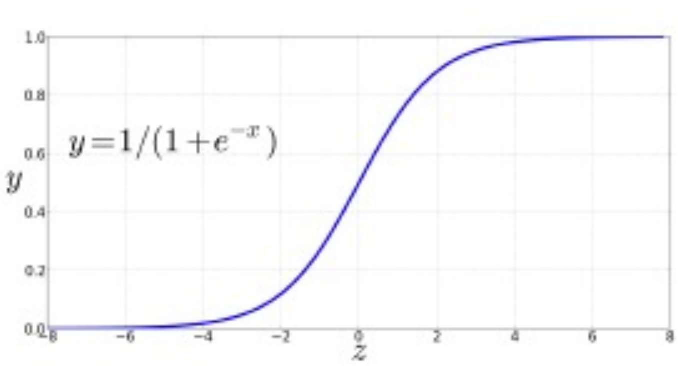
$$\boxed{z = w.x + b}$$

w is the weight vector (same dimensions as input)

x is the input vector (3 dimensions here)

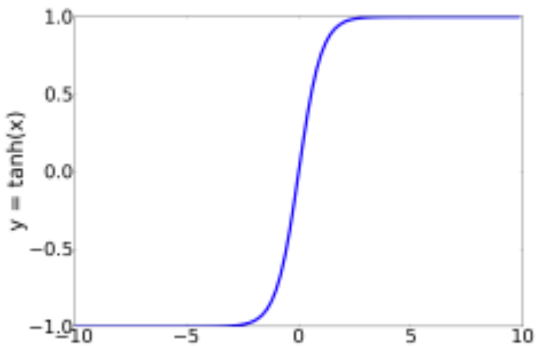
Activation functions

- Introduce non-linearity



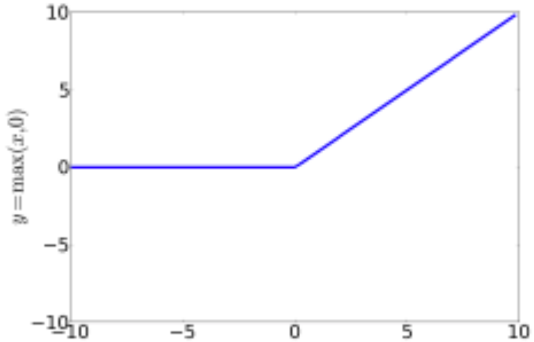
sigmoid

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$



tanh

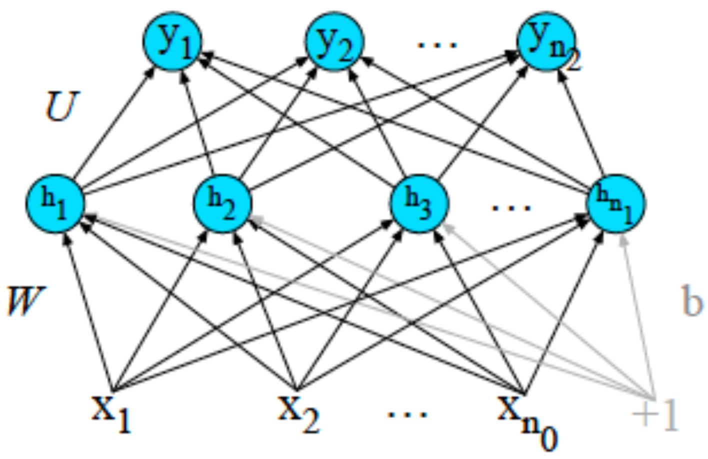
$$y = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



ReLU

$$y = \max(z, 0)$$

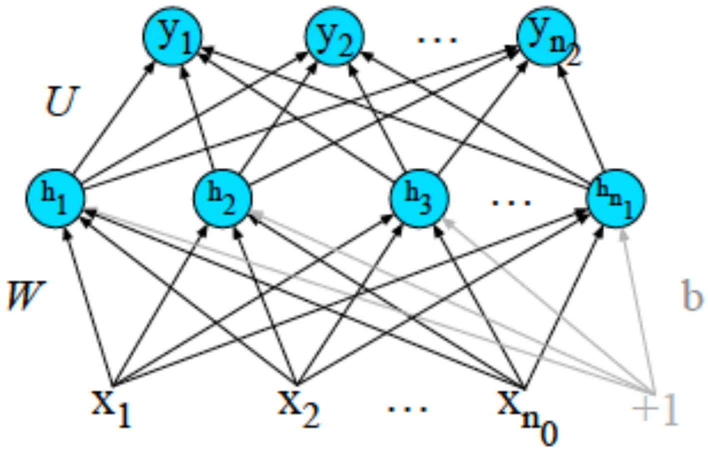
Feed-forward network



2-layer FF network with one hidden layer and one output layer

- Multi-layer network of connected neural units
- no cycles
- outputs from one layer are inputs to the next layer
- each unit takes a weighted sum of inputs and applies non-linearity
- fully-connected if each unit in each layer takes as input, outputs from all units in previous layer

Feed-forward network



2-layer FF network with one hidden layer and one output layer

$$y = \mathrm{softmax}(Uh)$$

$$h = \sigma(Wx + b)$$

Softmax is used to turn the outputs into a probability distribution

$$\boxed{y = \mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}}$$

Training neural networks

- For a **hard classification** task (i.e., one where there is only one correct answer)
  - **cross-entropy loss** or **negative log-likelihood loss** is simply the log probability, computed by the network, of the correct class. Let $\hat{y}$ be the output vector from the network and y be a k-dimensional one-hot vector encoding the correct answer (i)

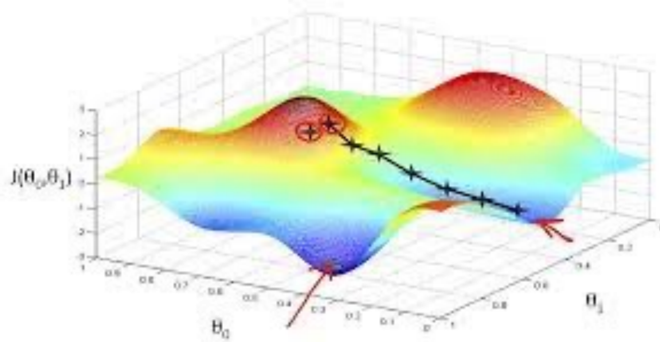
$$\mathrm{J} = \mathrm{Loss}(\hat{y}, y) = -\sum_{i=1}^{k} y_i \log \hat{y}_i = -\log \hat{y}_i$$

  - we want to find the parameters / weights which **minimize the loss function**
    - *loss functions* are also sometimes referred to as *cost functions* and *objective functions*. Loss or cost function should always refer to penalties which we want to minimize. An objective function is anything we want to optimize on (and could include a reward function we want to maximise)

Gradient descent

- Randomly initialise parameter settings (W)
- Compute predictions
- Compute loss function (J)
- Compute (partial) derivatives of loss function with respect to parameters
- Back-propagate i.e. update parameters:

$$\boxed{W = W - \underbrace{\alpha}_{\text{learning rate}} \frac{dJ}{dW}}$$



- Repeat until convergence / loss is acceptably small

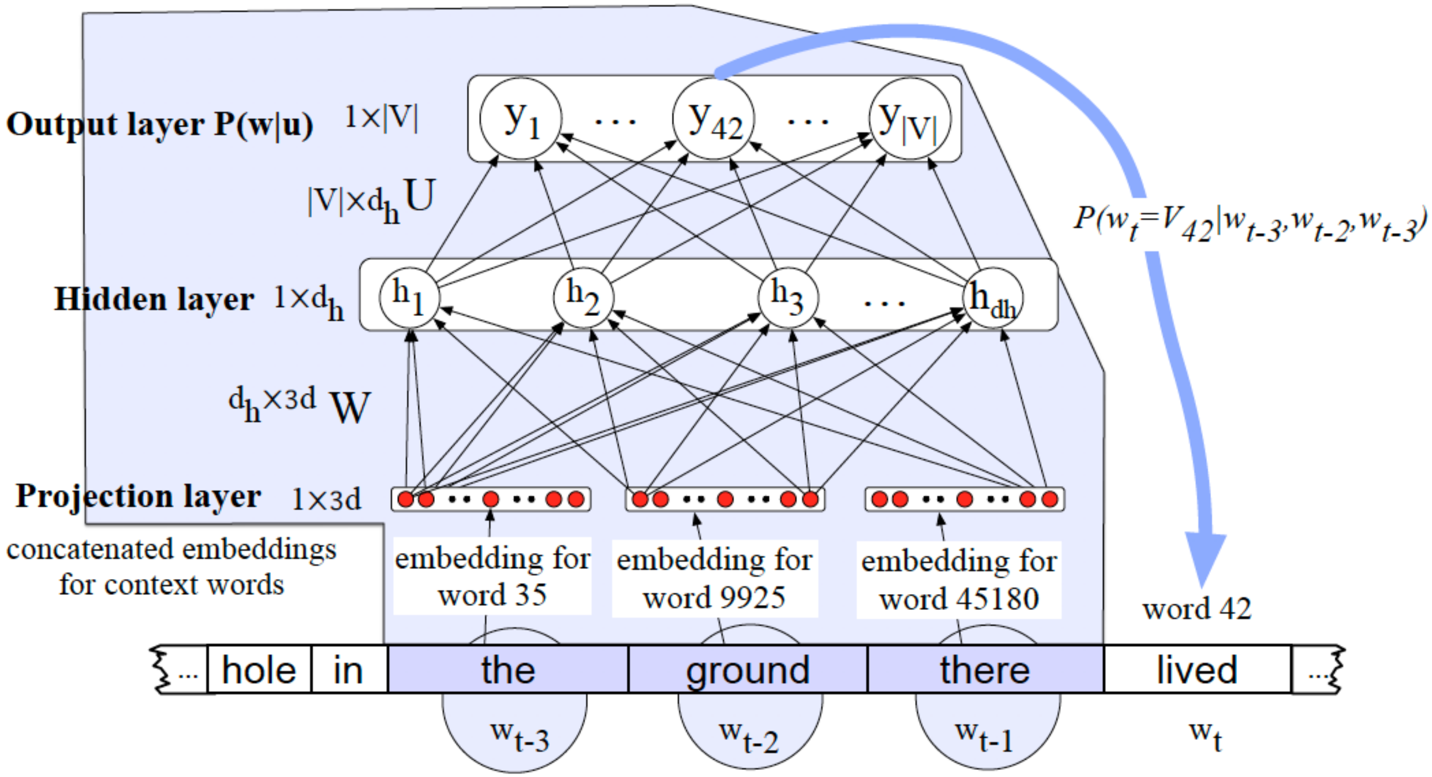
FF-NLM

Feed-forward neural language model

FF-NLMs (Bengio et al. 2003)

Output at time t: a probability distribution over possible next words

Input at time t: a representation of some number of previous words (w_{t-1}, w_{t-2}, etc)



FF-NLM

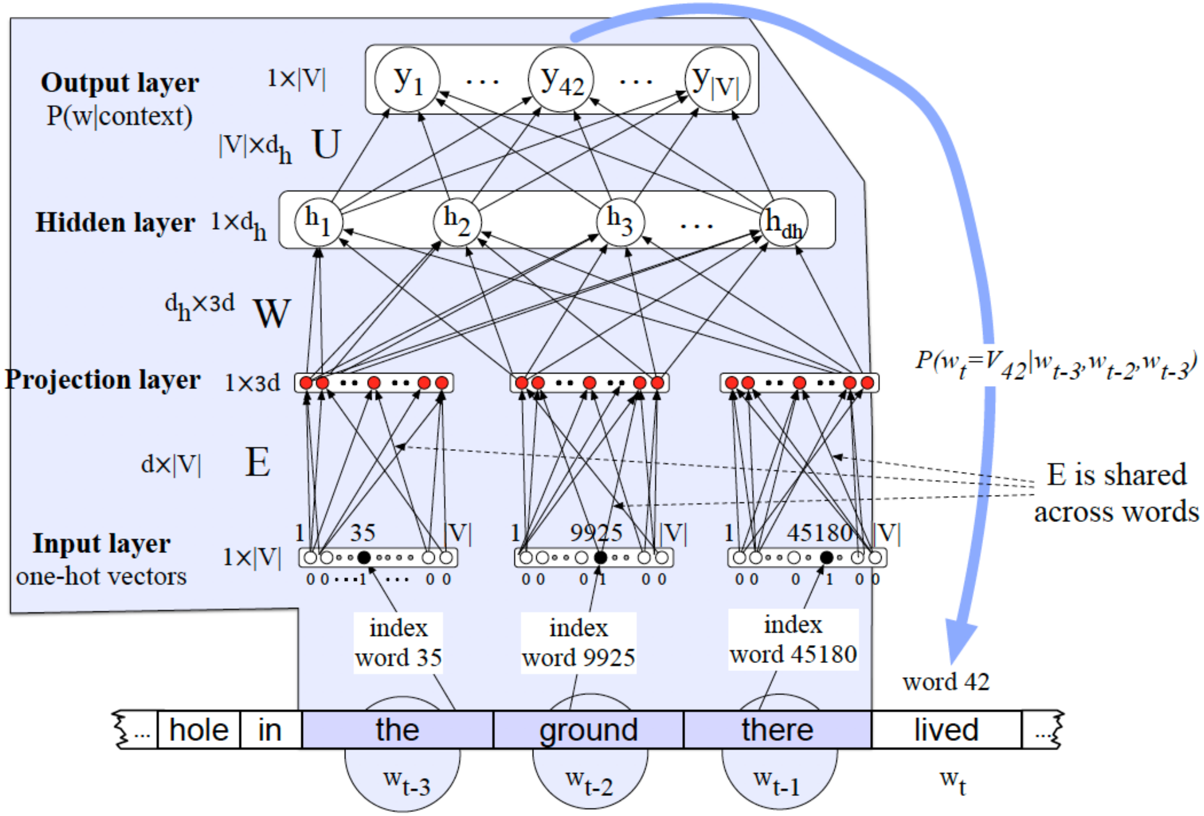
ADVANTAGES

- Generalizes over contexts of similar words (due to embeddings)
- Doesn’t need smoothing
- Can handle longer histories

DISADVANTAGES

- Slower to train than N-gram model
- Still based on Markov assumptions
  - probability of word given the entire context is approximated based on the N previous words

Where do the embeddings come from?



- Embedding weight matrix E learnt at the same time as W and U
  - 3 layer network
  - random initialisation
- Pre-trained embeddings (e.g., LSA)
  - 2 layer network (or 3 layer network with **frozen** embedding matrix)
  - or **tuned** in a 3 layer network

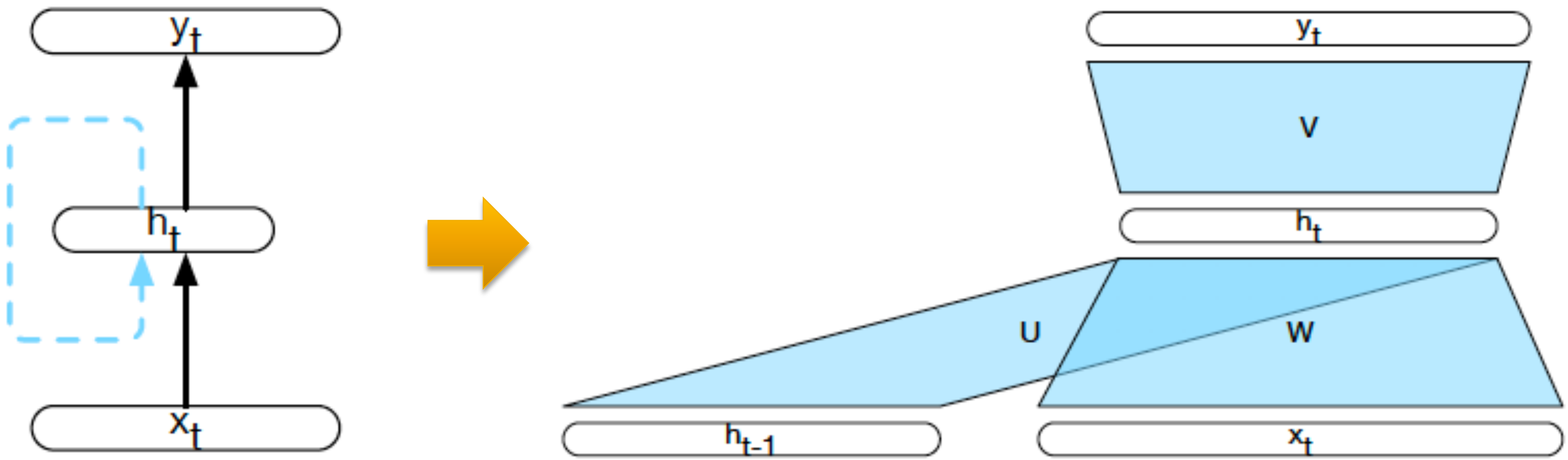
Log-bilinear language model

- Variant on FF-NLM
- No non-linearity in the hidden layer
- Faster to train
- See Mnih and Hinton (2008), Botha and Blunsom (2014)

RNNs

Recurrent neural networks

Simple Recurrent Networks



- activation value of the hidden layer depends on
  - the current input; and
  - the activation value of the hidden layer from the previous step

RNN

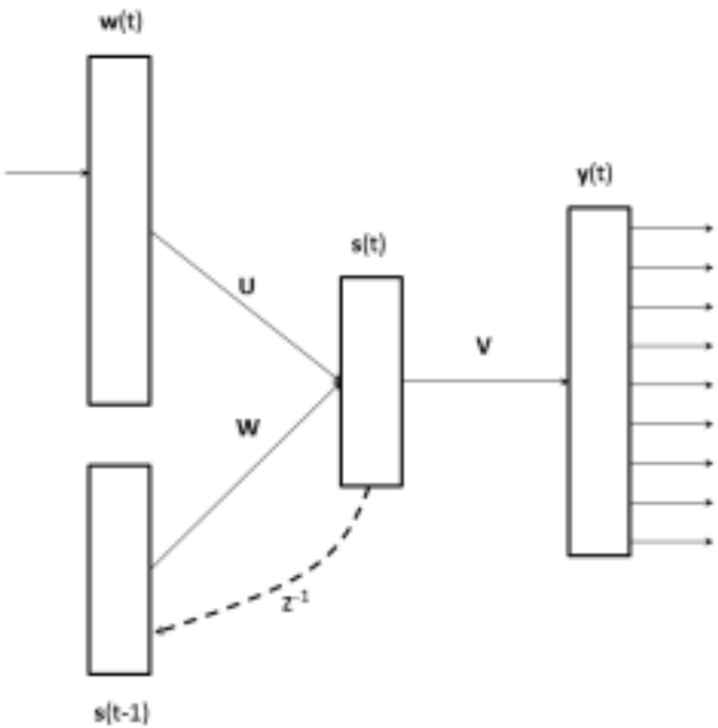
ADVANTAGES

- Hidden layer from previous timestep provides memory
- No fixed length limit on amount of prior context
  - The weights U determine how the network should use past context in calculating output for current input

DISADVANTAGES

- Whilst RNN theoretically has access to the entire preceding sequence, in practice, information tends to be fairly local
- Difficult to capture long-range semantic dependencies
  - Hidden layer weights need to capture information for short range dependencies (e.g., pluralisation agreement) and long range dependencies

Recurrent Neural Network Language Model (Mikolov et al. NAACL 2013)



- Input and output vectors (w and y) have dimensionality of the vocabulary
- Output vector y is a probability distribution over words
- Word representations are found in the columns of a matrix U (selected by the 1 of N encoding of the current word w(t)
- The hidden layer (s(t)) maintains a representation of sentence history
- Trained with back-propagation to maximise data log-likelihood

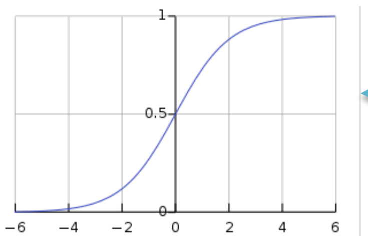
Computation of hidden and output layers

$$\mathbf{s}(t) = f\left(\overbrace{\mathbf{U}\mathbf{w}(t)}^{\text{embedding for current word}} + \overbrace{\mathbf{W}\mathbf{s}(t-1)}^{\text{representation of previous state}}\right) \tag{1}$$

$$\mathbf{y}(t) = \underbrace{g\left(\mathbf{V}\mathbf{s}(t)\right)}_{\text{probability distribution over words}}, \tag{2}$$

where

$$f(z) = \frac{1}{1+e^{-z}}, \quad g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}. \tag{3}$$



sigmoid function

softmax function ensures output is a probability function

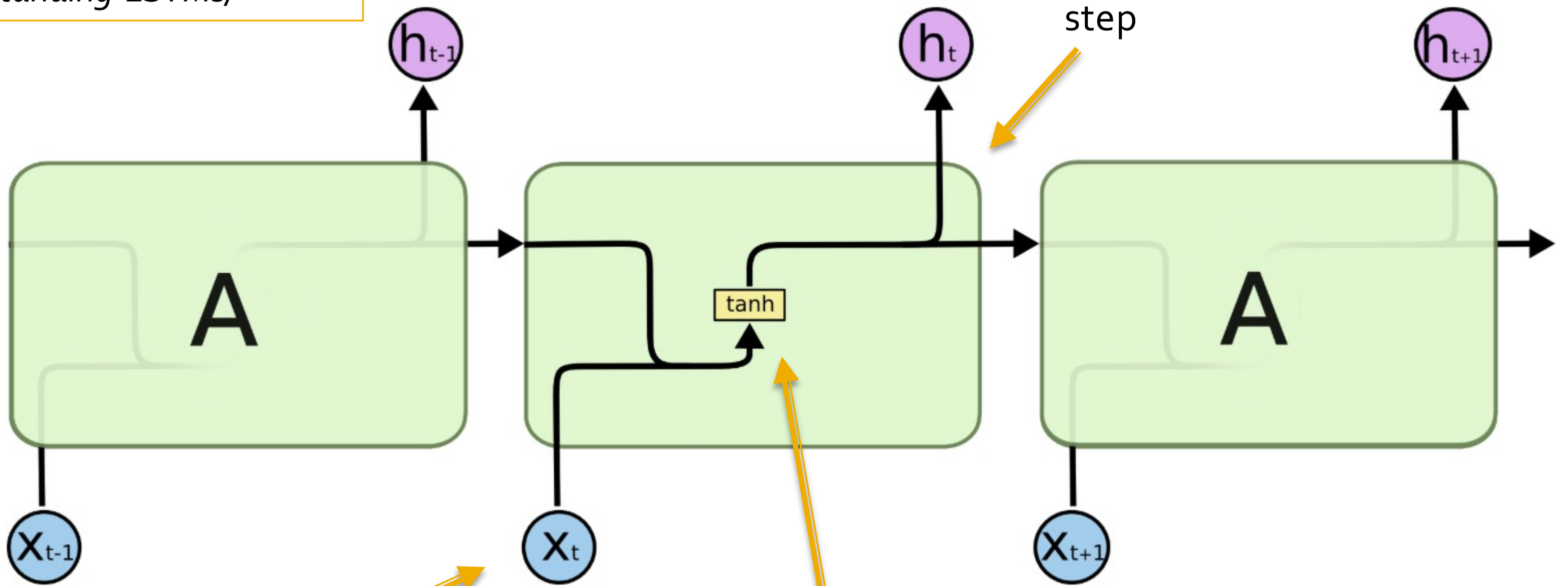
LSTMs

Long short-term memory networks

Unrolling a simple RNN

Diagrams from Colah's blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

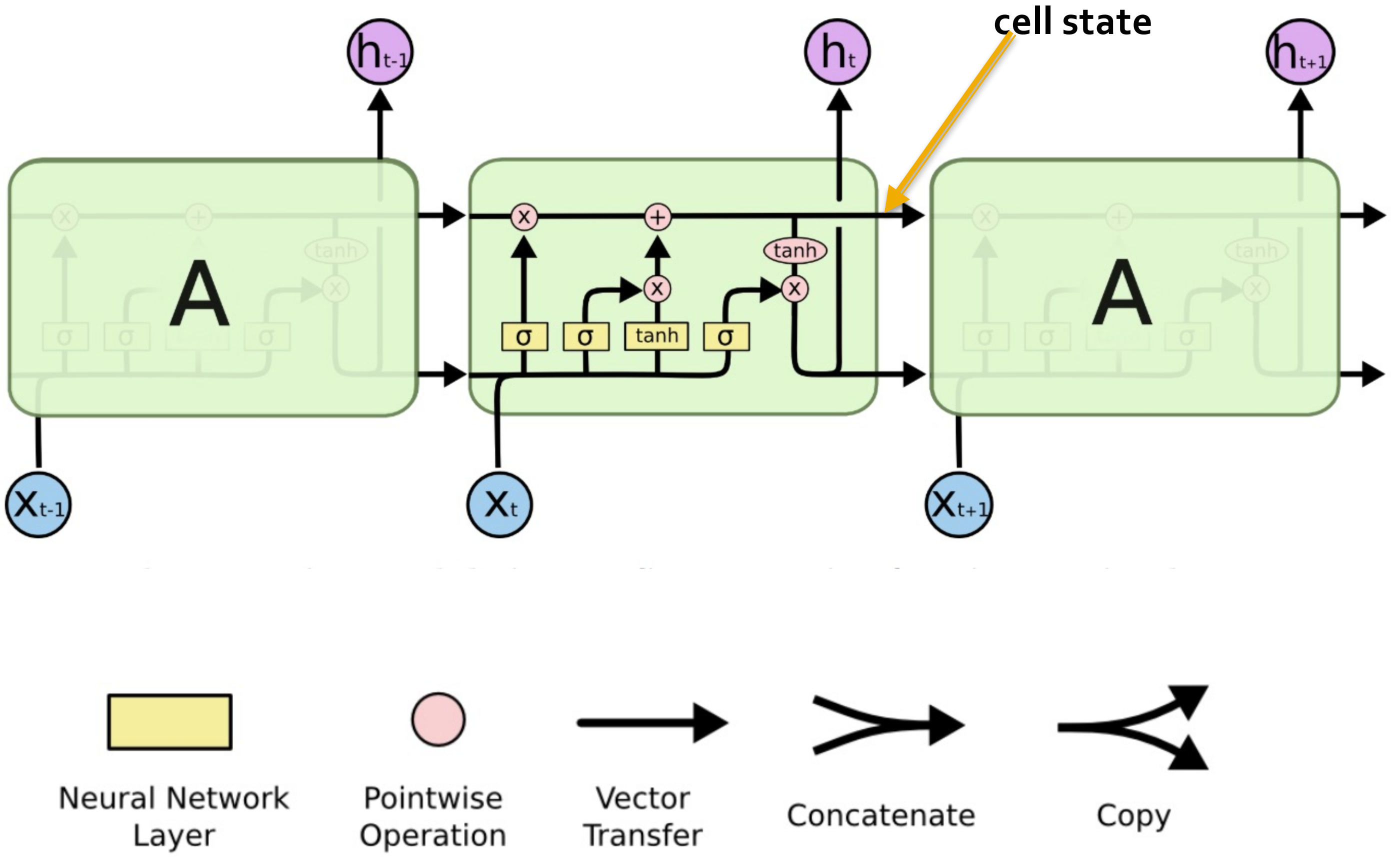
activation of hidden layer used for current output and input to next time step



current input

hidden layer units with tanh activation function

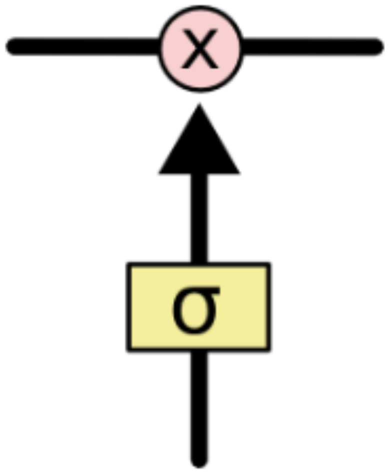
Unrolling an LSTM



Core idea is that at each time step, hidden layer activation values depend on

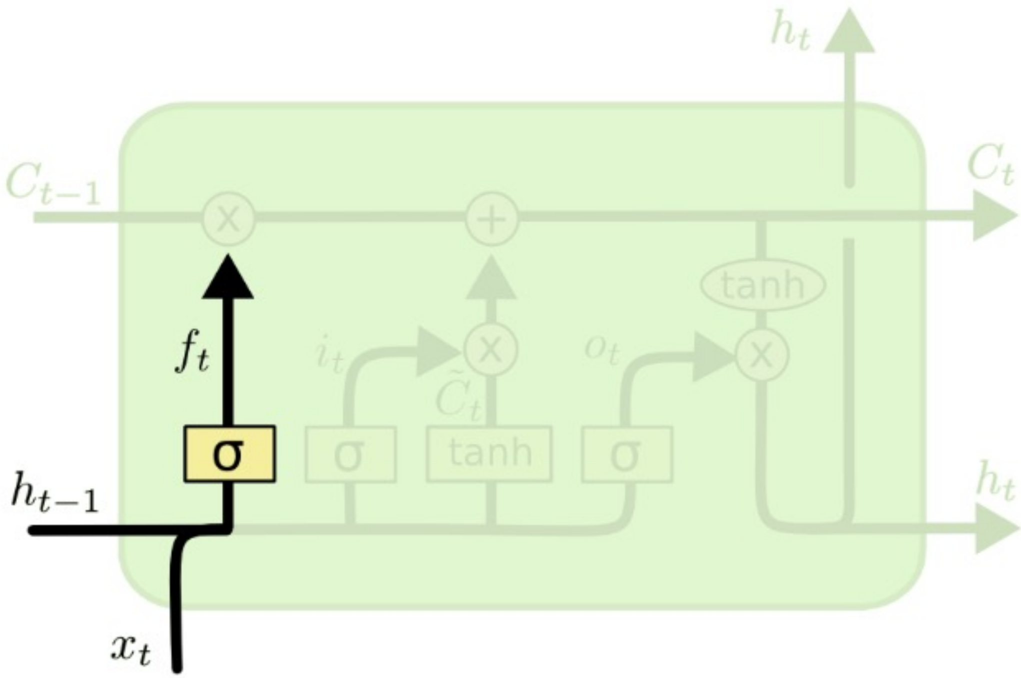
- current input
- activation values from the hidden layer at previous time step (*short term memory*)
- activation values from the **cell state** (*long term memory*)

Gates inside neural units



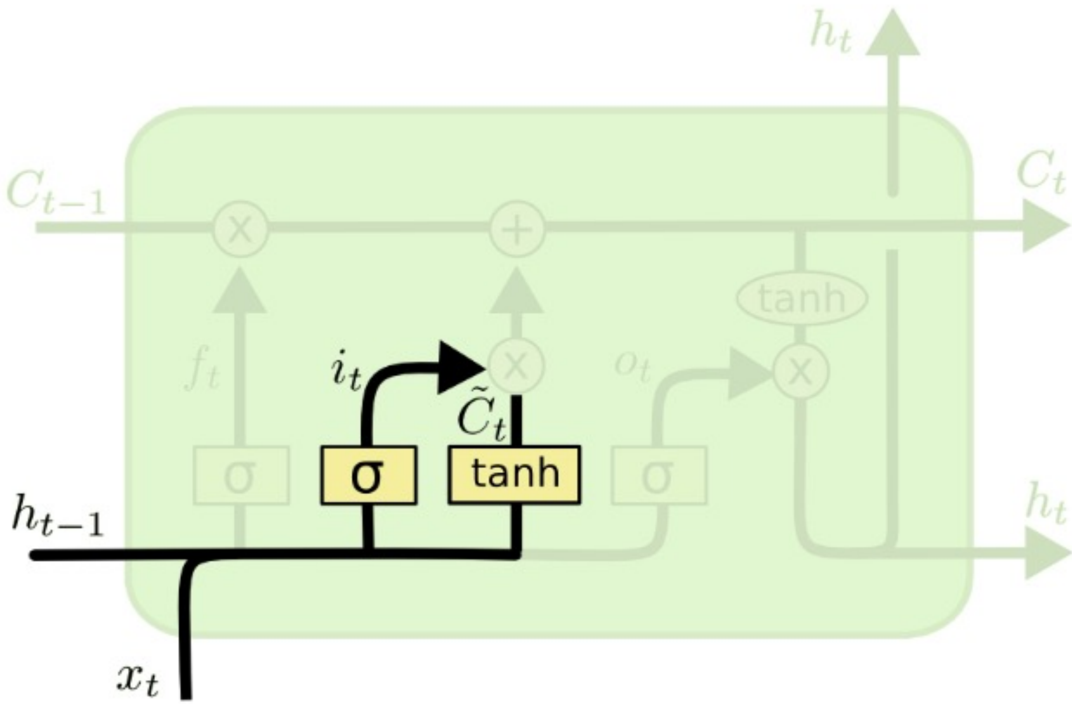
- a way to control flow of information
  - optionally allow information through
- composed of
  - a sigmoid neural net layer
  - a pointwise multiplication
- Sigmoid outputs numbers between 0 and 1, describing how much of each component should be let through
  - it controls the gate

Forget



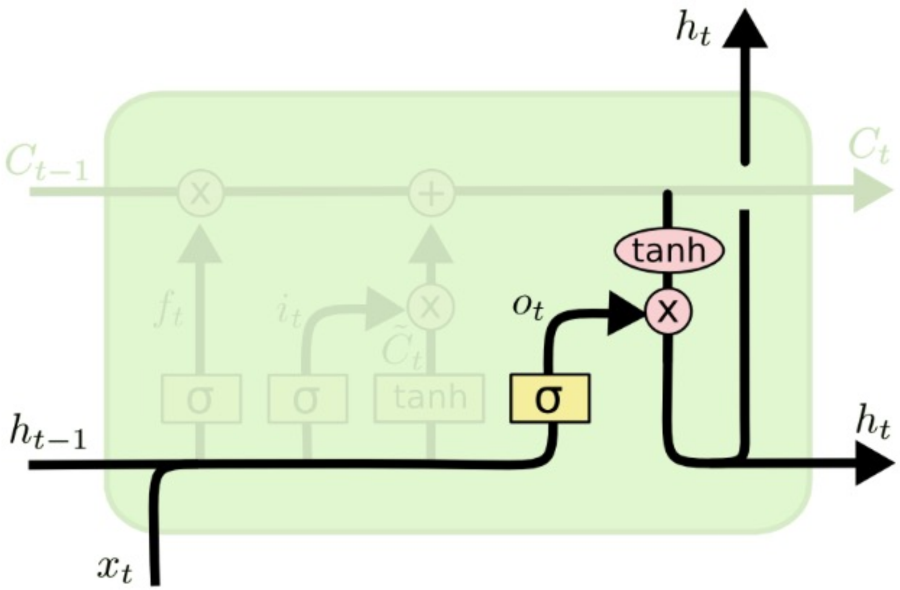
- What in the current cell state should be forgotten?
- Controlled by the current input and the short term memory

Input



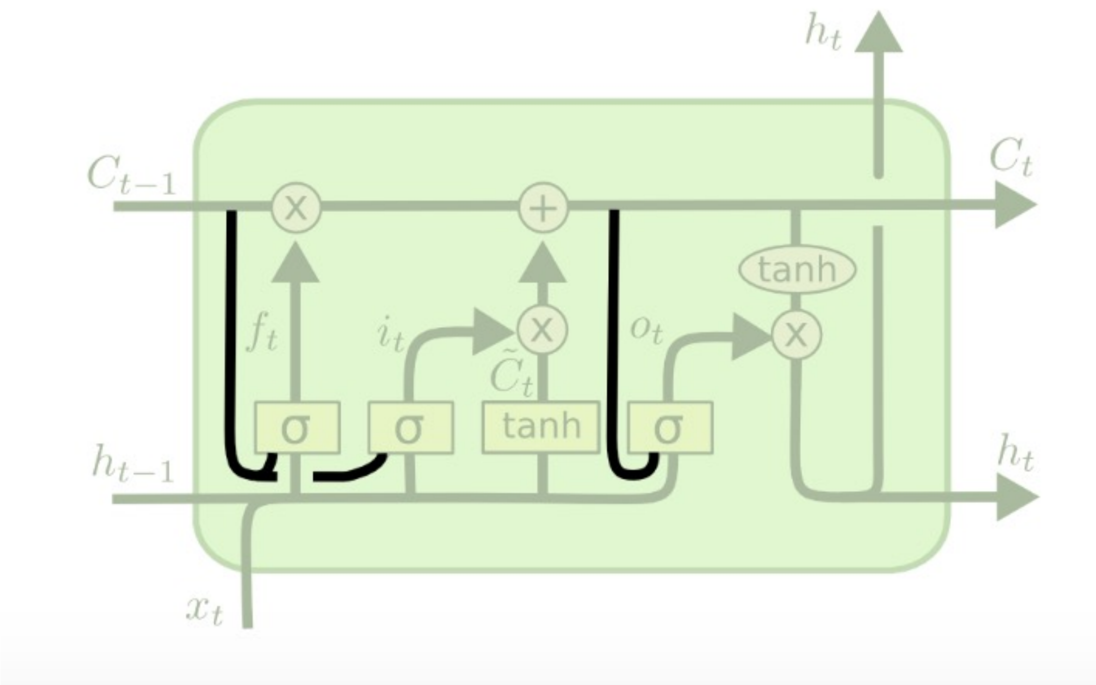
- What of the current input and short-term memory should be put into the long-term memory?
  - sigmoid decides which values to update
  - tanh decides what those values should be

Output



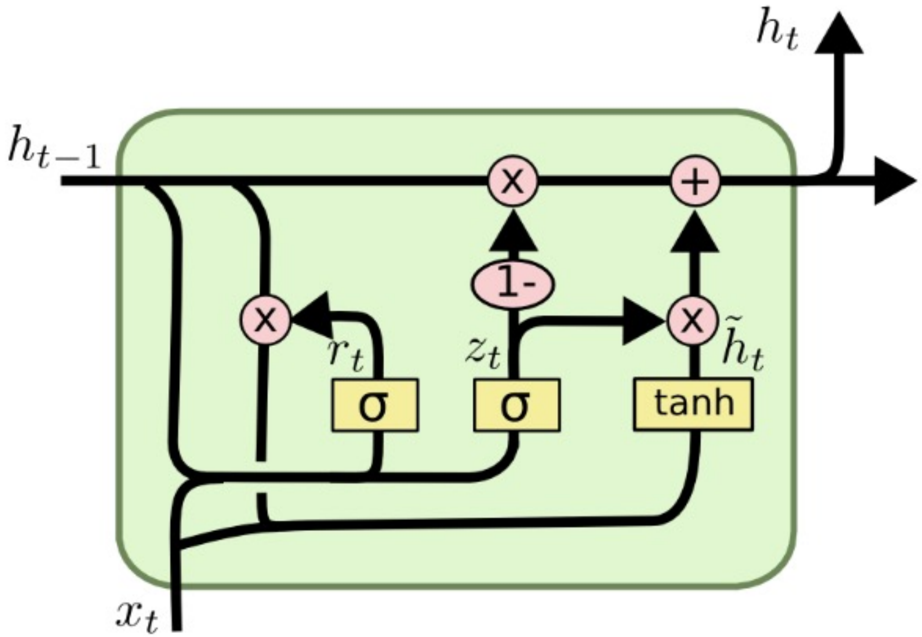
What should be **output** at this timestep?

Peephole variant



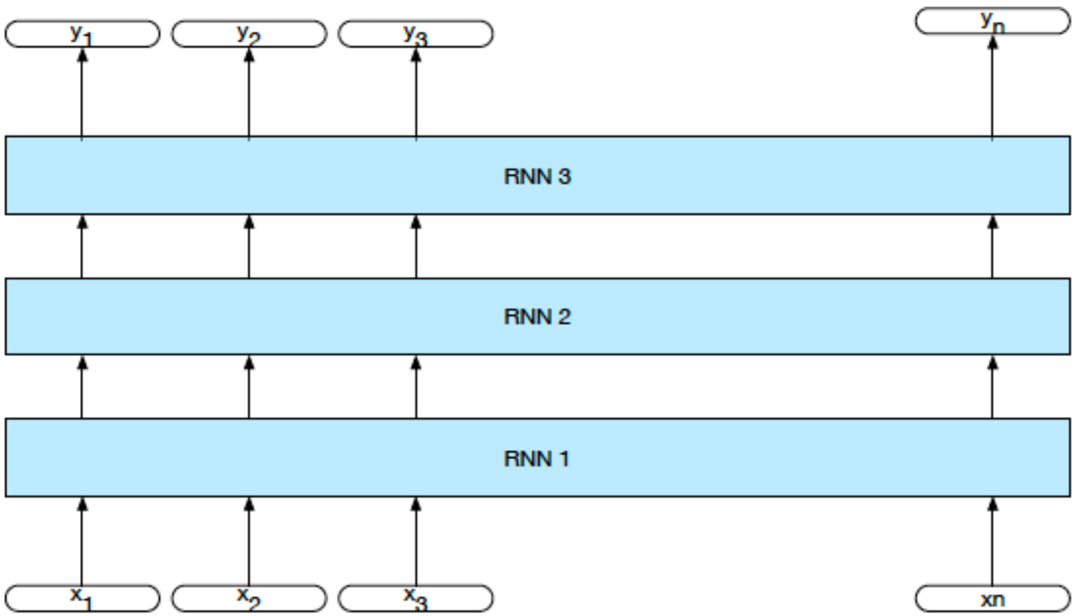
Gates also have access to the cell state when deciding what to let through

Gated recurrent unit (GRU) variant



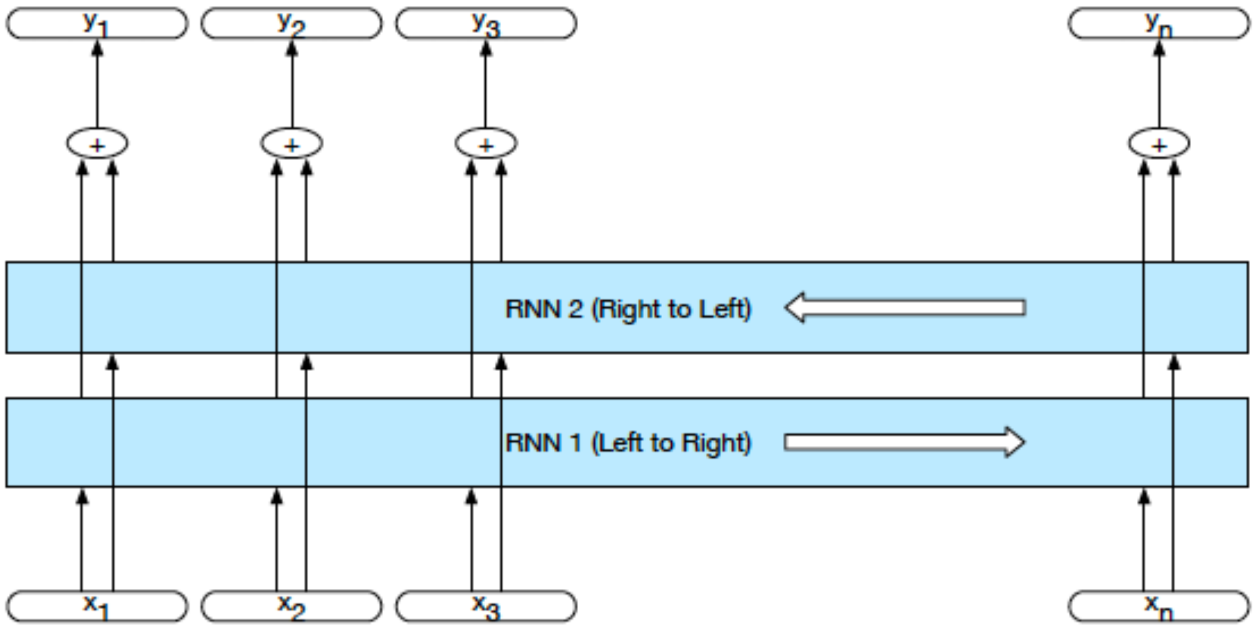
- Simpler alternative
  - Forget and input gates merged into a single update gate
  - hidden and cell state merged

Stacked RNNs



- Output (at each time step) from RNN is fed as input into another RNN
- Stacking leads to **deep** networks and **deep** learning
- Different layers induce representations at different levels of abstraction
  - short range syntactic dependencies in early layers
  - long range semantic dependencies in deeper layers

Bidirectional RNNs



- Take advantage of right context as well as left context
- Pair of RNNs
  - one is trained from left-to-right
  - the other is trained from right-to-left
- Add or concatenate hidden layers from both RNNs to produce output at each timestep

Beyond stacked bidirectional LSTMs

- Attention
  - at each step allow the RNN to pick information from larger collection of information
- Encoder-decoder networks
- Transformers
- Grid LSTMs

Characters and Convolutions

Part 2 of lecture

Problems for word-based NLMs

- Sparsity!
- Can generalise using similar words but ….
- How reliable are embeddings for low frequency / unknown words?
- Should we replace all low-frequency words with "UNK" token?
- Can we do better?

Character-based NLMs

- So far core input unit has been the word
  - sparsity problems
- What if we make the core input unit a character?
  - no sparsity problems
  - can we learn to make language predictions based on character embeddings?

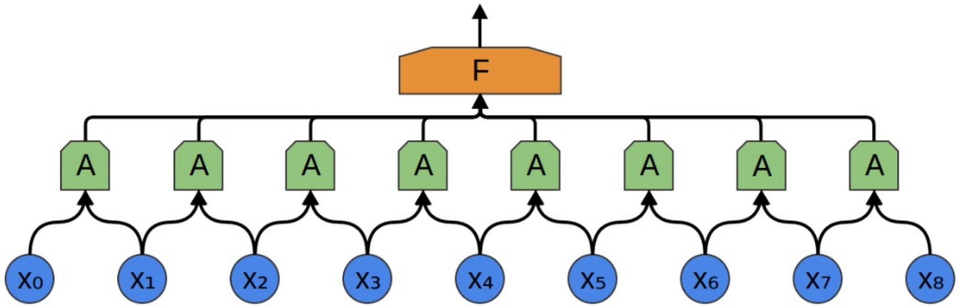
CNNs

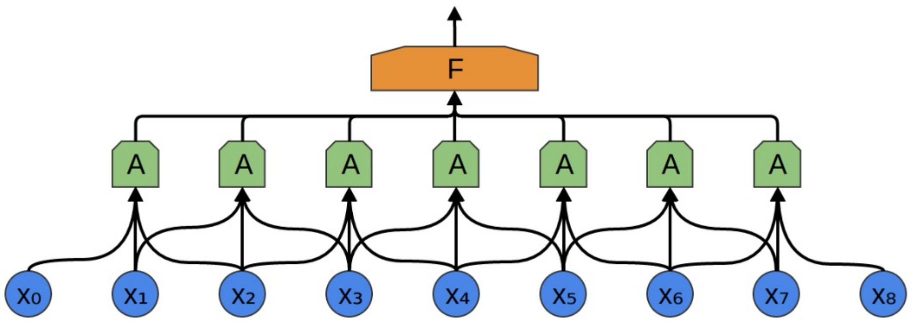
Convolutional neural networks

CNNs

- Very common in vision
  - convolutional layers used to detect features in an image
  - e.g. an edge
  - features can be anywhere in the image
  - but presence determined by looking at neighbouring pixels / inputs
- Can also be used on text

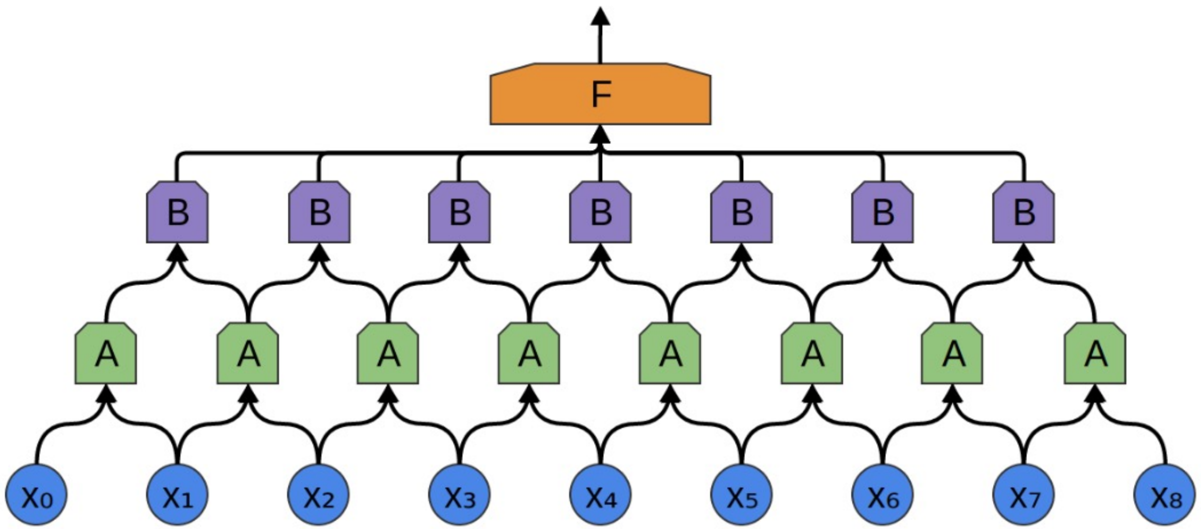
Convolutions





- Convolutional layers look at groups of inputs or neurons, e.g.,
  - convolution with window size 2
  - convolution with window size 3
- **kernel** function A is learnt which looks for or filters for a particular pattern / feature in the input e.g.,
  - 2 character sequence “un”
  - 3 character sequence “ing”

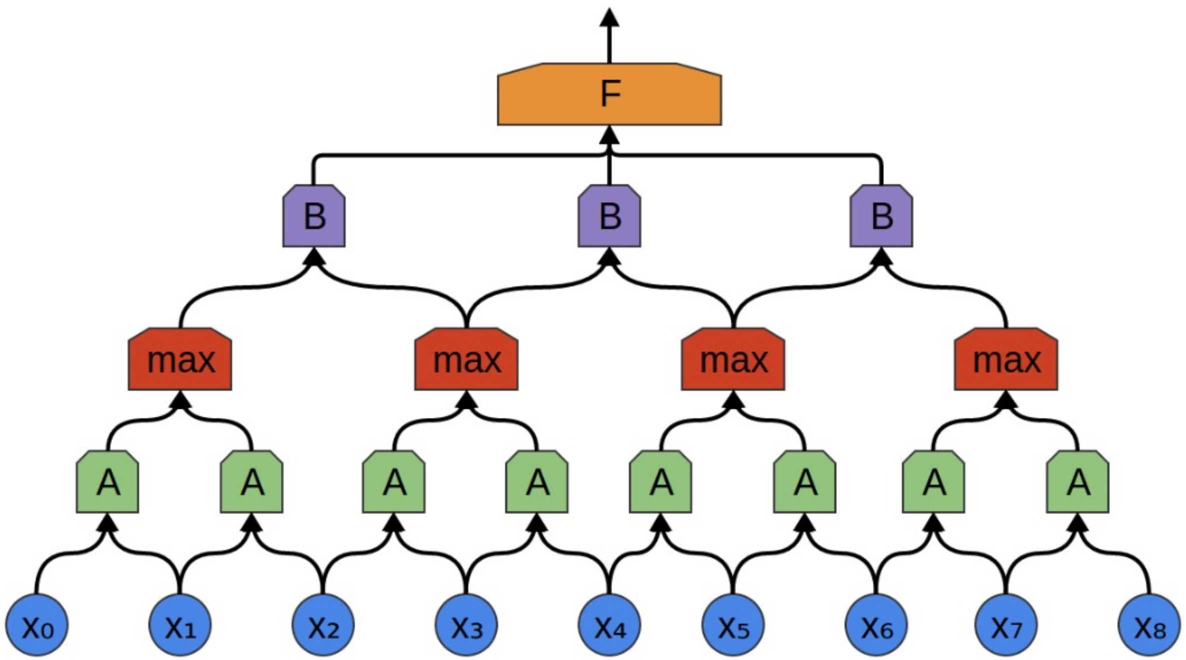
Stacking convolutional layers



Stacking allows detection of complex features composed from simpler features e.g.,

- face detection requires eye detection

Max pooling



Max-pooling layer takes the maximum of its inputs

- is a feature present anywhere in this chunk of input?
- does not care where it is in the input

Kernel Functions

- Individual kernel function detects an individual feature in the input
- In practice hundreds or thousands of kernel functions needed for different character combinations
- Kernel functions are just neural network weights
  - learnt by training on a corpus
  - kernels learnt depend on the corpus / task

Advantages of character-aware NLMs

- allows morphological information to be utilised
- particular useful for low-frequency words

Using NLMs

- What can we do with a NLM beyond predicting the next word?
- Calculate the probability of a word sequence
  - speech recognition
  - machine translation
  - spelling correction
- Use as input to text classification task
  - relevancy or sentiment classification
  - paraphrase identification
- Use in sequence labelling tasks
  - part of speech tagging
  - named entity recognition

Next time:

- Sequence labelling (Named Entity Recognition)

References

- Bengio, Y. et al. 2003. A neural probabilistic language model. In *Journal of Machine Learning Research*.
- Botha, J., and Blunsom, P. 2014. Compositional morphology for word representations and language modelling. In *Proceedings of ICML*
- Kim, Y. et al. 2016. Character-aware neural language models. In *Proceedings of the AAAI*.
- Mikolov, T. et al. 2010. Recurrent neural network based language model. In *Proceedings of Interspeech*
- Mnih and Hinton, G. 2008. A scalable hierarchical distributed language model. In *Proceedings of NIPS*.