NEURAL MACHINE TRANSLATION (PART 1)

David Talbot

23rd April, 2017

Computer Science Club, St. Petersburg, Russia

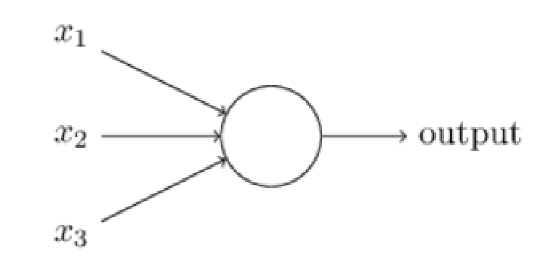
this morning's topics

- Neural Networks (NN)
- Convolutional NN for NLP
- Recurrent Neural Networks and extensions (LSTM, GRU)
- First NMT: Recurrent Continuous Translation Model
- Encoder-Decoders (sequence-to-sequence) models for MT

perceptrons (mcculloch & pitts, 1943)

Given input $x = (x_1, x_2, \ldots, x_m)$ where $x_i \in \{0, 1\}$

$$f(x) = \begin{cases} 1 & \text{if} \quad w \cdot x + b > 0 \\ 0 & \qquad\qquad \text{otherwise} \end{cases}$$



learning

Given training data

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

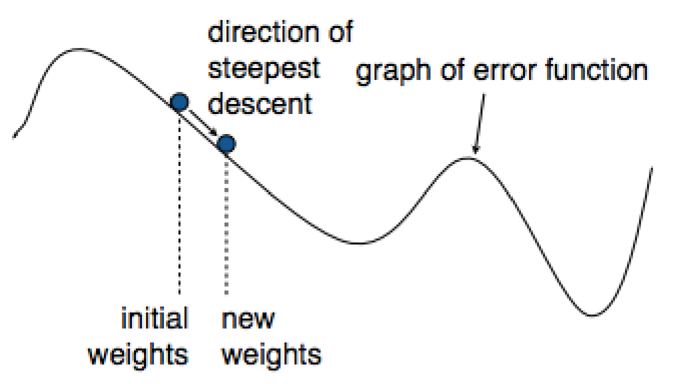
Measure the error on D using a cost-function, e.g.

$$C(w) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Minimize the error by updating w such that

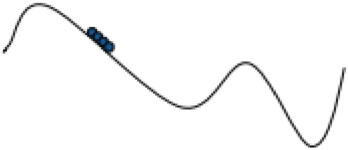
$$w \leftarrow w - \alpha \nabla C(w)$$

gradient descent

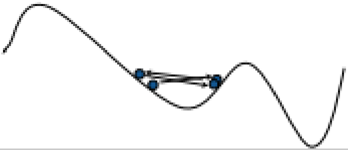


learning rate

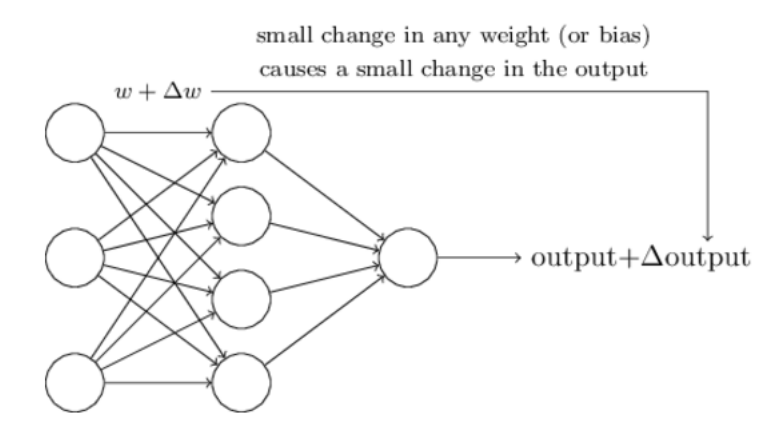
- α small: takes a long time to reach minimum of error function



- α large: may oscillate around minimum, without converging

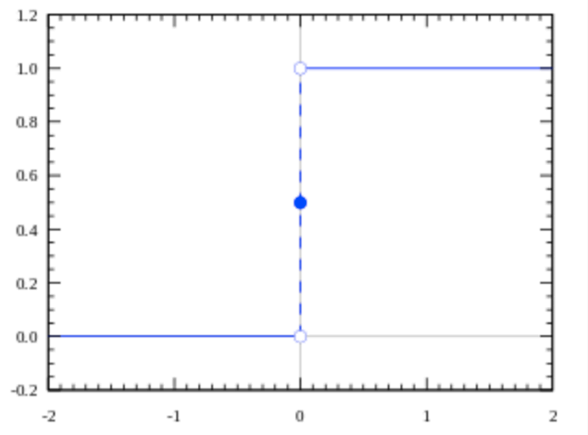


continuous



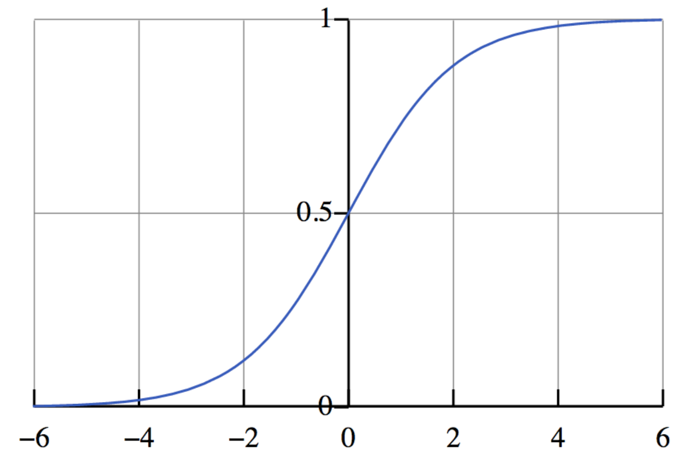
small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

perceptron activation function



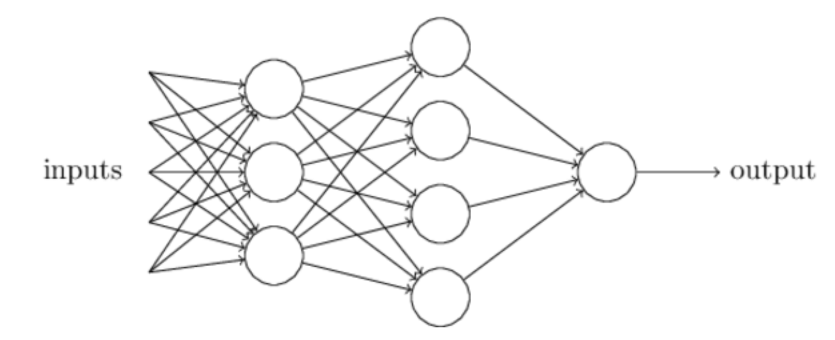
$$f(x) = \begin{cases} 1 & \text{if} \quad w \cdot x + b > 0 \\ 0 & \qquad\qquad \text{otherwise} \end{cases}$$

sigmoid



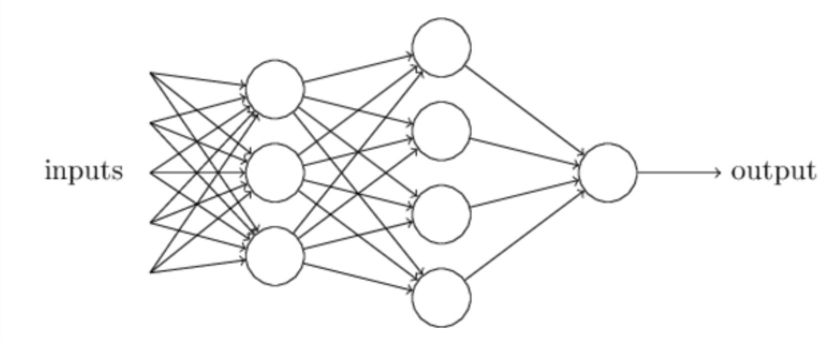
$$f(x) = \frac{1}{1 - \exp^{-\{w \cdot x + b\}}}$$

multilayer perceptron



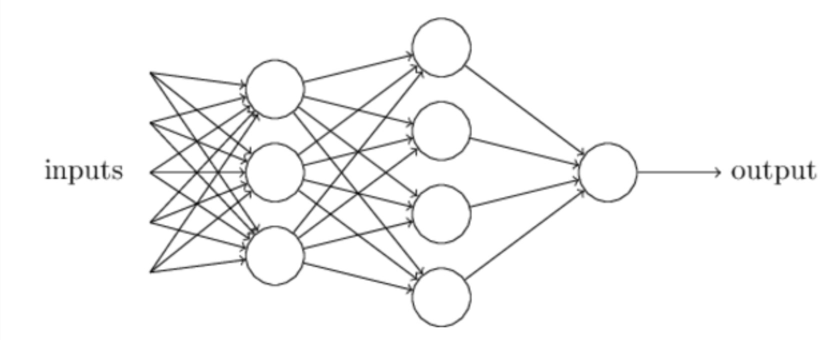
$$g(x) = f^3\left(\sum_{k=1}^{4} W^3_{1,k} o_k + b\right)$$

multilayer perceptron



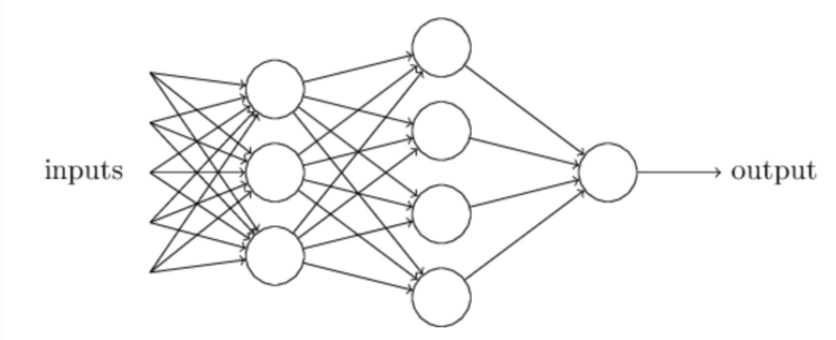
$$g(x) = f^3(\sum_{k=1}^{4} W^3_{1,k} f^2(\sum_{j=1}^{3} W^2_{k,j} o_j + b_k) + b)$$

multilayer perceptron



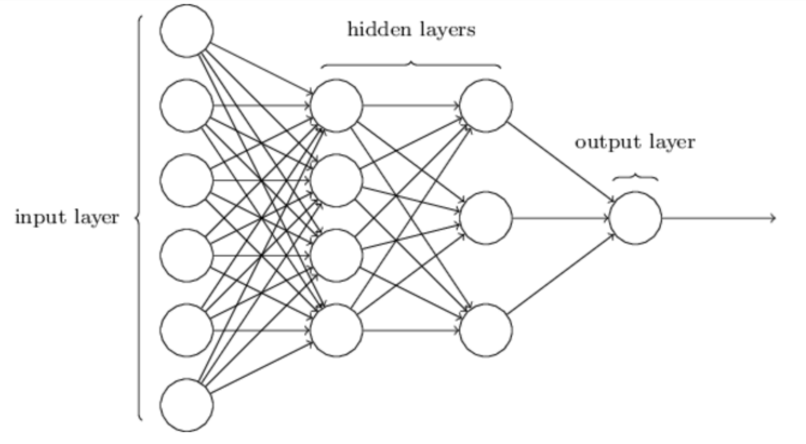
$$g(x) = f^3\left(\sum_{k=1}^{4} W^3_{1,k} f^2\left(\sum_{j=1}^{3} W^2_{k,j} f^1\left(\sum_{i=1}^{5} (W^1_{j,i} x_i + b_j)\right) + b_k\right) + b\right)$$

multilayer perceptron



What does this buy us if activations $f^i(\cdot)$ are linear?

deep network



stochastic gradient descent: cost functions

- Compute gradient on 'mini-batches' of the training data

$$\nabla C = \frac{\sum_{i=1}^{n} \nabla C_{X_i}}{n} \approx \frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m}$$

- What assumptions do we need on our cost functions?

stochastic gradient descent: cost functions

- Loss expressed as a function of the output layer
- Loss expressed as an average over data points

$$C_{mse} \equiv \frac{1}{2n} \sum_{i=1}^{n} \|y(x_i) - \hat{y}(x_i)\|^2$$

or

$$C_{cross\_entropy} \equiv -\frac{1}{n} \sum_{i=1}^{n} \sum_{y'} \mathbf{Pr}(y(x_i) = y') \log \mathbf{Pr}(\hat{y}(x_i) = y')$$

where $y(x)$ and $\hat{y}(x)$ are the true and predicted labels.

backpropagation

Compute derivatives for all parameters:

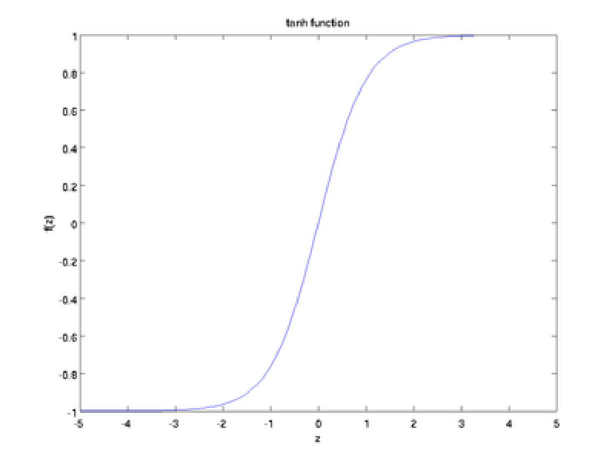
$$\frac{\partial C}{\partial w_{jk}^{l}}, \frac{\partial C}{\partial b_{j}^{l}} \quad \forall j, k, l$$

so that we can update the model to reduce the cost.

Recursion based on chain-rule: if f and g are both differentiable and $h(x) = f(g(x))$ then

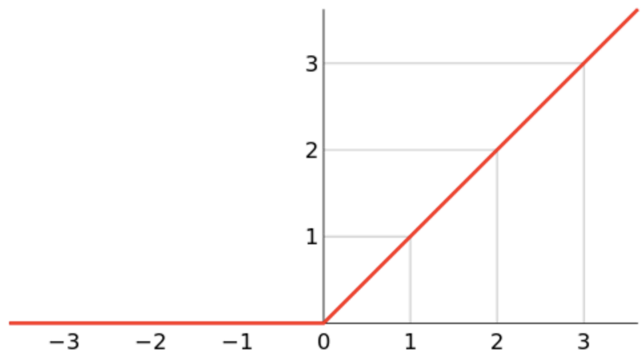
$$h'(x) = f'(g(x)) \cdot g'(x).$$

hyperbolic tangent



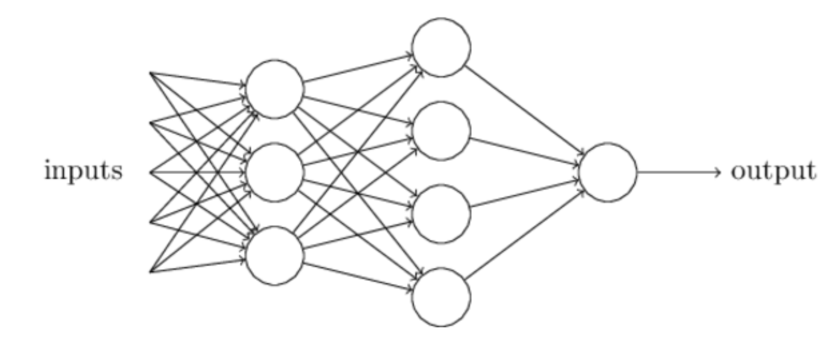
$$f(x) = tanh(wx + b)$$

rectified linear unit (relu)



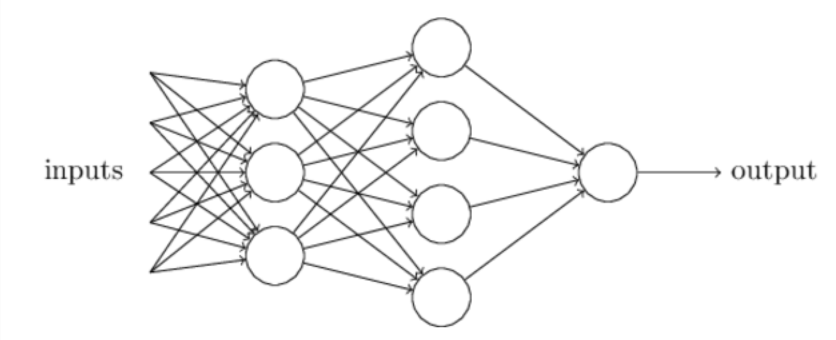
$$f(x) = max(0, wx + b)$$

hidden units



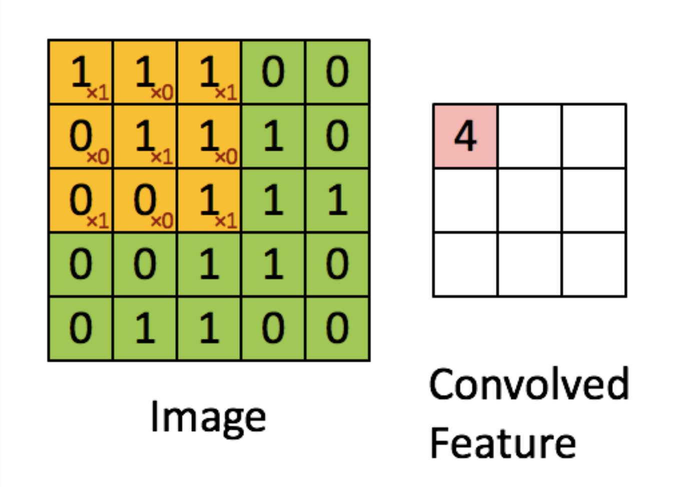
A neural network with one hidden layer can approximate an arbitrary functions (with enough hidden units)

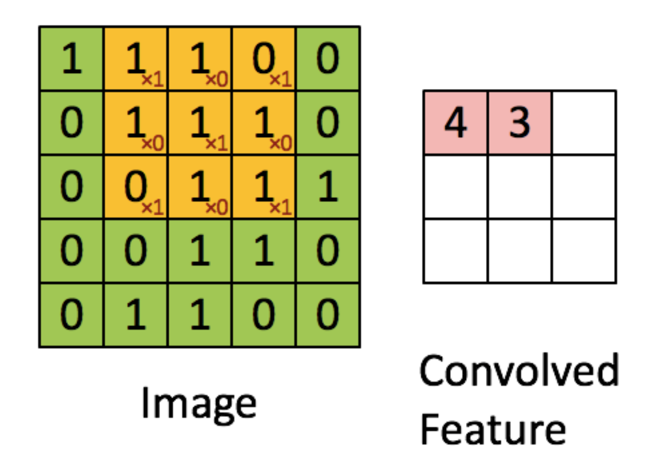
fully connected networks

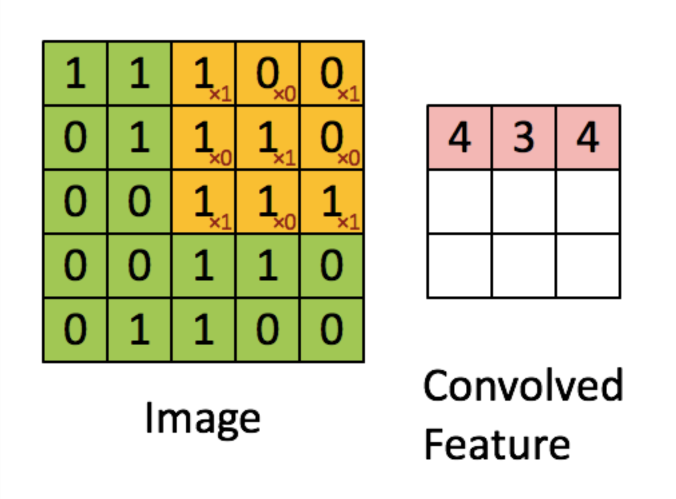


How about the inductive bias?

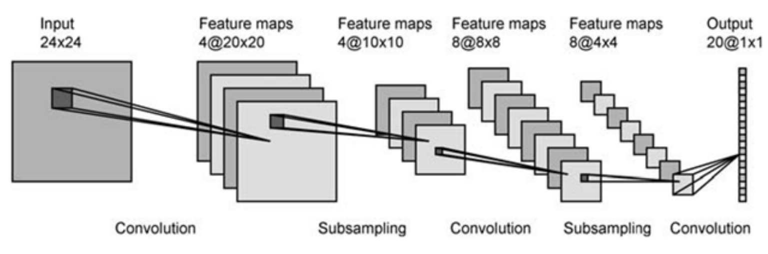
convolutional network




1
1 ×1
1 ×0
0 ×1
0
0
1 ×0
1 ×1
1 ×0
0
0
0 ×1
1 ×0
1 ×1
1
0
0
1
1
0
0
1
1
0
0
Image
4
3
Convolved Feature


1
1
1 ×1
0 ×0
0 ×1
0
1
1 ×0
1 ×1
0 ×0
0
0
1 ×1
1 ×0
1 ×1
0
0
1
1
0
0
1
1
0
0
4
3
4
Image
Convolved
Feature

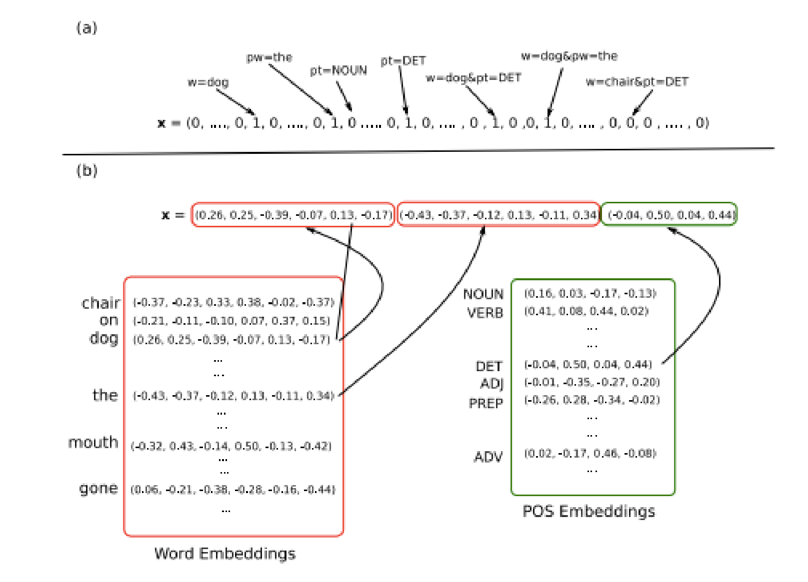
convolutional network

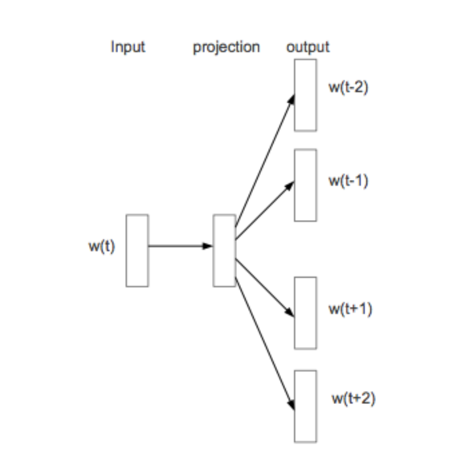


word embeddings: sparse vs dense representations

- Sparse: Each feature one dimension (binary value), each combination has its own dimension
- Dense: Each feature has a vector, no explicit encoding of feature combinations

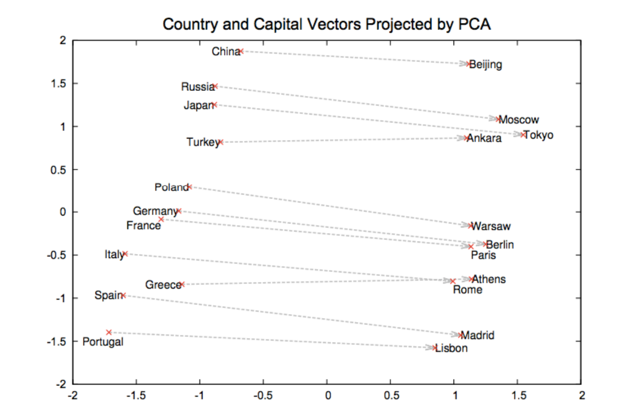
word embeddings: sparse vs dense representations





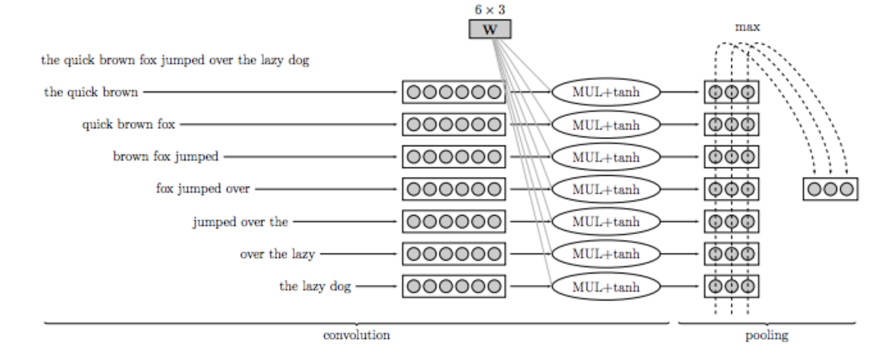
Skip gram model: predict word in random position close to w_t

word embeddings



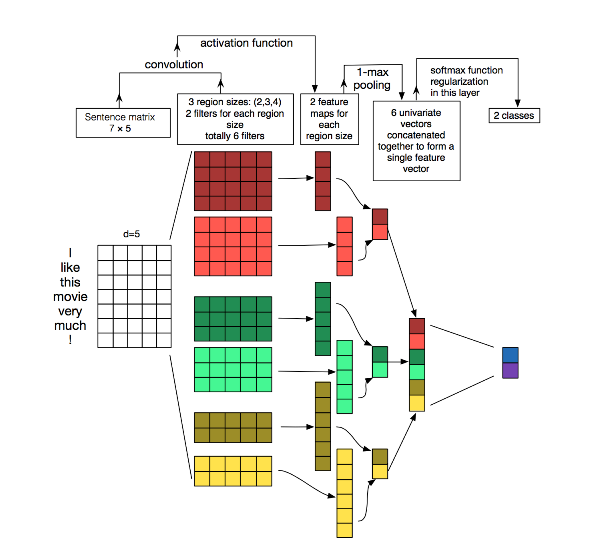
Magic of word embeddings? (More later)

convolutional network



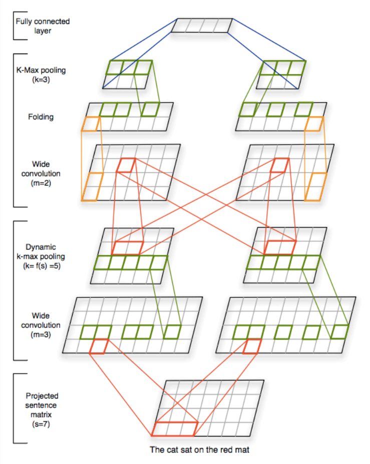
(Source: Goldberg, 2015)

convolutional network



Source: Zhang, Y., & Wallace, B. (2015)

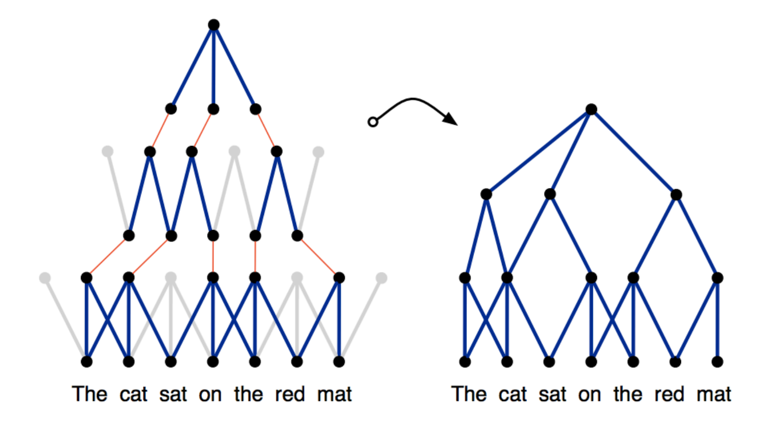
convolutional sentence model



Source: Kalchbrenner et al. (2015)

convolutional sentence model

Encoder in first NMT approach (Kalchbrenner & Blunsom 2013)



Source: Kalchbrenner et al. (2015)

neural probabilistic language model (bengio et al. 2003)

Given training sequences of words $w_1, \ldots, w_T$ where $w_t \in V$, we want to learn a function

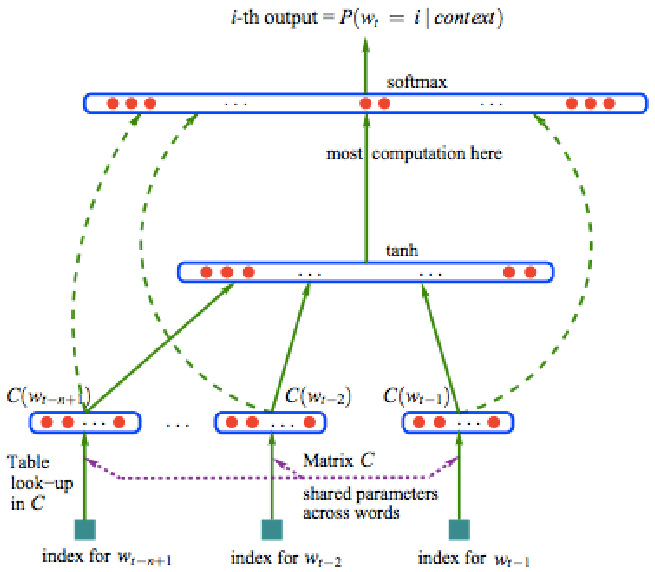
$$f(w_t, \ldots, w_{t-n+1}) = \Pr(w_t | w_1^{t-1})$$

Bengio et al., 2003 decomposes $f(\cdot)$ into

1. A mapping C from any element i of V to a real vector $C(i) \in \mathbb{R}^m$ (a $|V| \times m$ matrix)
2. A function (neural network) that assigns a probability $P(w_t = i | w_1^{t-1})$ as

$$f(i, w_{t1}, \ldots, w_{tn+1}) = g(i, C(w_{t1}), \ldots, C(w_{tn+1}))$$

neural probabilistic language model (bengio et al. 2003)



neural probabilistic language model (bengio et al. 2003)

The output softmax layer is most computational

$$\Pr(w_t | w_1, \ldots, w_{t-1}) = \frac{e^{y_{w_t}}}{\sum_{i \in V} e^{y_i}}$$

where

$$y = b + Wx + U \tanh(d + Hx)$$

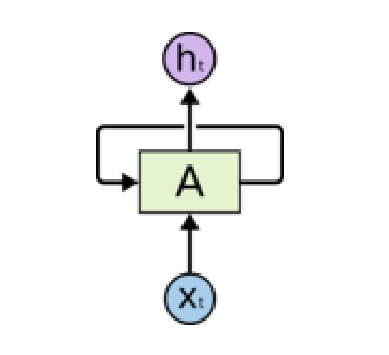
and

$$x = (C(w_{t-1}), \ldots, C(w_{t-n+1}))$$

- W connects inputs to output directly (may be zero)
- U connects hidden layer to output ($|V| \times h$ matrix)
- H connects inputs to hidden layer ($h \times (n-1)m$ matrix)
- b are input biases, d are hidden layer biases

neural probabilistic language model (bengio et al. 2003)

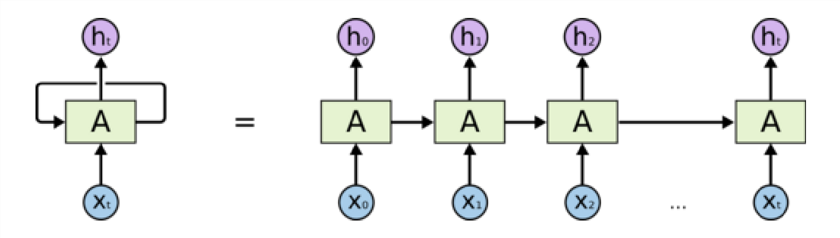
- Number of parameters scales linearly with the vocabulary (unlike *n*-gram models)
- Embedding matrix C is shared among all inputs $x_1, \ldots, x_t$
- Main bottleneck is due to computation of softmax



State A_t updated from current input x_t and previous state A_{t-1}

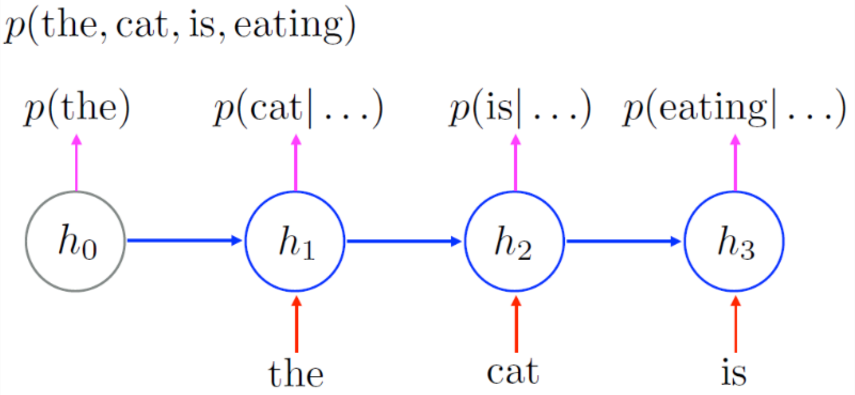
$$A_t = \tanh(U\mathsf{x}_t + W\mathsf{A}_{t-1} + \mathsf{b}) \quad \forall t \geq 1.$$

recurrent neural networks



Parameters shared across time steps

recurrent neural network language models (mikolov 2010)



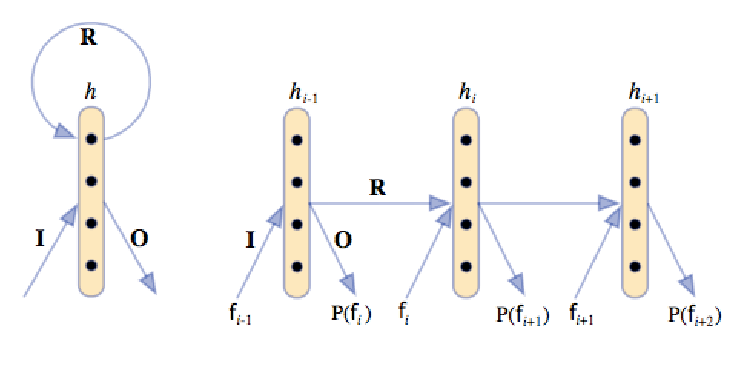
recurrent continuous translation models

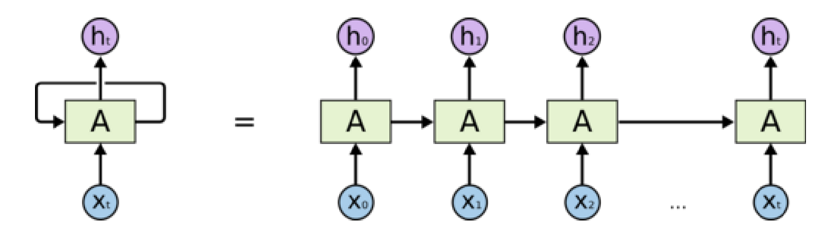
Kalchbrenner & Blunsom, (2013)

- First end-to-end NMT system
- Lower perplexity (average likelihood) than IBM models
- Encode source sentence with convolutional network
- RNN decoder generates target sentence conditioned on source sentence encoded in a ConvNN

$$\mathrm{Pr}(f|e) = \prod_{j=1}^{J} = \mathrm{Pr}(f_j|f_{1:j-1}, e).$$

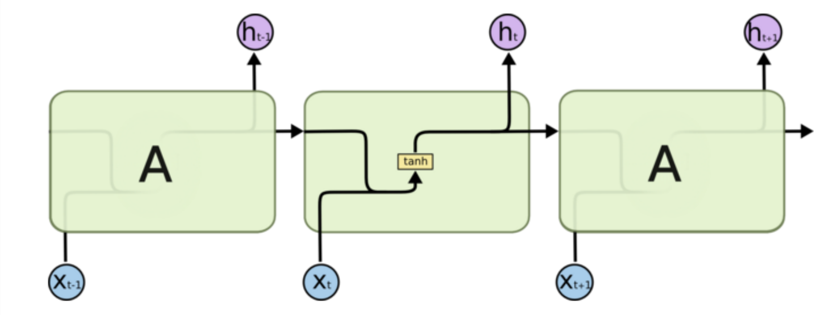
recurrent continuous translation models



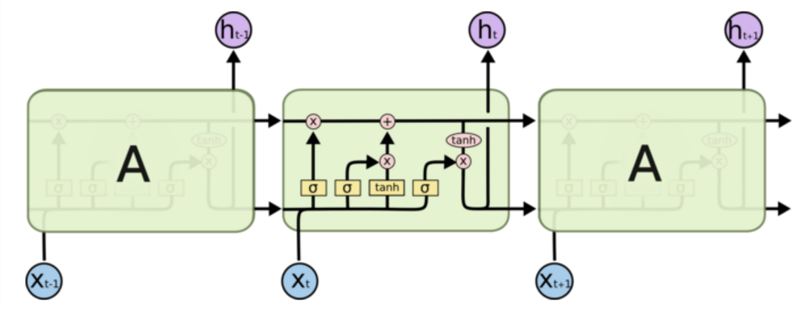


Why might backpropagation fail on such a ’deep’ network?

recurrent neural networks

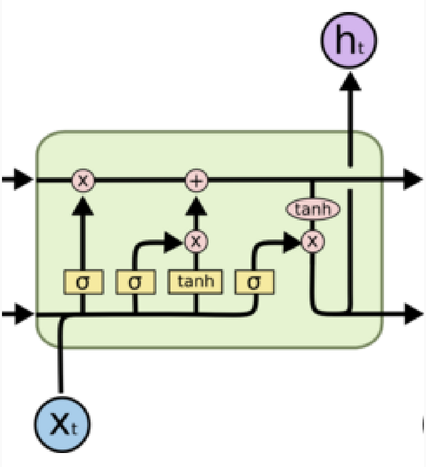


long short term memory units (hochreiter & schmidhuber, 1997)



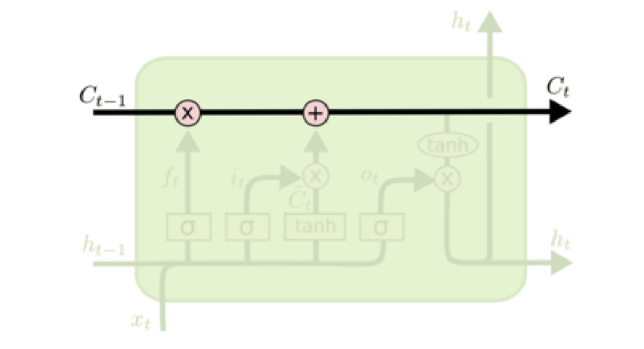
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs

lstm



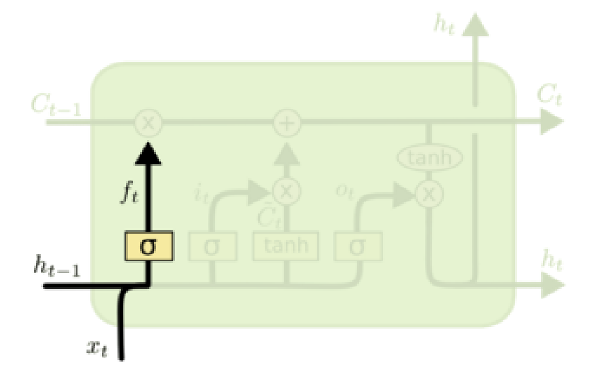
lstm: cell state

Separate cell C_t at each time frame to propagate information



lstm: forget gate activation

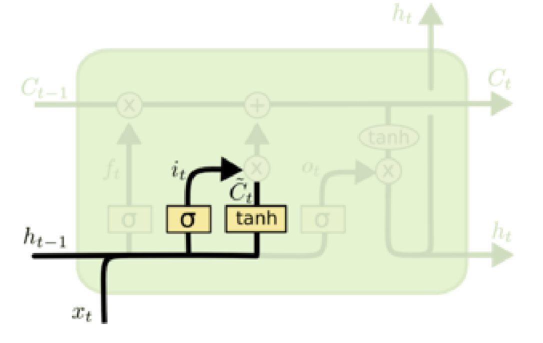
Controls how much each dimension of C_{t-1} is propagated to C_t



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

lstm: new candidate cell state

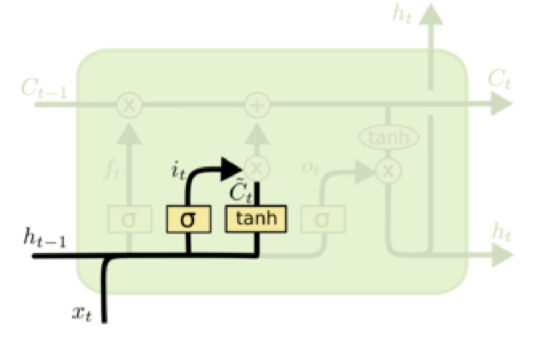
Constructs a preliminary cell state $\tilde{C}_t$



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

lstm: input gate activation

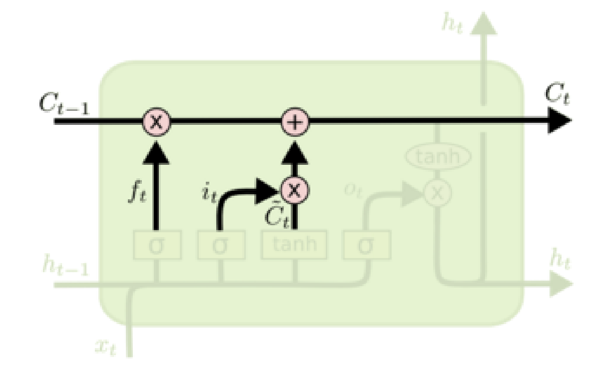
Determines how much each dimension should be updated



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

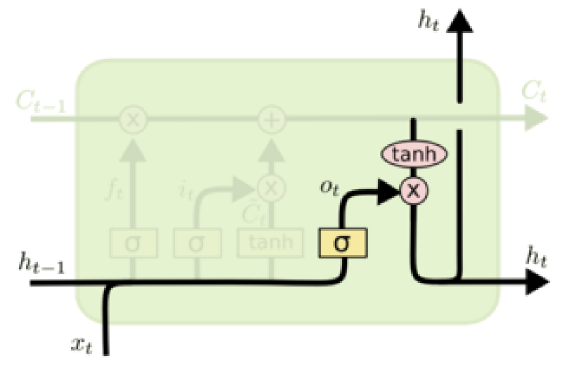
lstm: updating the cell state

Forgetting some of C_{t-1} and overwriting with some of $\tilde{C}_t$



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

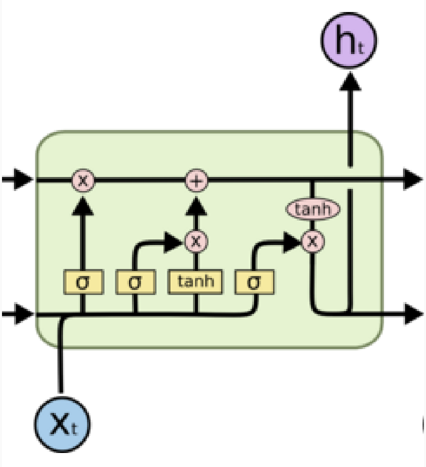
lstm: output gate



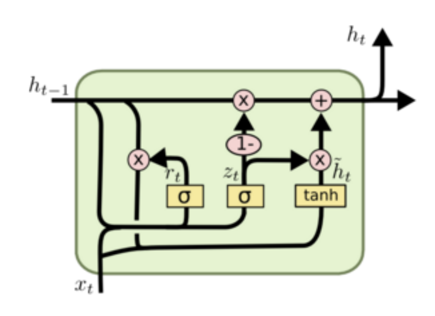
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot tanh(C_t)$$

lstm



gated recursive units (cho et al. 2014)



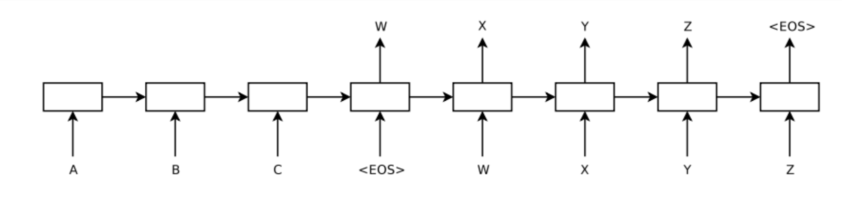
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = tanh(W \cdot [r_t \odot h_{t-1}, x_t])$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}t$$

sequence to sequence learning (sutskever et al. 2014)



Deep LSTM encoder-decoder

sequence to sequence learning (sutskever et al. 2014)

- Encode source sentence with deep LSTM
- Generate target words from decoder LSTM after <EOS>
- Bootstrap training by reversing the source sentence (why?)

challenges for vanilla nmt

- Performance on long sentences is poor (why?)
- Open vocabulary translation is computationally hard
- Open vocabulary translation requires lots of data
- How can we make use of monolingual training data?

references

- Michael Nielson, http://neuralnetworksanddeeplearning.com
- C. Olah,
  http://colah.github.io/posts/2015-08-Understanding-LSTMs