NEURAL MACHINE TRANSLATION (PART 1)

David Talbot

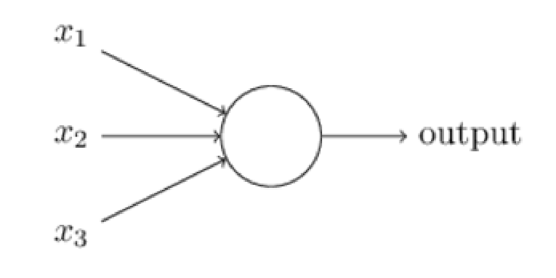
Spring 2017

Yandex School of Data Analysis

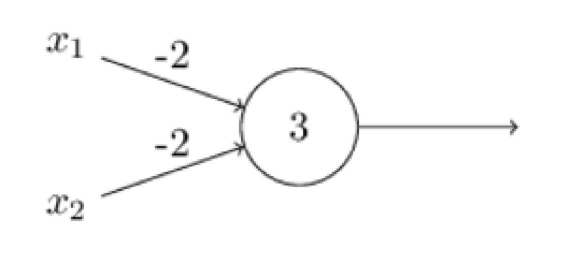
- Neural Networks (NN)
- Convolutional NN for NLP
- Recurrent Neural Networks and extensions (LSTM, GRU)
- First attempts at NMT (Recurrent Continuous Translation Model)
- Encoder-Decoders (sequence-to-sequence) models for MT

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

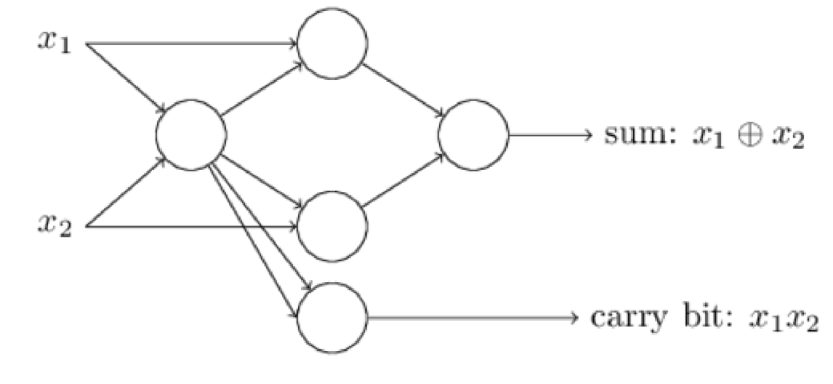
$$f(x) = \begin{cases} 1 & \text{if} \quad w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



What logic function does this compute?



What weights and biases will implement $f(x_1, x_2) = x_1 + x_2$?



Given training data

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

Choose parameters w and b such that $\forall x_i \in D$

$$f(x_i) = \left\{ \begin{array}{lr} 1 \quad \text{if} & w \cdot x_i + b > 0 \\ 0 & \text{otherwise} \end{array} \right\} \quad = y_i.$$

Perceptron learning rule

$$w_j \leftarrow w_j + \alpha(y_i - f(x_i))x_{ij}$$

Given training data

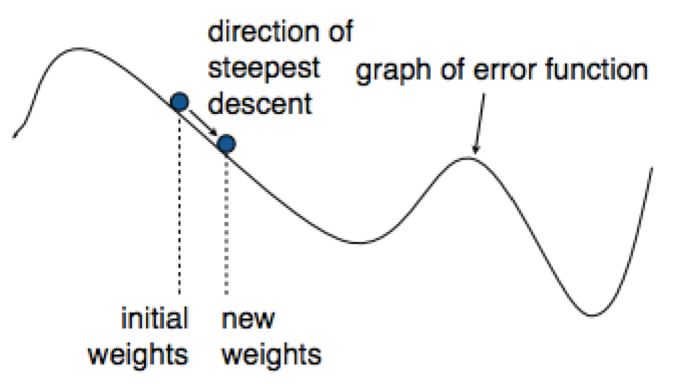
$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

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

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

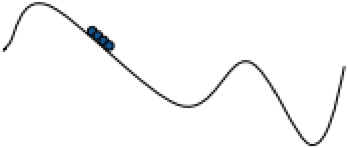
Minimize the error by updating w such that

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

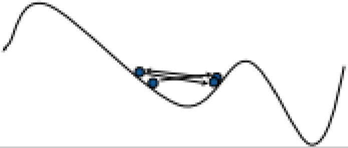

direction of
steepest
descent
graph of error function
initial
weights
new
weights

LEARNING RATE

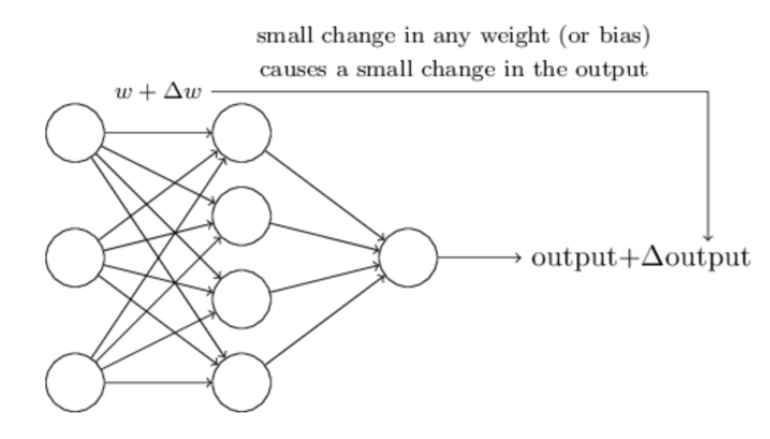
- α small: takes a long time to reach minimum of error function



- α large: may oscillate around minimum, without converging

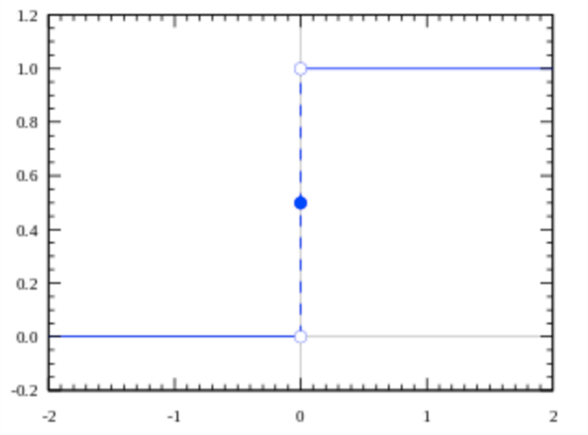


CONTINUOUS



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

PERCEPTRON ACTIVATION FUNCTION



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

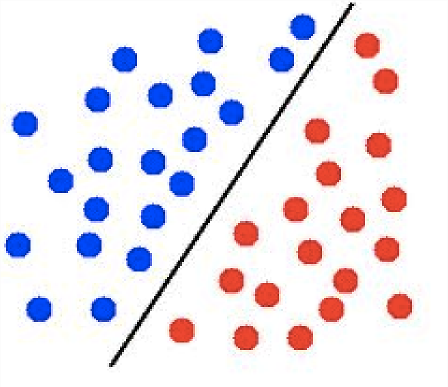
Removing the step, makes the cost function differentiable

$$C(w) \quad = \frac{1}{2}\sum_{i=1}^{n}(y_i - g(x_i))^2 \quad \text{where} \quad g(x_i) = w \cdot x_i + b$$

$$\begin{aligned}\frac{\partial C}{\partial w_j} &= \sum_{i=1}^{n}(y_i - w \cdot x_i - b)\frac{\partial}{\partial w_j}(y_i - w \cdot x_i - b) \\ &= -\sum_{i=1}^{n}(y_i - w \cdot x_i - b)x_{i,j}\end{aligned}$$

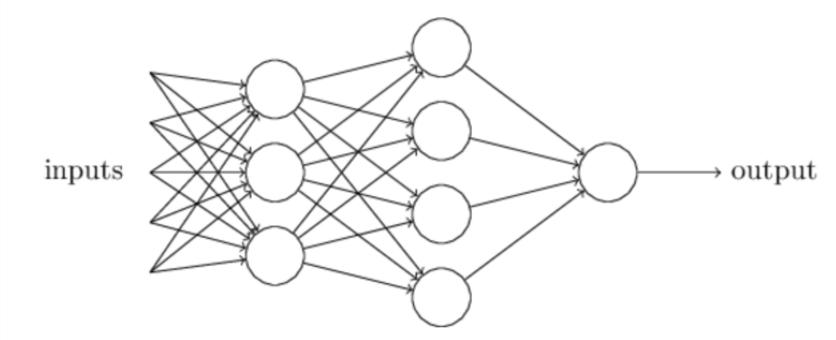
Gives us the Adaline update rule

$$w_j \leftarrow w_j + \alpha \sum_{i=1}^{n}(y_i - g(x_i))x_{i,j}$$



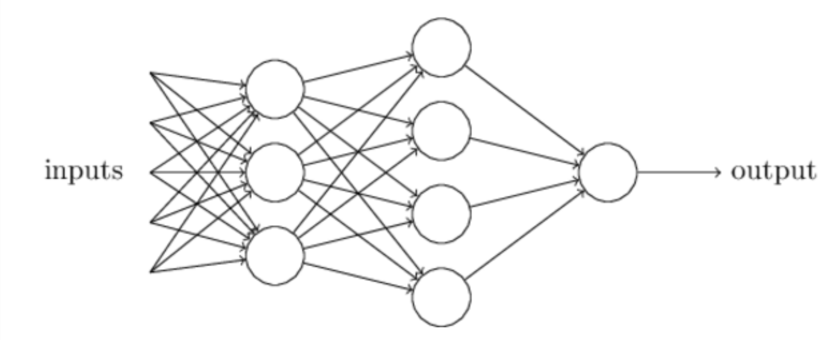
Perceptron classifies blue as 0 and red as 1 with weights
$$w = (?, ?), b = ?$$

MULTILAYER PERCEPTRON



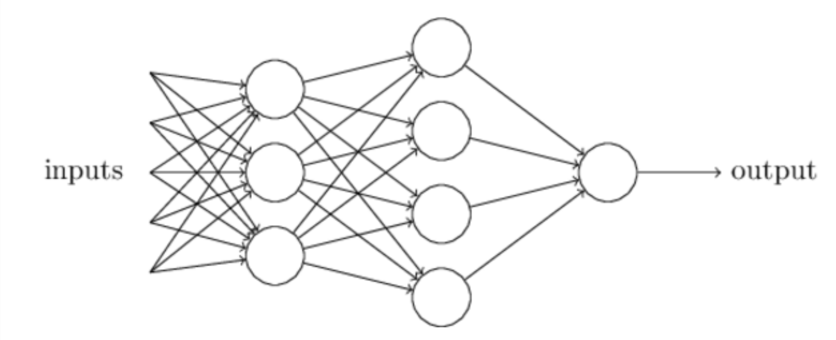
$$g(x) = f^3(\sum_{k=1}^{4} W^3_{1,k} o_k + b)$$

MULTILAYER PERCEPTRON

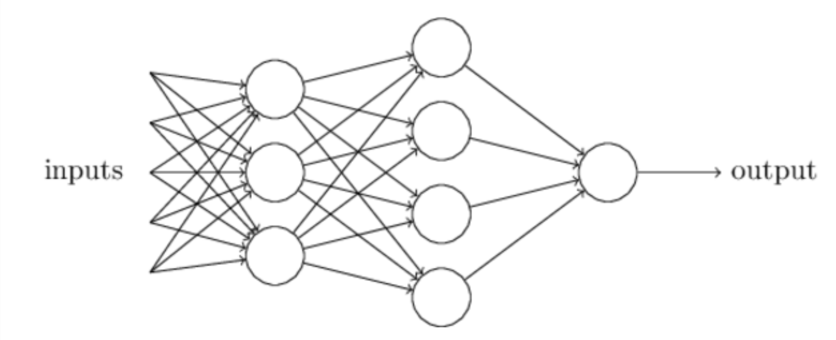


$$g(x) = f^3(\sum_{k=1}^{4} W^3_{1,k} f^2(\sum_{j=1}^{3} W^2_{k,j} o_j + b_k) + b)$$

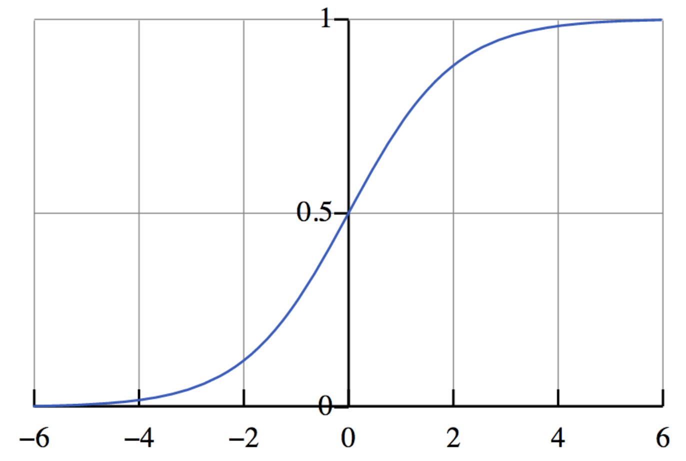
MULTILAYER PERCEPTRON



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

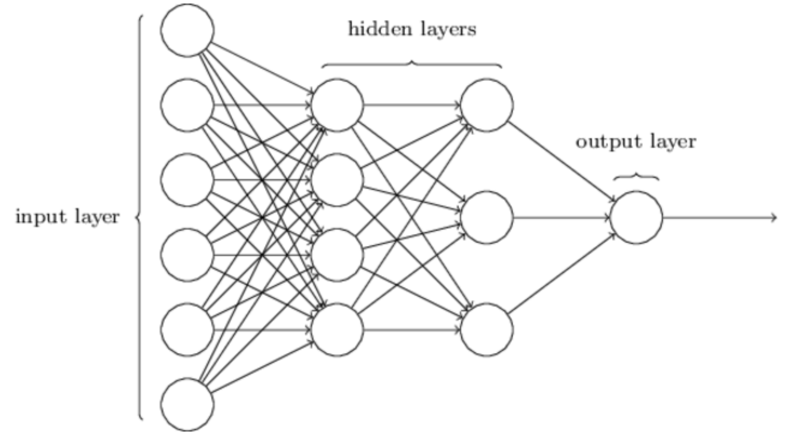


What does this buy us if activations $f^i(\cdot)$ are linear?



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

DEEP NETWORK



- 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

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

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

Compute derivatives for all parameters:

$$\frac{\partial C}{\partial w^l_{jk}}, \frac{\partial C}{\partial b^l_j} \quad \forall j, k, l$$

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

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

Let a^l be activation of j-th neuron at layer l

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)$$

Using vectors

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

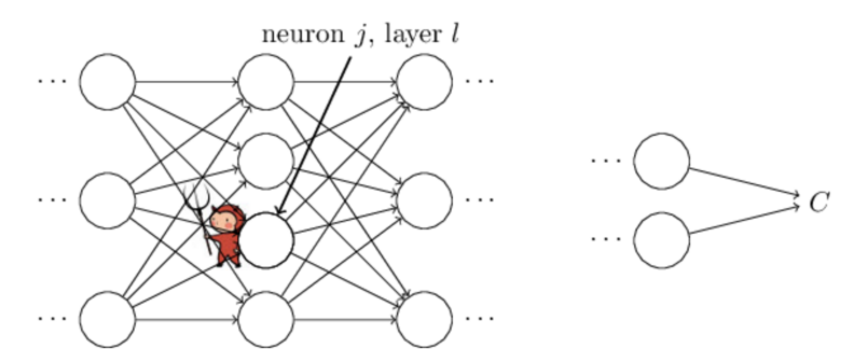
Weighted input to j-th neuron at layer l (useful below)

$$z_j^l \equiv \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

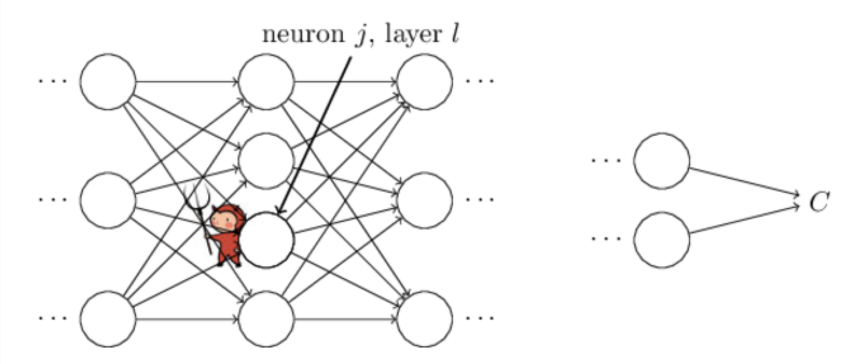
Using vectors

$$z^l \equiv w^l a^{l-1} + b^l$$

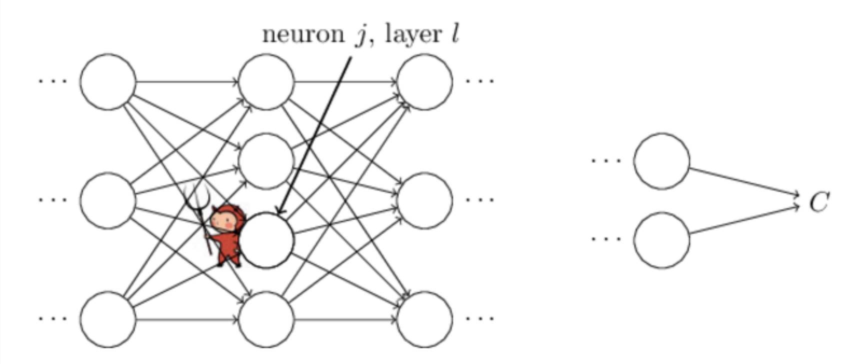
BACKPROPAGATION



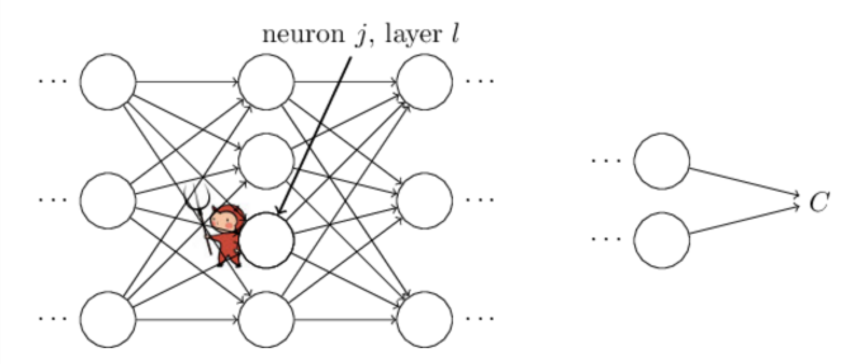
*Source: Michael Nielsen,
http://neuralnetworksanddeeplearning.com/chap2.html



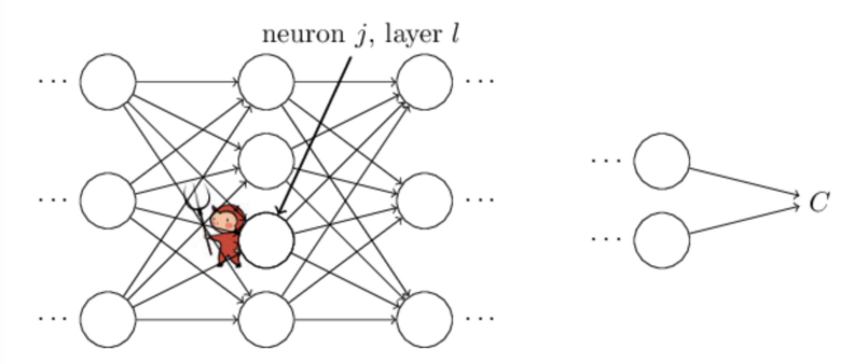
Can add Δz_j^l to input so output becomes $\sigma(z_j^l + \Delta z_j^l)$.



So overall cost changes by $\frac{\partial C}{\partial z_j^l} \Delta z_j^l$



Define error at neuron as: $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$



If $\frac{\partial C}{\partial z_j^l}$ is small, then not much to gain from changing z_j^l

By applying chain-rule to the final layer

$$
\begin{aligned}
\delta_j^L &\equiv \frac{\partial C}{\partial z_j^L} &(1)\\
&= \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{z_j^L} &(2)\\
&= \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{z_j^L} &(3)\\
&= \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) &(4)
\end{aligned}
$$

Rate of change of C w.r.t. output activations

$$\delta^L \quad = \quad \nabla_{a^L} C \odot \sigma'(z^L)$$

In case of quadratic cost

$$\delta^L \quad = \quad (a^L - y) \odot \sigma'(z^L)$$

δ_l expressed in terms of δ^{l+1}

$$
\begin{aligned}
\delta_j^l &\equiv \frac{\partial C}{\partial z_j^l} \\
&= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{z_j^l} \\
&= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}
\end{aligned}
$$

Since

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

and differentiating

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

so

$$\delta_j^l = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

BACKPROPAGATION: RECURSION (3)

$$\delta^l \quad = \quad ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$(w^{l+1})^T$ moves the error back through that layer

$\sigma'(z^l)$ moves it back through the activation

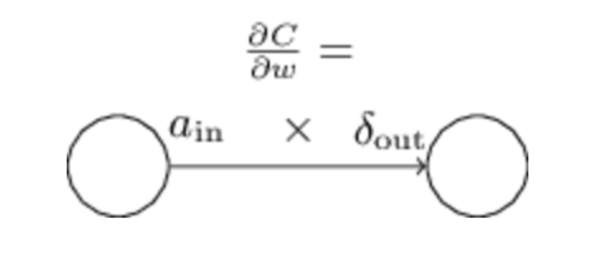
It turns out that

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

So if the error is small, the bias will not change much

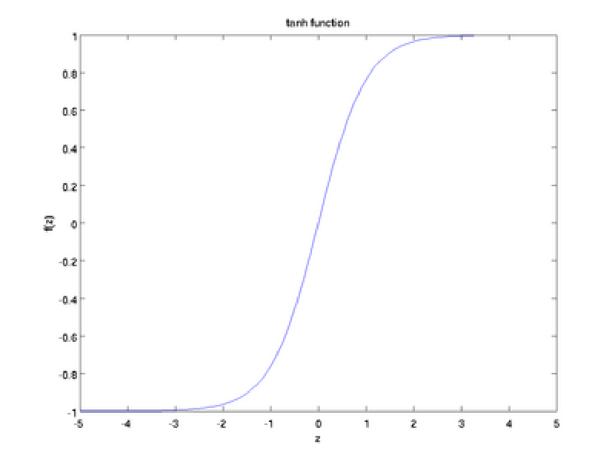
Similarly, it turns out that

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$



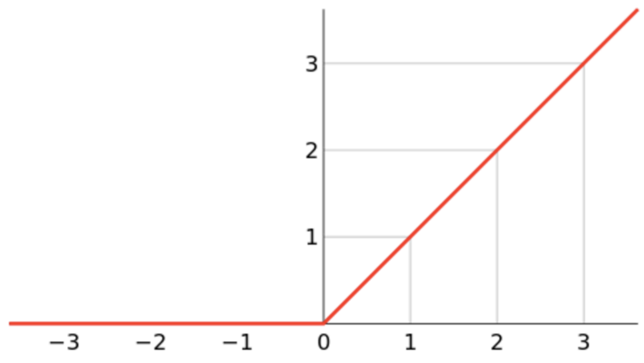
Weights from low activation neurons learn slowly

HYPERBOLIC TANGENT

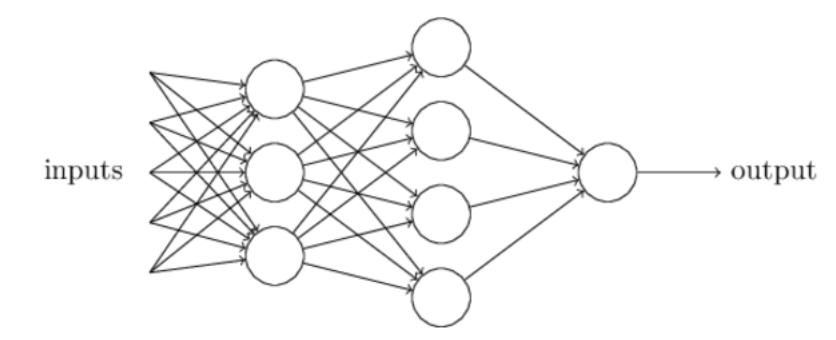


$$f(x) = tanh(wx + b)$$

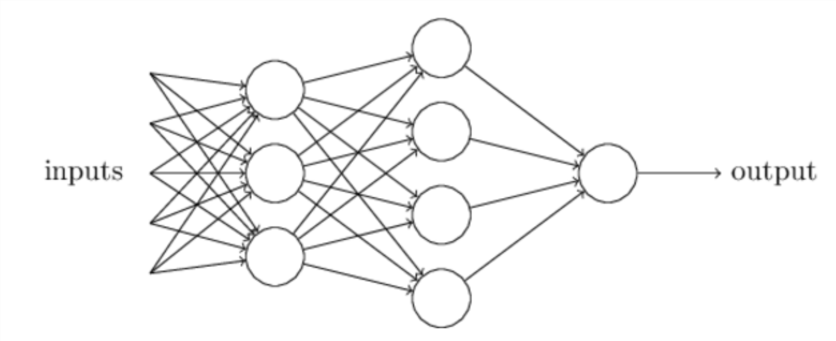
RECTIFIED LINEAR UNIT (RELU)



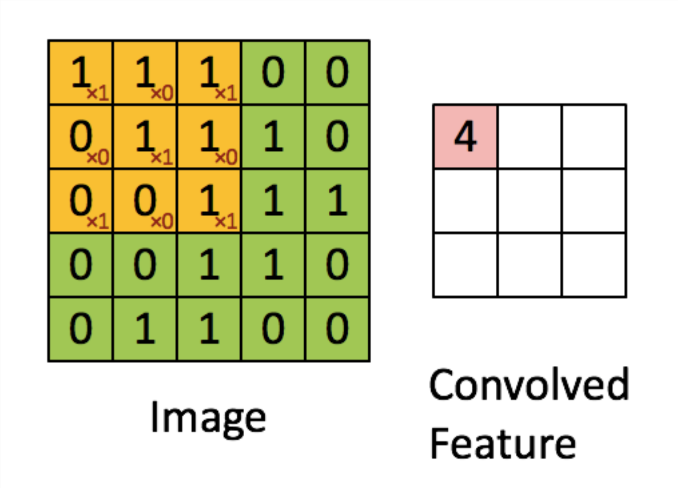
$$f(x) = max(0, wx + b)$$



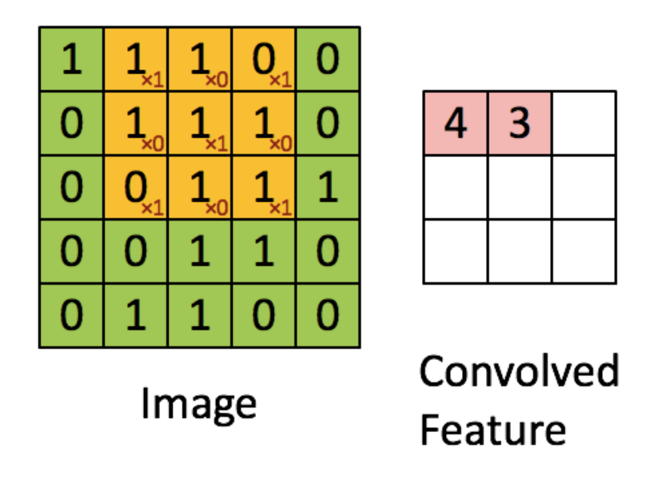
A neural network with one hidden layer can approximate an arbitrary functions (with enough hidden units)

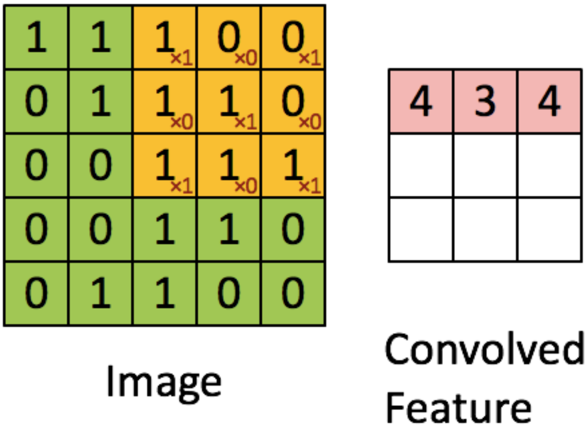


How about the inductive bias?

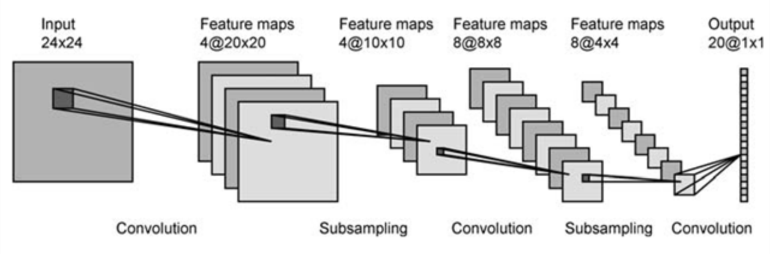

1 ×1
1 ×0
1 ×1
0
0
0 ×0
1 ×1
1 ×0
1
0
0 ×1
0 ×0
1 ×1
1
1
0
0
1
1
0
0
1
1
0
0
Image
4
Convolved Feature

CONVOLUTIONAL NETWORK




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

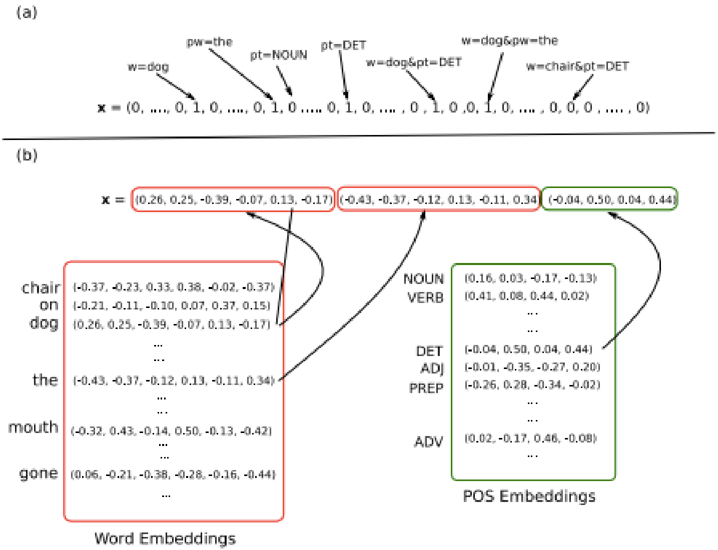
CONVOLUTIONAL NETWORK

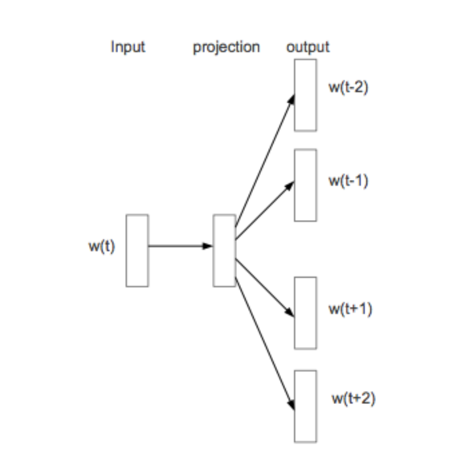


WORD EMBEDDINGS: SPARSE VS DENSE REPRESENTATIONS

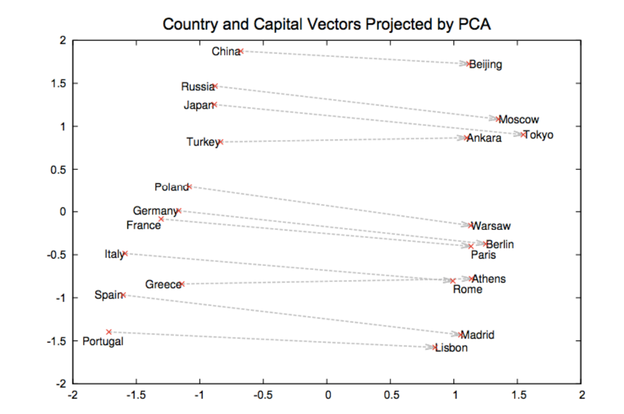
- Sparse: Each feature one dimension (binary value), each combination has its own dimension
- Dense: Each feature has a vector, no explicit encoding of feature combinations

WORD EMBEDDINGS: SPARSE VS DENSE REPRESENTATIONS

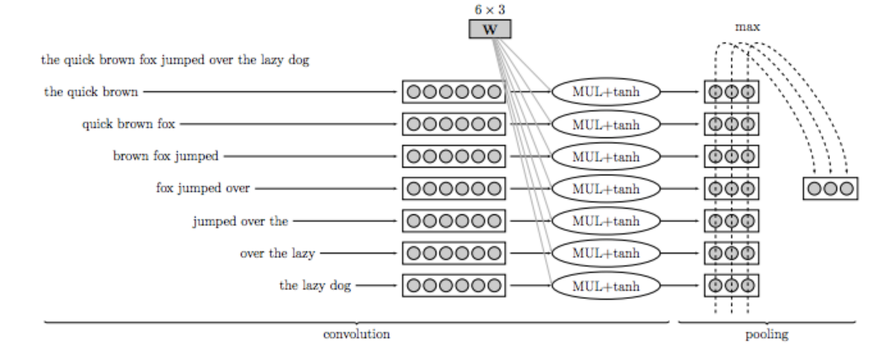




Skip gram model: predict word in random position close to w_t

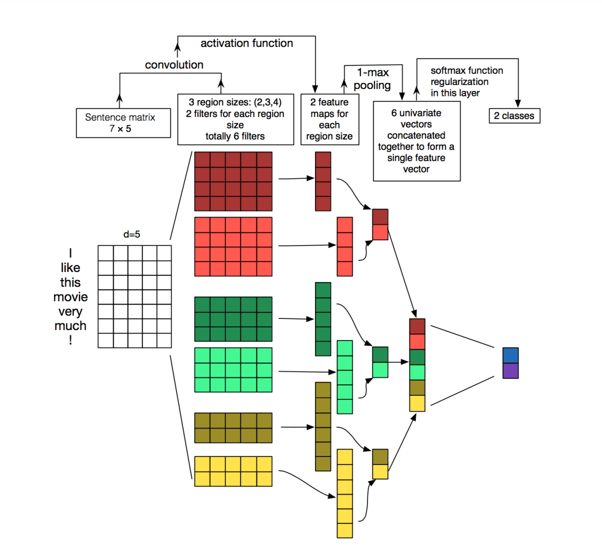


Magic of word embeddings? (More later)



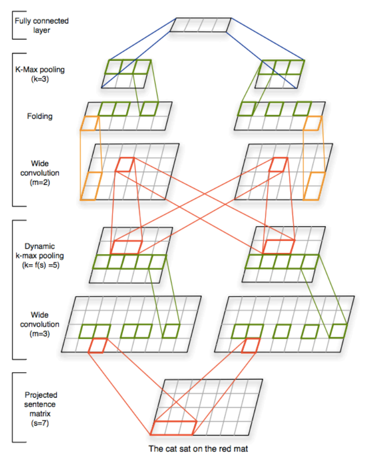
(Source: Goldberg, 2015)

CONVOLUTIONAL NETWORK



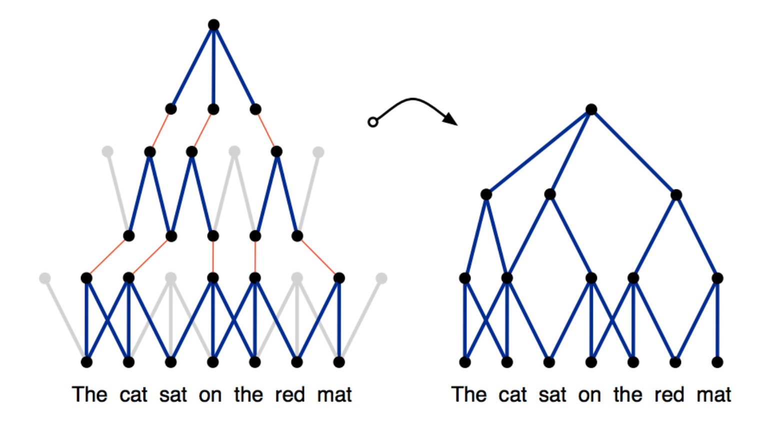
Source: Zhang, Y., & Wallace, B. (2015)

CONVOLUTIONAL SENTENCE MODEL



Source: Kalchbrenner et al. (2015)

Encoder in first NMT approach (Kalchbrenner & Blunsom 2013)



Source: Kalchbrenner et al. (2015)

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

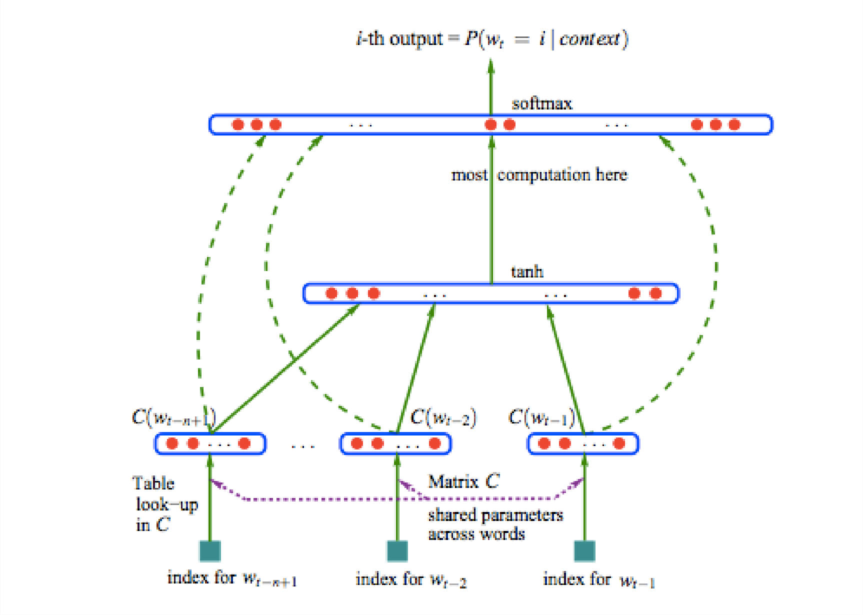
$$f(w_t, \ldots, w_{t-n+1}) = \Pr(w_t | w_1^{t-1})$$

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)



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

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

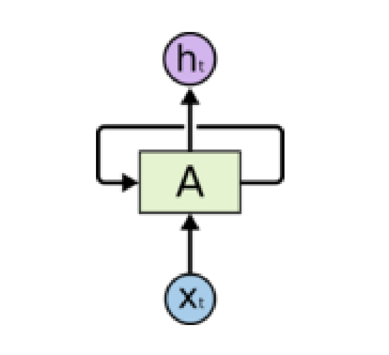
and

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

- 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

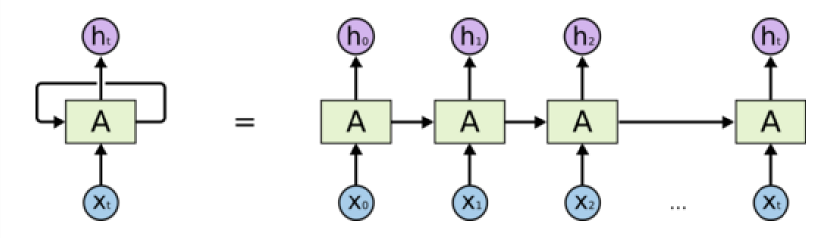
neural probabilistic language model (bengio et al. 2003)

- 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



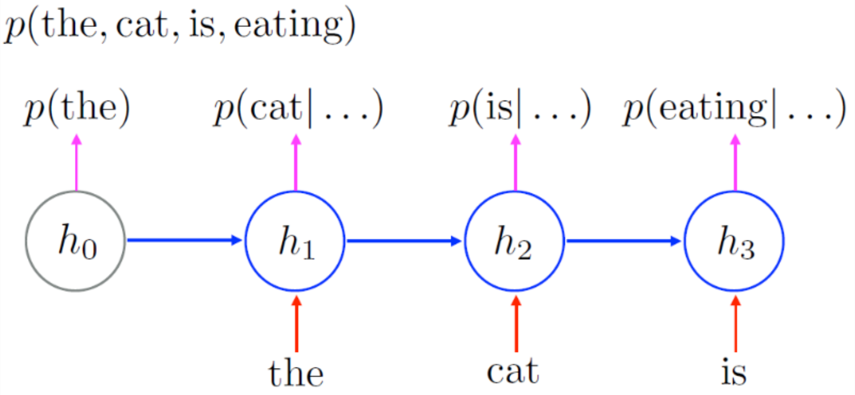
State A_t updated from current input x_t and previous state A_{t-1}

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



Parameters shared across time steps

RECURRENT NEURAL NETWORK LANGUAGE MODELS (MIKOLOV 2010)



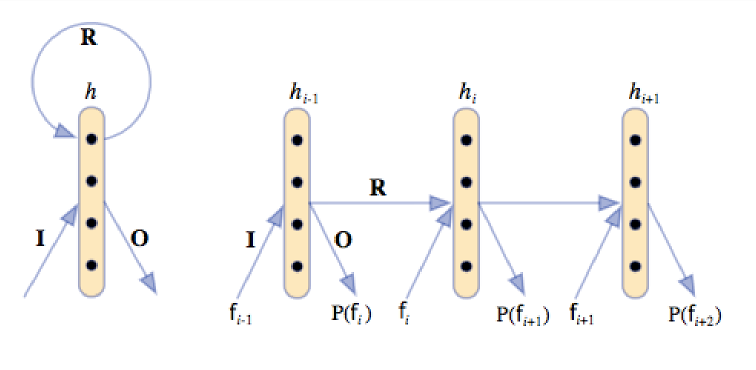
Kalchbrenner & Blunsom, (2013)

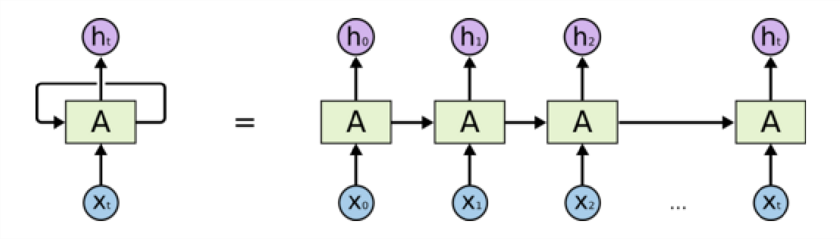
- First end-to-end NMT system
- Lower perplexity (average likelihood) than IBM models
- Encode source sentence with convolutional network
- RNN decoder generates target sentence conditioned on source sentence encoded in a ConvNN

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

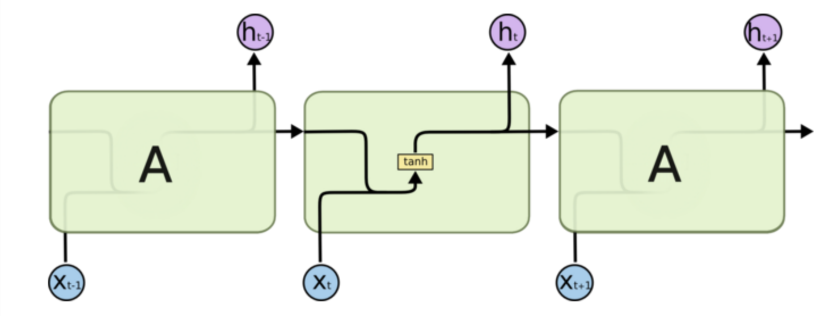
(See Appendix for details)

RECURRENT CONTINUOUS TRANSLATION MODELS



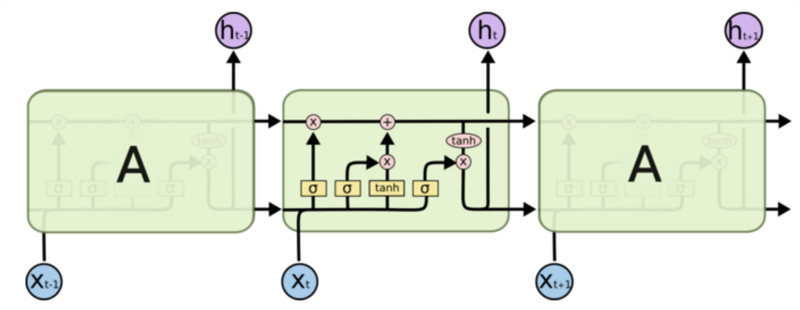


Why might backpropagation fail on such a ’deep’ network?



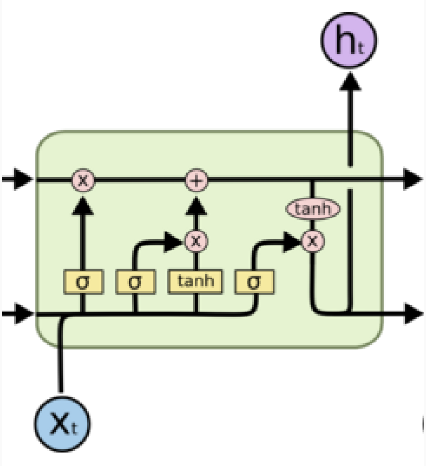
h_{t-1}
h_t
h_{t+1}
A
tanh
A
X_{t-1}
X_t
X_{t+1}

LONG SHORT TERM MEMORY UNITS (HOCHREITER & SCHMIDHUBER, 1997)

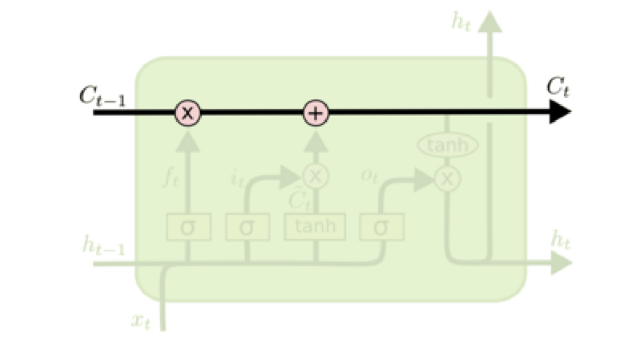


Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs

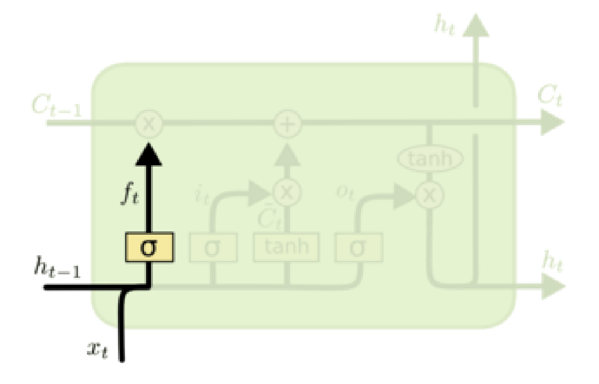
LSTM



Separate cell C_t at each time frame to propagate information

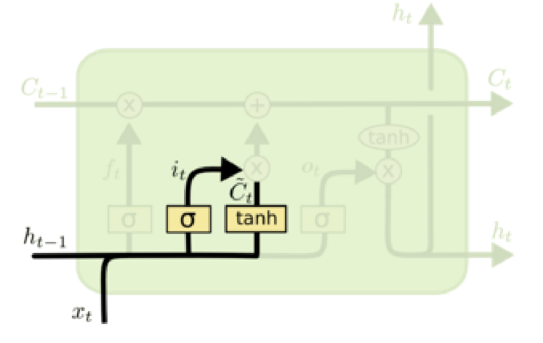


Controls how much each dimension of C_{t-1} is propagated to C_t



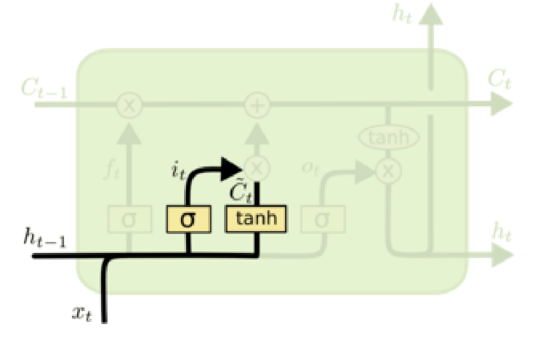
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

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



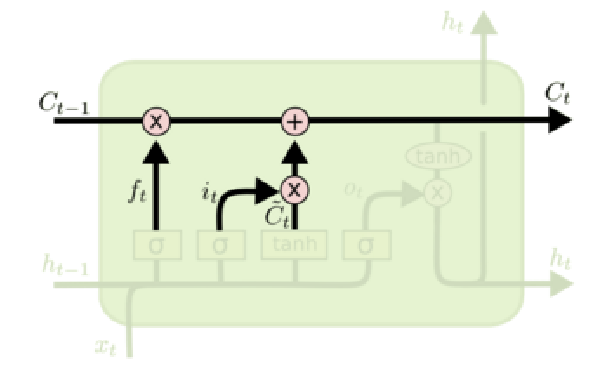
$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Determines how much each dimension should be updated



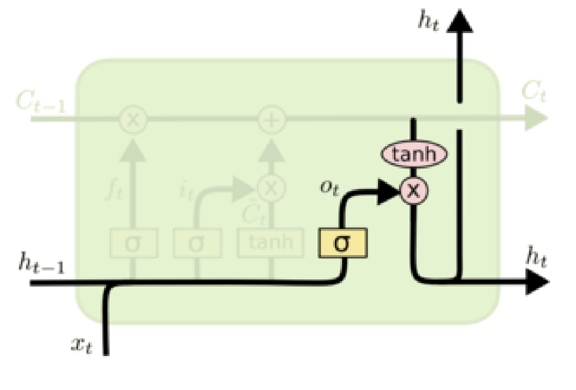
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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



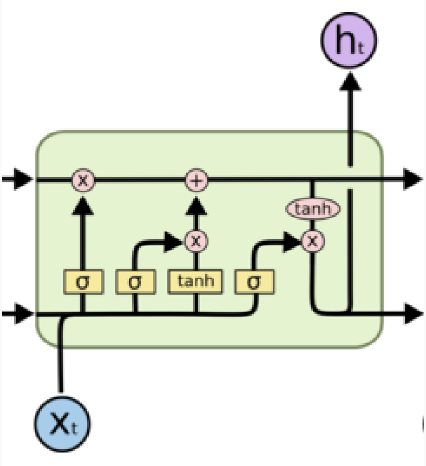
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

LSTM: OUTPUT GATE

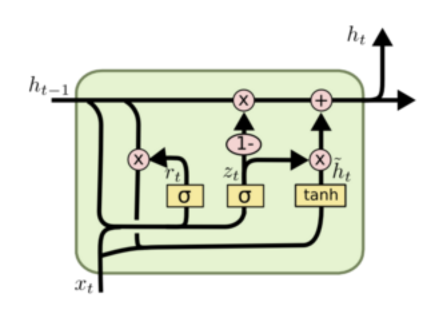


$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot tanh(C_t)$$

LSTM



GATED RECURSIVE UNITS (CHO ET AL. 2014)

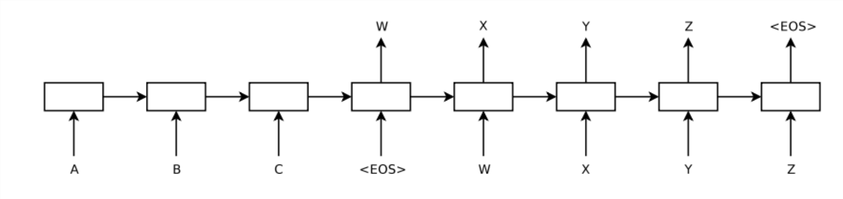


$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

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

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

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



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

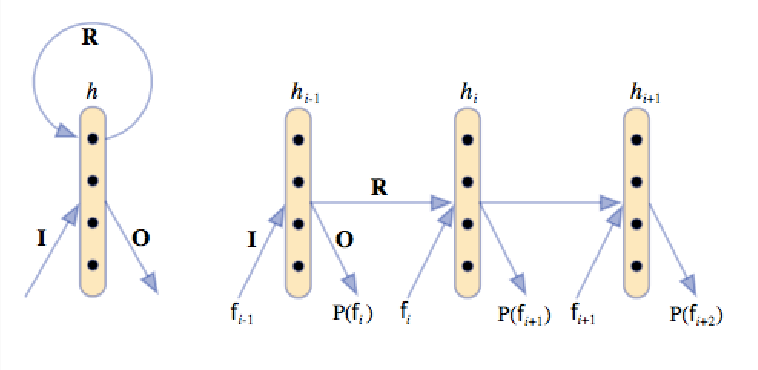
- Performance on long sentences is poor (why?)
- Open vocabulary translation is computationally hard
- Open vocabulary translation requires lots of data
- How can we make use of monolingual training data?

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

APPENDIX: RECURRENT CONTINUOUS TRANSLATION MODELS

- First end-to-end NMT system (Kalchbrenner & Blunsom, (2013))
- 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

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


R
h
I
O
h_{i-1}
h_i
h_{i+1}
R
I
O
f_{i-1}
$P(f_i)$
f_i
$P(f_{i+1})$
f_{i+1}
$P(f_{i+2})$

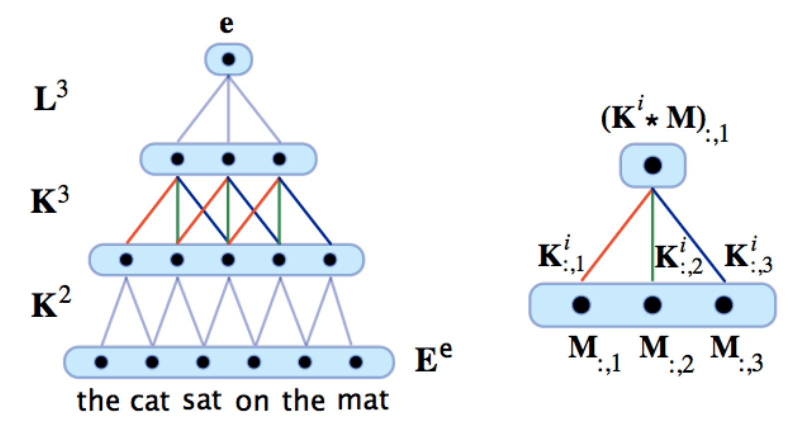
RECURRENT CONTINUOUS TRANSLATION MODELS

- $\mathrm{I} \in \mathbb{R}^{q \times |V|}$ (input vocabulary embedding)
- $\mathrm{R} \in \mathbb{R}^{q \times q}$ (recurrent transformation)
- $\mathrm{O} \in \mathbb{R}^{|V| \times q}$ (output vocabulary mapping)

With $v(f_i)$ as the one-hot representation of $f_i \in V$

$$
\begin{aligned}
h_1 &= \sigma(\mathrm{I} \cdot v(f_1)) \\
h_{i+1} &= \sigma(\mathrm{R} \cdot h_i + I \cdot v(f_{i+1})) \\
o_{i+1} &= \mathrm{O} \cdot h_i
\end{aligned}
$$

$$
\Pr(f_i = v | f_{1:i-1}) = \frac{\exp(o_{i,v})}{\sum_{v'=1}^{|V|} \exp(o_{i,v'})}
$$



e
L^3
K^3
K^2
E^e
the cat sat on the mat
$(K^i * M)_{:,1}$
$K^i_{:,1}$
$K^i_{:,2}$
$K^i_{:,3}$
$M_{:,1}$
$M_{:,2}$
$M_{:,3}$

Given a source sentence $e = e_1, \ldots, e_k$

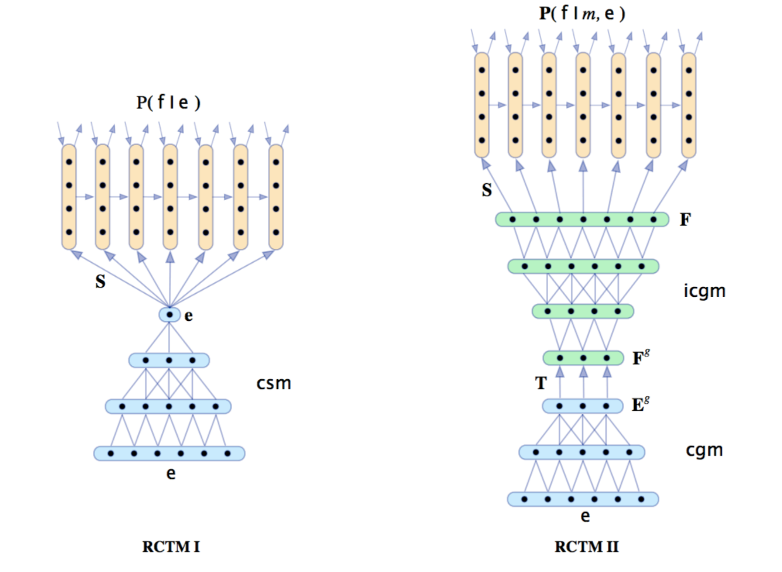
- $\mathrm{E}^{\mathrm{e}} \in \mathbb{R}^{q \times k}$ (source sentence matrix)
- $\mathrm{K}^{i} \in \mathbb{R}^{q \times i}$ ($r = \lceil \sqrt{2N} \rceil$ convolution filters; N is max length)

Each convolution reduces dimensionality by $i - 1$

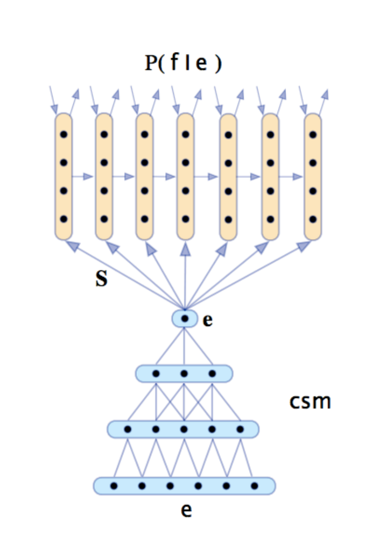
$$\begin{aligned} \mathrm{E}_1^{e} &= \mathrm{E}^{e} \\ \mathrm{E}_{i+1}^{e} &= \sigma(\mathrm{K}^{i+1} * \mathrm{E}_i^{e}) \end{aligned}$$

Result is a vector $\mathrm{e} \in \mathbb{R}^{q \times 1}$

(L^{j} used to map convolved matrix E_i^{e} to $\mathbb{R}^{q \times 1}$ if fewer than $i + 1$ columns.)


P(f | e)
S
e
csm
e
RCTM I
P(f | m, e)
S
F
icgm
F^g
T
E^g
cgm
e
RCTM II

RECURRENT CONTINUOUS TRANSLATION MODEL 1



- $\mathrm{I} \in \mathbb{R}^{q \times |V^f|}$ (target vocabulary embedding)
- $\mathrm{R} \in \mathbb{R}^{q \times q}$ (recurrent transformation)
- $\mathrm{O} \in \mathbb{R}^{|V^f| \times q}$ (output vocabulary mapping)
- $\mathrm{S} \in \mathbb{R}^{q \times q}$ (sentence mapping)

With $csm(e)$ as the convolutional sentence model

$$\begin{aligned} \mathrm{s} &= \mathrm{S} \cdot csm(e) \\ h_1 &= \sigma(\mathrm{I} \cdot v(f_1)) \\ h_{i+1} &= \sigma(\mathrm{R} \cdot h_i + I \cdot v(f_{i+1})) \\ o_{i+1} &= \mathrm{O} \cdot h_i \end{aligned}$$

$$\Pr(f_i = v | f_{1:i-1}, \mathrm{e}) = \frac{\exp(o_{i,v})}{\sum_{v'=1}^{|V|} \exp(o_{i,v'})}$$

Problems with Model 1

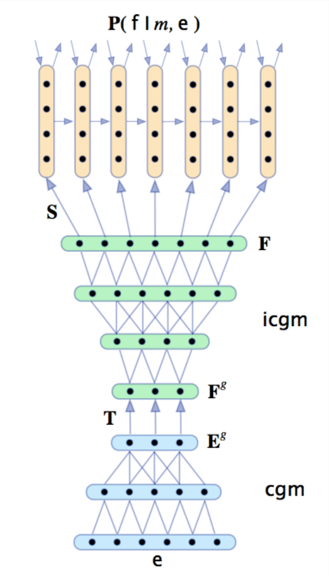
- RNN controls length of sentence (bias towards shorter sentences)
- Each target word is equally conditioned on the full source sentence

Model sentence length explicitly

$$\begin{aligned}\Pr(f|e) &= \Pr(m|e) \cdot \Pr(f|m, e) \\ &= \Pr(m|e) \prod_{i=1}^{m} \Pr(f_{i+1}|f_{1:i}, m, e)\end{aligned}$$

Where given source sentence e with length k

$$\Pr(m|e) \approx \Pr(m|k) \approx \mathit{Poisson}(\lambda_k)$$


P(f | m, e)
S
F
icgm
F^g
T
E^g
cgm
e

RECURRENT CONTINUOUS TRANSLATION MODEL 2

Retain some positional information (cf. attention later)

- E_i^e columns represent n-grams, e.g. E_2^e are bigrams
- Truncate convolutions to get 4-gram source features
- Invert convolutional sentence model on target side
- Target words are not uniformly conditioned on e
- Additional parameters: $T^{q \times q}$ translation matrix and inverted filters

- Trained with BPTT to optimize cross-entropy of parallel data
- Factor output to reduce computation

$$\Pr(f_i | f_{1:i-1}, \mathrm{e}) = \Pr(C(f_i) | f_{1:i-1}, \mathrm{e}) \cdot \Pr(f_i | C(f_i), f_{1:i-1}, \mathrm{e})$$

- Softmax with $|V^f|$ terms reduces two with $|C| + |V^f|/|C|$ terms

$$\Pr(C(f_i) = c | f_{1:i-1}, \mathrm{e}) = \frac{\exp(o_{i,c})}{\sum_{c'=1}^{|C|} \exp(o_{i,c'})}$$

$$\Pr(f_i = v | (C(f_i), f_{1:i-1}, \mathrm{e}) = \frac{\exp(o_{i,v})}{\sum_{v'=1}^{|C(f_i)|} \exp(o_{i,v'})}$$

WMT-NT	*2009*	*2010*	*2011*	*2012*
KN-5	218	213	222	225
RLM	178	169	178	181
IBM 1	207	200	188	197
FA-IBM 2	153	146	135	144
RCTM I	143	134	140	142
RCTM II	**86**	**77**	**76**	**77**

Table 1: Perplexity results on the WMT-NT sets.

WMT-NT PERM	*2009*	*2010*	*2011*	*2012*
RCTM II	174	168	175	178

Table 2: Perplexity results of the RCTM II on the WMT-NT sets where the words in the English source sentences are randomly permuted.

WMT-NT	*2009*	*2010*	*2011*	*2012*
RCTM I + WP	19.7	21.1	22.5	21.5
RCTM II + WP	19.8	21.1	22.5	21.7
`cdec` (12 features)	19.9	21.2	22.6	21.8

Table 4: Bleu scores on the WMT-NT sets of each RCTM linearly interpolated with a word penalty WP. The `cdec` system includes WP as well as five translation models and two language modelling features, among others.