Neural Nets for NLP

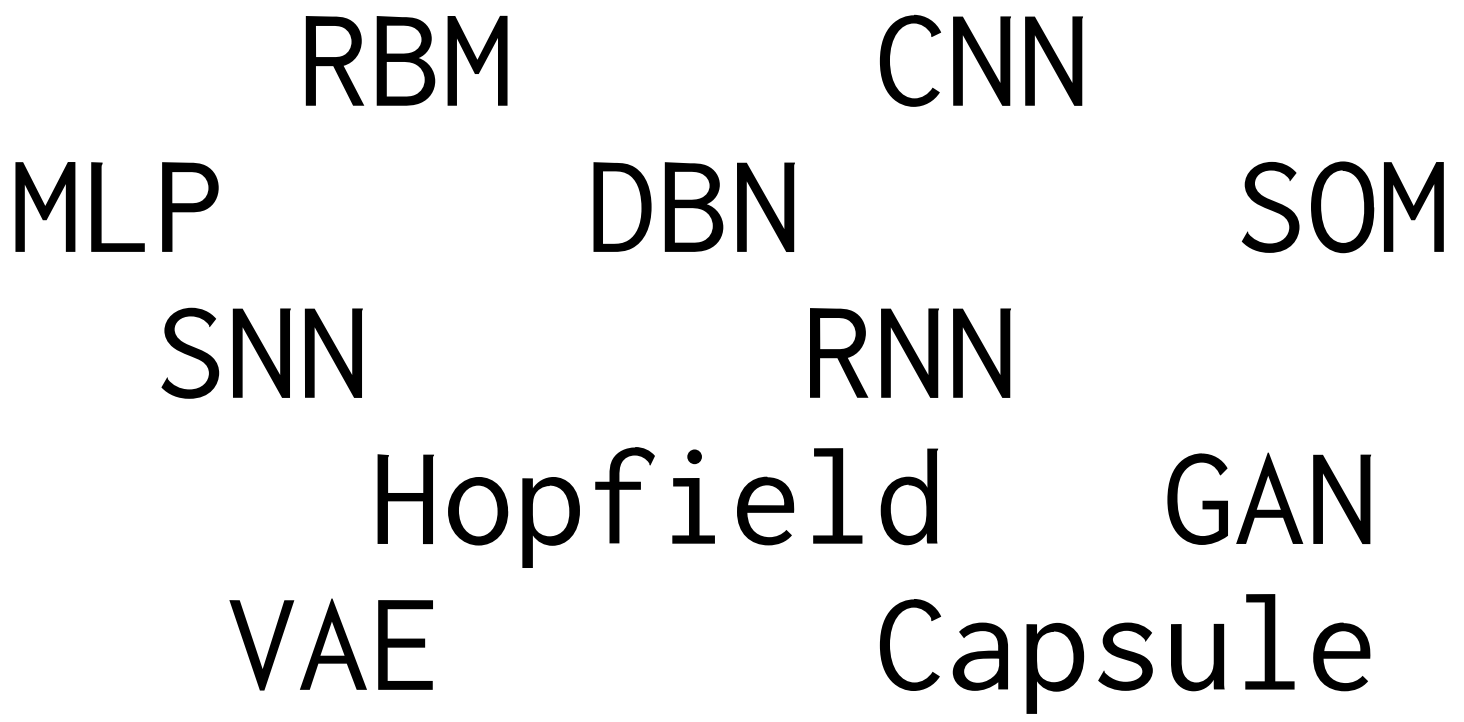
Vsevolod Dyomkin

prj-nlp-1, 2018-05-10

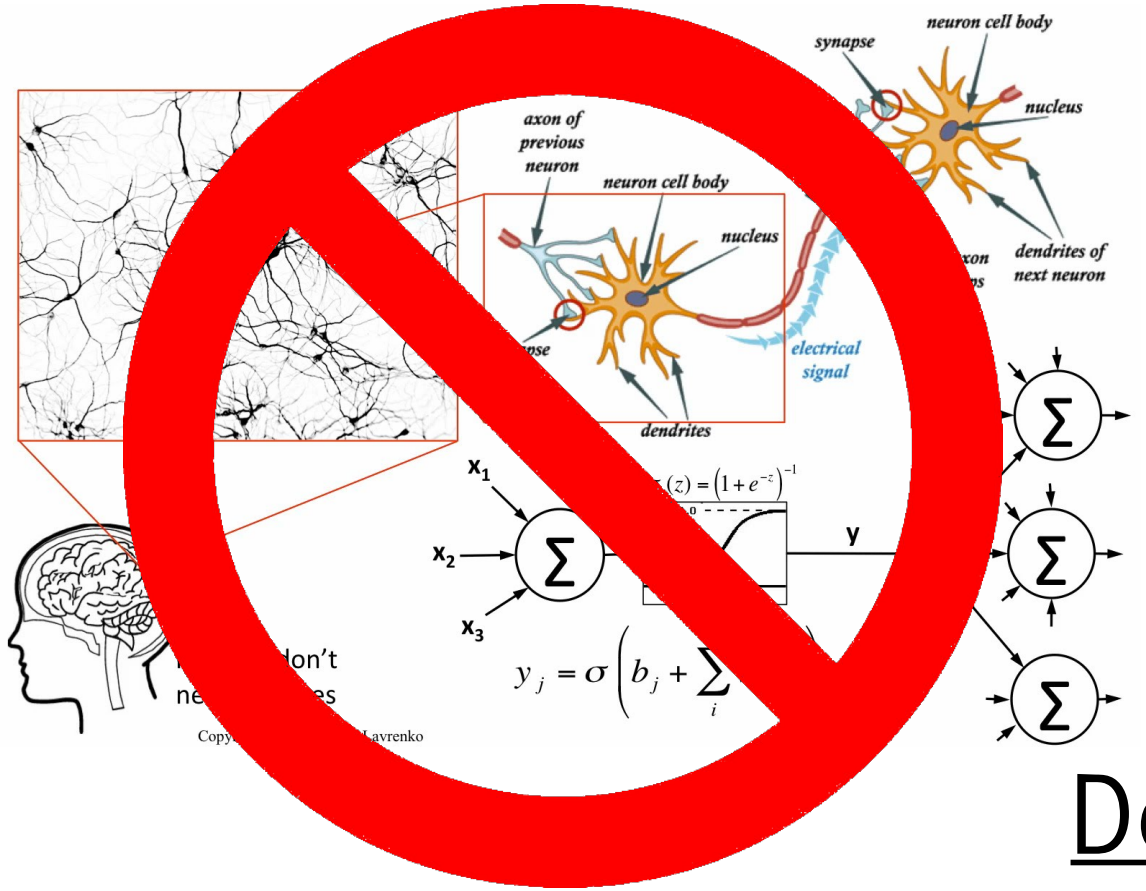
Limitations of "Classic" NLP

* categorical (1-hot) features
* extra large feature spaces
* UNK problems
* complicated feature engineering
* difficult domain adaptation
* need for markovization in sequence models
* what else?

Neural Nets to the Rescue



Terminology



Deep Learning?

https://blog.keras.io/the-limitations-of-deep-learning.html

Geometric View

The most surprising thing about deep learning is how simple it is. Ten years ago, no one expected that we would achieve such amazing results on machine perception problems by using simple parametric models trained with gradient descent. Now, it turns out that all you need is sufficiently large parametric models trained with gradient descent on sufficiently many examples. As Feynman once said about the universe, "It's not complicated, it's just a lot of it".

In deep learning, everything is a vector, i.e. everything is a point in a geometric space. Model inputs (it could be text, images, etc) and targets are first "vectorized", i.e. turned into some initial input vector space and target vector space. Each layer in a deep learning model operates one simple geometric transformation on the data that goes through it. Together, the chain of layers of the model forms one very complex geometric transformation, broken down into a series of simple ones. This complex transformation attempts to maps the input space to the target space, one point at a time. This transformation is parametrized by the weights of the layers, which are iteratively updated based on how well the model is currently performing. A key characteristic of this geometric transformation is that it must be differentiable, which is required in order for us to be able to learn its parameters via gradient descent. Intuitively, this means that the geometric morphing from inputs to outputs must be smooth and continuous—a significant constraint.

The whole process of applying this complex geometric transformation to the input data can be visualized in 3D by imagining a person trying to uncrumple a paper ball: the crumpled paper ball is the manifold of the input data that the model starts with. Each movement operated by the person on the paper ball is similar to a simple geometric transformation operated by one layer. The full uncrumpling gesture sequence is the complex transformation of the entire model. Deep learning models are mathematical machines for uncrumpling complicated manifolds of high-dimensional data.

That's the magic of deep learning: turning meaning into vectors, into geometric spaces, then incrementally learning complex geometric transformations that map one space to another. All you need are spaces of sufficiently high dimensionality in order to capture the full scope of the relationships found in the original data.

The Limitations of DL

The space of applications that can be implemented with this simple strategy is nearly infinite. And yet, many more applications are completely out of reach for current deep learning techniques—even given vast amounts of human-annotated data. Say, for instance, that you could assemble a dataset of hundreds of thousands—even millions—of English language descriptions of the features of a software product, as written by a product manager, as well as the corresponding source code developed by a team of engineers to meet these requirements. Even with this data, you could not train a deep learning model to simply read a product description and generate the appropriate codebase. That's just one example among many. In general, anything that requires reasoning—like programming, or applying the scientific method—long-term planning, and algorithmic-like data manipulation, is out of reach for deep learning models, no matter how much data you throw at them. Even learning a sorting algorithm with a deep neural network is tremendously difficult.

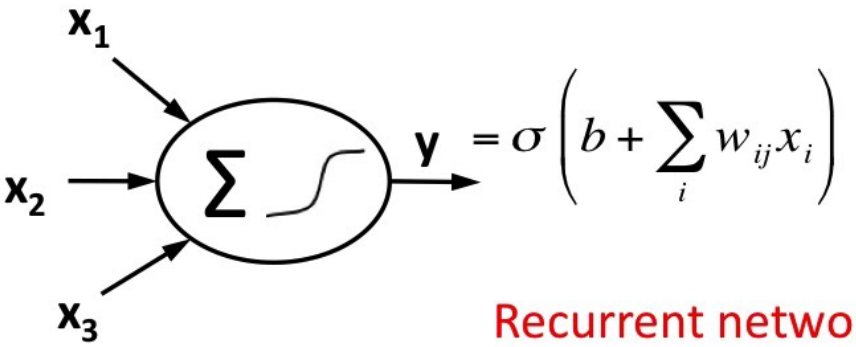
This is because a deep learning model is "just" a chain of simple, continuous geometric transformations mapping one vector space into another. All it can do is map one data manifold X into another manifold Y, assuming the existence of a learnable continuous transform from X to Y, and the availability of a dense sampling of X:Y to use as training data. So even though a deep learning model can be interpreted as a kind of program, inversely most programs cannot be expressed as deep learning models—for most tasks, either there exists no corresponding practically-sized deep neural network that solves the task, or even if there exists one, it may not be learnable, i.e. the corresponding geometric transform may be far too complex, or there may not be appropriate data available to learn it.

Scaling up current deep learning techniques by stacking more layers and using more training data can only superficially palliate some of these issues. It will not solve the more fundamental problem that deep learning models are very limited in what they can represent, and that most of the programs that one may wish to learn cannot be expressed as a continuous geometric morphing of a data manifold.

~~MLPs~~ FNNs

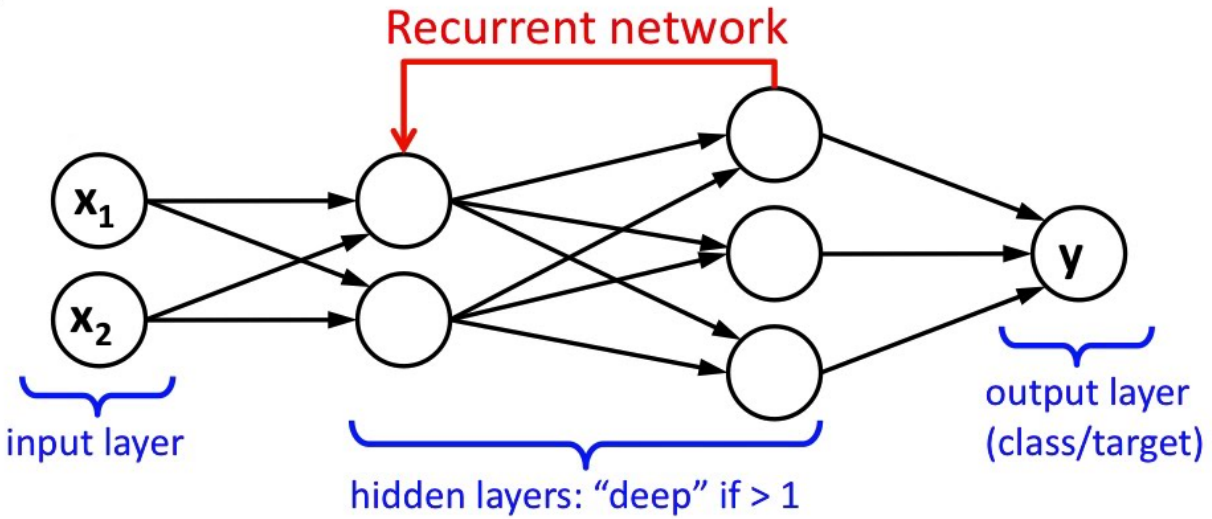
* computational graph
* composed of various layers
* backprop for learning
* GD for optimization

Types of Neural Networks



Single neuron: perceptron,

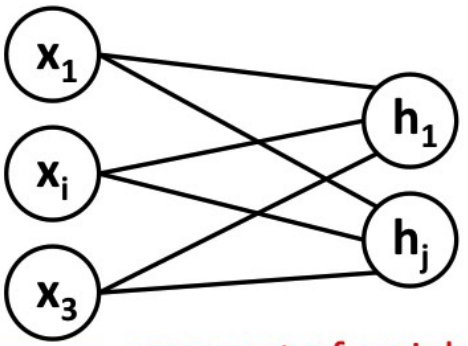
linear / logistic regression



Feed-forward network
(no cycles) -- non-linear
classification & regression

P (input | hidden)

$$\sigma\left(\beta_i + \sum_j w_{ij} h_j\right) =$$



P (hidden | input)

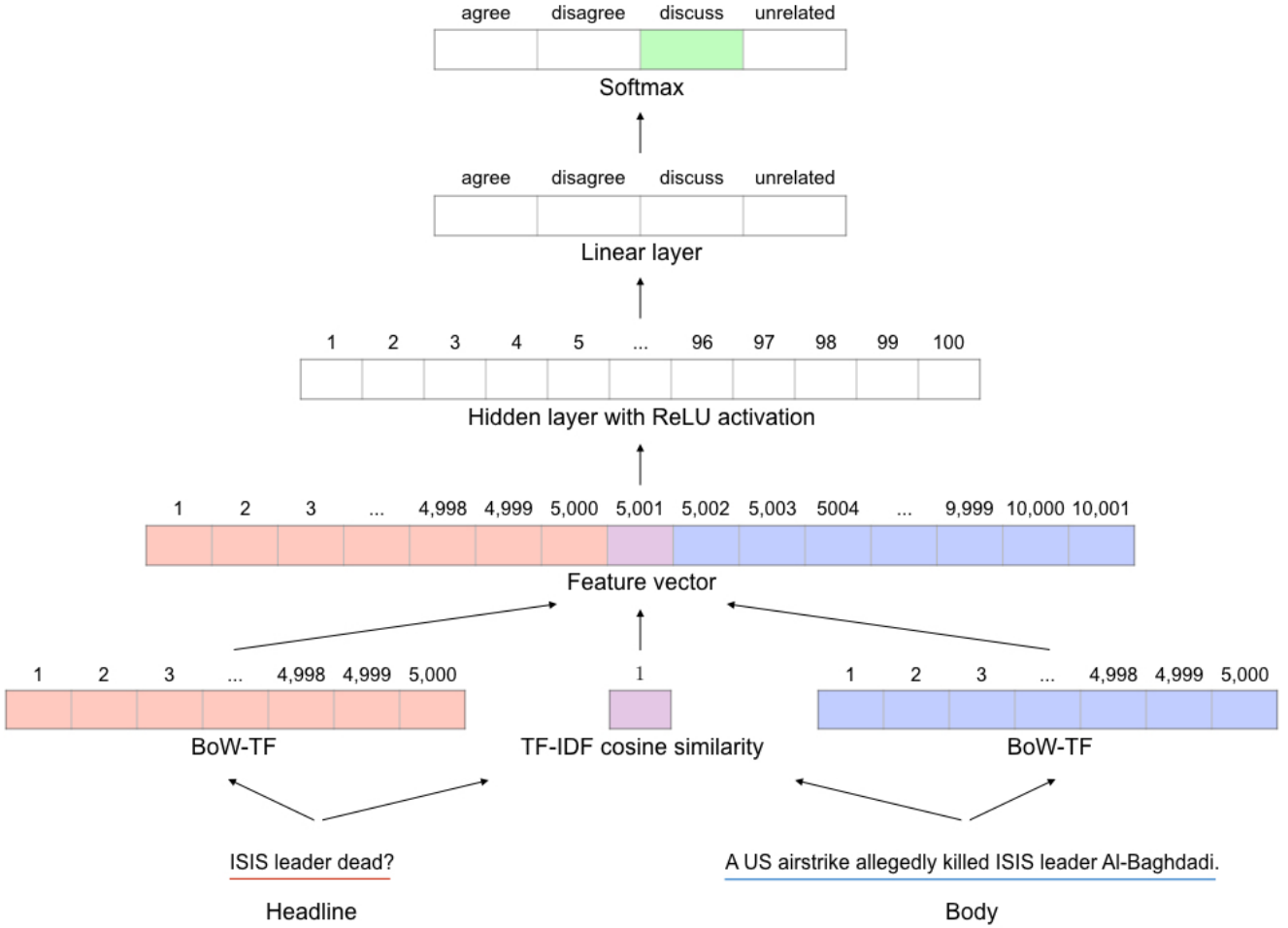
$$= \sigma\left(b_j + \sum_i w_{ij} x_i\right)$$

same set of weights

Symmetric (RBM)
unsupervised, trained
to maximize likelihood
of input data

a mixture model

Example: FNC-1



https://github.com/uclmr/fakenewschallenge

Layers

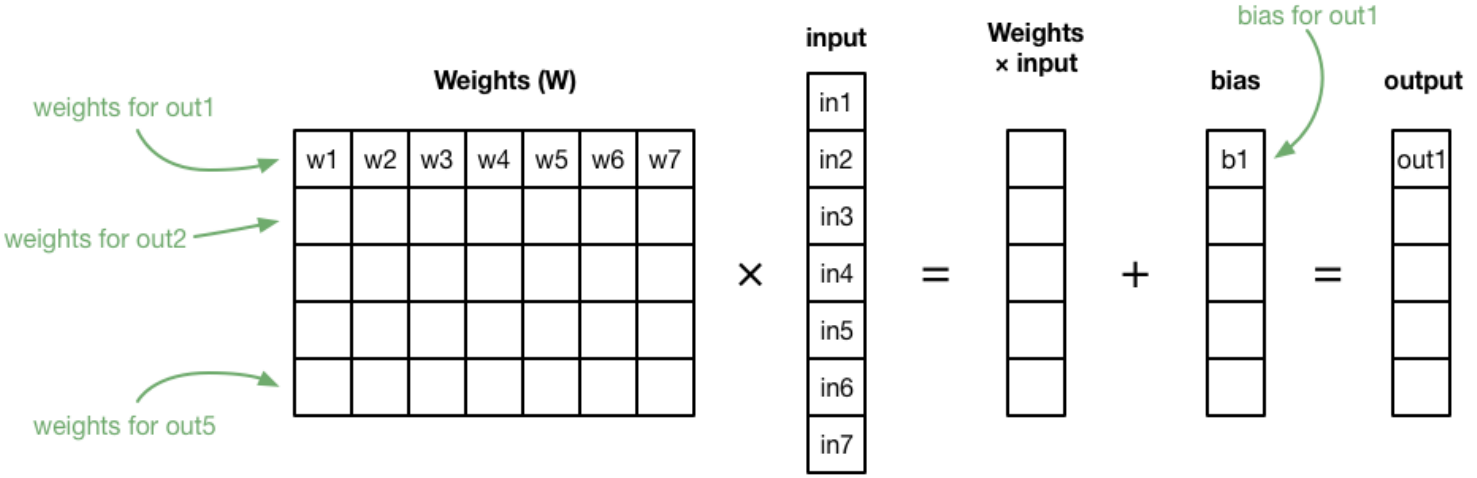
* input
* fully-connected
* convolutional
* non-linearities
* regularization
* output

Input Layers

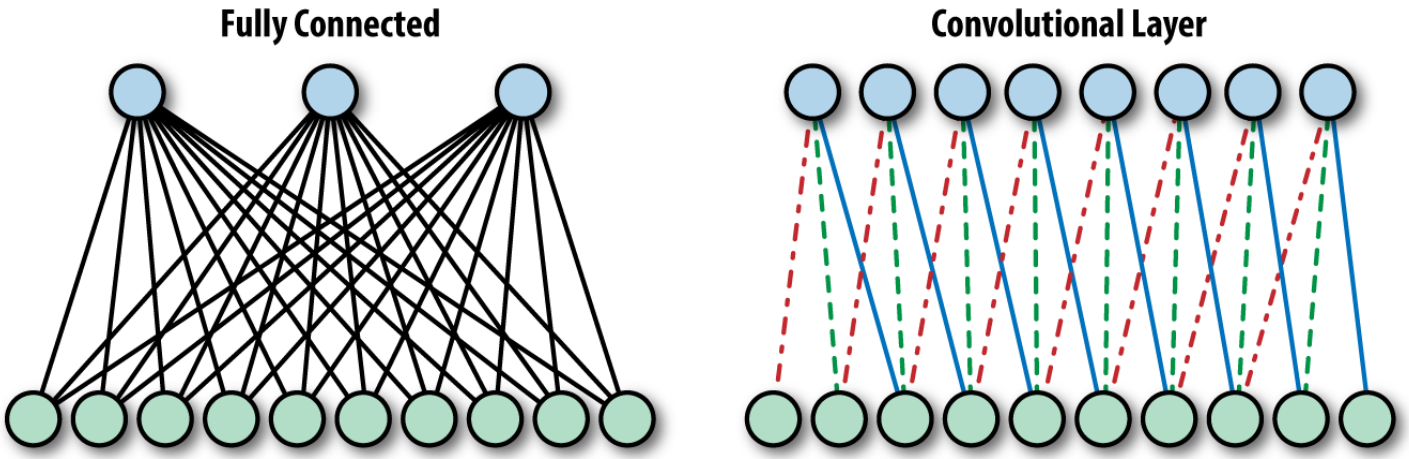
* 1-hot
* embedding
* mixed

Fully-Connected Layers

* linear transformation

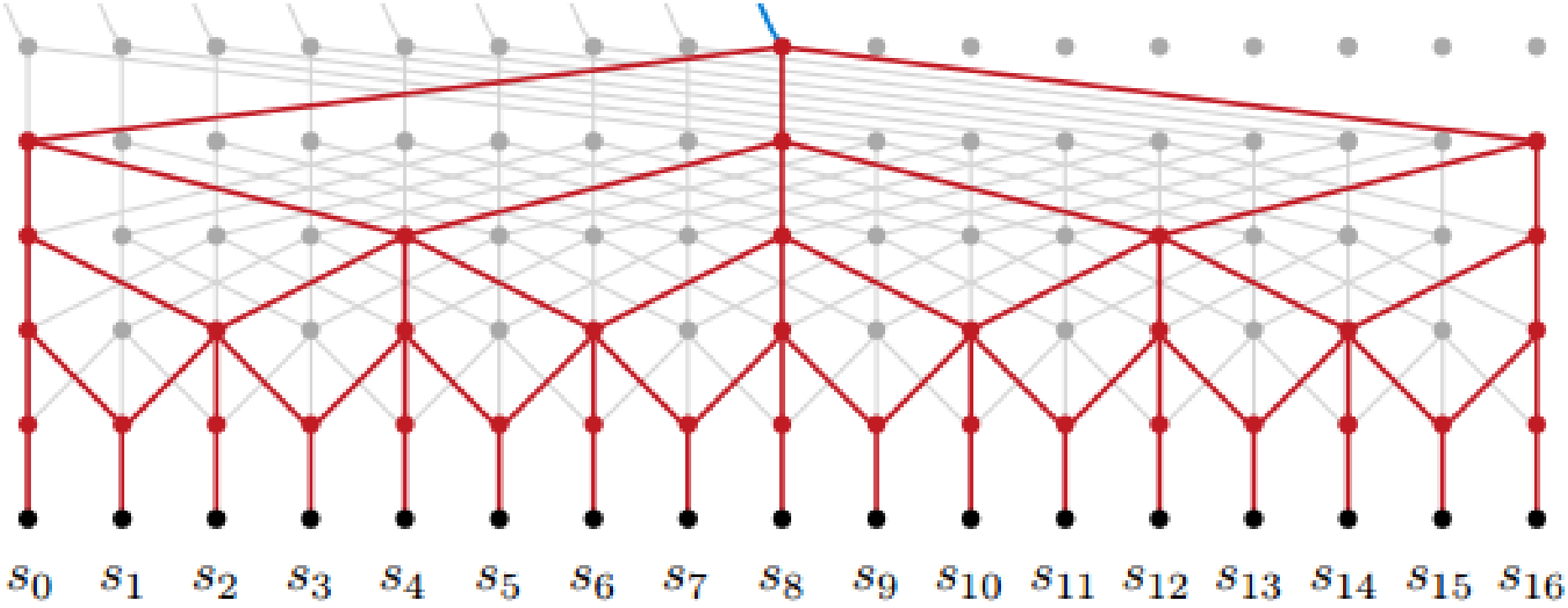


output = f(Weights × input + bias)



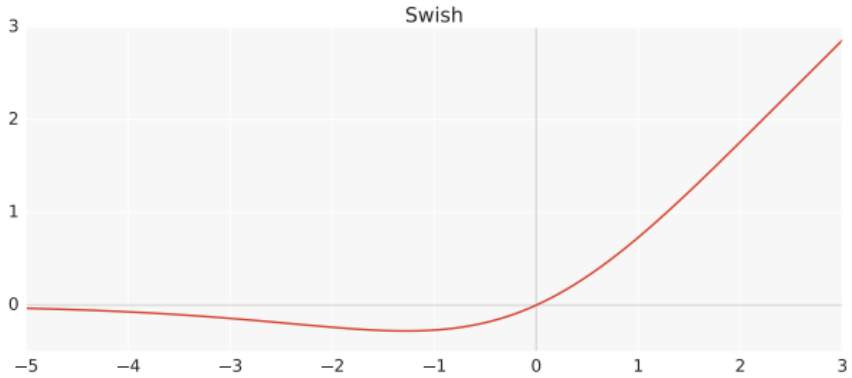
Convolutional Layers

* apply mask
+ pooling (max, mean,…)

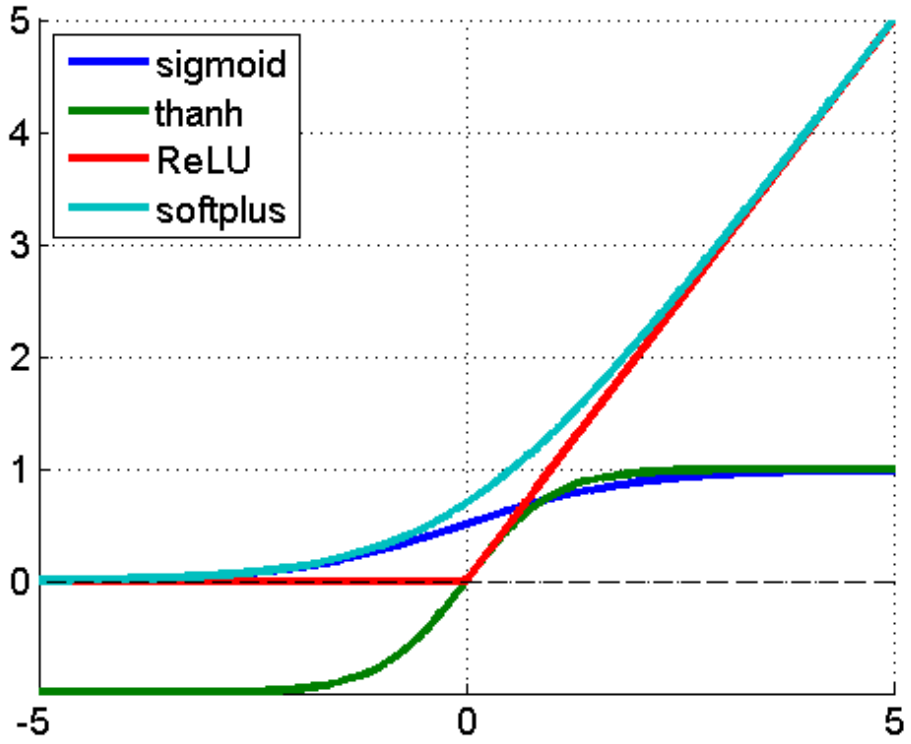


Nonlinearities

* sigmoid/logistic
* tanh
* ReLU/SeLU/ELU/leakyReLU/…
* softplus
* swish

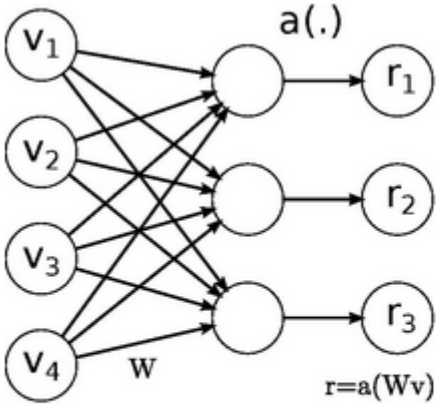


* maxout

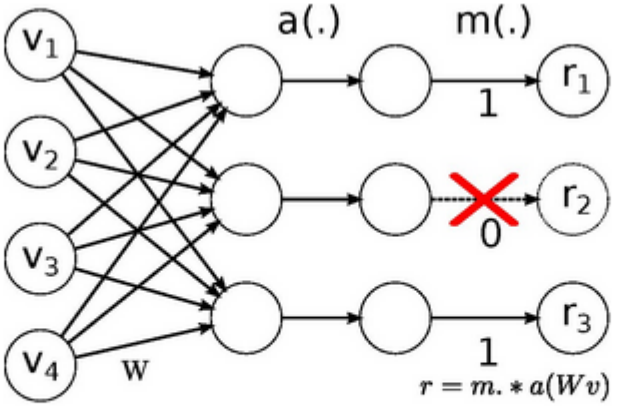


Regularization Layers

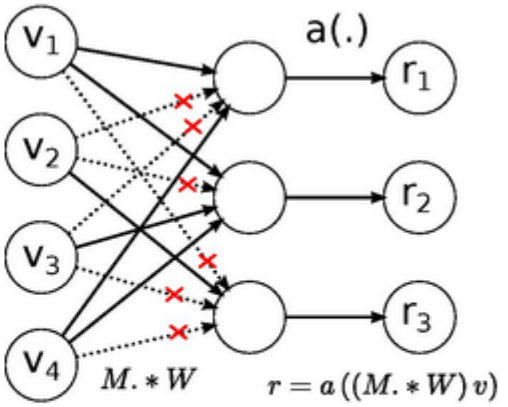
* nonlinearity regularization
* dropout
* dropconnect



No-Drop Network



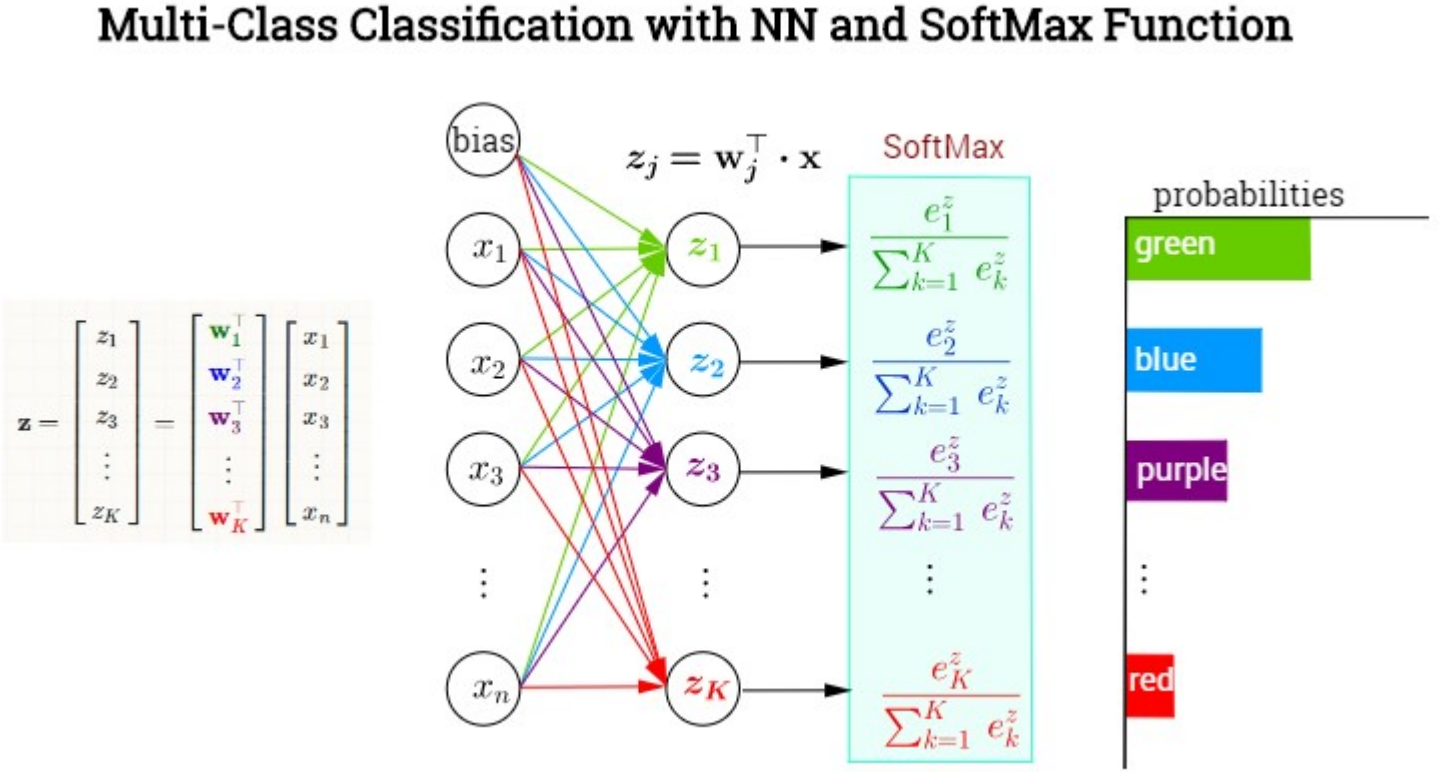
DropOut Network



DropConnect Network

Output Layers

* softmax/hierarchical softmax



Backprop

* efficient way to compute derivatives (using DP)
* automatic & symbolic differentiation

https://colah.github.io/posts/2015-08-Backprop/

Optimization Algorithm

* gradient descent
* SGD (+minibatch)
* Momentum
* Adagrad/Adadelta/...
* Adam

Example: FNN for Adjective Ordering

Input: noun & 2 adjectives
4 ReLU FC-layes
Sigmoid output

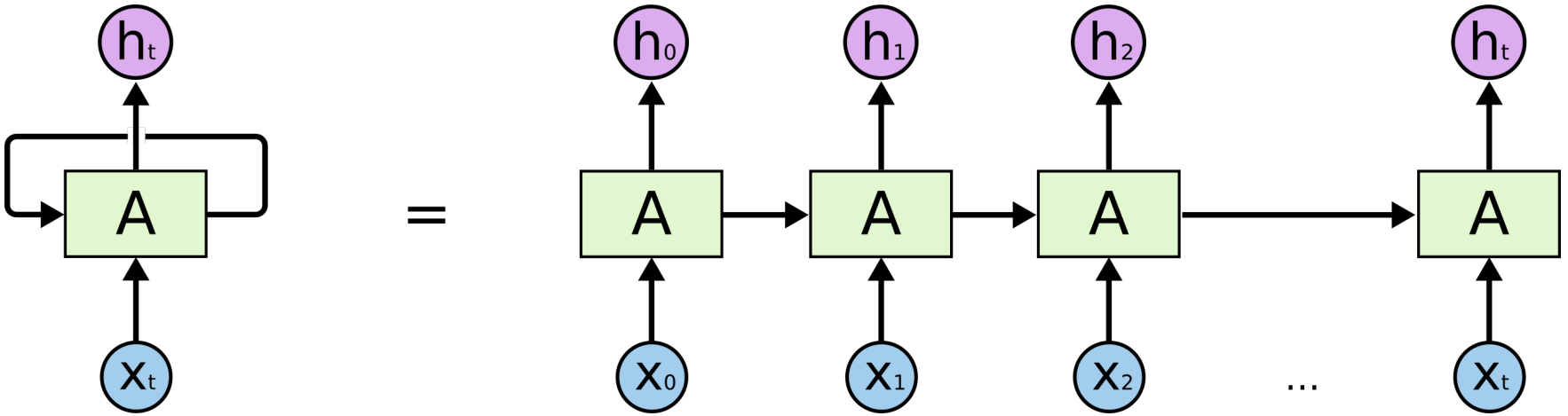
Run with both variants of order and select the better score

FNNs Recap

+ nonlinear, flexible
+ efficient training
+ allows to use embeddings
- fixed input
- limited context

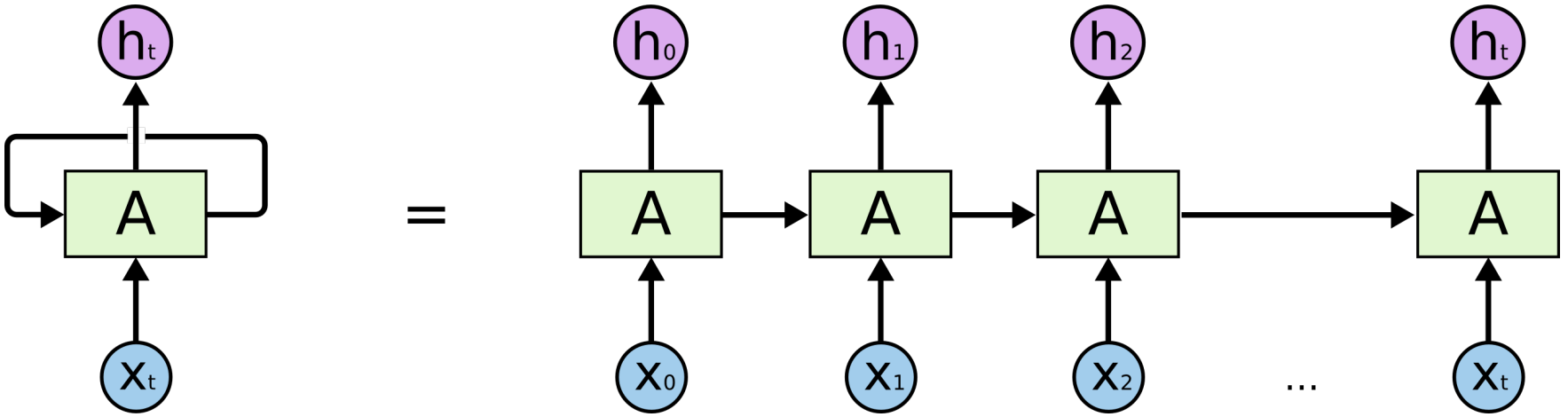
RNNs to the Rescue

* add previous state to input
* backpropagate through time



RNNs to the Rescue

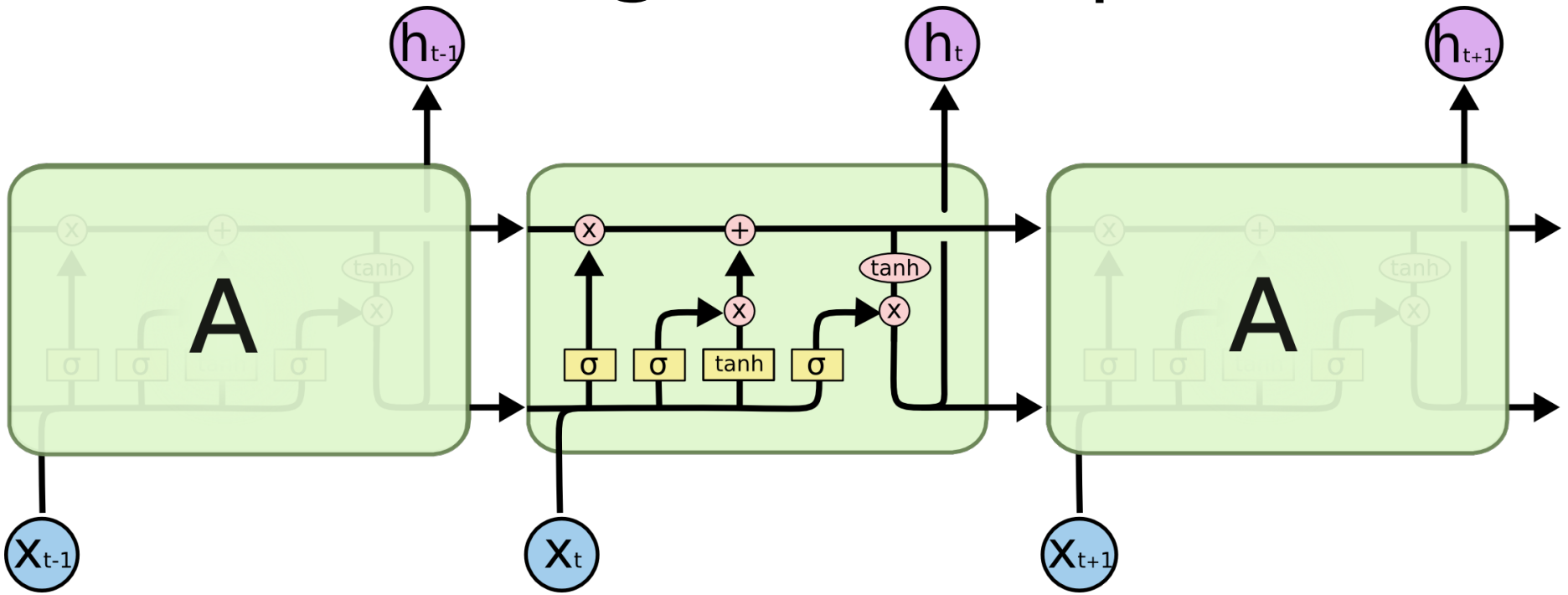
* add previous state to input
* backpropagate through time



* not so easy:
  - vanishing gradients
  - exploding gradients

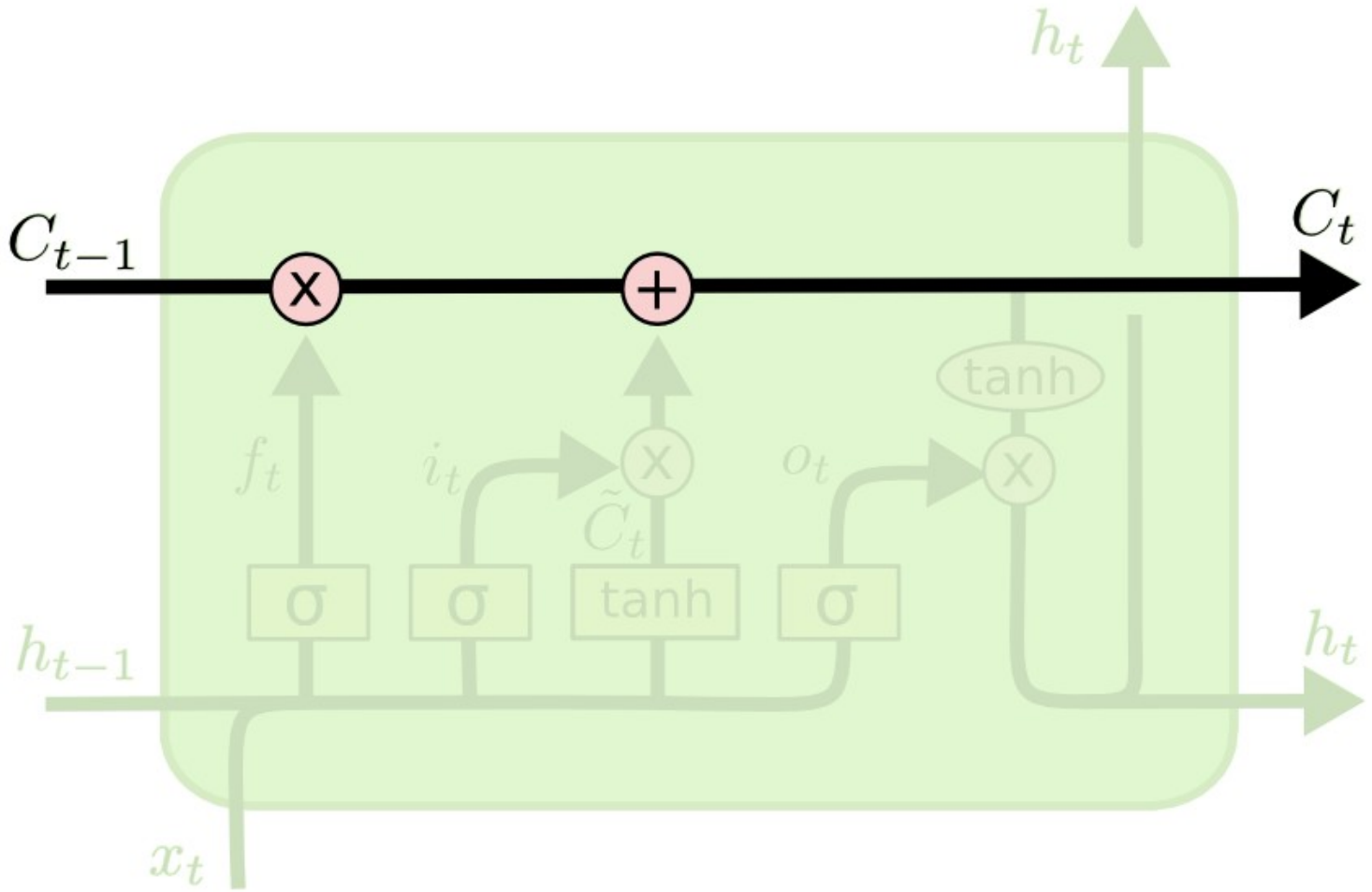
LSTM

specifically designed to remember long-term dependencies

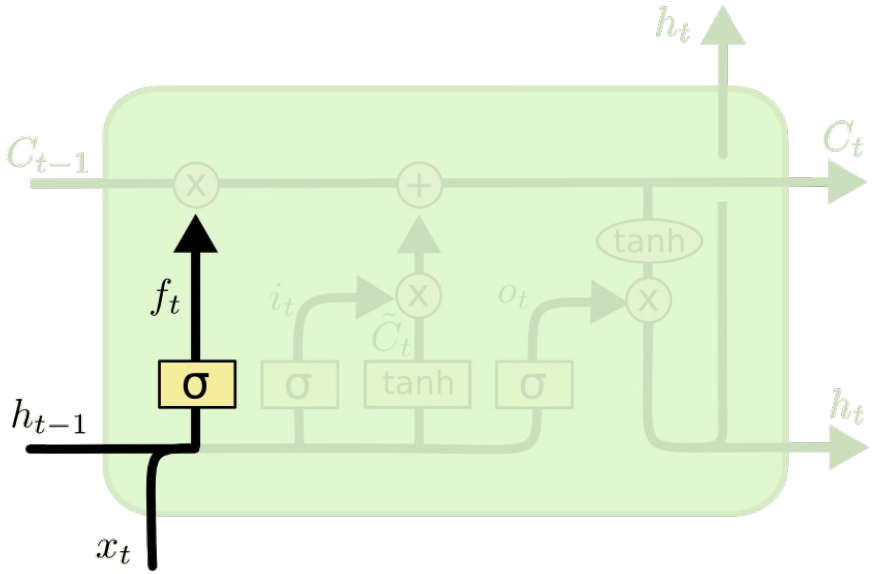


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Cell State

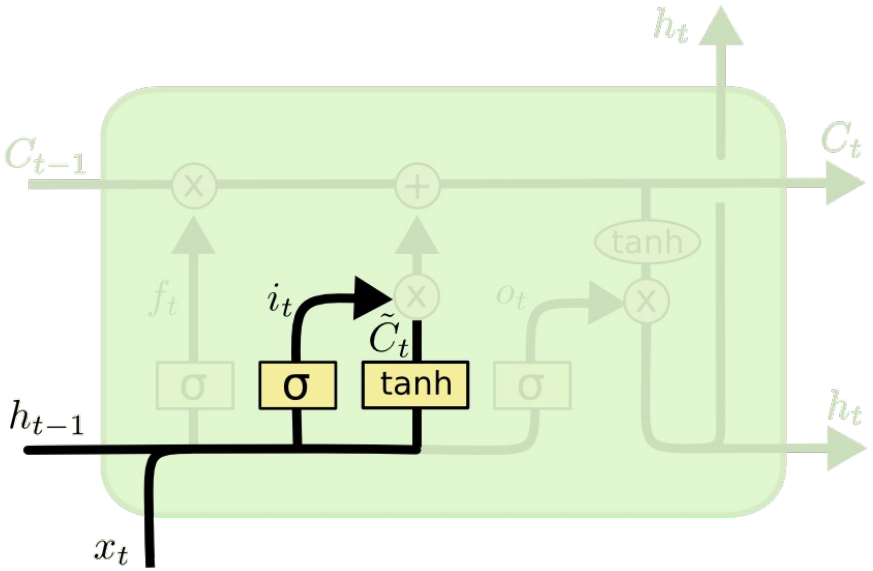


LSTM Forget Gate



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

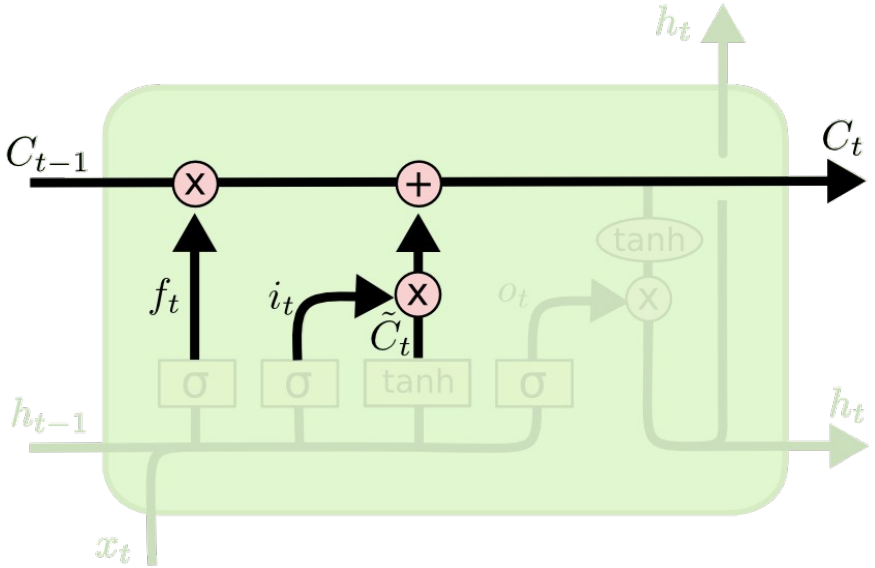
LSTM Remember Gate



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

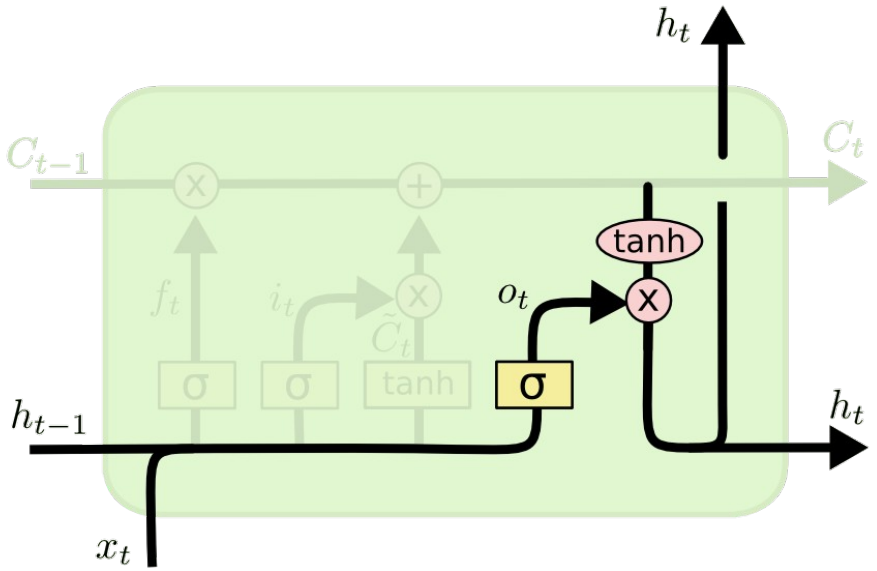
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM State Update



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

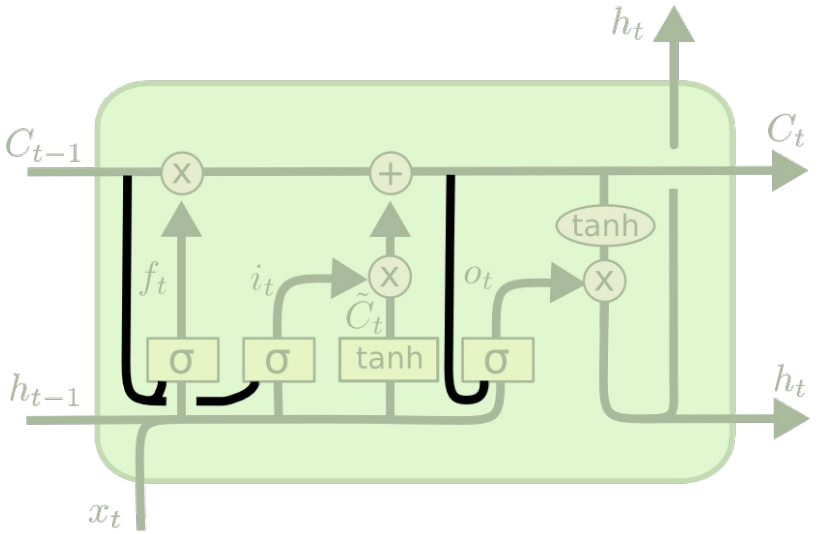
LSTM Output



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

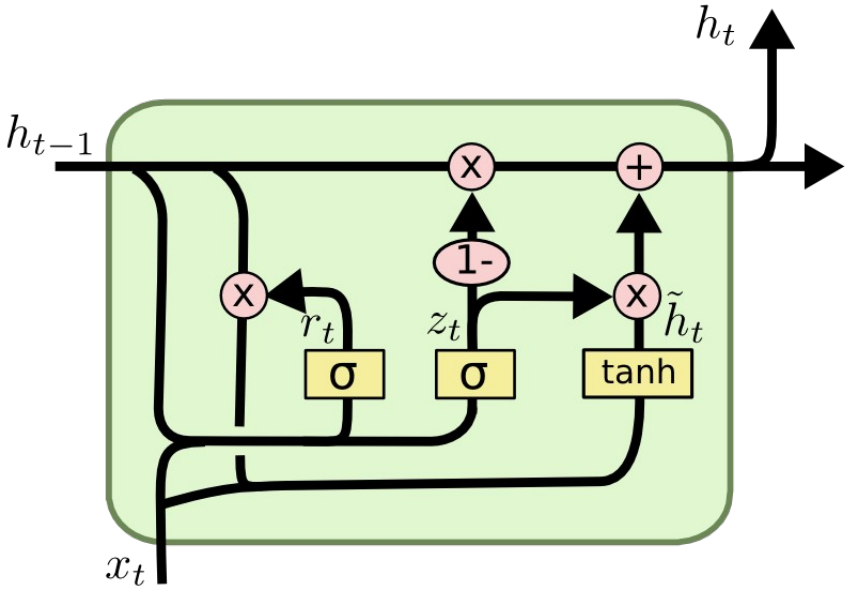
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM Peephole Connections



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Gated Recurrent Unit (GRU)



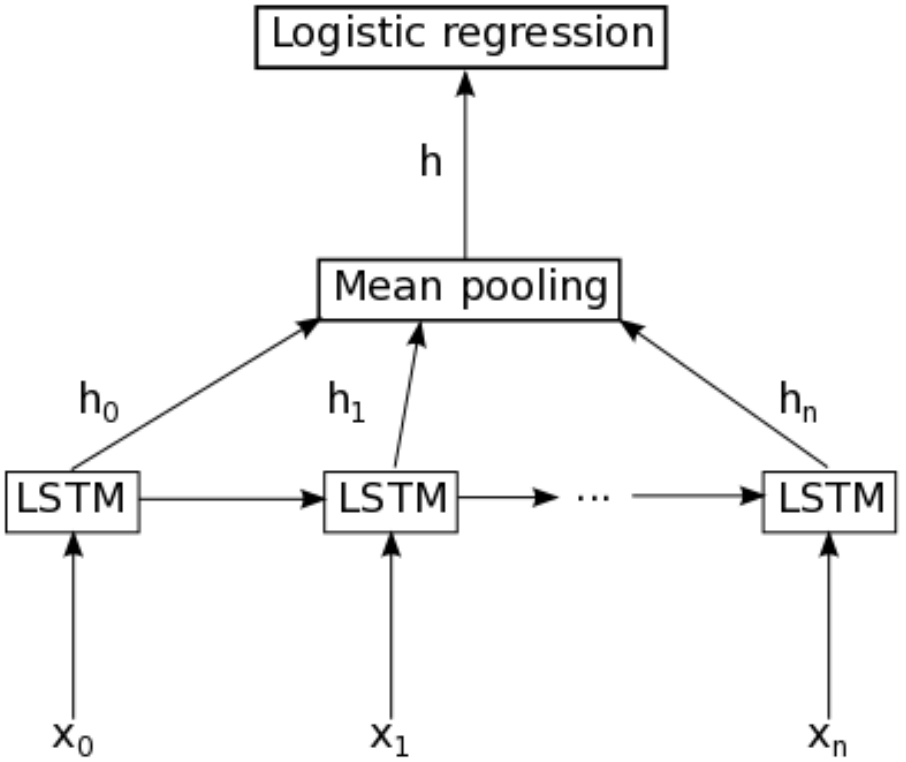
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

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

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

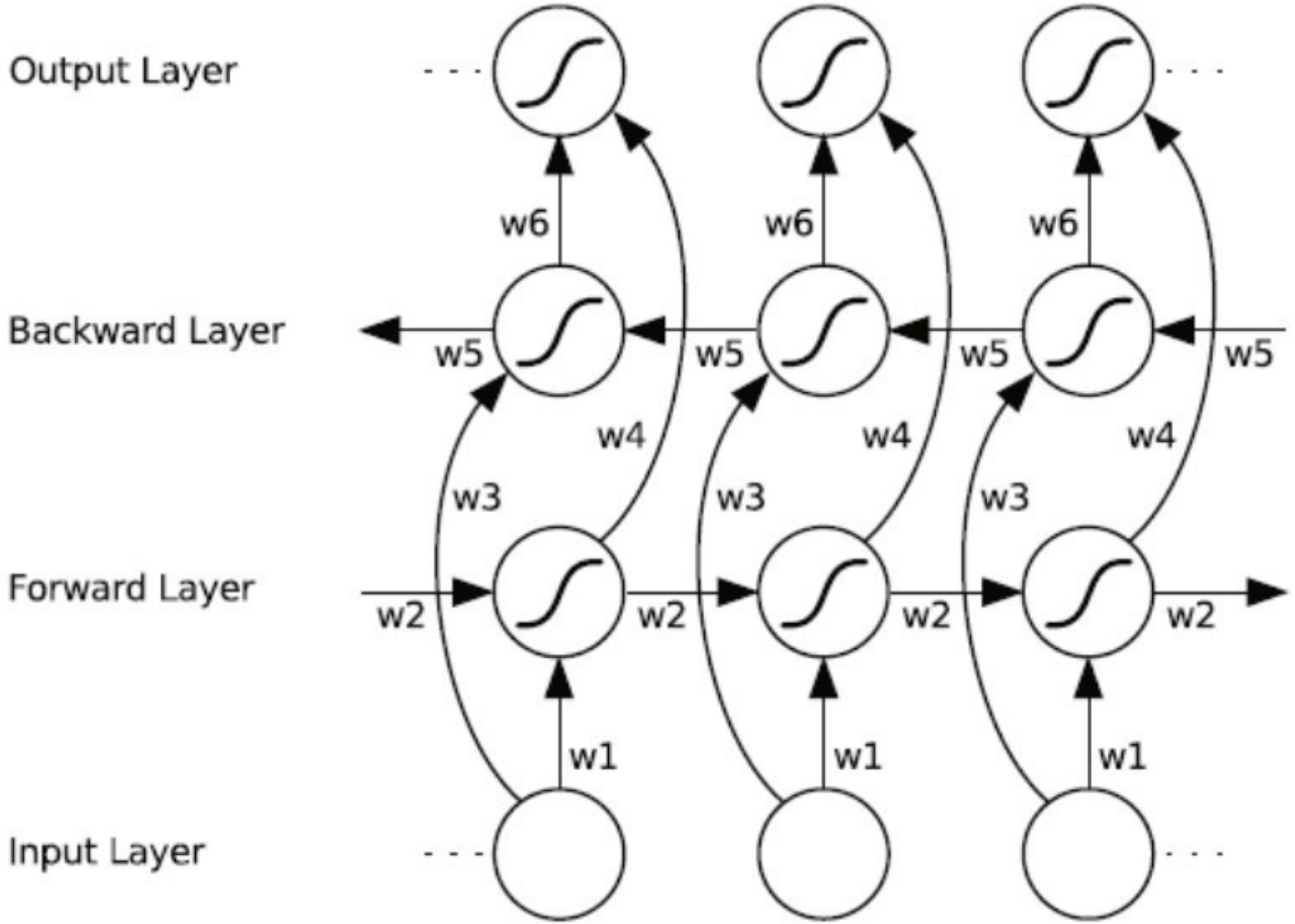
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM Example: Sentiment Analysis

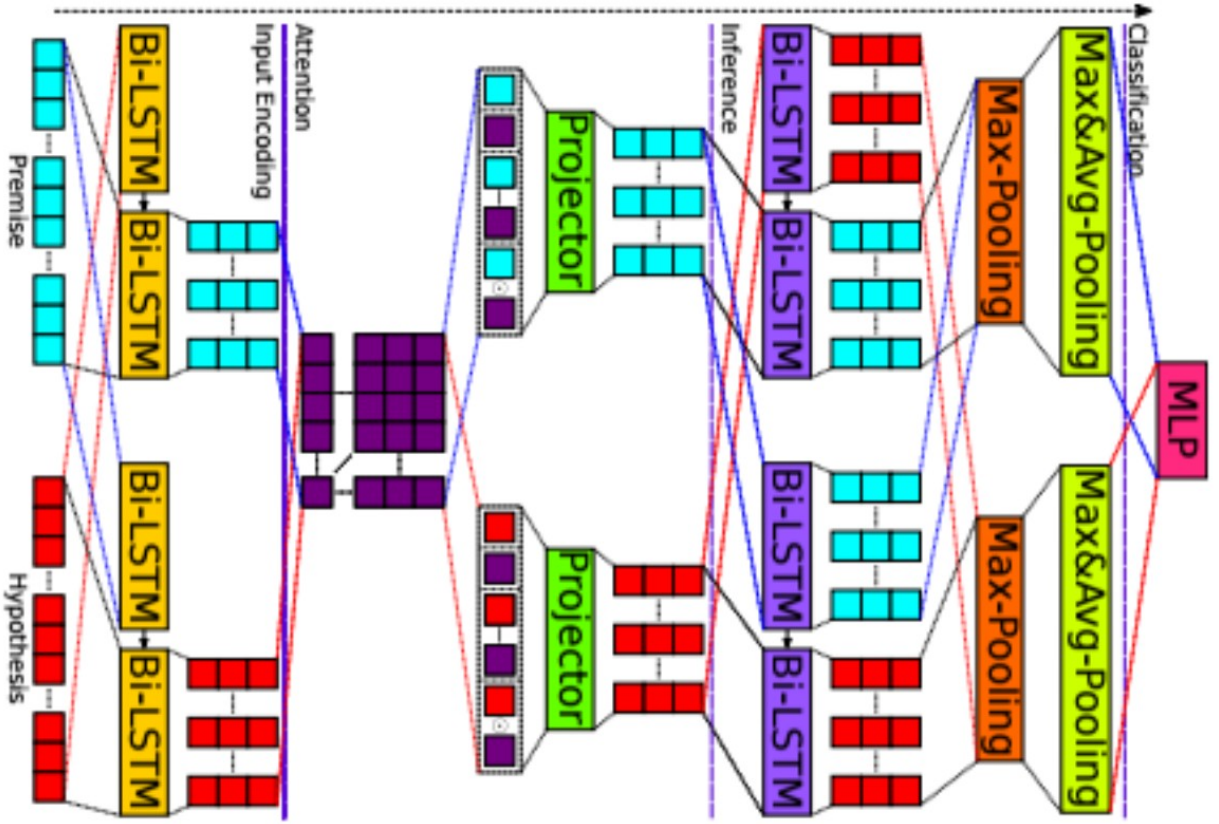


http://deeplearning.net/tutorial/lstm.html

BiLSTM


Output Layer
Backward Layer
Forward Layer
Input Layer
w1
w2
w3
w4
w5
w6

BiLSTM Example: Machine Reading

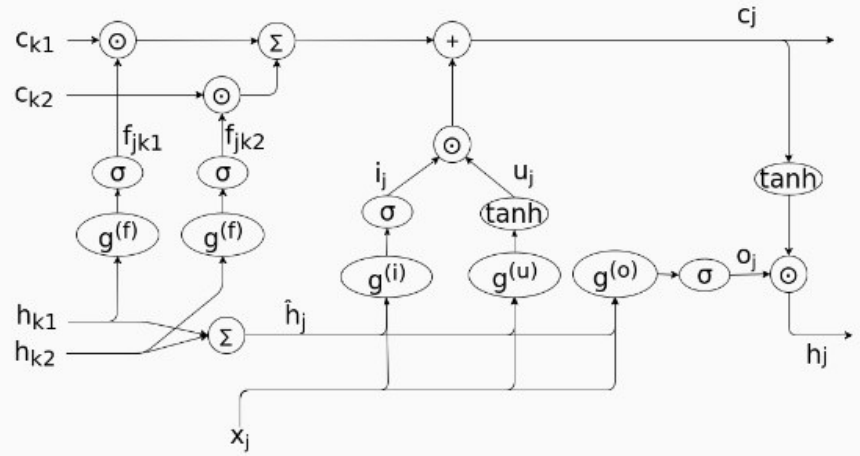


https://arxiv.org/pdf/1802.05577.pdf

TreeLSTM

Child-sum tree LSTM

Children outputs and memory cells are summed

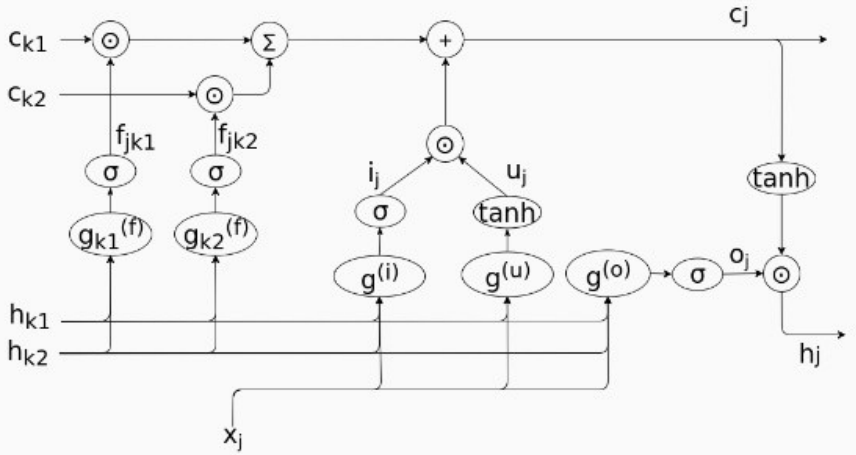


Child-sum tree LSTM at node j with children k_1 and k_2

10

N-ary tree LSTM

Given $g_k^{(n)}(x_t, h_{l_1}, \cdots, h_{l_N}) = W^{(n)}x_t + \sum_{l=1}^{N} U_{kl}^{(n)} h_{jl} + b^{(n)}$



Binary tree LSTM at node j with children k_1 and k_2

12

https://www.slideshare.net/tuvistavie/tree-lstm

TreeLSTM Example

Semantic relatedness

Task

Predict similarity score in $[1, K]$ between two sentences

Method

Similarity between sentences L and R annotated with score $\in [1, 5]$

- Produce representations h_L and h_R
- Compute distance h_+ and angle $h_\times$ between h_L and h_R
- Compute score using fully connected NN

$$h_s = \sigma\left(W^{(\times)} h_\times + W^{(+)} h_+ + b^{(h)}\right)$$

$$\hat{p}_\theta = \text{softmax}\left(W^{(p)} h_s + b^{(p)}\right)$$

$$\hat{y} = r^T \hat{p}_\theta \qquad r = [1, 2, 3, 4, 5]$$

- Error is computed using KL-divergence

LSTM Deficiencies

* computation not parallelizable
* neuron interpretability

Improvements/alternatives:
* RAN (Recurrent Additive Network)
  https://arxiv.org/pdf/1705.07393.pdf
* Janet (just the forget gate)
  https://arxiv.org/pdf/1804.04849.pdf
* CNN (Convolutional Neural Network)
* QRNN (Quasi-recurrent Neural Network)
  https://goo.gl/NUx7VC

BiLSTM+attention as SOTA

“Basically, if you want to do an NLP task, no matter what it is, what you should do is throw your data into a Bi-directional long-short term memory network, and augment its information flow with the attention mechanism.”
— Chris Manning

https://twitter.com/mayurbhangale/status/988332845708886016

Read More

Neural Nets for NLP:
http://cs231n.github.io
https://hackernoon.com/the-unreasonable-ineffectiveness-of-deep-learning-in-nlu-e4b4ce3a0da0
https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Nonlinearities:
https://towardsdatascience.com/selu-make-fnns-great-again-snn-8d61526802a9
https://medium.com/@jaiyamsharma/experiments-with-swish-activation-function-on-mnist-dataset-fc89a8c79ff7
https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464
http://building-babylon.net/2017/08/01/hierarchical-softmax/

Read More x2

Backprop & gradient descent:
https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b
http://ruder.io/optimizing-gradient-descent/
https://openreview.net/pdf?id=ryQu7f-RZ
https://fosterelli.co/executing-gradient-descent-on-the-earth

RNN & LSTM:
https://deeplearning4j.org/lstm.html
http://karpathy.github.io/2015/05/21/rnn-effectiveness/
https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9