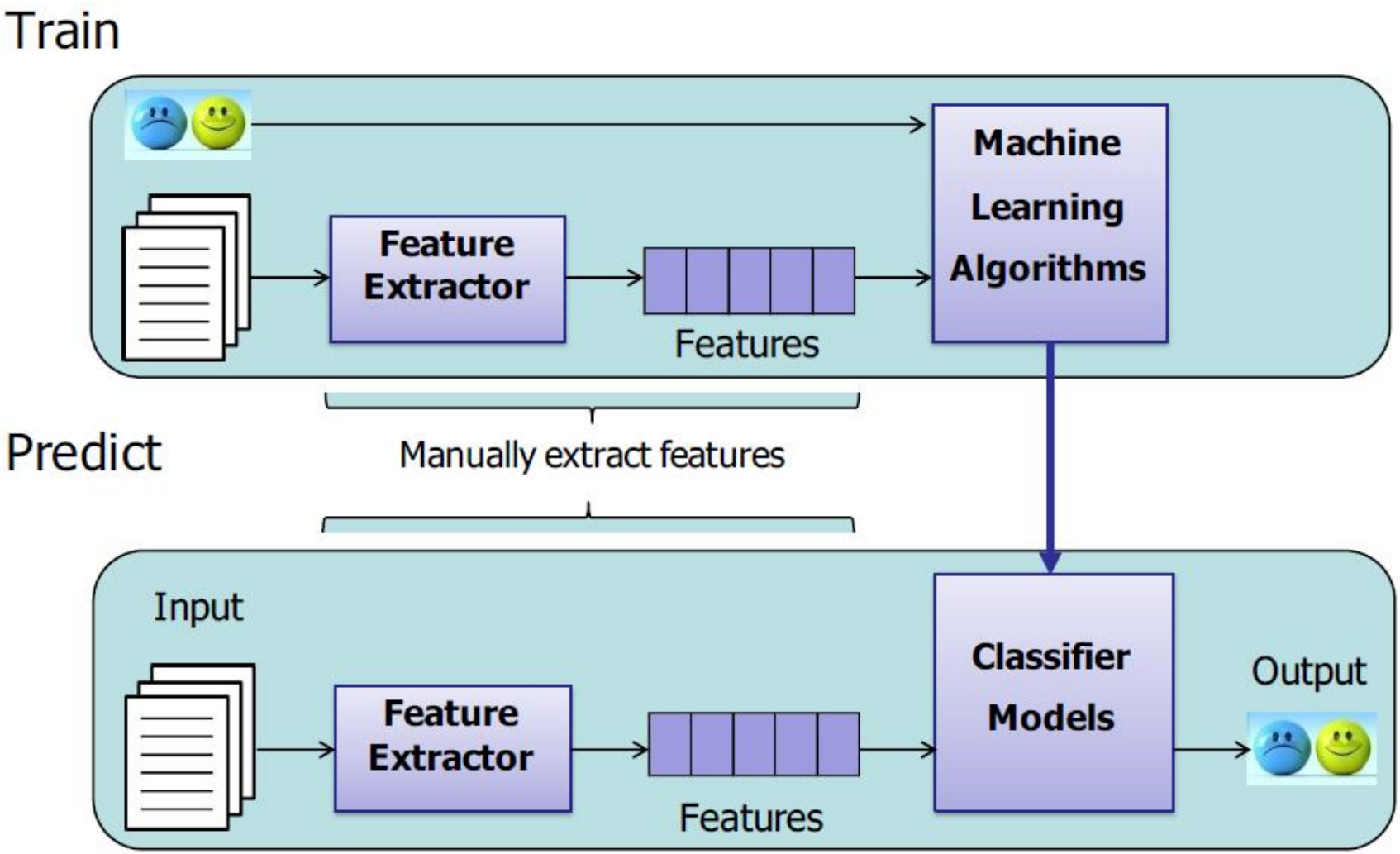
# Natural Language Processing

Text Classification

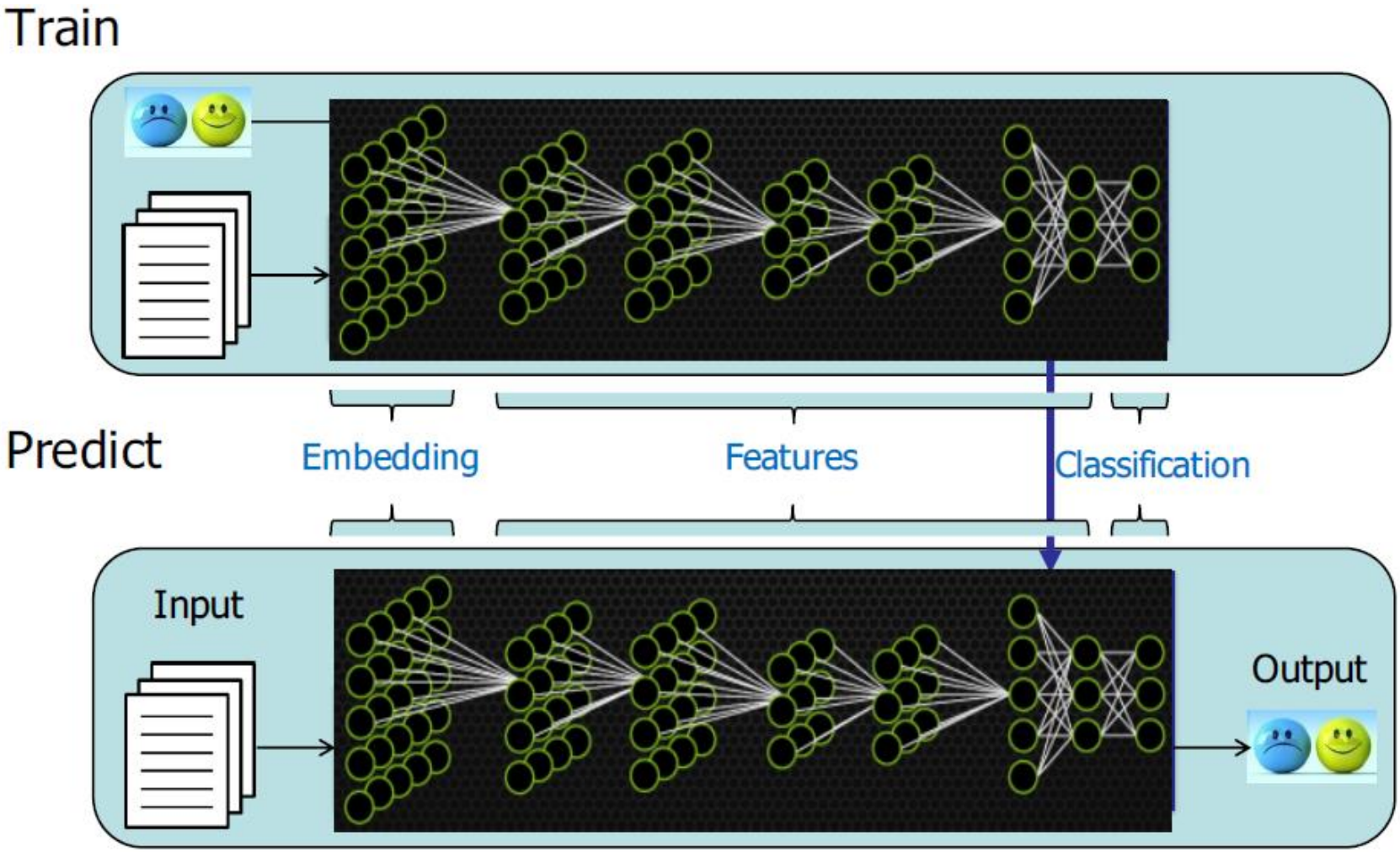
Instructor: Yangqiu Song

Slides credits: Yue Zhang, Dan Roth

# A General Pipeline



# Deep Learning



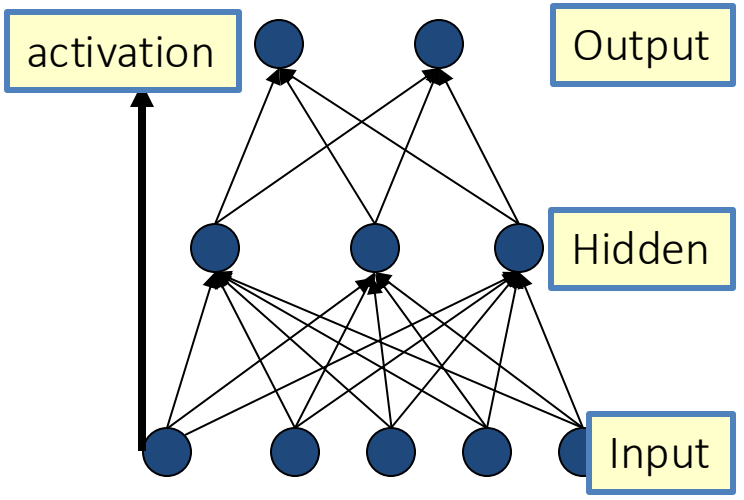
# Neural Networks

- Neural Networks are **functions**: $\mathrm{NN}: X \to Y$
  - where $X = [0,1]^n$, *or* $\{0,1\}^n$ *and* $Y = [0,1]$, $\{0,1\}$ (or $\{-1,1\}$)

- NN can be used as an approximation of a target classifier
  - In their general form, even with a single hidden layer, NN can approximate any function
  - Algorithms exist that can learn a NN representation from labeled training data (e.g., Backpropagation).

# Multi-Layer Neural Networks

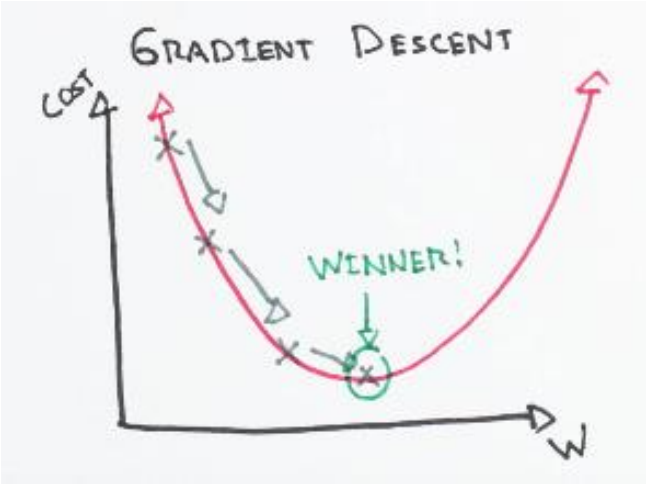
- Multi-layer network were designed to overcome the computational (**expressivity**) limitation of a single threshold element.
- The idea is to **stack** several layers of threshold elements, each layer using the output of the previous layer as input.



# Gradient Descent

- Goal: optimize parameters to minimize loss $\min_{\theta} f(\theta)$
- Step along the direction of steepest descent (negative gradient)

$$\theta_t = \theta_{t-1} - \alpha \nabla_{\theta_{t-1}} f(\theta_{t-1})$$



# Deep Learning for Text Classification

Building deep learning models on textual data requires:

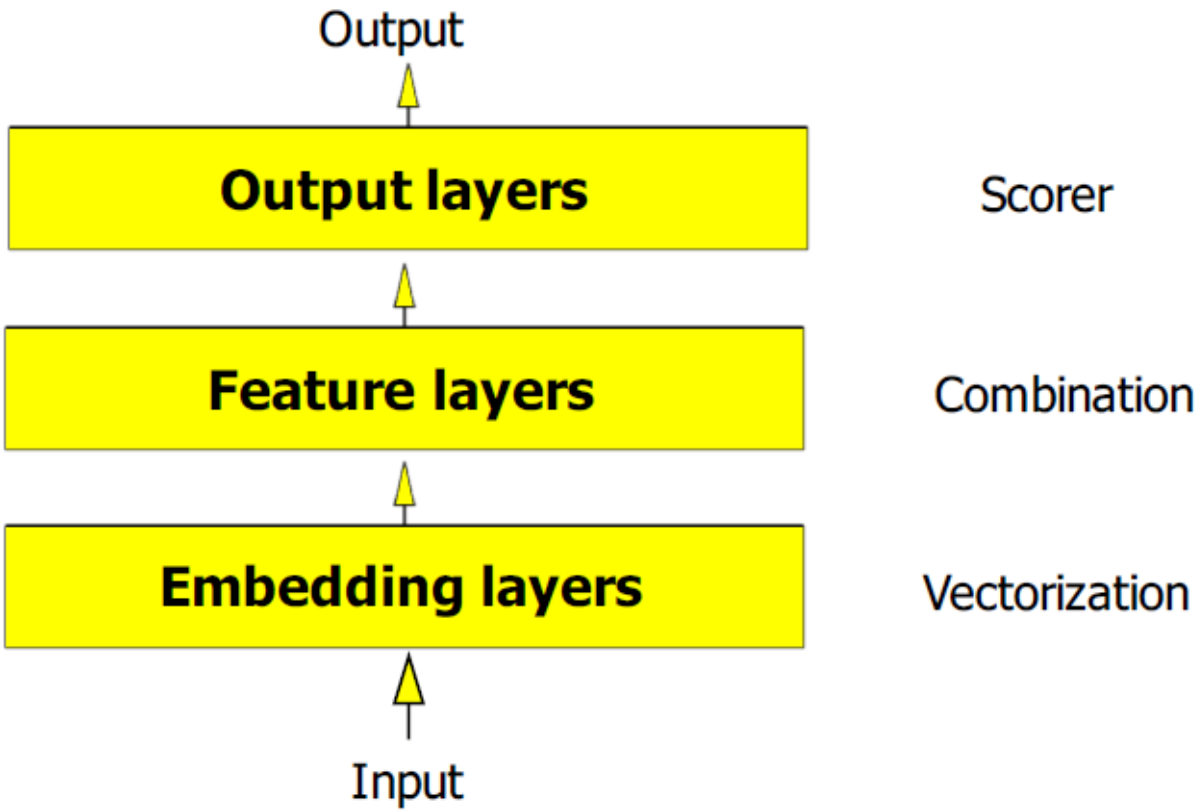
- Representation of the basic text unit, word.
- Neural network structure that can capture the sequential nature of text.

Deep learning models use:

- Vector representation of words (i.e., word embeddings)
- Neural network structures
  - Convolutional Neural networks
  - Recurrent Neural Network
  - Recursive Neural Network

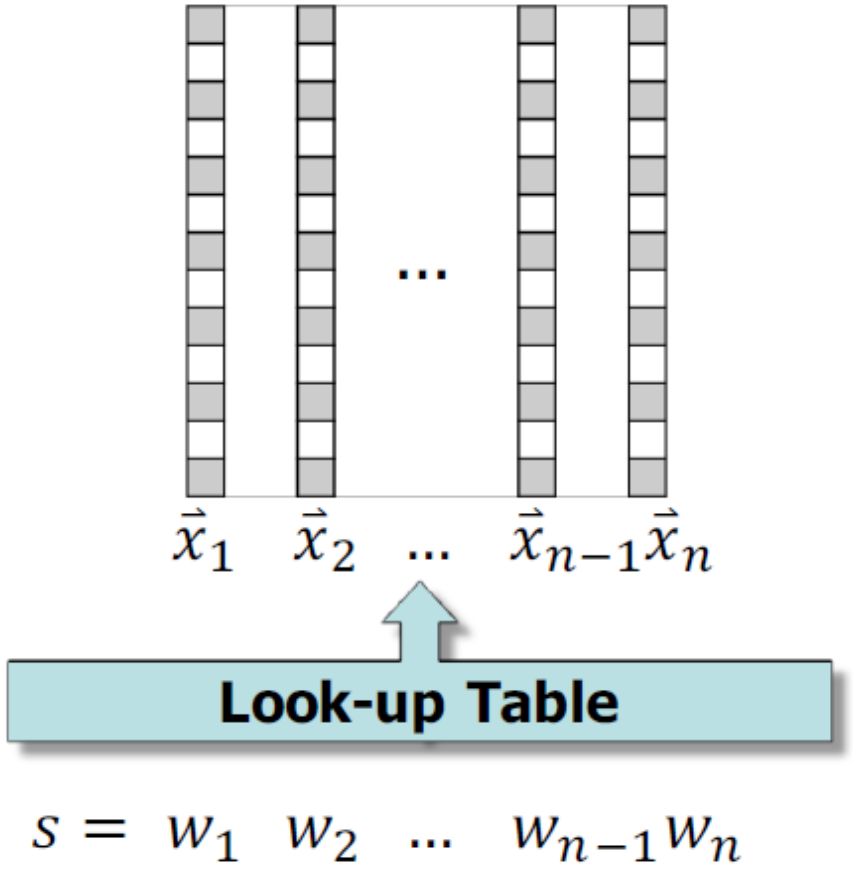
# Overview

- General model:



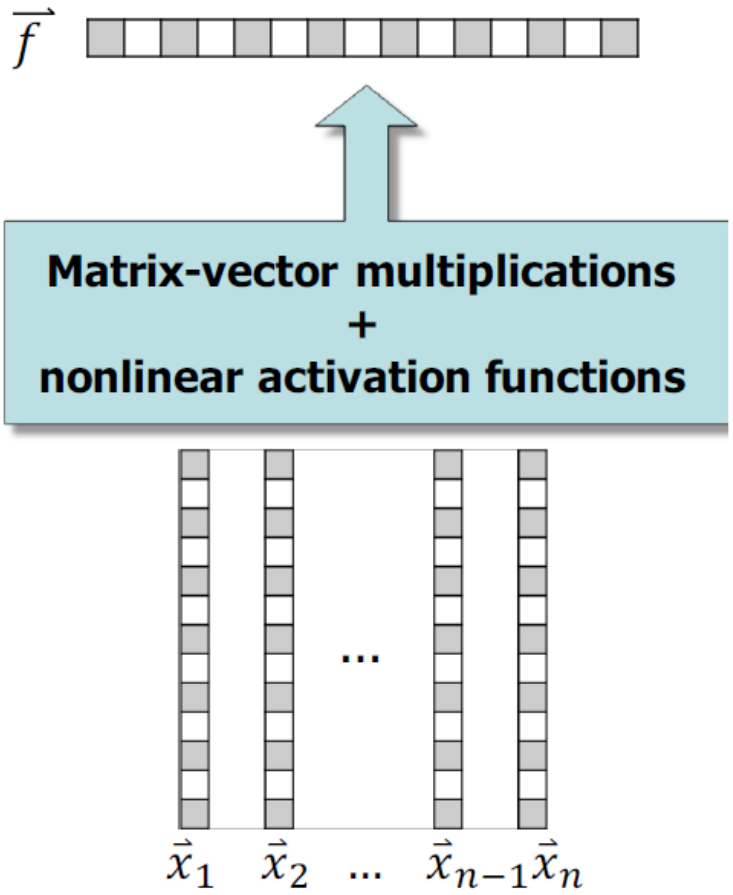
# Overview

- Embedding Layer
  - Word to vector
  - Look up table



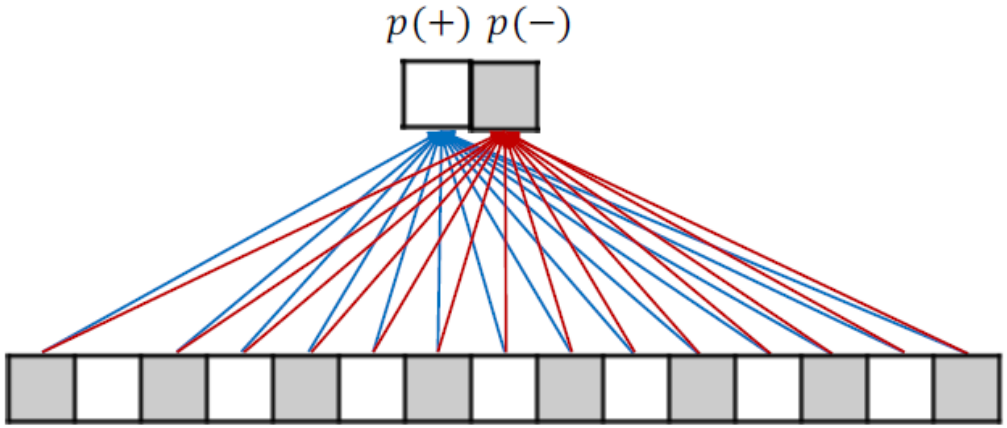
# Overview

- Feature Layer
  - Automatically learn the representation of inputs
  - Matrix-vector multiplication
  - Element-wise composition
  - Non-linear transformation



# Overview

- Output Layer
  - Softmax
  - Cross-entropy loss



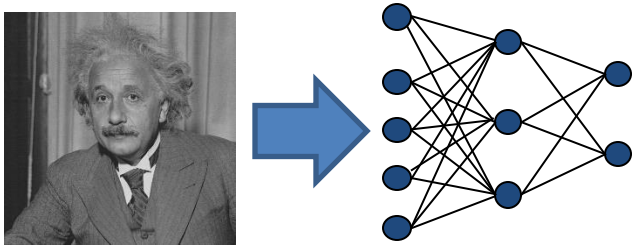
# Typical Feature Layers

- CNN
- RNN

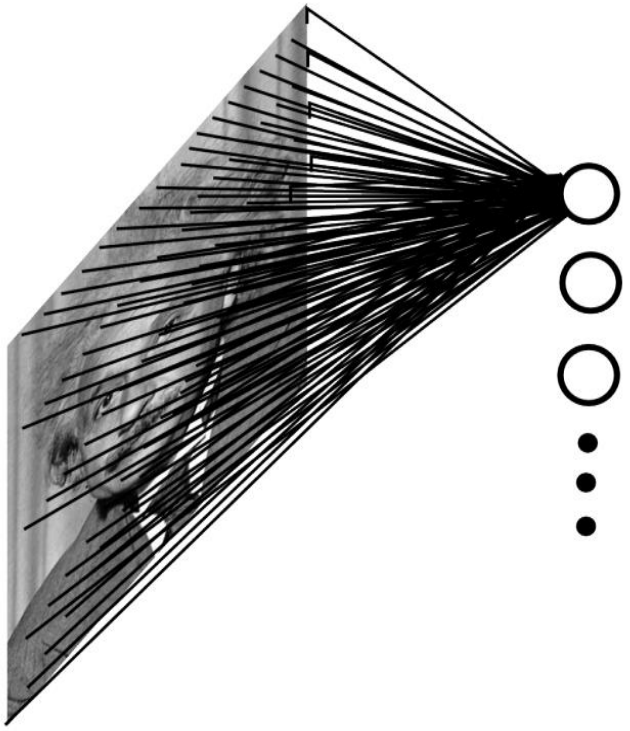
# Receptive Fields

- The **receptive field** of an individual sensory neuron is the particular region of the sensory space (e.g., the body surface, or the retina) in which a stimulus will trigger the firing of that neuron.
  - Designing "proper" receptive fields for the input Neurons is a significant challenge.

- Consider a task with image inputs
  - Receptive fields should give expressive features from the raw input to the system
  - How would you design the receptive fields for this problem?

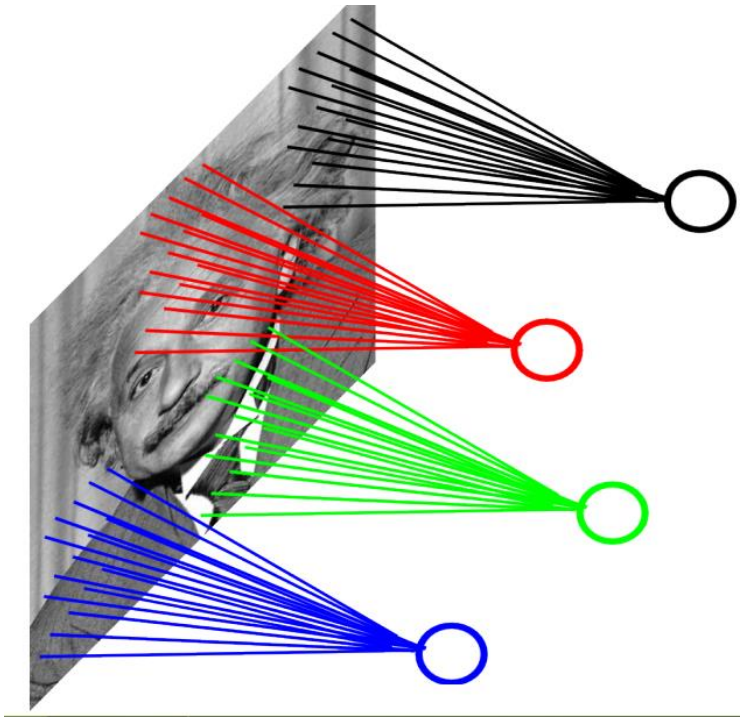


- A **fully connected layer**:
  - Example:
    - 100x100 images
    - 1000 units in the hidden
  - Problems:
    - 10^7 edges!
    - Spatial correlations lost!
    - Variables sized inputs.



Slide Credit: Marc'Aurelio Ranzato

- Consider a task with image inputs:
- A **locally connected layer**:
  - Example:
    - 100x100 images
    - 1000 units in the hidden
    - Filter size: 10x10
  - Local correlations preserved!
  - Problems:
    - 10^5 edges
    - This parameterization is good when input image is registered (e.g., face recognition).
    - Variable sized inputs, again.

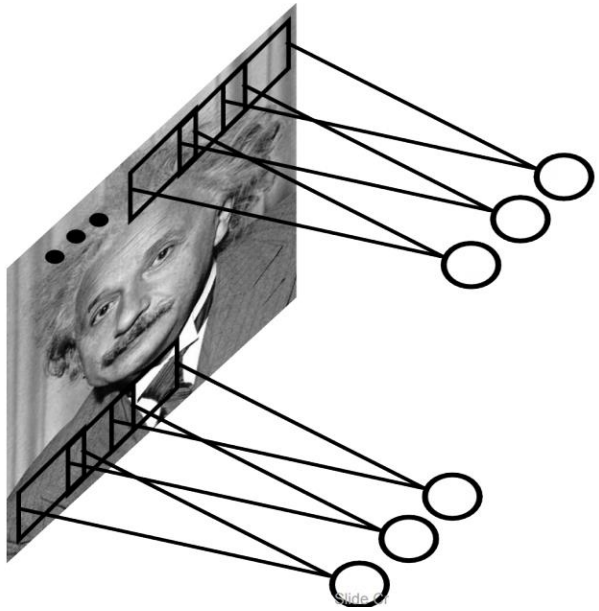


Slide Credit: Marc'Aurelio Ranzato

# Convolutional Layer

- A solution:
  - **Filters** to capture different patterns in the input space.
    - **Share** parameters across different locations (assuming input is stationary)
    - Convolutions with learned filters
  - Filters will be **learned** during training.
  - The issue of variable-sized inputs will be resolved with a **pooling** layer.

So what is a convolution?



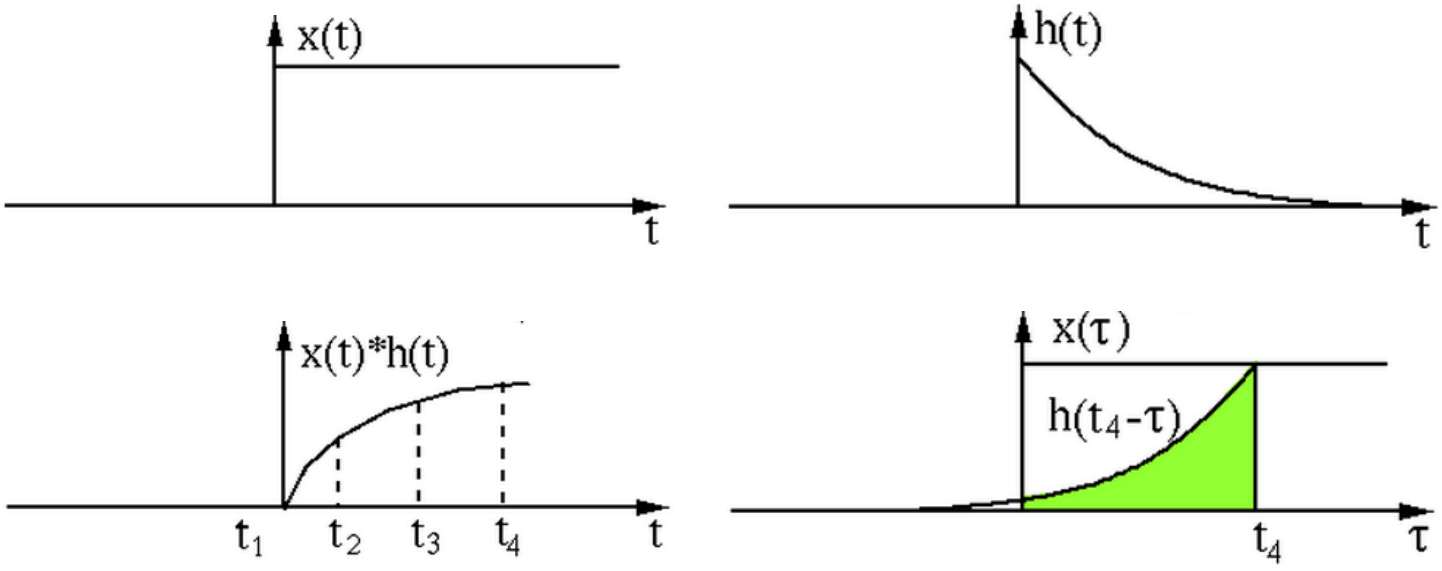
# Convolution Operator

- Convolution operator: $*$
  - takes two functions and gives another function
- One dimension:

$$(x * h)(t) = \int x(\tau)h(t-\tau)d\tau$$

$$(x * h)[n] = \sum_m x[m]h[n-m]$$

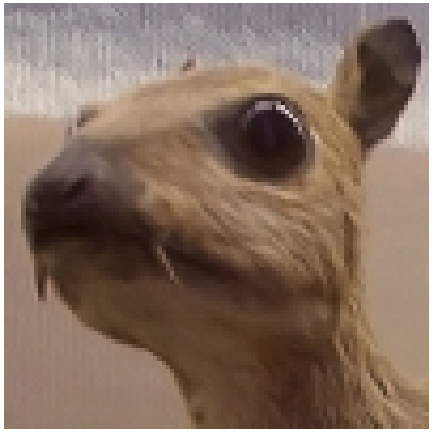
"Convolution" is very similar to "cross-correlation", except that in convolution one of the functions is flipped.



# Convolution Operator (2)

- Convolution in two dimension:
  - The same idea: flip one matrix and slide it on the other matrix
  - Example: edge detection kernel:

Input image



Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map

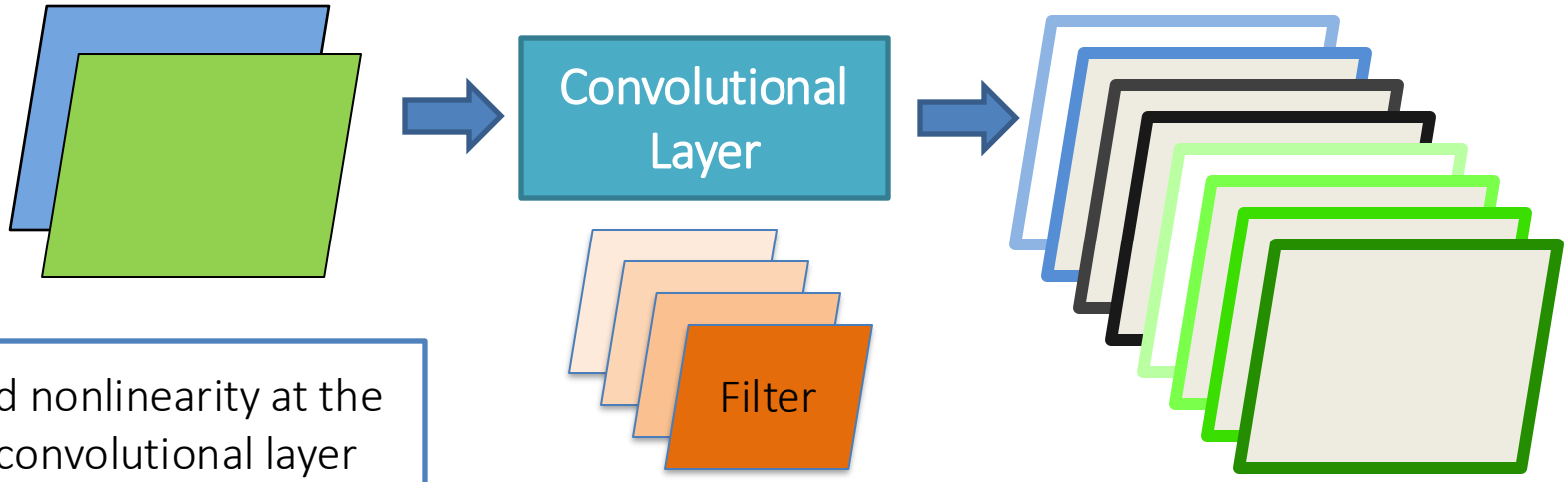
Try other kernels: http://setosa.io/ev/image-kernels/

# Demo of CNN

- https://setosa.io/ev/image-kernels/

# Convolutional Layer

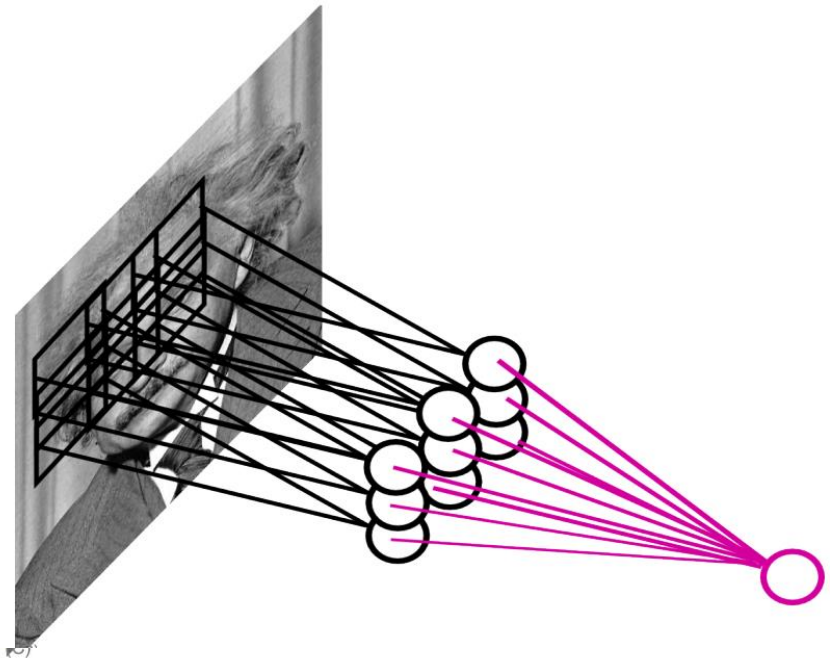
- The convolution of the **input (vector/matrix)** with weights **(vector/matrix)** results in a **response vector/matrix**.
- We can have **multiple filters** in each convolutional layer, each producing an output.
- If it is an intermediate layer, it can have **multiple inputs**!




One can add nonlinearity at the output of convolutional layer

# Pooling Layer

- How to handle variable sized inputs?
  - A layer which reduces inputs of different size, to a fixed size.
  - Pooling



Slide Credit: Marc'Aurelio Ranzato

# Pooling Layer

- How to handle variable sized inputs?
  - A layer which reduces inputs of different size, to a fixed size.
  - Pooling
  - Different variations
    - Max pooling

$$h_i[n] = \max_{i \in N(n)} \tilde{h}\,[i]$$

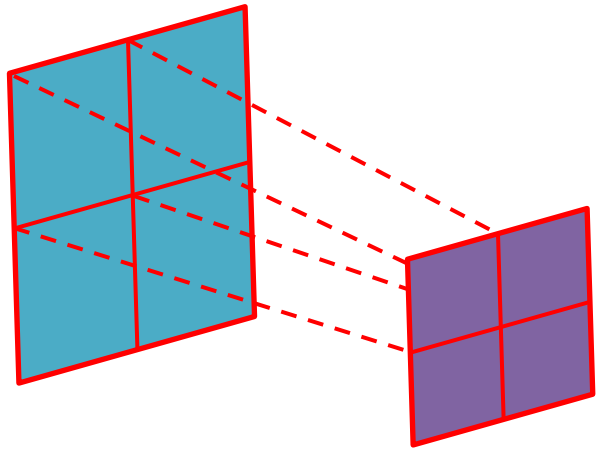
    - Average pooling

$$h_i[n] = \frac{1}{n} \sum_{i \in N(n)} \tilde{h}\,[i]$$

    - L2-pooling

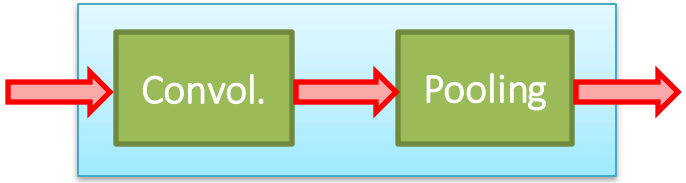
$$h_i[n] = \frac{1}{n} \sqrt{\sum_{i \in N(n)} \tilde{h}^2\,[i]}$$

    - etc

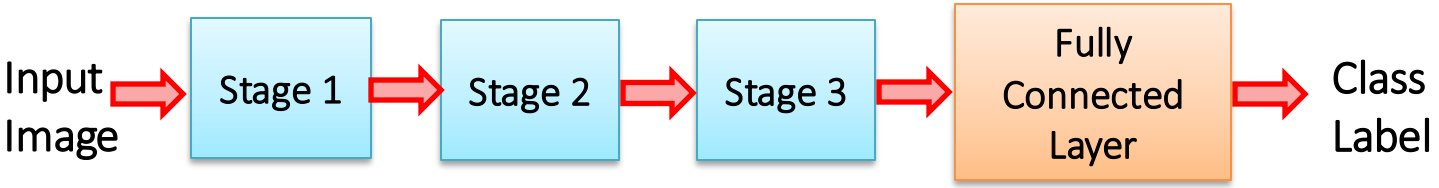


# Convolutional Nets

- One stage structure:

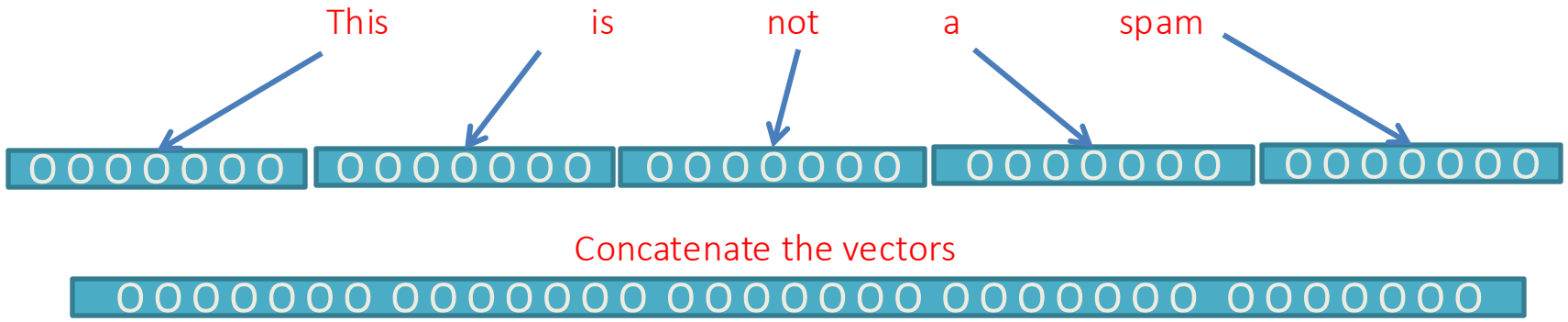



- Whole system:



# CNN for text (sequence) inputs

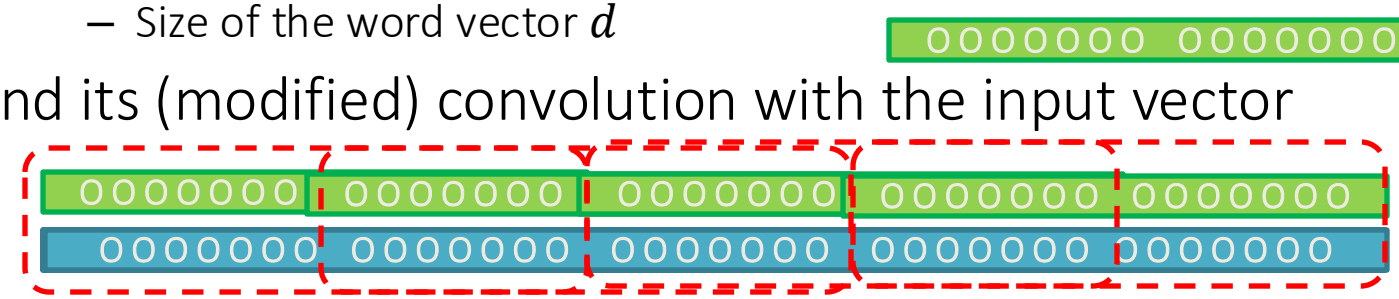
- Let's study the variant of CNN for language
  - Example: sentence classification (say spam or not spam)
- First step: represent each word with a vector in $\mathbb{R}^d$




- Now we can assume that the input to the system is a vector $\mathbb{R}^{dl}$
  - Where the input sentence has length $l$ ($l = 5$ in our example )
  - Each word vector's length $d$ ($d = 7$ in our example )

# Convolutional Layer on vectors

- Think about a single convolutional layer
  - A bunch of **vector** filters
    - Each defined in $\mathbb{R}^{dh}$
      - Where $h$ is the number of the words the filter covers
      - Size of the word vector $d$
  - Find its (modified) convolution with the input vector



$$c_1 = f(w.x_{1:hd}) \quad c_2 = f(w.x_{(d+1):(d+hd)}) \quad c_3 = f(w.x_{(2d+1):(2d+hd)}) \quad c_4 = f(w.x_{(3d+1):(3d+hd)})$$

  - Result of the convolution with the filter

$$c = [c_1, \ldots, c_{n-h+1}]$$

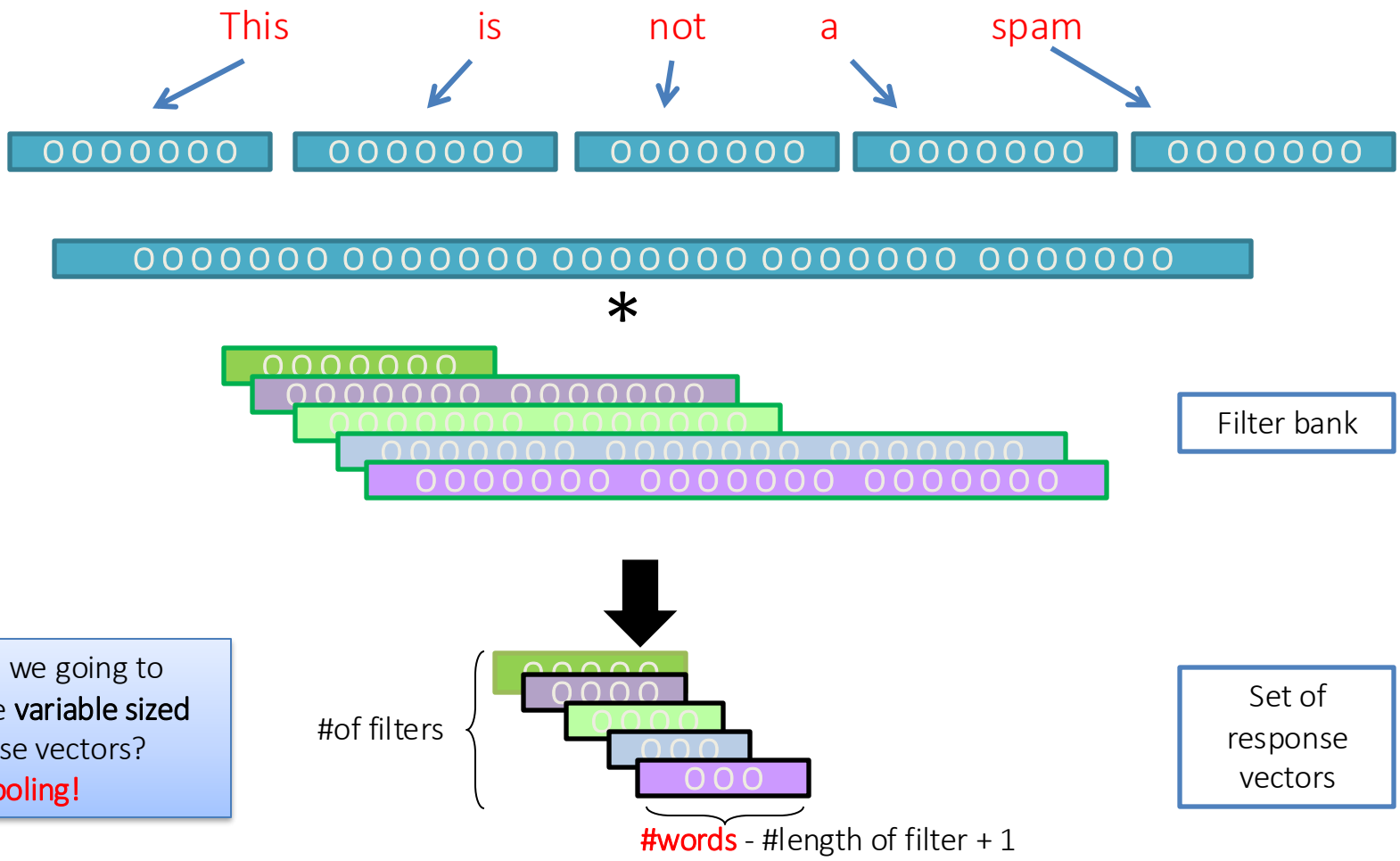
  - Convolution with a filter that spans 2 words, is operating on all of the bi-grams (vectors of two consecutive word, concatenated): “this is”, “is not”, “not a”, “a spam”.
  - Regardless of whether it is grammatical (not appealing linguistically)

# Convolutional Layer on vectors

Get word vectors for each words

Concatenate vectors

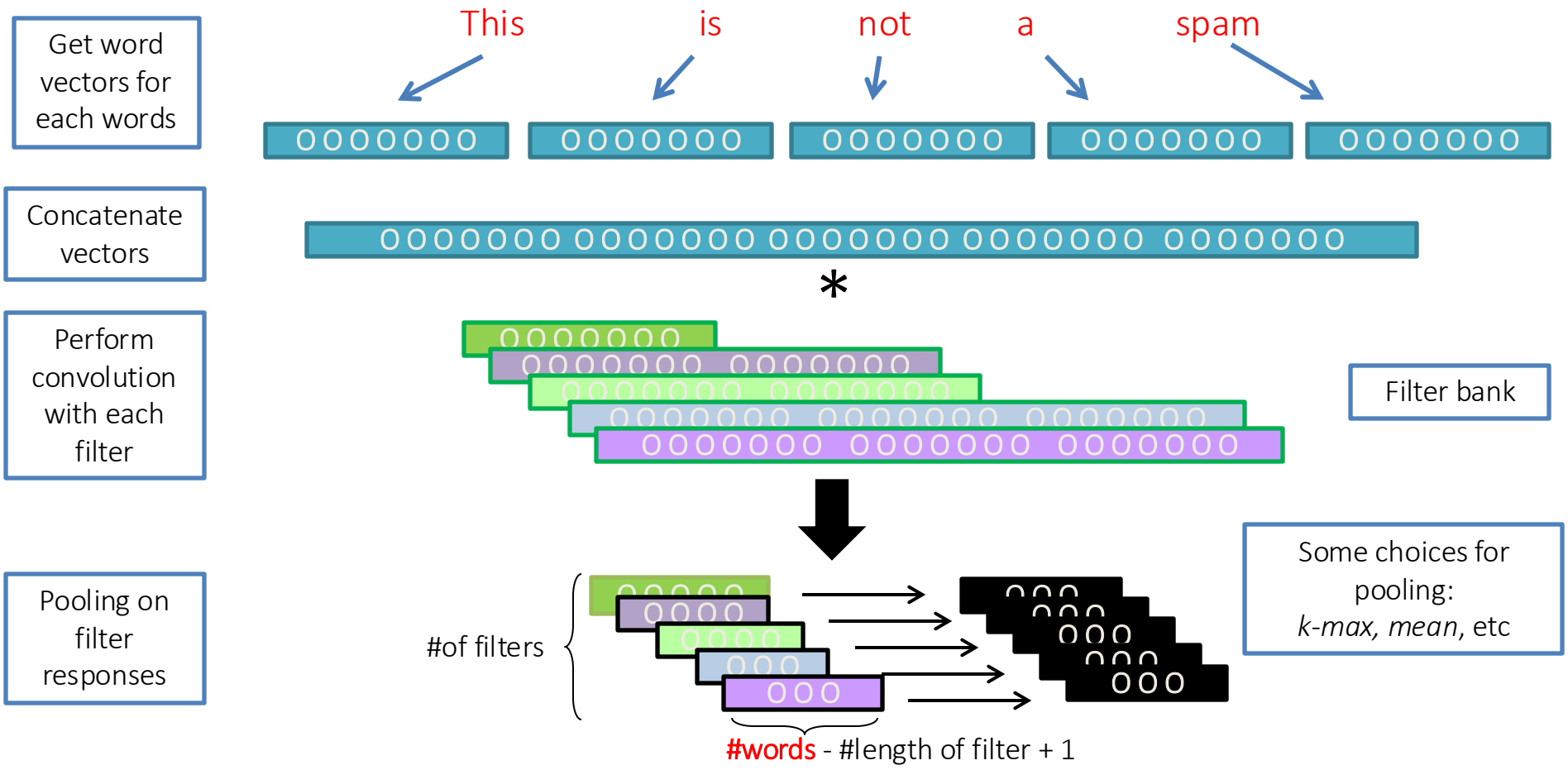
Perform convolution with each filter




How are we going to handle the **variable sized** response vectors? Pooling!

Set of response vectors

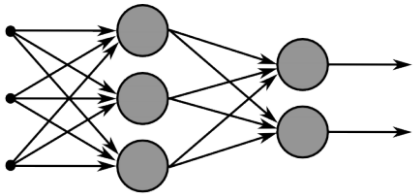
# Convolutional Layer on vectors




- Now we can pass the fixed-sized vector to a logistic unit (softmax), or give it to multi-layer network (last session)

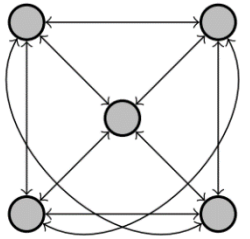
# Recurrent Neural Networks

- Multi-layer feed-forward NN: DAG
  - Just computes a fixed sequence of



non-linear learned transformations to convert an input patter into an output pattern

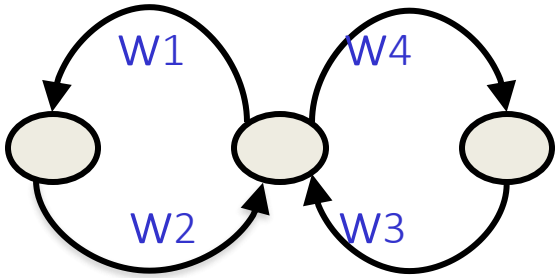
- Recurrent Neural Network: Digraph

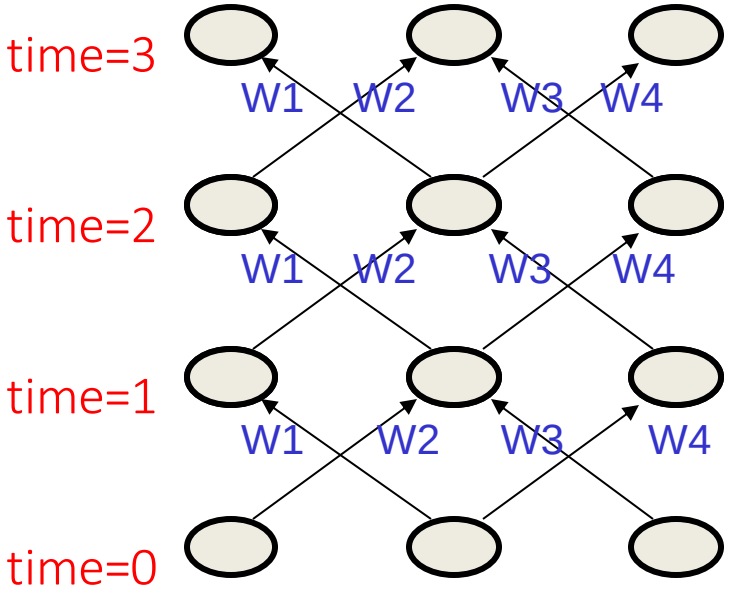


  - Has cycles.
  - Cycle can act as a memory;
  - The hidden state of a recurrent net can carry along information about a “potentially” unbounded number of previous inputs.
  - They can model sequential data in a much more natural way.

# Equivalence between RNN and Feed-forward NN

- Assume that there is a time delay of 1 in using each connection.
- The recurrent net is just a layered net that keeps reusing the same weights.



# Recurrent Neural Networks

- Training a general RNN's can be hard
  - Here we will focus on a **special family of RNN's**
- Prediction on chain-like input:
  - Language model

| | | | | | | |
|---|---|---|---|---|---|---|
| $X, h =$ | This | is | a | sample | sentence | . |
| $Y =$ | is | a | sample | sentence | . | <EOS> |

  - POS tagging words of a sentence

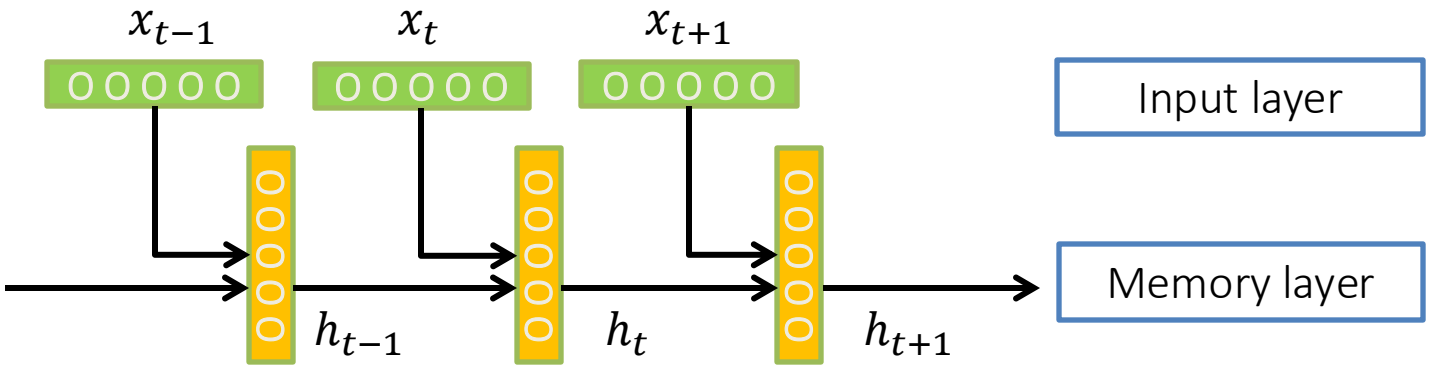
| | | | | | | |
|---|---|---|---|---|---|---|
| $X =$ | This | is | a | sample | sentence | . |
| $Y =$ | DT | VBZ | DT | NN | NN | . |

  - Sentiment classification

| | | | | | | |
|---|---|---|---|---|---|---|
| $X =$ | This | is | a | sample | sentence | . |
| $Y =$ | Positive/Negative | | | | | |

# Recurrent Neural Networks

- A chain RNN:
  - Has a chain-like structure
  - Each input is replaced with its vector representation $x_t$
  - Hidden (memory) unit $h_t$ contain information about previous inputs and previous hidden units $h_{t-1}, h_{t-2}$, etc
    - Computed from the past memory and current word. It summarizes the sentence up to that time.

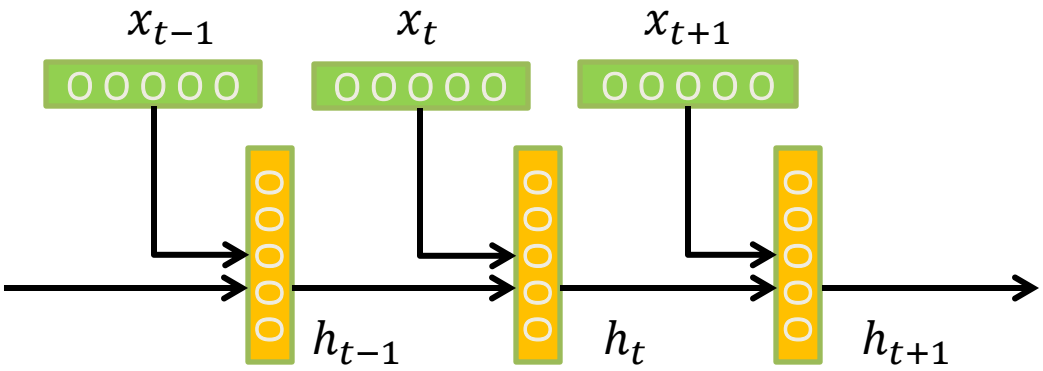


# Recurrent Neural Networks

- A popular way of formalizing it:

$$h_t = f(W_h h_{t-1} + W_i x_t)$$

  - Where $f$ is a nonlinear, differentiable (why?) function.
- Outputs?
  - Many options; depending on problem and computational resource



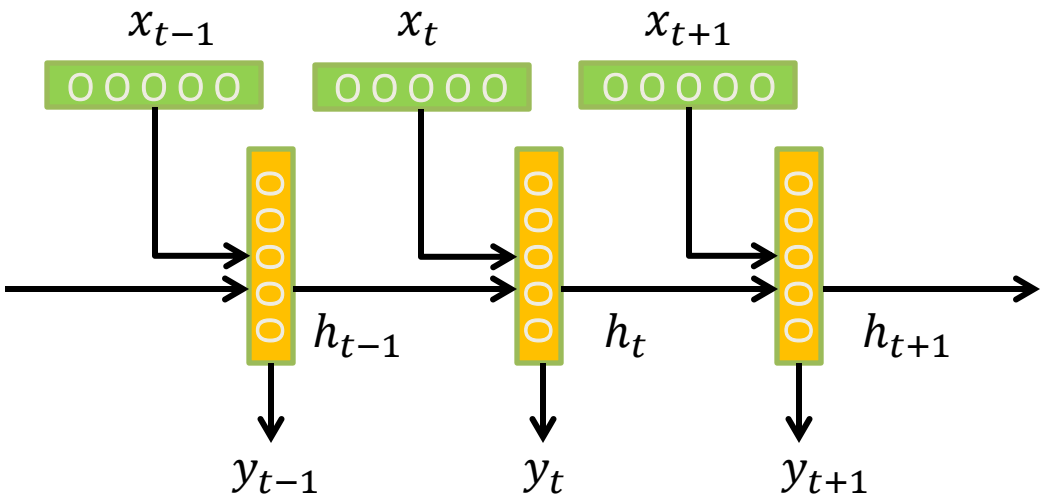
# Recurrent Neural Networks

- Prediction for $x_t$, with $h_t$
- Prediction for $x_t$, with $h_t, \dots, h_{t-\tau}$
- Prediction for the whole chain

$$y_t = \text{softmax}(W_o h_t)$$

$$y_t = \text{softmax}\left(\sum_{i=0}^{\tau} \alpha^i W_o^{t-i} h_{t-i}\right)$$

$$y_T = \text{softmax}(W_o h_T)$$



# Training RNNs

- How to train such model?
  - Generalize the same ideas from back-propagation
- Total output error: $E(\vec{y}, \vec{t}) = \sum_{t=1}^{T} E_t(y_t, t_t)$

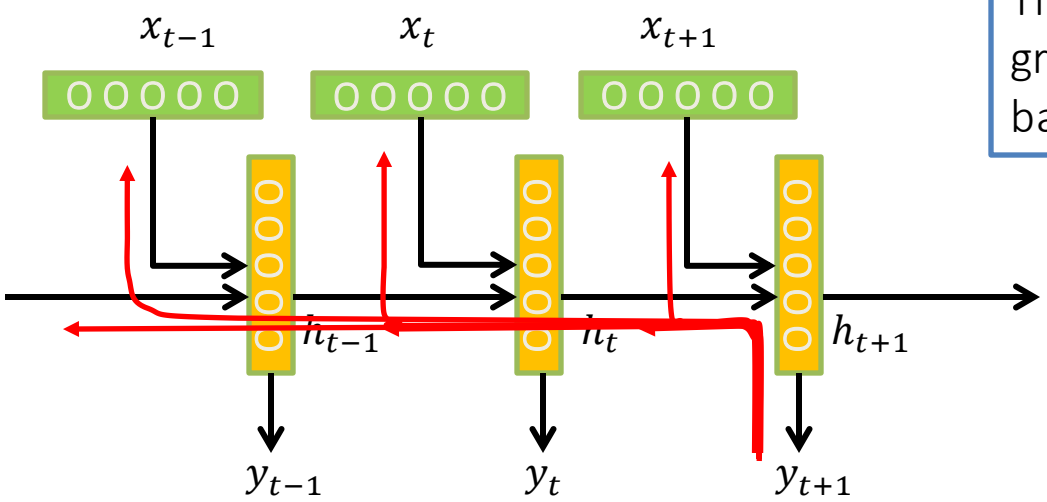
Parameters?
$W_o, W_i, W_h$ +
vectors for input

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-k}} \frac{\partial h_{t-k}}{\partial W}$$

Reminder:
$y_t = \text{softmax}(W_o h_t)$
$h_t = f(W_h h_{t-1} + W_i x_t)$

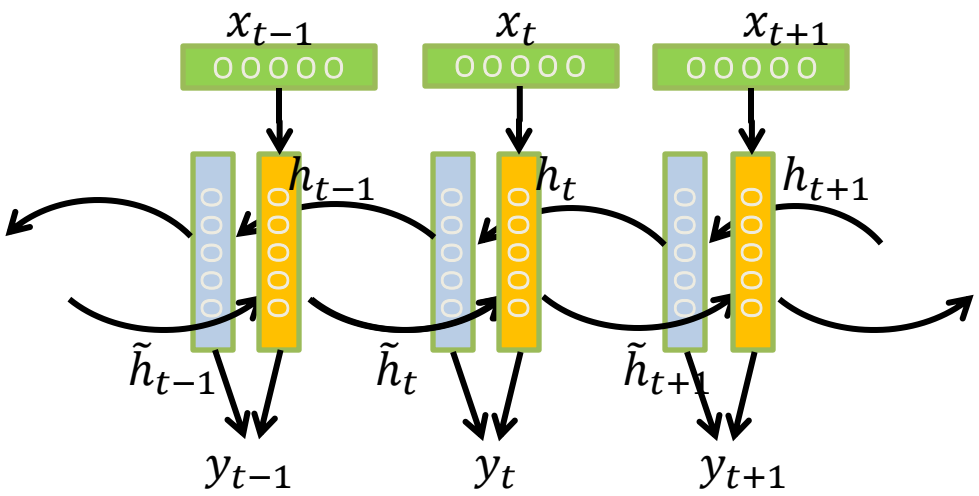
This sometimes is called "Backpropagation Through Time", since the gradients are propagated back through time.




Backpropagation for RNN

# Bi-directional RNN

- One of the issues with RNN:
- Hidden variables capture only one side context
- A bi-directional structure



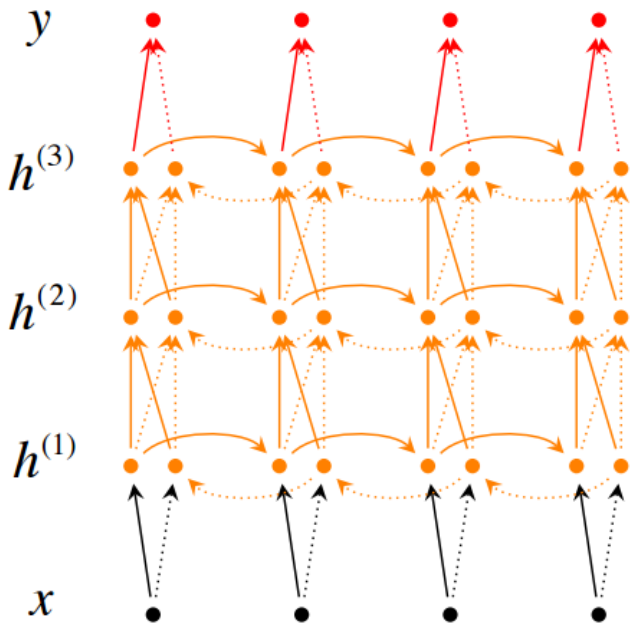

$$h_t = f(W_h h_{t-1} + W_i x_t)$$

$$\tilde{h}_t = f(\tilde{W}_h \tilde{h}_{t+1} + \tilde{W}_i x_t)$$

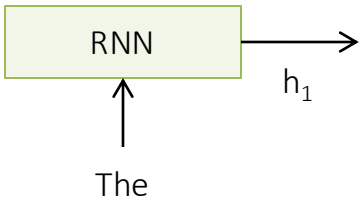
$$y_t = \text{softmax}\left(W_o h_t + \tilde{W}_o \tilde{h}_t\right)$$

# Stack of bi-directional networks

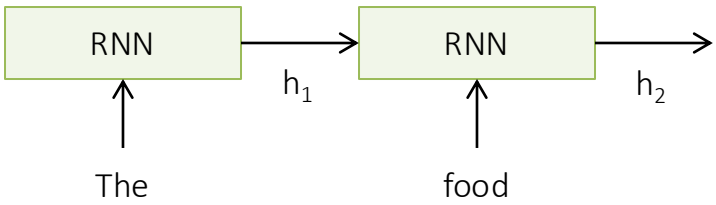
- Use the same idea and make your model further complicated:



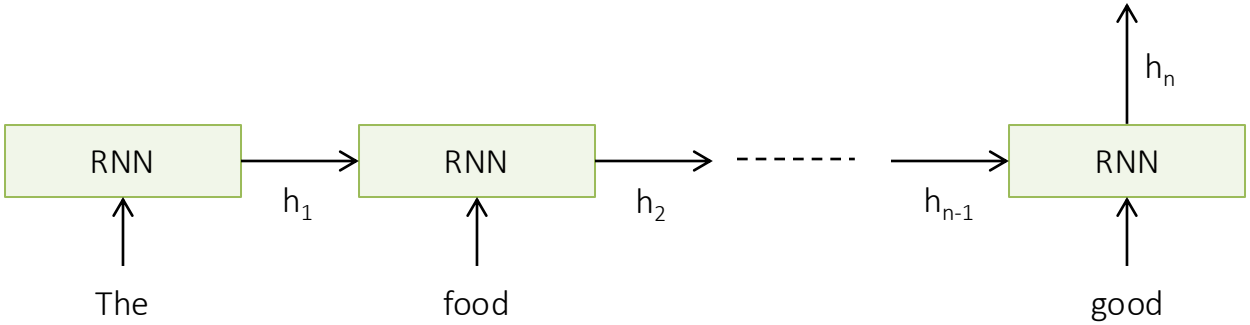
# Sentiment Classification



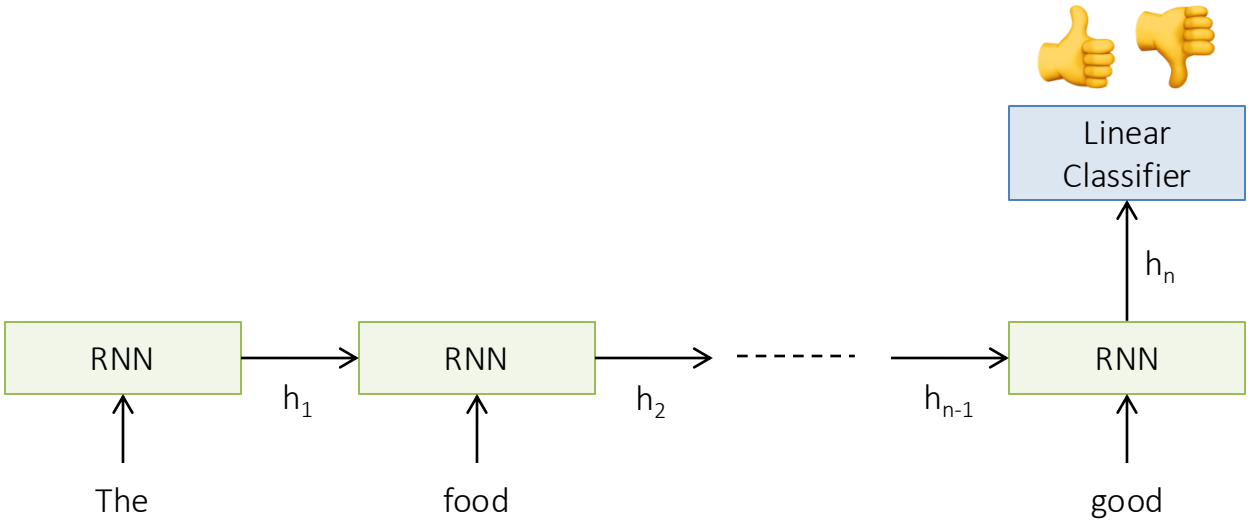
# Sentiment Classification



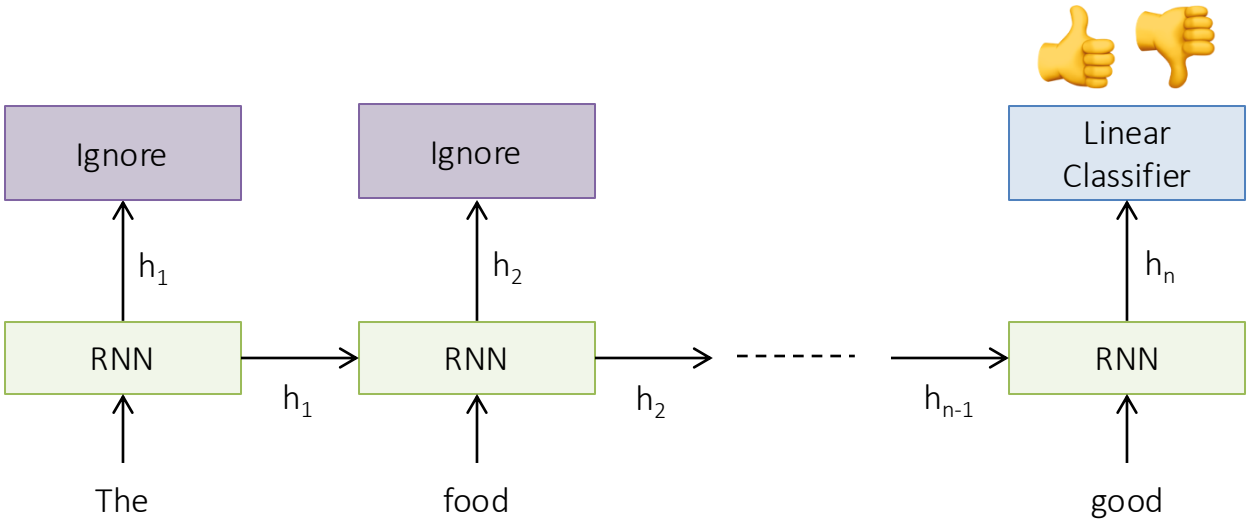
# Sentiment Classification



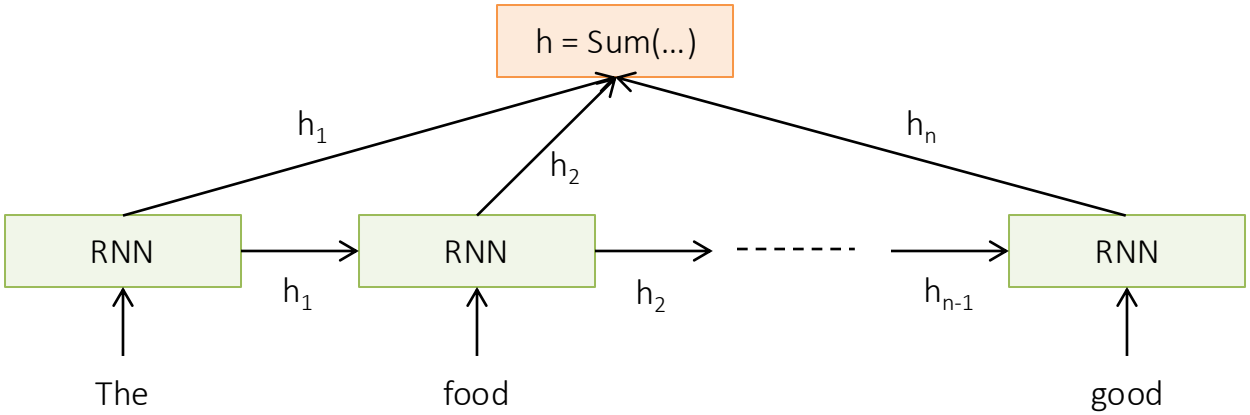
# Sentiment Classification



# Sentiment Classification

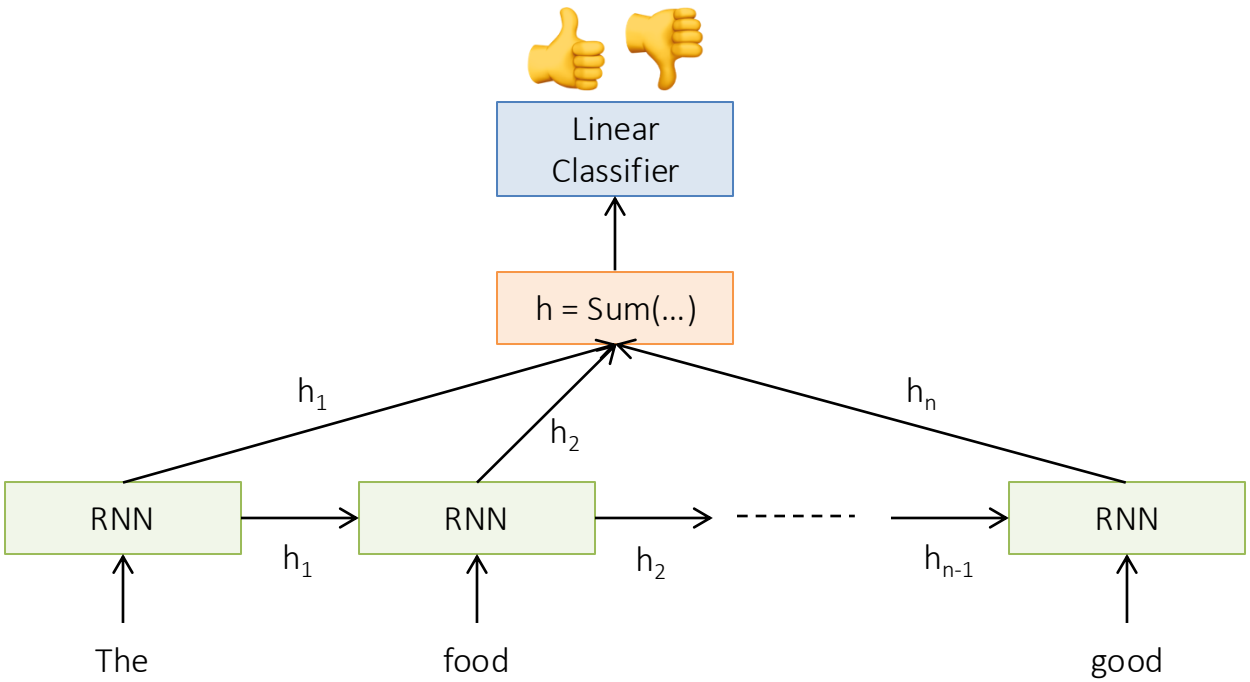


# Sentiment Classification



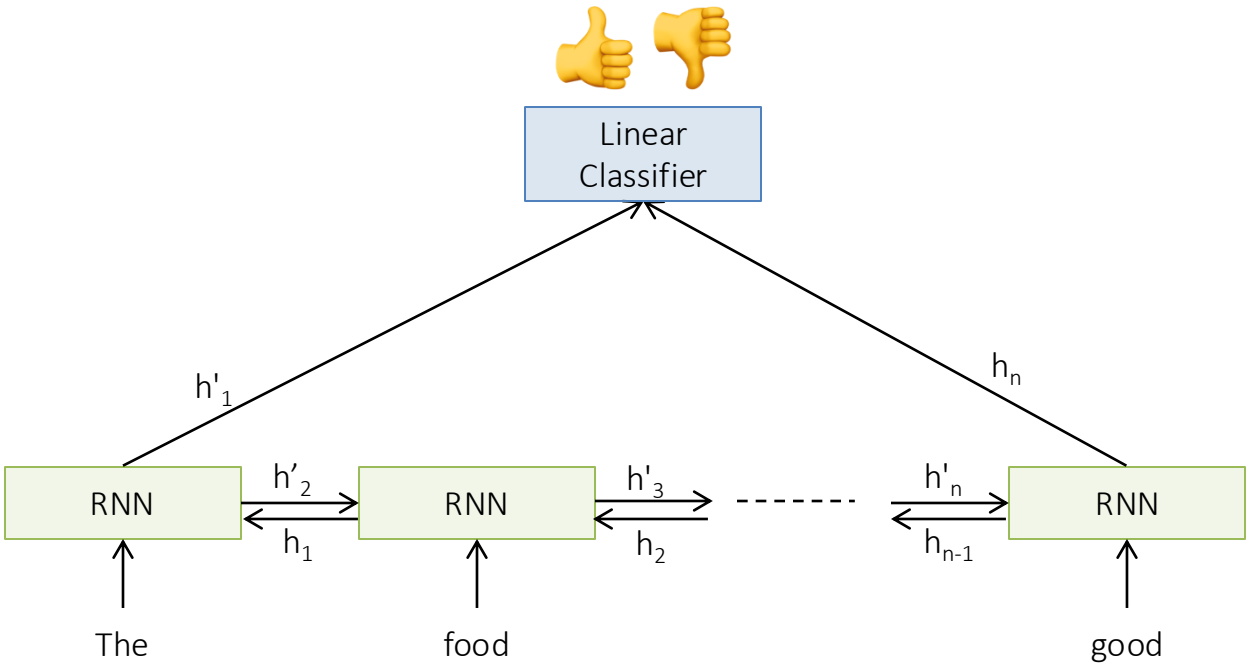

http://deeplearning.net/tutorial/lstm.html

# Sentiment Classification




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

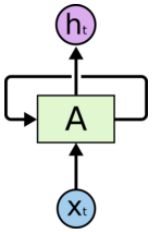
# Sentiment Classification




Bi-directional RNN

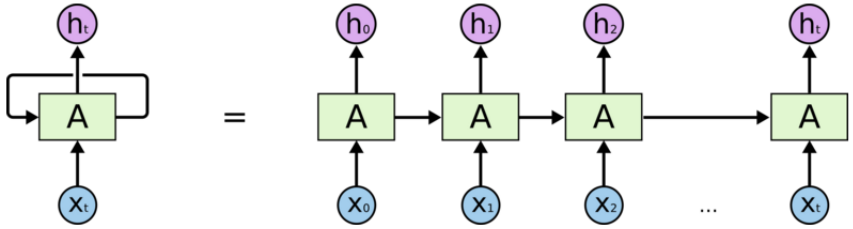
# More Notations Used in Papers

- Recurrent Neural Networks are networks with loops in them, allowing information to persist.




Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, $A$, looks at some input $x_t$ and outputs a value $h_t$.
A loop allows information to be passed from one step of the network to the next.

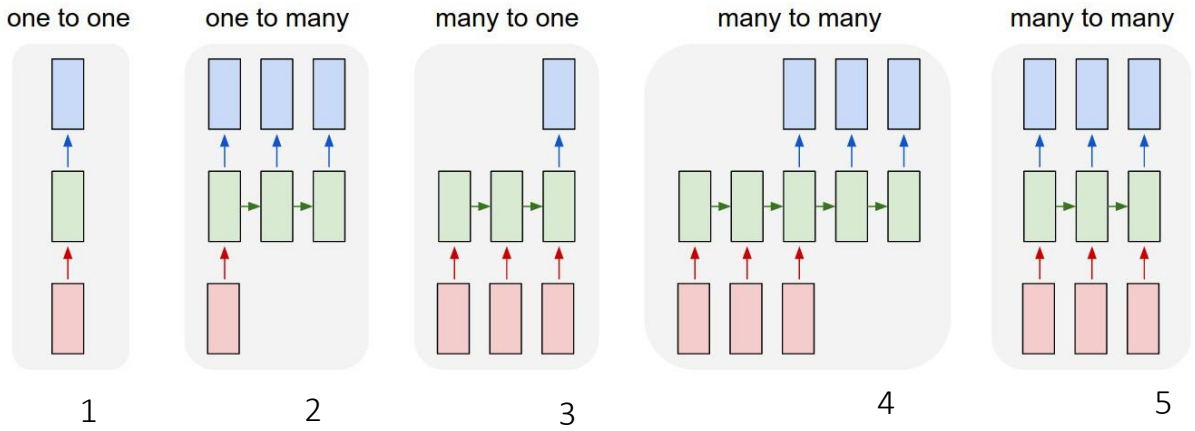



An unrolled recurrent neural network.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
The diagram above shows what happens if we unroll the loop.

# Recurrent Neural Networks

- Examples of Recurrent Neural Networks




- Each rectangle is a vector and arrows represent functions (e.g. matrix multiply).
- Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state

(1) Standard mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).
(2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).
(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

# Vanishing/exploding gradients

- In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - So RNNs have difficulty dealing with long-range dependencies.
- Many methods proposed for reducing the effect of vanishing gradients; although it is still a problem
  - Introduce shorter path between long connections
  - Abandon stochastic gradient descent in favor of a much more sophisticated Hessian-Free (HF) optimization
  - Add fancier modules that are robust to handling long memory; e.g. Long Short Term Memory (LSTM)
    - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
    - A very good explanatiom of LSTM
- One trick to handle the exploding-gradients:
  - Clip gradients with bigger sizes:

Define $g = \frac{\partial E}{\partial W}$

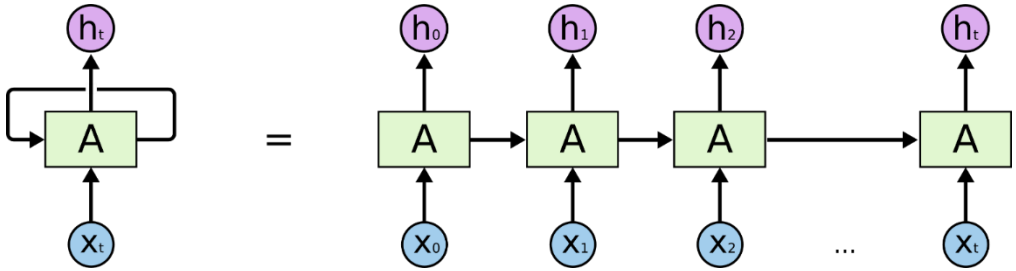
If $\|g\| \geq threshold$ then

$$g \leftarrow \frac{threshold}{\|g\|} g$$

# LSTM

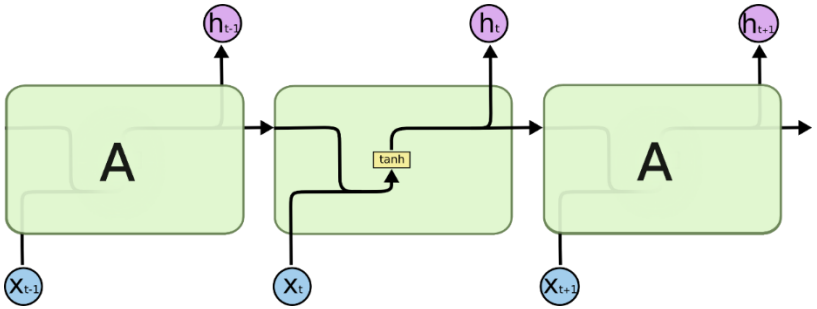
- Long short-term memory
  - Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780.

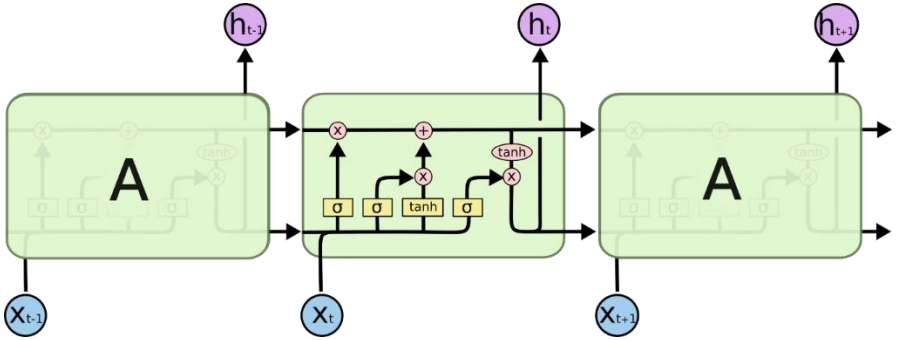
$$\vec{h}_i = f(\boldsymbol{W}_h \vec{h}_{i-1} + \boldsymbol{W}_x \vec{x}_i + \vec{b}_x)$$




LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

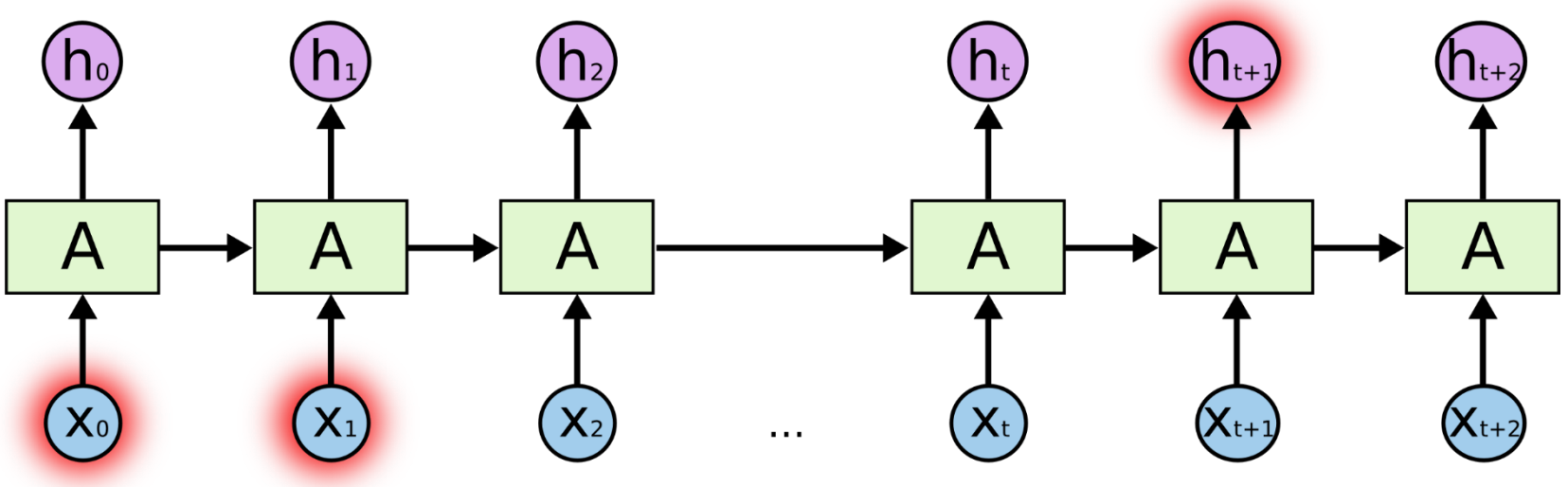
$$\vec{f}_t = \sigma(\boldsymbol{W}_f \vec{x}_t + \boldsymbol{U}_f \vec{h}_{t-1} + \vec{b}_f)$$
$$\vec{i}_t = \sigma(\boldsymbol{W}_i \vec{x}_t + \boldsymbol{U}_i \vec{h}_{t-1} + \vec{b}_i)$$
$$\vec{u}_t = tanh(\boldsymbol{W}_u \vec{x}_t + \boldsymbol{U}_u \vec{h}_{t-1} + \vec{b}_u)$$
$$\vec{c}_t = \vec{i}_t \odot \vec{u}_t + \vec{f}_t \odot \vec{c}_{t-1}$$
$$\vec{o}_t = \sigma(\boldsymbol{W}_o \vec{x}_t + \boldsymbol{U}_o \vec{h}_{t-1} + \vec{b}_o)$$
$$\vec{h}_t = \vec{o}_t \tanh \odot (\vec{c}_t)$$



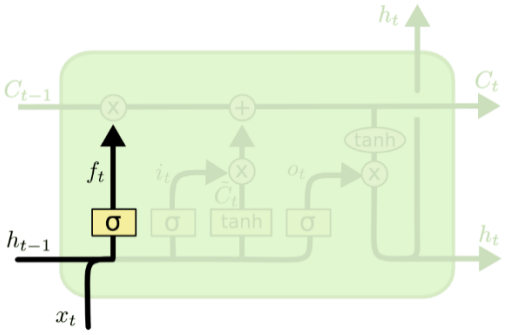




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

# LSTMs are explicitly designed to avoid the long-term dependency problem.

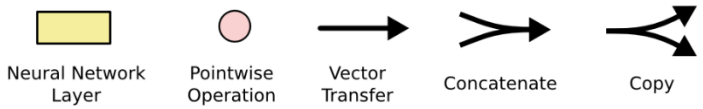


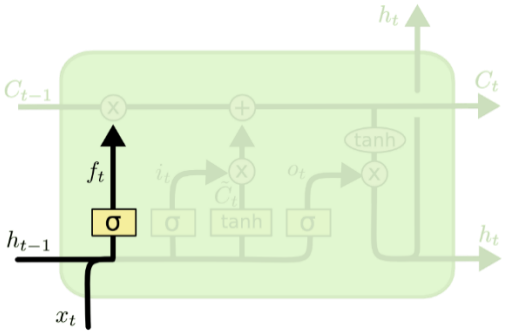

In theory, RNNs are absolutely capable of handling such “long-term dependencies.”
In practice, RNNs don’t seem to be able to learn them.




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

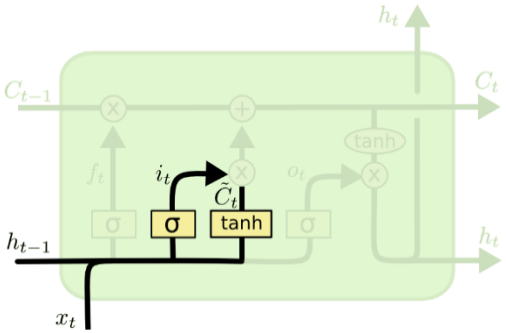
The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." A 1 represents "completely keep this" while a 0 represents "completely get rid of this."





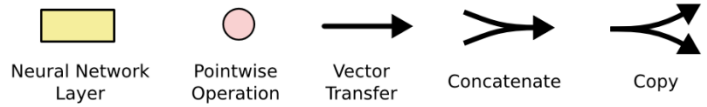

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

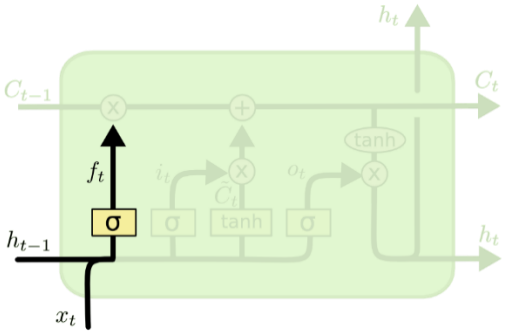
The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." A 1 represents "completely keep this" while a 0 represents "completely get rid of this."




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

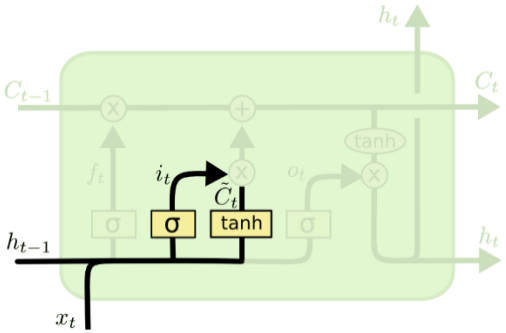
The next step is to decide what new information we're going to store in the cell state. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\hat{C}_t$, that could be added to the state.





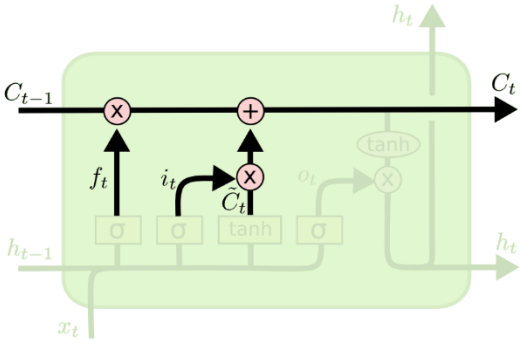

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



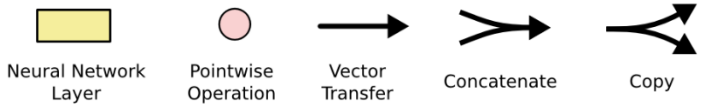

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

The next step is to decide what new information we're going to store in the cell state. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\hat{C}_t$, that could be added to the state.

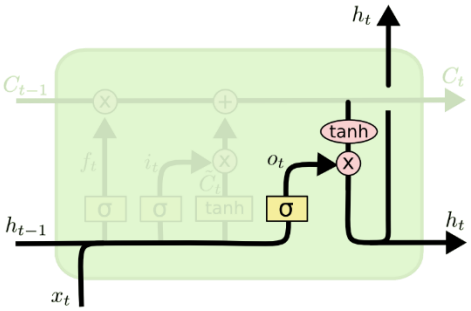



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

We multiply the old state, forgetting the things we decided to forget earlier.
The new candidate values to add is scaled by how much we decided to update each state value.

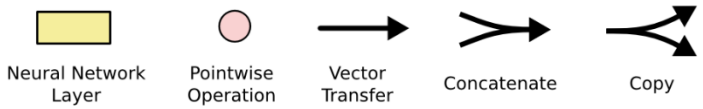


Finally, we need to decide what we're going to output.

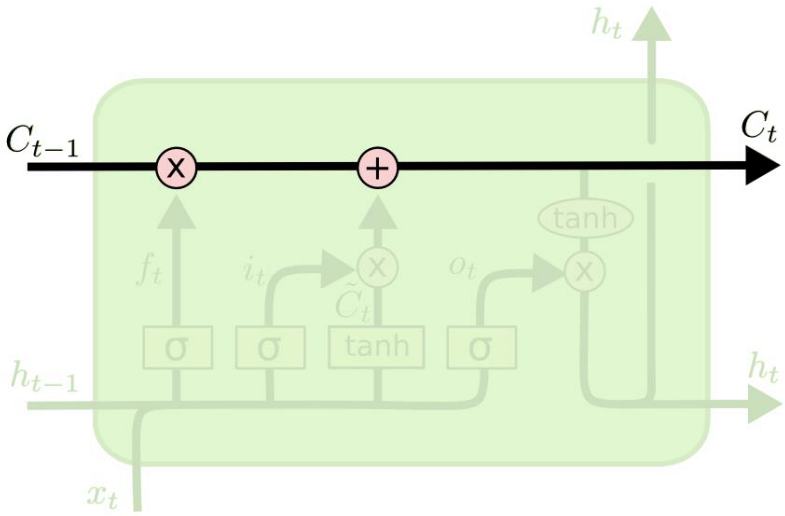



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

This output will be based on our cell state, but will be a filtered version.
First, we run a sigmoid layer which decides what parts of the cell state we're going to output gate.
Second, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



# The Core Idea Behind LSTMs




The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

# Limitation of RNNs

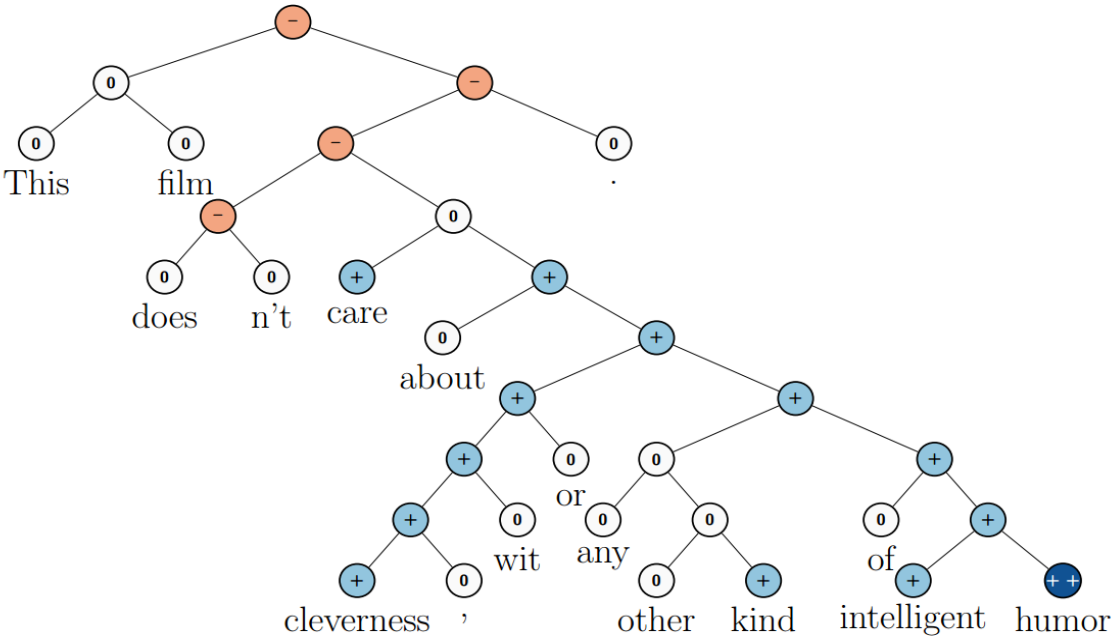
- Recurrent Neural Networks are unique as they allow us to operate over sequences of vectors.
  - Sequences in the input, the output, or in the most general case both

# Recursive Neural Networks

- Given the structural representation of a sentence, e.g. a parse tree:
  - Recursive Neural Networks recursively generate parent representations in a bottom-up fashion,
    - By combining tokens to produce representations for phrases, eventually producing the whole sentence.
  - The sentence-level representation (or, alternatively, its phrases) can then be used to make a final classification for a given input sentence — e.g. whether it conveys a positive or a negative sentiment.

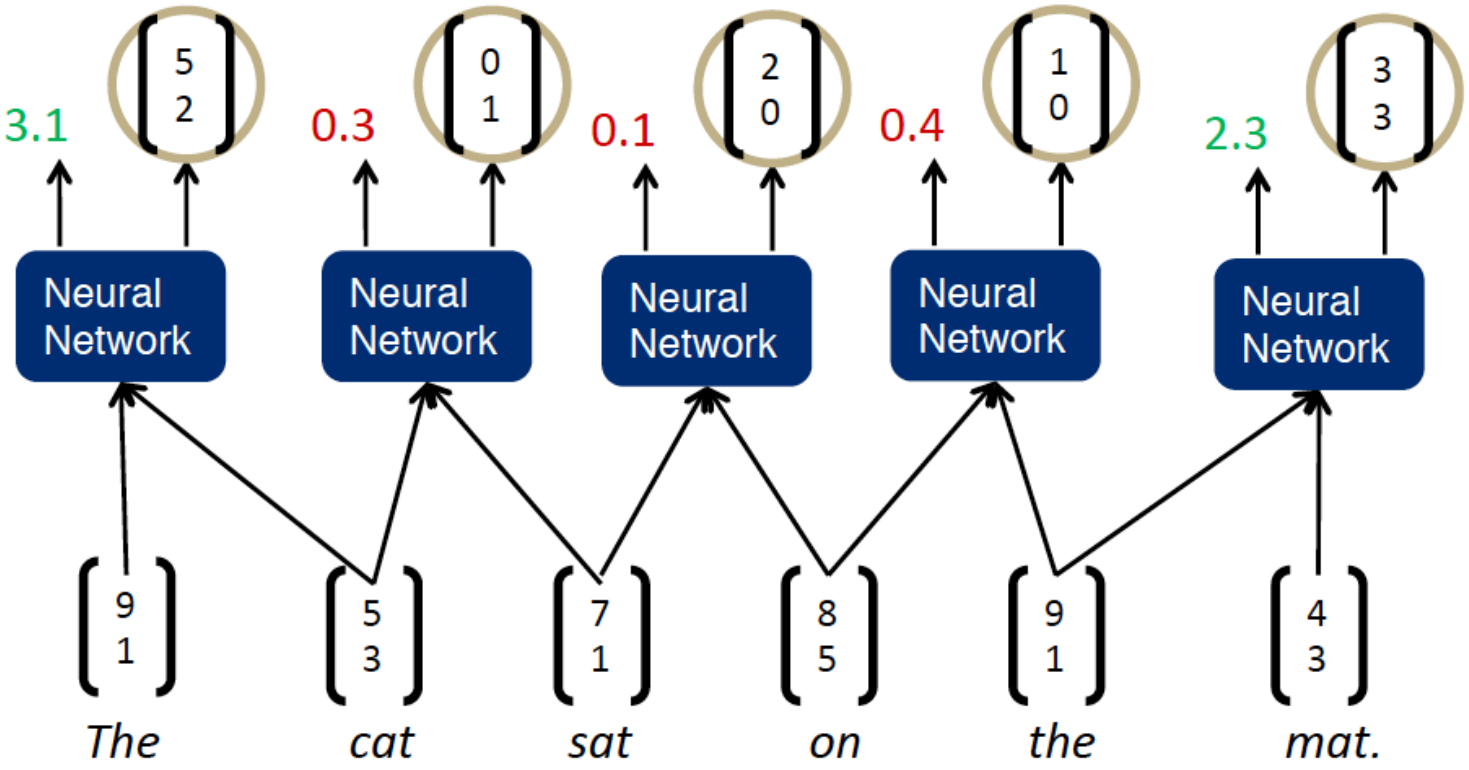
# Recursive Neural Networks

- Recursive neural networks have had significant successes in a number of NLP tasks.
  - Socher *et al.* (2013) uses a recursive neural network to predict sentence sentiment:

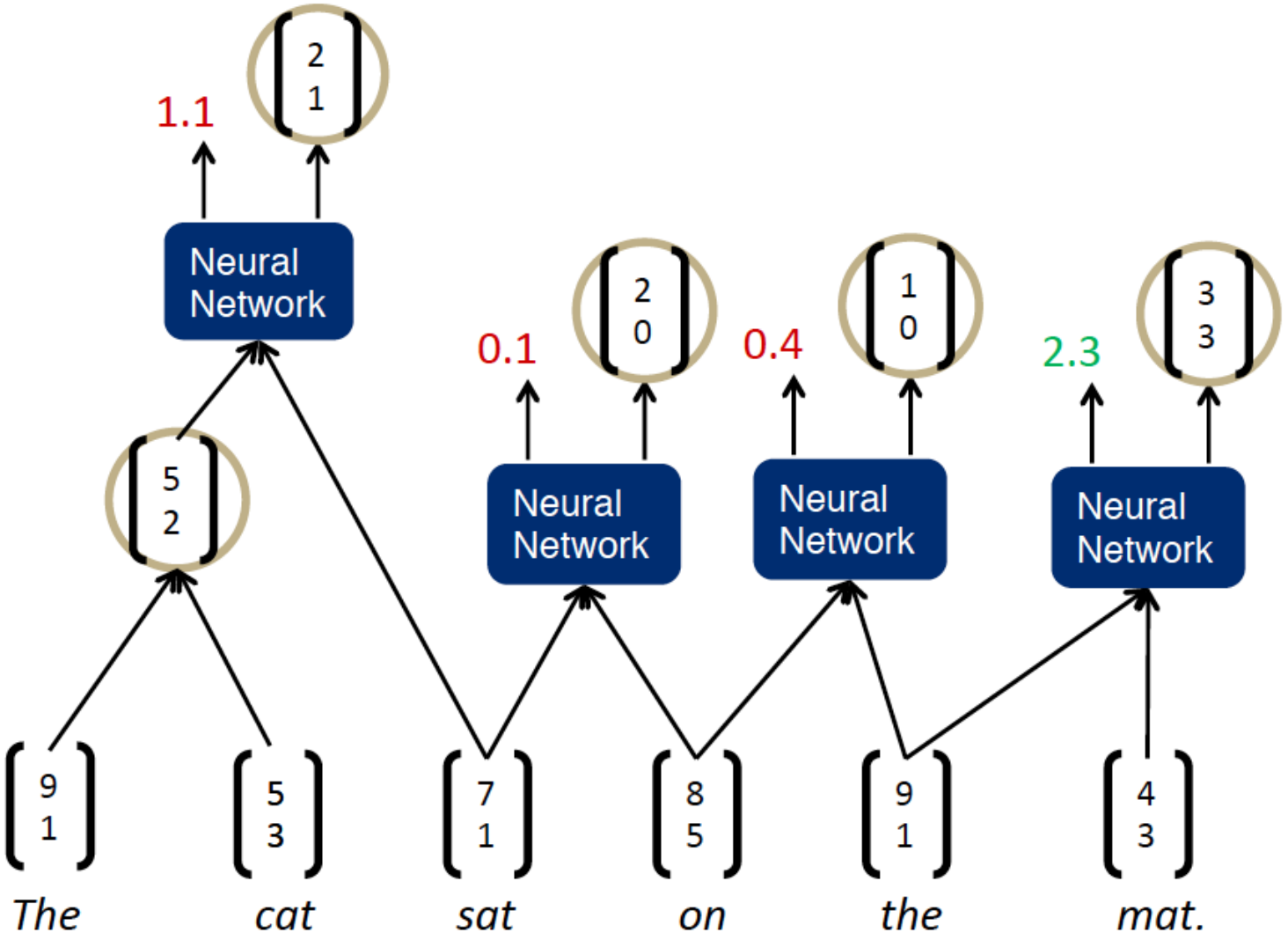



Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In Proceedings of EMNLP , 1631-1642.

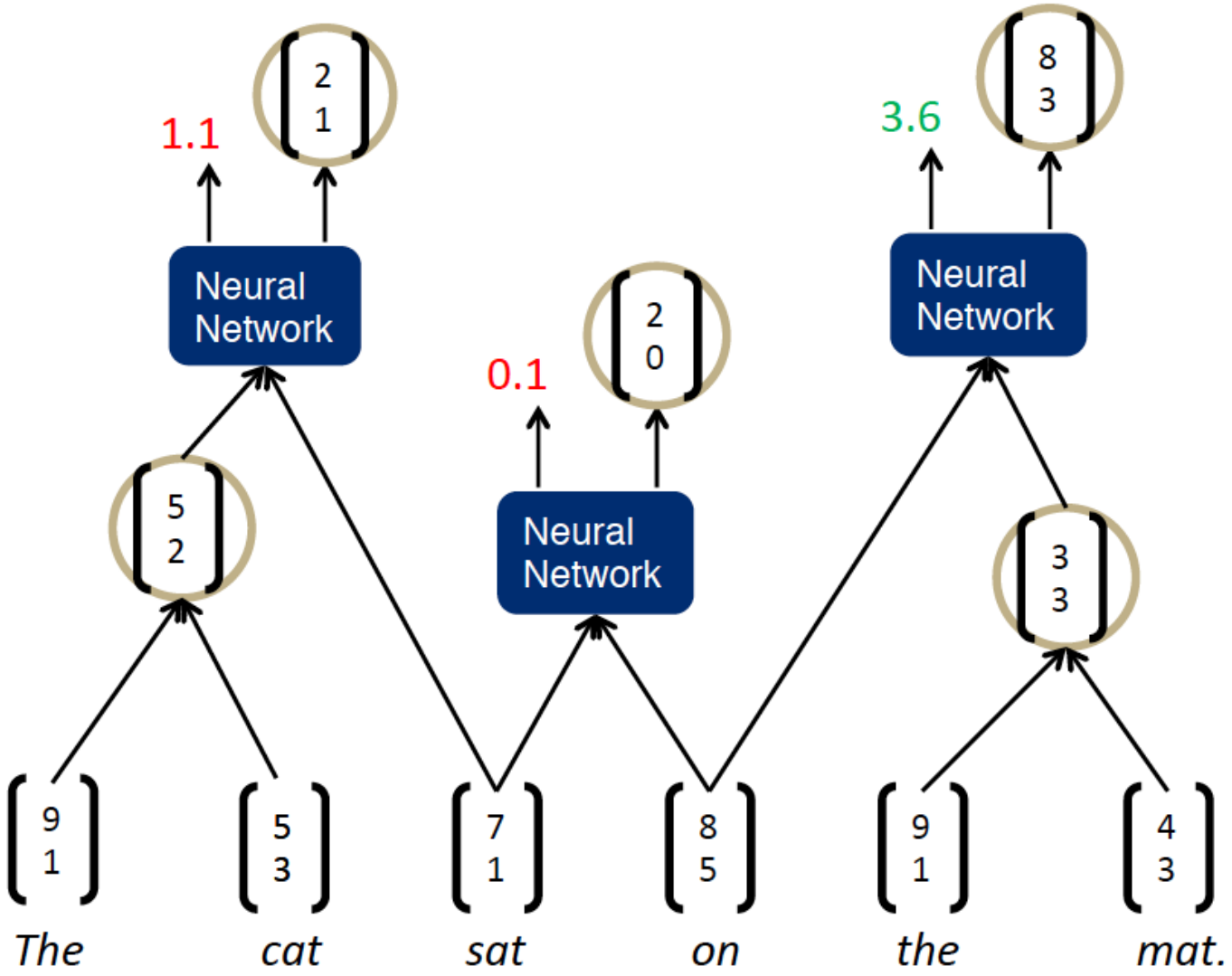
# Recursive Neural Network Illustration (Animation)



# Recursive Neural Network Illustration (Animation)



# Recursive Neural Network Illustration (Animation)

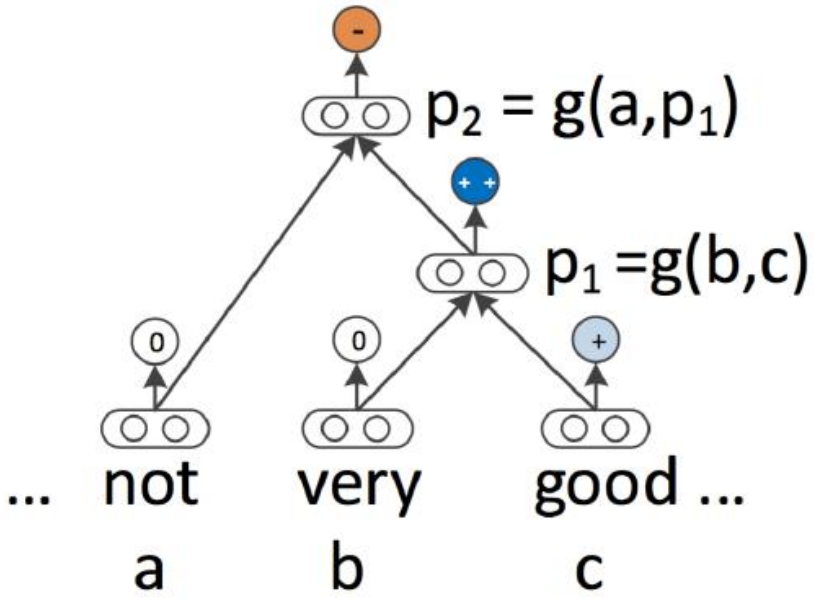


# Recursive Neural Network Component

$$\vec{h}_i = f(\boldsymbol{W}_l \vec{h}^l_{i-1} + \boldsymbol{W}_r \vec{h}^r_{i-1} + \vec{b}_h)$$

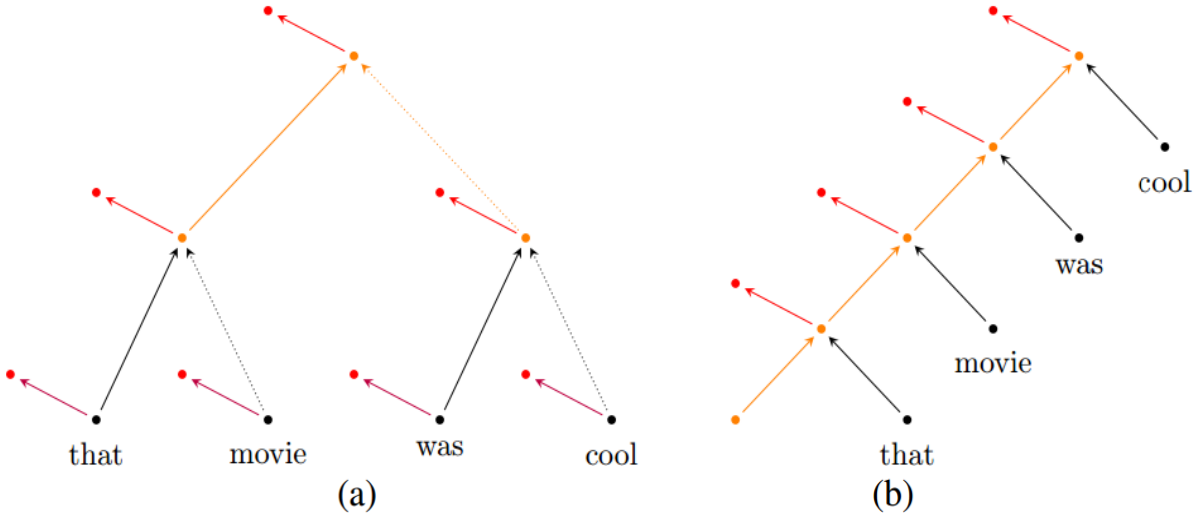
$$p_1 = f(\boldsymbol{W}\begin{bmatrix} b \\ c \end{bmatrix})$$

$$p_2 = f(\boldsymbol{W}\begin{bmatrix} a \\ p_1 \end{bmatrix})$$



# A recursive NN can be seen as a generalization of the recurrent NN

- A recurrent neural network is in fact a recursive neural network with the structure of a linear chain
- Recursive NNs operate on hierarchical structure, Recurrent NN operate on progression of time




- Operation of a recursive net (a), and a recurrent net (b) on an example sentence. Note the linear chain in (b)
- Black, orange and red dots represent input, hidden and output layers, respectively.
- Directed edges having the same color-style combination denote shared connections.

# Other Variants

- How to deal with long documents?
- Recurrent Neural Network
  - Disadvantage: “a biased model, where later words are more dominant than earlier words”
- Convolutional Neural Network
  - Advantages:
    - capture the semantic of texts better than recursive or recurrent NN.
    - Time complexity of CNN is O(n).
  - Disadvantages:
    - It is difficult to determine the window size

# Other Variants (RNN+CNN)

- RNN+CNN
- Also, connect embeddings to hidden