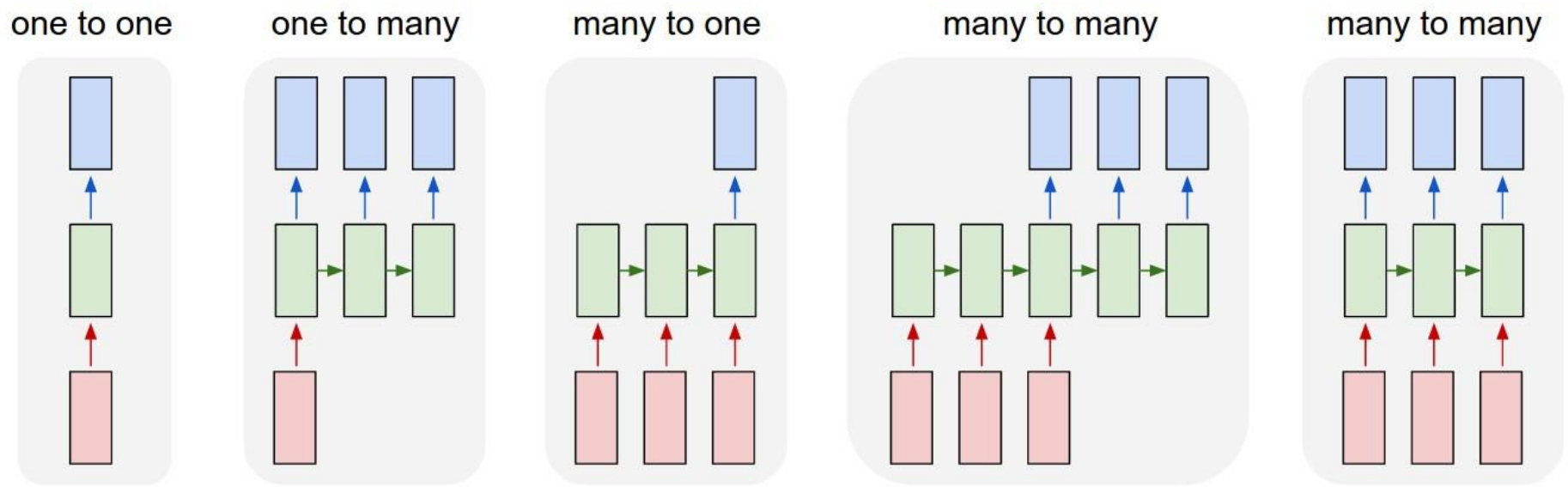
Recurrent Neural Networks

©Dr. Min Chi
mchi@ncsu.edu

AI Academy

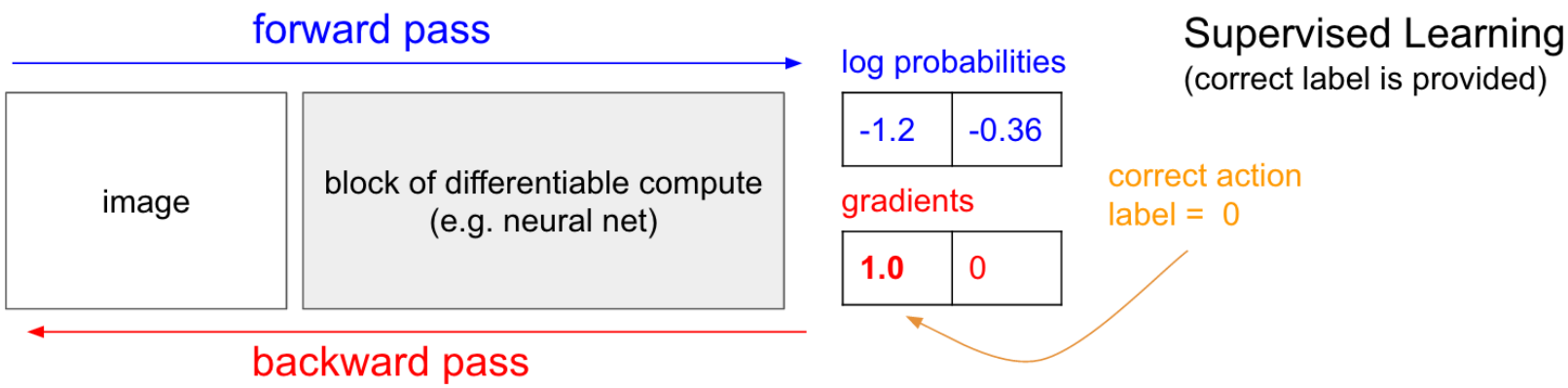
Flavors of Neural Networks



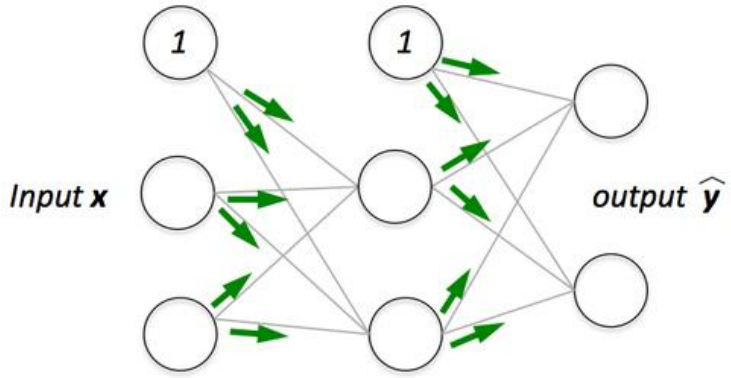
"Vanilla" Neural Networks

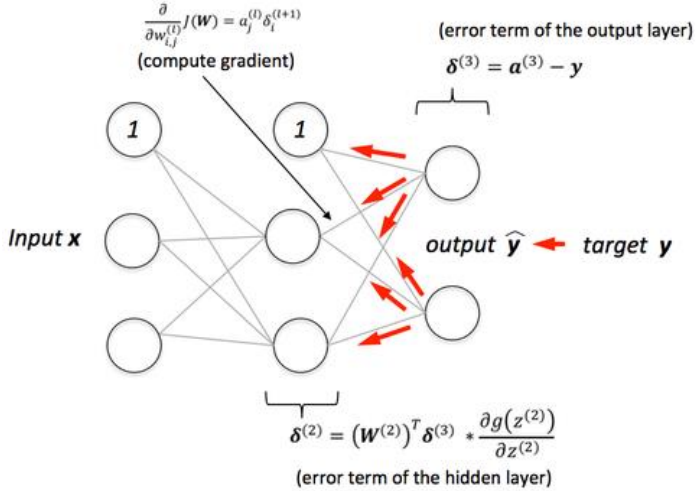
Recurrent Neural Networks

Back to Basics: Backpropagation



Adjust the weights to reduce the error:

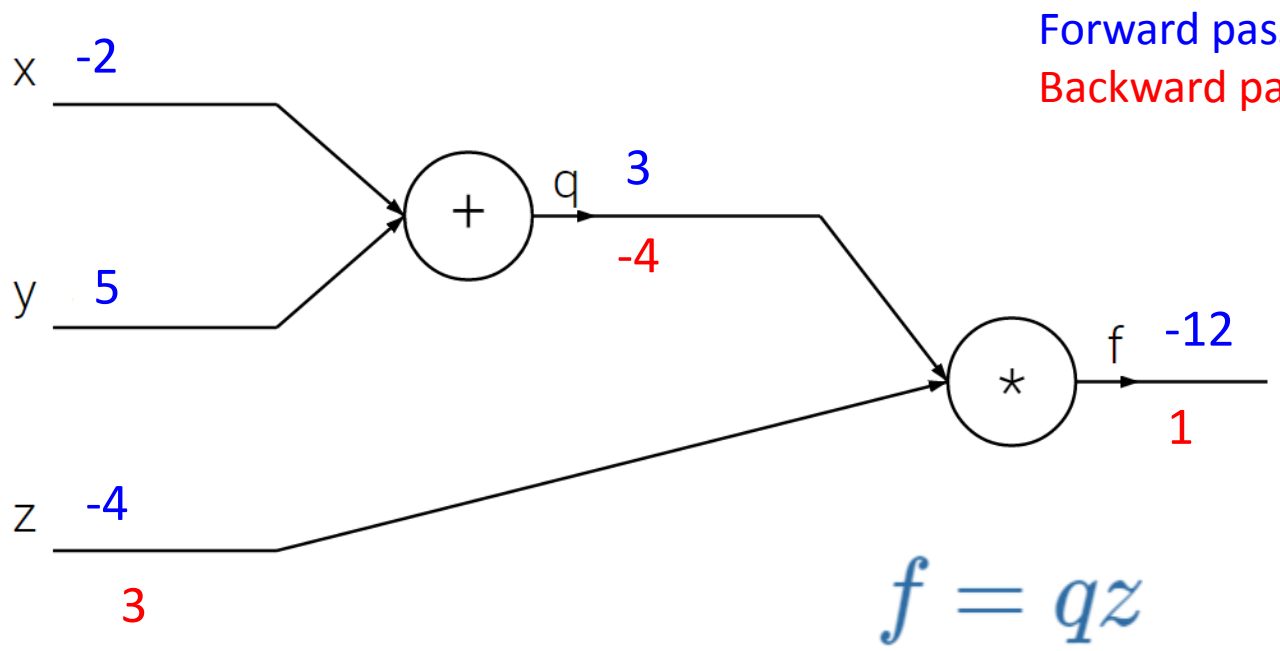




AI Academy

NC STATE

Backpropagation: Backward Pass

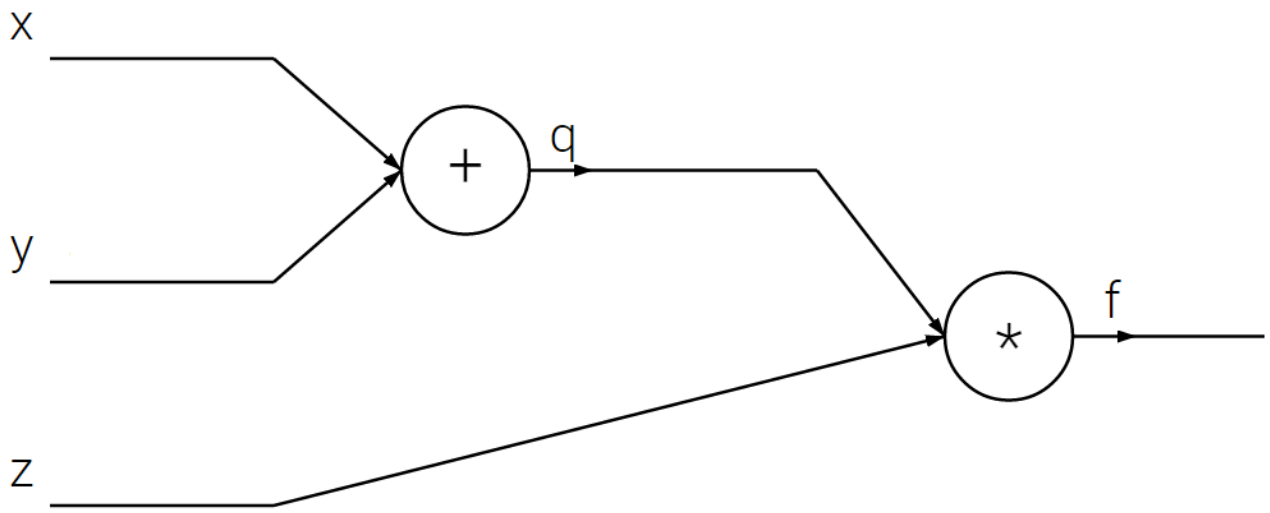


Let's compute the local gradient on *f*:

$$\frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

Backpropagation: By Example

$$f(x, y, z) = (x + y)z$$

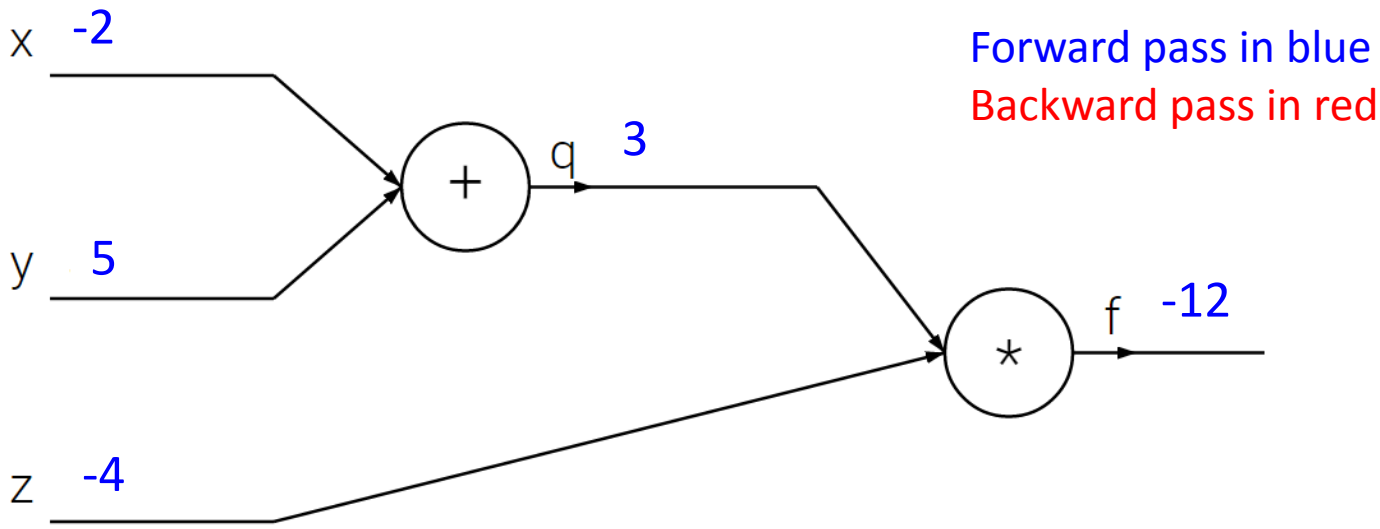


Modularity: We compute an arbitrary function locally in stages

$$q = x + y \qquad f = qz$$

Backpropagation: Forward Pass

$$f(x, y, z) = (x + y)z$$

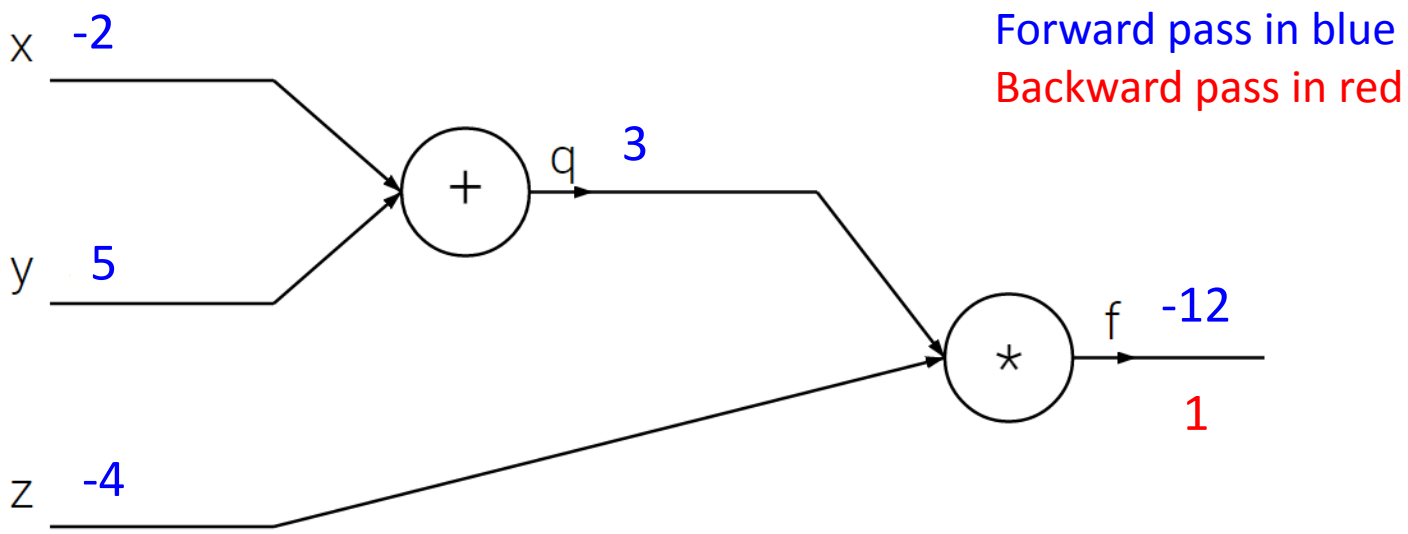


f(x, y, z) is “happy” when the output is positive.

How do we “teach” it to produce a higher output?

Backpropagation: Forward Pass

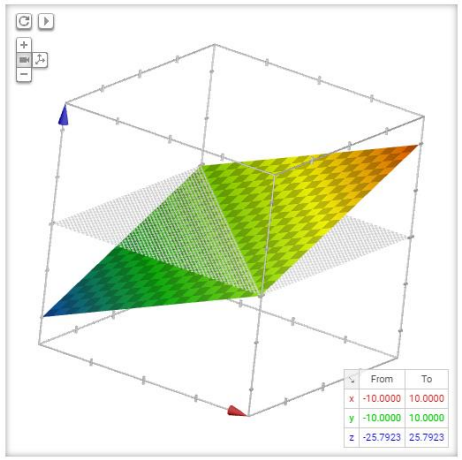
$$f(x, y, z) = (x + y)z$$



f(x, y, z) is “happy” when the output is positive.

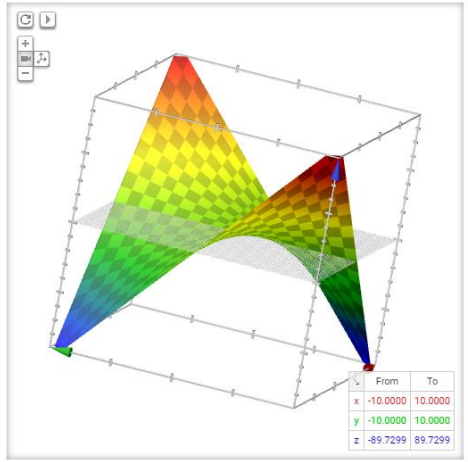
How do we “teach” it to produce a higher output?

Backpropagation: By Example



Addition:

$$f(x,y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial y} = 1$$



Multiplication:

$$f(x,y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$

Modular Magic: Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

So, instead of computing the gradient of this:

$$f(x, y, z) = (x + y)z$$

We compute the gradients of these:

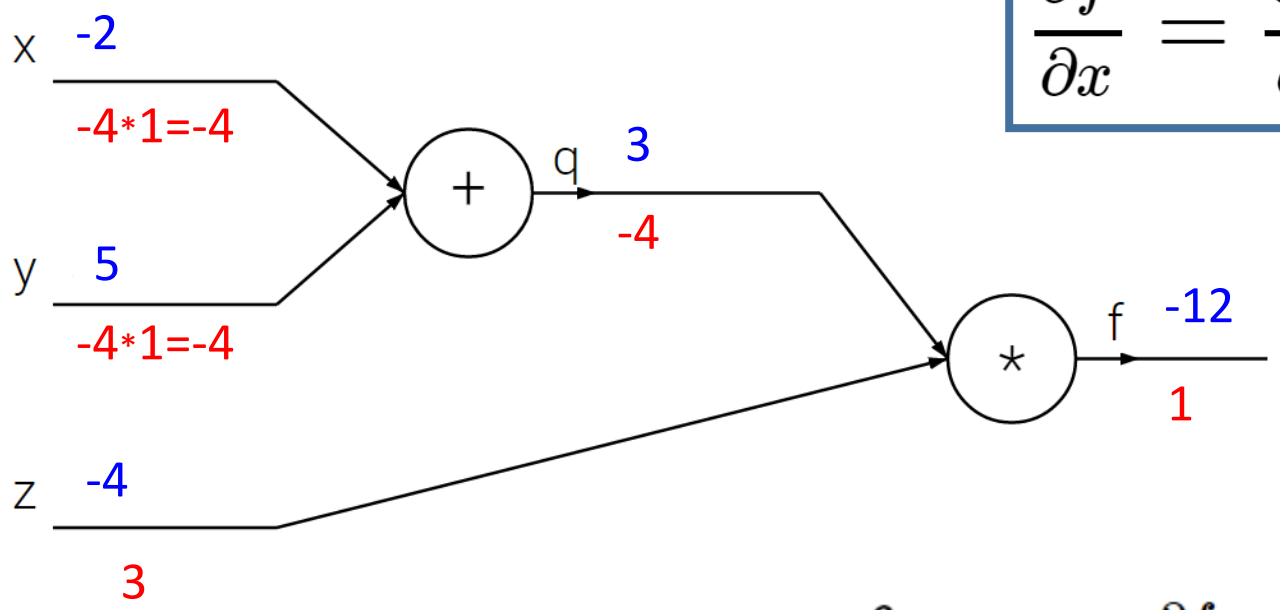
$$q = x + y \qquad f = qz$$

Backpropagation: Backward Pass

Forward pass in blue
Backward pass in red

Modular Magic: Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



$$q = x + y \qquad f = qz \quad \frac{\partial f}{\partial q} = z$$

Let's compute the local gradient on *q*:

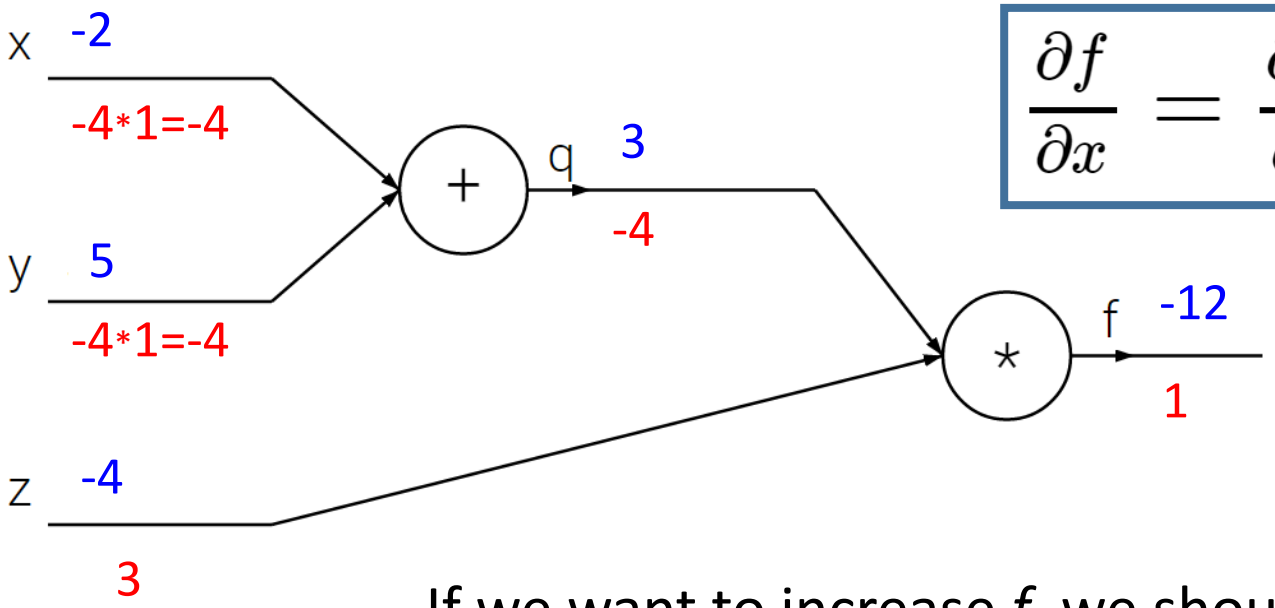
$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

Forward pass in blue
Backward pass in red

Interpreting Gradients

Modular Magic: Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

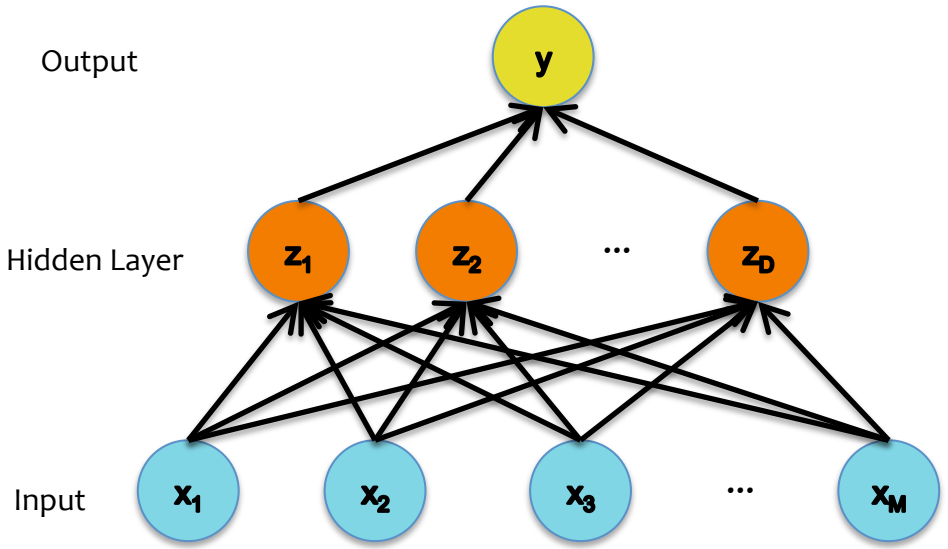


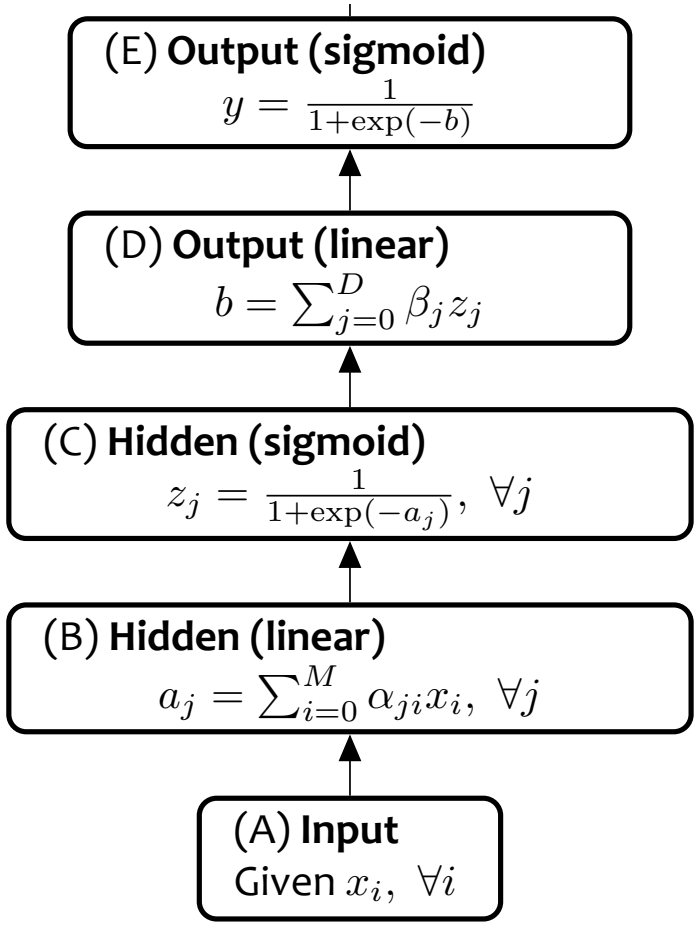
If we want to increase *f*, we should:

- Decrease *q*
- Decrease *x*
- Decrease *y*
- Increase *z*

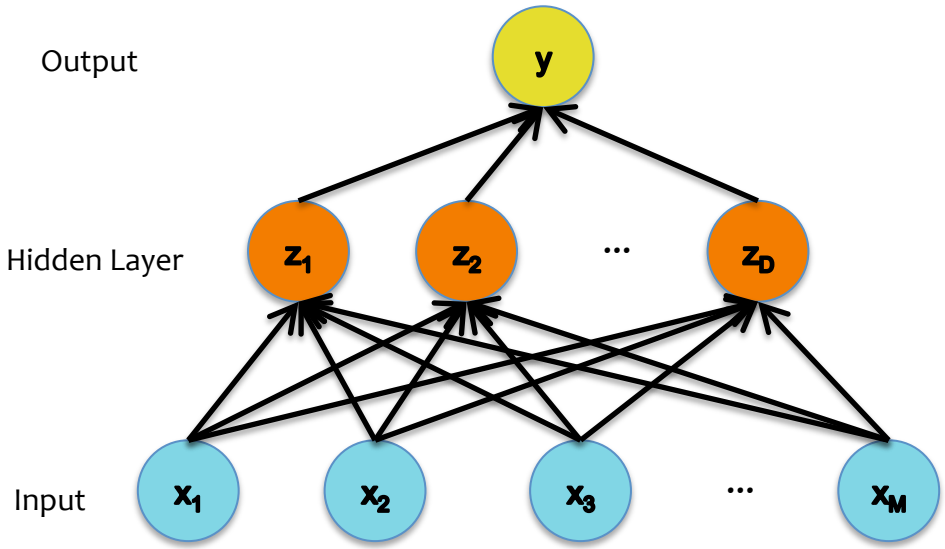
Every local gradient is a local worker in a global chase for greater *f*.

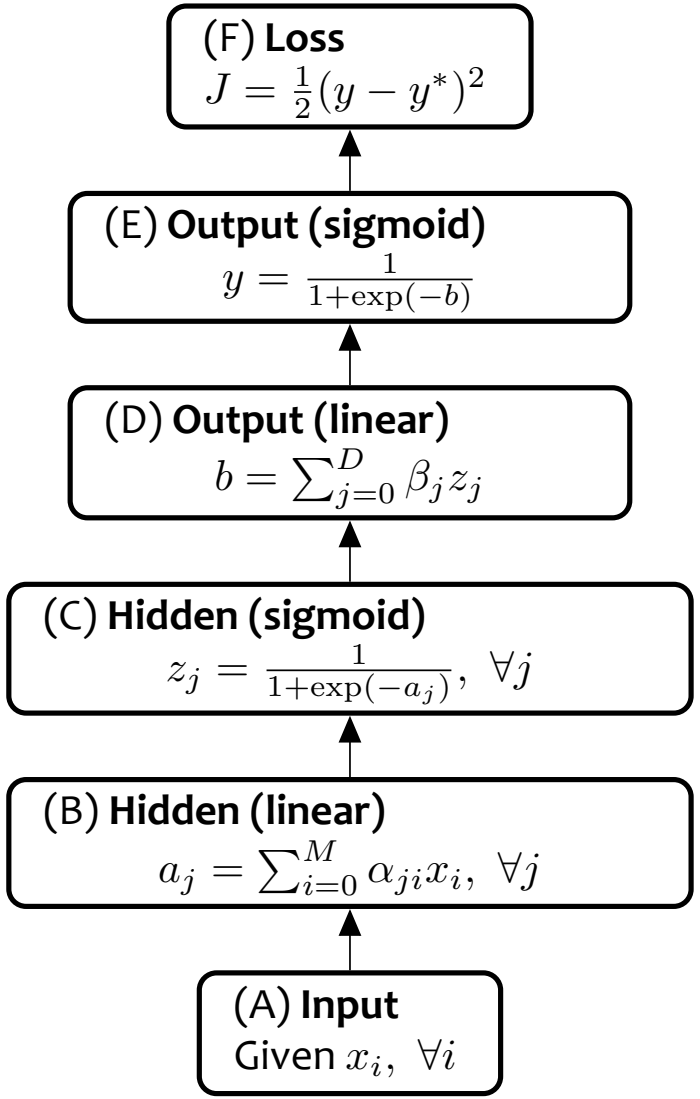
Backpropogation in a One-layer NN





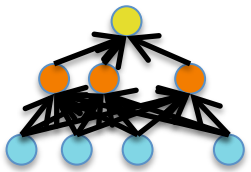
Backpropogation in a One-layer NN





Backpropogation in a One-layer NN

Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

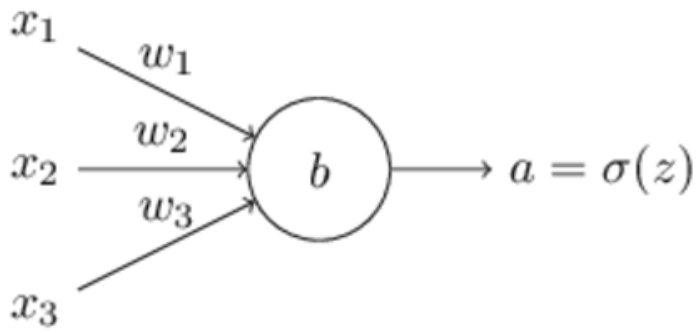
$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$$

Learning with Backpropagation

Task: Update the **weights** and **biases** to decrease **loss function**



Loss (aka cost, objective) function:

$$C = \frac{(y-a)^2}{2}$$

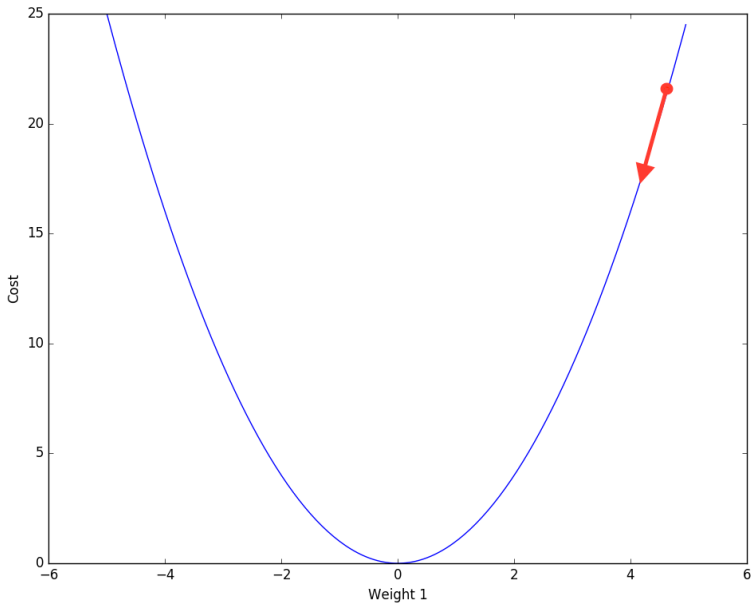
Subtasks:

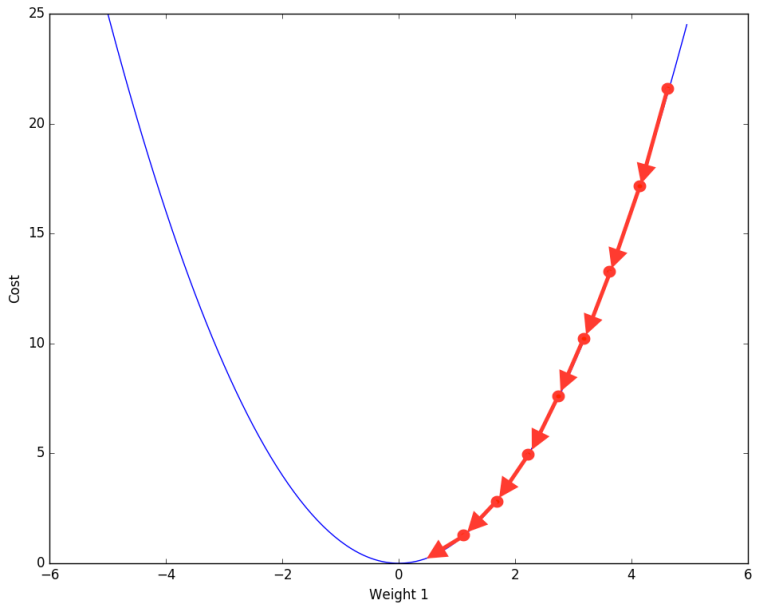
1. Forward pass to compute network output and "error"
2. Backward pass to compute gradients
3. A fraction of the weight's gradient is subtracted from the weight.

Learning Rate

Learning is an Optimization Problem

Task: Update the **weights** and **biases** to decrease **loss function**

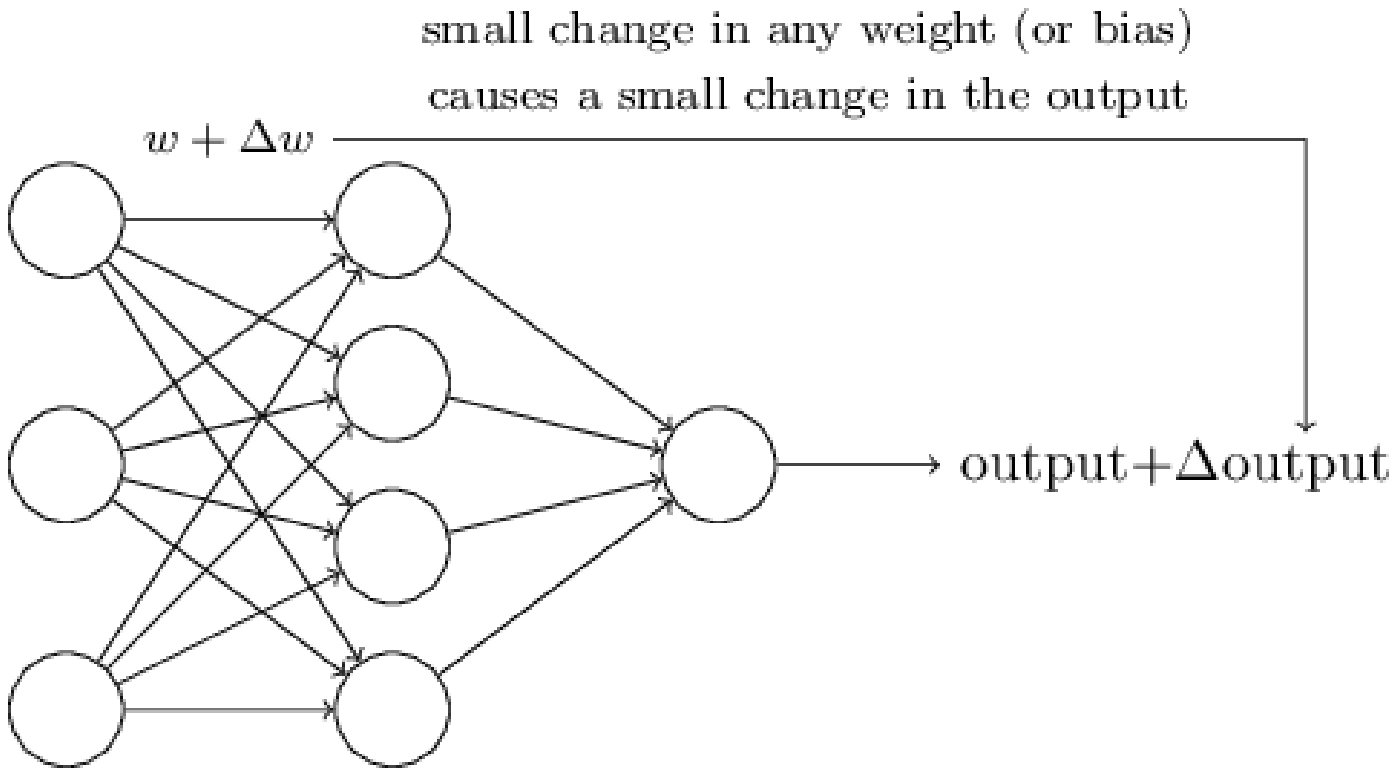




Use mini-batch or stochastic gradient descent.

The Process of Learning:

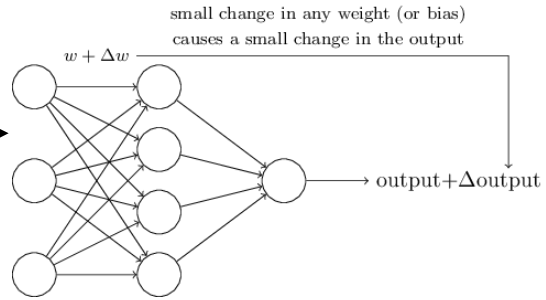
Small Change in Weights → Small Change in Output

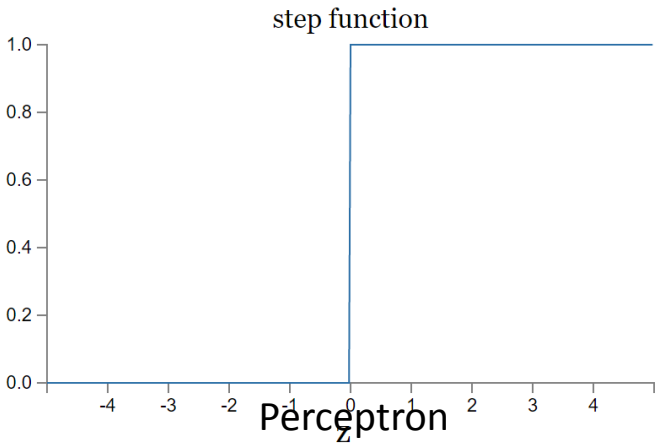


The Process of Learning:

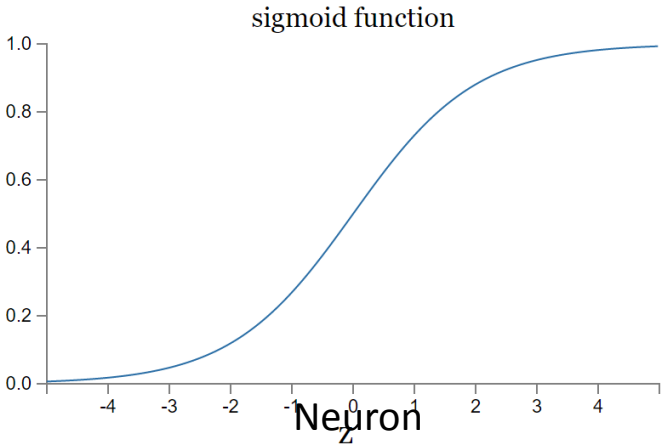
Small Change in Weights → Small Change in Output

This requires a "smoothness"





Perceptron



Neuron

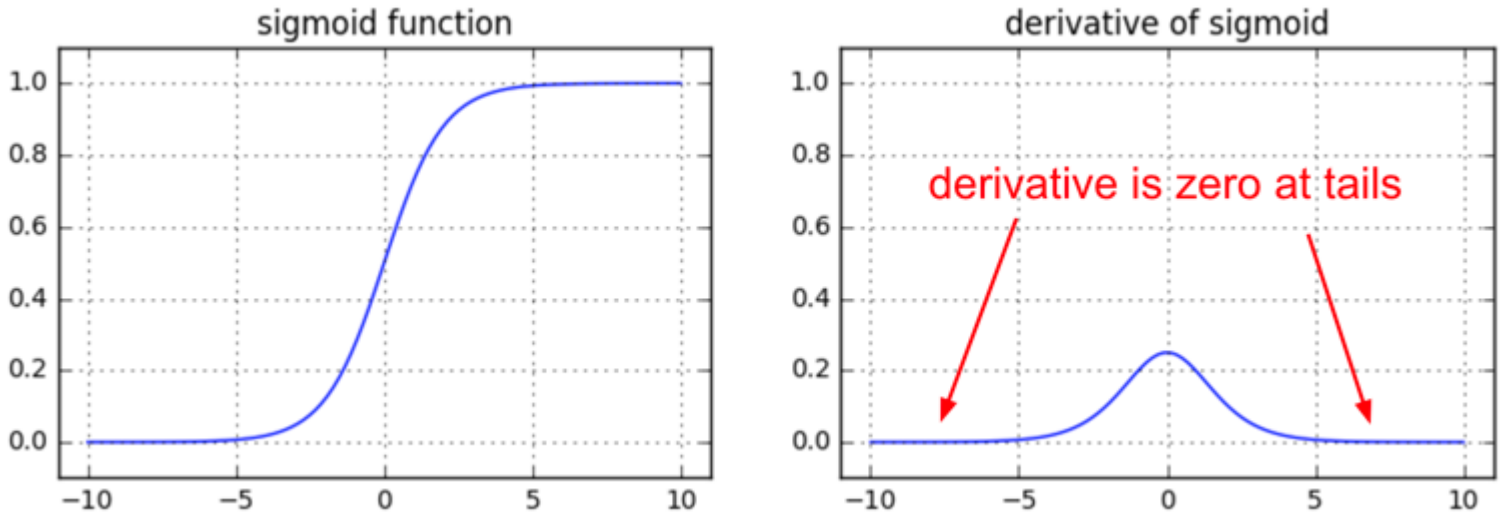
Smoothness of activation function means: **the Δoutput is a linear function of the Δweights and Δbias**

Learning is the process of gradually adjusting the weights to achieve any gradual change in the output.

Backpropogation in a One-layer NN

	Forward	Backward
Module 5	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Module 4	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Module 3	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$
Module 2	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Module 1	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \alpha_{ji}$

Optimization is Hard: Vanishing Gradients



$$\frac{d\sigma(x)}{dx} = (1 - \sigma(x))\,\sigma(x)$$

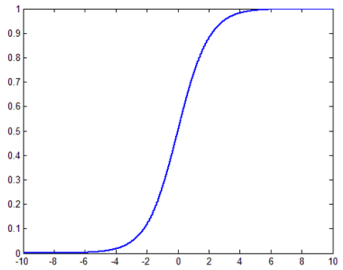
Partial derivatives are small = Learning is slow

Saturating Neurons with Vanishing Gradients:
Zero-ish gradients drives gradients in earlier layers to zero.

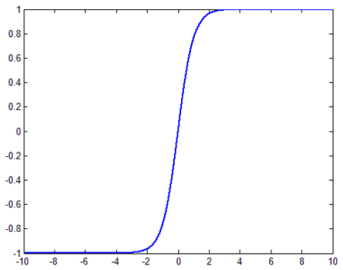
Activation Functions

- Applied on the hidden units
- Achieve nonlinearity
- Popular activation functions

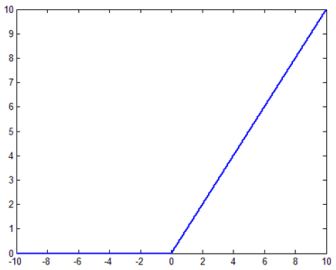
Sigmoid



Tanh

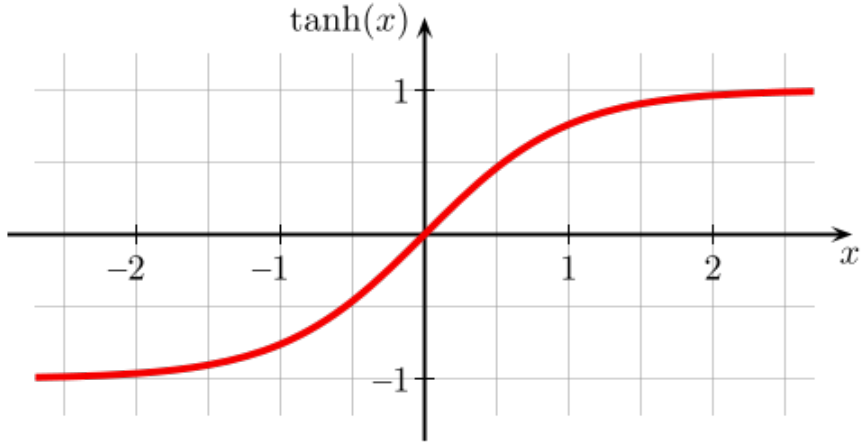


Rectified Linear



Activation Functions

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs

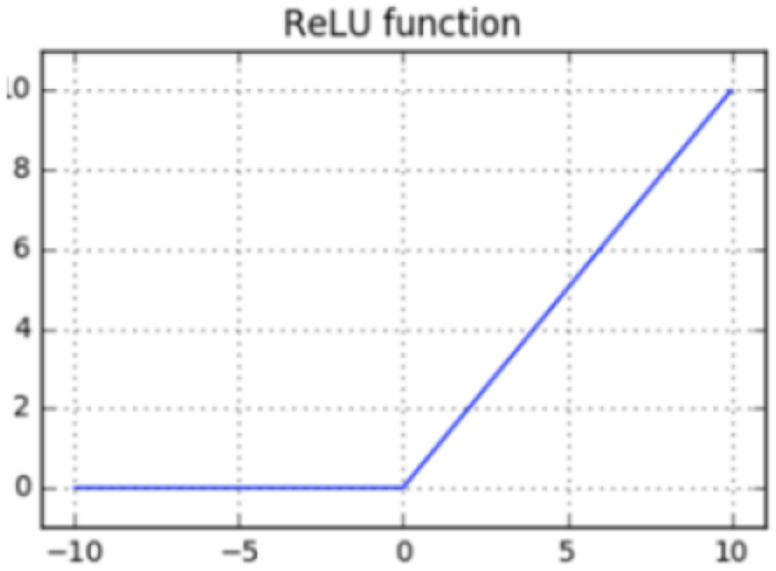


Alternate :
tanh

Like logistic function but shifted to range [-1, +1]

Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks



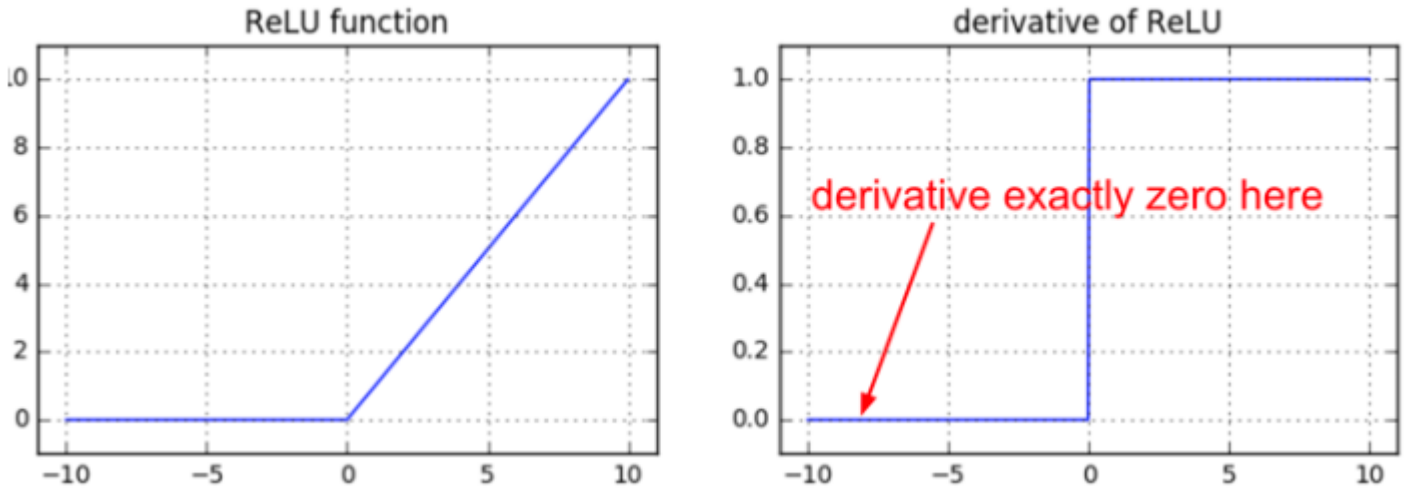
Alternate: rectified linear unit

Linear with a cutoff at zero

(Implementation: clip the gradient when you pass zero)

$$\max(0, w \cdot x + b).$$

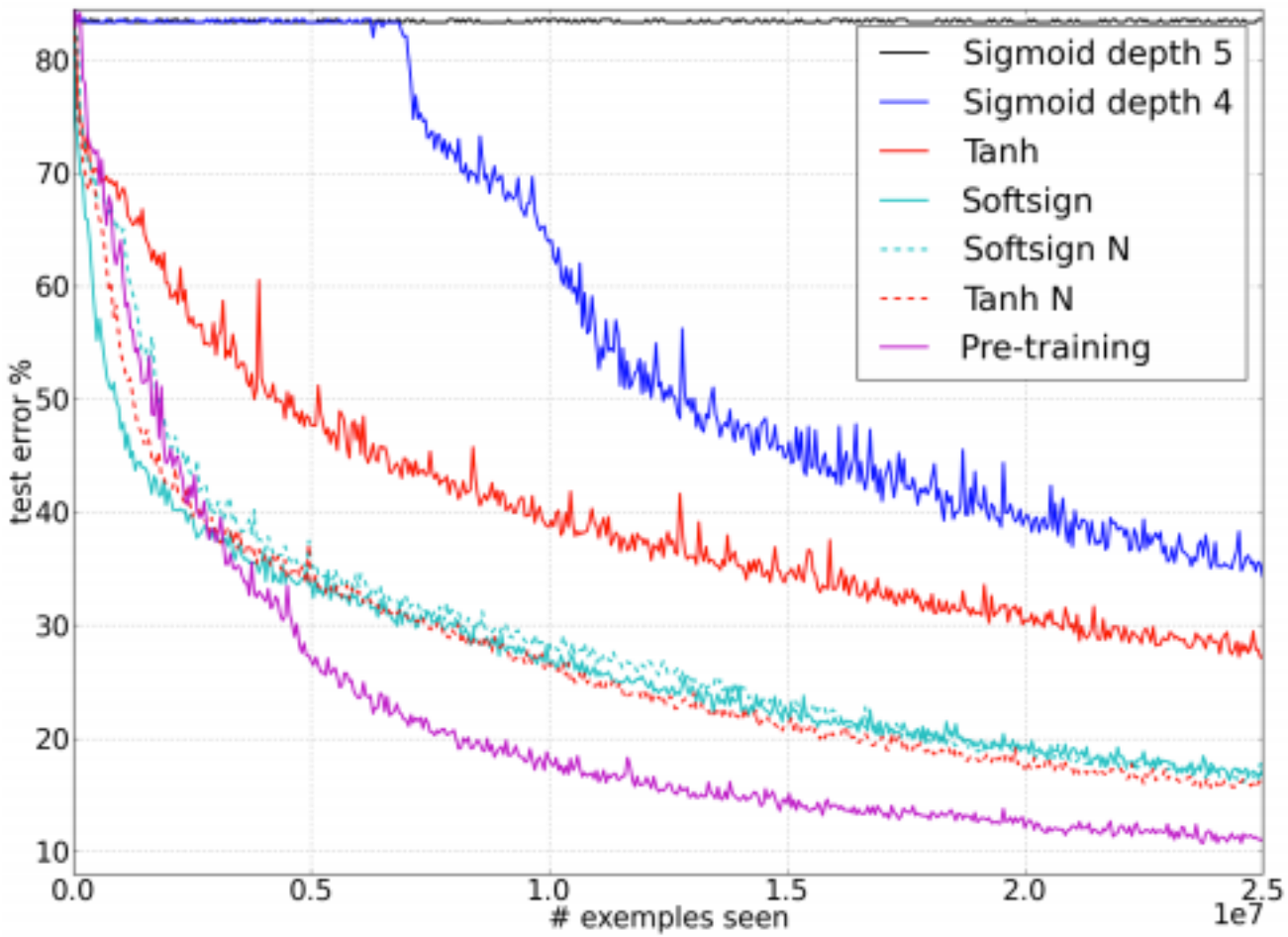
Optimization is Hard: Vanishing Gradients



- If a neuron is initialized poorly, it might not fire for entire training dataset.
- Large parts of your network could be dead ReLUs!

Saturating Neurons with Vanishing Gradients:
Zero-ish gradients drives gradients in earlier layers to zero.

Understanding the difficulty of training deep feedforward neural networks

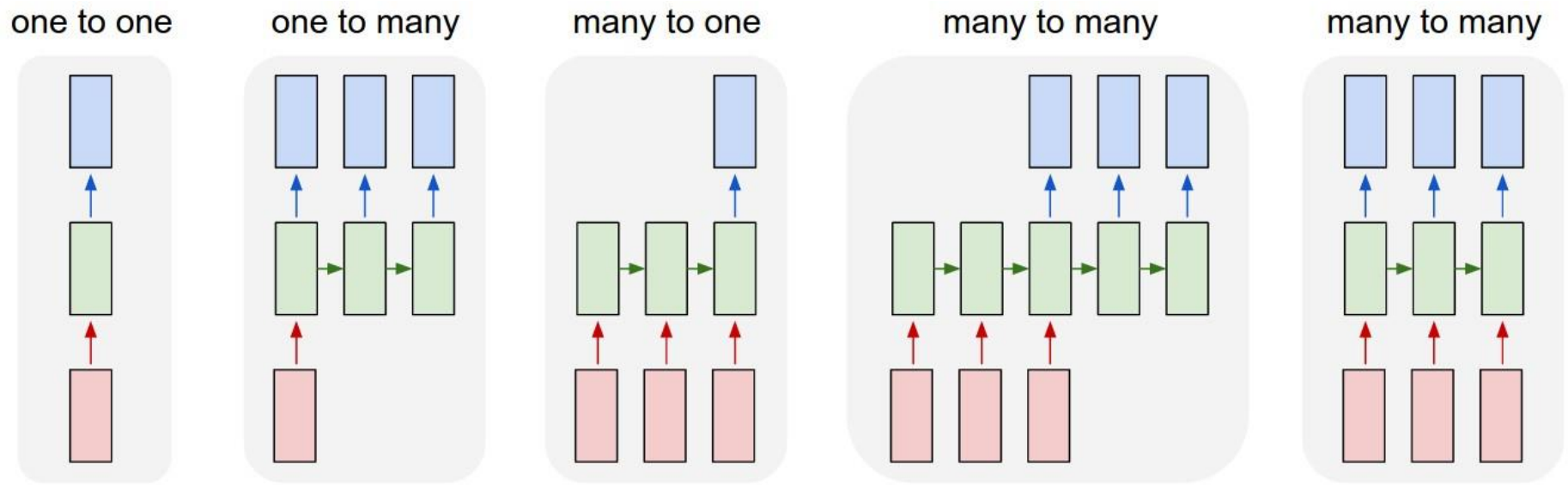


AI Stats

sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

Back to Recurrent Neural Networks (RNNs)



"Vanilla" Neural Networks

Recurrent Neural Networks

RNN's are **amazing**. But tricky to train.

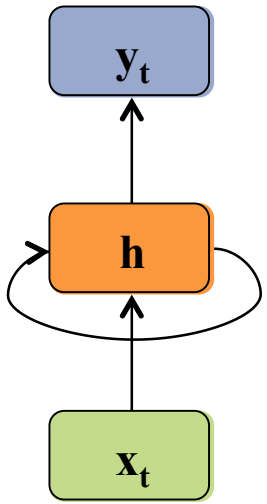
Recurrent Neural Networks (RNNs)

$$\begin{aligned}
\text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity: } & \mathcal{H}
\end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



Memory →

- **Input *(x)*:** (example: word of a sentence)
- **Hidden state *(h)*:** function of previous hidden state and new input
- **Output *(y)*:** (example: predict next word in the sentence)

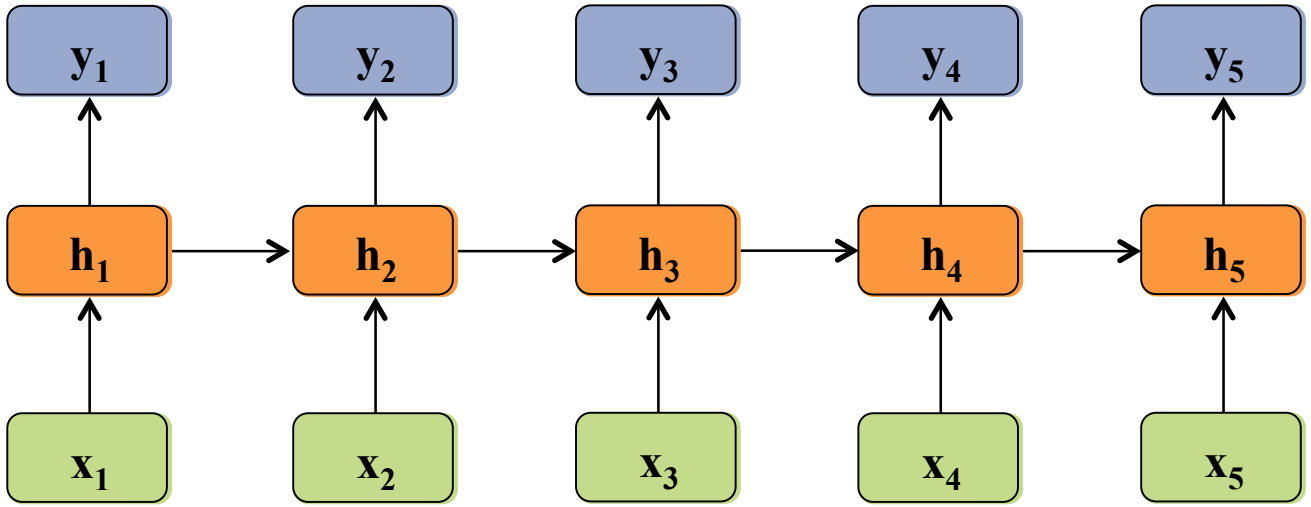
Recurrent Neural Networks (RNNs)

$$\begin{aligned}
\text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity: } & \mathcal{H}
\end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



- Parameters W_{hh}, W_{xh}, W_{hy} are shared across time
  - Similar to CNNs: this reduces the # of parameters we need to optimize
  - And it allow us to process arbitrary "temporal size" of input
- Process is the same for any input / output mapping:

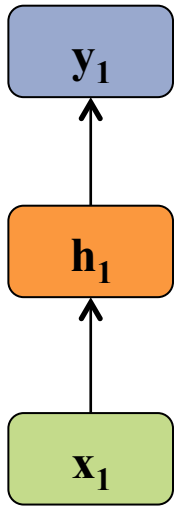
Recurrent Neural Networks (RNNs)

$$\begin{aligned}
\text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity: } & \mathcal{H}
\end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



If $T=1$, then we have a standard feed-forward **neural net with one hidden layer**

Recurrent Neural Networks (RNNs)

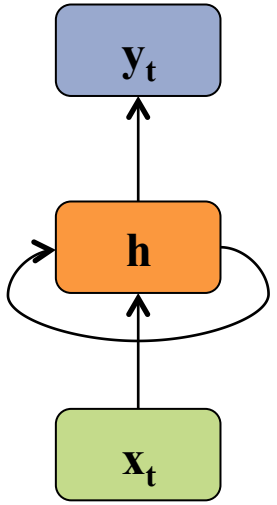
$$\begin{aligned}
\text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity: } & \mathcal{H}
\end{aligned}$$

Definition of the RNN:

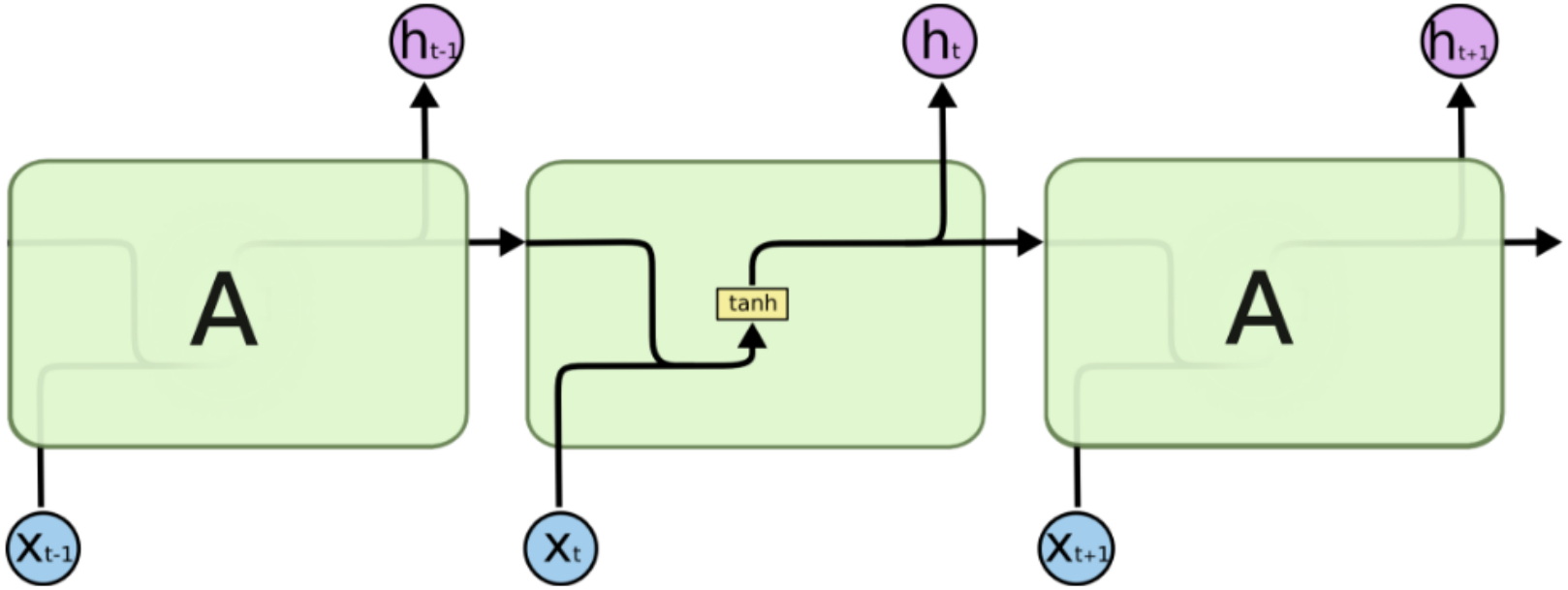
$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$

- By unrolling the RNN through time, we can **share parameters** and accommodate **arbitrary length** input/output pairs
- Applications: **time-series data** such as sentences, speech, stock-market, signal data, etc.

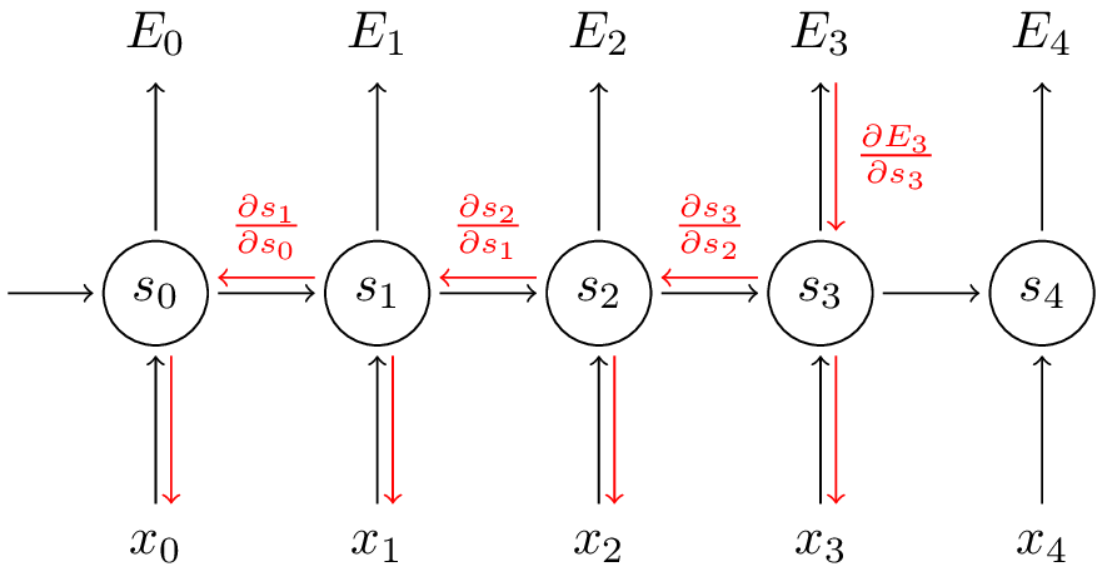


Vanilla RNN



Backpropagation Through Time (BPTT)

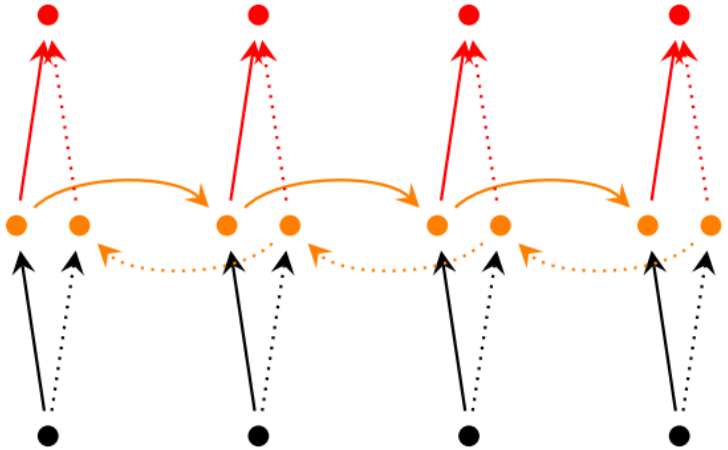
(Fancy name for regular backpropagation on an unrolled RNN)



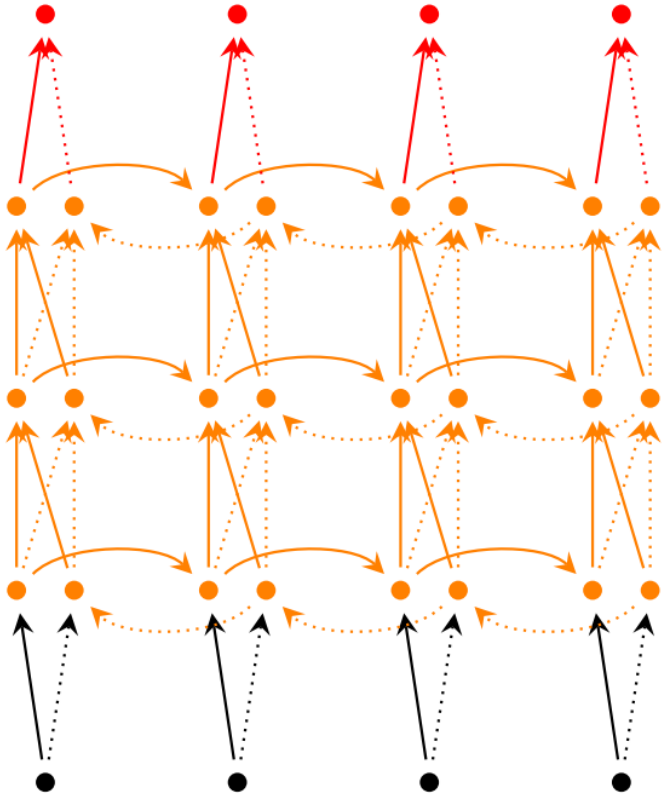
Saturating Neurons with Vanishing Gradients:
Zero-ish gradients drives gradients in earlier layers to zero.

RNN Variants: Bidirectional RNNs

(Shallow)



(Deep)



- Example:
  - Filling in missing words
- Deeper =
  - more learning capacity
  - but needs lots of training data

Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

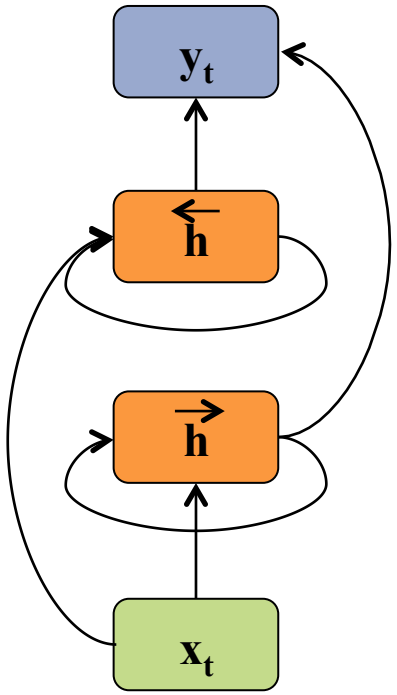
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

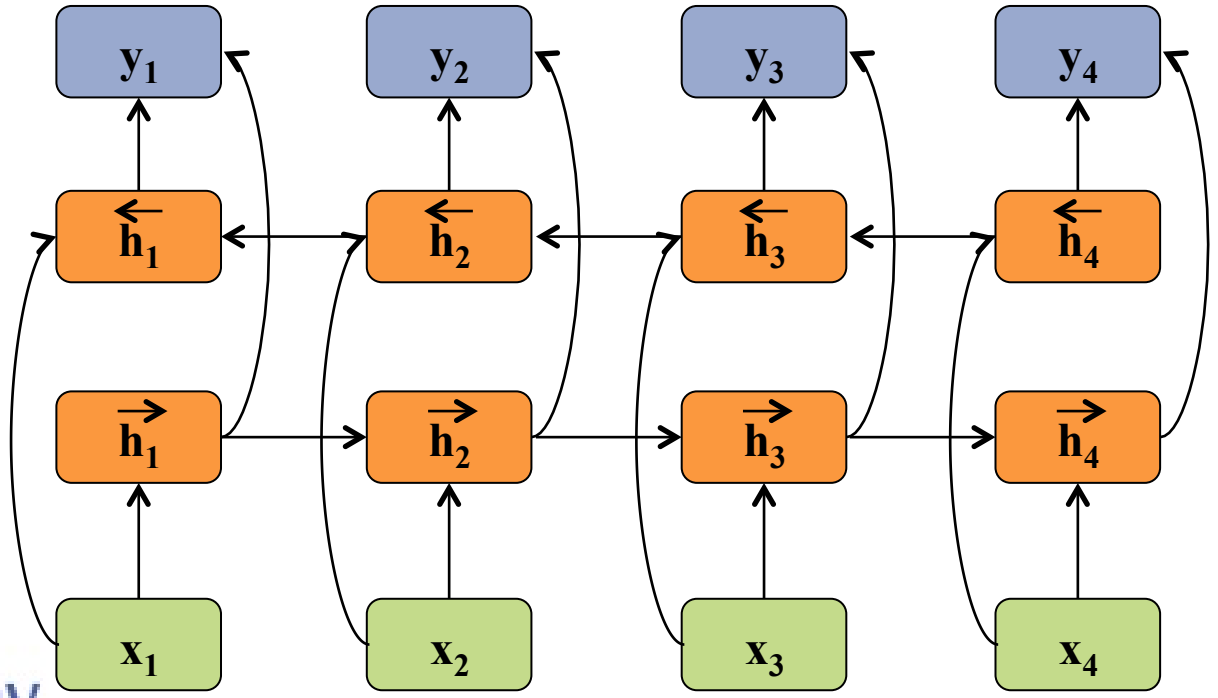
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Deep RNNs

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

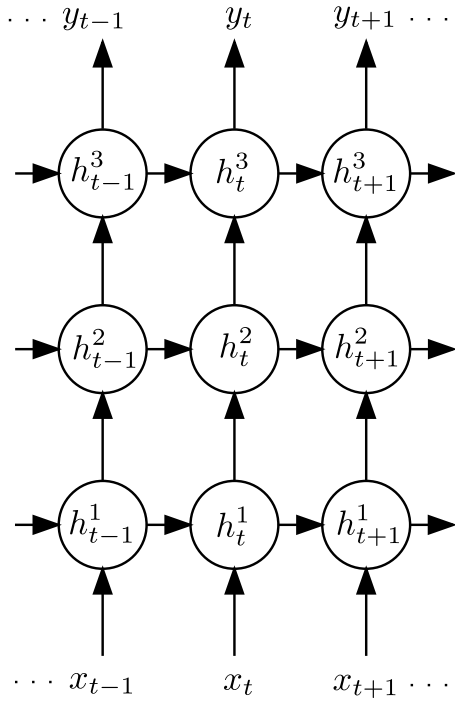
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Recursive Definition:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right)$$

$$y_t = W_{h^N y} h_t^N + b_y$$



AI Academy

Figure from (Graves et al.)

NC STATE

Deep Bidirectional RNNs

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

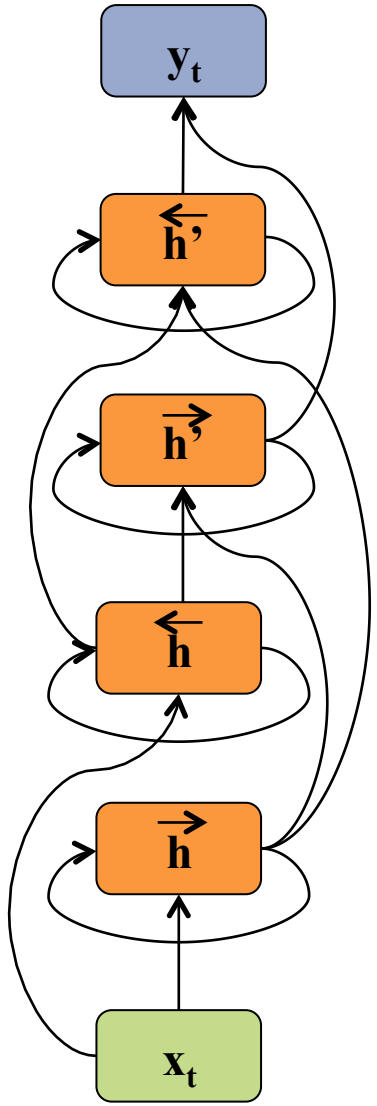
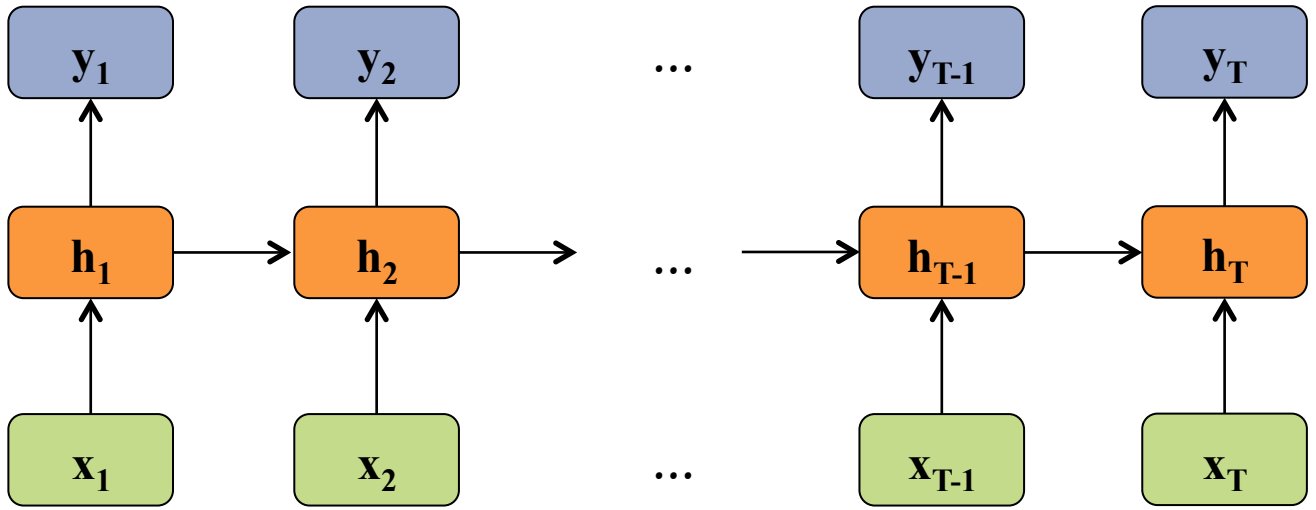


Figure from (Graves et al.)

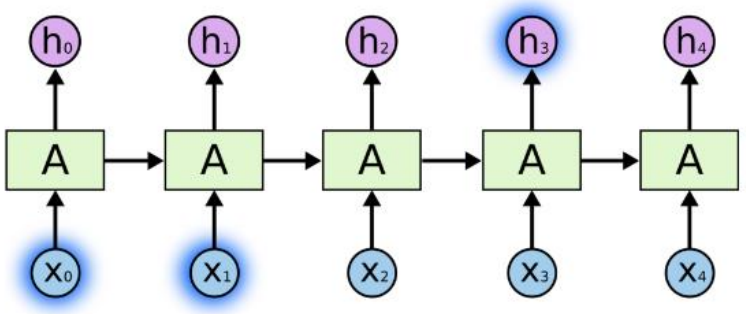
Long Short-Term Memory (LSTM)

Motivation:

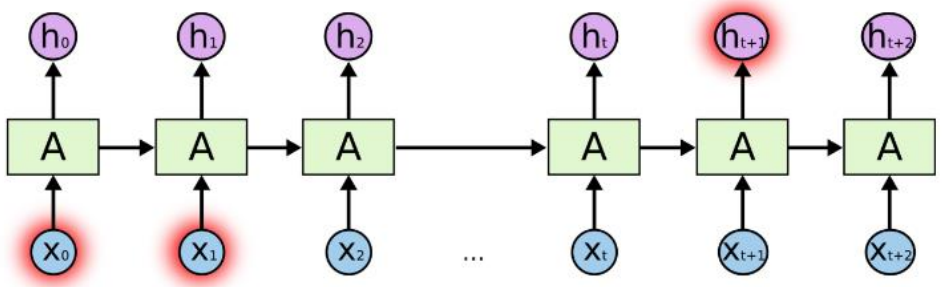
- Standard RNNs have trouble learning long distance dependencies
- LSTMs combat this issue



Long-Term Dependency



- Short-term dependence:
  Bob is eating an **apple.**
- Long-term dependence:
  Context → **Bob** likes **apples**. He is hungry and decided to have a snack. So now he is eating an **apple.**



In theory, vanilla RNNs can handle arbitrarily long-term dependence.

In practice, it's difficult.

Long Short-Term Memory (LSTM)

Motivation:

- Vanishing gradient problem for Standard RNNs
- Figure shows sensitivity (darker = more sensitive) to the input at time t=1

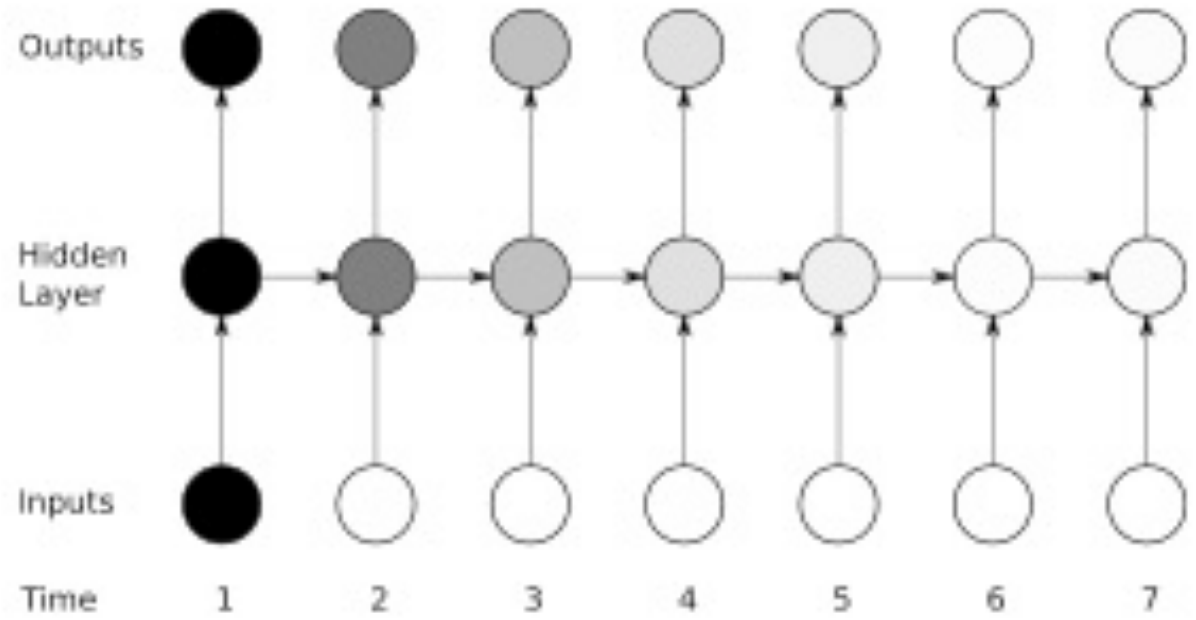


Figure from (Graves)

Long Short-Term Memory (LSTM)

Motivation:

- LSTM units have a rich internal structure
- The various "gates" determine the propagation of information and can choose to "remember" or "forget" information

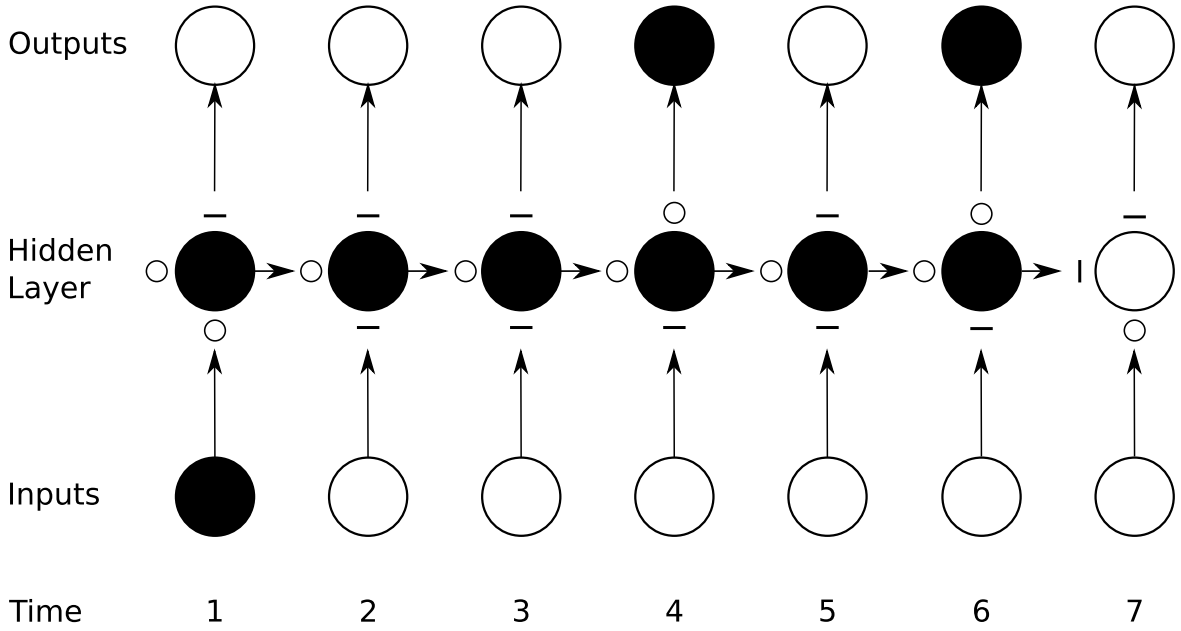
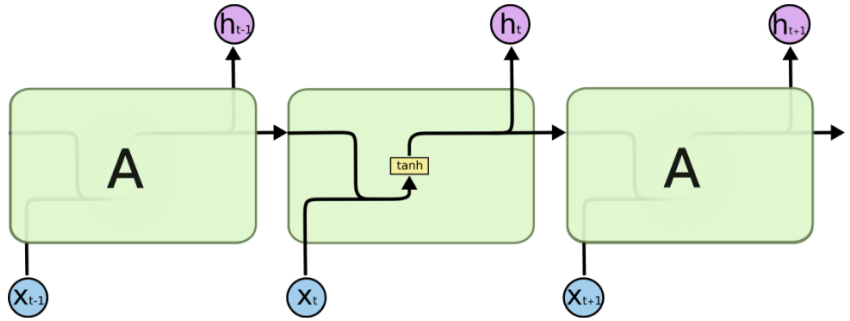


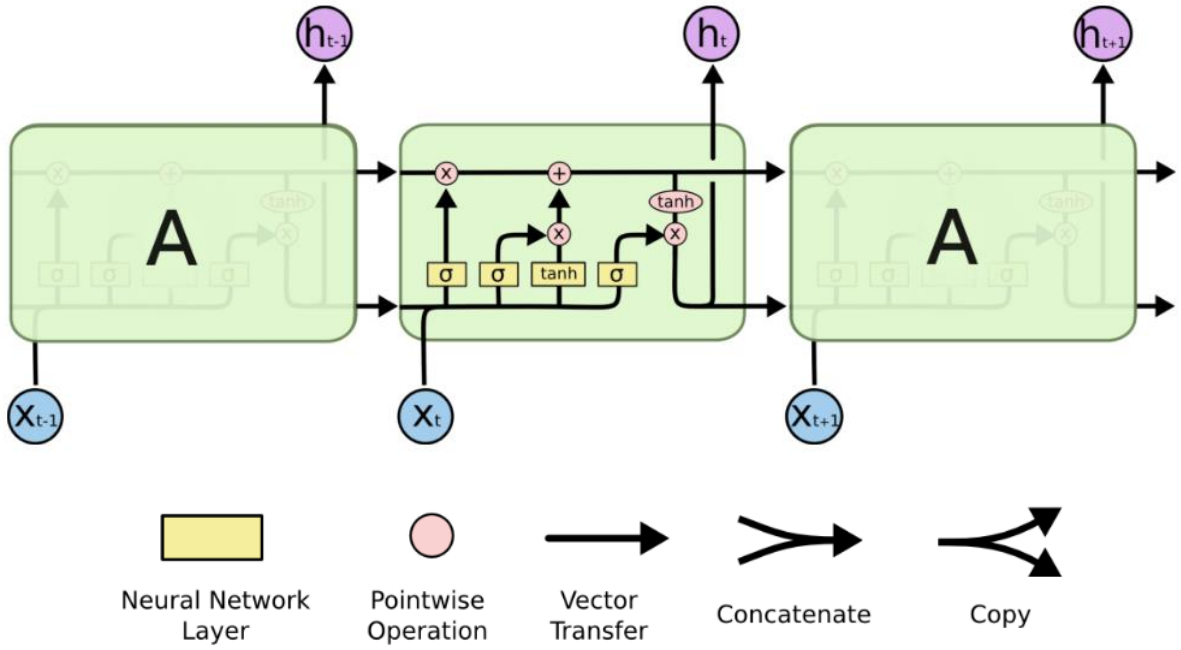
Figure from (Graves)

Long Short Term Memory (LSTM) Networks

Vanilla RNN:

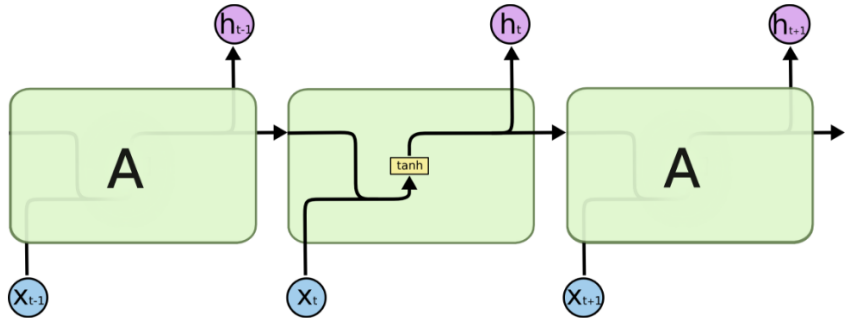


LSTM:

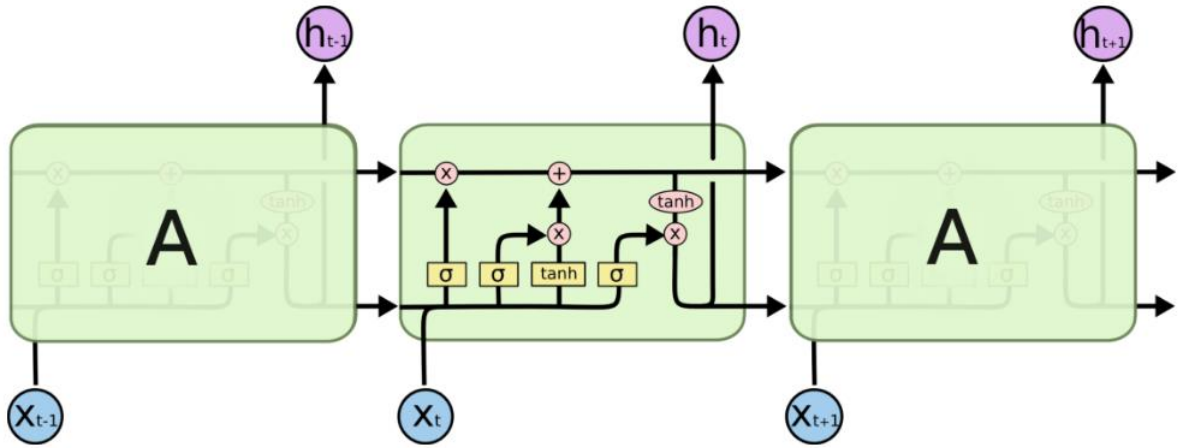


LSTM: Gates Regulate

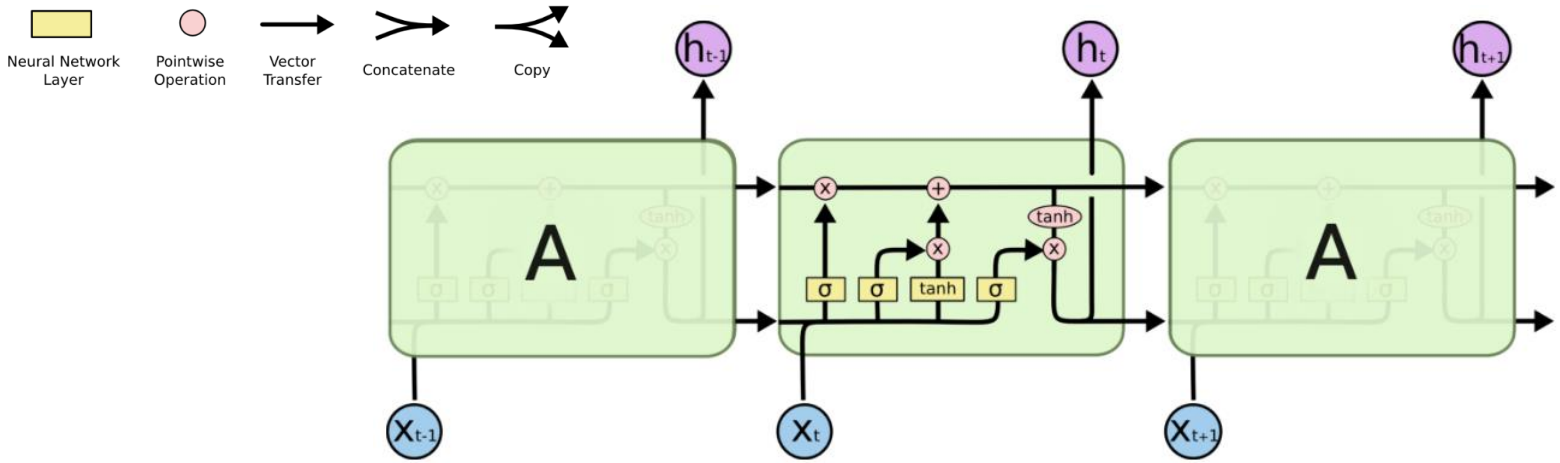
Vanilla RNN:



LSTM:



LSTM: Pick What to Forget and What To Remember

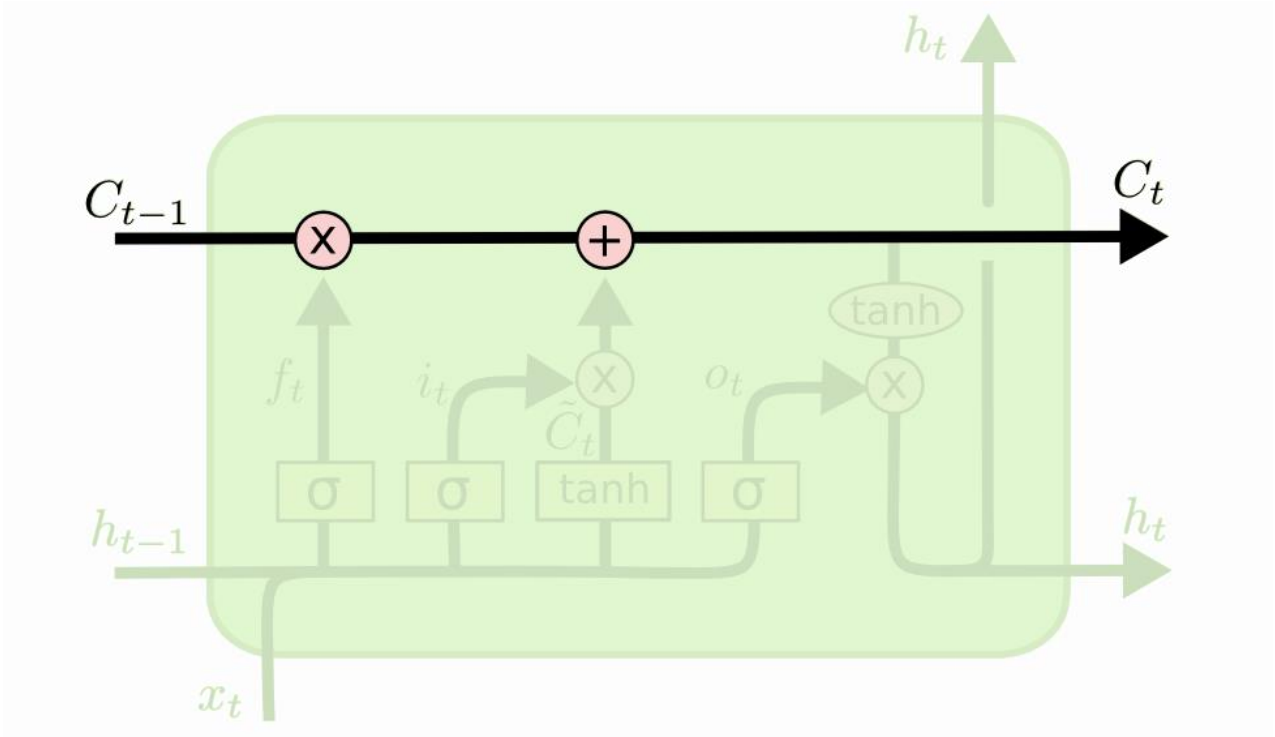


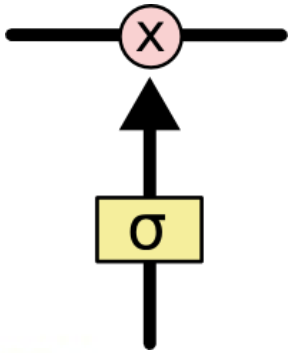
Bob and Alice are having lunch. Bob likes apples. Alice likes oranges. **She is eating an orange.**

Conveyer belt for **previous state** and **new data**:

1. Decide what to forget (state)
2. Decide what to remember (state)
3. Decide what to output (if anything)

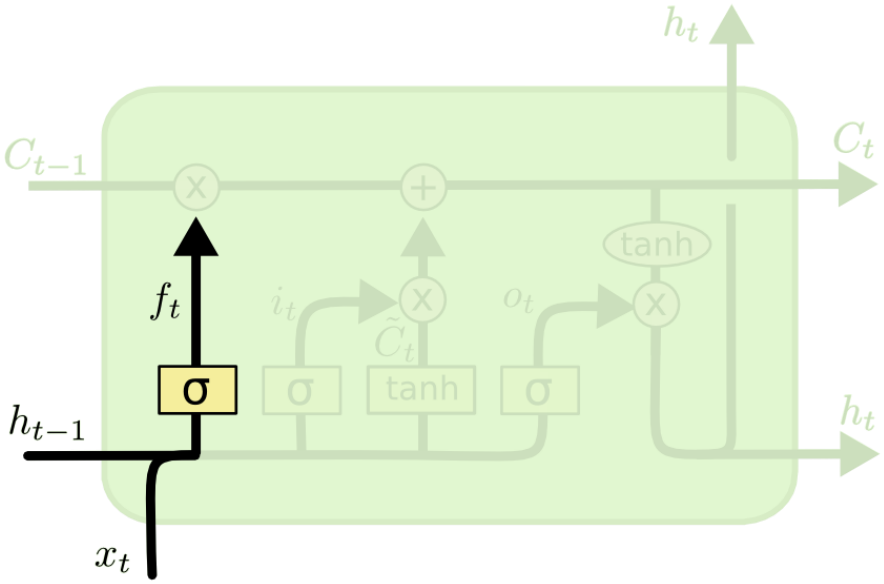
LSTM Conveyer Belt





- State run through the cell
- 3 sigmoid layers output deciding which information is let through (~1) and which is not (~0)

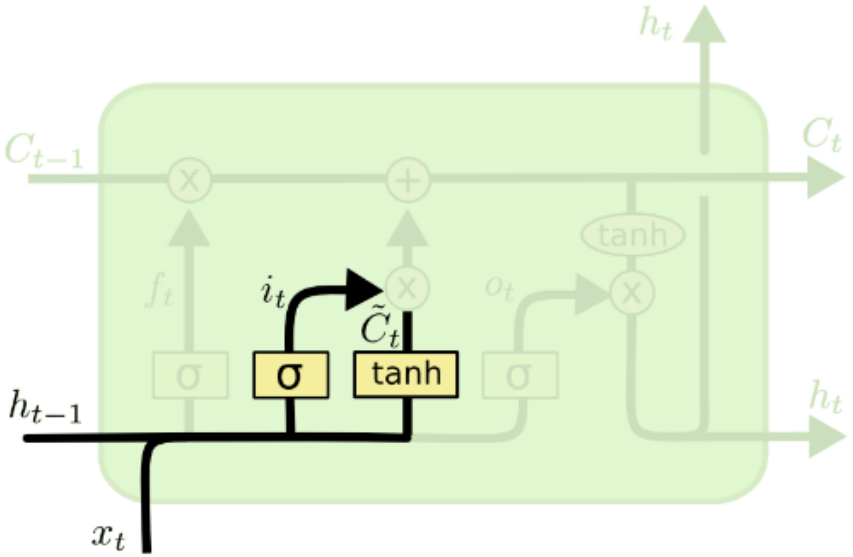
LSTM Conveyer Belt



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

Step 1: Decide what to forget / ignore

LSTM Conveyer Belt

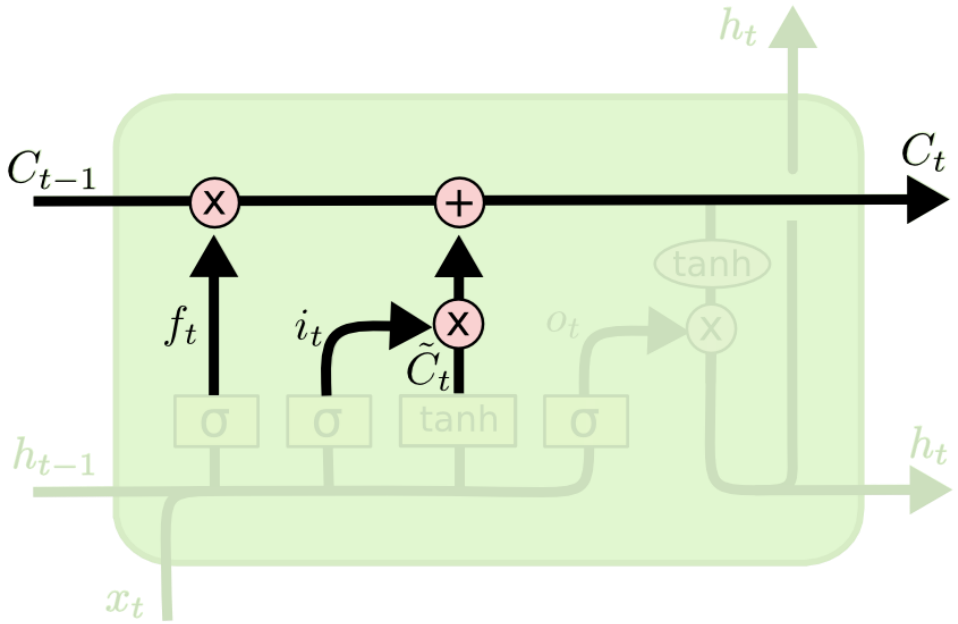


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Step 2: Decide which state values to update (w/ sigmoid) and what values to update with (w/ tanh)

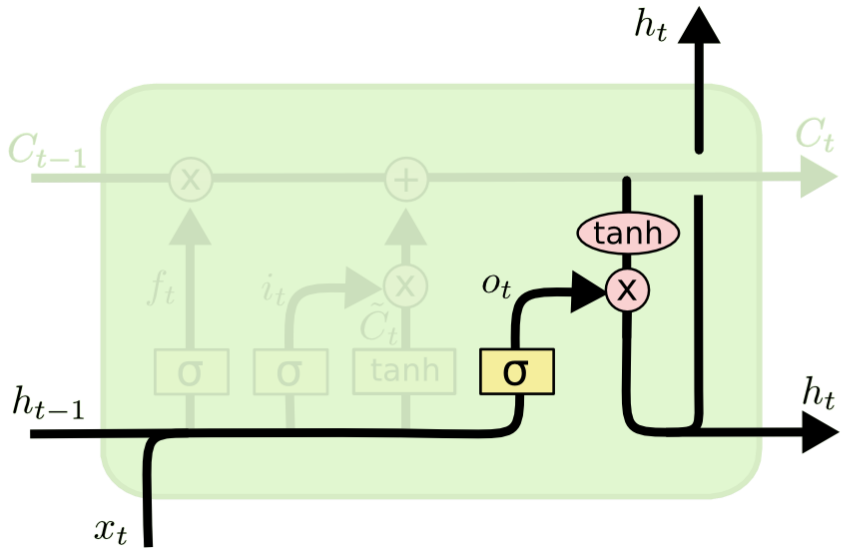
LSTM Conveyer Belt



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

Step 3: Perform the forgetting and the state update

LSTM Conveyer Belt

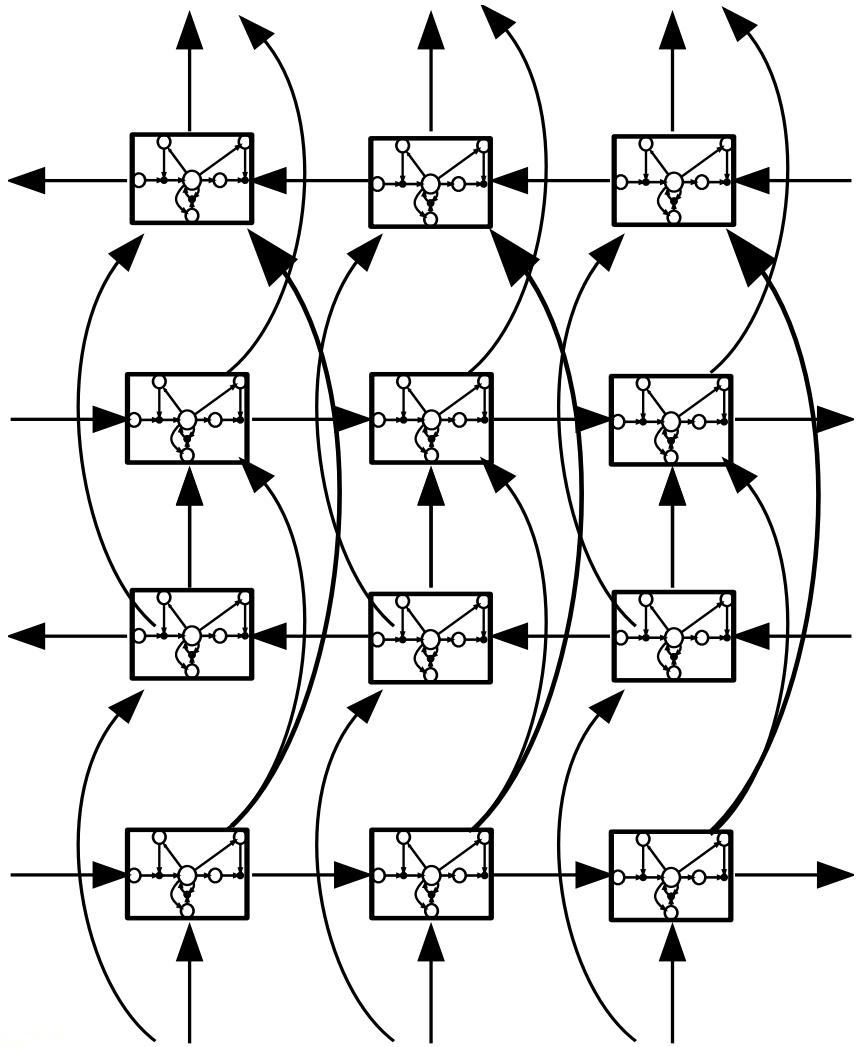


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Step 4: Produce output with tanh [-1, 1] deciding the values and sigmoid [0, 1] deciding the filtering

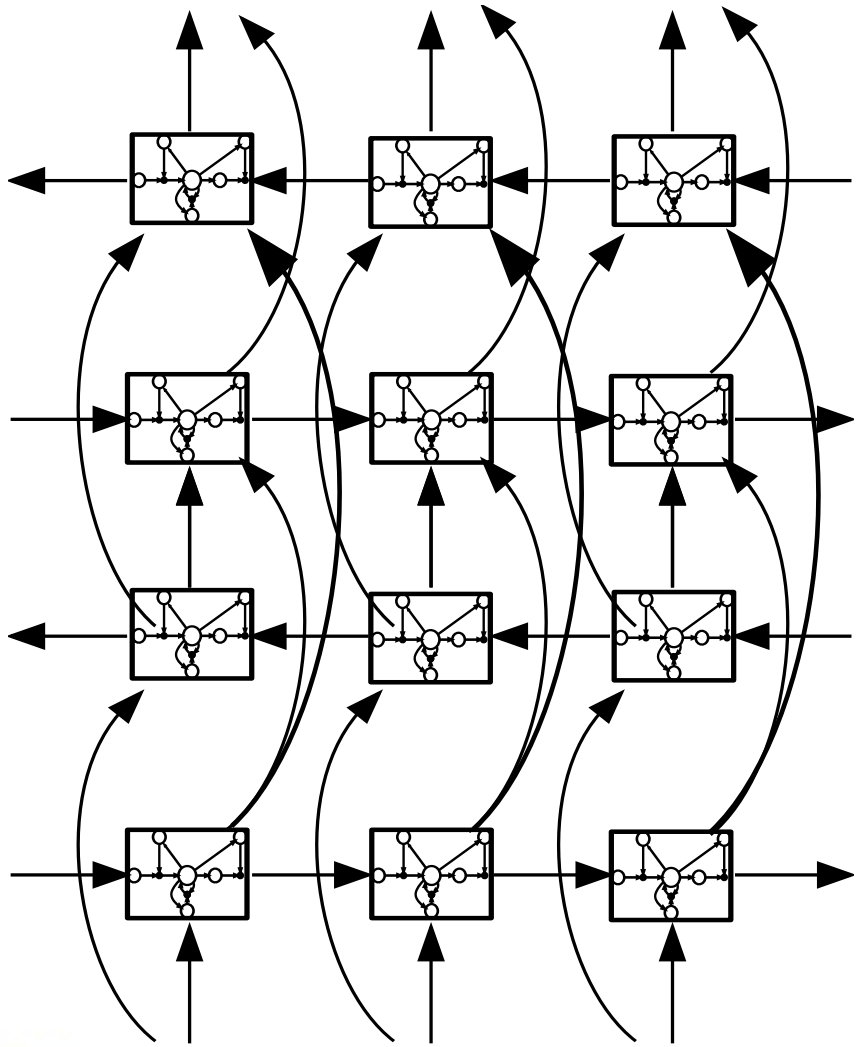
Deep Bidirectional LSTM (DBLSTM)



- Figure: input/output layers not shown
- **Same general topology** as a Deep Bidirectional RNN, but with **LSTM units** in the hidden layers
- No additional **representational power** over DBRNN, but **easier to learn** in practice

Figure from (Graves et al.)

Deep Bidirectional LSTM (DBLSTM)

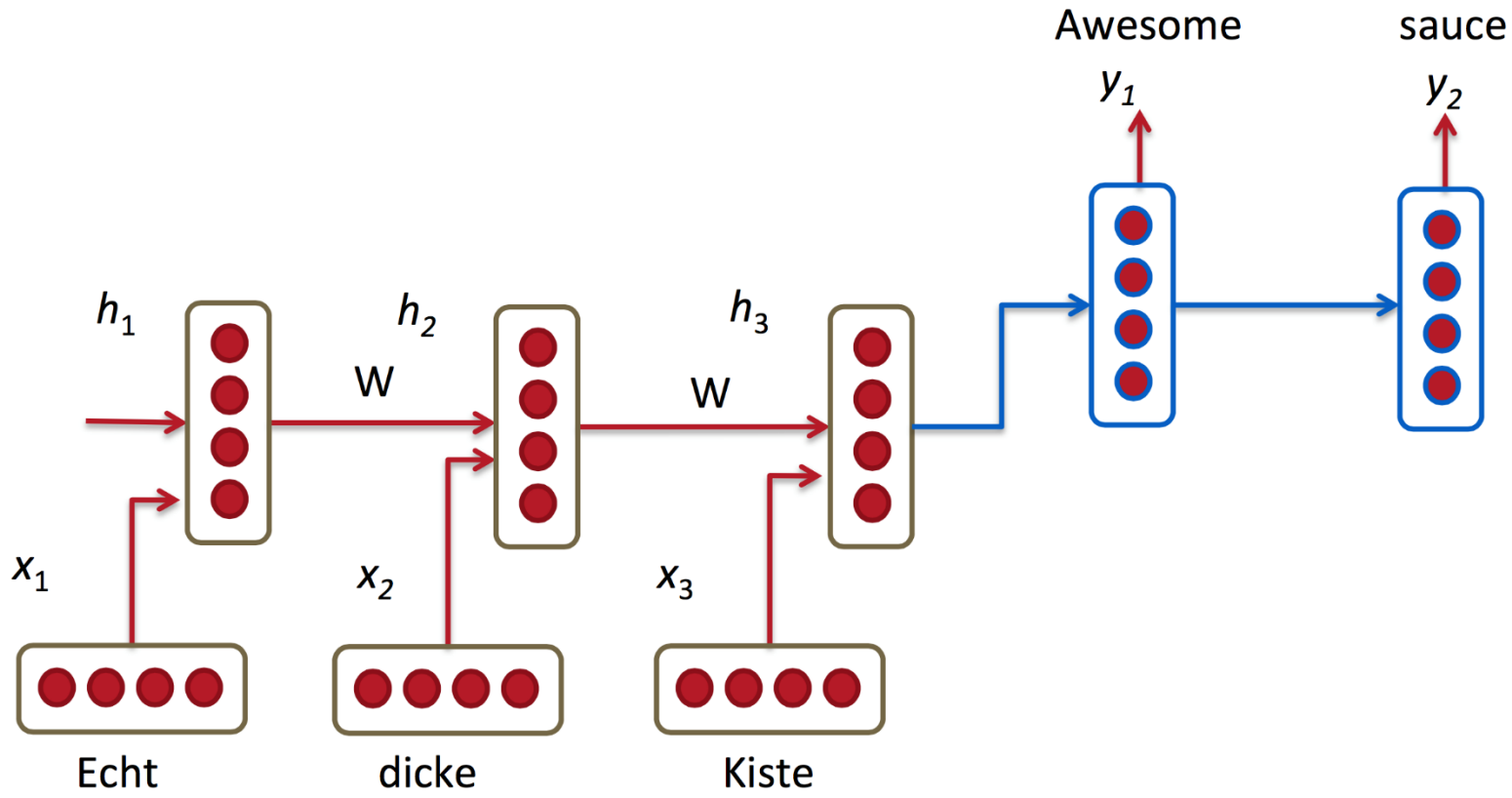


How important is this particular architecture?

Jozefowicz et al. (2015) **evaluated 10,000 different LSTM-like architectures** and found several variants that worked just as well on several tasks.

Figure from (Graves et al.)

Application: Machine Translation



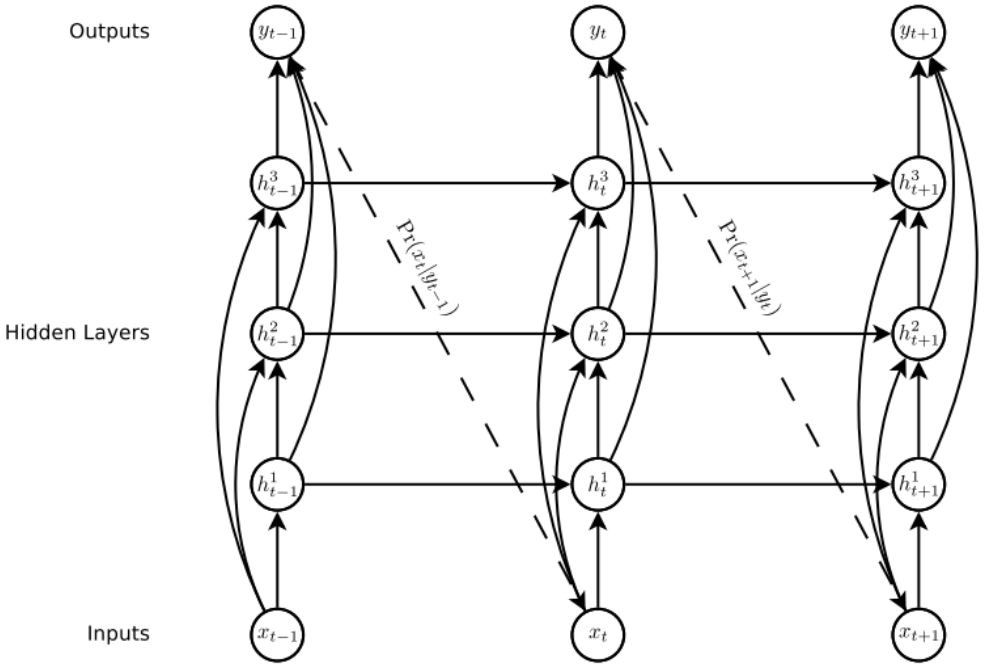
Application: Handwriting Generation from Text

Text --- up to 100 characters, lower case letters work best

Input:

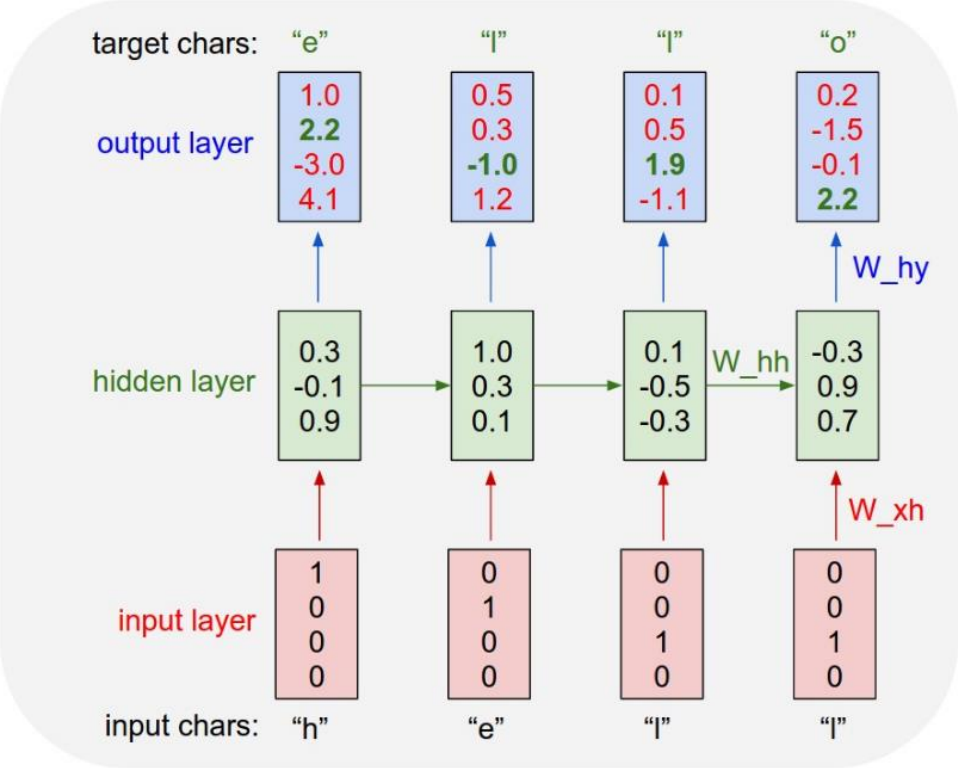
Deep Learning

Output: Deep Learning



Alex Graves. **"Generating sequences with recurrent neural networks."** .

Application: Character-Level Text Generation



Life Is About The Weather!
Life Is About The True Love Of Mr. Mom
Life Is About Kids
Life Is About An Eating Story
Life Is About The Truth Now

The meaning of life is
literary recognition

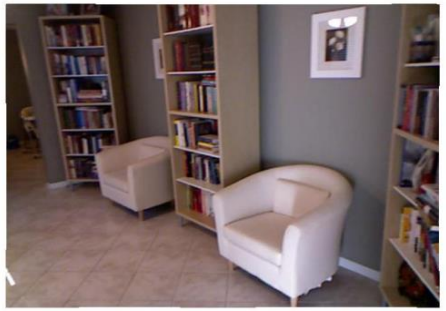
The meaning of life is
the tradition of the ancient human reproduction

Andrej Karpathy. "The Unreasonable Effectiveness of Recurrent Neural Networks.".

Application: Image Question Answering

COCOQA 33827
What is the color of the cat?
Ground truth: black
IMG+BOW: black (0.55)
2-VIS+LSTM: black (0.73)
BOW: gray (0.40)

COCOQA 33827a
What is the color of the couch?
Ground truth: red
IMG+BOW: red (0.65)
2-VIS+LSTM: black (0.44)
BOW: red (0.39)



DAQUAR 1522
How many chairs are there?
Ground truth: two
IMG+BOW: four (0.24)
2-VIS+BLSTM: one (0.29)
LSTM: four (0.19)

DAQUAR 1520
How many shelves are there?
Ground truth: three
IMG+BOW: three (0.25)
2-VIS+BLSTM: two (0.48)
LSTM: two (0.21)

COCOQA 14855
Where are the ripe bananas sitting?
Ground truth: basket
IMG+BOW: basket (0.97)
2-VIS+BLSTM: basket (0.58)
BOW: bowl (0.48)

COCOQA 14855a
What are in the basket?
Ground truth: bananas
IMG+BOW: bananas (0.98)
2-VIS+BLSTM: bananas (0.68)
BOW: bananas (0.14)

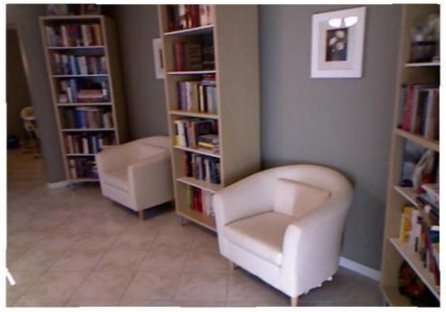
DAQUAR 585
What is the object on the chair?
Ground truth: pillow
IMG+BOW: clothes (0.37)
2-VIS+BLSTM: pillow (0.65)
LSTM: clothes (0.40)

DAQUAR 585a
Where is the pillow found?
Ground truth: chair
IMG+BOW: bed (0.13)
2-VIS+BLSTM: chair (0.17)
LSTM: cabinet (0.79)

Application: Image Question Answering

COCOQA 33827
What is the color of the cat?
Ground truth: black
IMG+BOW: black (0.55)
2-VIS+LSTM: black (0.73)
BOW: gray (0.40)

COCOQA 33827a
What is the color of the couch?
Ground truth: red
IMG+BOW: red (0.65)
2-VIS+LSTM: black (0.44)
BOW: red (0.39)



DAQUAR 1522
How many chairs are there?
Ground truth: two
IMG+BOW: four (0.24)
2-VIS+BLSTM: one (0.29)
LSTM: four (0.19)

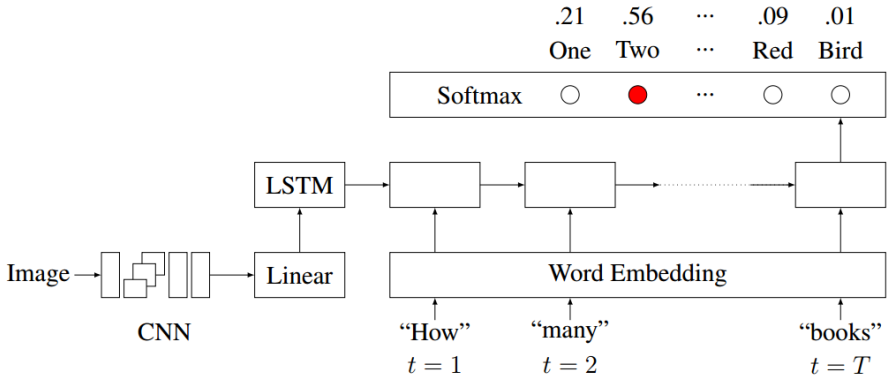
DAQUAR 1520
How many shelves are there?
Ground truth: three
IMG+BOW: three (0.25)
2-VIS+BLSTM: two (0.48)
LSTM: two (0.21)

COCOQA 14855
Where are the ripe bananas sitting?
Ground truth: basket
IMG+BOW: basket (0.97)
2-VIS+BLSTM: basket (0.58)
BOW: bowl (0.48)

COCOQA 14855a
What are in the basket?
Ground truth: bananas
IMG+BOW: bananas (0.98)
2-VIS+BLSTM: bananas (0.68)
BOW: bananas (0.14)

DAQUAR 585
What is the object on the chair?
Ground truth: pillow
IMG+BOW: clothes (0.37)
2-VIS+BLSTM: pillow (0.65)
LSTM: clothes (0.40)

DAQUAR 585a
Where is the pillow found?
Ground truth: chair
IMG+BOW: bed (0.13)
2-VIS+BLSTM: chair (0.17)
LSTM: cabinet (0.79)



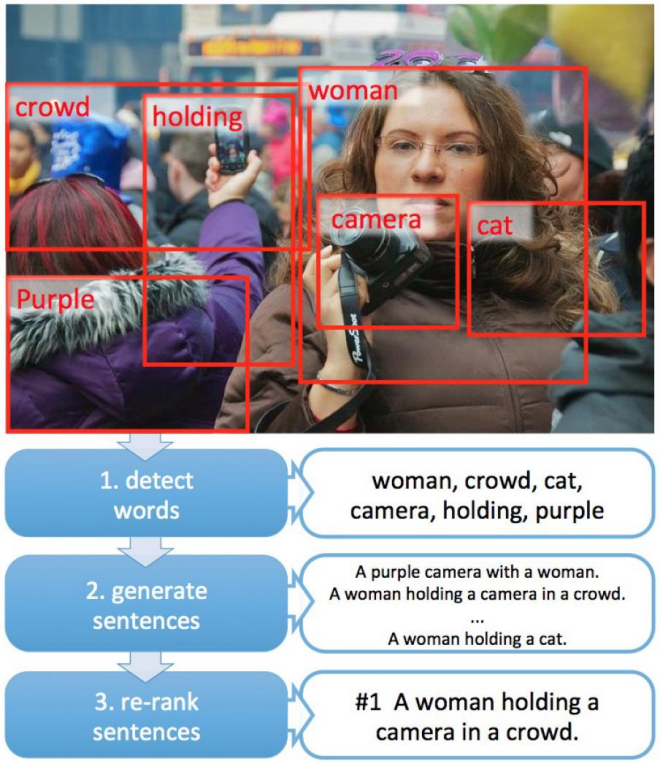
Ren et al. "Exploring models and data for image question answering."

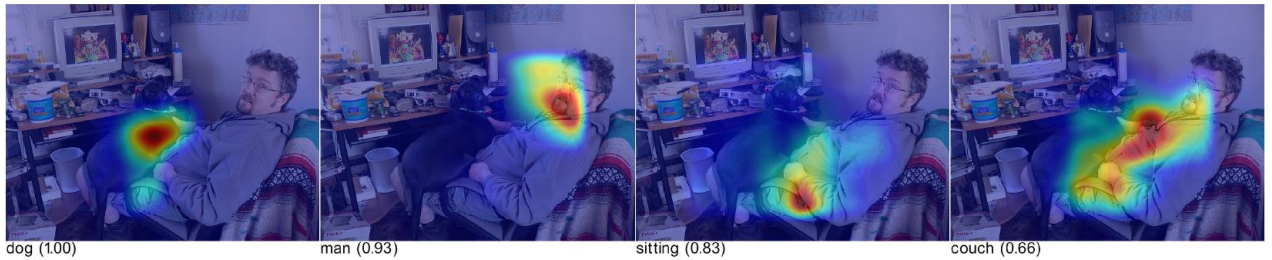
Code: https://github.com/renmengye/imageqa-public

Application: Image Caption Generation

a man sitting on a couch with a dog

a man sitting on a chair with a dog in his lap

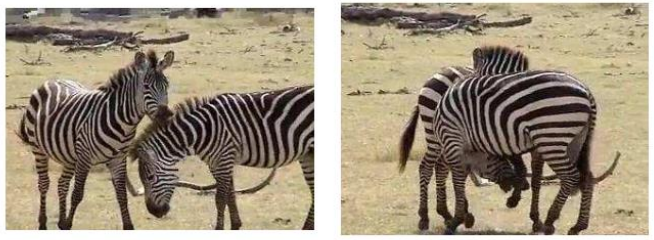




Application: Video Description Generation

Correct descriptions.

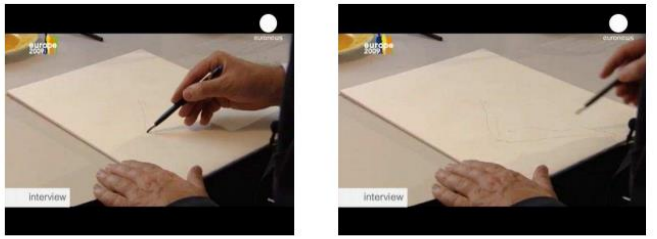
S2VT: A man is doing stunts on his bike.



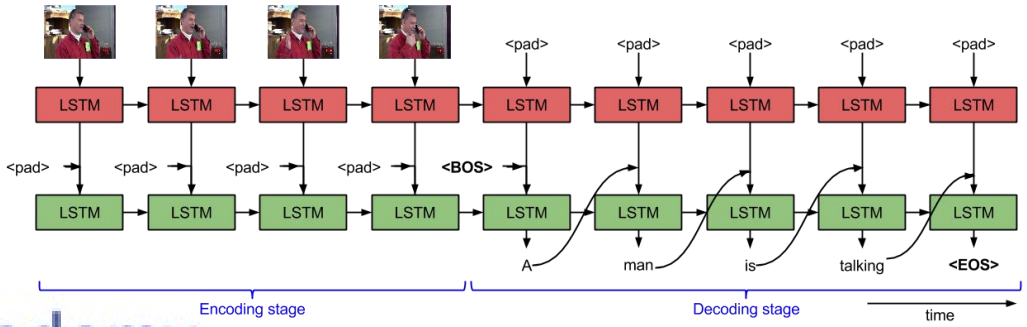
S2VT: A herd of zebras are walking in a field.

Relevant but incorrect descriptions.

S2VT: A small bus is running into a building.



S2VT: A man is cutting a piece of a pair of a paper.

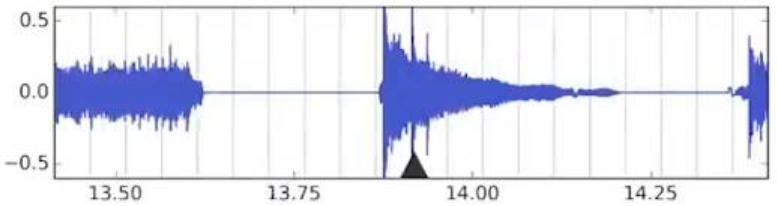


Venugopalan et al.
"Sequence to sequence-video to text."

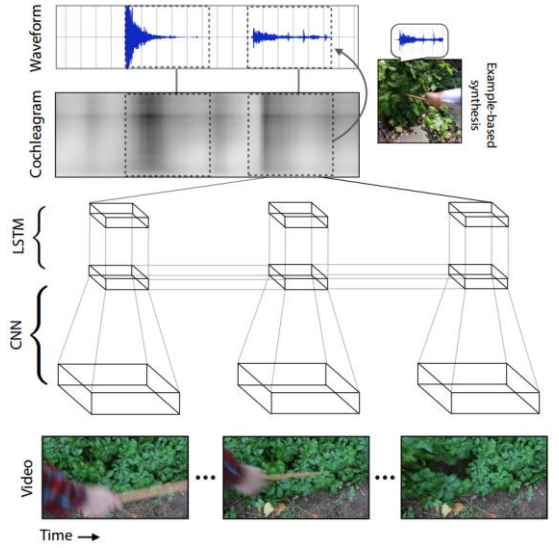
Code: https://vsubhashini.github.io/s2vt.html

Application: Adding Audio to Silent Film

Silent video

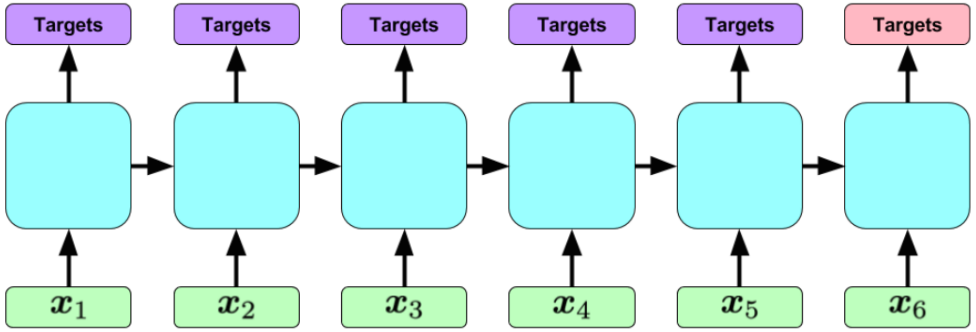


Predicted soundtrack



Owens, Andrew, Phillip Isola, Josh McDermott, Antonio Torralba, Edward H. Adelson, and William T. Freeman. **"Visually Indicated Sounds."**

Application: Medical Diagnosis



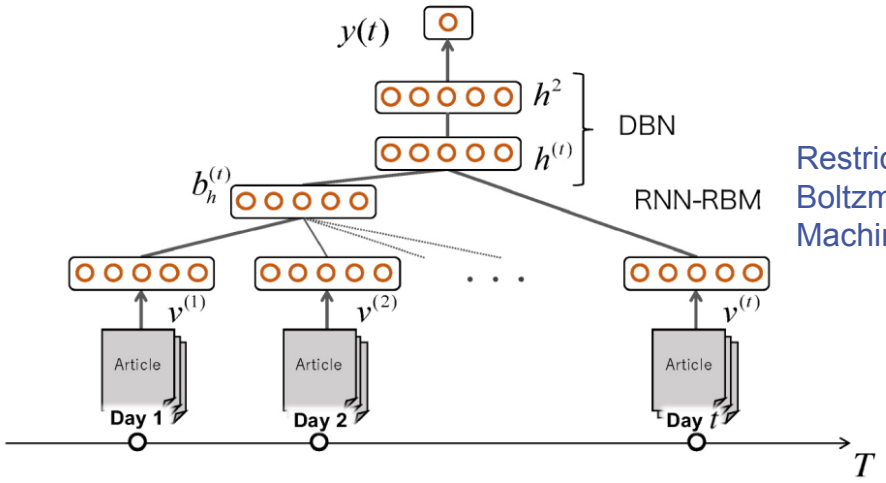
- **Input:** patients electronic health record (EHR) data over multiple visits (meaning, variable length sequences)
- **Output:** 128 diagnoses

Top 6 diagnoses measured by F1 score

Label	*F1*	AUC	Precision	Recall
Diabetes mellitus with ketoacidosis	0.8571	0.9966	1.0000	0.7500
Scoliosis, idiopathic	0.6809	0.8543	0.6957	0.6667
Asthma, unspecified with status asthmaticus	0.5641	0.9232	0.7857	0.4400
Neoplasm, brain, unspecified	0.5430	0.8522	0.4317	0.7315
Delayed milestones	0.4751	0.8178	0.4057	0.5733
Acute Respiratory Distress Syndrome (ARDS)	0.4688	0.9595	0.3409	0.7500

Lipton et al. "Learning to diagnose with LSTM recurrent neural networks."

Application: Stock Market Prediction



Restricted Boltzmann Machine (RBM)

Table 3. Test error rates for stock price prediction

Brands	Baseline	SVM	DBN	RNN-RBM + DBN
Nikkei Average	49.57	48.73	45.50	**43.62**
Hitachi	35.71	37.29	32.00	**32.00**
Toshiba	39.52	41.95	**38.50**	**38.50**
Fujitsu	40.00	40.25	**32.00**	34.00
Sharp	42.00	47.88	**40.00**	**40.00**
Sony	43.00	47.46	41.43	**40.95**
Nissan Motor	40.00	45.34	39.50	**37.00**
Toyota Motor	44.29	53.39	43.81	**42.38**
Canon	43.81	53.39	43.00	**39.11**
Mitsui	46.96	47.88	**41.43**	**41.43**
Mitsubishi	43.81	49.15	43.33	**40.43**
Average	42.61	46.61	40.05	**39.04**

Table 5. Comparison of test error rates after a significant financial crisis

Brands	SVM	RNN-RBM + DBN
Nikkei Average	51.61	**38.70**
Hitachi	61.29	**32.25**
Toshiba	54.83	**38.70**
Fujitsu	45.16	**32.25**
Sharp	58.06	**45.16**
Sony	**41.93**	**41.93**
Nissan Motor	**29.03**	35.48
Toyota Motor	48.38	**45.16**
Canon	**54.83**	**54.83**
Mitsui	41.93	**38.70**
Mitsubishi	29.03	**25.80**
Average	46.92	**39.00**

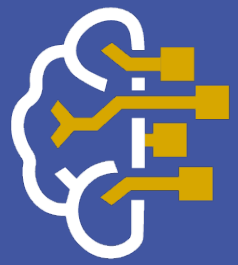
Yoshihara et al. "Leveraging temporal properties of news events for stock market prediction."

Stay Connected

Dr. Min Chi

Associate Professor
mchi@ncsu.edu
(919) 515-7825

AI Academy



AI Academy

go.ncsu.edu/aiacademy

NC STATE UNIVERSITY