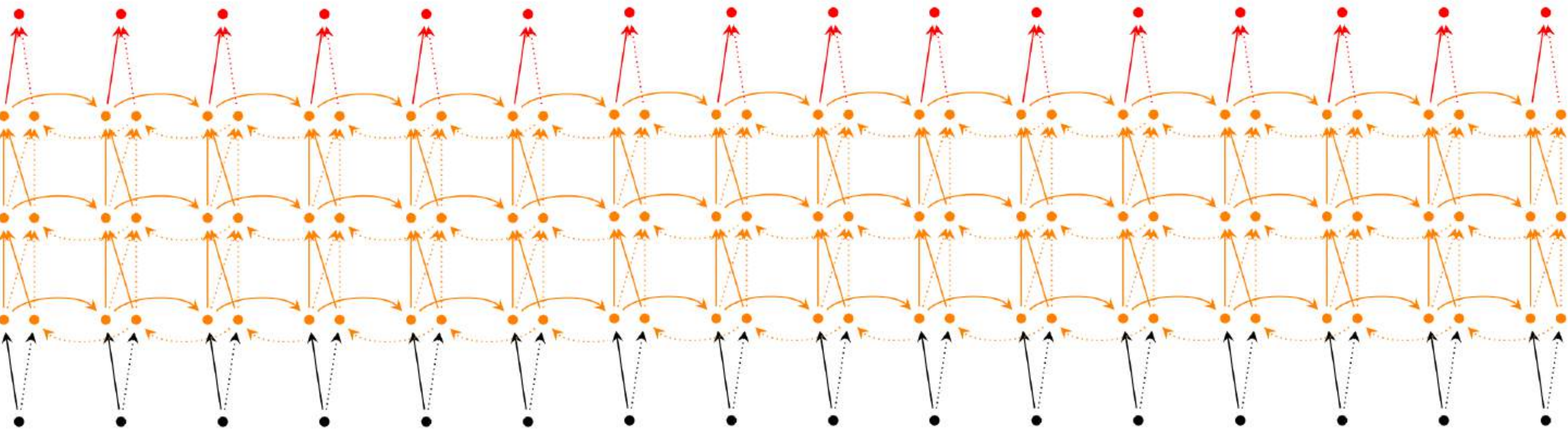


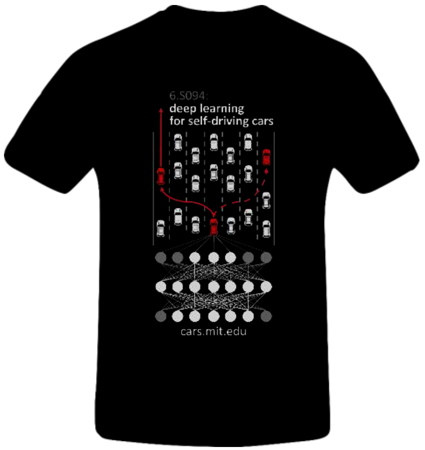
6.S094: Deep Learning for Self-Driving Cars

Recurrent Neural Networks for Steering Through Time

cars.mit.edu

Administrative

- **Website:** cars.mit.edu
- **Email:** deepcars@mit.edu
- **Tasks:**
  - Create an account on the website.
  - Submit code online for DeepTrafficJS that exceeds 65mph
  - Submit code online for DeepTeslaJS (no performance requirement)
  - Fill out "Deep Thoughts" in your "Edit Profile" page
- **Shirts:** Handed out at the end of class on Friday
- **DeepTraffic Competition:**
  - Leaderboard shuts down Friday 11am
  - Winners announced and congratulated on Friday



DeepTraffic Leaderboard

DeepTraffic - About DeepTraffic

Rank	User		MPH
1	yufuin		74.20
2	michael_gump	🎓 MIT	74.04
3	JSC6	MIT	73.78
4	Jeffrey Hu	🎓 MIT	73.59
5	p_dolly	🎓 MIT	73.50
6	Lex Fridman	MIT	73.48
7	cogbal		73.46
8	tancik	🎓 MIT	73.45
9	Timothy Kassis	MIT	73.33
10	naveen		73.09

MIT - MIT Affiliated
🎓 - Registered Student

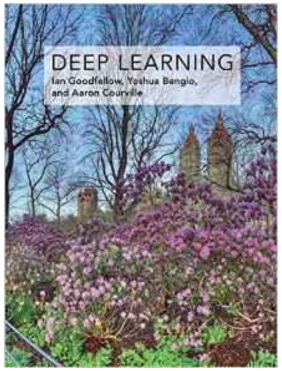
Competition Prizes:

Top 3
($800 value)

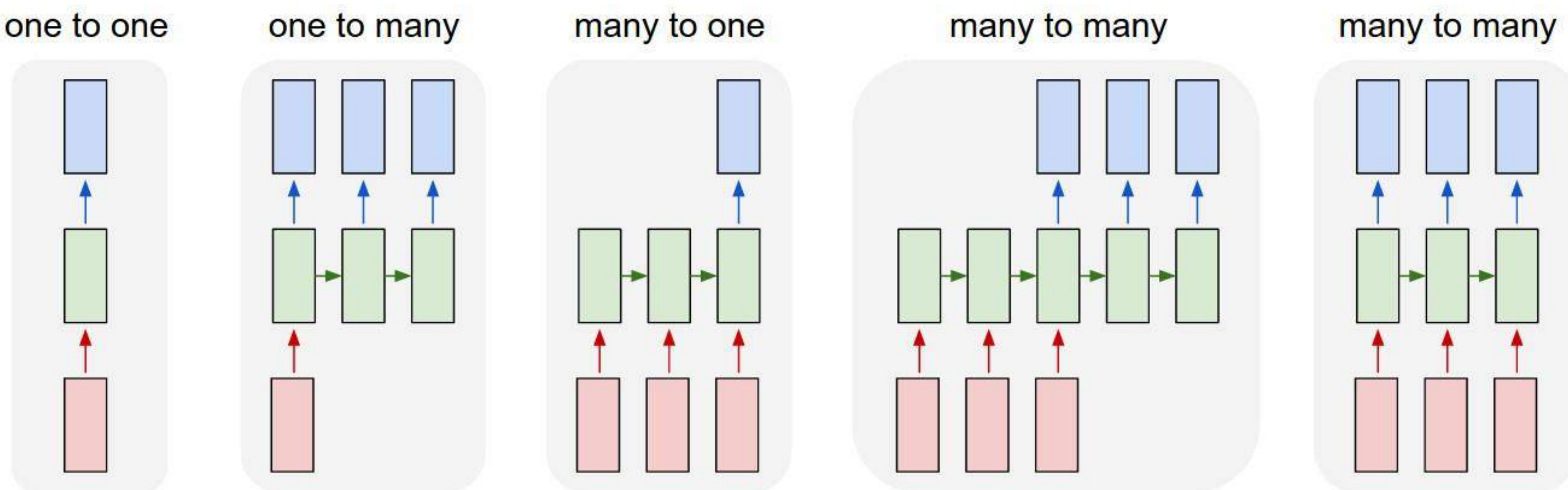
UDACITY
Self-Driving Car Engineer Nanodegree

+

Top 1
(Priceless)



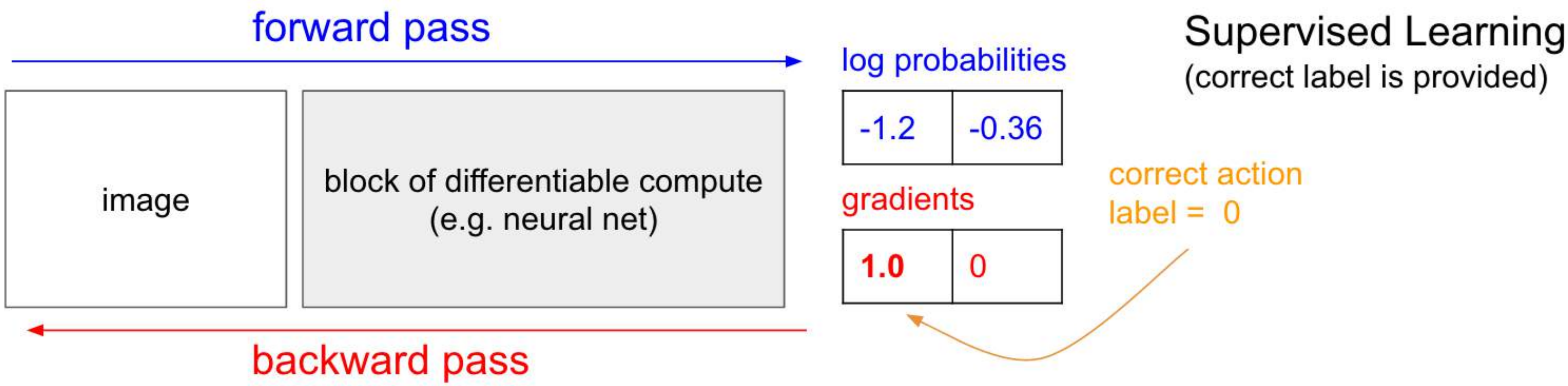
Flavors of Neural Networks



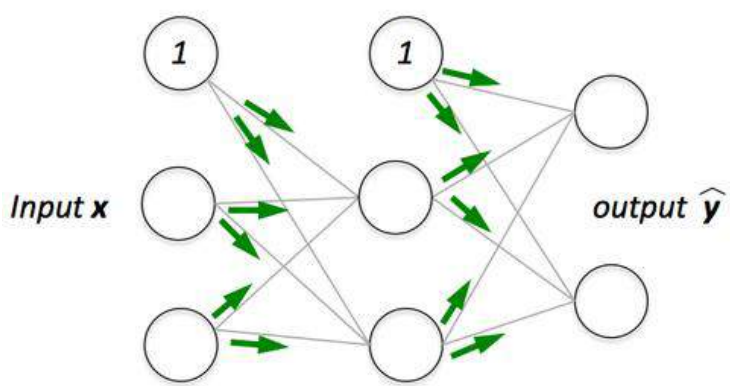
"Vanilla" Neural Networks

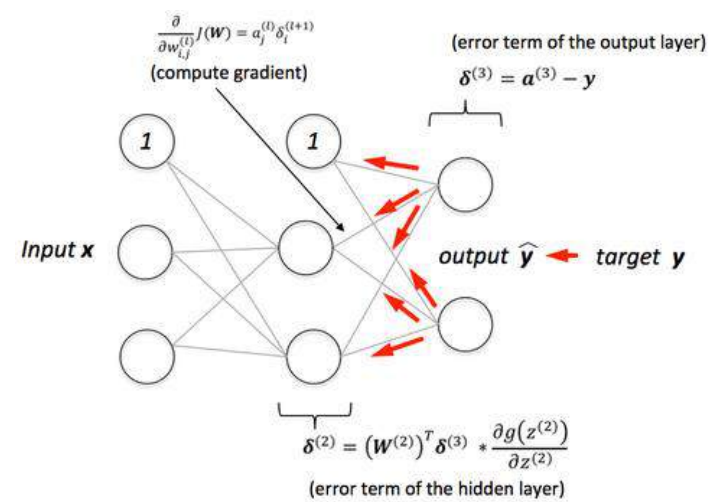
Recurrent Neural Networks

Back to Basics: Backpropagation



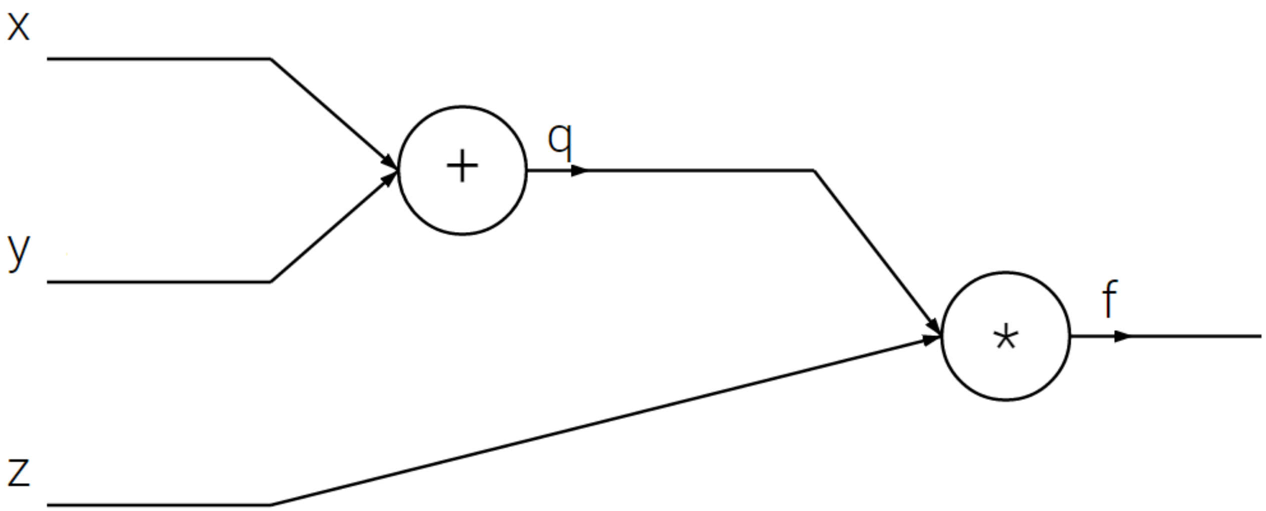
Adjust the weights to reduce the error:





Backpropagation: By Example

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

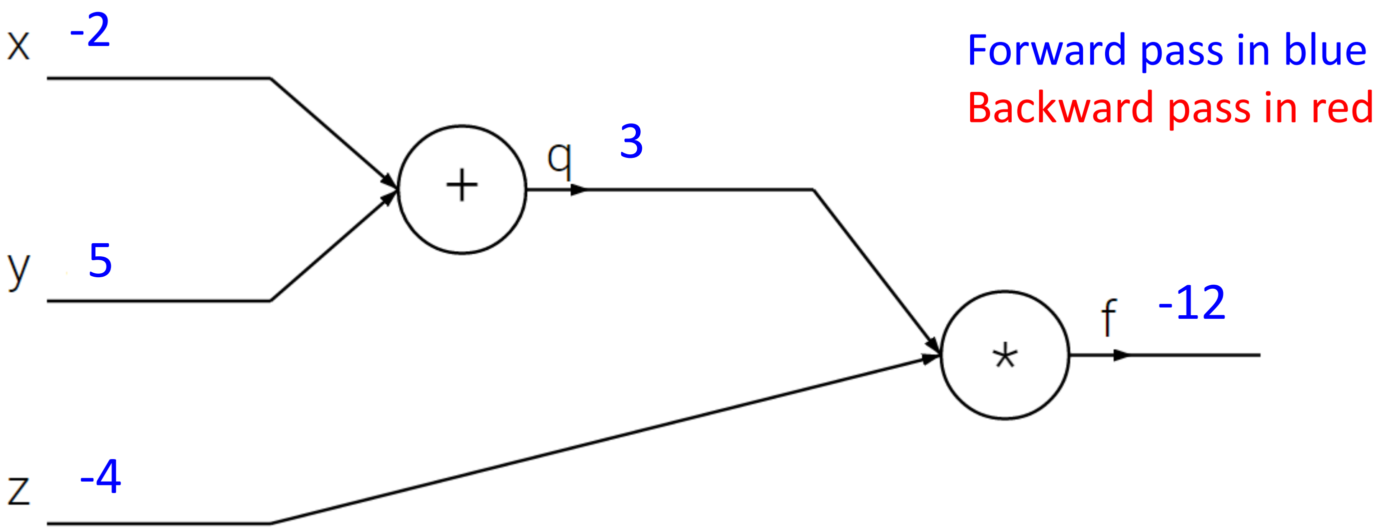


Modularity: We compute an arbitrary function locally in stages

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

Backpropagation: Forward Pass

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



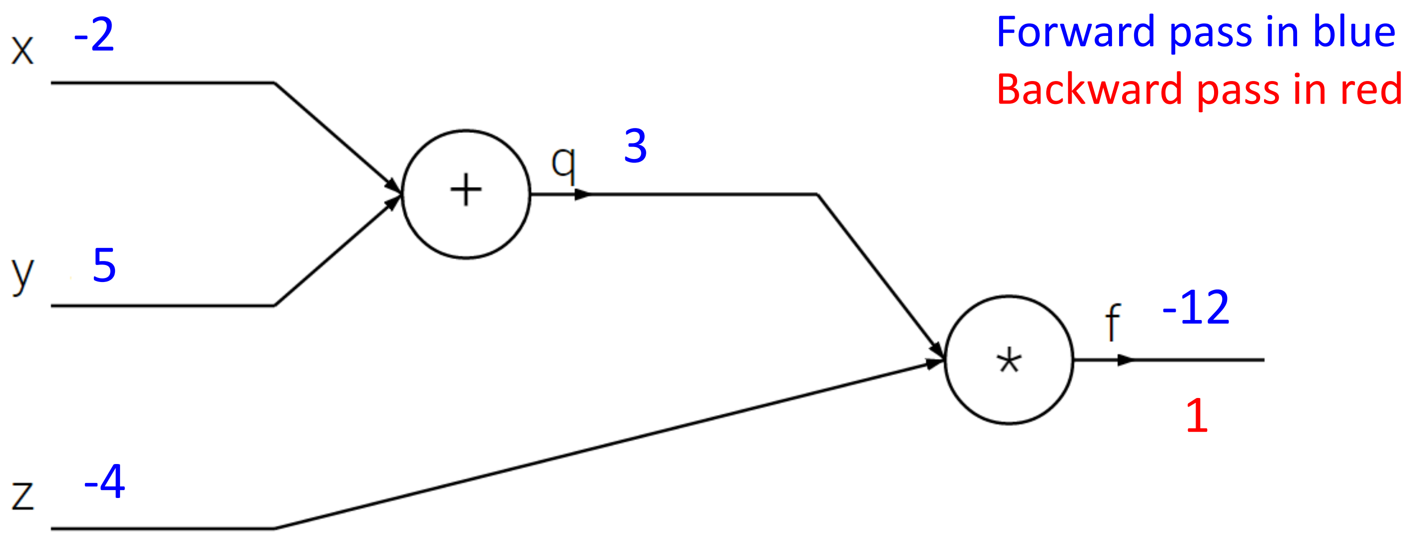
f(x, y, z) is “happy” when the output is as high as possible

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

References: [101]

Backpropagation: Forward Pass

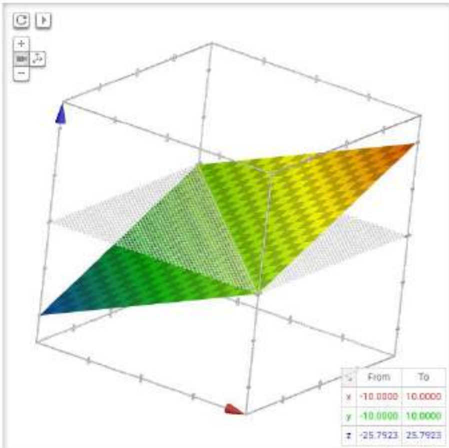
$$f(x, y, z) = (x + y)z$$



f(x, y, z) is “happy” when the output is as high as possible

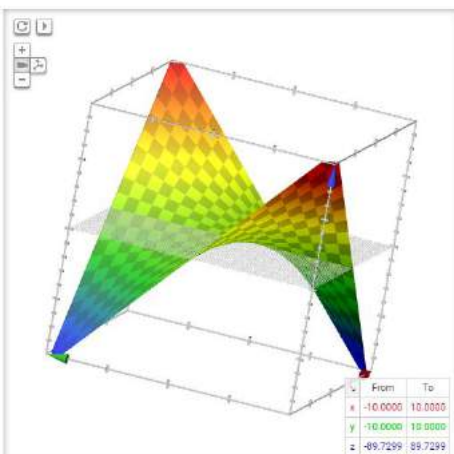
How do we “teach” it to produce a higher output?

Backpropagation: By Example



Addition:

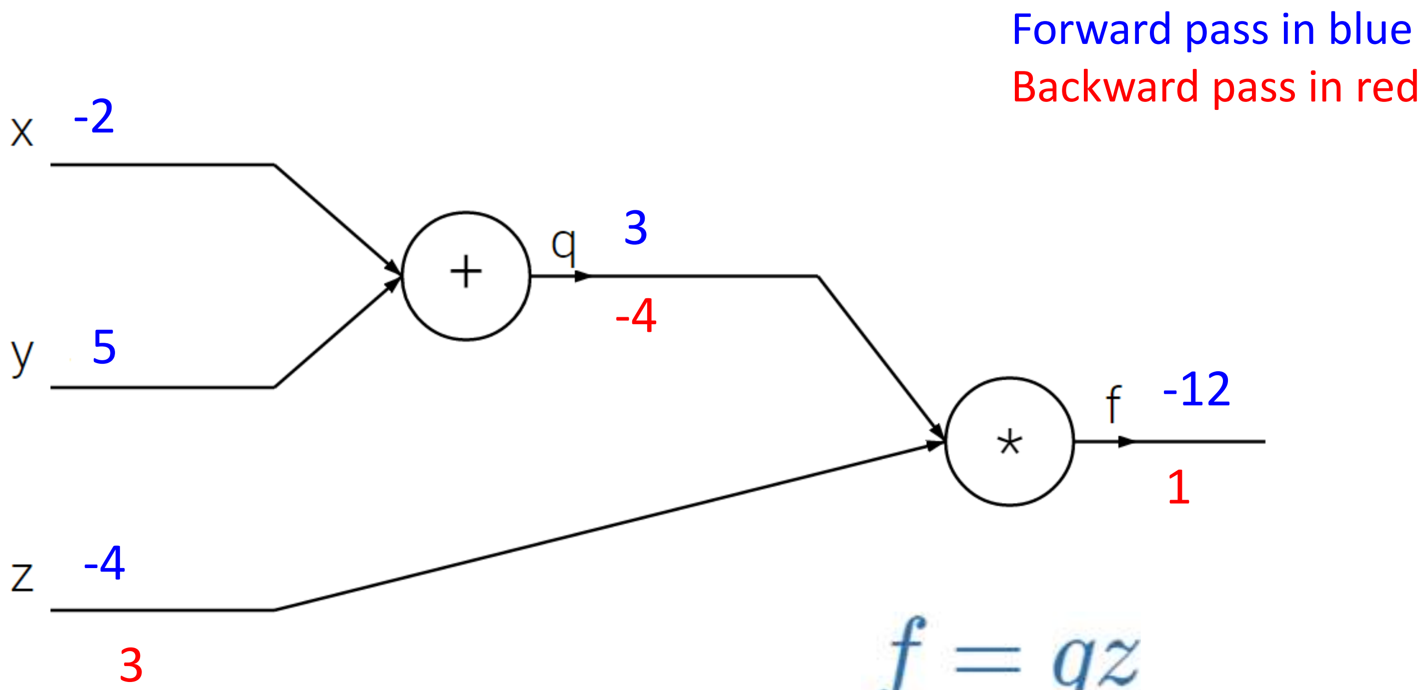
$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$



Multiplication:

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

Backpropagation: Backward Pass



$$f = qz$$

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

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

Modular Magic: Chain Rule

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

So, instead of computing the gradient of this:

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

We compute the gradients of these:

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

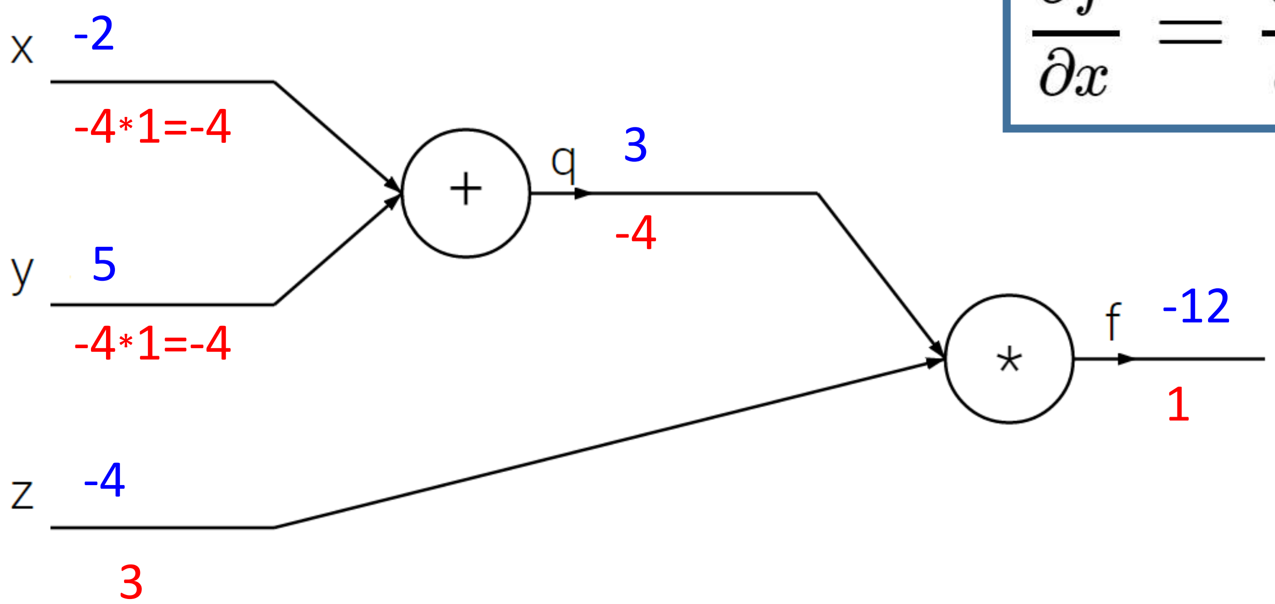
Backpropagation: Backward Pass

Forward pass in blue

Backward pass in red

Modular Magic: Chain Rule

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



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

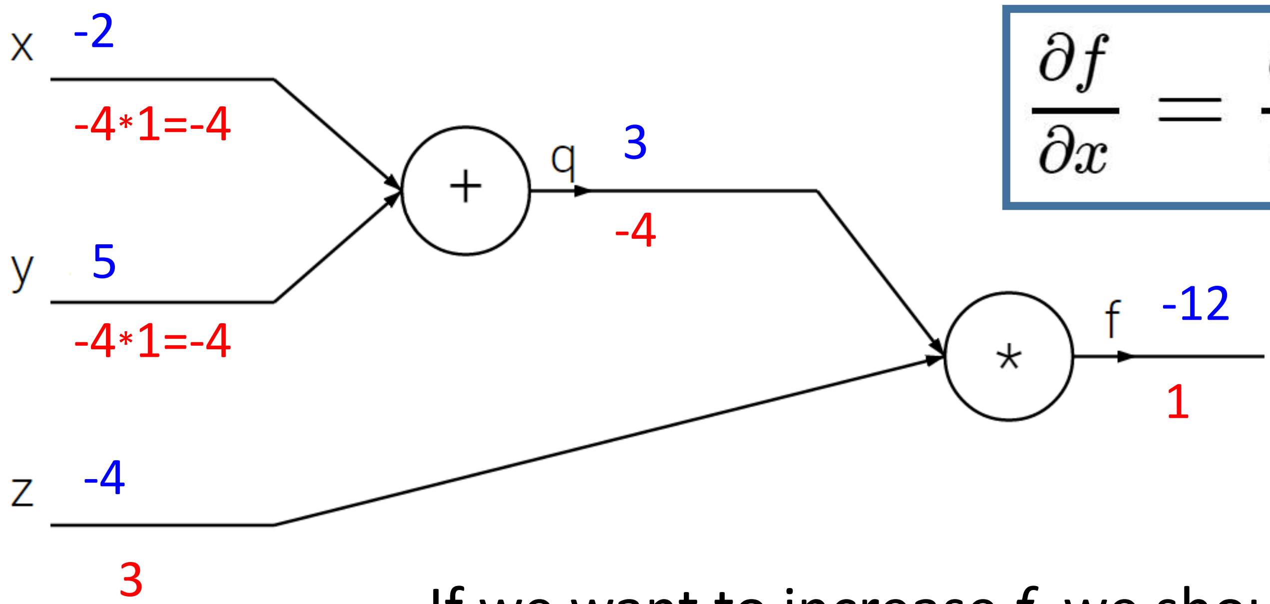
$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

Forward pass in blue
Backward pass in red

Interpreting Gradients

Modular Magic: Chain Rule

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



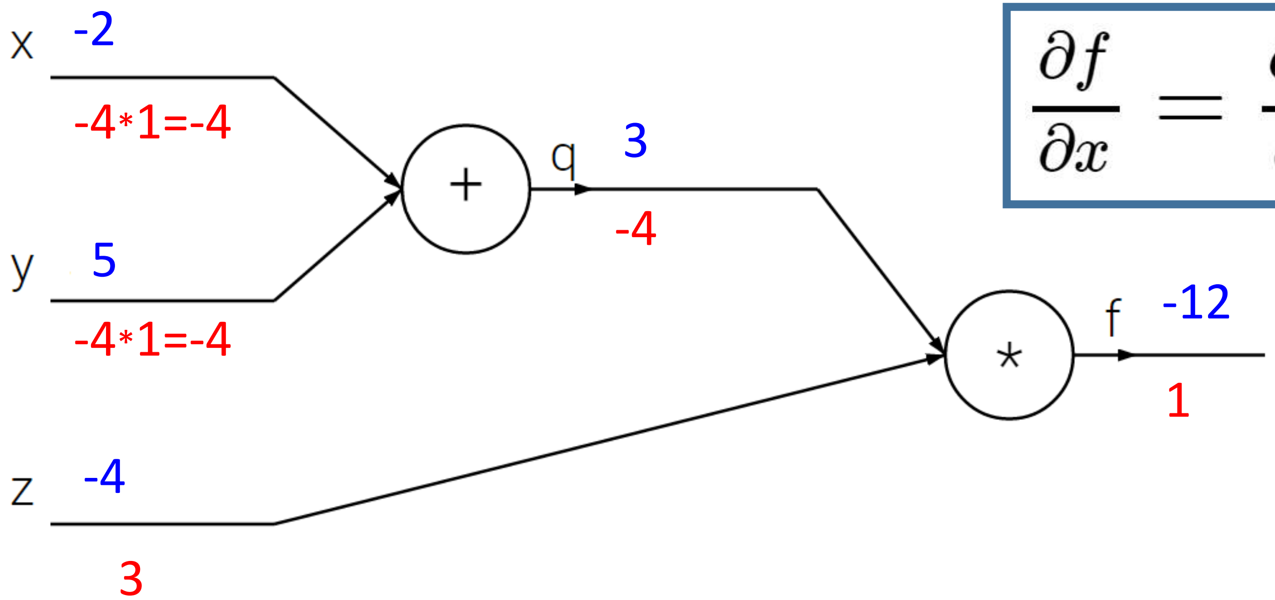
If we want to increase *f*, we should:

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

* Note the beautiful **simplicity**:
Every local gradient is a local worker in a global chase for greater *f*.

Forward pass in blue
Backward pass in red

Interpreting Gradients



Modular Magic: Chain Rule

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

Add gate:

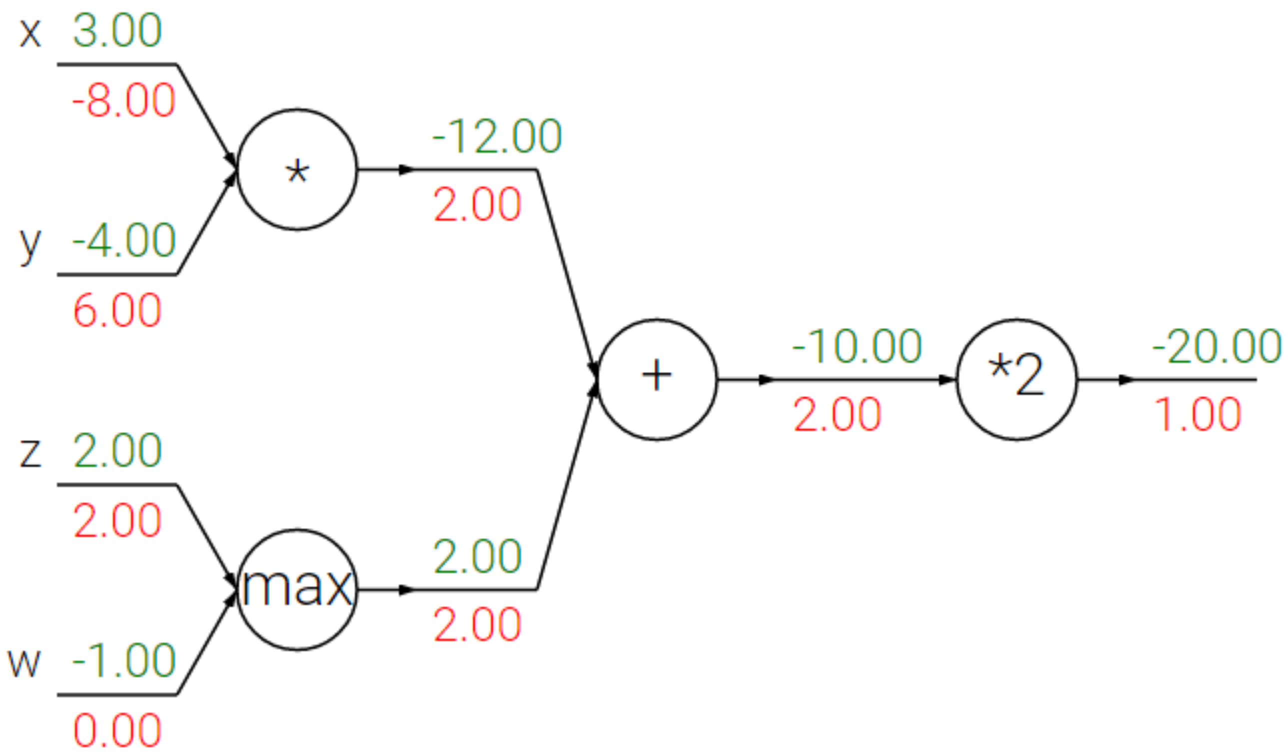
- Equally distributes the gradient from output to input
- Ignores forward pass values

Multiply gate:

- Switch forward pass values
- Multiply by gradient on output

Forward pass in green
Backward pass in red

Interpreting Gradients



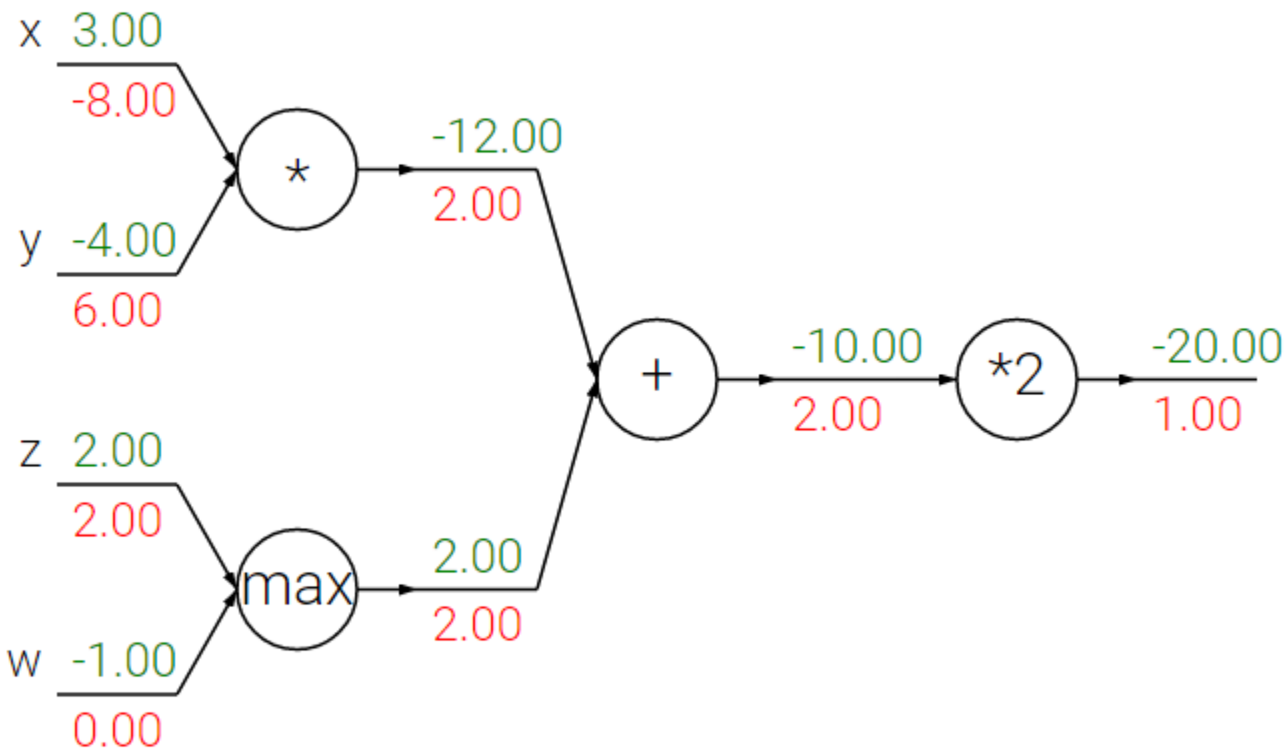
Add gate:
- Equally distributes the gradient from output to input
- Ignores forward pass values

Multiply gate:
- Switch forward pass values
- Multiply by gradient on output

Max gate:
- Distributes the gradient from output to just one input (the one with the largest forward pass value)

Forward pass in green
Backward pass in red

Interpreting Gradients



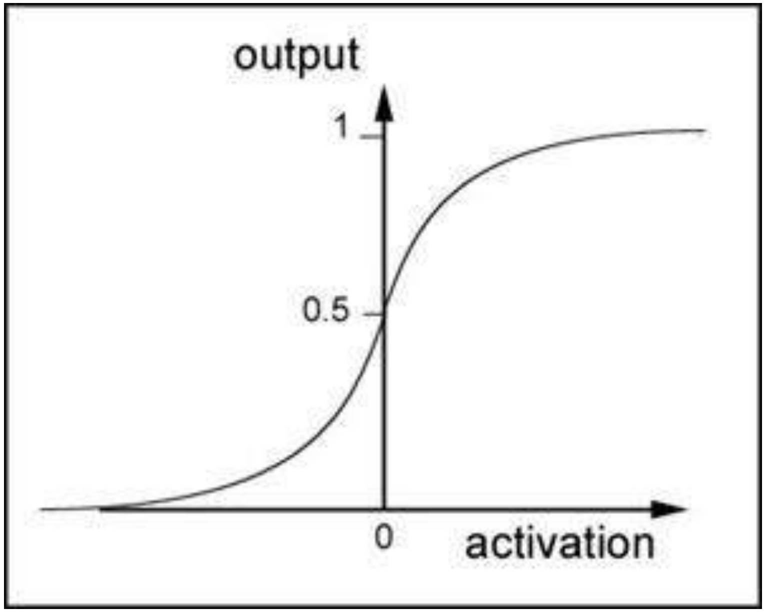
Add gate:
- Equally distributes the gradient from output to input
- Ignores forward pass values

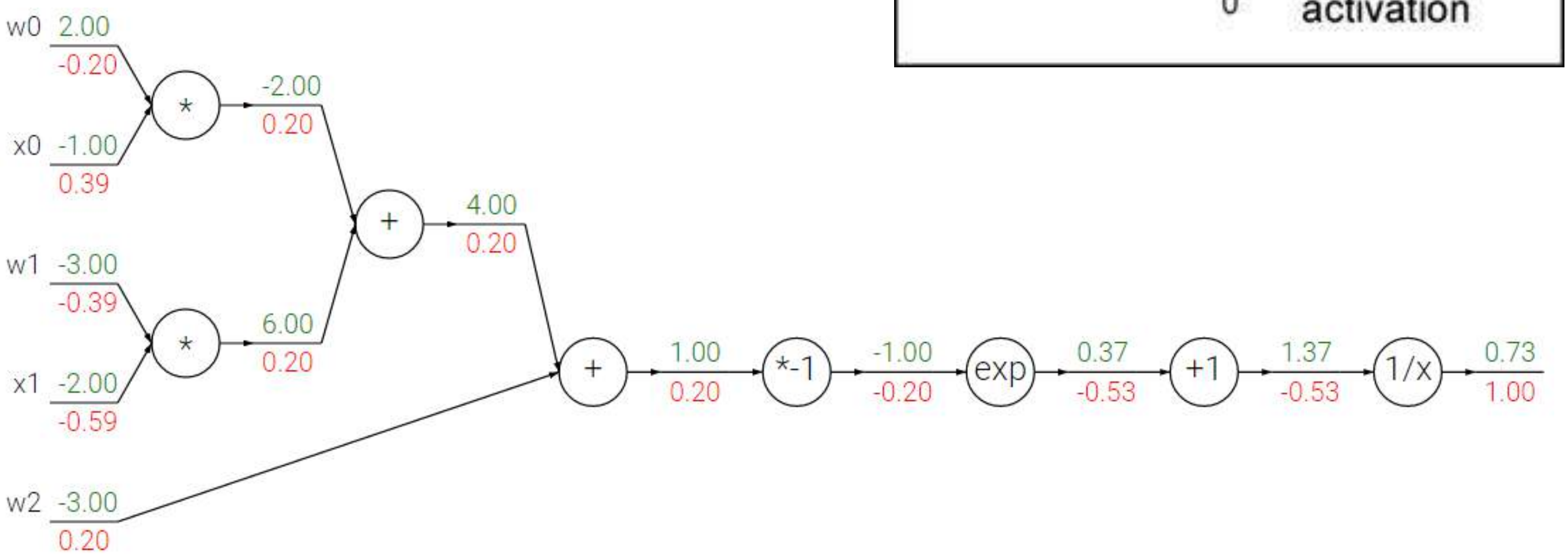
Multiply gate:
- Switch forward pass values
- Multiply by gradient on output

Max gate:
- Distributes the gradient from output to just one input (the one with the largest forward pass value)

Modularity Expanded: Sigmoid Activation Function

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

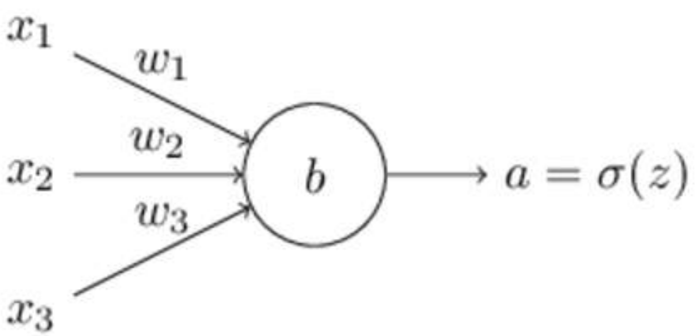




Side note, the derivative of the sigmoid function simplifies to: $$\frac{d\sigma(x)}{dx} = (1-\sigma(x))\,\sigma(x)$$

Learning with Backpropagation

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



Loss (aka cost, objective) function:

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

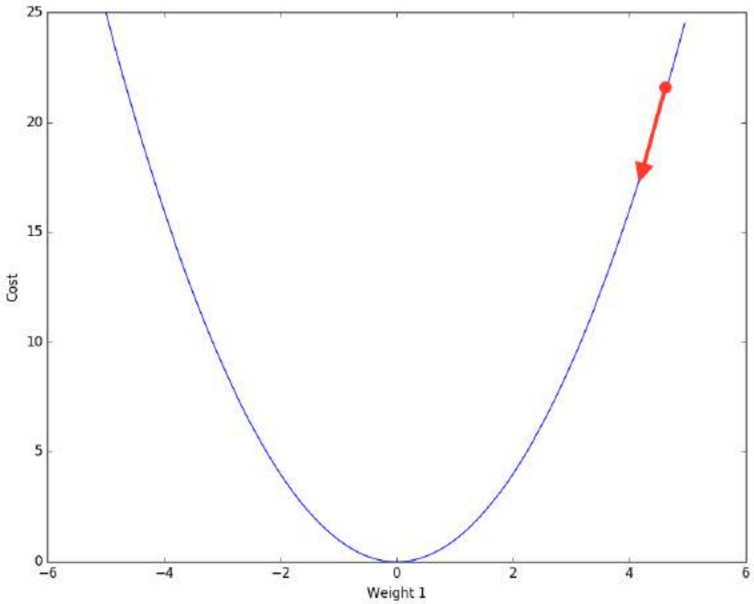
Subtasks:

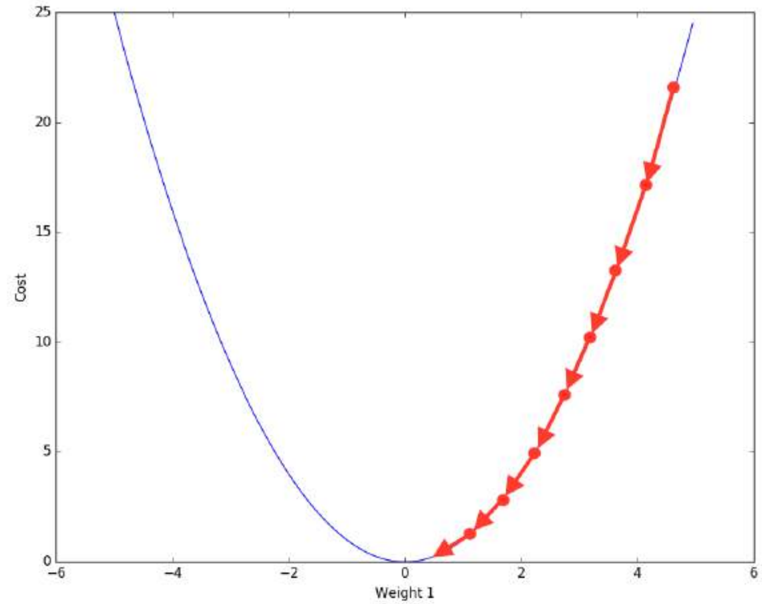
1. Forward pass to compute network output and "error"
2. Backward pass to compute gradients
3. A fraction of the weight's gradient is subtracted from the weight.

Learning Rate

Learning is an Optimization Problem

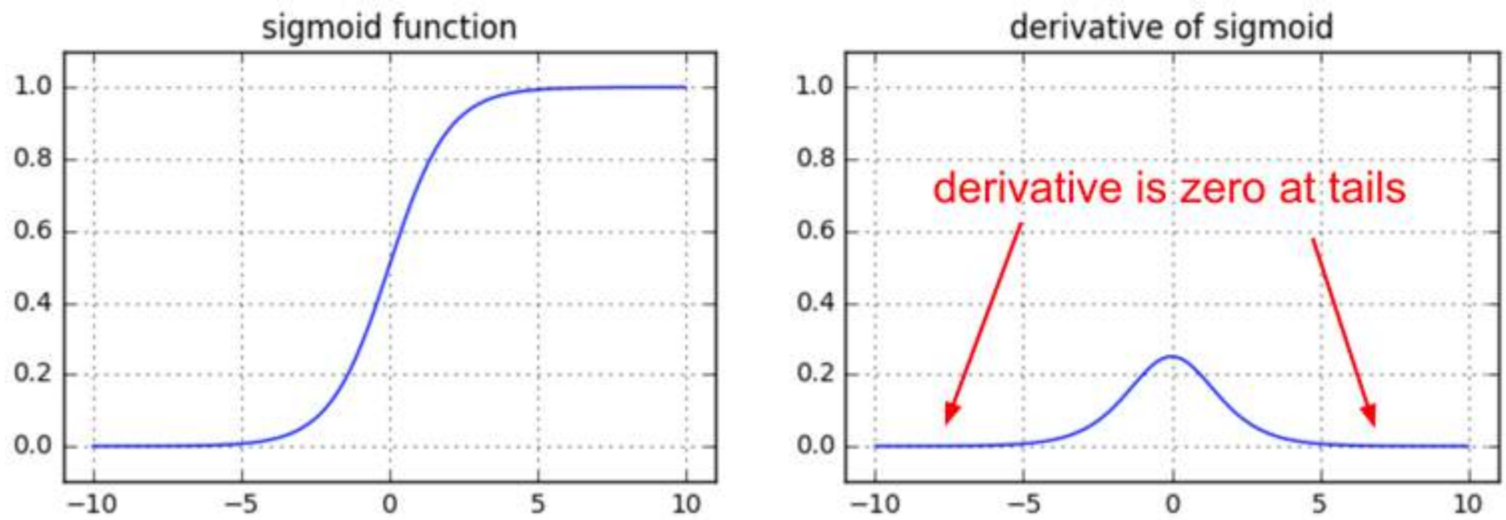
Task: Update the **weights** and **biases** to decrease **loss function**





Use mini-batch or stochastic gradient descent.

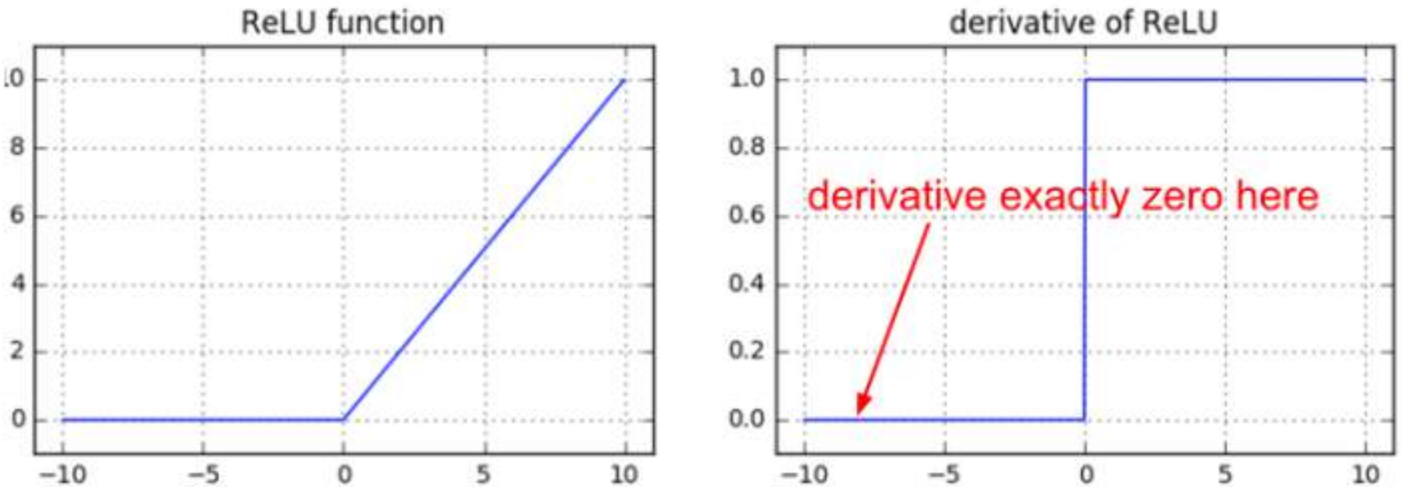
Optimization is Hard: Vanishing Gradients



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

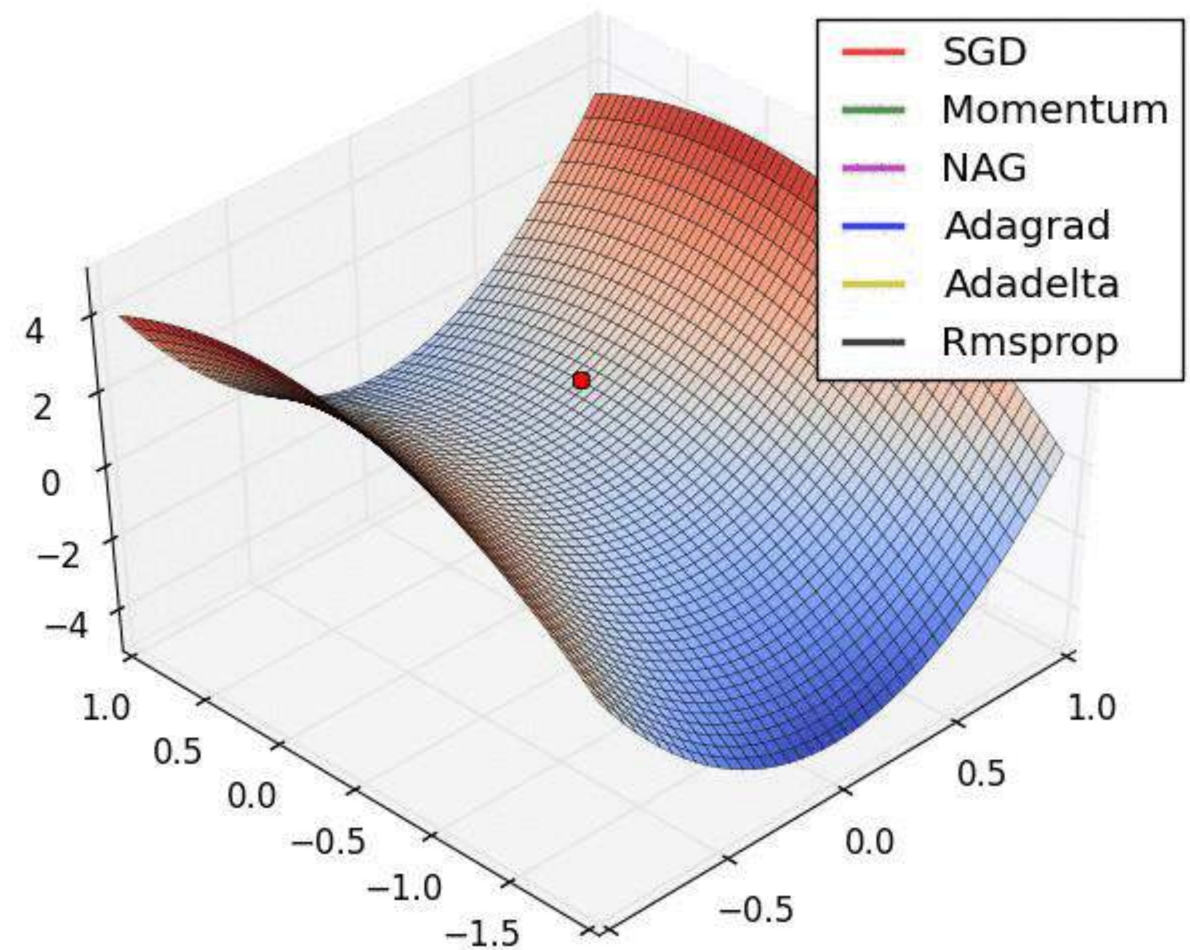
Partial derivatives are small = Learning is slow

Optimization is Hard: Dying ReLUs



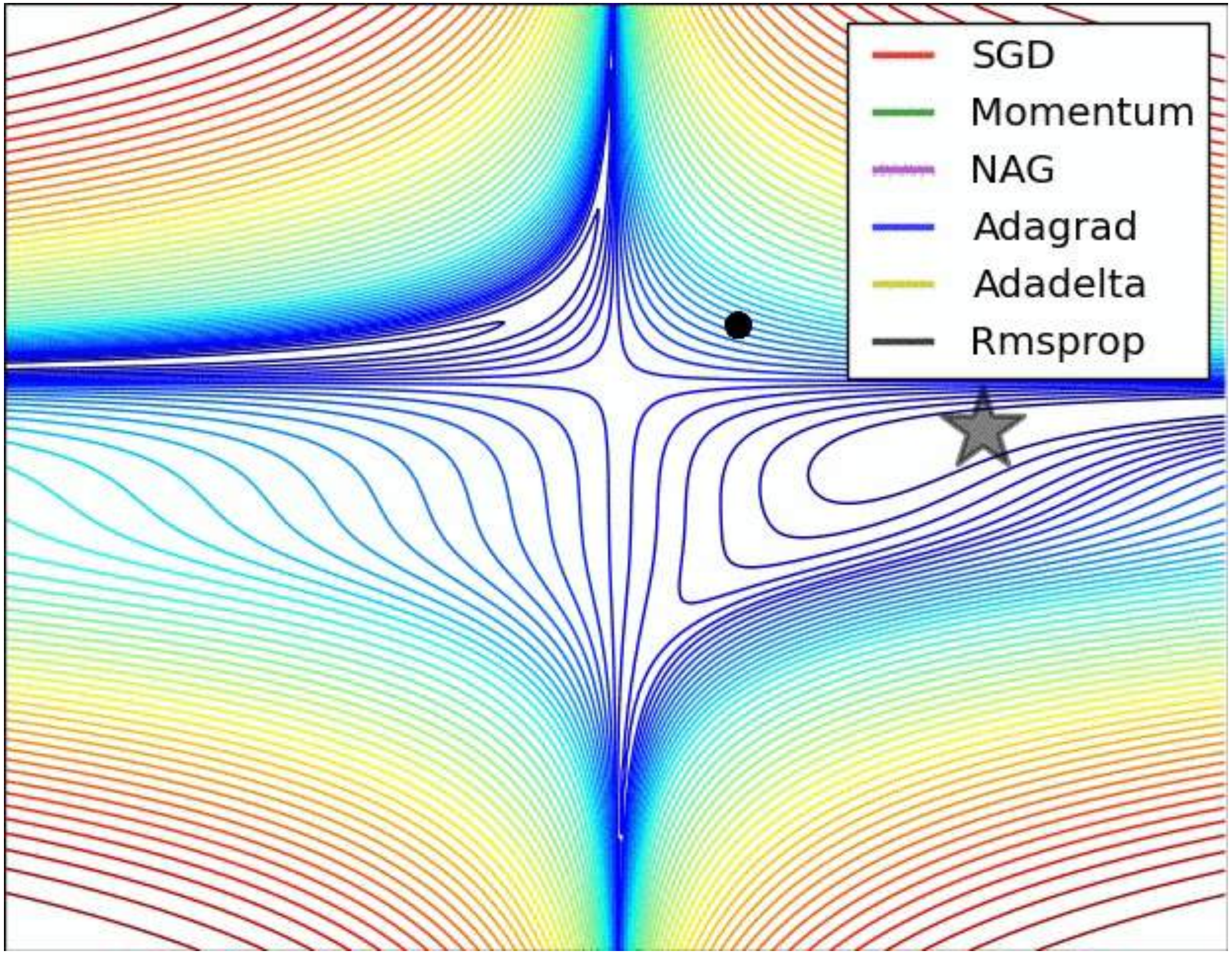
- If a neuron is initialized poorly, it might not fire for entire training dataset.
- Large parts of your network could be dead ReLUs!

Optimization is Hard: Saddle Point



Hard to break symmetry

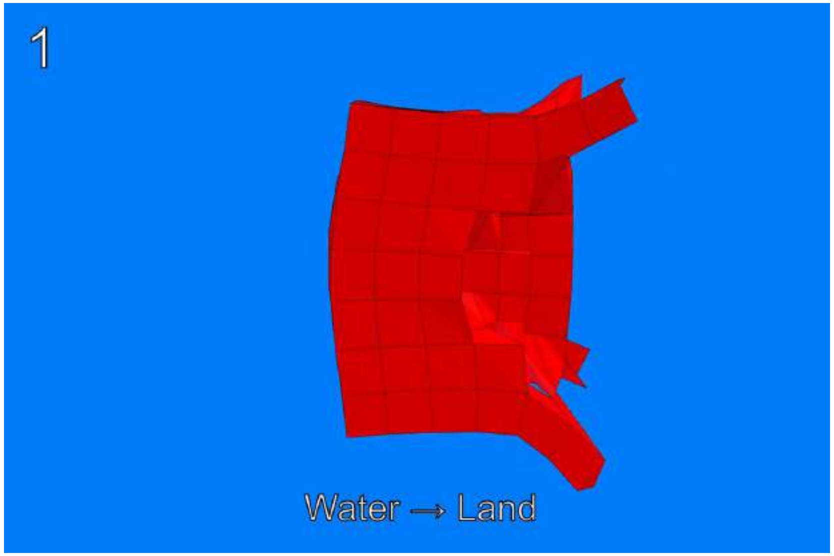
Learning is an Optimization Problem



Takeaway: Vanilla SGD gets your there, but is slow sometimes.

Reflections on Backpropagation

- **Pause to reflect:** Backpropagation and gradient descent is the mechanism of machine intelligence. Can it lead to "human-level reasoning"?

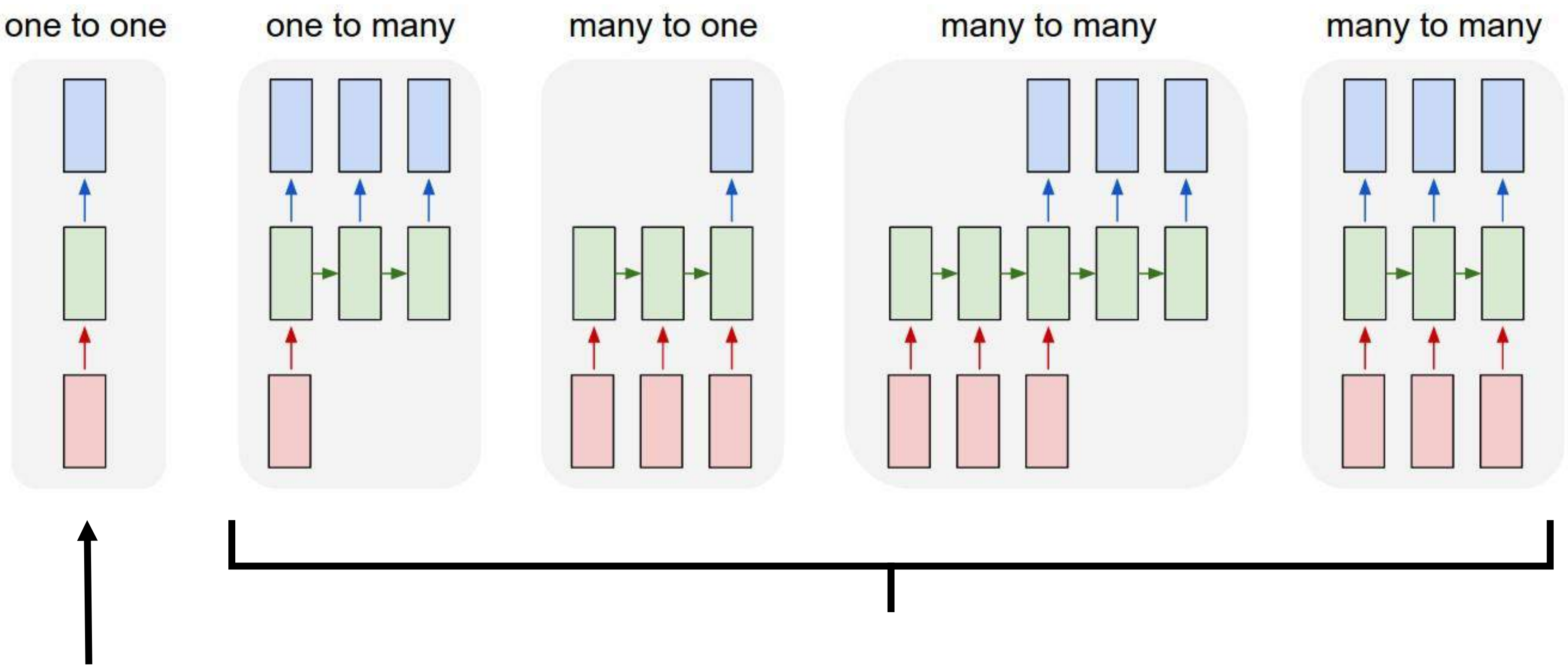


- Some alternatives:
    - Genetic Algorithms
    - Particle Swarm Optimization
    - Ant Colony Optimization

Corucci et al. "Evolving swimming soft-bodied creatures." 2016.

- **Q1:** What other ways can we optimize the weights of a neural network?
- **Q2:** What other ways can we optimize (evolve) the design of the network?

Back to Recurrent Neural Networks (RNNs)

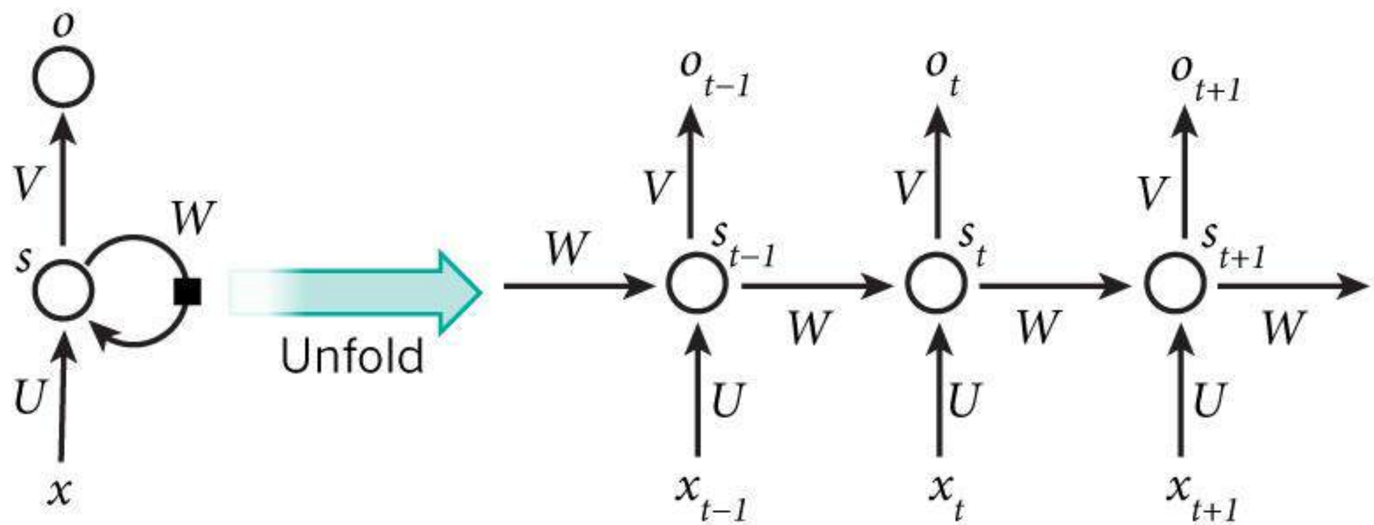


"Vanilla" Neural Networks

Recurrent Neural Networks

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

Unrolling a Recurrent Neural Network

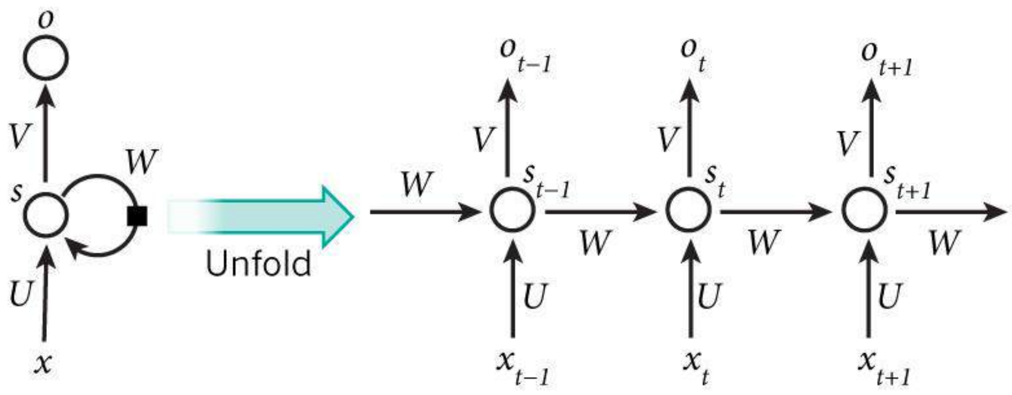


- **Input *(x)***: (example: word of a sentence)

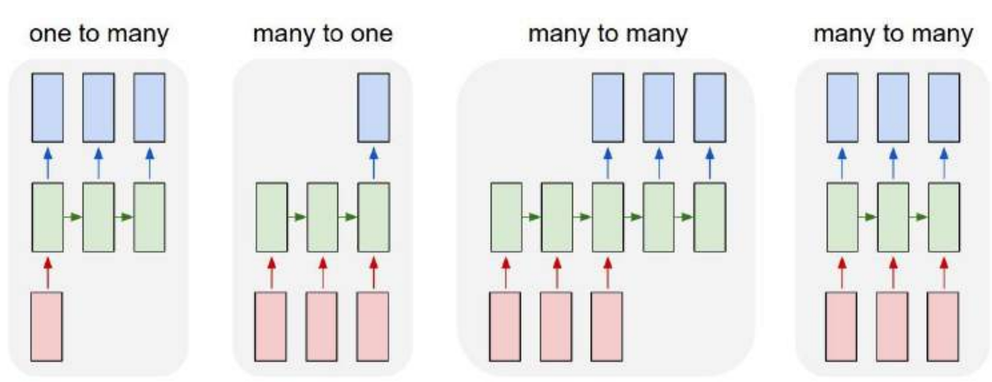
Memory →

- **Hidden state *(s)***: function of previous hidden state and new input
- **Output *(o)***: (example: predict next word in the sentence)

RNN Observations

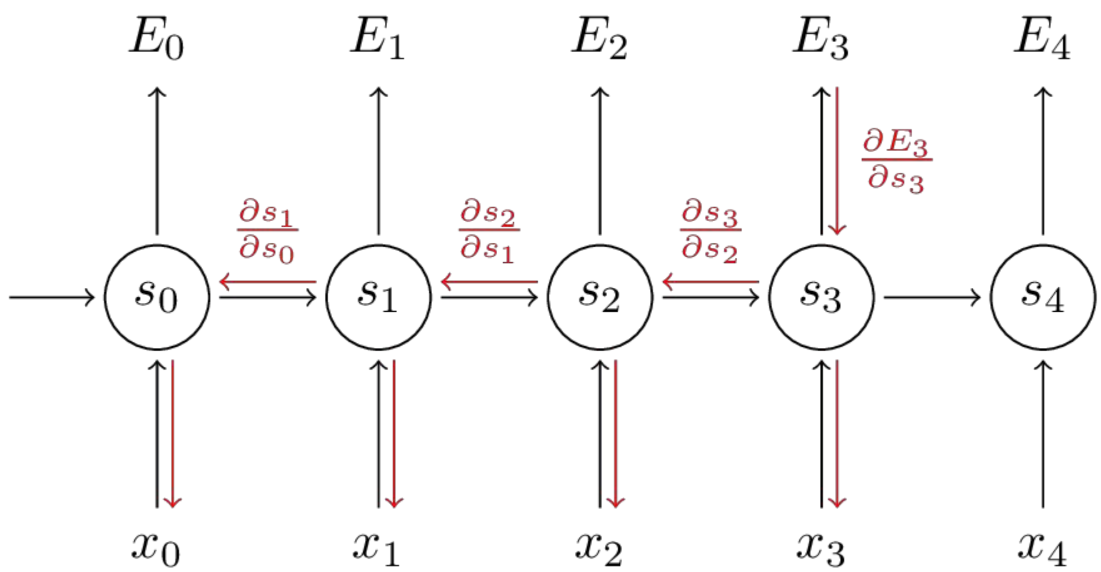


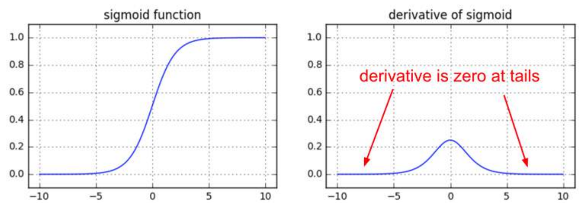
- Parameters U, V, W are shared across time
  - Similar to CNNs: this reduces the # of parameters we need to optimize
  - And it allow us to process arbitrary “temporal size” of input
- Process is the same for any input / output mapping:



Backpropagation Through Time (BPTT)

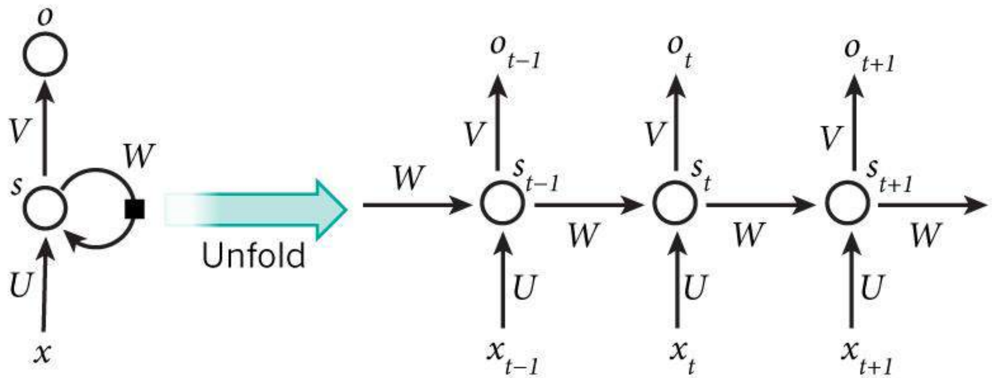
(Fancy name for regular backpropagation on an unrolled RNN)





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

Gradients Can Explode or Vanish



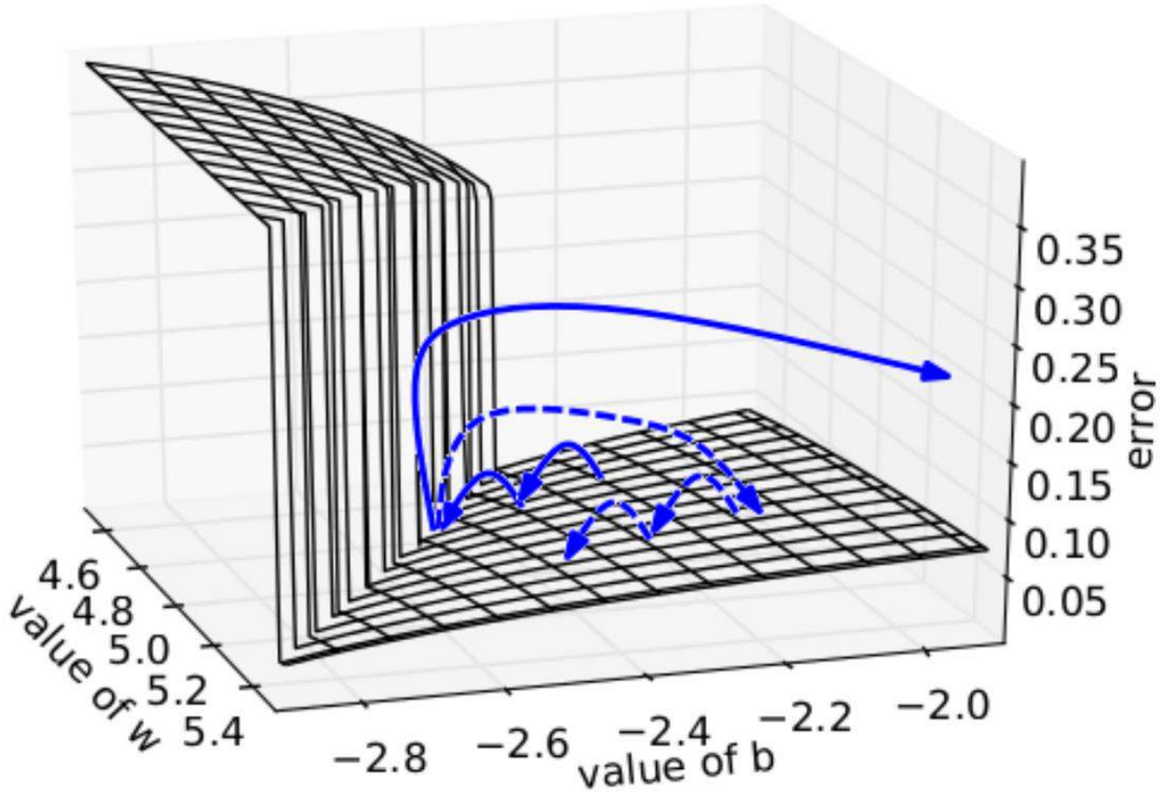

```
H = 5    # dimensionality of hidden state
T = 50   # number of time steps
Whh = np.random.randn(H,H)

# forward pass of an RNN (ignoring inputs x)
hs = {}
ss = {}
hs[-1] = np.random.randn(H)
for t in xrange(T):
    ss[t] = np.dot(Whh, hs[t-1])
    hs[t] = np.maximum(0, ss[t])

# backward pass of the RNN
dhs = {}
dss = {}
dhs[T-1] = np.random.randn(H) # start off the chain with random gradient
for t in reversed(xrange(T)):
    dss[t] = (hs[t] > 0) * dhs[t] # backprop through the nonlinearity
    dhs[t-1] = np.dot(Whh.T, dss[t]) # backprop into previous hidden state
```

if the largest eigenvalue is > 1, gradient will explode
if the largest eigenvalue is < 1, gradient will vanish

Gradients Can Explode or Vanish
(Geometric Interpretation)

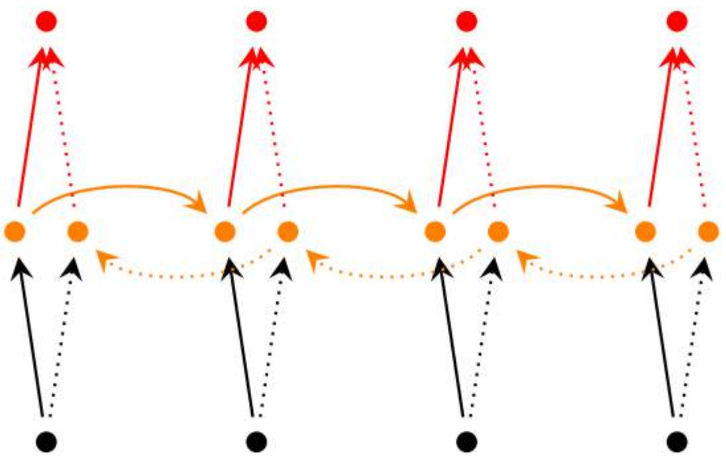
Error surface for single hidden unit RNN

$$x_t = w\sigma(x_{t-1}) + b$$

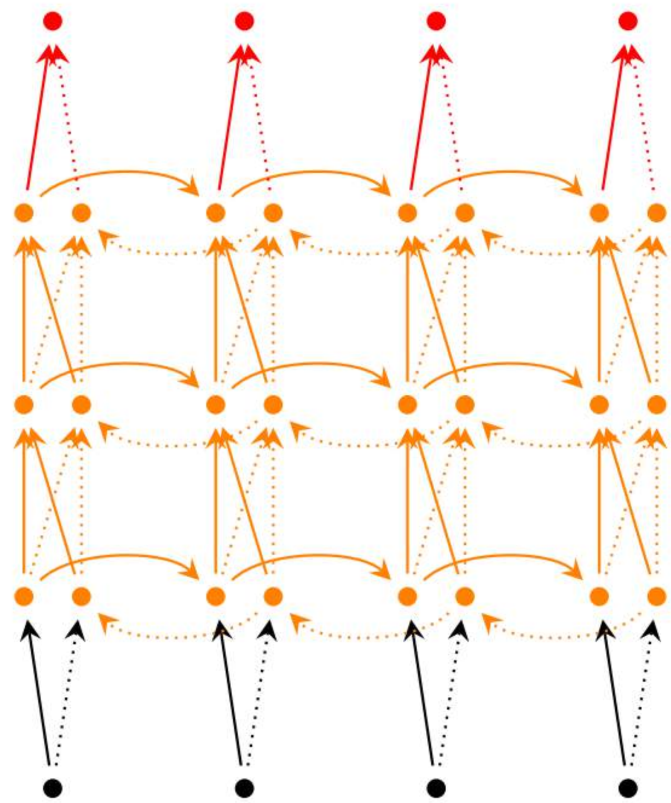


RNN Variants: Bidirectional RNNs

(Shallow)

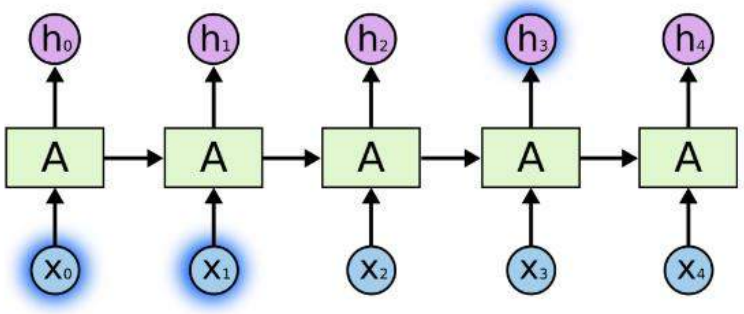


(Deep)



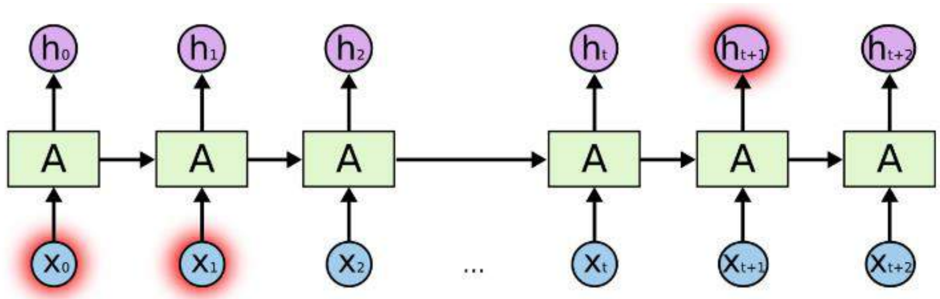
- Example:
  - Filling in missing words
- Deeper =
  - more learning capacity
  - but needs lots of training data

Long-Term Dependency



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

Context →

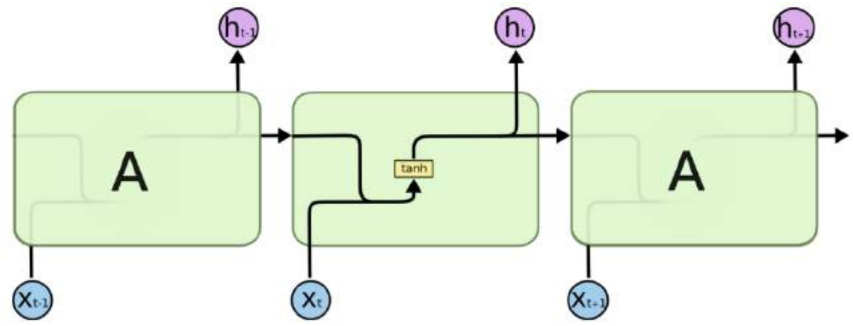


In theory, vanilla RNNs can handle arbitrarily long-term dependence.

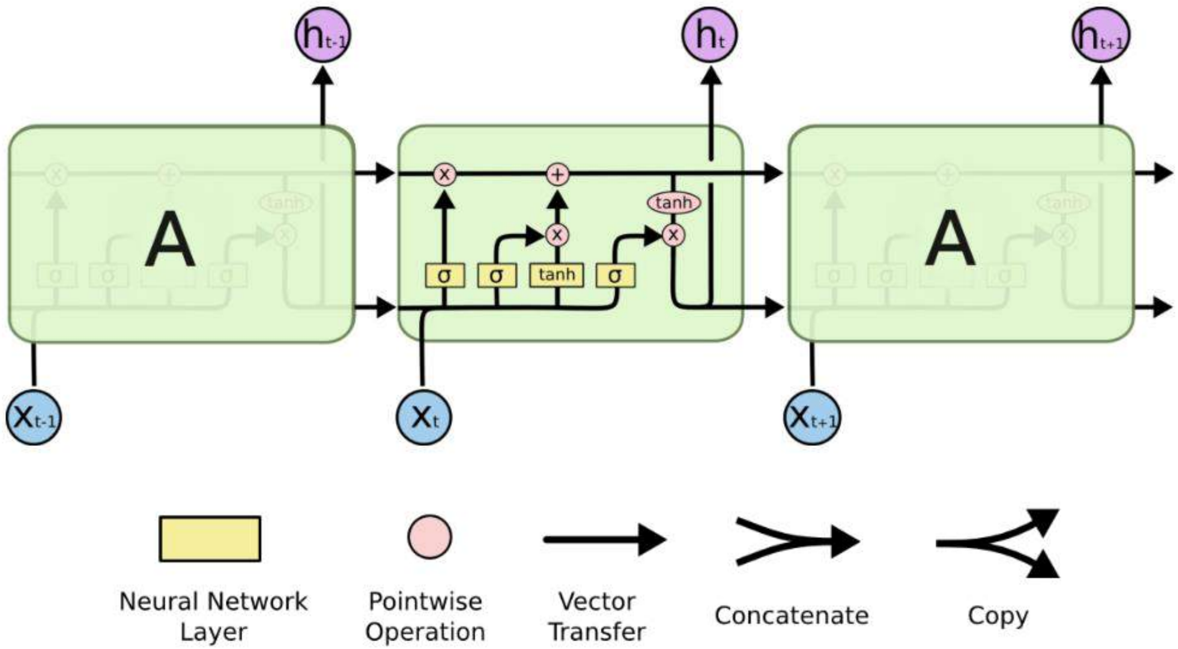
In practice, it's difficult.

Long Short Term Memory (LSTM) Networks

Vanilla RNN:

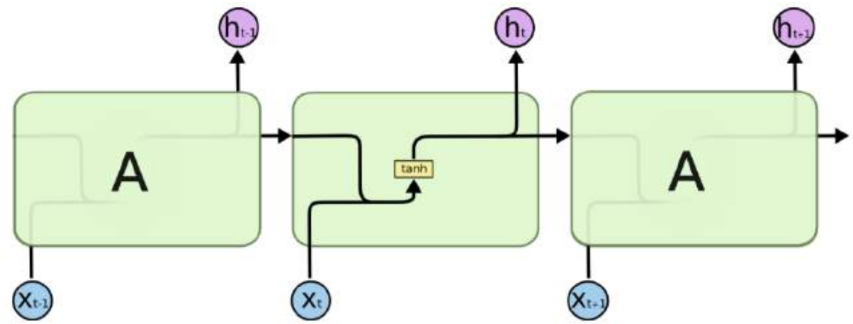


LSTM:

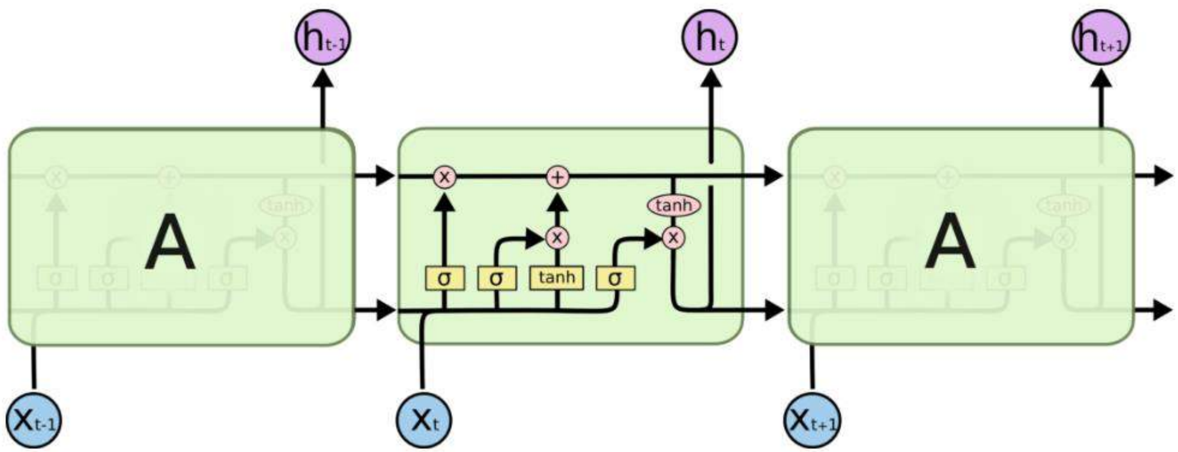


LSTM: Gates Regulate

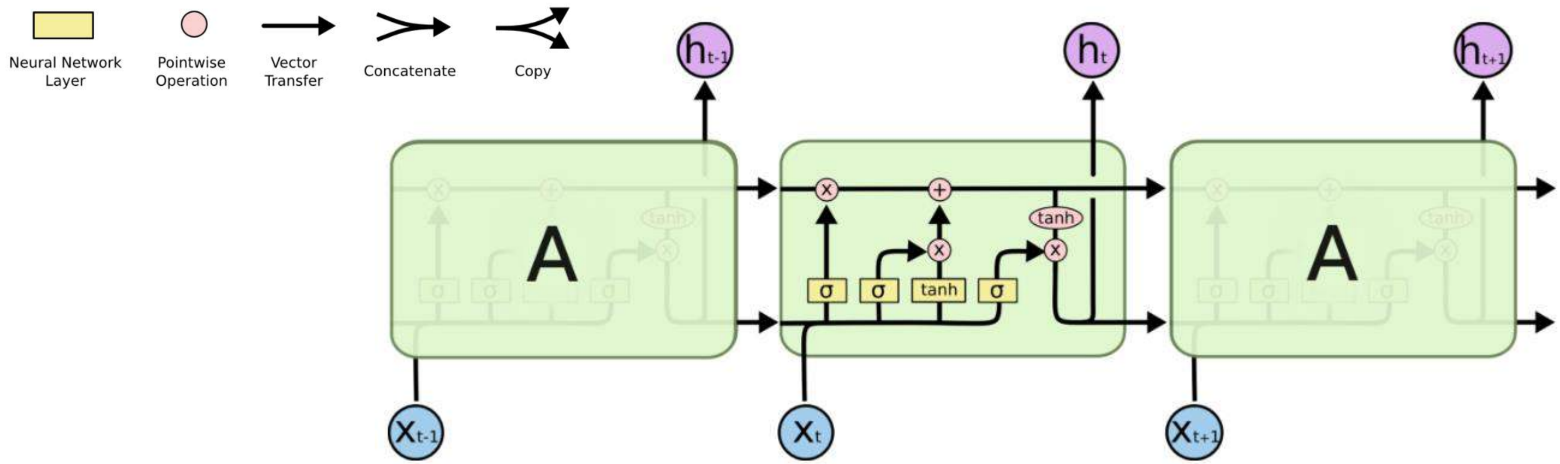
Vanilla RNN:



LSTM:



LSTM: Pick What to Forget and What To Remember

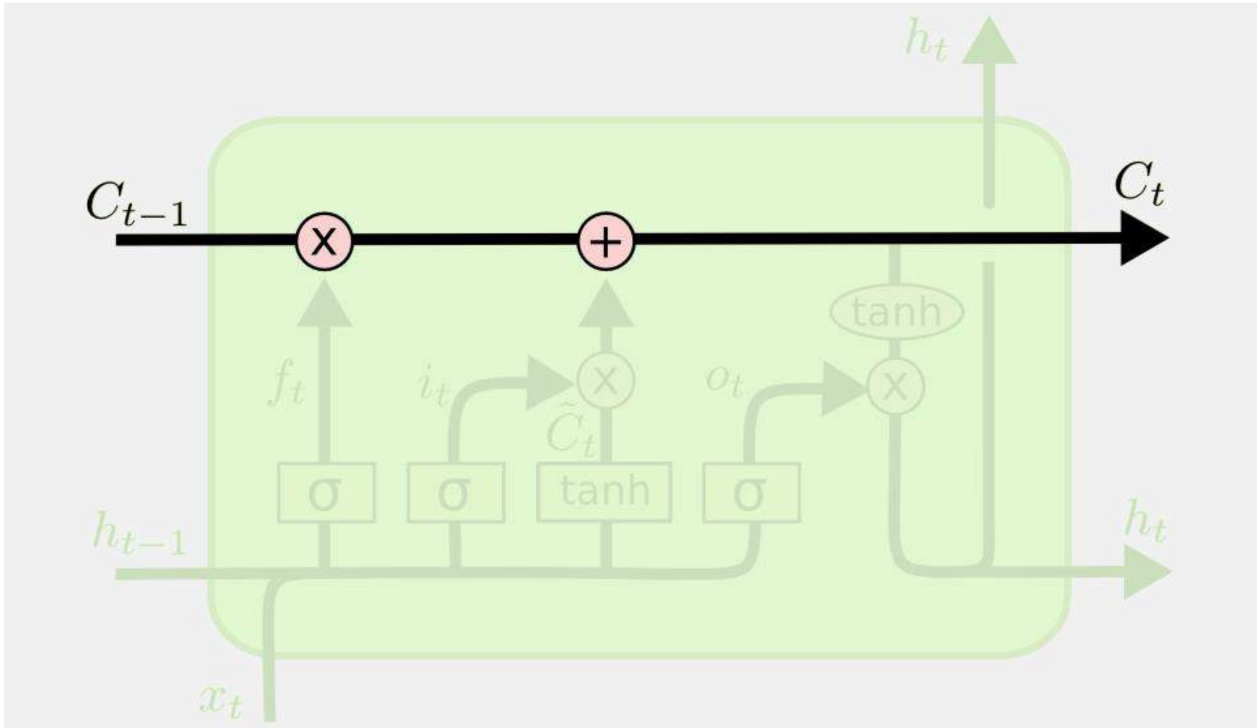


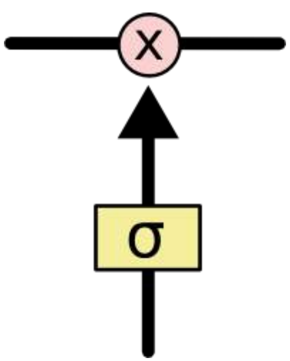
Bob and Alice are having lunch. Bob likes apples. Alice likes oranges. **She is eating an orange.**

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)

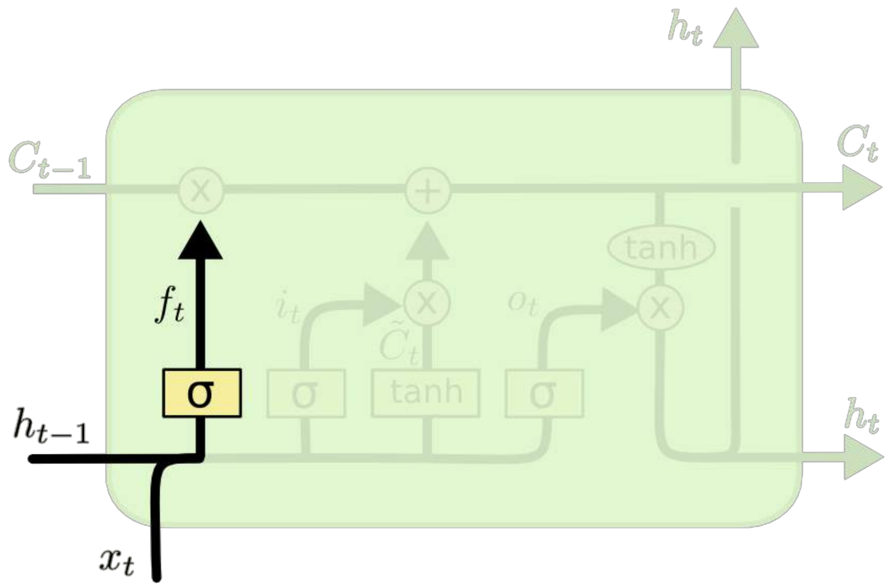
LSTM Conveyer Belt





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

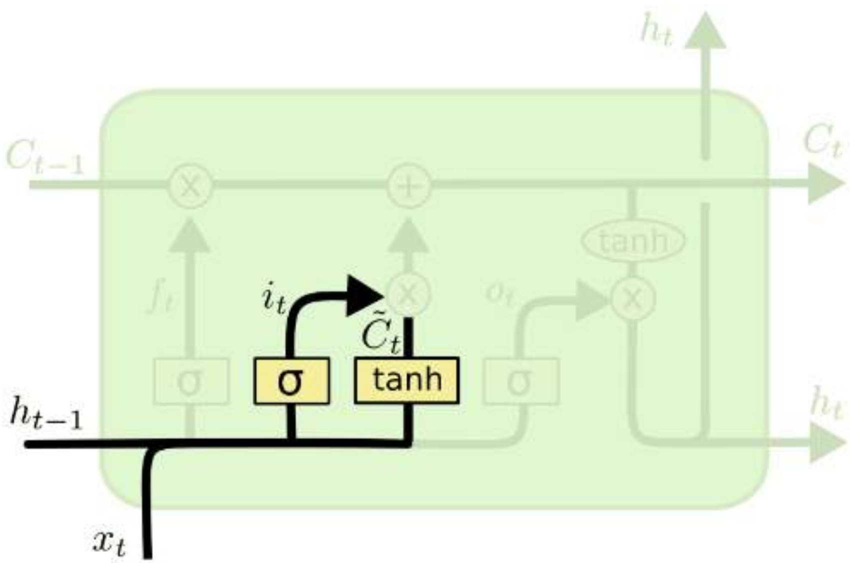
LSTM Conveyer Belt



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

Step 1: Decide what to forget / ignore

LSTM Conveyer Belt

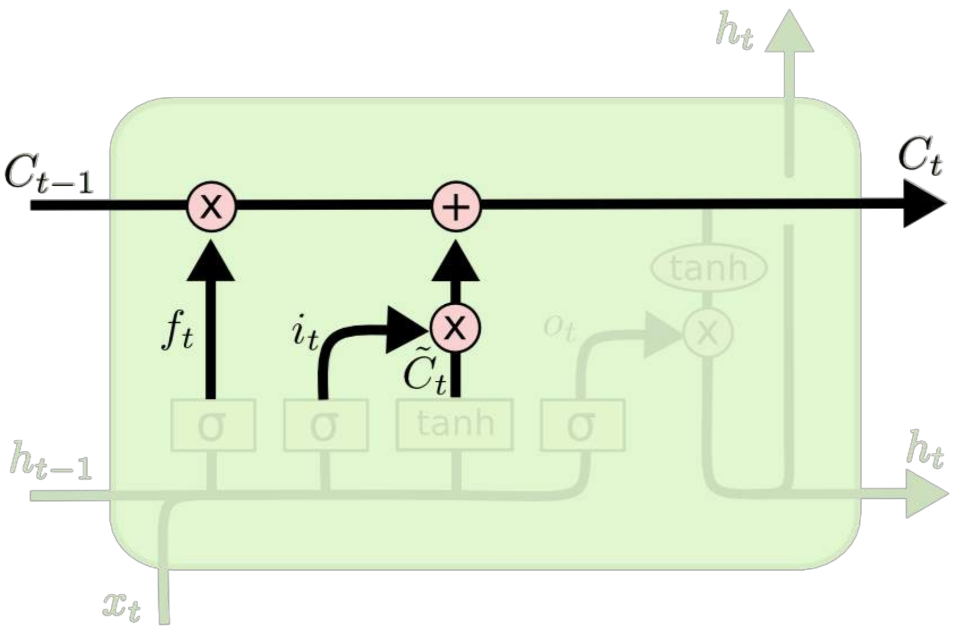


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

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

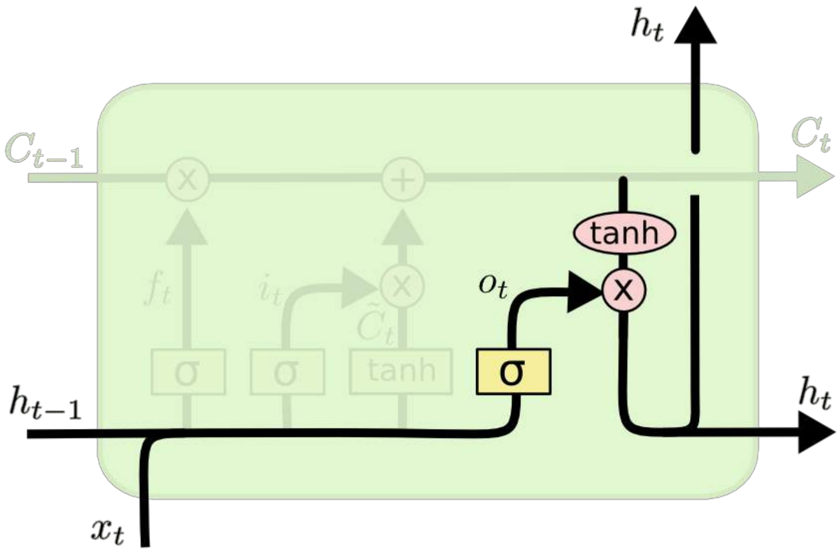
LSTM Conveyer Belt



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

Step 3: Perform the forgetting and the state update

LSTM Conveyer Belt

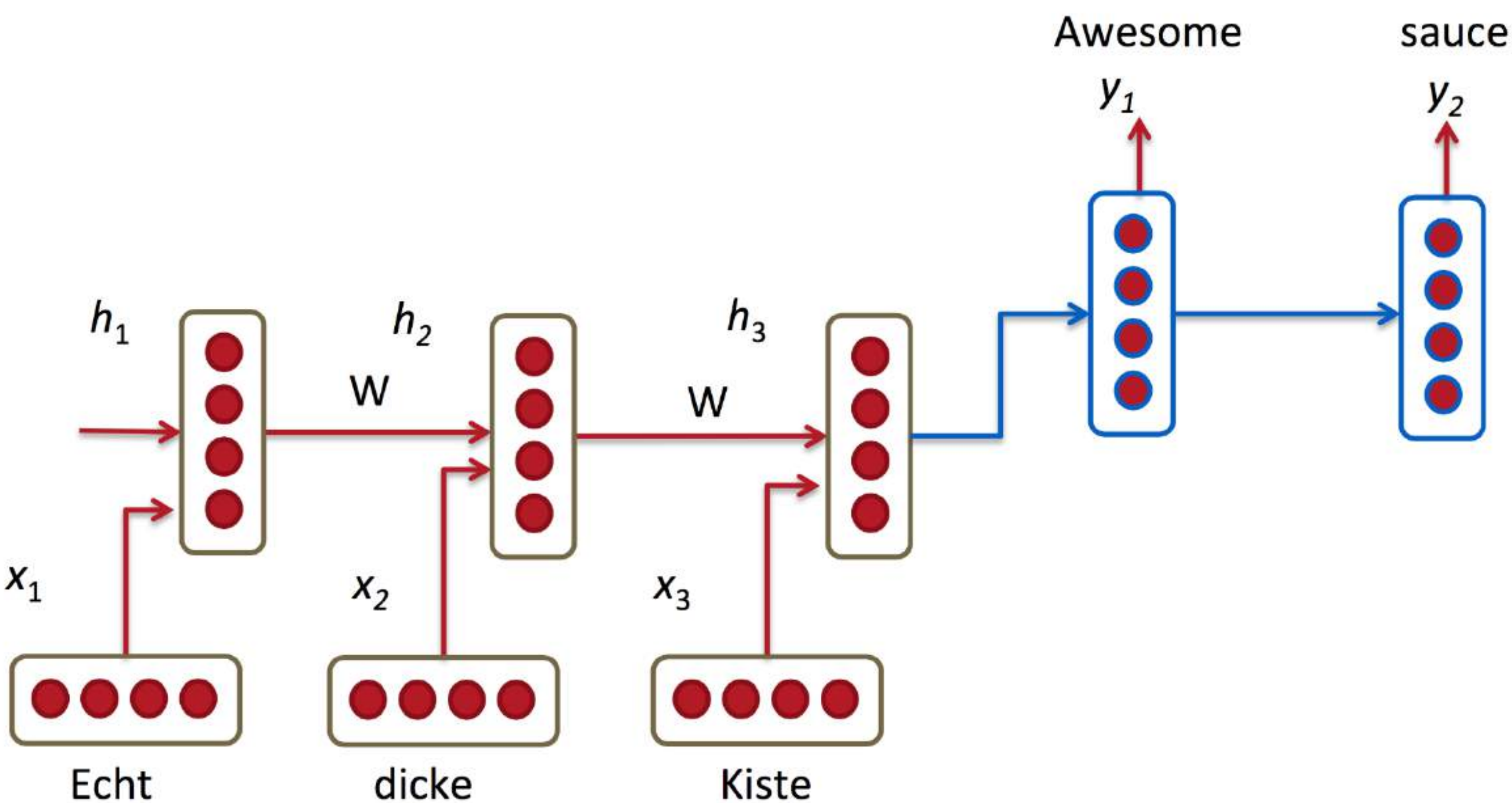


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

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

Application: Machine Translation



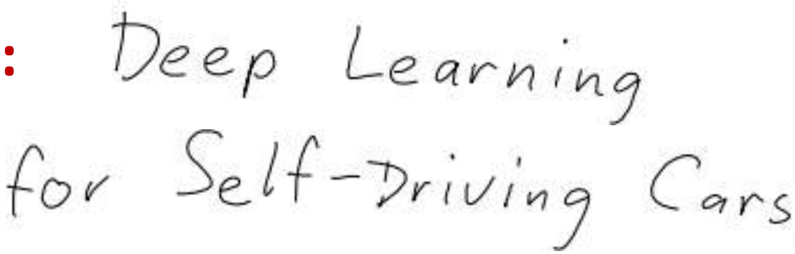
Application: Handwriting Generation from Text

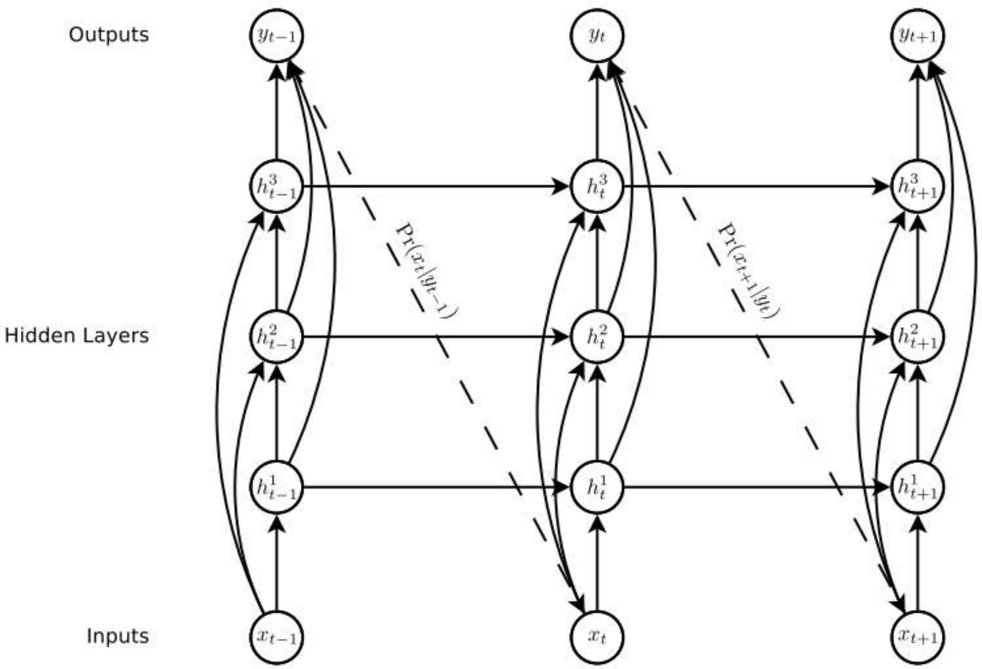
Input:

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

Deep Learning for Self Driving Cars

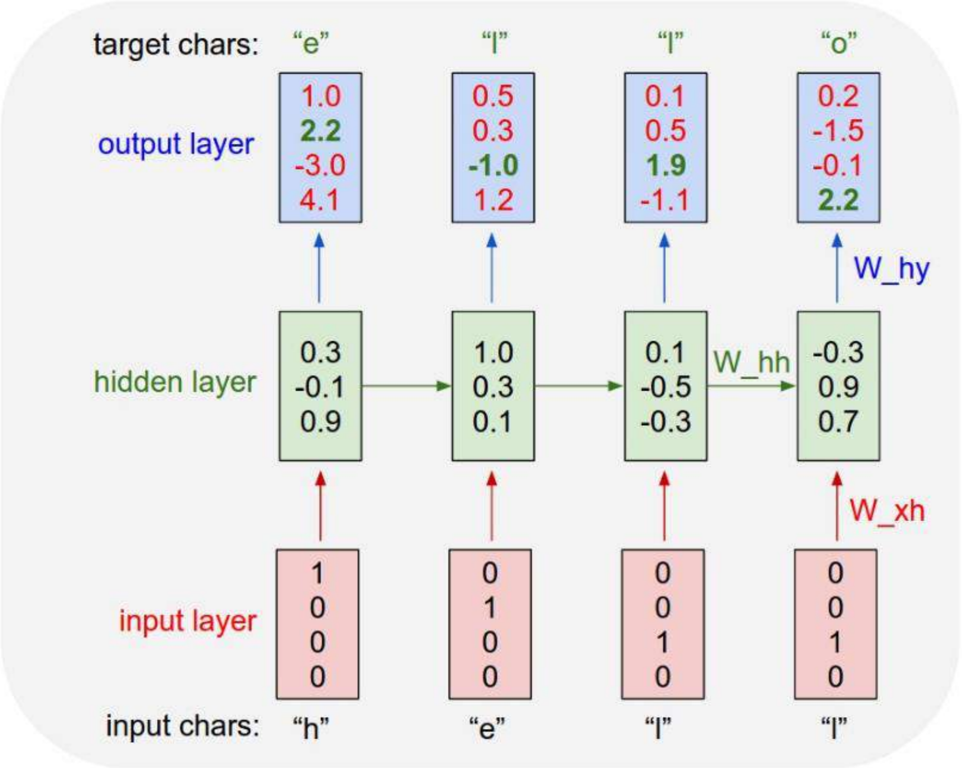
Output:





Alex Graves. **"Generating sequences with recurrent neural networks."** (2013).

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." (2015).

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)

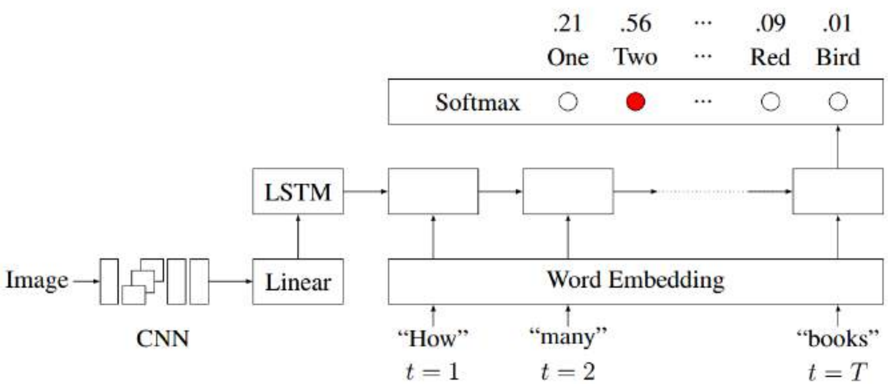
DAQUAR 1520
How many shelves are there?
Ground truth: three
IMG+BOW: three (0.25)
2-VIS+BLSTM: two (0.48)
LSTM: two (0.21)

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)



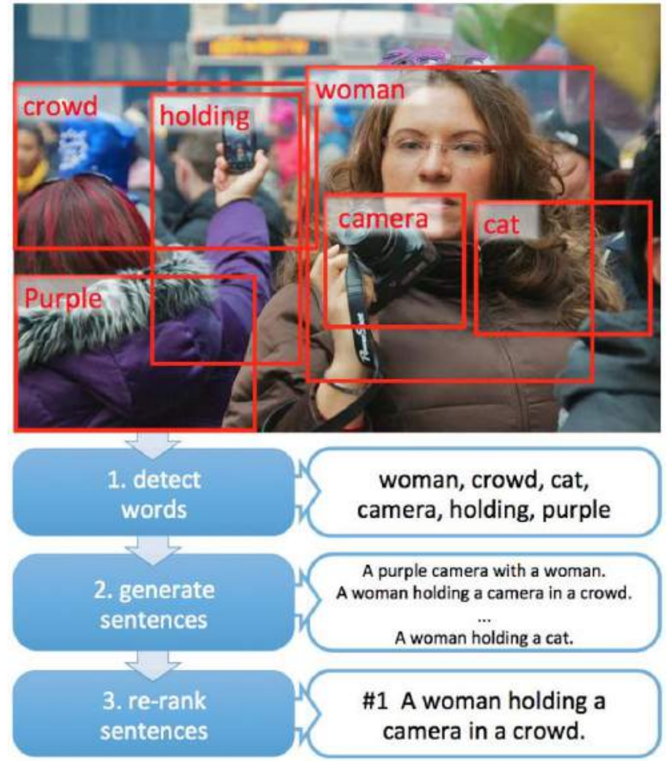
Ren et al. "Exploring models and data for image question answering." 2015.

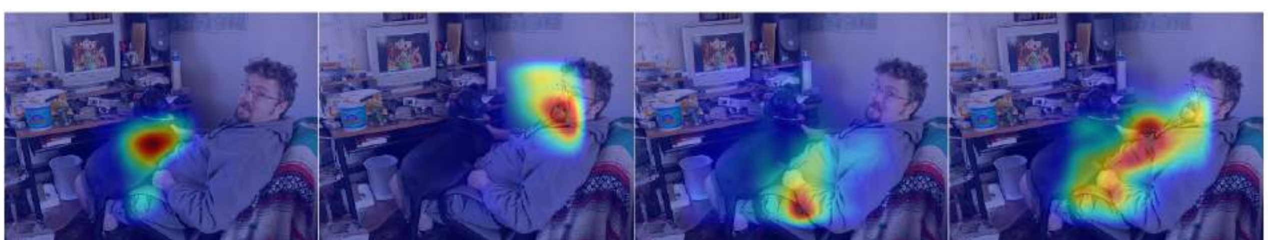
Code: https://github.com/renmengye/imageqa-public

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



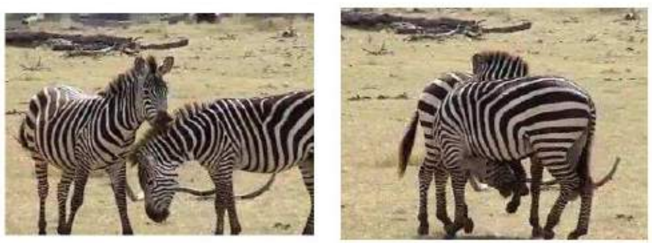


dog (1.00) man (0.93) sitting (0.83) couch (0.66)

Application: Video Description Generation

Correct descriptions.

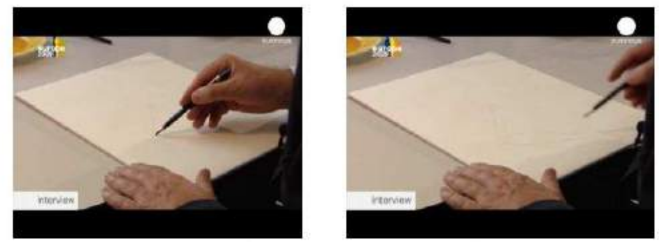
S2VT: A man is doing stunts on his bike.



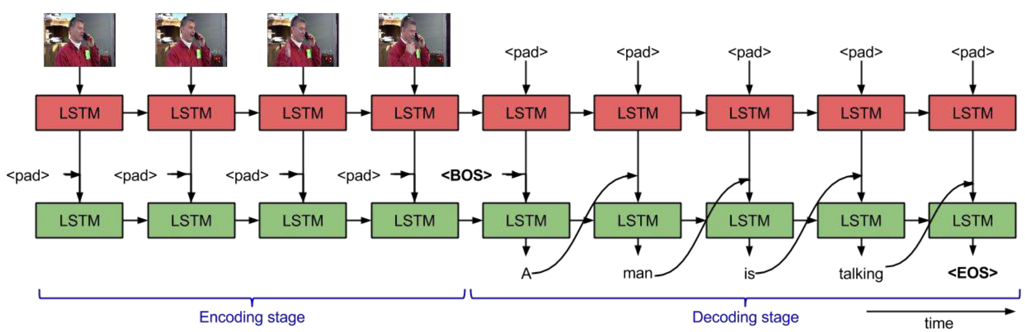
S2VT: A herd of zebras are walking in a field.

Relevant but incorrect descriptions.

S2VT: A small bus is running into a building.



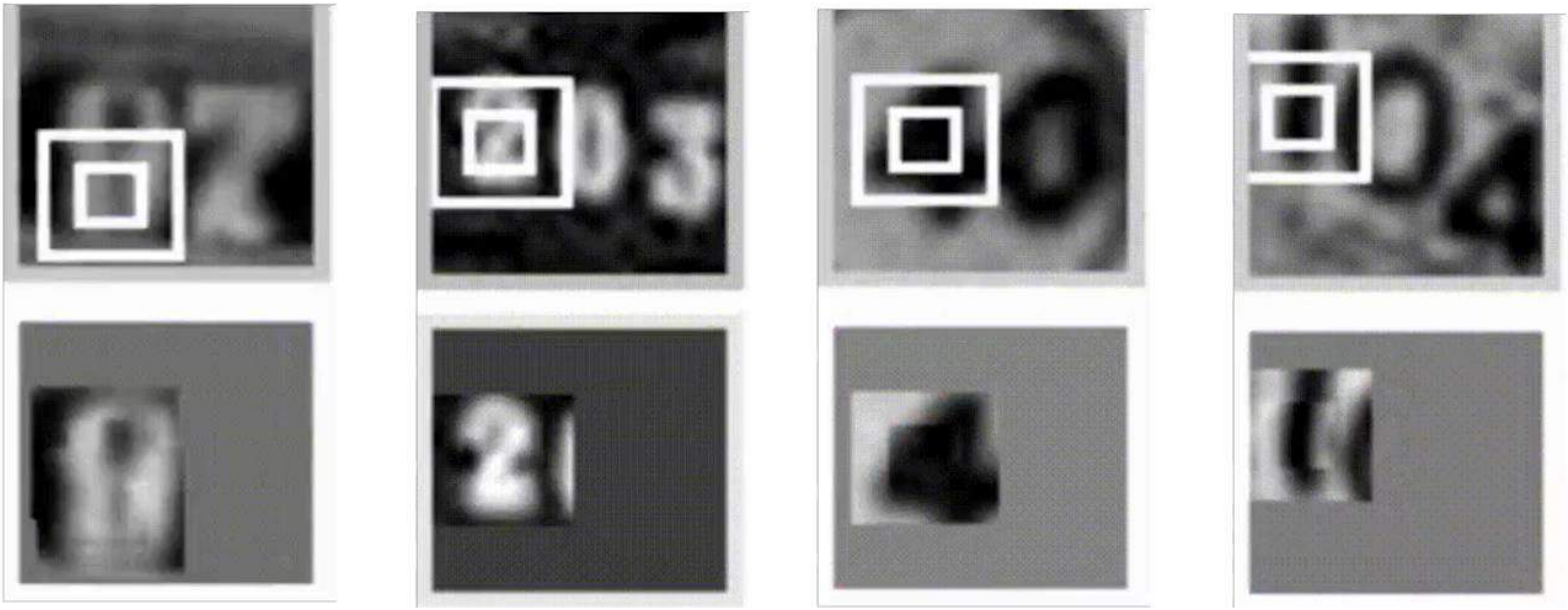
S2VT: A man is cutting a piece of a pair of a paper.

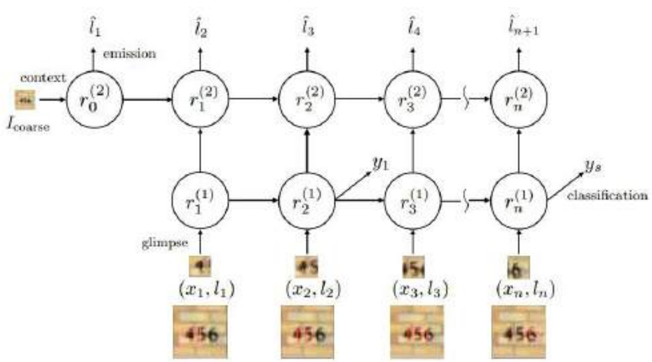


Venugopalan et al.
"Sequence to sequence-video to text." 2015.

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

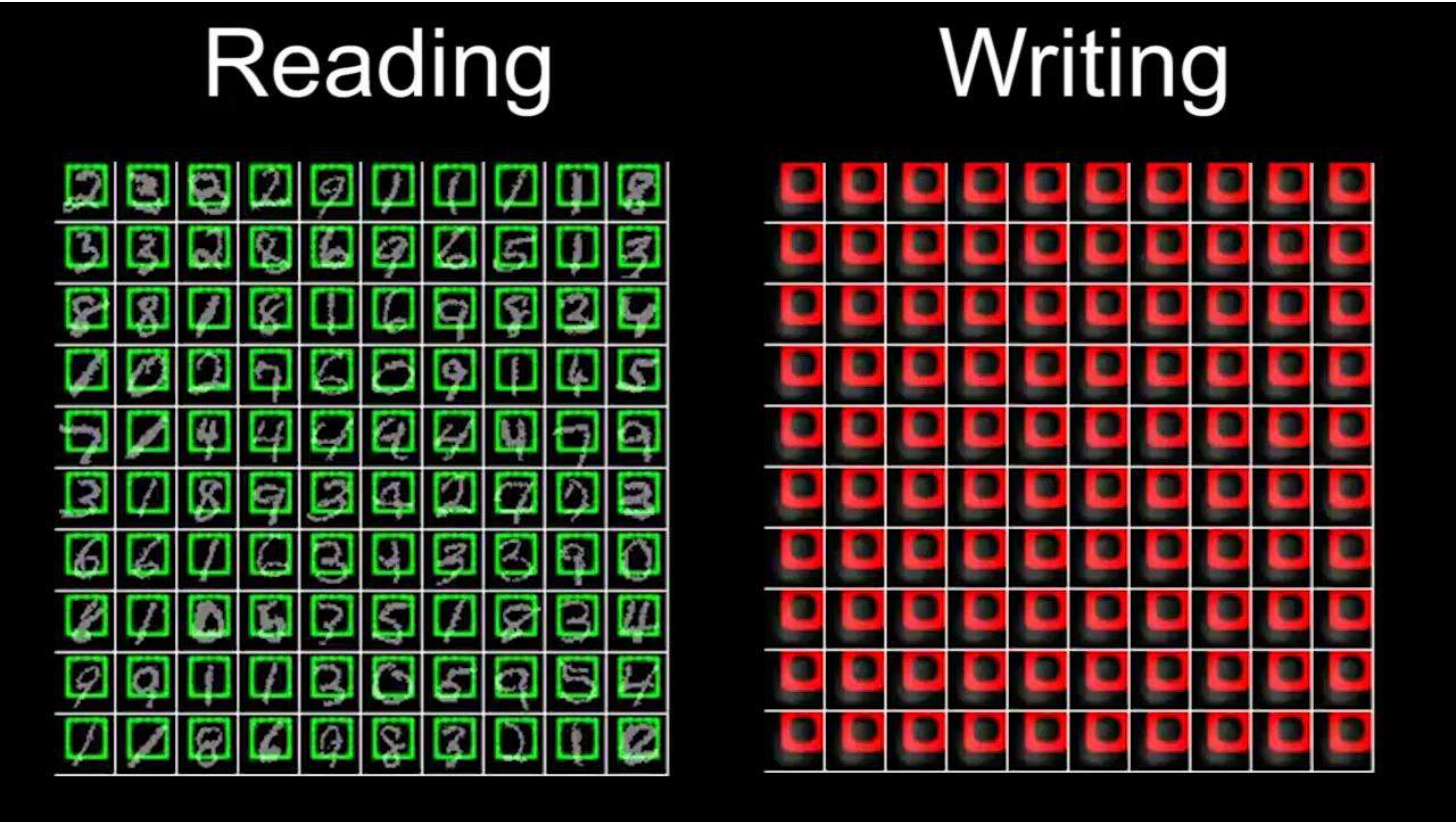
Application: Modeling Attention Steering





Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. **"Multiple object recognition with visual attention."** (2014).

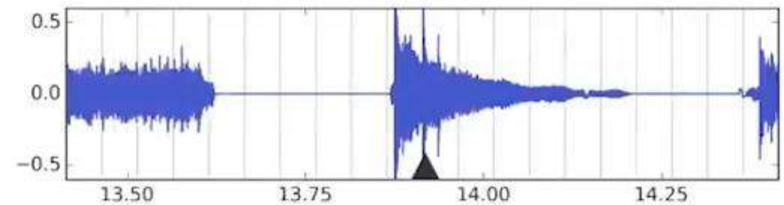
Application: Drawing with Selective Attention



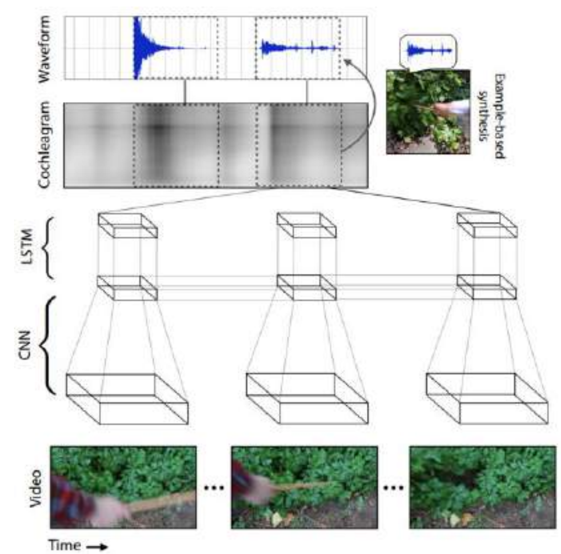
Gregor et al. "DRAW: A recurrent neural network for image generation." (2015). Code: https://github.com/ericjang/draw

Application: Adding Audio to Silent Film

Silent video

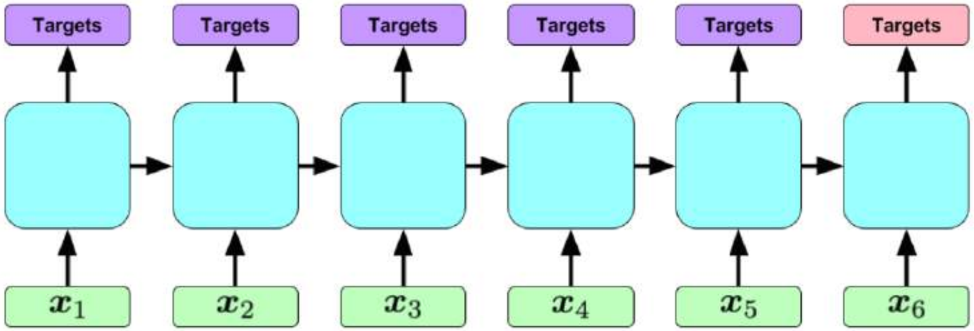


Predicted soundtrack



Owens, Andrew, Phillip Isola, Josh McDermott, Antonio Torralba, Edward H. Adelson, and William T. Freeman. **"Visually Indicated Sounds."** (2015).

Application: Medical Diagnosis



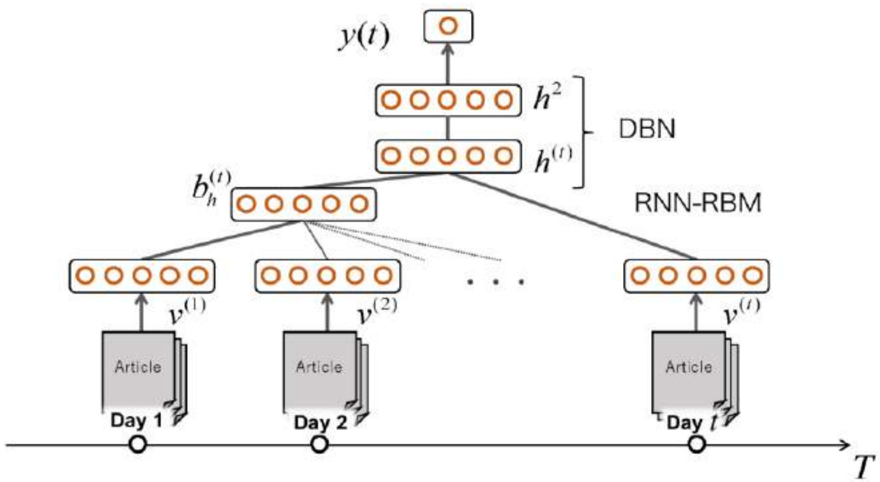
- **Input:** patients electronic health record (EHR) data over multiple visits (meaning, variable length sequences)
- **Output:** 128 diagnoses

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." (2015).

Application: Stock Market Prediction



Yoshihara et al. "Leveraging temporal properties of news events for stock market prediction." 2015.

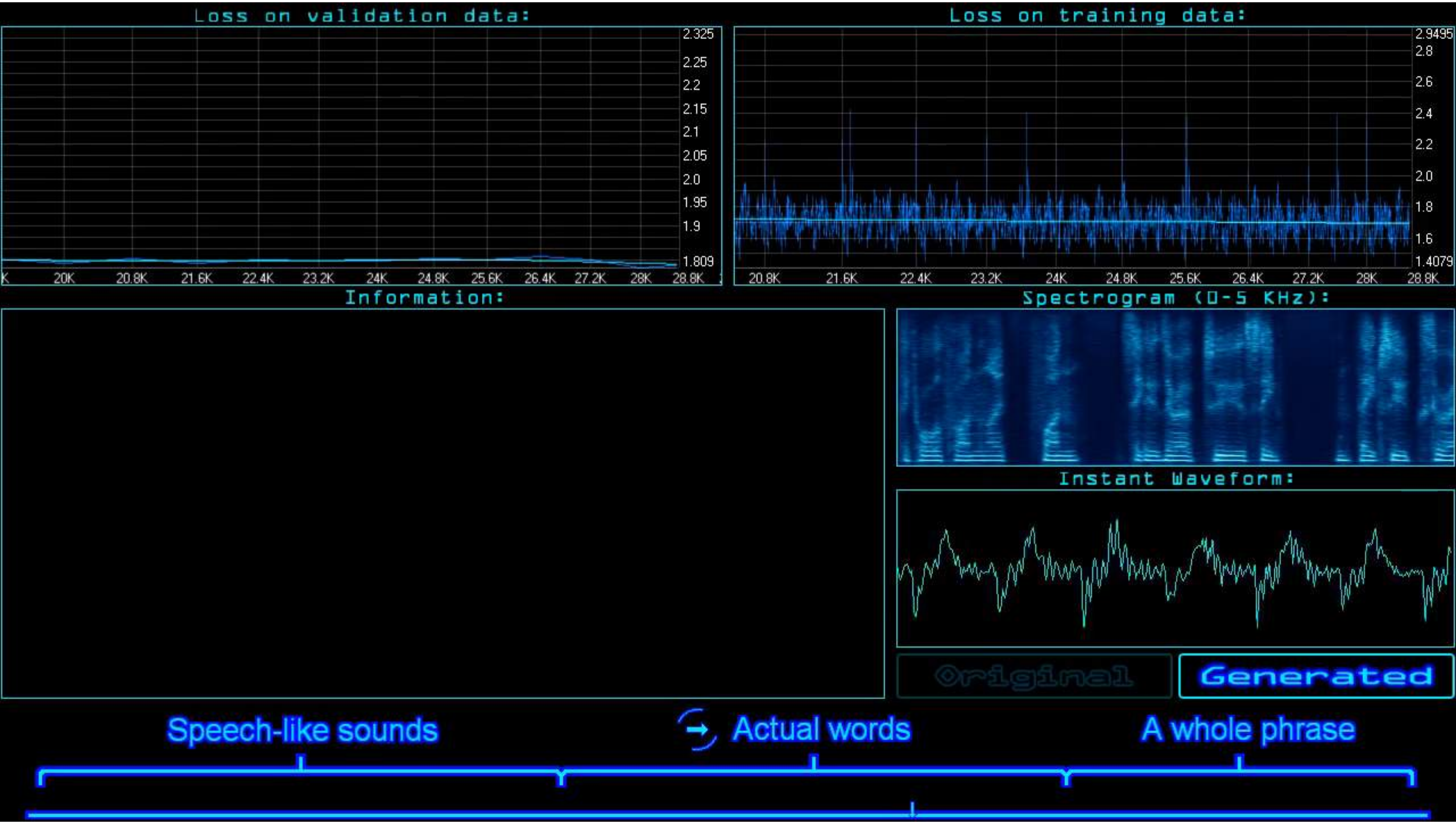
Table 3. Test error rates for stock price prediction

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

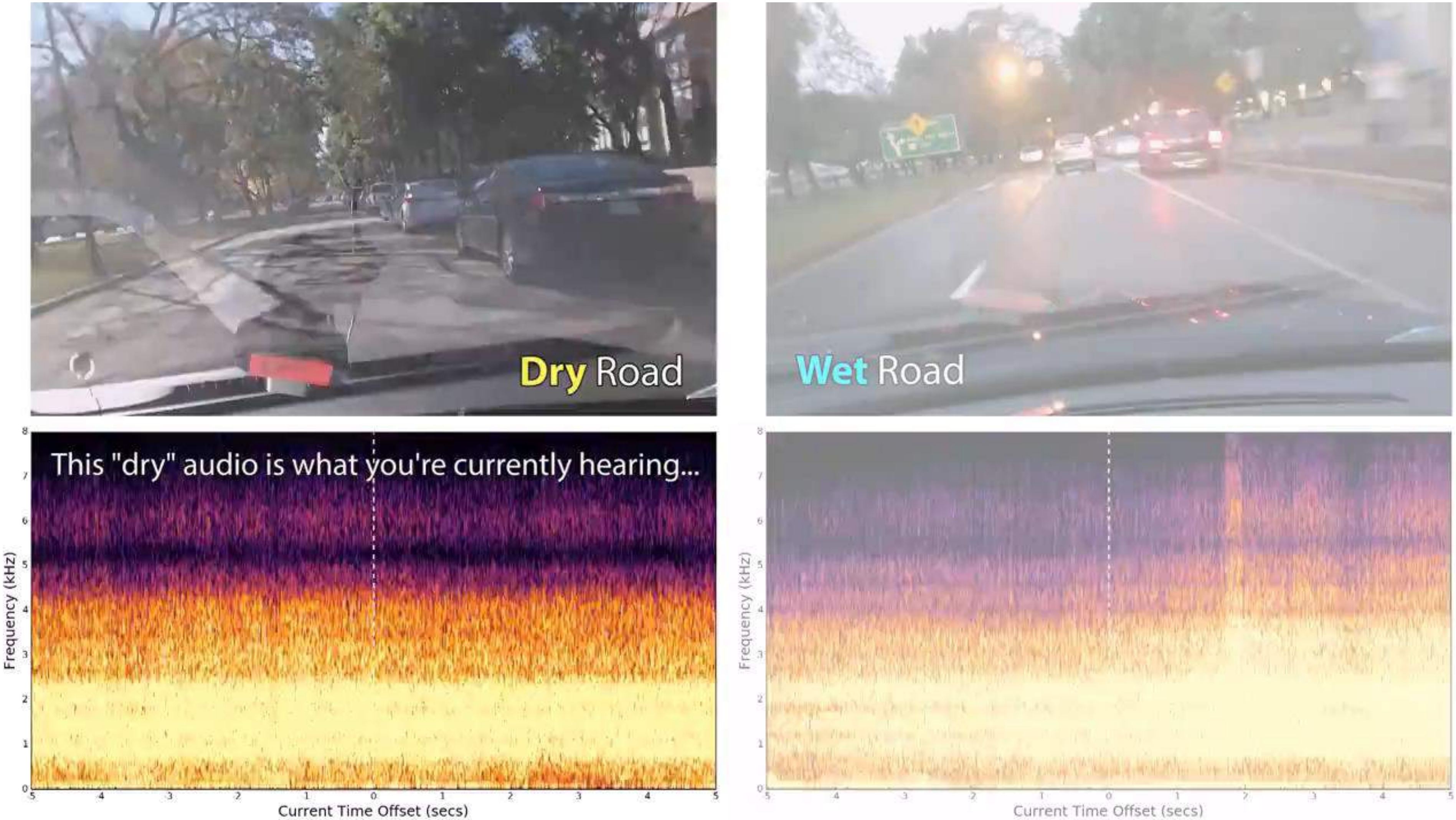
Brands	SVM	RNN-RBM + DBN
Nikkei Average	51.61	**38.70**
Hitachi	61.29	**32.25**
Toshiba	54.83	**38.70**
Fujitsu	45.16	**32.25**
Sharp	58.06	**45.16**
Sony	**41.93**	**41.93**
Nissan Motor	**29.03**	35.48
Toyota Motor	48.38	**45.16**
Canon	**54.83**	**54.83**
Mitsui	41.93	**38.70**
Mitsubishi	29.03	**25.80**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

Application: Audio Generation



(Using Torch-RNN, which is a more efficient fork of Char-RNN)

Application: Audio Classification

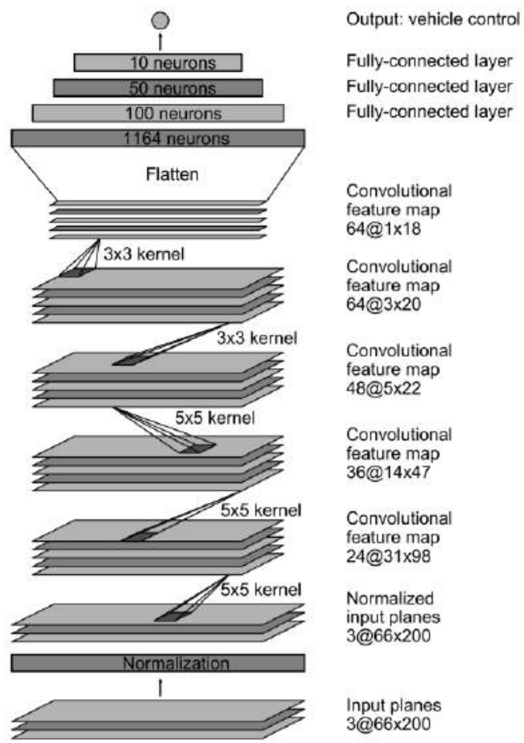


References: [131]

Reminder:

Original NVIDIA Approach to End-to-End Driving

- 9 layers
  - 1 normalization layer
  - 5 convolutional layers
  - 3 fully connected layers
- 27 million connections
- 250 thousand parameters



DeepTesla: End-to-End Driving with ConvnetJS

Tutorial on http://cars.mit.edu/deeptesla

End-to-End Driving with RNNs

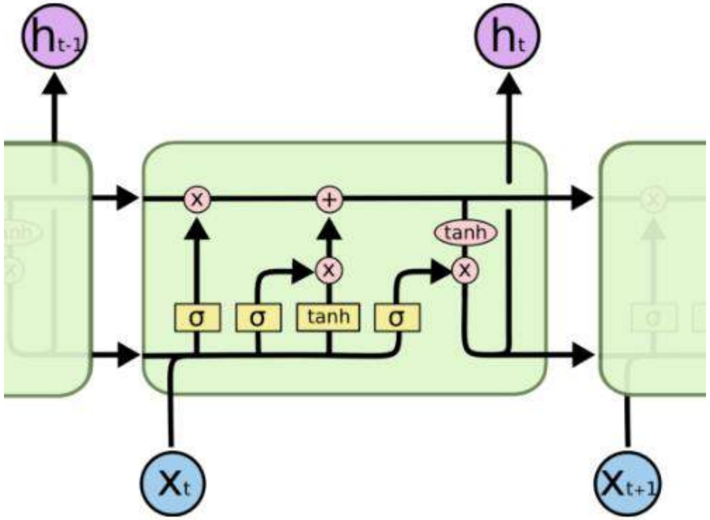
UDACITY
Self-Driving Car Engineer Nanodegree

1st and 3rd place winner of the Udacity end-to-end steering competition used RNNs:

- Sequence-to-sequence mapping from images to steering angles.

End-to-End Driving with RNNs

(1st Place Winner: Team Komanda)

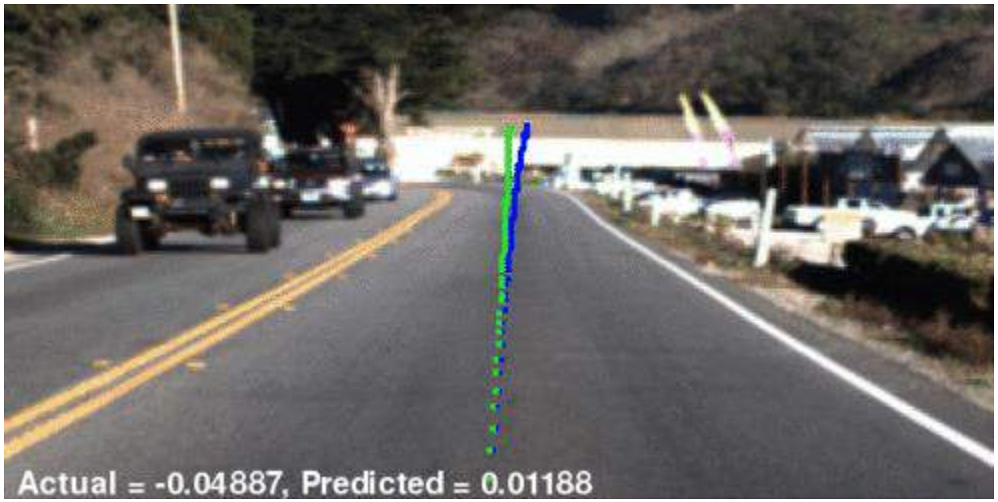


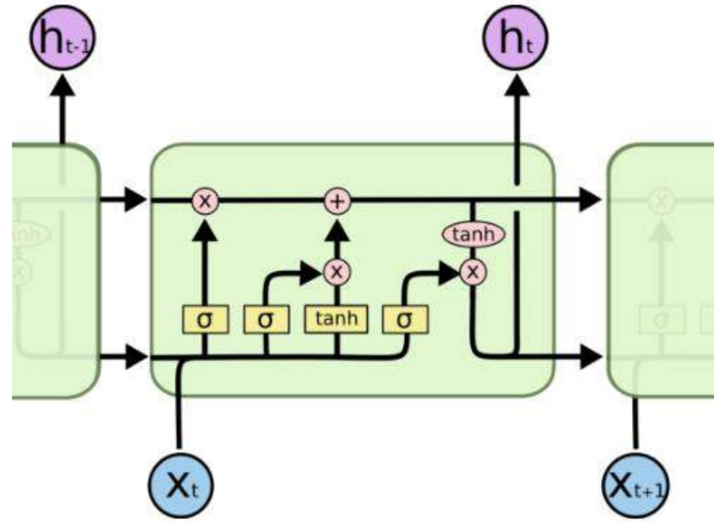
x: 3d convolution of image sequence

h: predicted steering angle, speed, torque

Sequence length: 10

End-to-End Driving with RNNs
(3rd Place Winner: Team Chauffeur)





Transfer learning: stacked CNN (pruned to 3000 features)

x: 3000 features extracted with CNN

h: predicted steering angle

Sequence length: 50

References

All references cited in this presentation are listed in the following Google Sheets file:

https://goo.gl/9Xhp2t