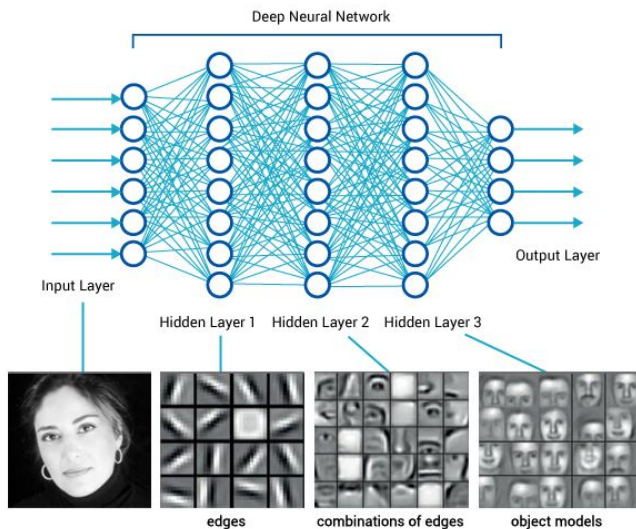


Recurrent Neural Network

The Deepest of Deep Learning

Chen Liang



Deep Learning

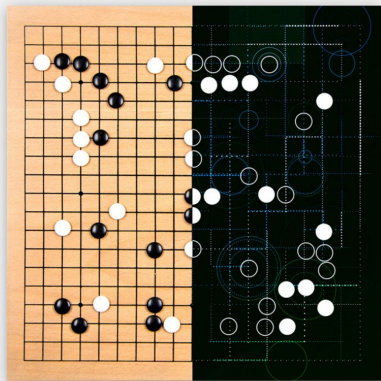
Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.

AI Is Transforming Google Search. The Rest of the Web Is Next

**IN A HUGE BREAKTHROUGH,
GOOGLE'S AI BEATS A TOP
PLAYER AT THE GAME OF GO**

Drive.ai to test 'deep learning'
autonomous cars on California roads



Baidu's Deep-Learning System Rivals People at Speech Recognition

Deep learning

Field of study

Machine learning

Field of study

Artificial neural n..

Topic

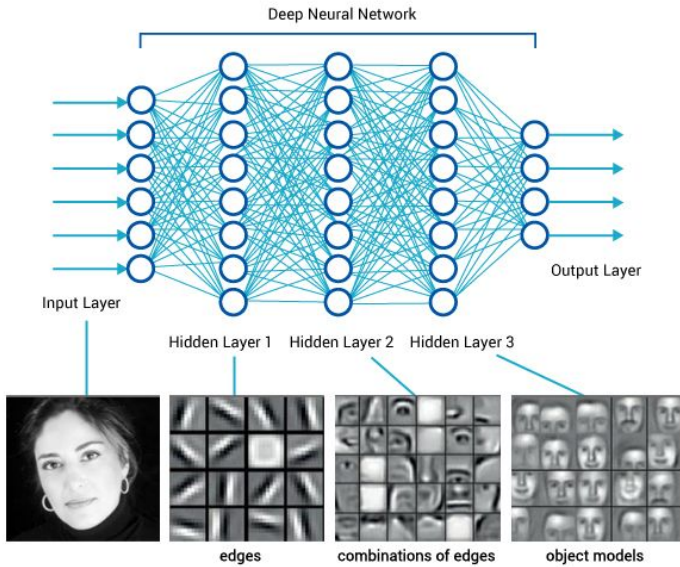
+ Add term

Beta: Measuring search interest in *topics* is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific *query*, select the "search term" option. ?

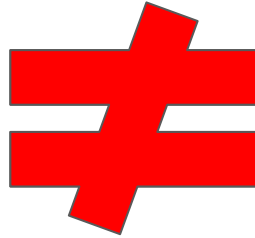
Interest over time ?

News headlines ? Forecast ?





Demystify Deep Learning



Deep learning works
like human brain?





Demystify Deep Learning

Deep learning works
like human brain?





Deep Learning



Deep learning works
like human brain?



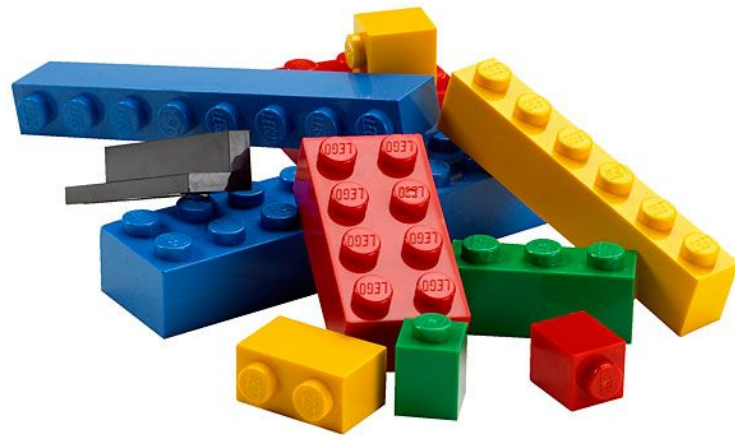
Deep Learning: Building Blocks

- ▶ Linear transformations

$$h_{k+1} = Wh_k$$

- ▶ Non-linear activation functions

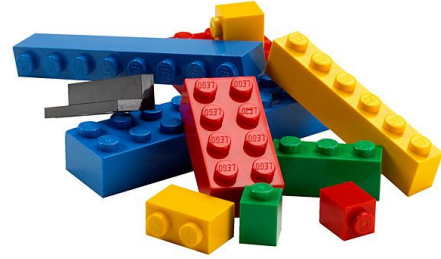
$$h_{k+2} = f(h_{k+1})$$



Deep Learning: Deep Composition

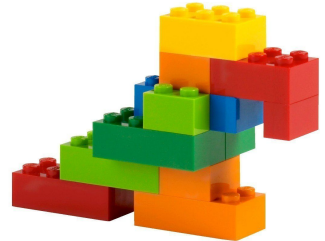
- ▶ A **deep representation** is a composition of many functions

$$x \xrightarrow{w_1} h_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} h_n \xrightarrow{w_{n+1}} y$$



- ▶ Its gradient can be **backpropagated** by the chain rule

$$\begin{array}{ccccccc} \frac{\partial h_1}{\partial x} & \leftarrow & \frac{\partial h_2}{\partial h_1} & \leftarrow & \dots & \leftarrow & \frac{\partial y}{\partial h_n} & \leftarrow & \frac{\partial}{\partial y} \\ & & \downarrow & & & & \downarrow & & \downarrow \\ & & \frac{\partial h_1}{\partial w_1} & & \dots & & \frac{\partial h_n}{\partial w_n} & & \frac{\partial y}{\partial w_{n+1}} \end{array}$$



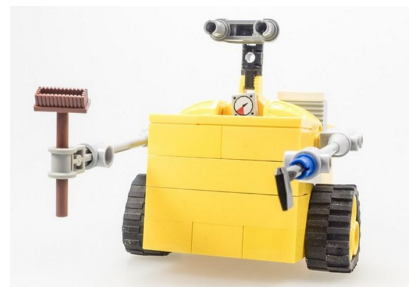
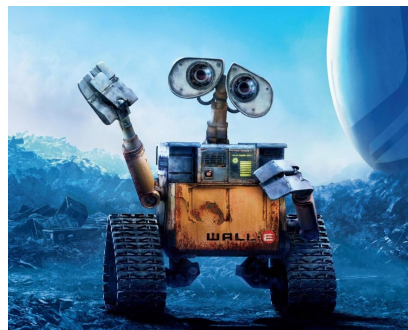
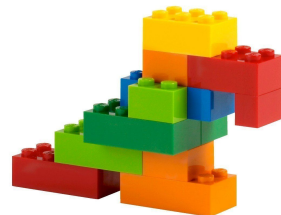
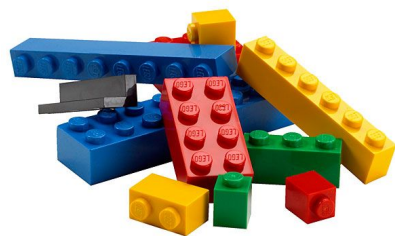
Deep Learning: Gradient Descent

- ▶ A **loss function** $l(y)$ measures goodness of output y , e.g.
 - ▶ Mean-squared error $l(y) = ||y^* - y||^2$
 - ▶ Log likelihood $l(y) = \log \mathbb{P}[y^*|x]$
- ▶ The loss is appended to the forward computation

$$x \xrightarrow{w_1} h_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} h_n \xrightarrow{w_{n+1}} y \longrightarrow l(y)$$

- ▶ Gradient of loss is appended to the backward computation

$$\begin{array}{ccccccc} \frac{\partial h_1}{\partial x} & \longleftarrow & \frac{\partial h_2}{\partial h_1} & \longleftarrow & \dots & \longleftarrow & \frac{\partial y}{\partial h_n} \longleftarrow \frac{\partial l(y)}{\partial y} \\ & & \downarrow & & & & \downarrow \\ & & \frac{\partial h_1}{\partial w_1} & & \dots & & \frac{\partial h_n}{\partial w_n} \quad \frac{\partial y}{\partial w_{n+1}} \end{array}$$



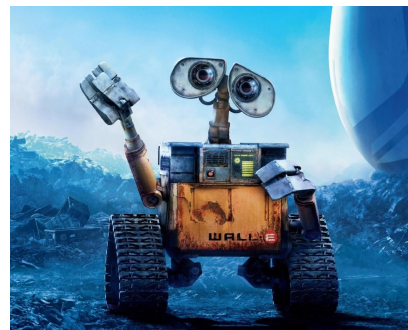
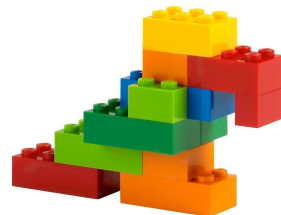
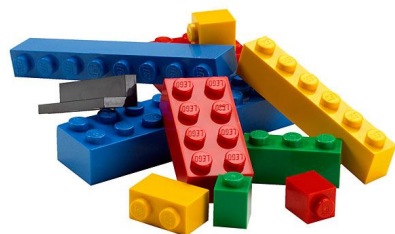
Deep Learning: Gradient Descent

- ▶ A **loss function** $l(y)$ measures goodness of output y , e.g.
 - ▶ Mean-squared error $l(y) = ||y^* - y||^2$
 - ▶ Log likelihood $l(y) = \log \mathbb{P}[y^*|x]$
- ▶ The loss is appended to the forward computation

$$x \xrightarrow{w_1} h_1 \xrightarrow{w_2} \dots \xrightarrow{w_n} h_n \xrightarrow{w_{n+1}} y \longrightarrow l(y)$$

- ▶ Gradient of loss is appended to the backward computation

$$\begin{array}{ccccccc} \frac{\partial h_1}{\partial x} & \leftarrow & \frac{\partial h_2}{\partial h_1} & \leftarrow & \dots & \leftarrow & \frac{\partial y}{\partial h_n} \leftarrow \frac{\partial l(y)}{\partial y} \\ & & \downarrow & & & & \downarrow \\ & & \frac{\partial h_1}{\partial w_1} & & \dots & & \frac{\partial h_n}{\partial w_n} \quad \frac{\partial y}{\partial w_{n+1}} \end{array}$$



Deep Learning: Demo TensorFlow Playground



Iterations
000,000

Learning rate

0.03

Activation

ReLU

Regularization

L2

Regularization rate

0

Problem type

Classification

DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

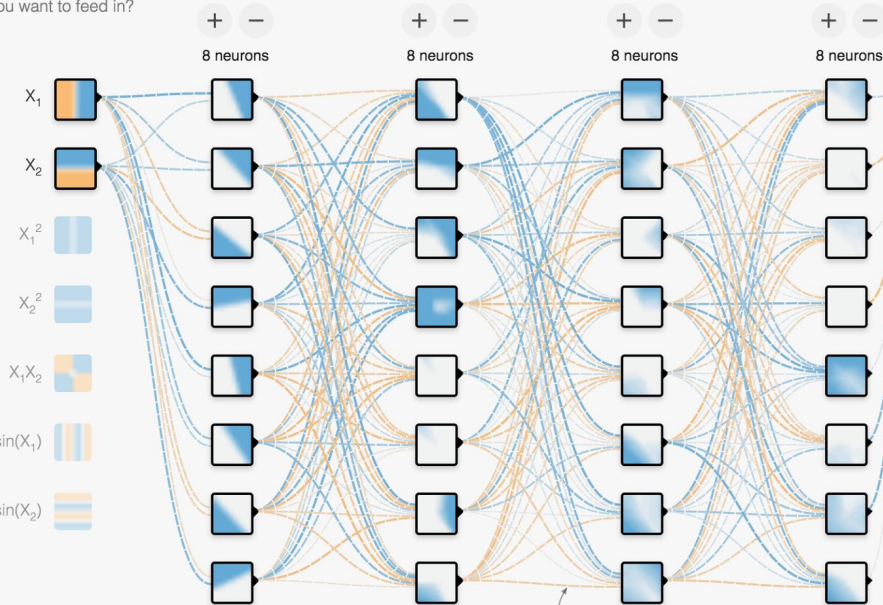
REGENERATE

INPUT

Which properties do you want to feed in?

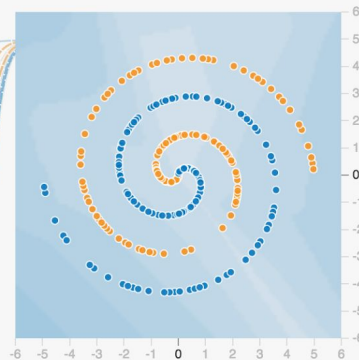
X_1
 X_2
 X_1^2
 X_2^2
 $X_1 X_2$
 $\sin(X_1)$
 $\sin(X_2)$

4 HIDDEN LAYERS



OUTPUT

Test loss 0.544
Training loss 0.534



Colors shows data, neuron and weight values.

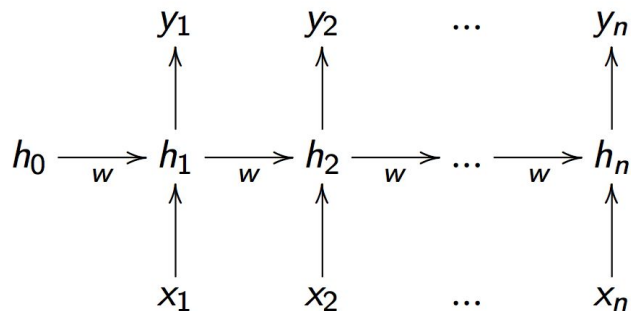
☐ Show test data ☐ Discretize output

This is the output from one neuron. Hover to see it.

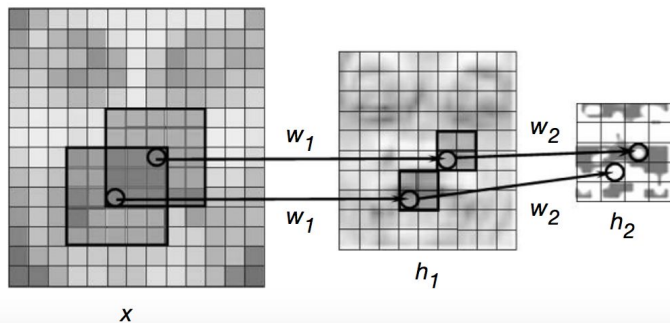
The outputs are mixed with varying weights, shown by the thickness.

Deep Learning: Weight Sharing

Recurrent neural network shares weights between time-steps



Convolutional neural network shares weights between local regions



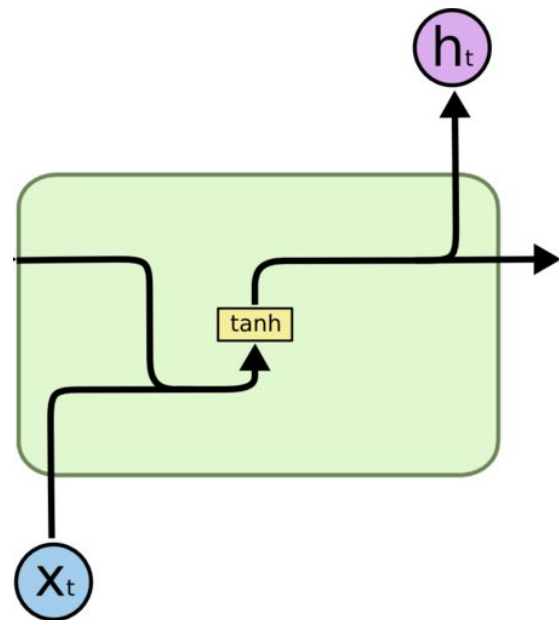
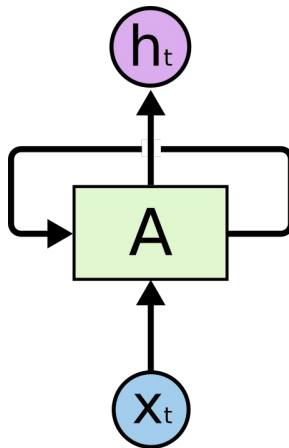
Recurrent Neural Network

Deepest of Deep learning?

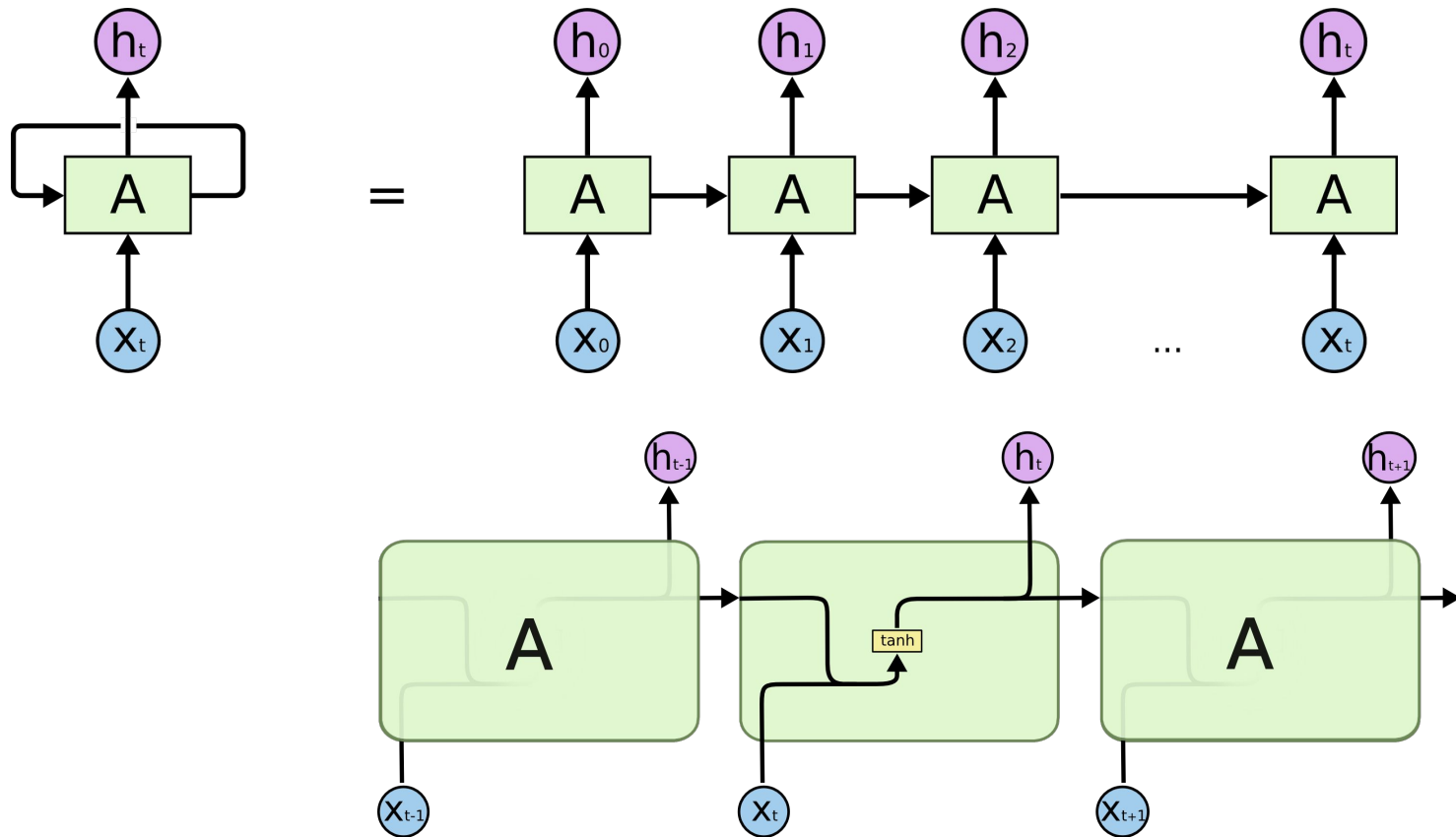
- Can be infinitely deep

Equation

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

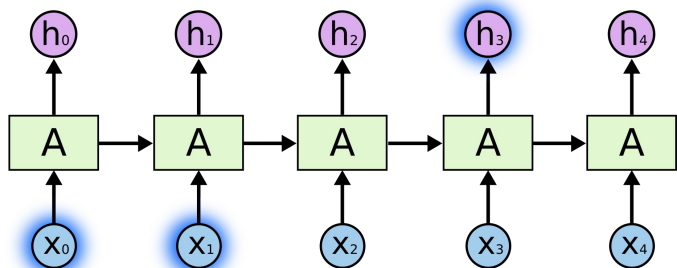


BPTT: Backpropagation Through Time

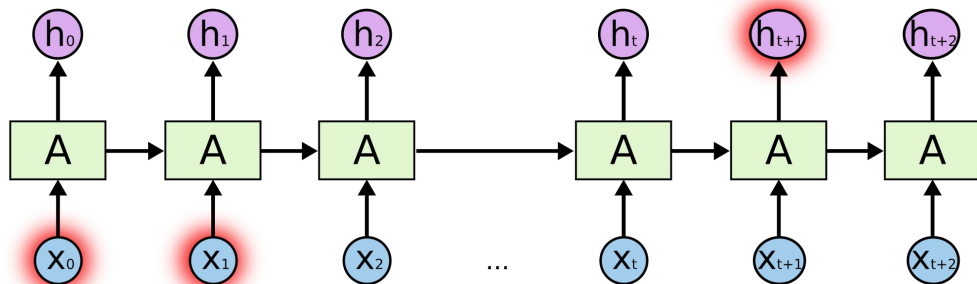


Recurrent Neural Network

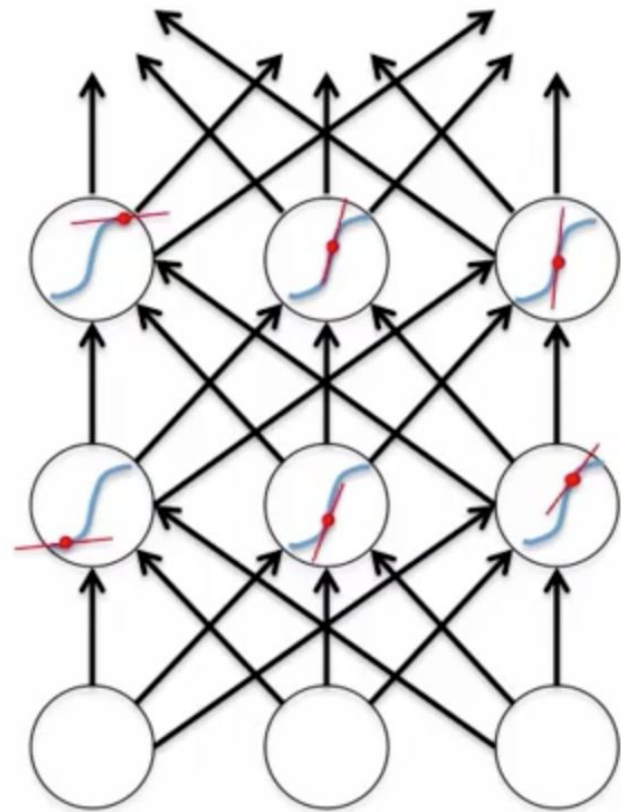
Short term dependency



Long term dependency

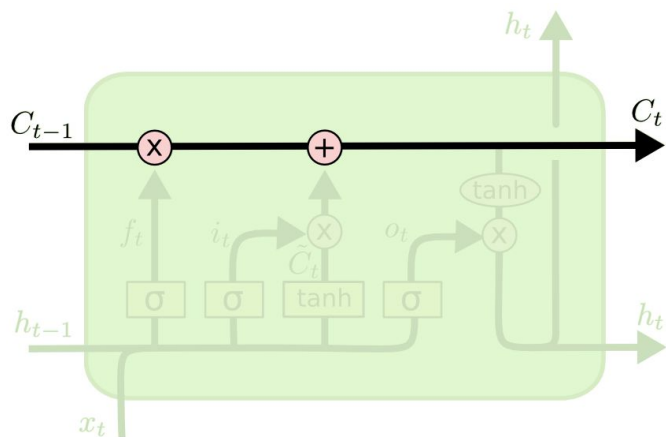


Exploding/vanishing gradient



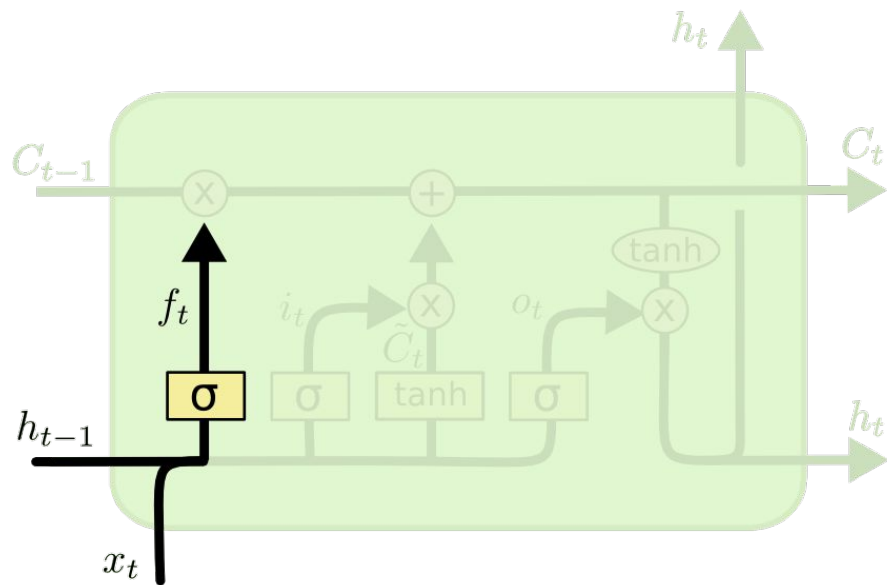
LSTM: Long-Short Term Memory

Add a direct pathway for gradient



LSTM: Long-Short Term Memory

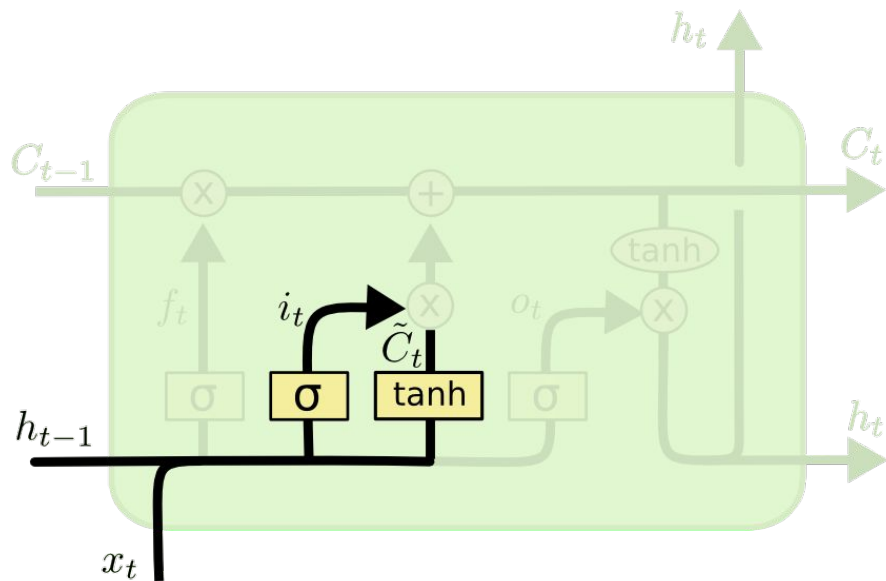
Forget gate



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

LSTM: Long-Short Term Memory

Input gate

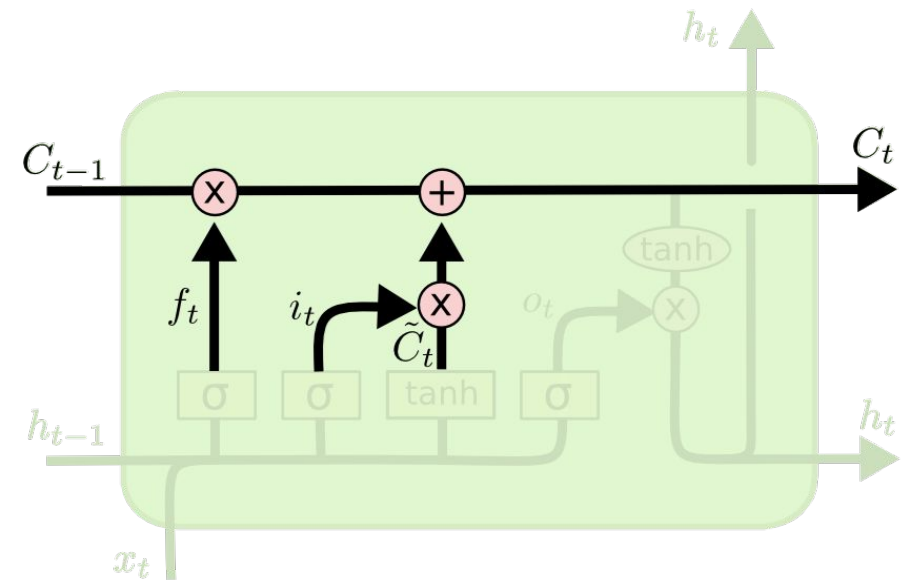


$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

LSTM: Long-Short Term Memory

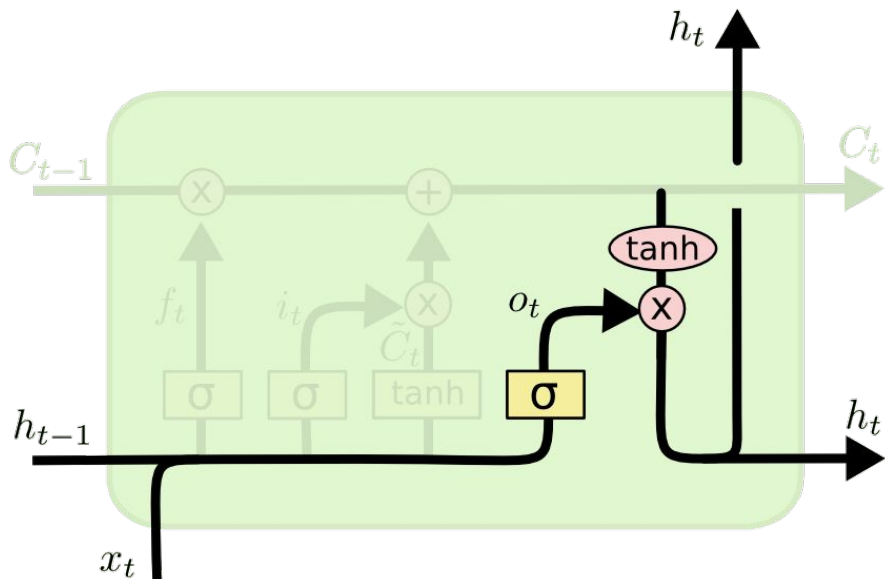
Update memory using forget gate and input gate



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

LSTM: Long-Short Term Memory

Output gate

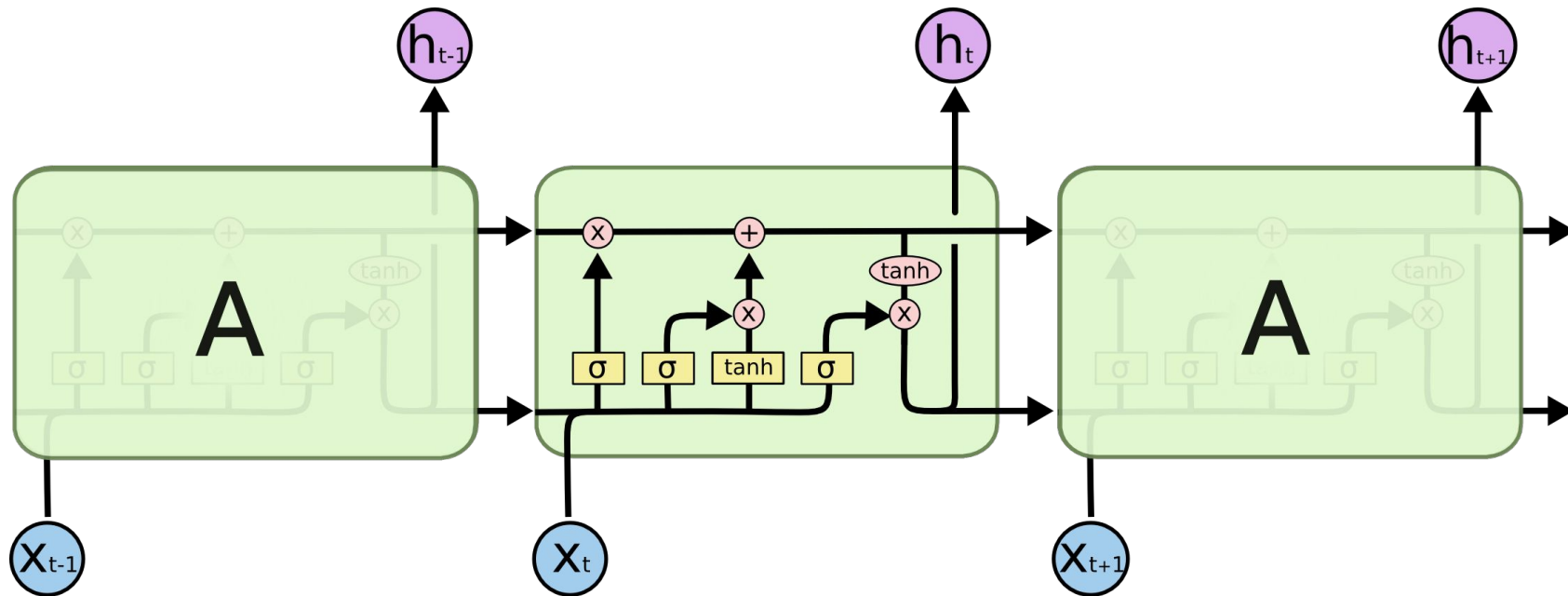


$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

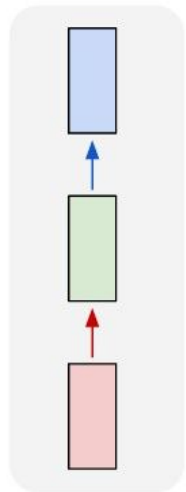
LSTM: Long-Short Term Memory

Putting together



RNN: A General Framework

one to one



one to many

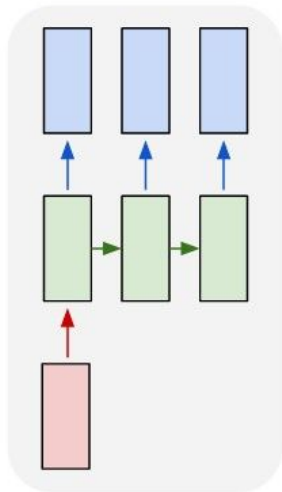
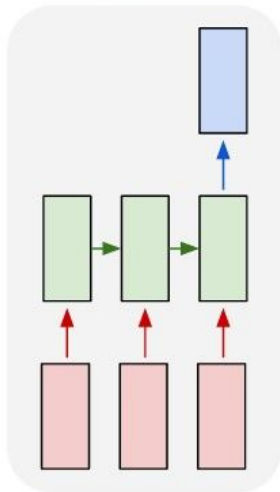


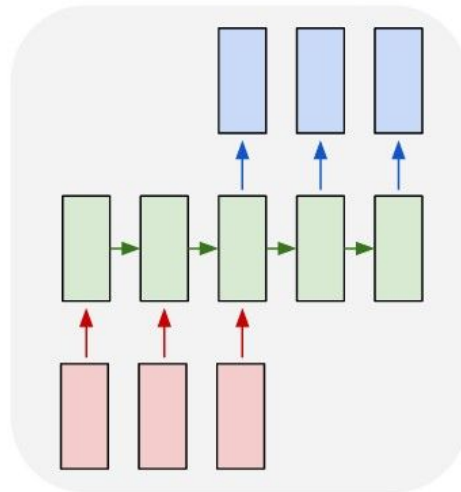
Image Caption
Generation

many to one



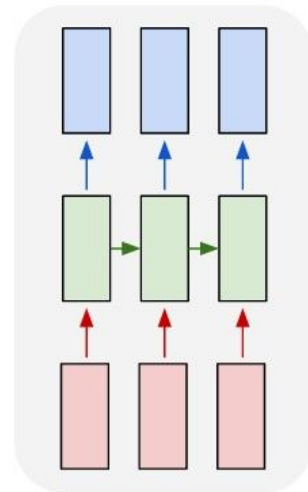
Sentiment Analysis
Image Recognition

many to many



Machine Translation
Speech recognition

many to many



Language Modeling

Char-RNN

How it works?

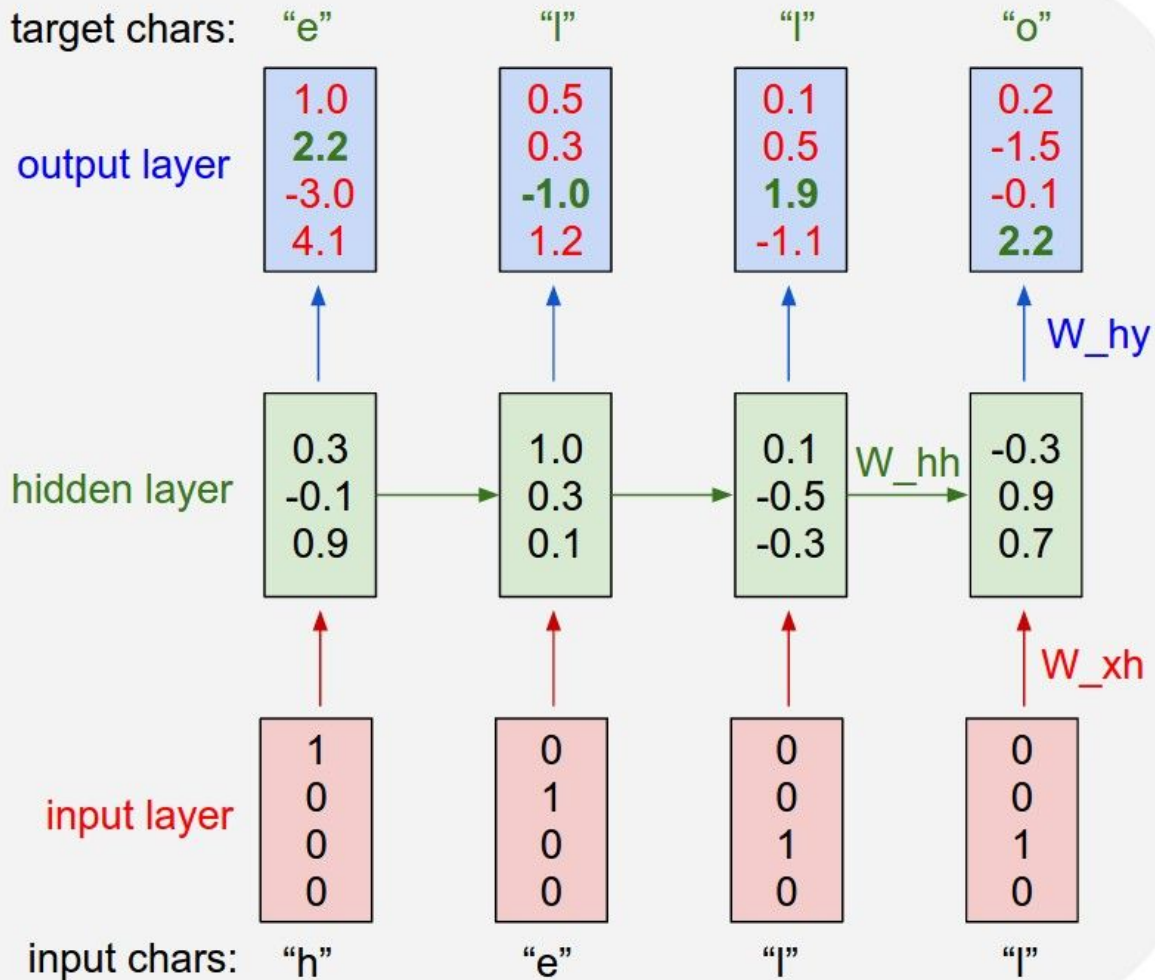
Vocabulary:

["h", "e", "l", "o"]

Training

sequence:

"hello"



Char-RNN

Linux

Latex

Wikipedia

Music

...

Check out the blog:

[The Unreasonable Effectiveness of RNN](#)



DeepDrumpf

@DeepDrumpf

Follow

We have competence. Our people don't need anybody. I have smart people.

1:46 PM - 3 Mar 2016

127 101



DeepDrumpf

@DeepDrumpf

Follow

I'll bring back our jobs. They all have everything, they don't have a clue.

1:36 PM - 3 Mar 2016

24 21



DeepDrumpf

@DeepDrumpf

Follow

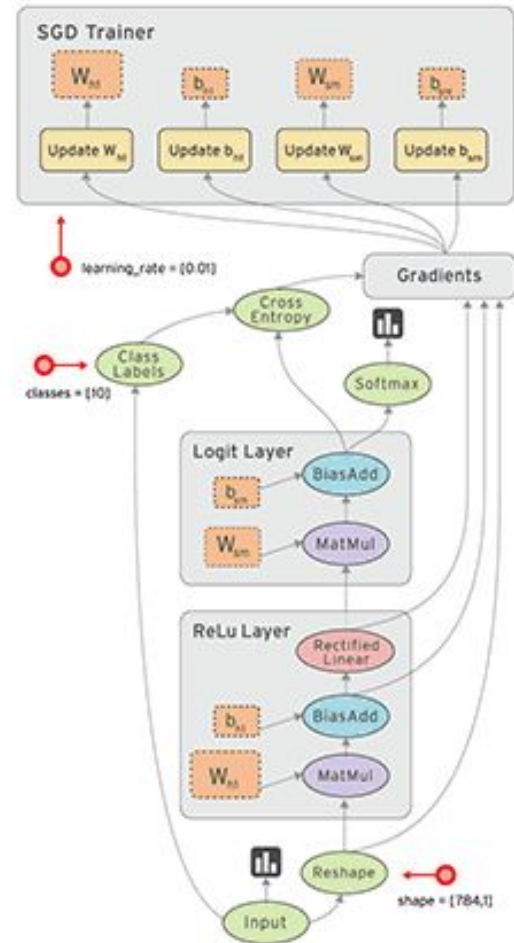
OK, it's amazing right now with ISIS, I tell you what? I don't want them to vote, the worst very social people. I love me.

1:49 PM - 3 Mar 2016

327 371

TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs.



TensorFlow: Computation Graph

```
import tensorflow as tf
import numpy as np
```

```
# Create 100 phony x, y data points in NumPy,  $y = x * 0.1 + 0.3$ 
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3
```

```
# Try to find values for W and b that compute  $y\_data = W * x\_data + b$ 
```

```
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
```

```
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
```

```
b = tf.Variable(tf.zeros([1]))
```

```
y = W * x_data + b
```

```
# Minimize the mean squared errors.
```

```
loss = tf.reduce_mean(tf.square(y - y_data))
```

```
optimizer = tf.train.GradientDescentOptimizer(0.5)
```

```
train = optimizer.minimize(loss)
```

Import TensorFlow
and NumPy

TensorFlow: Computation Graph

```
import tensorflow as tf
import numpy as np
```

```
# Create 100 phony x, y data points in NumPy,  $y = x * 0.1 + 0.3$ 
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3
```

```
# Try to find values for W and b that compute  $y\_data = W * x\_data + b$ 
```

```
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
```

```
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
```

```
b = tf.Variable(tf.zeros([1]))
```

```
y = W * x_data + b
```

```
# Minimize the mean squared errors.
```

```
loss = tf.reduce_mean(tf.square(y - y_data))
```

```
optimizer = tf.train.GradientDescentOptimizer(0.5)
```

```
train = optimizer.minimize(loss)
```

Synthesize some
noisy data from a
linear model

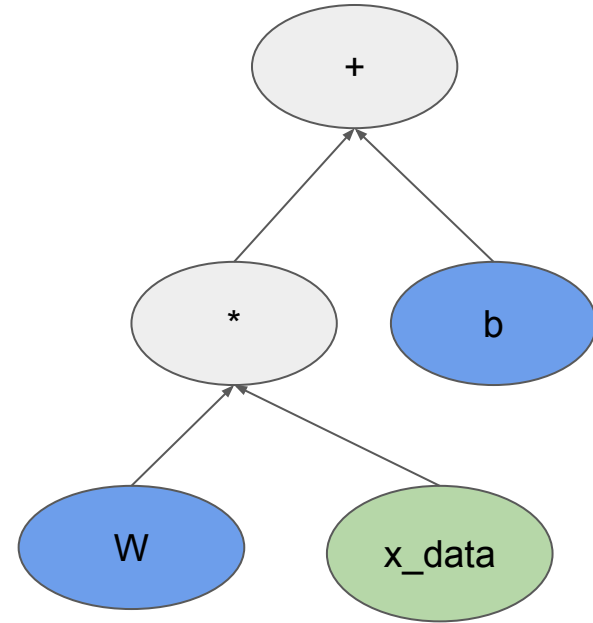
TensorFlow: Computation Graph

```
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy,  $y = x * 0.1 + 0.3$ 
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute  $y_{data} = W * x_{data} + b$ 
#
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```



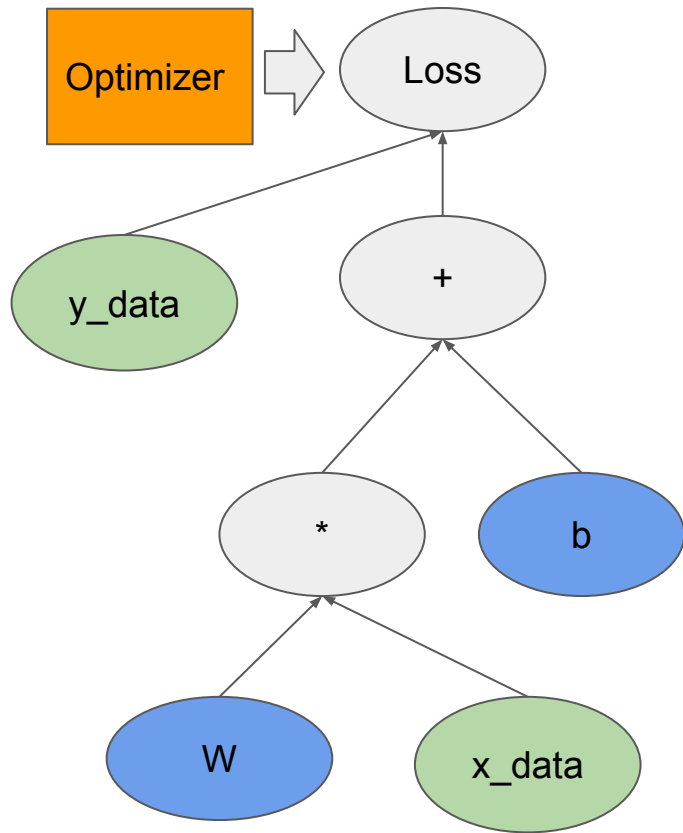
TensorFlow: Computation Graph

```
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy,  $y = x * 0.1 + 0.3$ 
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute  $y\_data = W * x\_data + b$ 
b
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```



TensorFlow: Session

```
# Before starting, initialize the variables. We will 'run' this first.
```

```
init = tf.initialize_all_variables()
```

```
# Launch the graph.
```

```
sess = tf.Session()
```

```
sess.run(init)
```

```
# Fit the line.
```

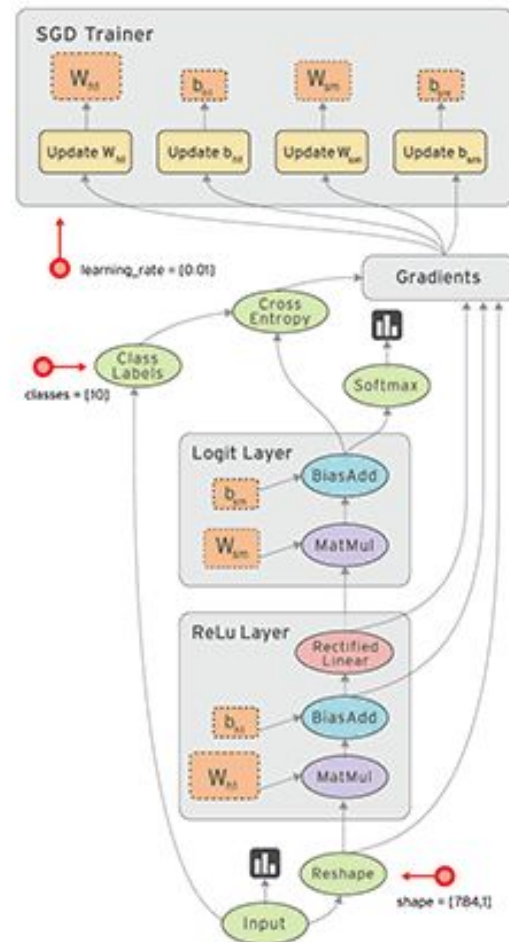
```
for step in xrange(201):
```

```
    sess.run(train)
```

```
    if step % 20 == 0:
```

```
        print(step, sess.run(W), sess.run(b))
```

```
# Learns best fit is W: [0.1], b: [0.3]
```



TensorFlow: Session

Before starting, initialize the variables. We will 'run' this first.

```
init = tf.initialize_all_variables()
```

```
# Launch the graph.
```

```
sess = tf.Session()
```

```
sess.run(init)
```

```
# Fit the line.
```

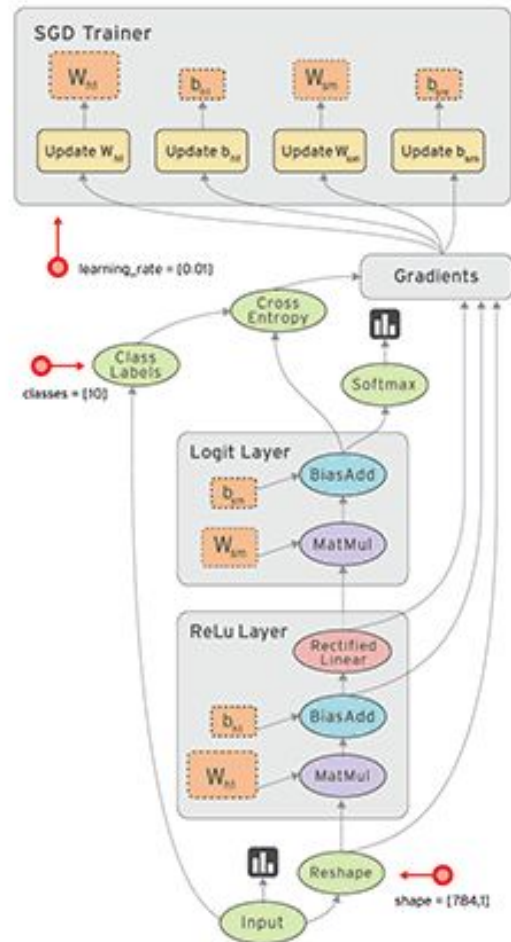
```
for step in xrange(201):
```

```
    sess.run(train)
```

```
    if step % 20 == 0:
```

```
        print(step, sess.run(W), sess.run(b))
```

```
# Learns best fit is W: [0.1], b: [0.3]
```



TensorBoard Demo

TensorBoard

EVENTS

IMAGES

GRAPH

HISTOGRAMS



Fit to screen



Download PNG

Run

train

(1)

Session

runs (0)

Upload

Choose File

Color ☒ Structure

☐ Device

color: same substructure
gray: unique substructure

Graph (* = expandable)



Namespace*



OpNode



Unconnected series*



Connected series*



Constant



Summary



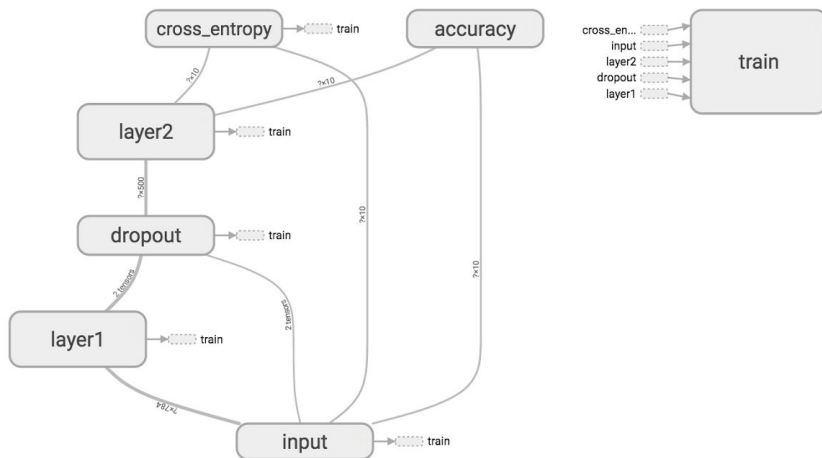
Dataflow edge



Control dependency edge



Reference edge



TensorBoard Demo

TensorBoard

EVENTS

IMAGES

GRAPH

HISTOGRAMS



☒ Regex filter



☐ Split on underscores

☐ Data download links

Horizontal Axis

STEP

RELATIVE

WALL

Runs

☒ test

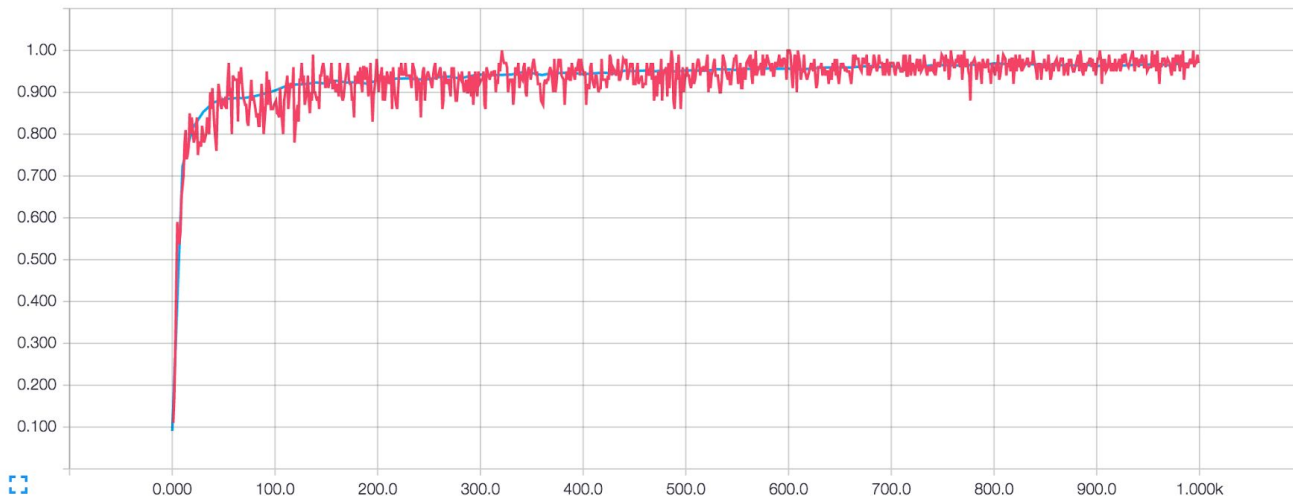
☒ train

TOGGLE ALL RUNS

accuracy

1

accuracy



cross entropy

1

dropout_keep_probability

1

max

4

mean

4

TensorBoard Demo

TensorBoard

EVENTS

IMAGES

GRAPH

HISTOGRAMS



☒ Regex filter



☐ Split on underscores

Horizontal Axis

STEP

RELATIVE

WALL

Runs

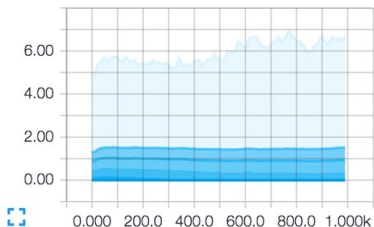
☒ test

☒ train

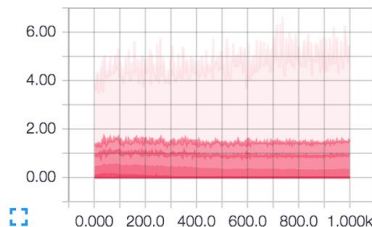
TOGGLE ALL RUNS

layer1

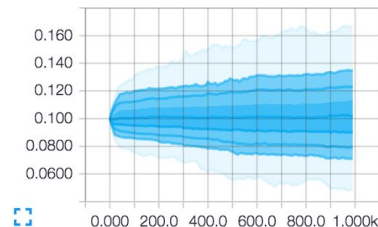
layer1/activations



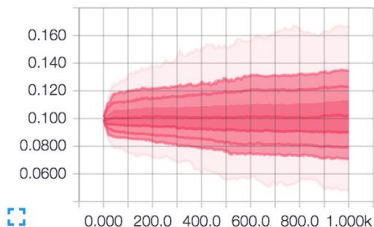
layer1/activations



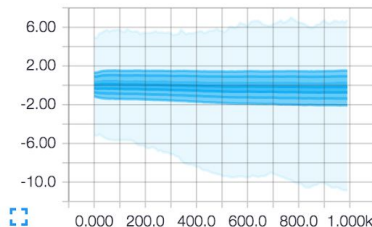
layer1/biases



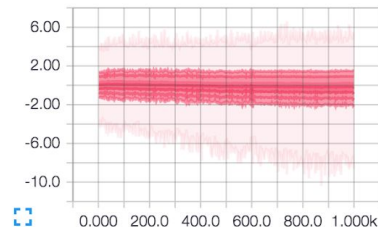
layer1/biases



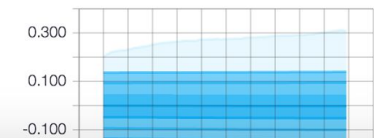
layer1/pre_activations



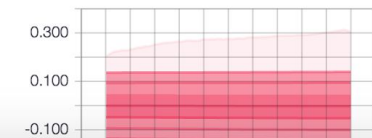
layer1/pre_activations



layer1/weights



layer1/weights



Now the part that everybody hates...



Jon Snow is dead.

Homework

Part 1: Backpropagation and gradient check

- NumPy

Part 2: Char-RNN

- Undergrad/Grad Descent
 - Gradient descent => graduate descent
 - Systematic search of hyperparameters
- Do something fun with it!

Use gradient to find the best parameters

Use graduate student to find the best hyperparameters

References

Christopher Colah's Blog: <http://colah.github.io/>

Andrej Karpathy's Blog: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

David Silver's Talk: http://videolectures.net/rldm2015_silver_reinforcement_learning/

Geoffrey Hinton's Coursera Talk:

<https://class.coursera.org/neuralnets-2012-001/lecture>