
Lecture Notes for Machine Learning in Python

Professor Eric Larson
Week 12, 13, 14 Deep Learning

Class logistics

- Questions on Dask/Graphlab?
- Last time, age calculation on the fly
 - see my solution on GitHub

Lecture Agenda

- Sampling and Modeling
- History
- Convolution Neural Networks
- Remaining Lectures
 - TensorFlow
 - Long- Short-term Memory Networks

THE NOVEMBER 2016
GUIDE TO MAKING PEOPLE
FEEL OLD

IF THEY'RE [AGE], YOU SAY:

"DID YOU KNOW [THING] HAS BEEN
AROUND FOR A MAJORITY OF YOUR LIFE?"

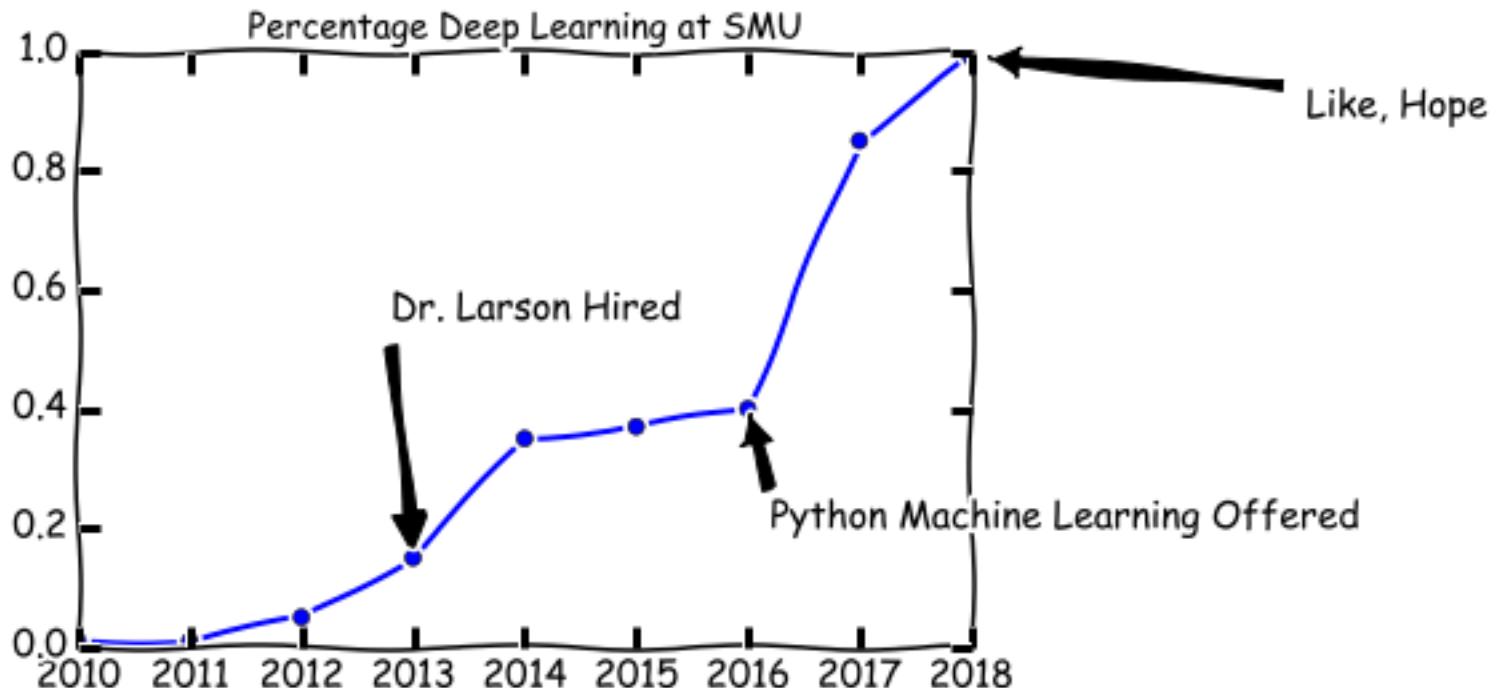
AGE	THING
16	GRAND THEFT AUTO IV
17	RICKROLLING
18	AQUARIUM HUNGER FORCE COLON MOVIE FILM FOR THEATRES
19	THE NINTENDO WII
20	TWITTER
21	THE XBOX 360, XKCD
22	CHUCK NORRIS FACTS
23	OPPORTUNITY'S MARS EXPLORATION
24	FACEBOOK
25	GMAIL, PIRATES OF THE CARIBBEAN
26	IN DA CLUB
27	FIREFOX
28	THE WAR IN AFGHANISTAN
29	THE IPOD
30	SHREK, WIKIPEDIA
31	THOSE X-MEN MOVIES
32	THE SIMS
33	AUTOTUNED HIT SONGS
34	THE STAR WARS PREQUELS
35	THE MATRIX
36	POKÉMON RED & BLUE
37	NETFLIX, HARRY POTTER, GOOGLE
38	DEEP BLUE'S VICTORY
39	TUPAC'S DEATH
40	THE LAST CALVIN AND HOBBES STRIP
41	JOY STICKY
>41	[DON'T WORRY, THEY'VE GOT THIS COVERED]

Sampling and Modeling

Sampling and Modeling

- As of November 8th (morning) the highest rank for Trump winning the presidency was 33% (Nate Silver)
- Most other models were at <1% and were critical of *fivethirtyeight* (Nate Silver)
- **SELF TEST:** Why was this polling/modeling flawed?
 - (A) Insufficient sampling in key geographic areas
 - Under-sampling voters in rural areas (more difficult to contact)
 - Under-sampling minority voters (assume mirroring to Obama)
 - (B) Reliance on Bayesian estimation models
 - (C) Impossible cross validation due to changing environment
 - (D) Insufficient previous examples, even if environment was not changing
- Was there a way to make a better model of the election outcome?
- Some ensemble models performed well, but used features that only included:
 - time incumbent was in office, approval ratings for incumbent, and GDP growth, (other economic factors decreased performance of models)
 - even these models had wide confidence intervals, no clear winner

Some History of Deep Learning



```
# Data to plot
dates = range(2010,2019)
percents = [0.01, 0.01, 0.05, 0.15, 0.35, 0.37, 0.40, 0.85, 0.99]

# Set the style to XKCD
plt.xkcd()

# Plot the percents
plt.plot(dates,percents, marker='o')
```

Neural Networks: Where we left it

- Before 1986: AI Winter
- 1986: *Rumelhart, Hinton, and Williams* popularize gradient calculation for multi-layer network
 - *technically* introduced by Werbos in 1982
- **difference:** Rumelhart *et al.* validated ideas with a computer
- until this point no one could train a multiple layer network consistently
- algorithm is popularly called **Back-Propagation**
- wins pattern recognition prize in 1993, becomes de-facto machine learning algorithm in the 90's

David Rumelhart



1942-2011

Geoffrey Hinton

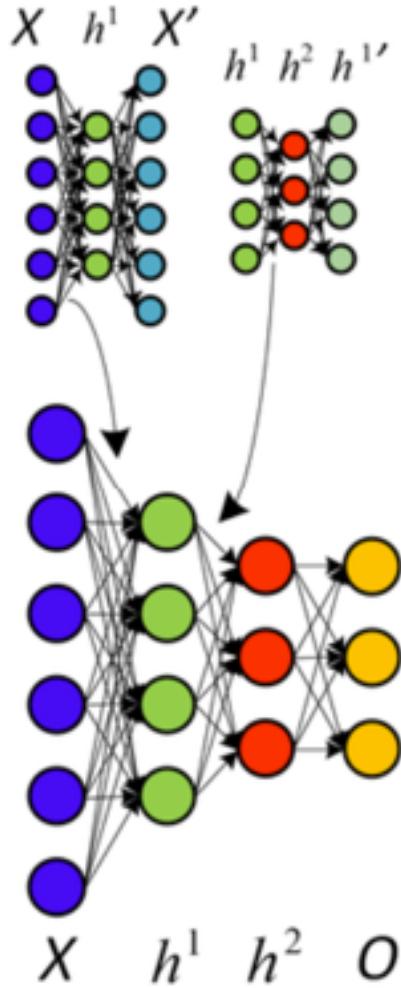


History of Deep Learning: Winter

- Up to this point: back propagation saved AI winter for NN (Hinton and others!)
- 80's, 90's, 2000's: neural networks for image processing start to get deeper
 - but back propagation no longer efficient for training
 - Back propagation gradient stagnates research—can't train deeper networks
- 2001: SVMs and Random Forests gain traction...
 - The second AI winter begins, research in NN plummets
 - Funding for and accepted papers that incorporate Neural Networks asymptotically approaches zero

History of Deep Learning: Winter

- Winter is coming:



Easy to train, performs on par with other methods

Hard to train, performs worse than other methods

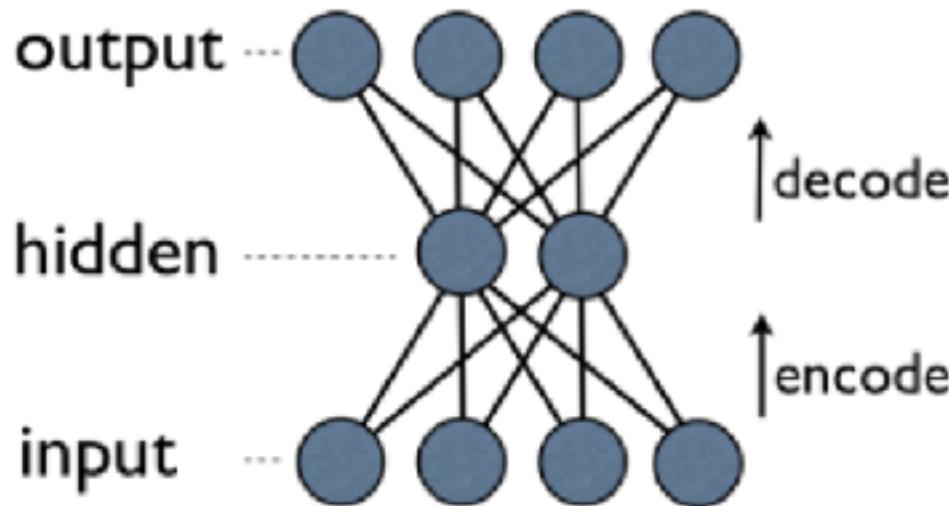
Researcher have difficulty reconciling expressiveness with performance

History of Deep Learning

- 2004: Hinton secures funding from CIFAR based on his reputation
 - *eventually*: Canada would be savior for NN
 - Hinton rebrands: Deep Learning
- 2006: Hinton publishes paper on using pre-training and Restricted Boltzmann Machines
- 2007: Another paper: Deep networks are more efficient when pre-trained
 - RBMs not really the important part

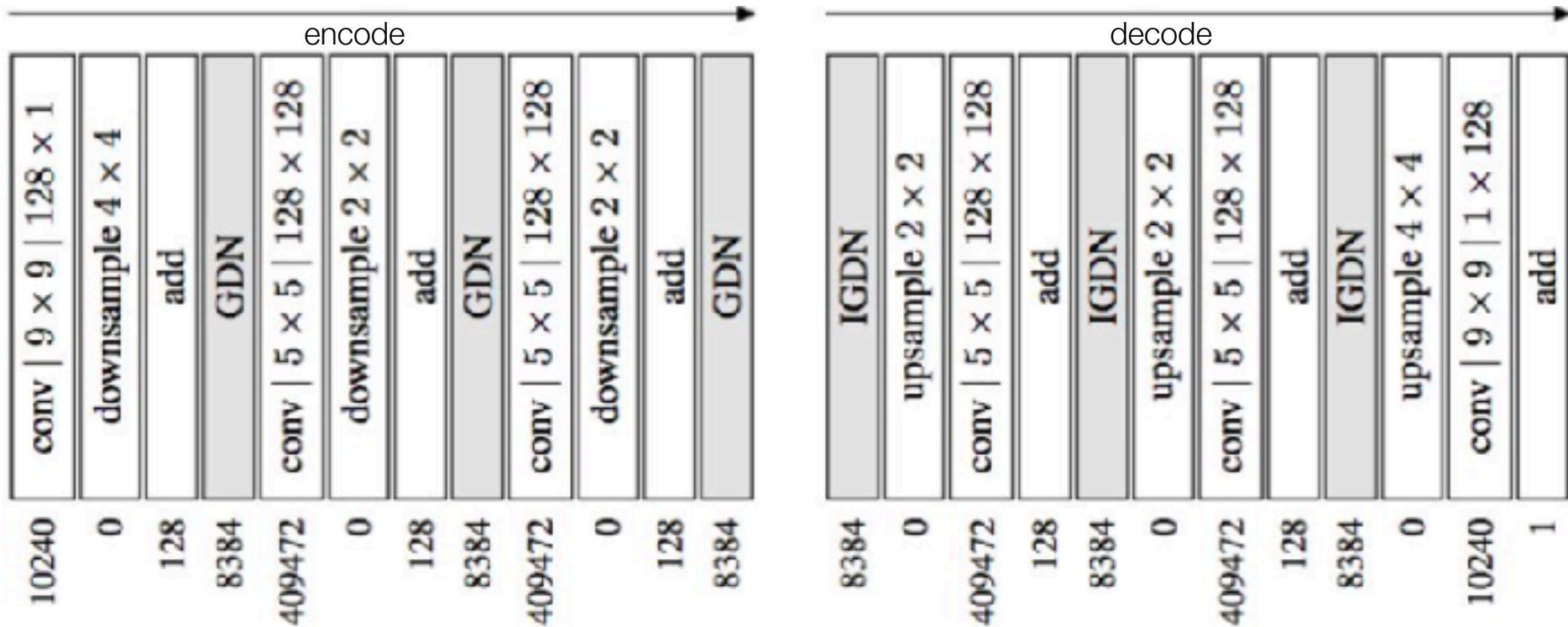
Pre-training: still in the long winter

- auto-encoding



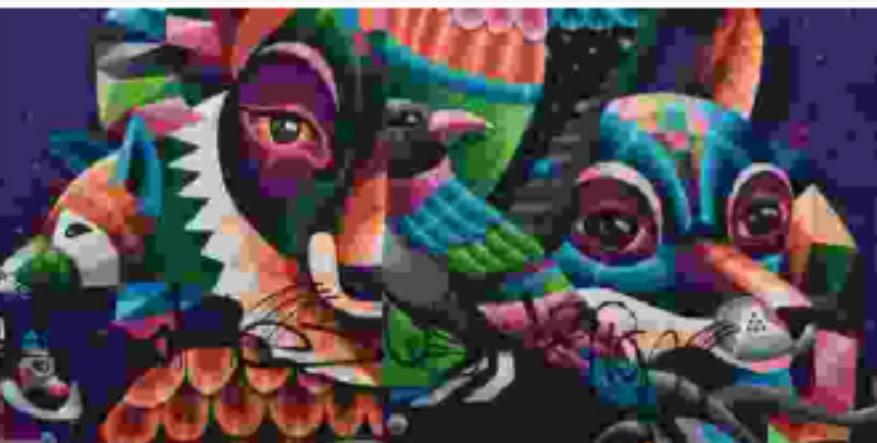
Pre-training: modern example

- auto-encoding: example from paper published Nov 5, 2016





JPEG, 5006 bytes (0.170 bit/px), RMSE: 19.75



JPEG, 5923 bytes (0.168 bit/px), RMSE: 15.44/12.40, PSNR: 24.36 dB/26.26 dB



RMSE: 11.07/10.60, PSNR: 27.25 dB/27.63 dB



Proposed method, 5910 bytes (0.167 bit/px), RMSE



Proposed method, 5685 bytes (0.161 bit/px), RMSE: 10.41/5.98, PSNR: 27.78 dB/32.60 dB



bit/px), RMSE: 6.10/5.09, PSNR: 32.43 dB/34.00 dB



JPEG 2000, 5918 bytes (0.167 bit/px), RMSE: 17



JPEG 2000, 5724 bytes (0.162 bit/px), RMSE: 13.75/7.00, PSNR: 25.36 dB/31.20 dB



bit/px), RMSE: 8.56/5.71, PSNR: 29.49 dB/32.99 dB

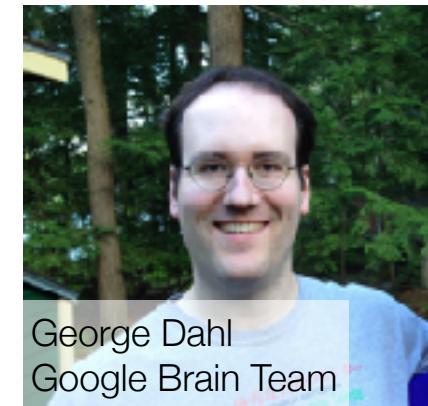
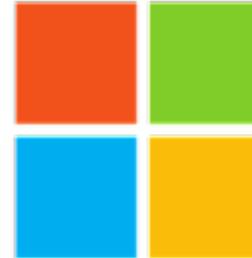
History of Deep Learning

- 2009:
 - Andrew Ng gets involved
 - Hinton's lab start using GPUs
 - GPUs decrease training time by 70 fold...
- 2010: Hinton's and Ng's students go to internships with Microsoft, Google, IBM, and Facebook



Navdeep Jaitly
Google Brain Team

- Xbox Voice
- Android Speech Recognition
- IBM Watson
- DeepFace
- All of Baidu



George Dahl
Google Brain Team

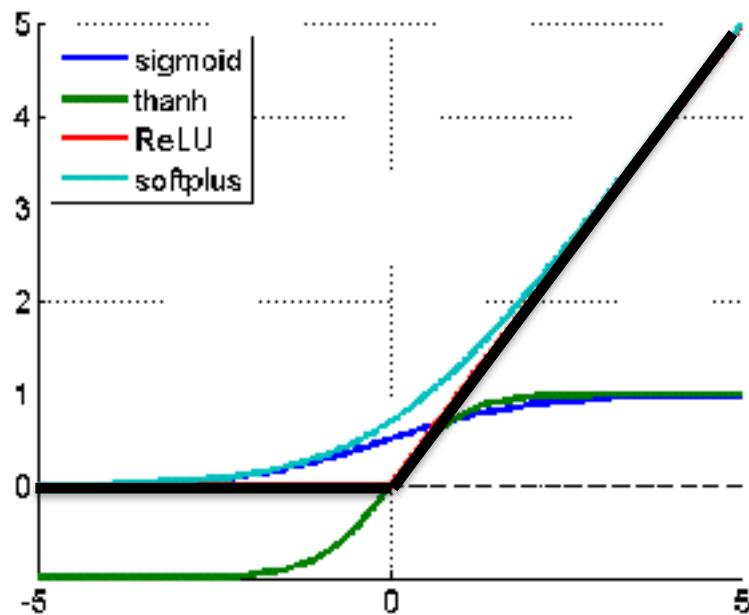


Abdel-rahman Mohamed
Microsoft Research
Redmond, Washington | Computer Software
Current: Microsoft
Previous: University of Toronto, IBM, Microsoft
Education: University of Toronto

Read this: <http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/>

History of Deep Learning

- 2011: Glort and Y. Bengio investigate more systematic methods for why past deep architectures did not work
 - discover some interesting fixes: the type of neurons chosen and the selection of initial weights
 - do not require pre-training to get deep networks properly trained, just sparser representations and less complicated derivatives



ReLU: $f(x) = \max(0, x)$
 $f'(x) = 1 \text{ if } x > 0 \text{ else } 0$

that's a really easy gradient to compute!
...and it makes the weights more sparse
...and helps to solve the vanishing gradient problem
...and its inspired by biological vision community

History of Deep Learning

- 2012: ImageNet competition occurs
 - **Second place:** 26.2% error rate
 - **First place:**
 - From Hinton's lab, uses convolutional neural networks with ReLU neurons and dropout
 - 15.2% error rate
 - Computer vision adopts deep learning with convolutional neural networks en masse



Fei Fei Li
Former
Director of Stanford's
AI Lab

I happened to witness this critical juncture in time first hand because the ImageNet challenge was over the last few years organized by [Fei-Fei Li](#)'s lab (my lab), so I remember when my labmate gasped in disbelief as she noticed the (very strong) ConvNet submission come up in the submission logs. And I remember us pacing around the room trying to digest what had just happened. In the next few months ConvNets went from obscure models that were shrouded in skepticism to rockstars of Computer Vision, present as a core building block in almost every new Computer Vision paper.

History of Deep Learning

- 2012: Hinton Lab, Google, IBM, and Microsoft jointly publish paper, popularity for deep learning methods increases

Deep Neural Networks for Acoustic Modeling in Speech Recognition

[The shared views of four research groups]

[Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly,
Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury]

[https://www.cs.toronto.edu/~gdahl/papers/
deepSpeechReviewSPM2012.pdf](https://www.cs.toronto.edu/~gdahl/papers/deepSpeechReviewSPM2012.pdf)

History of Deep Learning

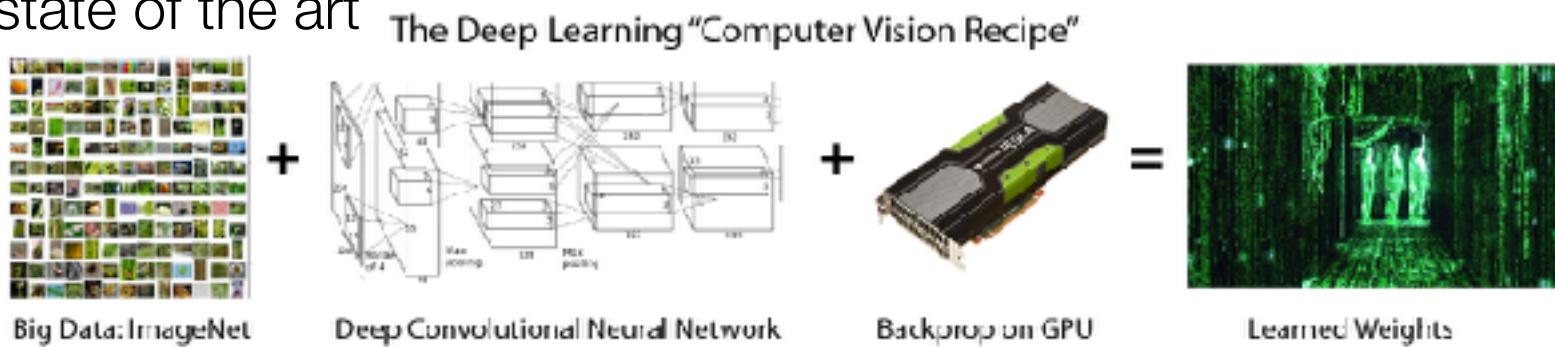
- 2013: Ng and Google (founded BrainTeam)
 - run unsupervised feature creation on YouTube videos (becomes computer vision benchmark)

The work resulted in unsupervised neural net learning of an unprecedented scale - 16,000 CPU cores powering the learning of a whopping 1 billion weights. The neural net was trained on YouTube videos, entirely without labels, and learned to recognize the most common objects in those videos.



History of Deep Learning

- Hinton summarized what we learned in deep learning from the 2006 to present. Where we went wrong before present day:
 - labeled dataset were 1000s of times too small
 - computers were millions of times too slow
 - weights were initialized in stupid ways
 - we used the wrong non-linearities
- Or in my terms:
 - use a GPU, dropout, and ReLU neurons where it makes sense (like in early layers)
- Modern day deep learning uses simple, tried methods to achieve state of the art



Read this: <http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/>

A summary of the Deep Learning people:



Yoshua
Bengio

Stayed at
University

Advises IBM



Yann
LeCun

Heads
Facebook
AI Team



Geoffrey
Hinton

Google



Andrew
Ng

Coursera
Baidu
Google

Read this paper from 2015,
as it sums up advancements
nicely

And predicts the future of
deep learning

REVIEW

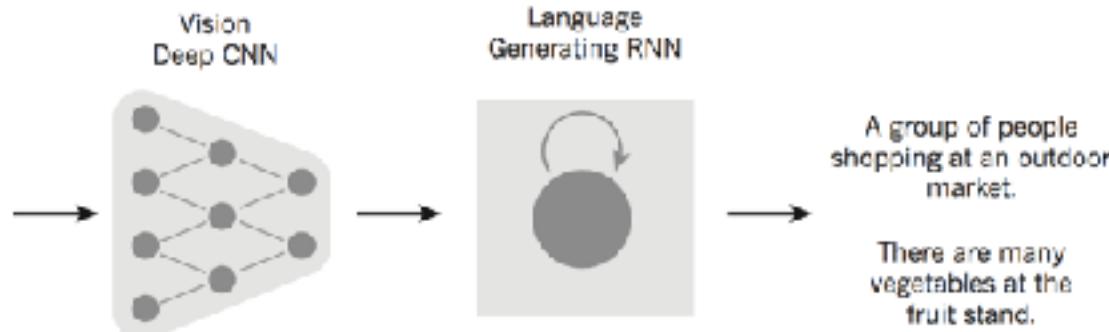
doi:10.1038/nature14539

Deep learning

Yann LeCun^{1,2}, Yoshua Bengio³ & Geoffrey Hinton^{4,5}

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genetics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

Famous examples:



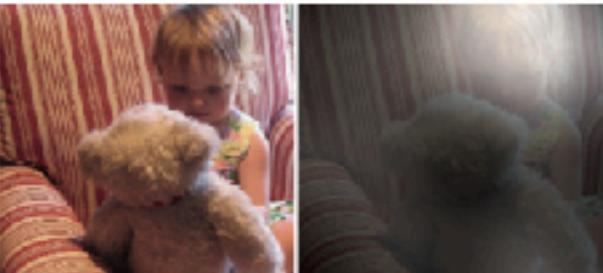
A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



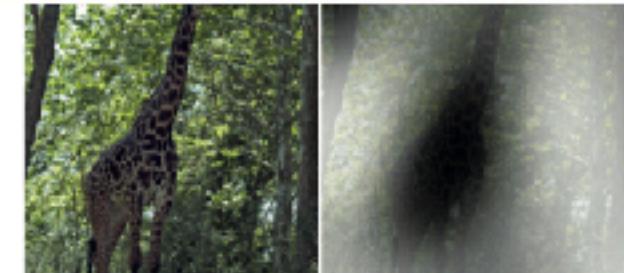
A stop sign is on a road with a mountain in the background



A little girl sitting on a bed with a teddy bear.



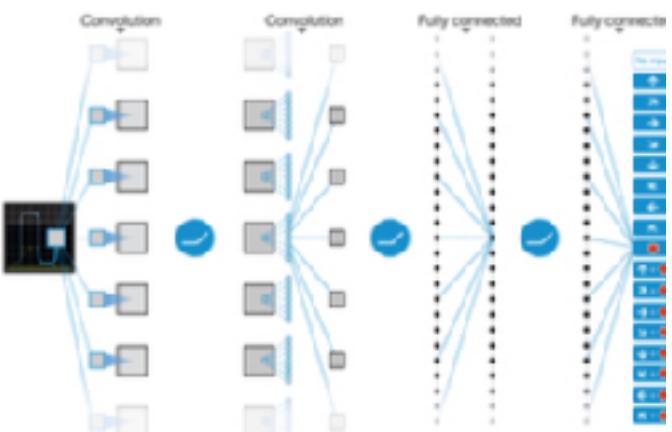
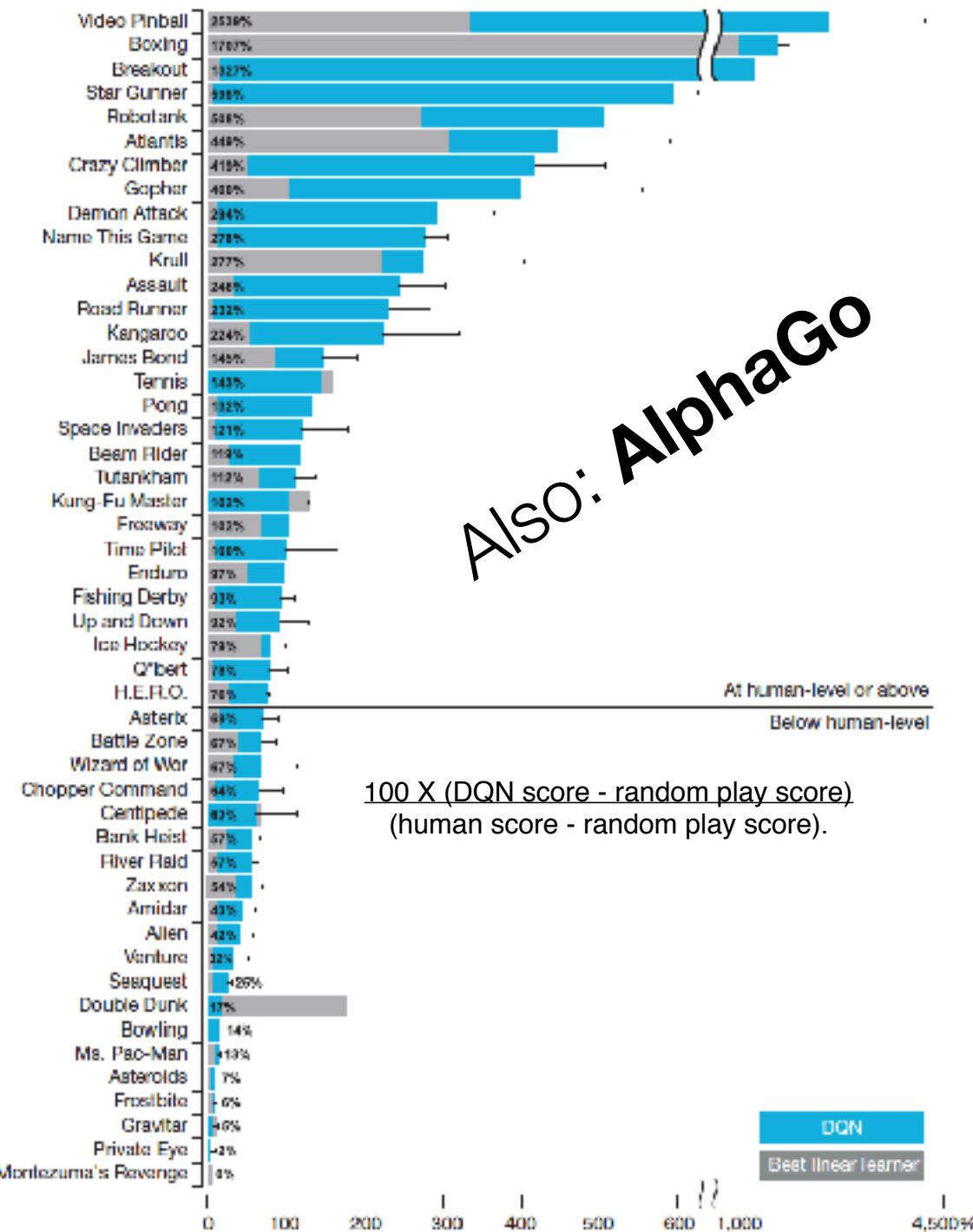
A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

Famous examples

Also: AlphaGo



Convolutional Neural Networks



IN CS, IT CAN BE HARD TO EXPLAIN
THE DIFFERENCE BETWEEN THE EASY
AND THE VIRTUALLY IMPOSSIBLE.

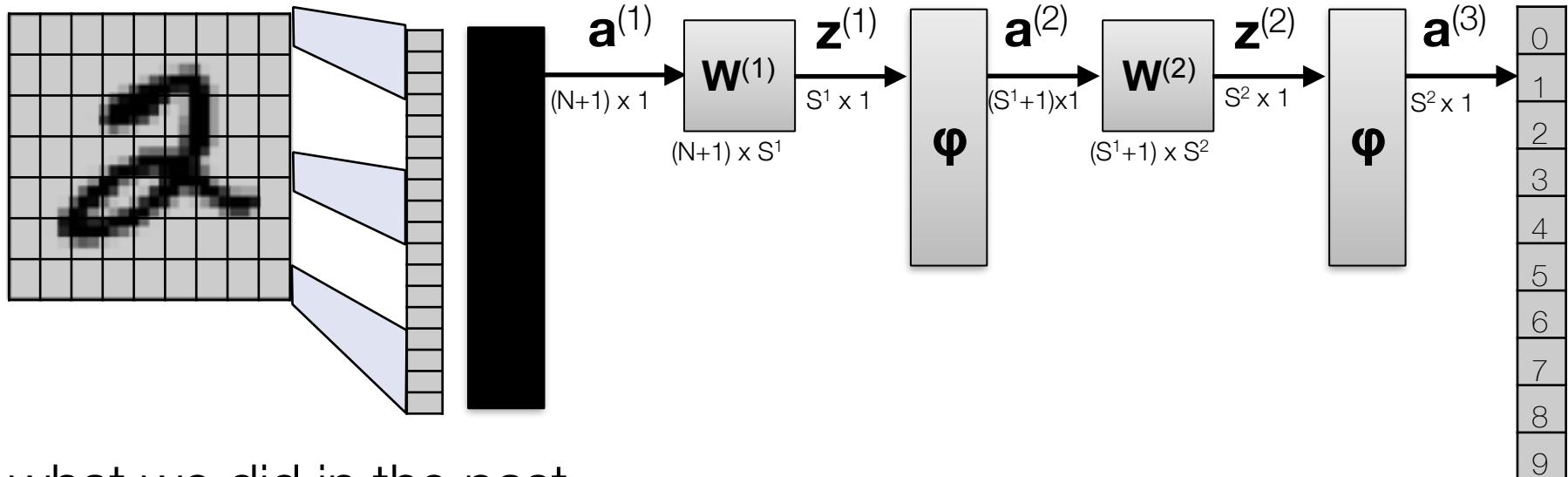
CNN Overview

- First layer(s):
 - convolution with different filters
 - nonlinearity
 - pooling
 - Each pooling layer can make the input image “smaller”
 - allowing more summative explanations
- Final layers are densely connected
 - typically multi-layer perceptrons

CNN Overview: Self Test

- First layer(s):
 - convolution with different filters
 - nonlinearity
 - pooling
- Final layers are densely connected
 - typically multi-layer perceptrons
- Where are unstable gradients **most** problematic?
 - (A) During Convolution Layer(s) updates
 - (B) During Fully Connected Layer(s) updates
 - (C) Both A and B
 - (D) They are not a problem

From Fully Connected to CNN



what we did in the past

If image is 9×9 , and each fully connected layer is 20 hidden neurons wide, how many parameters are in this NN (ignore bias)?

$$(K^2 \times 20) + (20 \times 20) + (20 \times 10) = 600 + 20 K^2$$

$$\text{for } 9 \times 9 = 600 + 20 \times 9^2 = 2,220 \text{ parameters}$$

Lecture Notes for Machine Learning in Python

Professor Eric Larson
Week 12, 13, 14 Deep Learning

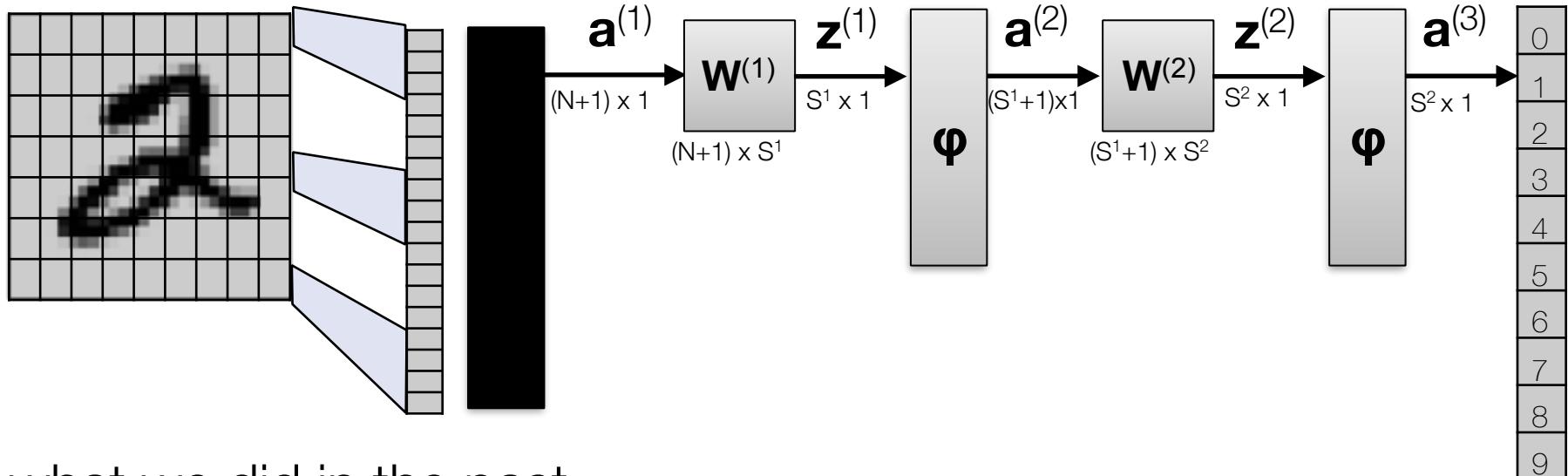
Class logistics

- Lab 3 has been up!
 - Due Dec 13 by 5PM
 - Sorry, No Resubmits for A3
 - No Project Presentations
- Grades will be coming in more steadily

Lecture Agenda

- Finish Convolutional Neural Networks
- Introduction to TensorFlow
 - Tensors
 - Namespaces
 - Numerical methods
- Demo of TensorFlow with CNNs
- Remaining lecture topics:
 - Long- Short-term Memory Networks

From Fully Connected to CNN



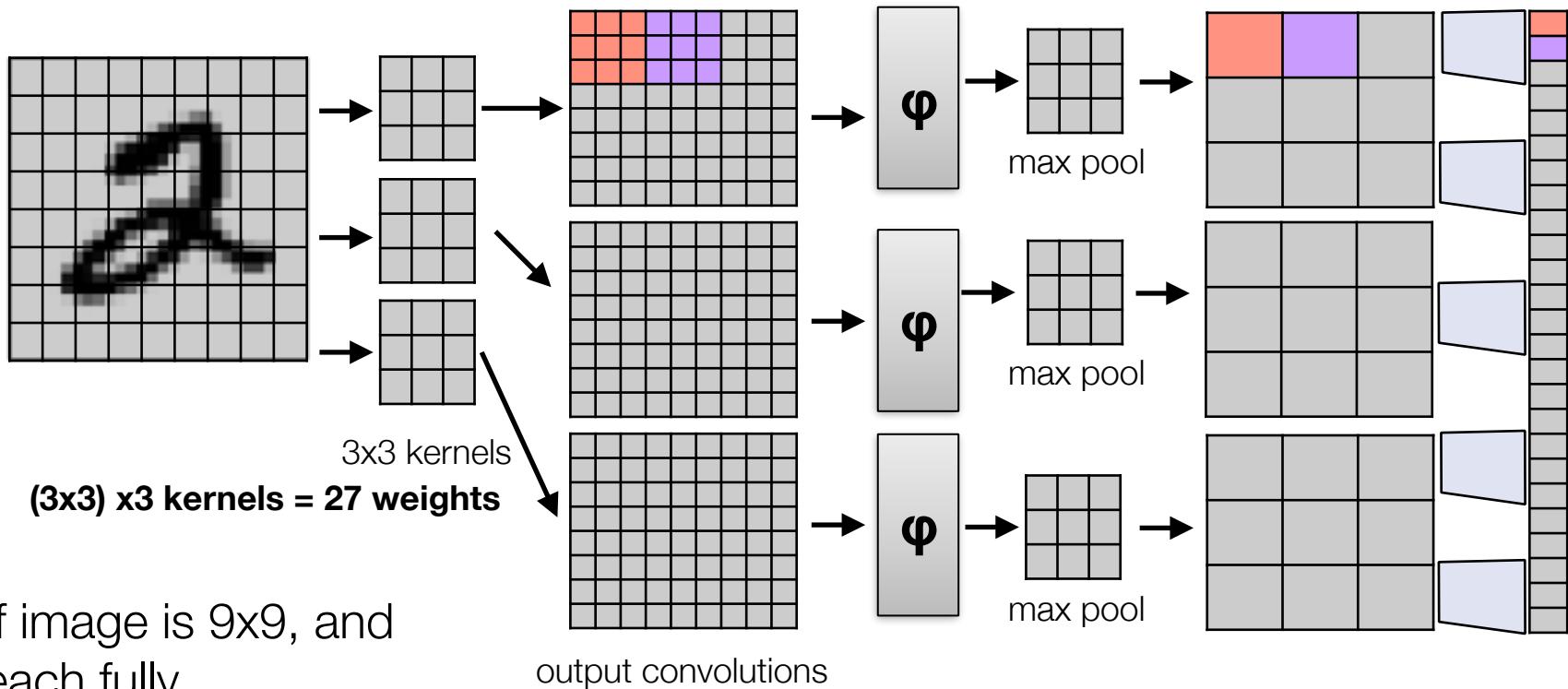
what we did in the past

If image is 9x9, and each fully connected layer is 20 hidden neurons wide, how many parameters are in this NN (ignore bias)?

$$(K^2 \times 20) + (20 \times 20) + (20 \times 10) = 600 + 20 K^2$$

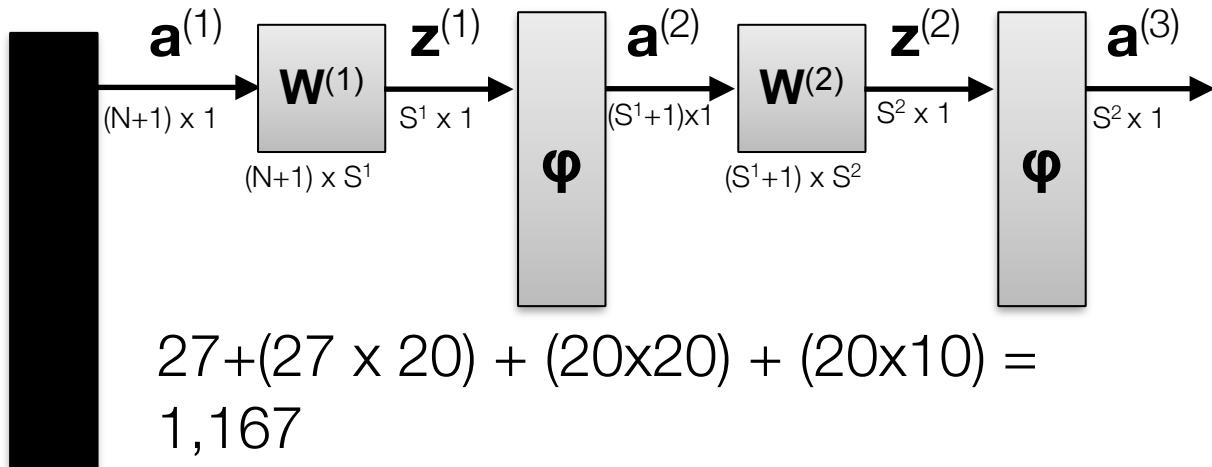
$$\text{for } 9 \times 9 = 600 + 20 \times 9^2 = 2,220 \text{ parameters}$$

From Fully Connected to CNN

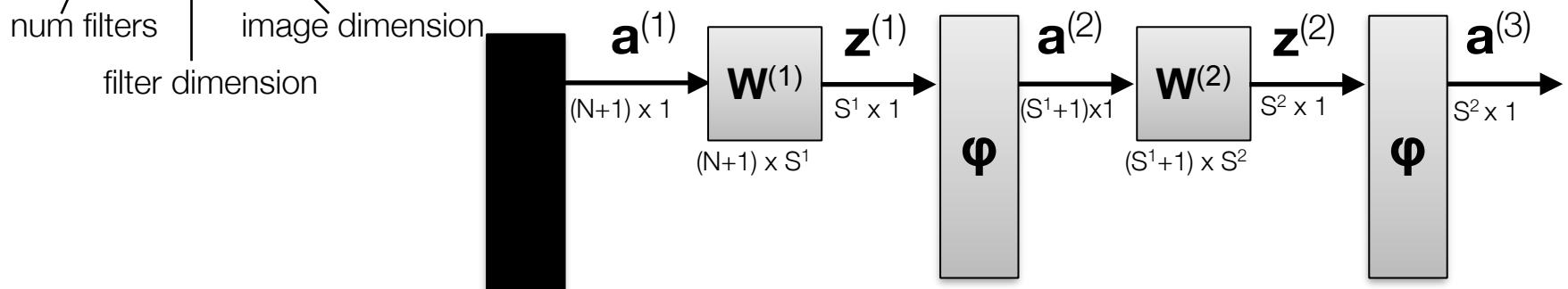
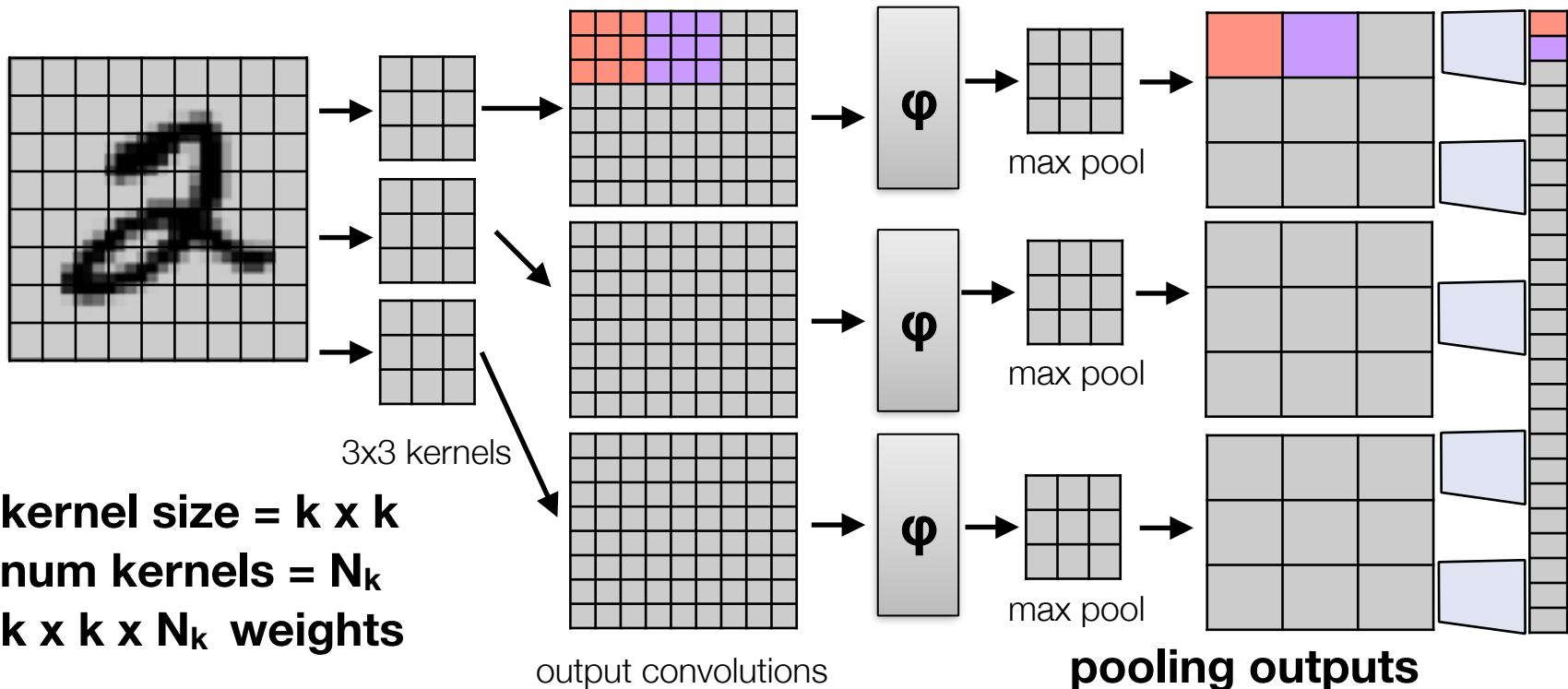


If image is 9x9, and each fully connected layer is 20 hidden neurons wide, how many parameters are in this NN (ignore bias)?

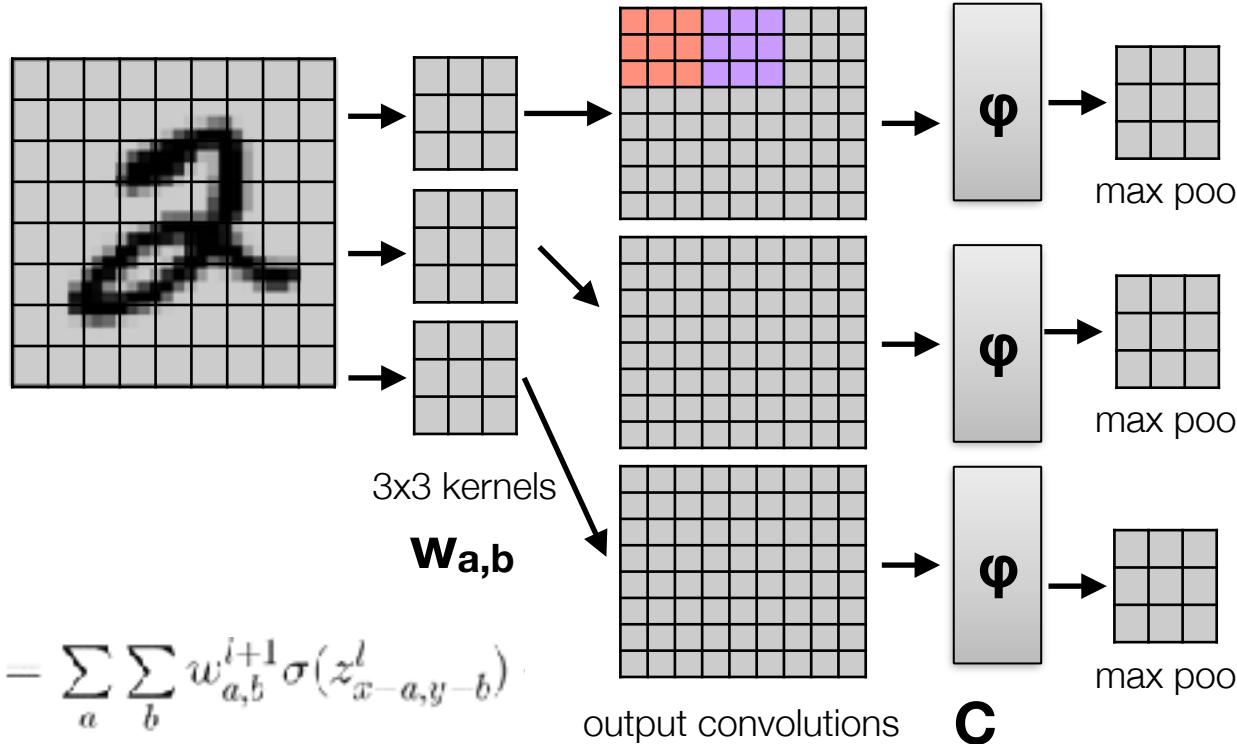
$$3 \times 3 \times 3 = 27 \text{ weights}$$



From Fully Connected to CNN



CNN gradient



Derivative of convolution
is more involved:

$$\frac{\partial C}{\partial w_{a,b}^l} = \sum_x \sum_y \frac{\partial C}{\partial z_{x,y}^l} \frac{\partial z_{x,y}^l}{\partial w_{a,b}^l} = \sum_x \sum_y \delta_{x,y}^l \frac{\partial (\sum_{a'} \sum_{b'} w_{a',b'}^l \sigma(z_{x-a',y-b'}^l) + b_{x,y}^l)}{\partial w_{a,b}^l} =$$
$$\sum_x \sum_y \delta_{x,y}^l \sigma(z_{x-a,y-b}^{l-1}) = \delta_{a,b}^l * \sigma(z_{-a,-b}^{l-1}) = \delta_{a,b}^l * \sigma(ROT180(z_{a,b}^{l-1}))$$

Derivative of max pool is
easy:

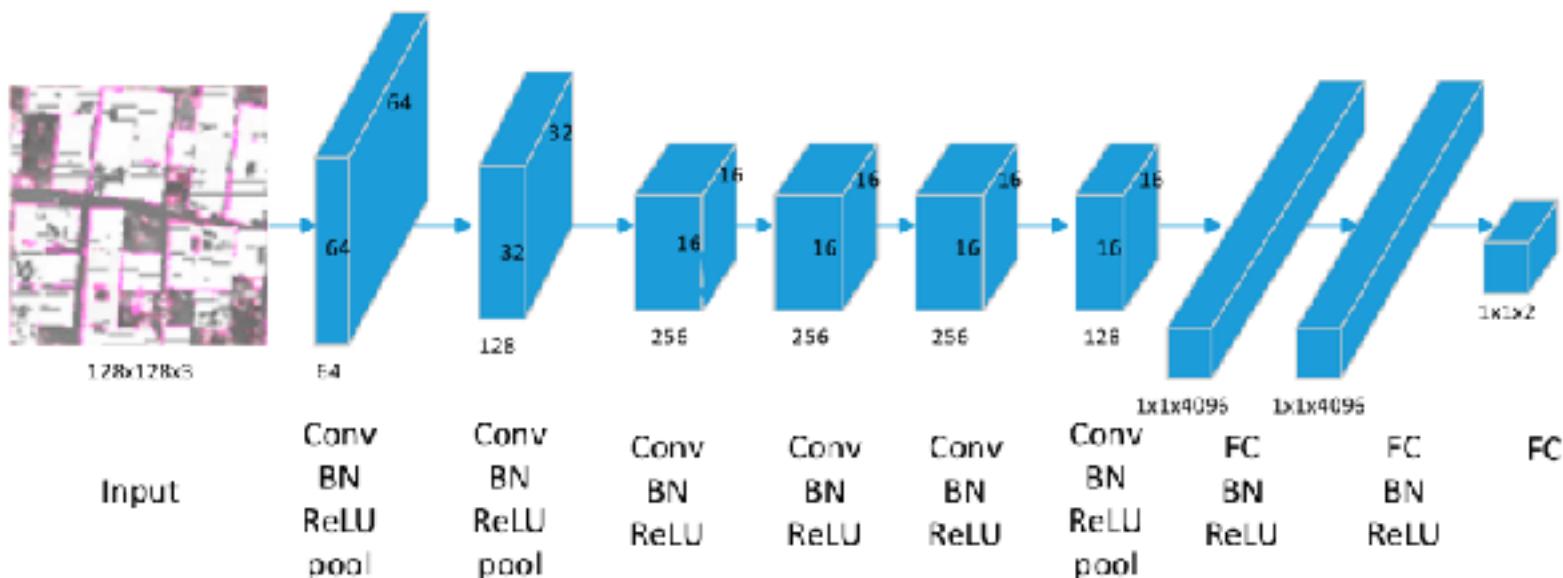
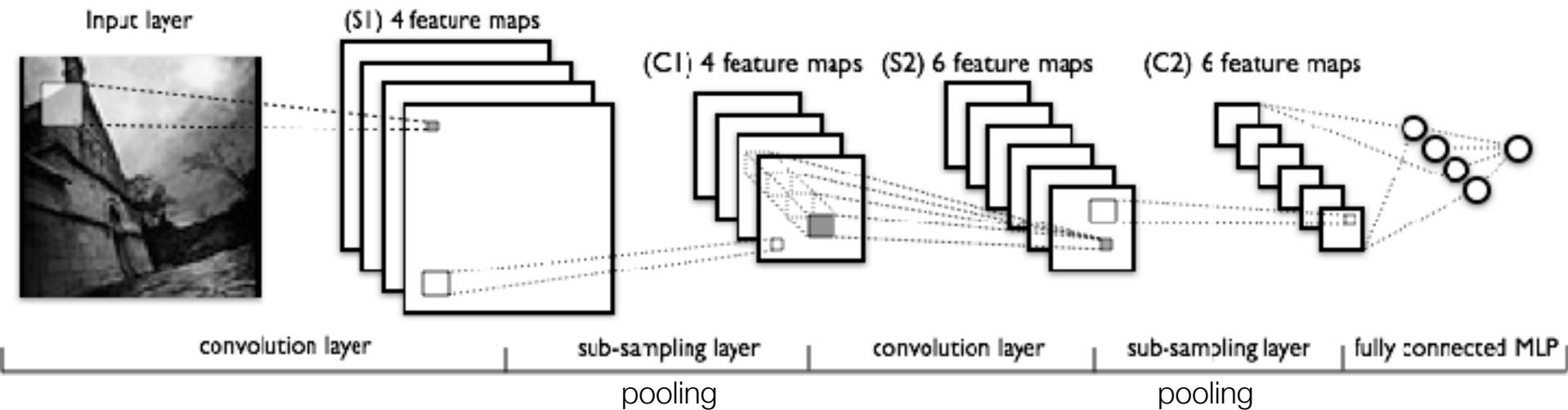
for each input x_i

$$f'(x_i) = \begin{cases} 1 & \text{if } x_i \text{ is max} \\ 0 & \text{else} \end{cases}$$

CNN gradient

- But we really want to understand the process!
- These are great guides:
 - [https://grzegorzgwardys.wordpress.com/
2016/04/22/8/](https://grzegorzgwardys.wordpress.com/2016/04/22/8/)
 - [http://andrew.gibiansky.com/blog/machine-
learning/convolutional-neural-networks/](http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/)

Common CNN Architecture



CNN: What does it all mean?

Deep Visualization Toolbox

yosinski.com/deepvis

#deepvis



Jason Yosinski



Jeff Clune



Anh Nguyen



Thomas Fuchs



Hod Lipson



Cornell University

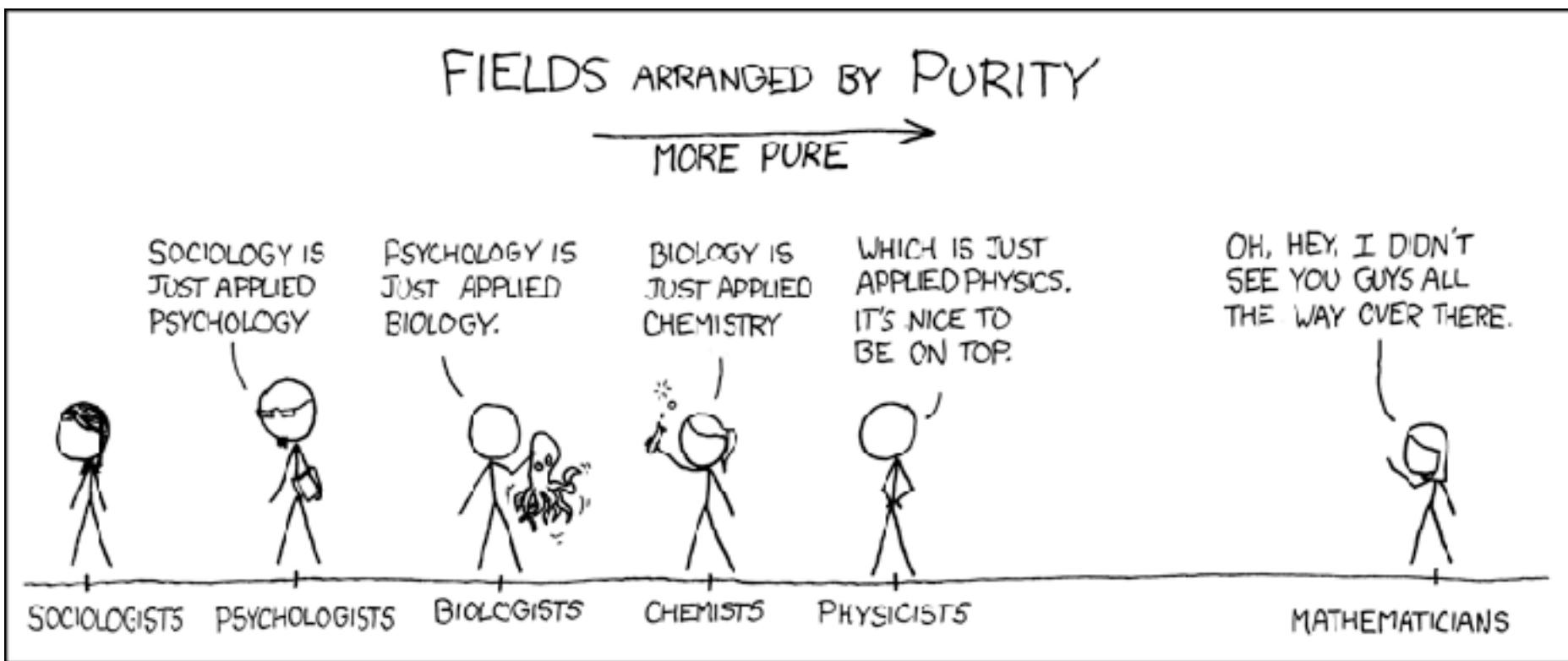


Jet Propulsion Laboratory
California Institute of Technology

Programming CNNs

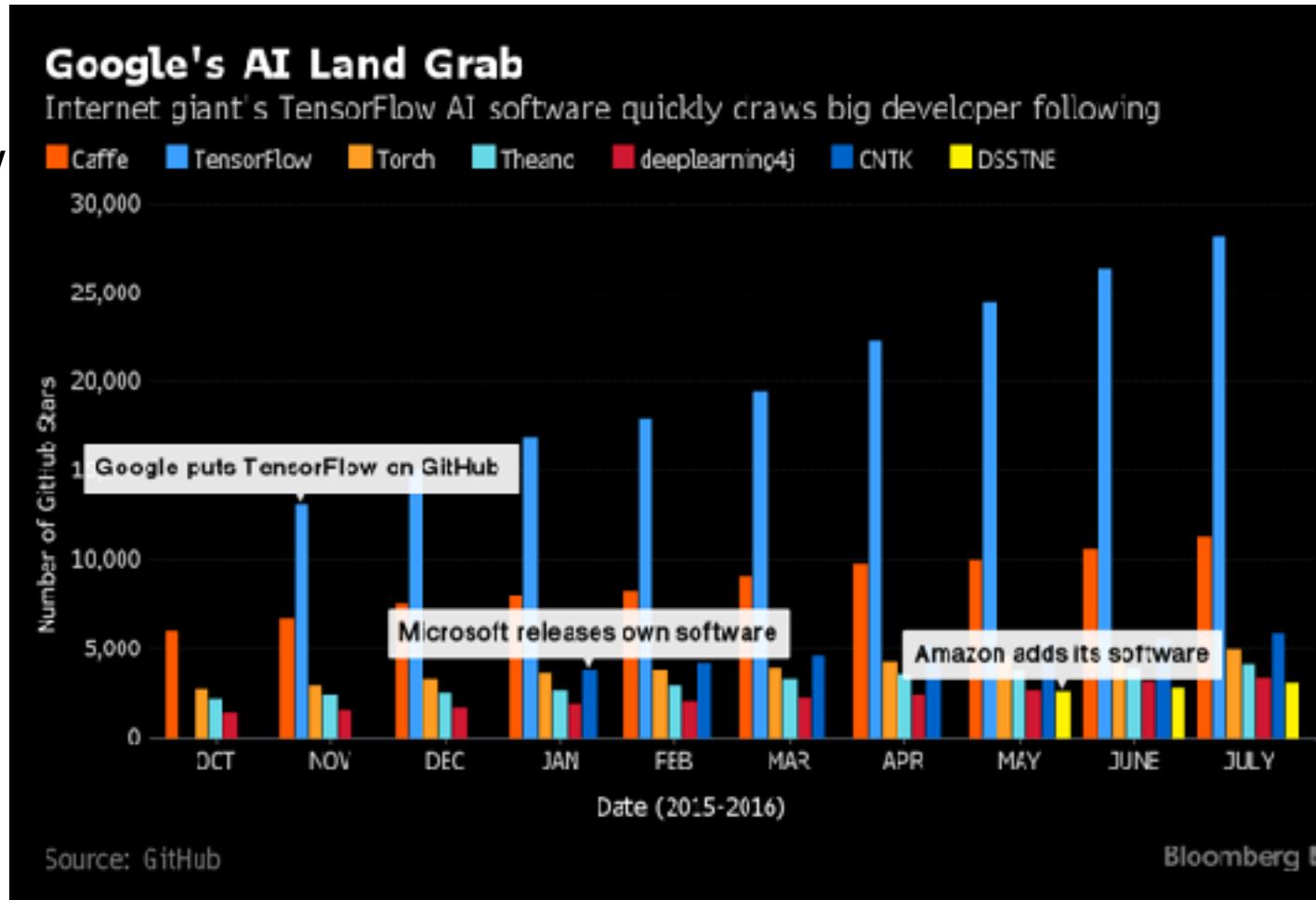
- We need a deep learning toolkit!

TensorFlow



Options for Deep Learning Toolkits

- Caffe
- TensorFlow
- Torch
- CuDNN
- MxNet
- Theano
- CNTK
- DSSTNE



Programmatic creation

- Most toolkits use python to build a computation graph of operations
 - In many ways similar to Dask and Graphlab
 - Build up computations
 - Execute computations
- Theano/CNTK are completely valid alternatives
 - its really up to you what you want to use
 - Theano originated at Berkeley and can be wrapped with Keras or Lasagne
 - <http://deeplearning.net/software/theano/>
 - CNTK is a Microsoft product with python wrappers and has some impressive speed graphics compared to all other languages (as of one year ago)
 - <https://github.com/Microsoft/CNTK>

Tensorflow

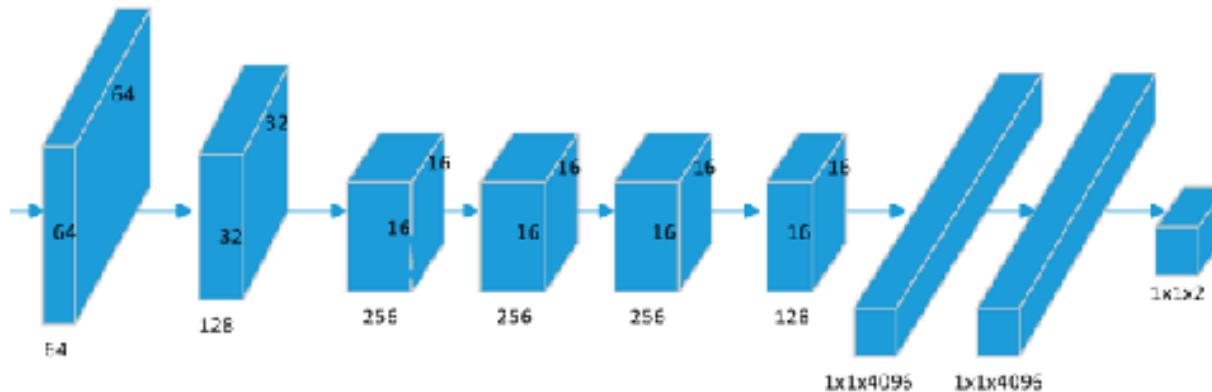
- Open sourced library from Google
- Second generation release from Google Brain
 - supported for Linux and Unix (some windows support if you build from source)
 - Also works on Android/iOS
- Released November 9th, 2015
- **Supports:**
 - tensor creation
 - functions on tensors
 - automatic derivative computation

Tensors

- Tensors are just multidimensional arrays

- like in Numpy
- typically, for NN,
 - scalars (biases and constants)
 - vectors (inputs)
 - 2D matrices (images)
 - 3D matrices (color images)
 - 4D matrices (batches of color images)

```
a = tf.constant(5.0)  
b = tf.constant(6.0)
```



Tensor basic functions

```
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
```

- Easy to define operations on tensors

Numpy	TensorFlow
<code>a = np.zeros((2,2)); b = np.ones((2,2))</code>	<code>a = tf.zeros((2,2)), b = tf.ones((2,2))</code>
<code>np.sum(b, axis=1)</code>	<code>tf.reduce_sum(a, reduction_indices=[1])</code>
<code>a.shape</code>	<code>a.get_shape()</code>
<code>np.reshape(a, (1,4))</code>	<code>tf.reshape(a, (1,4))</code>
<code>b * 5 + 1</code>	<code>b * 5 + 1</code>
<code>np.dot(a,b)</code>	<code>tf.matmul(a, b)</code>
<code>a[0,0], a[:,0], a[0,:]</code>	<code>a[0,0], a[:,0], a[0,:]</code>

- Also supports convolution: `tf.nn.conv2d`, `tf.nn.conv3D`

Tensor neural network functions

- Easy to define operations on layers of networks
- `tf.nn.relu(features, name=None)`
- `tf.nn.bias_add(value, bias, data_format=None, name=None)`
- `tf.sigmoid(x, name=None)`
- `tf.tanh(x, name=None)`
- `tf.nn.conv2d(input, filter, strides, padding)`
- `tf.nn.conv1d(value, filters, stride, padding)`
- `tf.nn.conv3d(input, filter, strides, padding)`
- `tf.nn.conv3d_transpose(value, filter, output_shape, strides)`
- `tf.nn.sigmoid_cross_entropy_with_logits(logits, targets)`
- `tf.nn.softmax(logits, dim=-1)`
- `tf.nn.log_softmax(logits, dim=-1)`
- `tf.nn.softmax_cross_entropy_with_logits(logits, labels, dim=-1)`
- Each function creates layers easily, *knows its gradient*
- But... lets start simple...

Tensor function evaluation

- Easy to define operations on tensors
- Nothing evaluated until you define a session and tell it to evaluate it
- Session defines configuration of execution
 - like GPU versus CPU

```
a = tf.constant(5.0)
```

```
b = tf.constant(6.0)
```

```
c = a * b
```

```
with tf.Session() as sess:  
    print(sess.run(c))  
    print(c.eval())
```

output = 30

Tensor Variables

- Variables that must be initialized before use

```
W1 = tf.ones((2,2)) immutable
```

```
W2 = tf.Variable(tf.zeros((2,2)), name="weights") mutable
```

```
with tf.Session() as sess:  
    print(sess.run(W1))  
    sess.run(tf.initialize_all_variables())  
    print(sess.run(W2))
```

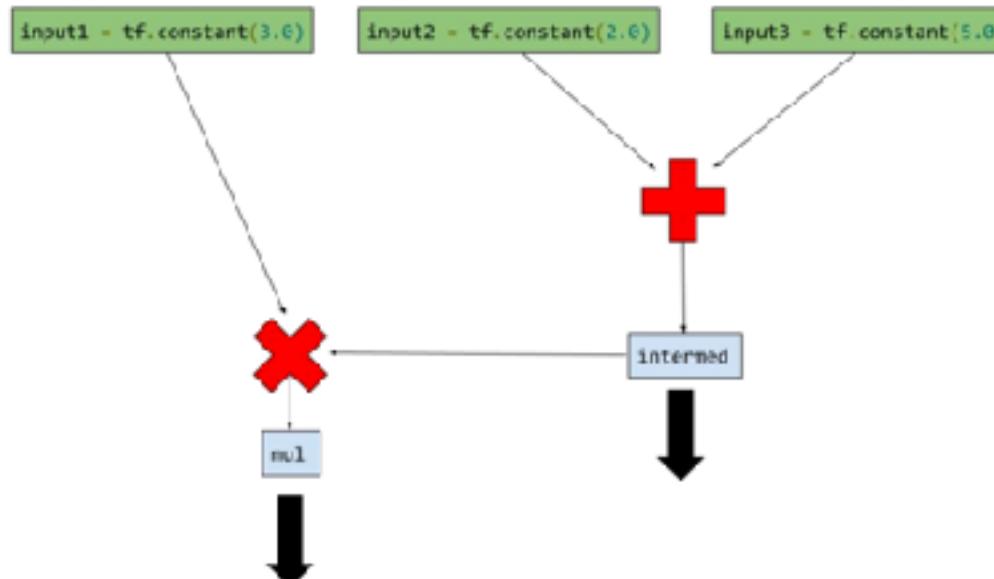
```
W = tf.Variable(tf.zeros((2,2)), name="weights")
```

```
R = tf.Variable(tf.random_normal((2,2)), name="random_weights")
```

Tensor Variables

- Variables that must be initialized before use

```
input1 = tf.constant(3.0)
input2 = tf.constant(2.0)
input3 = tf.constant(5.0)
intermed = tf.add(input2, input3) mutable variable
mul = tf.mul(input1, intermed) mutable variable
with tf.Session() as sess:
    result = sess.run([mul, intermed]) fetch variable values
    print(result)
```



Tensor Placeholders

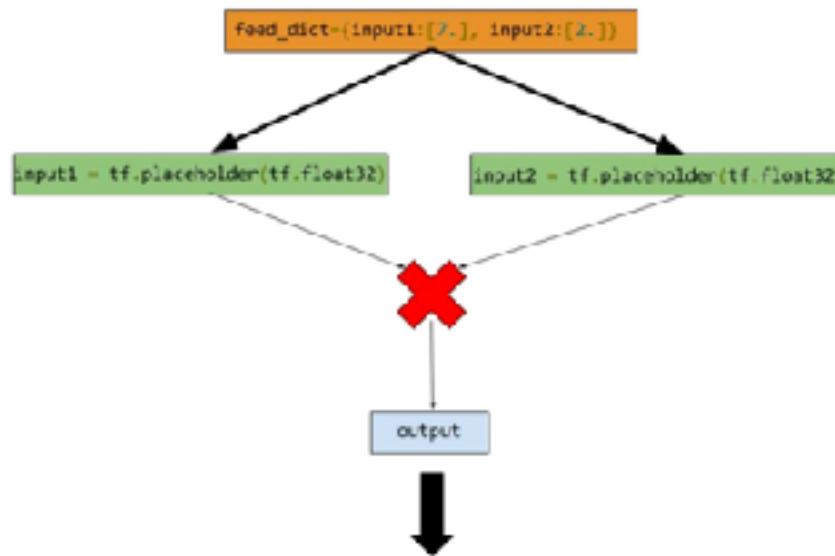
- Placeholders await values and can be streamed in from the computation graph

```
input1 = tf.placeholder(tf.float32)

input2 = tf.placeholder(tf.float32)

output = tf.mul(input1, input2)

with tf.Session() as sess:
    print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))
```



Tensor get_variable and scope

- Just a form of scoping
 - define and reuse variables
 - if you have not defined variable as “reuse,” then you can only access it once via get_variable

Variable Scope Example

Variable Scope mechanism in TensorFlow consists of 2 main functions:

- `tf.get_variable(<name>, <shape>, <initializer>)`: Creates or returns a variable with a given name.
- `tf.variable_scope(<scope_name>)`: Manages namespaces for names passed to `tf.get_variable()`.

```
with tf.variable_scope("foo"):  
    v = tf.get_variable("v", [1])      v.name == "foo/v:0"  
  
with tf.variable_scope("foo", reuse=True):  
    v1 = tf.get_variable("v", [1])      without "reuse" this is an error
```

Tensor get_variable and scope

- Why do we need this?
 - Reusing variable creation code

```
def my_image_filter(input_images):
    conv1_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),
                               name="conv1_weights")
    conv1_biases = tf.Variable(tf.zeros([32]), name="conv1_biases")
    conv1 = tf.nn.conv2d(input_images, conv1_weights,
                        strides=[1, 1, 1, 1], padding='SAME')
    relu1 = tf.nn.relu(conv1 + conv1_biases)

    conv2_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),
                               name="conv2_weights")
    conv2_biases = tf.Variable(tf.zeros([32]), name="conv2_biases")
    conv2 = tf.nn.conv2d(relu1, conv2_weights,
                        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv2 + conv2_biases)
```

**explicit definition of
repeated operations**

Tensor get_variable and scope

- Why do we need this?
 - Reusing variable creation code

```
def conv_relu(input, kernel_shape, bias_shape):
    # Create variable named "weights".
    weights = tf.get_variable("weights", kernel_shape,
        initializer=tf.random_normal_initializer())
    # Create variable named "biases".
    biases = tf.get_variable("biases", bias_shape,
        initializer=tf.constant_initializer(0.0))
    conv = tf.nn.conv2d(input, weights,
        strides=[1, 1, 1, 1], padding='SAME')
    return tf.nn.relu(conv + biases)

def my_image_filter(input_images):
    with tf.variable_scope("conv1"):
        # Variables created here will be named "conv1/weights", "conv1/biases".
        relu1 = conv_relu(input_images, [5, 5, 32, 32], [32])
    with tf.variable_scope("conv2"):
        # Variables created here will be named "conv2/weights", "conv2/biases".
        return conv_relu(relu1, [5, 5, 32, 32], [32])
```

more readable code

Tensor Differentiation

- This seems like a lot of new syntax
- Why can't we just use numpy?
 - automatic differentiation, i.e., “*your life, easier*”

```
# Define variables to be learned
with tf.variable_scope("linear-regression"):
    W = tf.get_variable("weights", (1, 1),
                        initializer=tf.random_normal_initializer())
    b = tf.get_variable("bias", (1, ),
                        initializer=tf.constant_initializer(0.0))
    y_pred = tf.matmul(X, W) + b
    loss = tf.reduce_sum((y - y_pred)**2/n_samples)

opt = tf.train.AdamOptimizer()
opt_operation = opt.minimize(loss)
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    sess.run([opt_operation], feed_dict={X: X_data, y: y_data})
```

Tensor Mini-batching?

```
opt = tf.train.AdamOptimizer()
opt_operation = opt.minimize(loss)

with tf.Session() as sess:
    # Initialize Variables in graph
    sess.run(tf.initialize_all_variables())
    # Gradient descent loop for 500 steps
    for _ in range(500):
        # Select random minibatch
        indices = np.random.choice(n_samples, batch_size)
        X_batch, y_batch = X_data[indices], y_data[indices]
        # Do gradient descent step
        _, loss_val = sess.run([opt_operation, loss], feed_dict={X: X_batch, y: y_batch})
```

- Each node in a TensorFlow graph knows its gradient
- Which means back-propagation is easy to carry out computationally

Tensor-flow Simplification

- It seems like many common NN optimizations can be preprogrammed
- **Self Test:** Can the syntax be simplified?
 - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary inputs
 - (B) **Yes**, but we lose control over the optimization procedures
 - (C) **Yes**, but we lose control over the NN models that we can create via Tensorflow
 - (D) **Yes**, and Dr. Larson is going to make us write it ourselves

Lecture Notes for Machine Learning in Python

Professor Eric Larson
Week 12, 13, 14 Deep Learning

Class logistics

- Lab 3 has been up!
 - Due Dec 13 by 5PM
 - Sorry, No Resubmits for A3
 - No Project Presentations
- Grades will be coming in more steadily

Lecture Agenda

- Finish Convolutional Neural Networks
- Introduction to TensorFlow
 - Tensors
 - Namespaces
 - Numerical methods
- Demo of TensorFlow with CNNs
- Remaining lecture topics:
 - RNNs

Tensor-flow Simplification

- It seems like many common NN optimizations can be preprogrammed
- **Self Test:** Can the syntax be simplified?
 - (A) **Yes**, we could write a generic mini-batch optimization computation graph, then use it for arbitrary inputs
 - (B) **Yes**, but we lose control over the optimization procedures
 - (C) **Yes**, but we lose control over the NN models that we can create via Tensorflow
 - (D) **Yes**, and Dr. Larson is going to make us write it ourselves

Tensor-flow Simplification: Scikit-Flow (SKFlow)

- Mirrors syntax of scikit-learn
 - but is TensorFlow under the hood
- Defines many standard models out of the box
- Allows creation of custom models
 - with simplifications for commonly created layers
- Allows many controls over the optimization used
- Was third party (while in alpha)
 - but now integrated and merged with TensorFlow core
 - `tf.contrib.learn`
 - **TF Learn** created and maintained by Google developers!
 - Google recommends this:
 - for learning Tensorflow
 - for getting a model off the ground quickly

TensorFlow and SKFlow

Building and Simplifying
Linear Regression

Other tutorials:

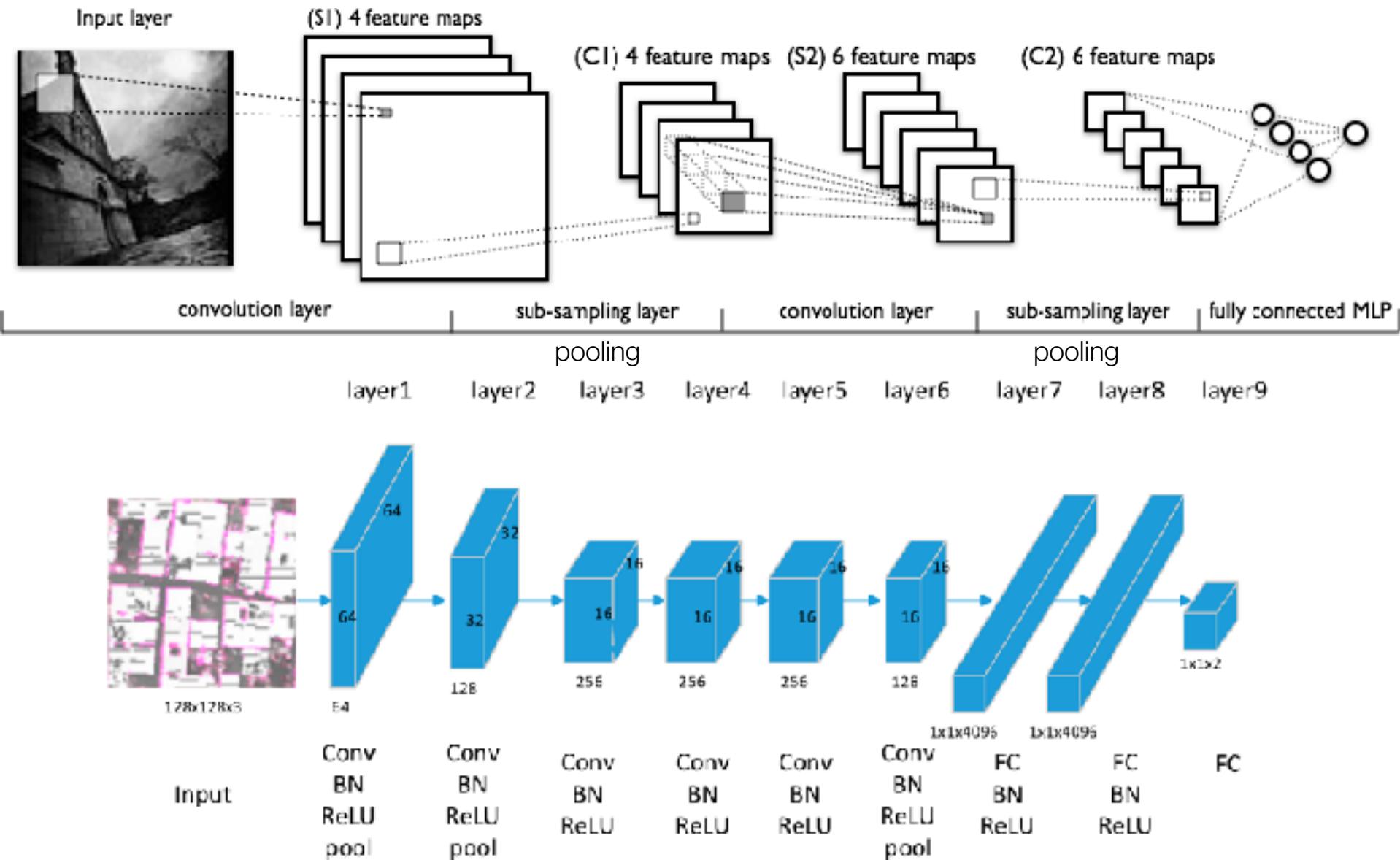
<https://github.com/jtoy/awesome-tensorflow>

<https://github.com/tensorflow/tensorflow/tree/r0.11/tensorflow/examples/skflow>

Or do a Google search!!! They are everywhere!!!



A brief reminder about CNNs:



TensorFlow and SKFlow

Convolutional Neural Networks
in TensorFlow
with SKFlow



Next Lecture

- Recurrent Neural Network Architectures
 - RNNs, GRUs, LSTMs
 - Training for character generation

Lecture Notes for Machine Learning in Python

Professor Eric Larson
Week 12, 13, 14 Deep Learning

Class logistics

- Grades!
- For resubmitting:
 - tell me what you would have changed in the code
 - if it affects A3, be sure to fix it for A3

Lecture Agenda

- Long- Short-term Memory Networks
 - Overview
 - Training
 - Demo
- A great resource:
 - <http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html>

Recurrent Neural Networks

WHAT WERE THINGS
LIKE IN THE OLD DAYS?
I HEAR YOU HAD TO...
COMPILE THINGS FOR
DIFFERENT PROCESSORS?



YEAH.

TO COMPILE YOUR
CODE, YOU HAD TO
MAIL IT TO IBM.
TOOK 4-6 WEEKS.



BEFORE GARBAGE COLLECTION,
DATA WOULD PILE UP UNTIL
THE COMPUTER GOT FULL AND
YOU HAD TO THROW IT AWAY.



EARLY COMPILERS
COULD HANDLE CODE
FINE, BUT COMMENTS
HAD TO BE WRITTEN
IN ASSEMBLY.



C COULD ONLY BE WRITTEN
ON PUNCH CARDS. YOU HAD
TO PICK A COMPACT FONT,
OR YOU'D ONLY FIT A FEW
CHARACTERS PER CARD.



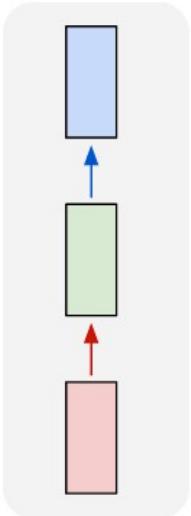
C++ WAS BIG BECAUSE IT
SUPPORTED FLOPPY DISKS.
IT STILL JUST PUNCHED HOLES
IN THEM, BUT IT WAS A START.



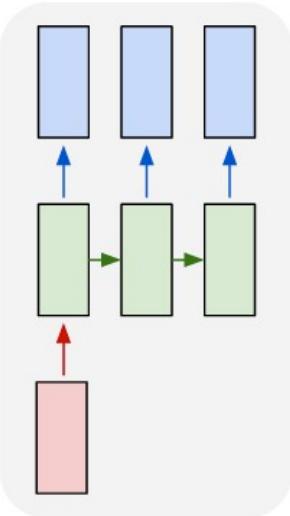
WOW.

Recurrent Networks: Problem Types

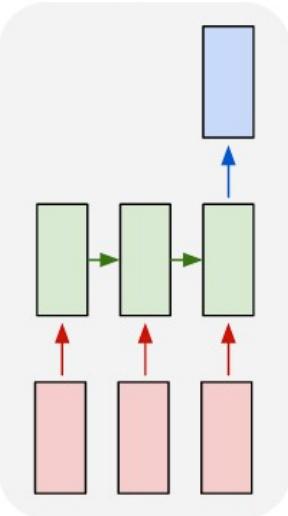
one to one



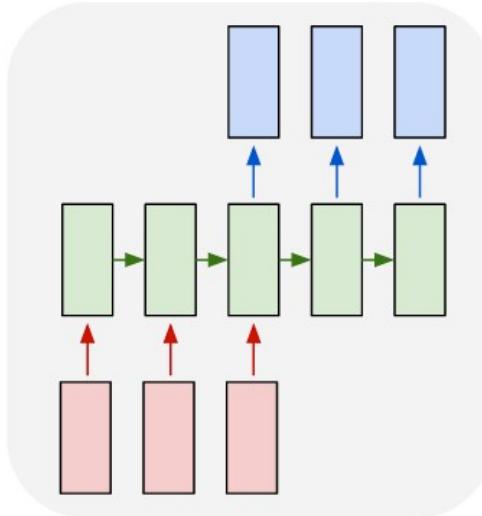
one to many



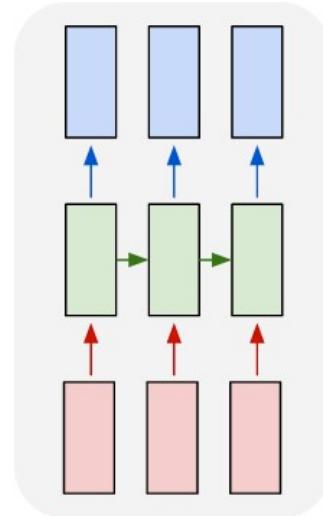
many to one



many to many

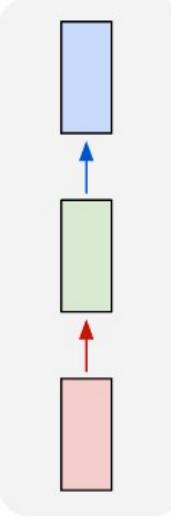


many to many

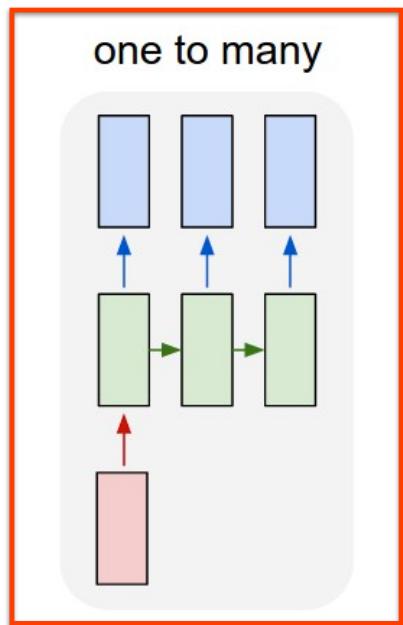


Recurrent Networks: Problem Types

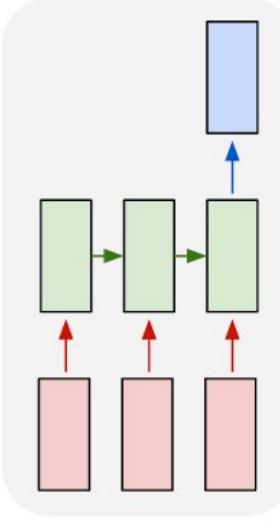
one to one



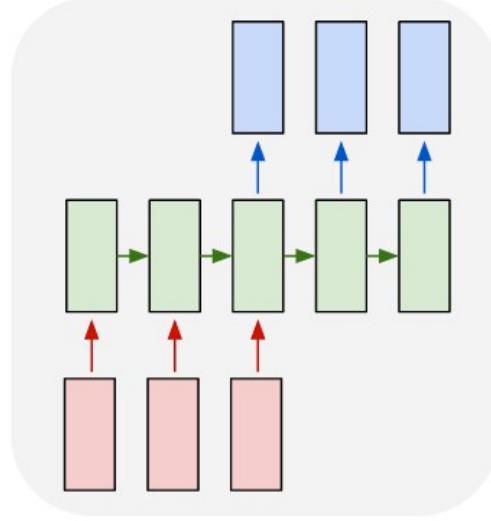
one to many



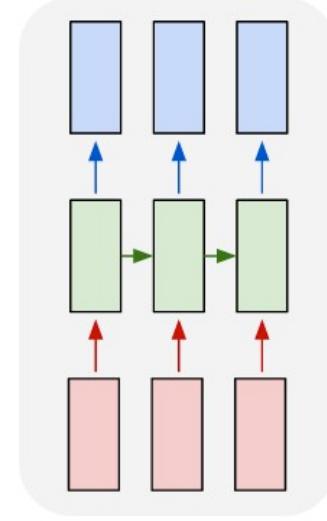
many to one



many to many



many to many



A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



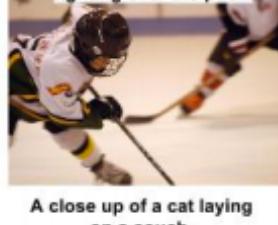
A skateboarder does a trick on a ramp.



A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



A little girl in a pink hat is blowing bubbles.



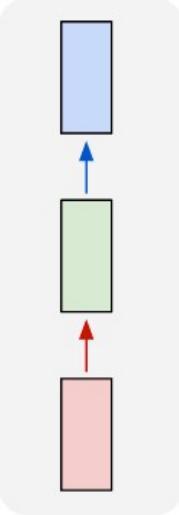
A herd of elephants walking across a dry grass field.

A close up of a cat laying on a couch.

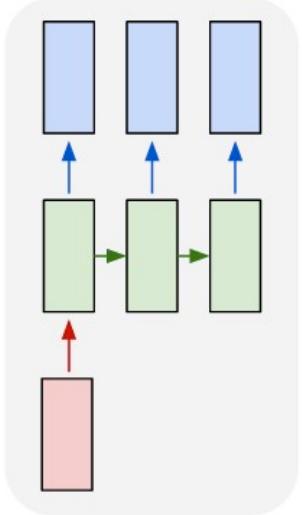
A red motorcycle parked on the side of the road.

Recurrent Networks: Problem Types

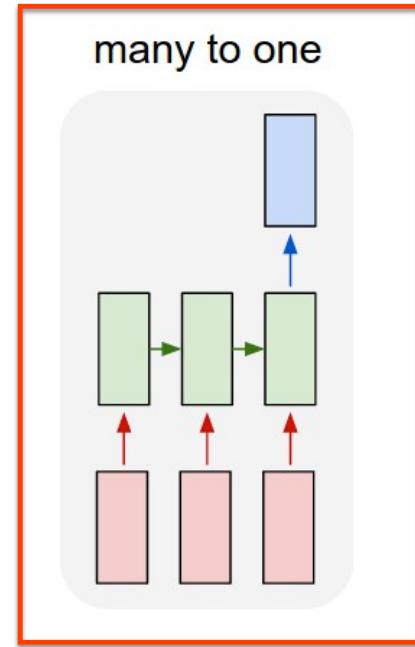
one to one



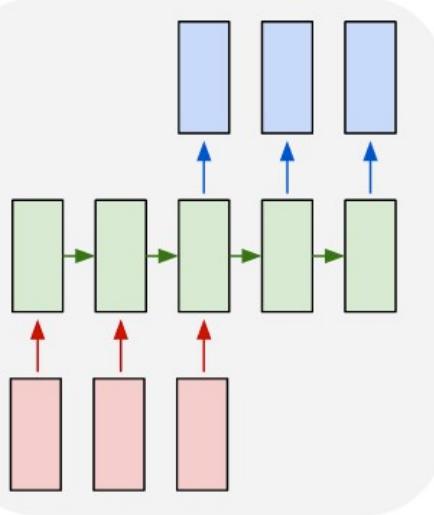
one to many



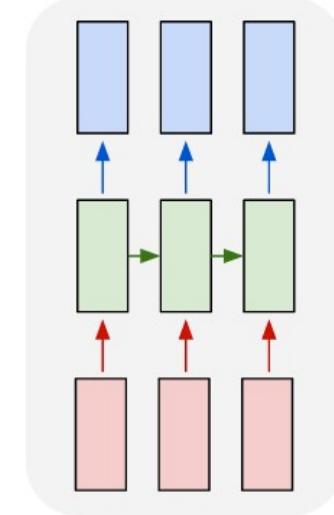
many to one



many to many



many to many



The movie is great.



The movie stars Mr. X

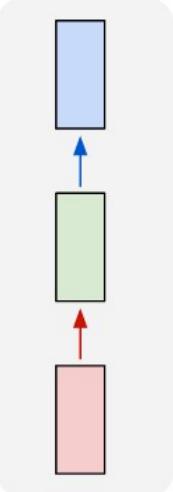


The movie is horrible.

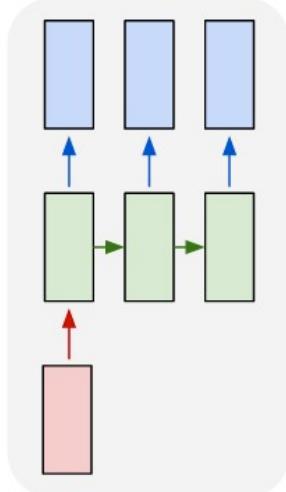


Recurrent Networks: Problem Types

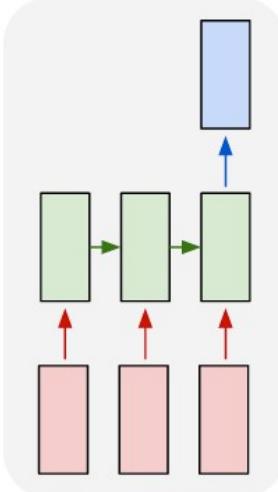
one to one



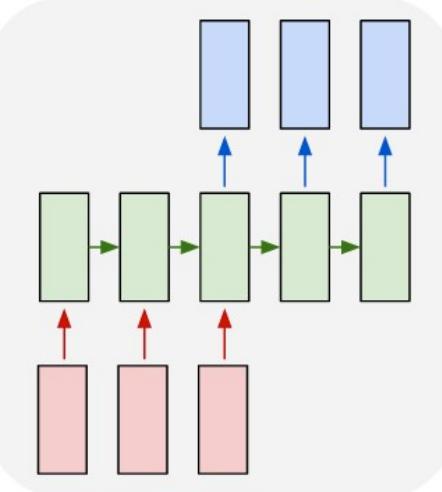
one to many



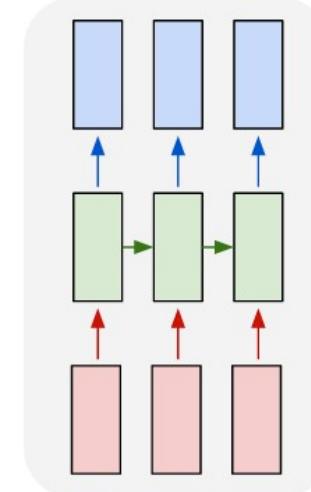
many to one



many to many



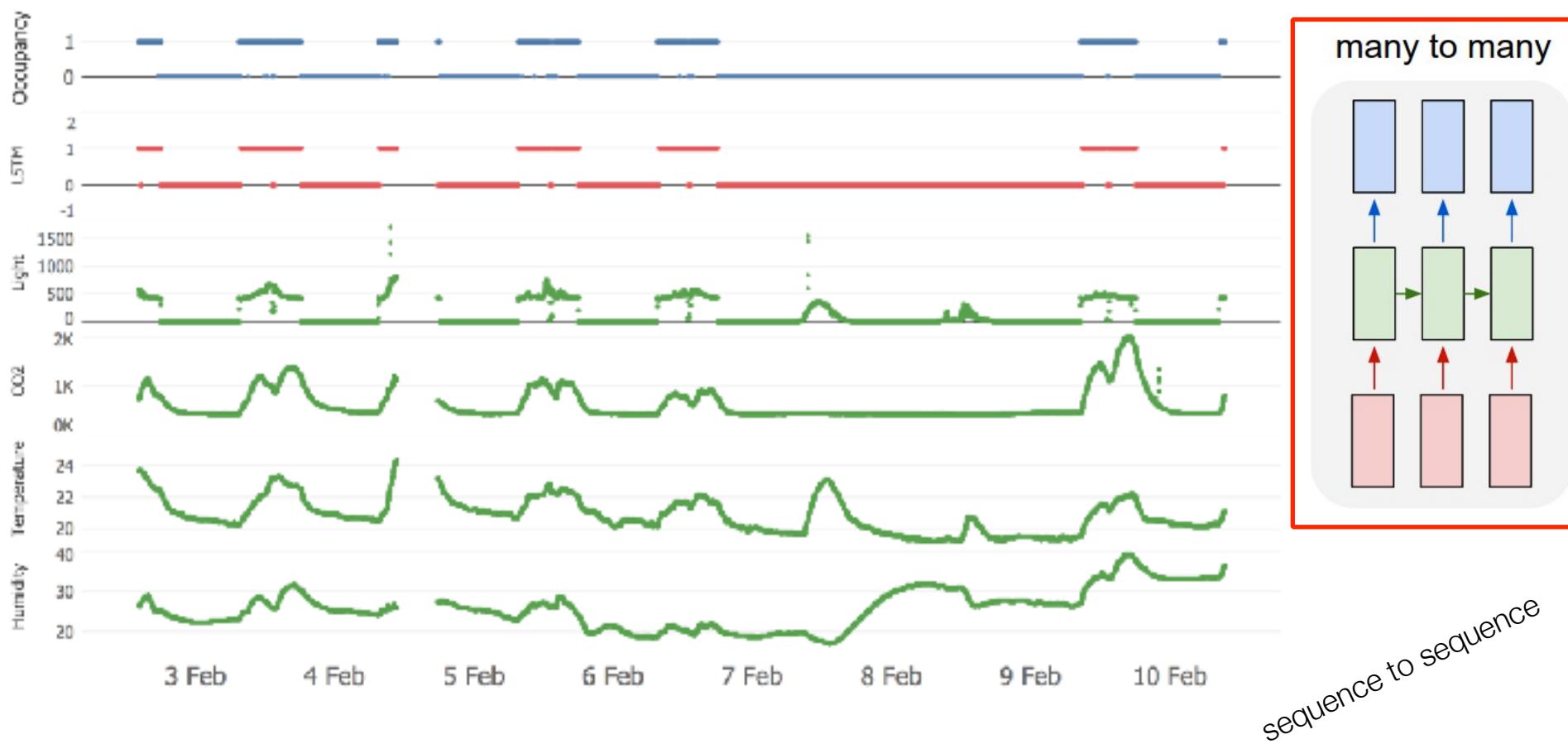
many to many



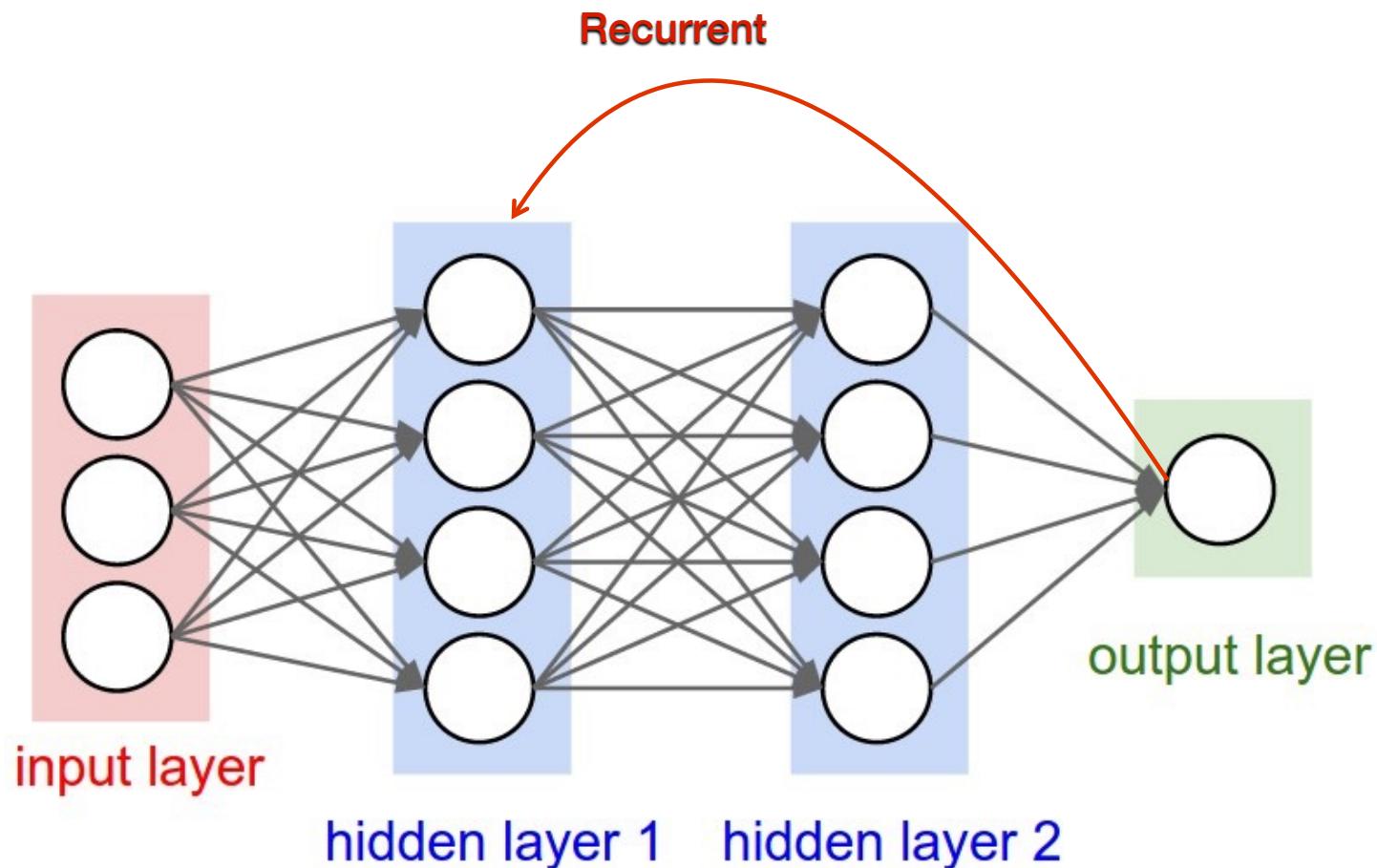
Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt .
Economic growth has slowed down in recent years .

La croissance économique s' est ralentie ces dernières années .

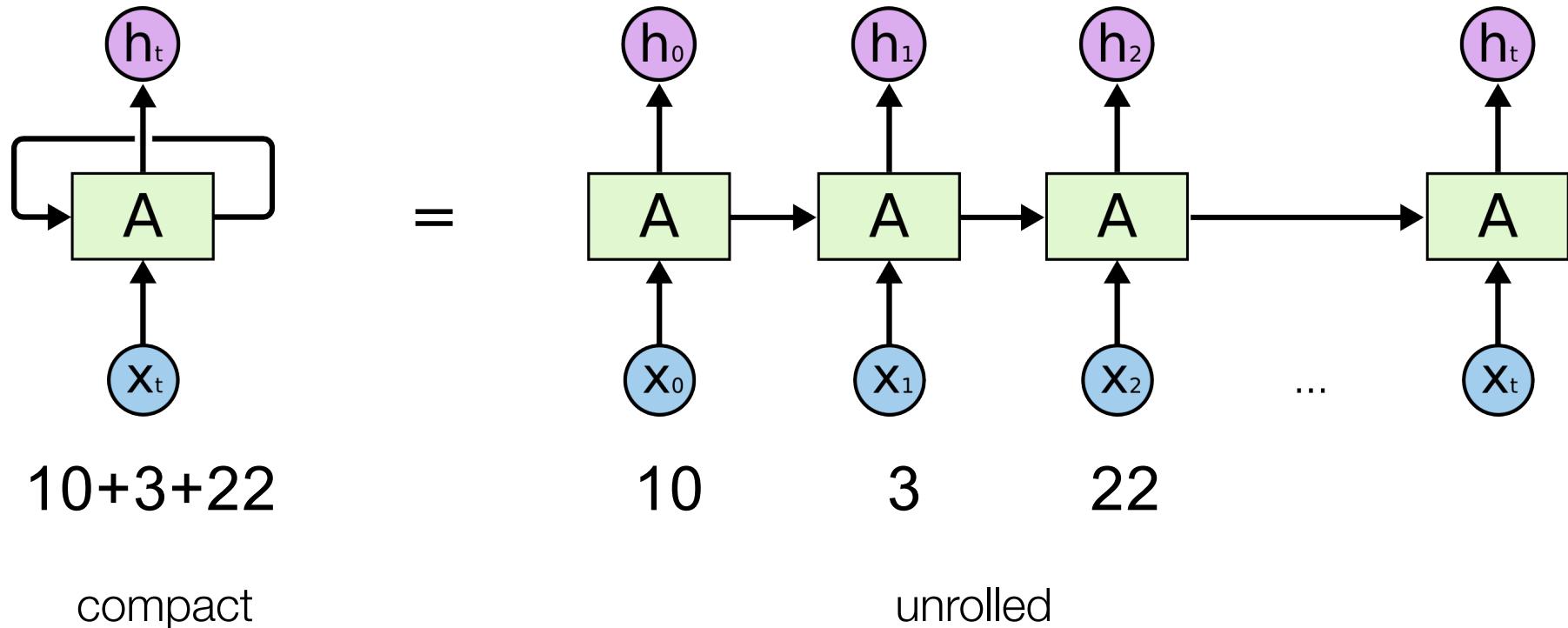
Recurrent Networks: Problem Types



Recurrent Networks: Main Idea

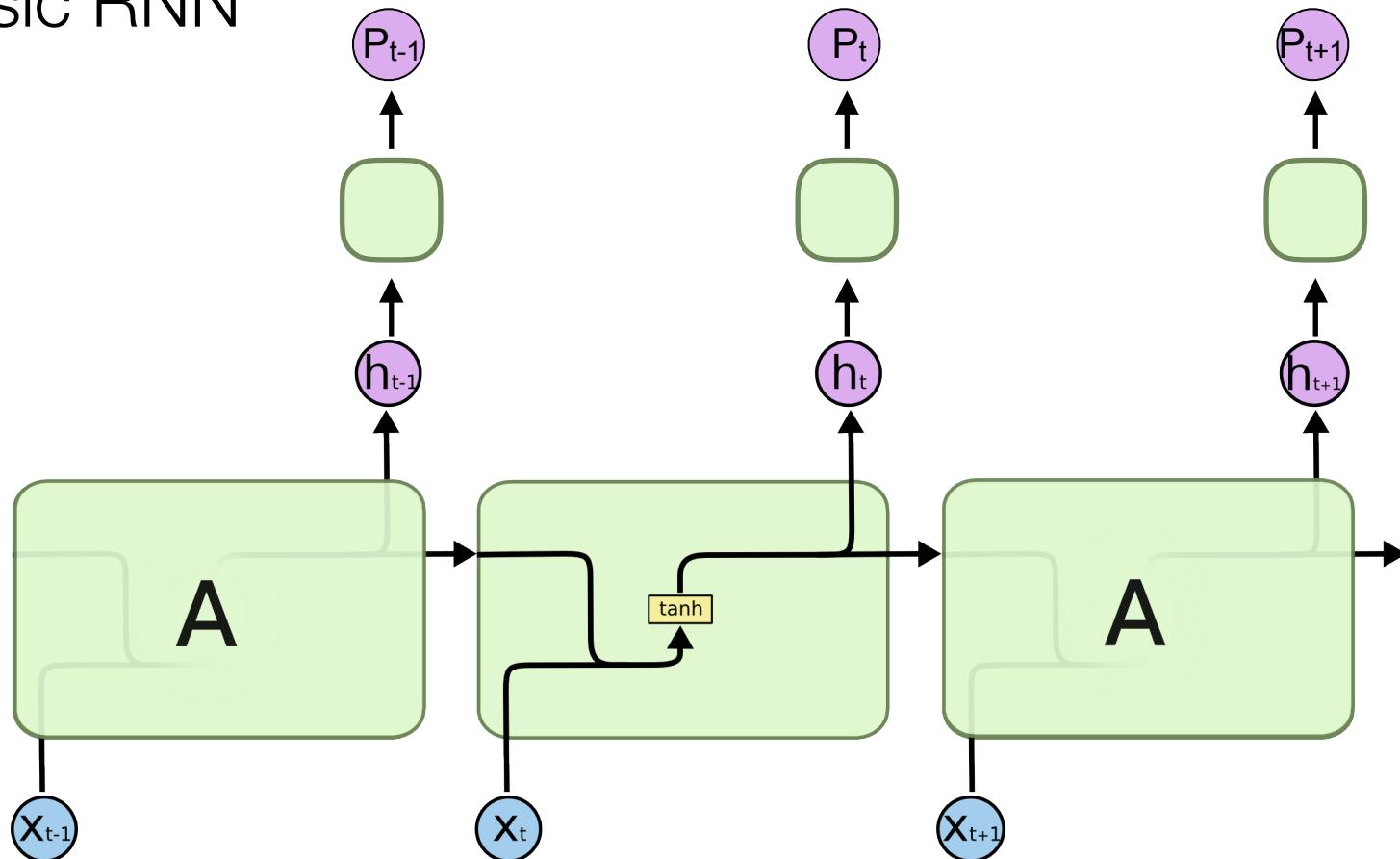


Recurrent Networks: Main Idea



Recurrent Networks

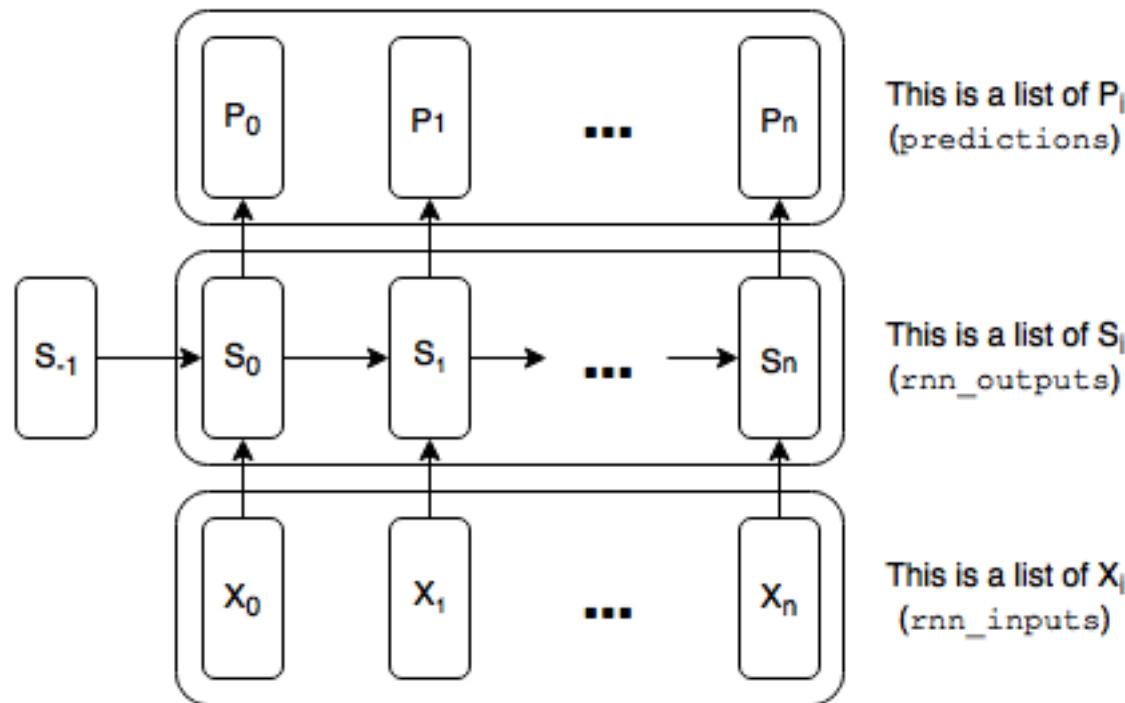
- basic RNN



$$h_t = \tanh(W_A (X_t @ h_{t-1}) + b_A)$$
$$P_t = \text{softmax}(W_P h_t + b_P)$$

Recurrent Networks: Representation

- python lists



character: C A T

0	1	0
0	0	1
1	0	0

TensorFlow

<http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html>

```
with tf.variable_scope('rnn_cell'):
    W = tf.get_variable('W', [num_classes + state_size, state_size])
    b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(0.0))

def rnn_cell(rnn_input, state):
    with tf.variable_scope('rnn_cell', reuse=True):
        W = tf.get_variable('W', [num_classes + state_size, state_size])
        b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(0.0))
    return tf.tanh(tf.matmul(tf.concat([rnn_input, state]), W) + b)

state = init_state
rnn_outputs = []
for rnn_input in rnn_inputs:
    state = rnn_cell(rnn_input, state)
    rnn_outputs.append(state)
final_state = rnn_outputs[-1]

#logits and predictions
with tf.variable_scope('softmax'):
    W = tf.get_variable('W', [state_size, num_classes])
    b = tf.get_variable('b', [num_classes], initializer=tf.constant_initializer(0.0))
logits = [tf.matmul(rnn_output, W) + b for rnn_output in rnn_outputs]
predictions = [tf.nn.softmax(logit) for logit in logits]

# Turn our y placeholder into a list labels
y_as_list = [tf.squeeze(i, squeeze_dims=[1]) for i in tf.split(1, num_steps, y)]

#losses and train_step
losses = [tf.nn.sparse_softmax_cross_entropy_with_logits(logit,label) for \
          logit, label in zip(logits, y_as_list)]
total_loss = tf.reduce_mean(losses)
train_step = tf.train.AdagradOptimizer(learning_rate).minimize(total_loss)
```

TensorFlow

<http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html>

```
def train_network(num_epochs, num_steps, state_size=4, verbose=True):
    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())
        for idx, epoch in enumerate(gen_epochs(num_epochs, num_steps)):
            training_state = np.zeros((batch_size, state_size))
            for X, Y in epoch:
                tr_losses, training_loss_, training_state, _ = \
                    sess.run([losses,
                             total_loss,
                             final_state,
                             train_step],
                            feed_dict={x:X, y:Y, init_state:training_state})
```

TensorFlow (simplified)

<http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html>

```
cell = tf.nn.rnn_cell.BasicRNNCell(state_size)
rnn_outputs, final_state = tf.nn.rnn(cell, rnn_inputs, initial_state=init_state)

loss_weights = [tf.ones([batch_size]) for i in range(num_steps)]
losses = tf.nn.seq2seq.sequence_loss_by_example(logits, y_as_list, loss_weights)

x = tf.placeholder(tf.int32, [batch_size, num_steps], name='input_placeholder')
y = tf.placeholder(tf.int32, [batch_size, num_steps], name='labels_placeholder')
init_state = tf.zeros([batch_size, state_size])

x_one_hot = tf.one_hot(x, num_classes)
rnn_inputs = tf.unpack(x_one_hot, axis=1)

cell = tf.nn.rnn_cell.BasicRNNCell(state_size)
rnn_outputs, final_state = tf.nn.rnn(cell, rnn_inputs, initial_state=init_state)

with tf.variable_scope('softmax'):
    W = tf.get_variable('W', [state_size, num_classes])
    b = tf.get_variable('b', [num_classes], initializer=tf.constant_initializer(0.0))
logits = [tf.matmul(rnn_output, W) + b for rnn_output in rnn_outputs]
predictions = [tf.nn.softmax(logit) for logit in logits]

y_as_list = [tf.squeeze(i, squeeze_dims=[1]) for i in tf.split(1, num_steps, y)]

loss_weights = [tf.ones([batch_size]) for i in range(num_steps)]
losses = tf.nn.seq2seq.sequence_loss_by_example(logits, y_as_list, loss_weights)
total_loss = tf.reduce_mean(losses)
train_step = tf.train.AdagradOptimizer(learning_rate).minimize(total_loss)
```

Recurrent Networks: Character Data

First Citizen:

Before we proceed any further, hear me speak.

All:

Speak, speak.

First Citizen:

You are all resolved rather to die than to famish?

All:

Resolved. resolved.

First Citizen:

First, you know Caius Marcius is chief enemy to the people.

All:

We know't, we know't.

First Citizen:

Let us kill him, and we'll have corn at our own price.

Is't a verdict?

All:

No more talking on't; let it be done: away, away!

Second Citizen:

One word, good citizens.

ervices he has done for his country?

First Citizen:

We are accounted poor citize
What authority surfeits on w
would yield us but the super
wholesome, we might guess th
but they think we are too de
afflicts us, the object of o
inventory to particularise t
sufferance is a gain to them
our pikes, ere we become rak
speak this in hunger for bre

Second Citizen:

Would you proceed especially

All:

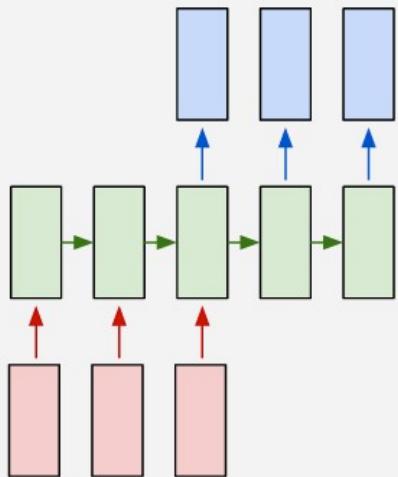
Against him first: he's a ve

Second Citizen:

Consider you what s

Recurrent Networks: Character Generation

many to many



ATOOOS

UIEAOUYOUZZZZUZAAAYAYf n fsflflrurctuateot t ta's a wtut
Whith then, a do makes and them and to sees,
I wark on this ance may string take thou honon
To sorriccorn of the bairer, whither, all
I'd see if yiust the would a peid.

LARYNGLe:

To would she troust they fould.

PENMES:

Thou she so the havin to my shald woust of
As tale we they all my forder have
As to say heant thy wansing thag and
Whis it thee shath his breact, I be and might, she
Tirs you desarvishensed and see thee: shall,
What he hath with that is all time,
And sen the have would be sectiens, way thee,
They are there to man shall with me to the mon,
And mere fear would be the balte, as time an at
And the say oun touth, thy way womers thee.

Recurrent Networks: Character Generation

more data, star wars + star trek + tarantino + the matrix

DENT ' SUEENCK

Bartholomew of the TIE FIGHTERS are stunned. There is a crowd and armored
switcheroos.

PICARD

(continuing)

Couns two dim is tired. In order to the sentence...

The sub bottle appears on the screen into a small shuttle shift of the
ceiling. The DAMBA FETT splash fires and matches them into the top, transmit to stable high above
upon their statels,
falling from an alien shaft.

ANAKIN and OBI-WAN stand next to OBI-WAN down the control plate of smoke at the TIE fighter. They
stare at the centre of the station loose into a comlink cover -- comes up to the General, the
GENERAL HUNTAN AND FINNFURMBARD from the PICADOR to a beautiful Podracisly.

ENGINEER

Naboo from an army seventy medical
security team area re-weilergular.

EXT.

Recurrent Networks: Ontology Classification

Eva Ingolf is a well known Icelandic violinist particularly recognized for her authoritative performances of solo works by J. S. Bach. She comes from a leading musical family and her father Ingólfur Guðbrandsson premiered many of the great choral works in Iceland and six of her sisters and brothers are professional musicians who have made an important contribution to the high quality of the musical life in the country. Eva Ingolf currently lives in New York City with her husband Kristinn Sv.

Artist

Shaun Norris (born 14 May 1982) is a South African professional golfer. Norris plays on the Sunshine Tour where he has won twice. He won the inaugural Africa Open in 2008 and the Nashua Masters in 2011. He also began playing on the European Tour in 2011 after graduating from qualifying school.

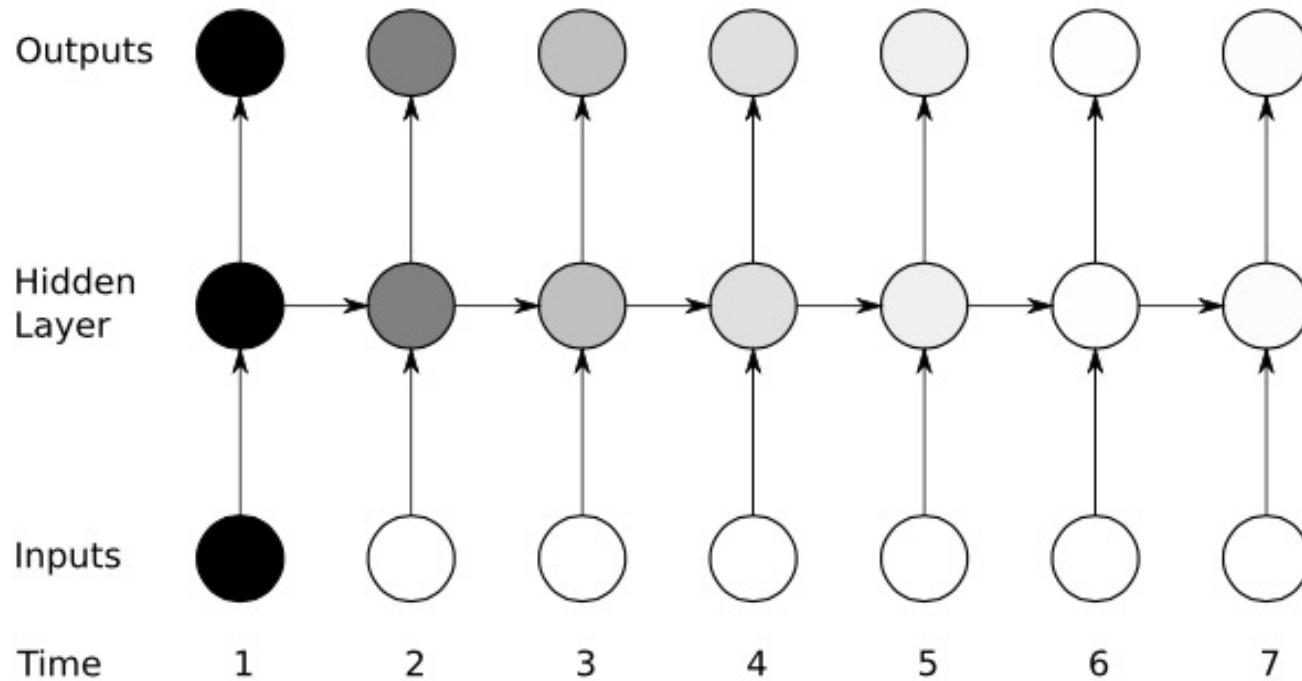
Athlete

Palace Software was a British video game publisher and developer during the 1980s based in London England. It was notable for the Barbarian and Cauldron series of games for 8-bit home computer platforms in particular the ZX Spectrum Amstrad CPC and Commodore 64.

Company

Recurrent Networks Problem

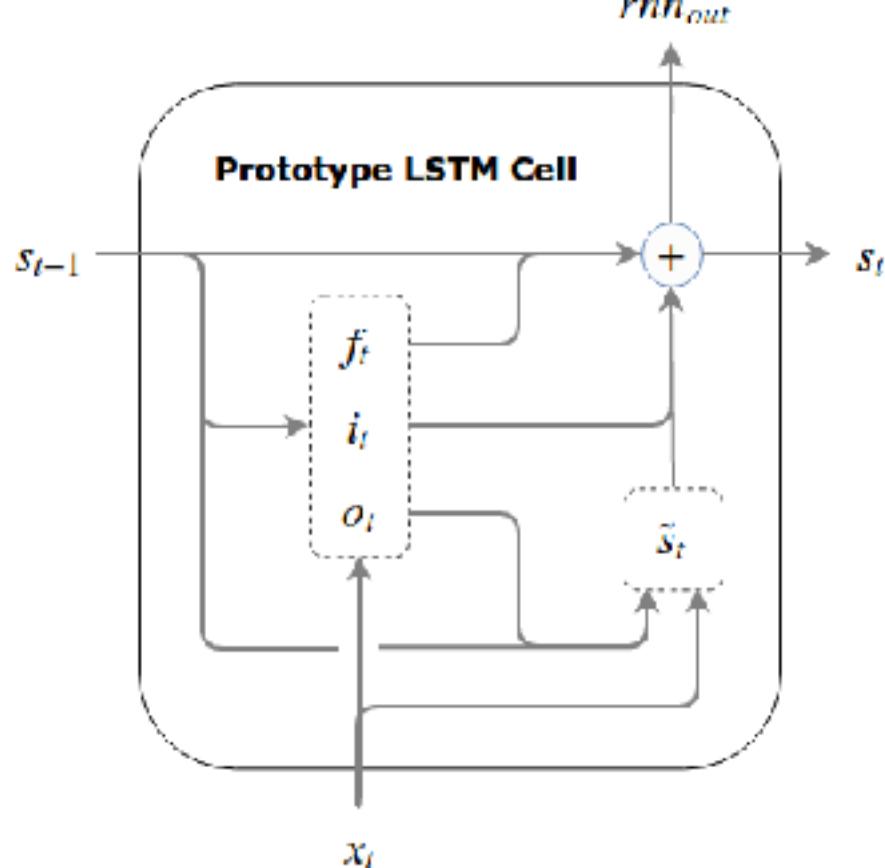
- vanishing gradients: how to mitigate?



$$\frac{\partial E_t}{\partial S_{t-k}} = \frac{\partial E_t}{\partial S_t} \frac{\partial S_t}{\partial S_{t-k}} = \frac{\partial E_t}{\partial S_t} \left(\frac{\partial S_t}{\partial S_{t-1}} \frac{\partial S_{t-1}}{\partial S_{t-2}} \dots \frac{\partial S_{t-k+1}}{\partial S_{t-k}} \right) = \frac{\partial E_t}{\partial S_t} \prod_{i=1}^k \frac{\partial S_{t-i+1}}{\partial S_{t-i}}$$

Recurrent Networks: Gen 1 LSTM

- LSTM prototype



Selectivity controls

$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

$$f_t = \sigma(W_f s_{t-1} + U_f x_t + b_f)$$

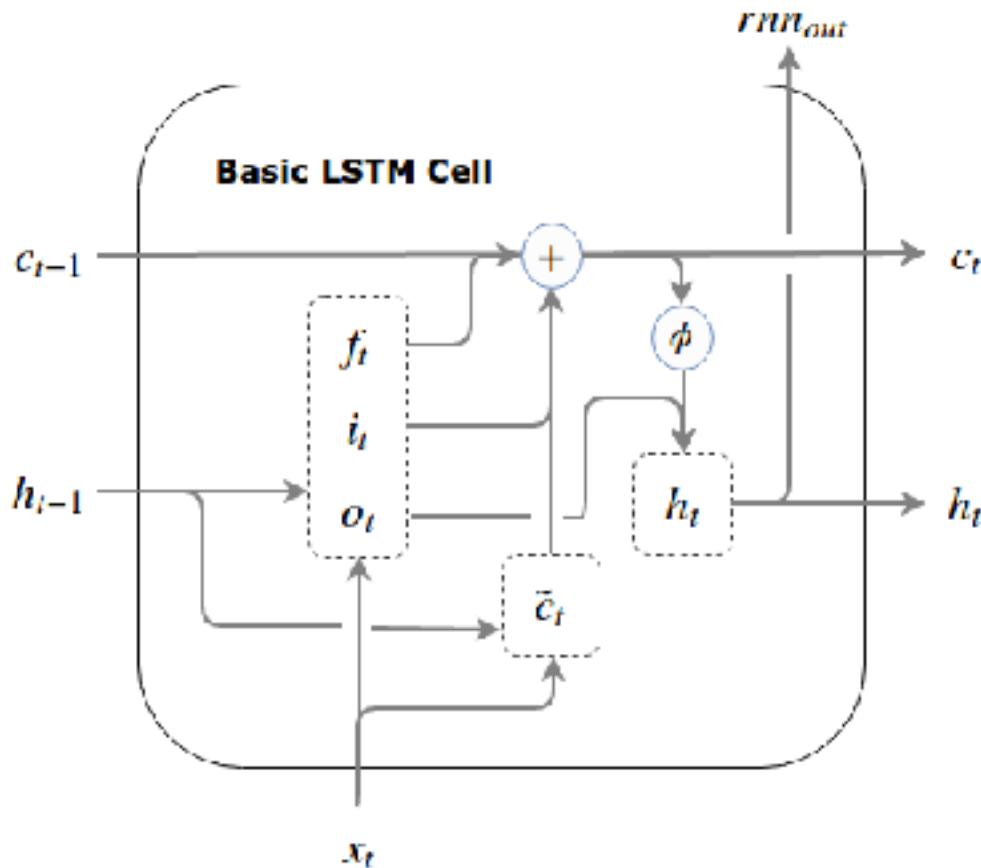
$$\tilde{s}_t = \phi(W(o_t \odot s_{t-1}) + Ux_t + b)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

remember past with input

Recurrent Networks: Gen 2 LSTM

- LSTM in TensorFlow



Selectivity controls

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

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

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

explicit remembering state

$$\tilde{c}_t = \phi(W h_{t-1} + U x_t + b)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

remember state remember output

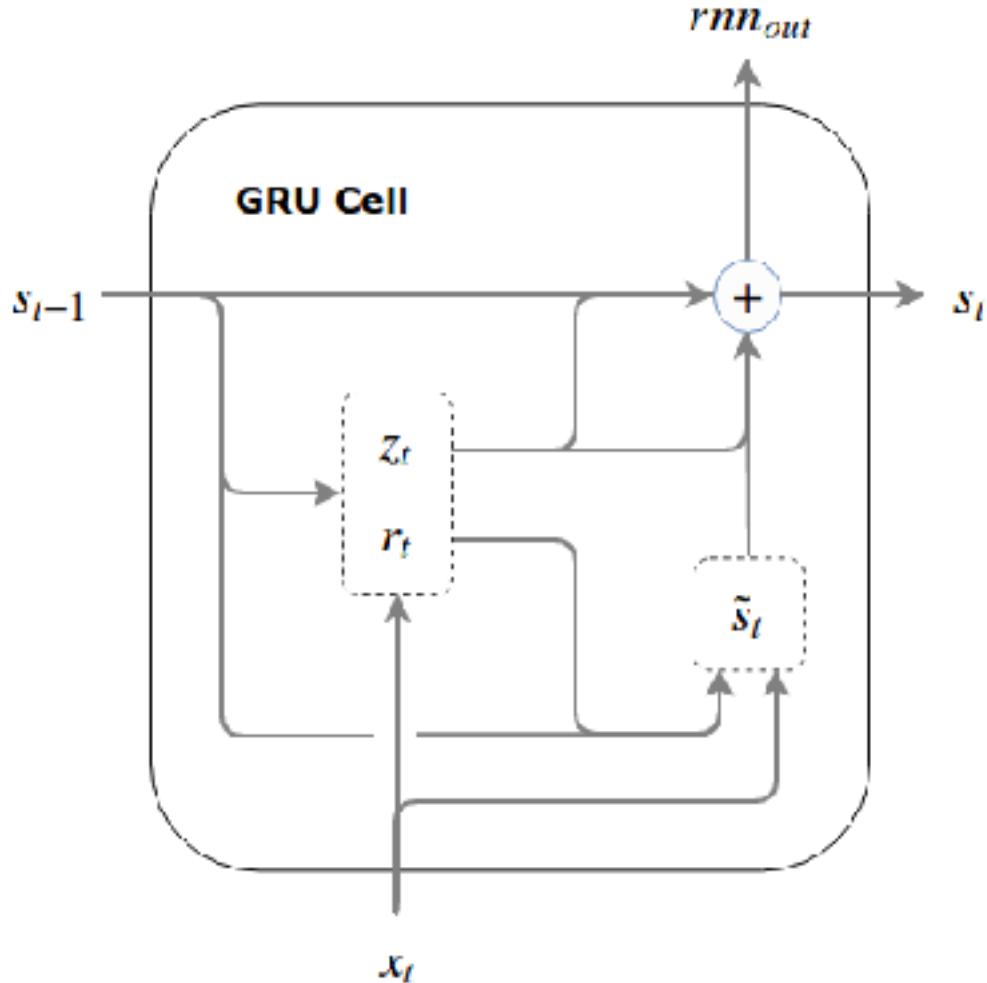
$$h_t = o_t \odot \phi(c_t)$$

remember both

$$rnn_{out} = h_t$$

Recurrent Networks: GRUs

- gated recurrent units



Selectivity controls

$$r_t = \sigma(W_r s_{t-1} + U_r x_t + b_r)$$

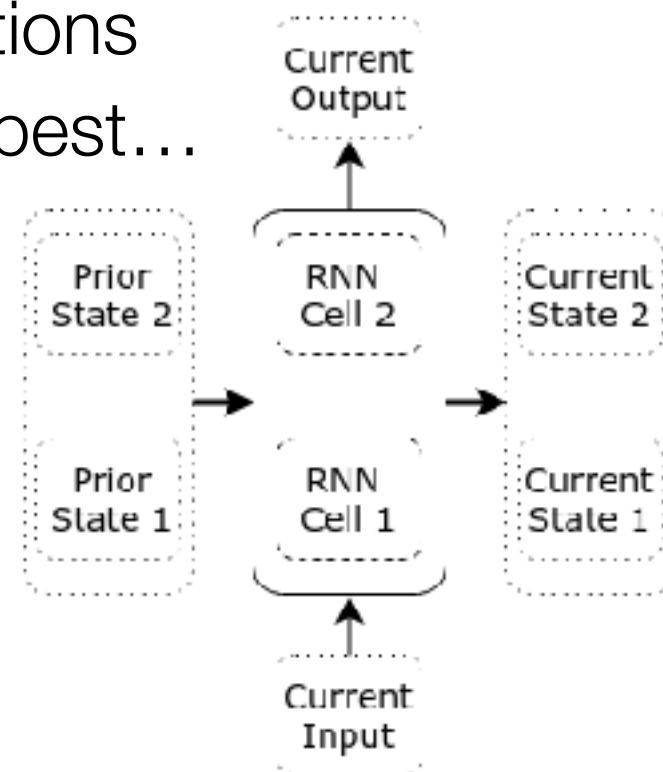
$$z_t = \sigma(W_z s_{t-1} + U_z x_t + b_z)$$

$$\begin{aligned}\tilde{s}_t &= \phi(W(r_t \odot s_{t-1}) + Ux_t + b) \\ s_t &= z_t \odot s_{t-1} + (1 - z_t) \odot \tilde{s}_t\end{aligned}$$

remember with input
remember past remember with input

What to choose?

- There is no hard and fast rule
 - try both
 - Basic LSTM has had great success
 - GRU also sometimes is easier to train
 - you will see many variations
 - still developing what is best...



Final Demo

TensorFlow and SKFlow

Many to one:

RNNs

GRUs

LSTMs

More examples:

<https://github.com/tensorflow/tensorflow/tree/r0.11/tensorflow/examples/skflow>

<http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html>

Seq2Seq:

https://github.com/tensorflow/tensorflow/blob/r0.11/tensorflow/examples/skflow/neural_translation_word.py

