BRV CTO Summit

- UC Berkeley Undergrad/Stanford Grad EE
    - Statistical physics helped/tensors, Lagrangian
- Tech Lead @Google/Waymo started w/Book Search
    - self driving cars/RTOS C++

Outline

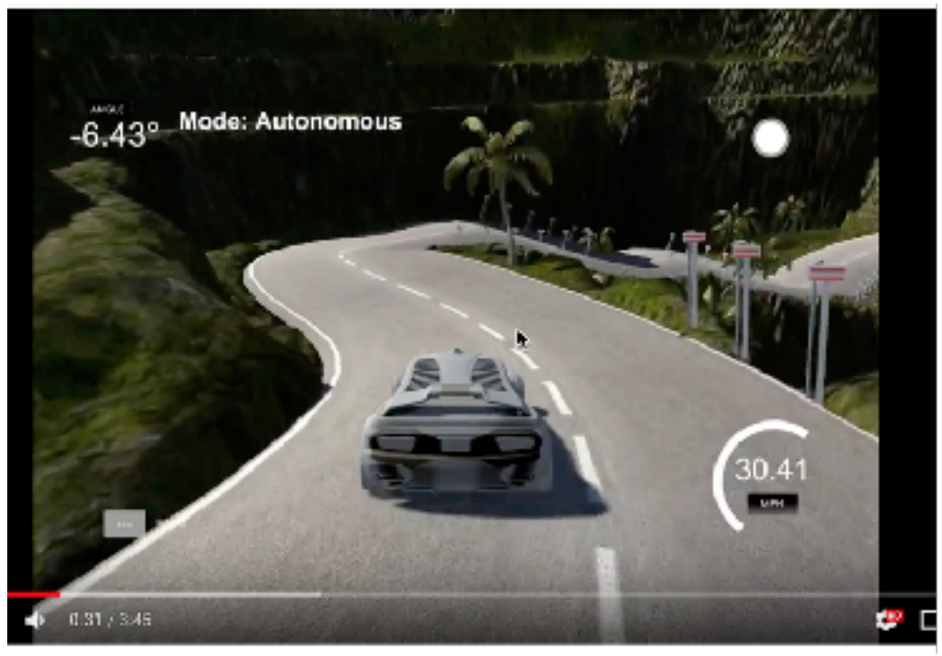
- Basic Architectures — Group discussion first
- What I need from you/similar to Andrei Pop discussions
  - CNNs
    - Motivation (better than human performance)
    - CNN applications
    - Backprop/ deep dive demo only (replicating papers).
      - tools for getting to 30 deep layers
    - CNN / building a model -> tips for DL
  - RNNs applications
    - RNN language models
    - Word Embeddings(word2Vec) -> Entity Embeddings
      - Rossmann stores
        - State of the art >90% modeling performance.

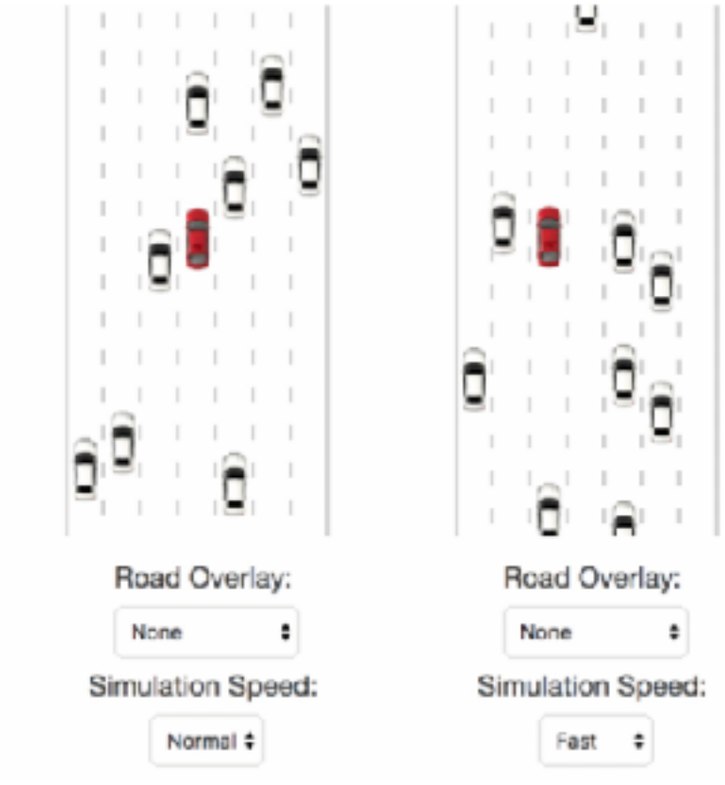
  Group discussion/ Don't need the functional spec <-> need enough experience with your data to be useful
- deepRL (remove) replace w/request for reinforcement learning vs. mobile

Types of Machine Learning

- Supervised(data train)
- Unsupervised(data/dist)
- Reinforcement Learning

Data vs. algorithms/data is used in place of an algorithm





Data vs. Algorithm

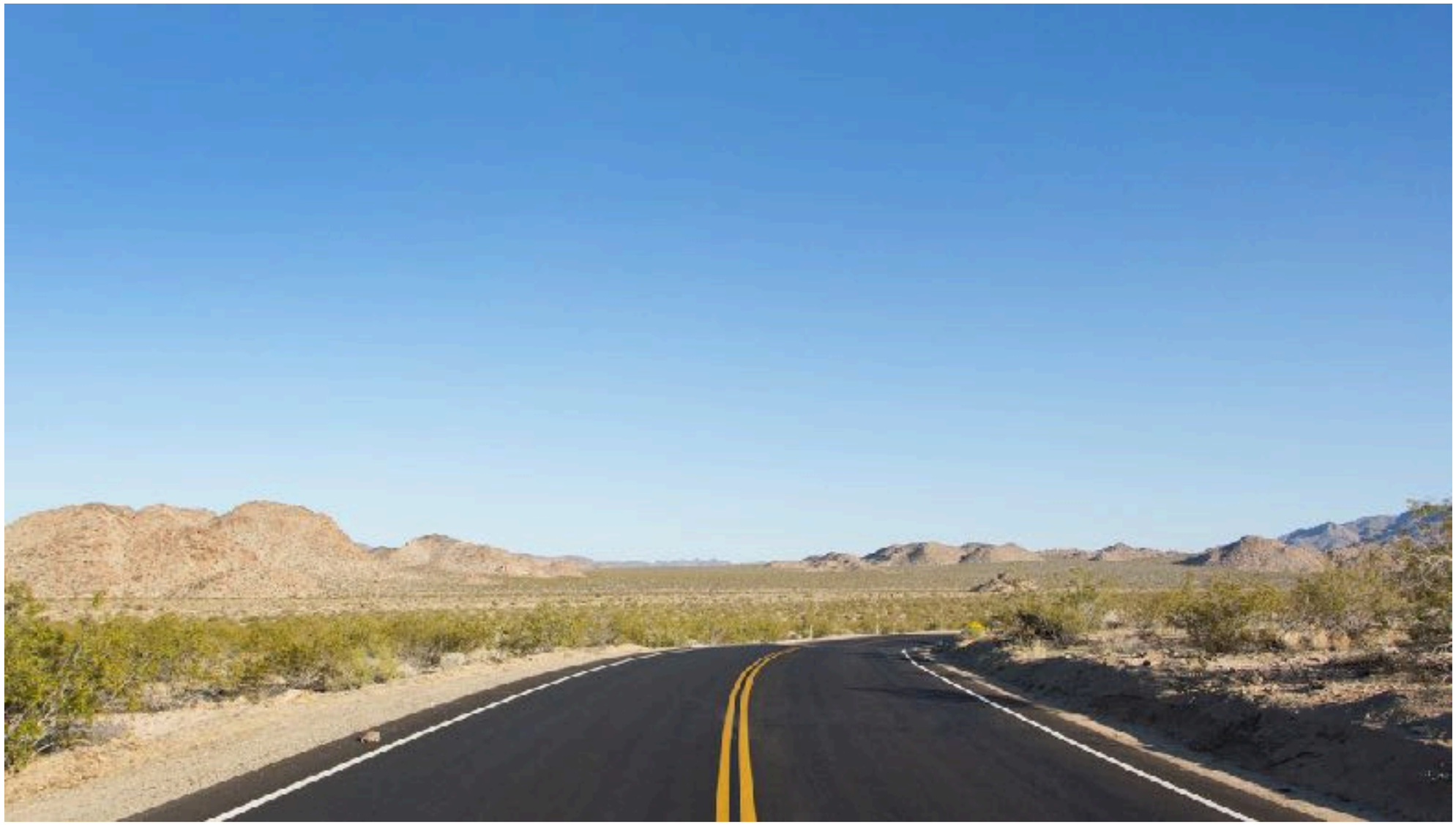
No collision assumption

Computer vision approach

Which approach is cheaper by 50x+?

$600 vs 30k

Some things algos can’t do



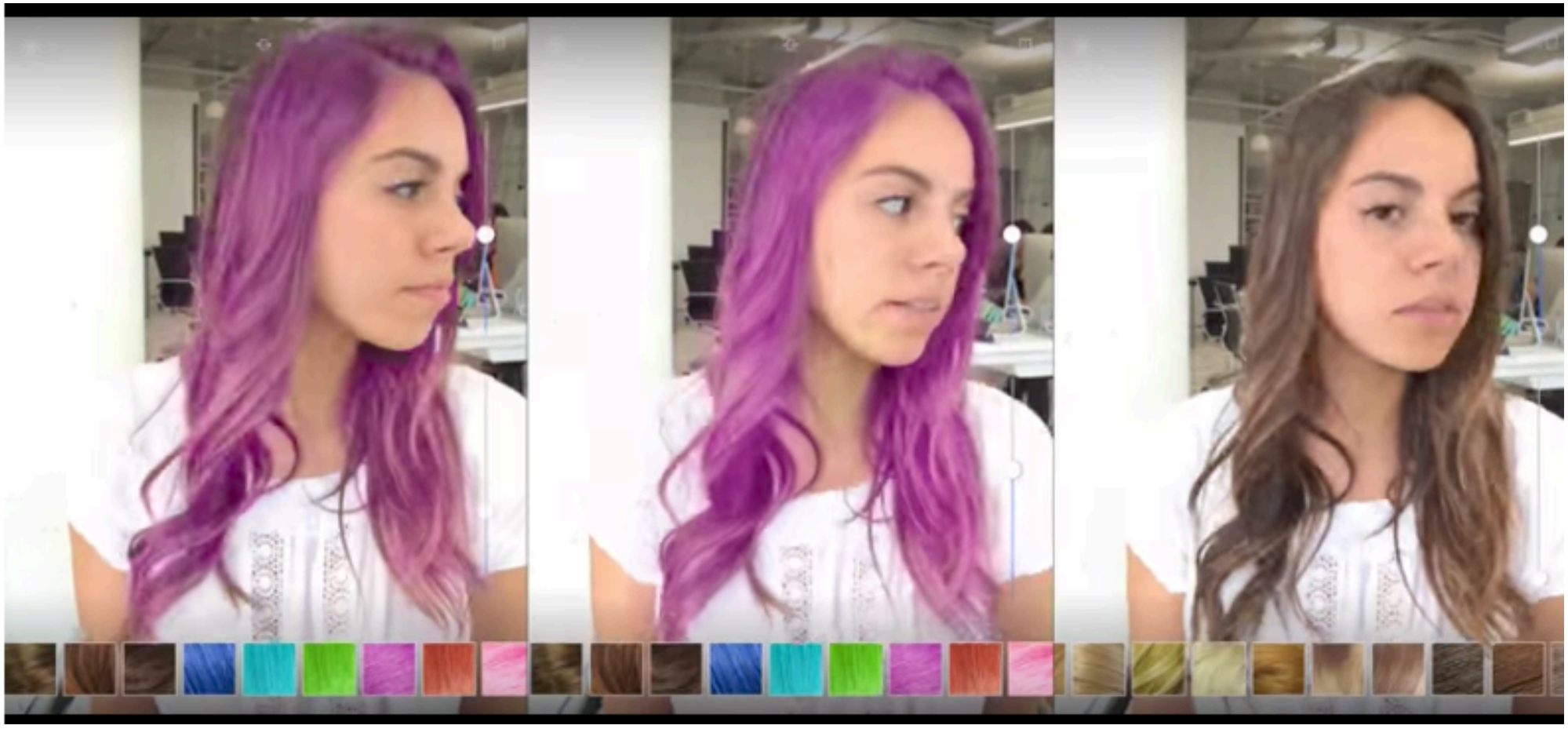
How to solve the turtle problem?

Source: Nvidia

Vincent AI turns a few lines like these ...

... into art like this.

Source: Nvidia



Source: Nvidia

based on the keywords provided by the user (e.g. "restaurant night" or "building beach").

Source

Refs ("restaurant night")

Ours

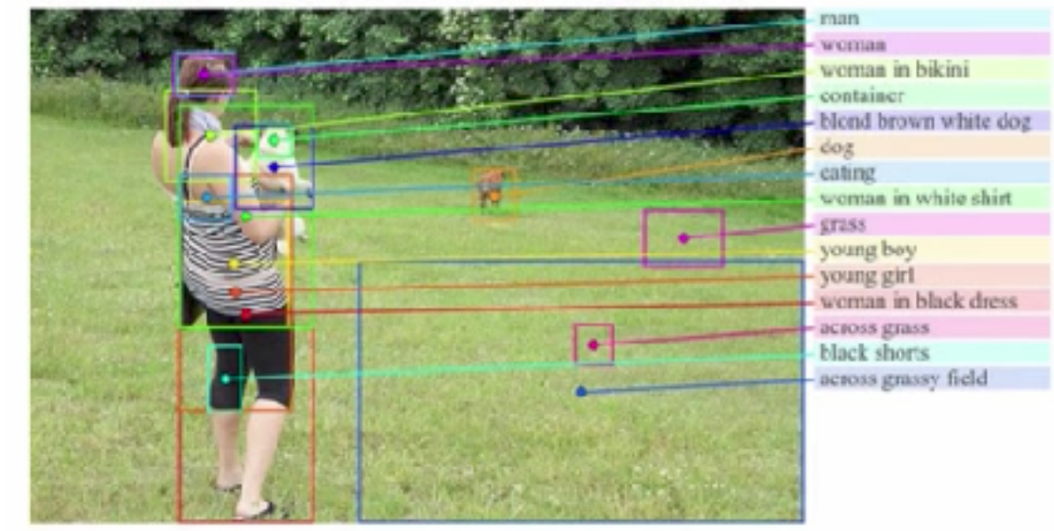
Source

Refs ("building beach")

Ours

Source: Nvidia

https://github.com/jcjohnson/neural-style



Source: [Karpathy and Fei-Fei 2015]

CNN / Image recognition

Better than humans

"construction worker in orange safety vest is working on road."

"baseball player is throwing ball in game."

"black and white dog jumps over bar."

Image captioning

CNN+RNN

RNN for language modeling

JULY 6, 2017

Stanford computer scientists develop an algorithm that diagnoses heart arrhythmias with cardiologist-level accuracy

A new deep learning algorithm can diagnose 14 types of heart rhythm defects, called arrhythmias, better than cardiologists. This could speed diagnosis and improve treatment for people in rural locations.

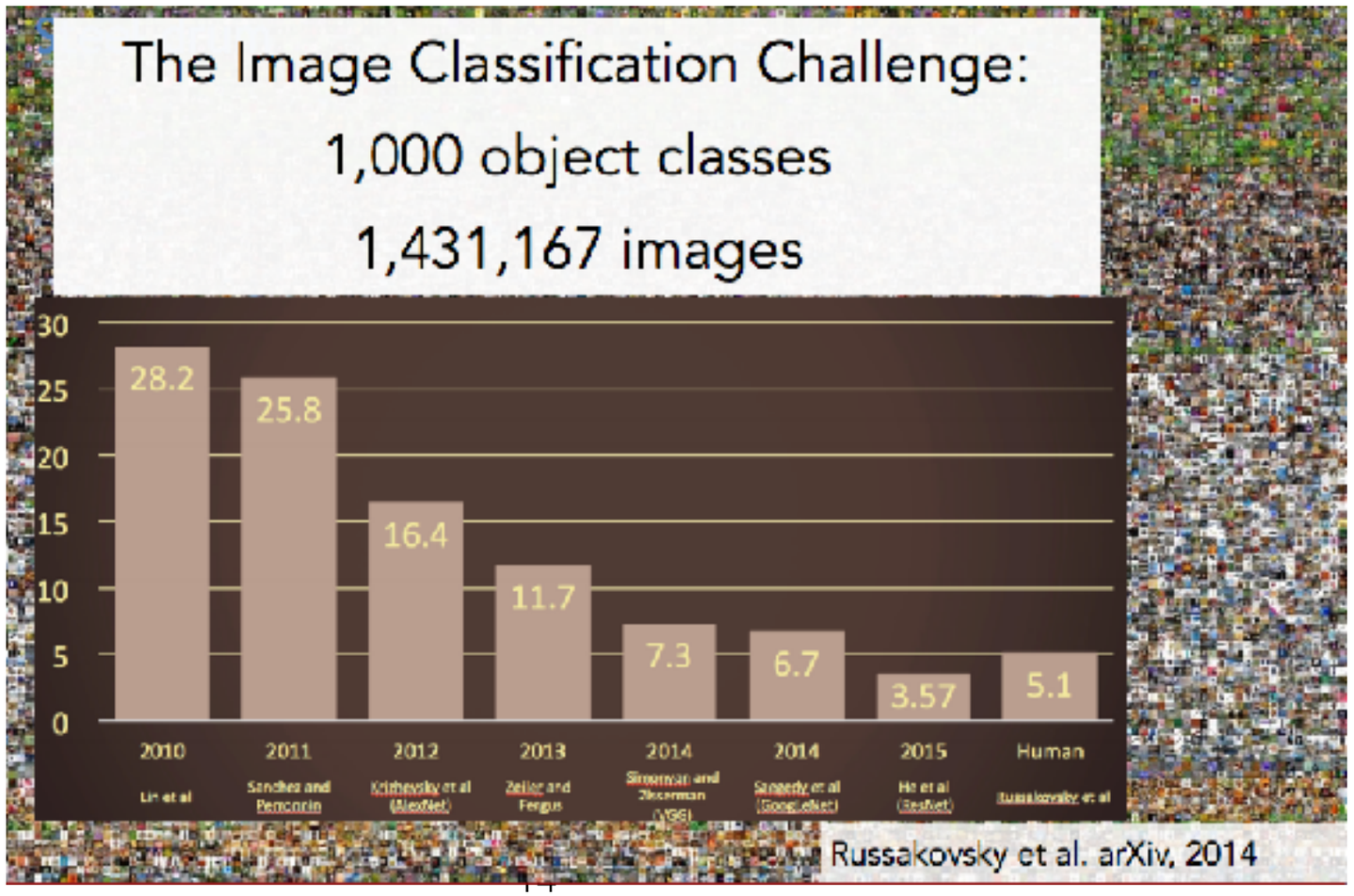
Deep Learning

- Position above Udacity/Coursera classes/solutions
    - Quality/Relevance based on quality of input. (group feed back based for next lecture)
        - HealthAPI embeddings/database; have seen their data and use cases 2+ years ago.
    - Poll: DL in production? Replicating papers? PyTorch vs. TF(production)? Structured Data? Tools for 32layer deep CNN?
- Start CNN/RNN architectures/ following the class format

CNNs - Convoutional Neural Networks

cifar10/cifar100

- ~2015 Beat humans(94% vs. 99%)/ caused creation of cs231n



Source:cs231n

IMAGENET **Large Scale Visual Recognition Challenge**

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

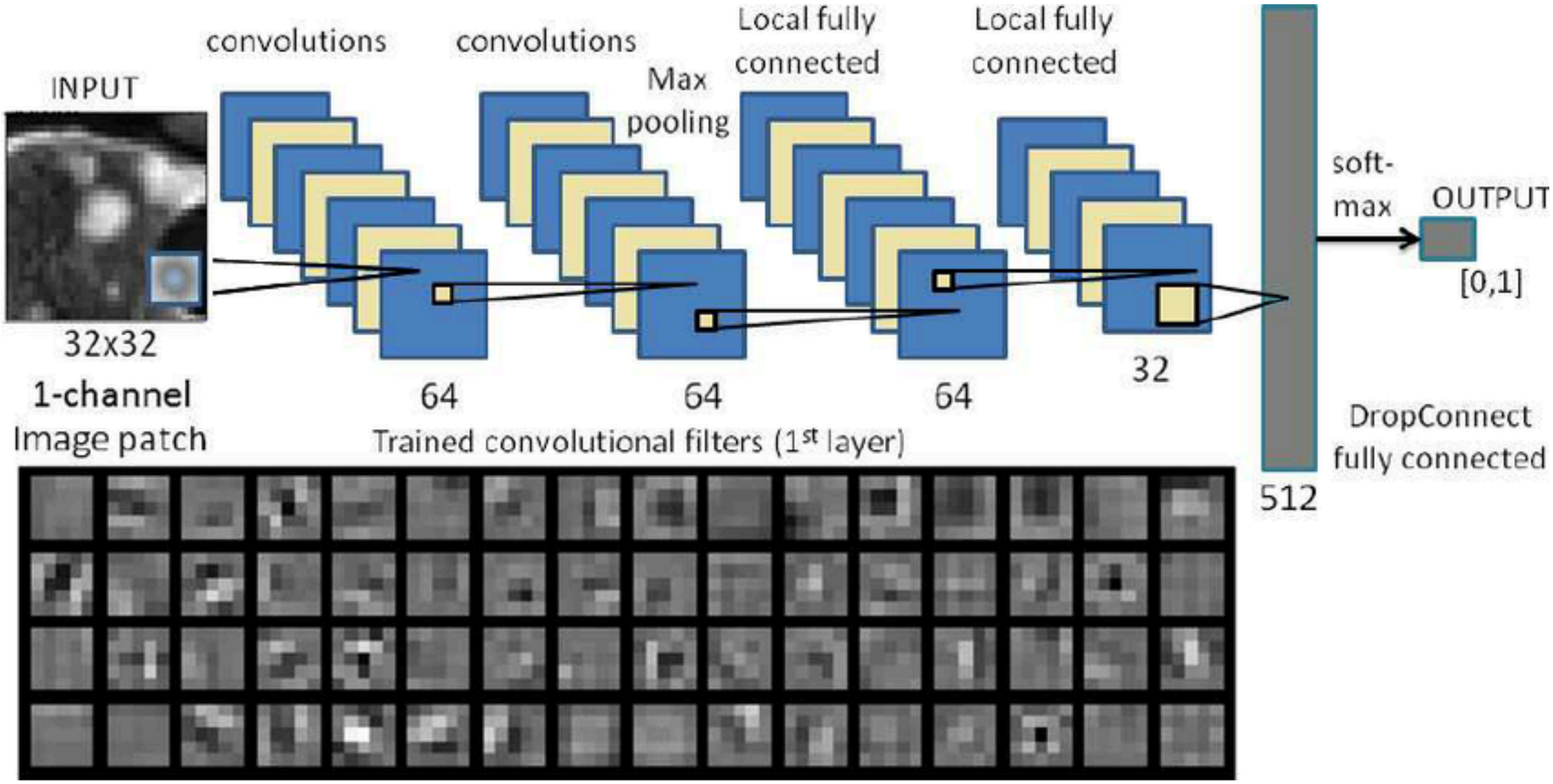
✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗

Russakovsky et al. arXiv, 2014

LeNet CNN


INPUT
32x32
1-channel
Image patch
convolutions
64
convolutions
64
Max
pooling
Local fully
connected
64
Local fully
connected
32
soft-
max
OUTPUT
[0,1]
512
DropConnect
fully connected
Trained convolutional filters (1st layer)

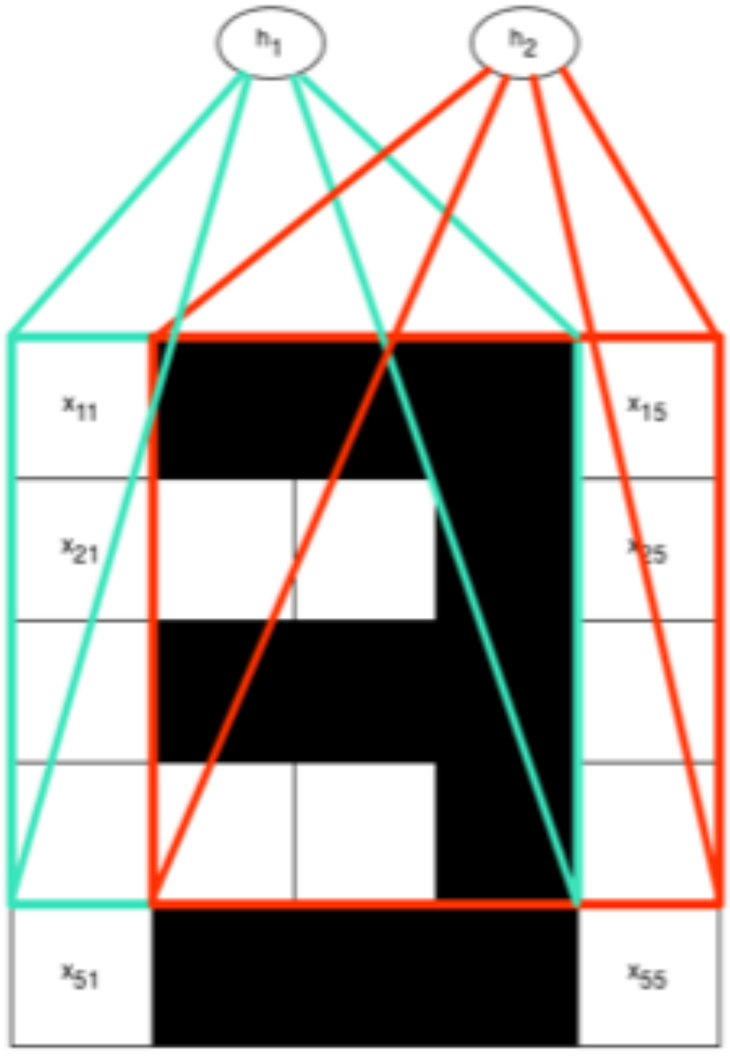
tf.conf2d

- 2D convolution; take a window and slide over the image. The amount sliding to right then down when end is reached is the stride
- The size of the convolution window is kernel/filter
- padding is an extra line of pixels on the edges added if padding is SAME. Valid has no padding. Truncate the multiply. Assume 0.

```
model.add(Convolution2D(32, kernel_size=(5, 5),padding='same',
                        input_shape=(img_rows,img_cols,img_ch),name='conv1'))
model.add(Activation('relu',name='relu1'))
model.add(MaxPooling2D(pool_size=(2,2),strides=1,name='maxpool1'))
model.add(Convolution2D(64, kernel_size=(5, 5),padding='same',
                        name='conv2'))
```

```
    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(patch_size, patch_size, num_color, depth), mean = mu, stddev = s
    conv1_b = tf.Variable(tf.zeros(depth))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b
```

are the same for each of the four hidden units.



Source: Hinton/Coursera

Numerical examples for debugging 1 image

○ $y_1 = 4, y_2 = 8, y_3 = 2, y_4 = 4$

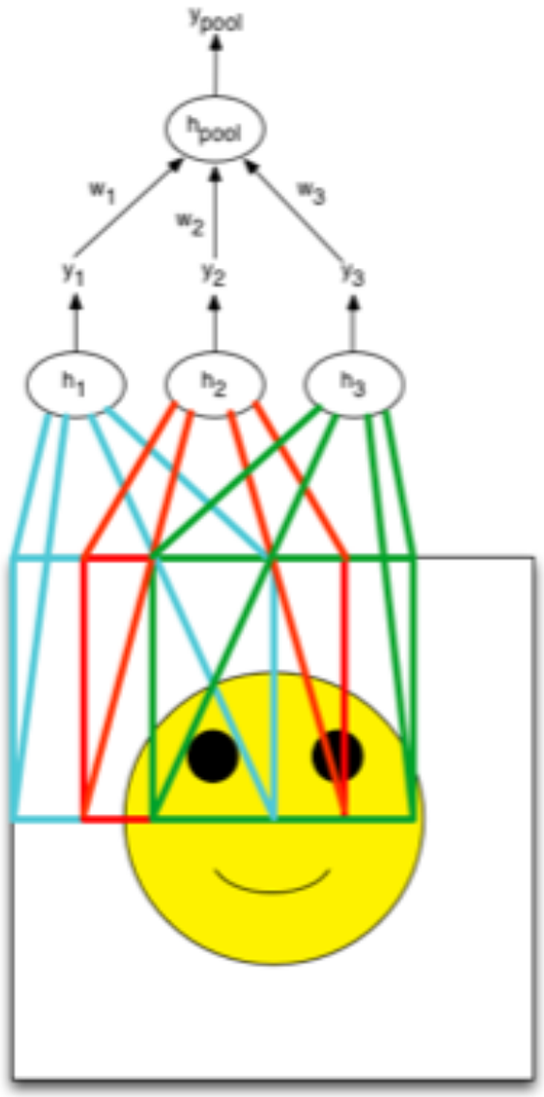
Correct

To get the answer we need to multiply the weight matrix by the pixels that it is assigned to. For hidden unit h_4 this would mean:

$$
\begin{aligned}
y_4 &= w_{11}x_{22} + w_{12}x_{23} + w_{13}x_{24} + w_{14}x_{25} \\
&\quad + w_{21}x_{32} + w_{22}x_{33} + w_{23}x_{34} + w_{24}x_{35} \\
&\quad + w_{31}x_{42} + w_{32}x_{43} + w_{33}x_{44} + w_{34}x_{45} \\
&\quad + w_{41}x_{52} + w_{42}x_{53} + w_{43}x_{54} + w_{44}x_{55} \\
&= (1)(0) + (1)(0) + (1)(1) + (0)(0) \\
&\quad + (0)(1) + (0)(1) + (1)(1) + (0)(0) \\
&\quad + (1)(0) + (1)(0) + (1)(1) + (0)(0) \\
&\quad + (0)(1) + (0)(1) + (1)(1) + (0)(0) \\
&= 4
\end{aligned}
$$

Gap between udacity/coursera/cs231n assignments building application/deployment

Source: Hinton/Coursera



Average pooling is defined as $y_{pool} = \frac{y_1+y_2+y_3}{3}$.

In the image shown to the network, black pixels have a value of 1 while white pixels have a value of 0, so only two pixels in this image have a value of 1. Suppose we wanted to pool the output of each hidden unit using *max pooling*. If $y_1 = 2, y_2 = 0, y_3 = 1, y_4 = 0$ and max-pooling is defined as $y_{pool} = \max_i y_i$, then what will be the value of y_{pool} in this example?

- [ ] 2
- [ ] 0
- [ ] 1
- [ ] 3

Fully connected layer for outputting probability

tf implementation page1/2

```
def LeNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variables for the weights and biases for each layer
    mu = 0
    sigma = 0.1

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(patch_size, patch_size, num_color, depth), mean = mu, stddev = sig
    conv1_b = tf.Variable(tf.zeros(depth))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b

    # SOLUTION: Activation.
    conv1 = tf.nn.relu(conv1)

    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
    #tf.nn.dropout(conv1, .9)


    # SOLUTION: Layer 2: Convolutional. Output = 10x10x16.
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigma))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b

    # SOLUTION: Activation.
    conv2 = tf.nn.relu(conv2)
    #tf.nn.dropout(conv2, .7)


    # SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    flat = tf.contrib.layers.flatten(conv2)

    # SOLUTION: Layer 3: Fully Connected. Input = 400. Output = 120.
    fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 200), mean = mu, stddev = sigma))
    fc1_b = tf.Variable(tf.zeros(200))
    fc1   = tf.matmul(flat, fc1_W) + fc1_b
```

Source: https://github.com/dougc333/UdacityCarStuff/tree/master/CarND-Traffic-Sign-Classifier-Project

tf implementation page2/2

```
# SOLUTION: Activation.
fc1    = tf.nn.relu(fc1)
h_fc1_drop = tf.nn.dropout(fc1, keep_prob)

# SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.
fc2_W  = tf.Variable(tf.truncated_normal(shape=(200, 100), mean = mu, stddev = sigma))
fc2_b  = tf.Variable(tf.zeros(100))
fc2    = tf.matmul(h_fc1_drop, fc2_W) + fc2_b

# SOLUTION: Activation.
fc2    = tf.nn.relu(fc2)
#print ("after relu fc2 shape:", fc2.get_shape())

h_fc2_drop = tf.nn.dropout(fc2, keep_prob)

# SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 43.
fc3_W  = tf.Variable(tf.truncated_normal(shape=(100, 43), mean = mu, stddev = sigma))
fc3_b  = tf.Variable(tf.zeros(43))
logits = tf.matmul(h_fc2_drop, fc3_W) + fc3_b

print ('logits shape:',logits.get_shape())
return logits
```

Source: https://github.com/dougc333/UdacityCarStuff/tree/master/CarND-Traffic-Sign-Classifier-Project

Keras / use for kaggle/practice/replicate papers

```
model = Sequential()
model.add(Convolution2D(32, kernel_size=(5, 5),padding='same',
                        input_shape=(img_rows,img_cols,img_ch),name='conv1'))
model.add(Activation('relu',name='relu1'))
model.add(MaxPooling2D(pool_size=(2,2),strides=1,name='maxpool1'))
model.add(Convolution2D(64, kernel_size=(5, 5),padding='same',
                        name='conv2'))
model.add(Activation('relu',name='relu2'))
model.add(MaxPooling2D(pool_size=(2,2),strides=None,name='maxpool2'))
model.add(Convolution2D(128, kernel_size=(5, 5), padding='same',
                        name='conv3'))
model.add(Activation('relu',name='relu3'))
model.add(MaxPooling2D(pool_size=(2,2),strides=None,name='maxpool3'))
model.add(Flatten(name='flatten'))
model.add(Dropout(0.5,name='dropout1'))
model.add(Dense(128, name='hidden1'))
model.add(Activation('relu',name='relu4'))
model.add(Dropout(0.5,name='dropout2'))
model.add(Dense(128,  name='hidden2'))
model.add(Activation('relu',name='relu5'))
model.add(Dense(nb_classes, name='output'))
model.add(Activation('softmax',name='softmax'))

model.summary()
```

Source: https://github.com/dougc333/UdacityCarStuff/tree/master/CarND-Traffic-Sign-Classifier-Project

Models Iteration 1

OO Models from cs20SI

```
tf.reset_default_graph()

class Model:
    def __init__(self, mnist):
        self.mnist = mnist
        self.learning_rate = .1
    def create_placeholders(self):
        self.x = tf.placeholder(tf.float32,shape = [None,784],name="x")
        self.y = tf.placeholder(tf.float32,shape = [None,10],name="y")
    def create_embeddings(self):
        self.W = tf.Variable(tf.zeros([784,10]),name="W")
        self.b = tf.Variable(tf.zeros([10]),name="b")
        self.output = tf.nn.softmax(tf.add(tf.matmul(self.x,self.W), self.b),name="output")
    def create_loss(self):
        self.loss = tf.reduce_mean(-tf.reduce_sum(self.y * tf.log(self.output), reduction_i
    def create_optimizer(self):
        self.optimizer = tf.train.GradientDescentOptimizer(self.learning_rate,name="optimize
    def train(self):
        global_step = tf.Variable(0)
        self.sess = tf.Session()
        init = tf.global_variables_initializer()
        self.sess.run(init)
        for x in range(100):
            batch_xs, batch_ys = mnist.train.next_batch(100)
            self.sess.run(self.optimizer, feed_dict={self.x: batch_xs, self.y: batch_ys})
    def test(self):
        correct_prediction = tf.equal(tf.argmax(self.output,1), tf.argmax(self.y,1),name="ec
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),name="accuracy")
        print(self.sess.run(accuracy, feed_dict={self.x: self.mnist.test.images, self.y: sel
```

```
mnist = input_data.read_data_sets(".", one_hot=True)
model = Model(mnist)
model.create_placeholders()
model.create_embeddings()
model.create_loss()
model.create_optimizer()
model.train()
model.test()
```

Source: https://github.com/dougc333/DeepLearning

TF tips:

After building hundreds of models here are some problems I still have:

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
      sess.run(init)
```

Error message won't say no init. Will seem like you have an unrelated problem

```
init_ab = tf.variables_initializer([a, b], name="init_ab")
with tf.Session() as sess:
      sess.run(init_ab)
```

```
W = tf.Variable(tf.zeros([784,10]))
with tf.Session() as sess:
      sess.run(W.initializer)
```

debug adding new variable

Can't print variable w/o session run()

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
      sess.run(init)
```

Source: Stanford cs20si slides

Unlike normal programming languages Printing a variable is hard

```
W=tf.Variable(tf.truncated_normal[700,10]))
print(W)
```

no init/
won't see: no session/no eval

```
with tf.Session() as sess:
sess.run(init)
print(W)
Tensor("Variable/read:100", shape(700,10))
```

no var name

there is global init

```
with tf.Session() as sess:
sess.run(init)
print(W.eval())
Tensor("Variable/read:100", shape(700,10))
```

I want to change the variable value while debugging someone's code

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
    sess.run(W.initializer)
    print W.eval() # >> 10
```

no output?

```
W = tf.Variable(10)
assign_op = W.assign(100)
with tf.Session() as sess:
    sess.run(W.initializer)
    sess.run(assign_op)
print W.eval() # >> 100
```

create an assign op first then run the op

assign the assign!!!!

delayed execution

not in the web pages/might be in slack now

Source: Stanford cs20si slides

tensorflow graphs when debugging/order is not guaranteed

A+B+C = (A+B) +C or A + (B+C)

with g.control_dependencies([a,b,c]):

easy to figure out when you get to it..
a reminder this is a lazy eval graph language/
not like a normal programming language

things take longer with TF
the old versions are stable in production

Source: Stanford cs20si slides

Lazy graph problems

in a training loop/want to change variable

make op an add to keep simple

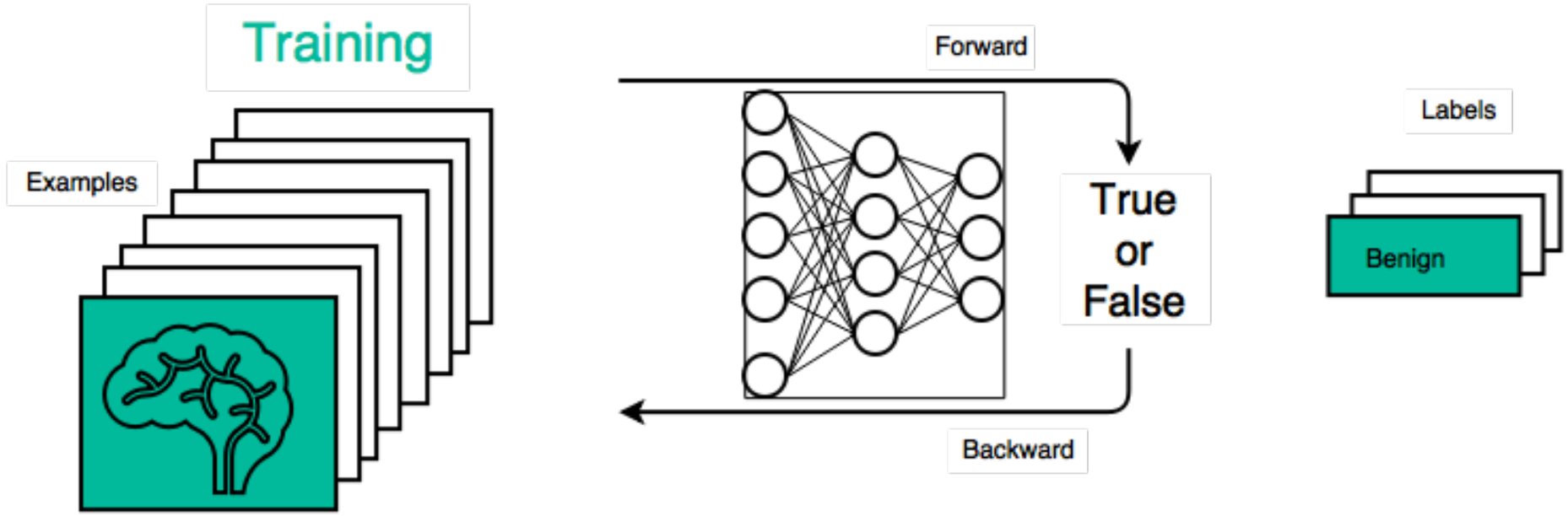
in tf/second example creates 10 adds common error/works but graph too large

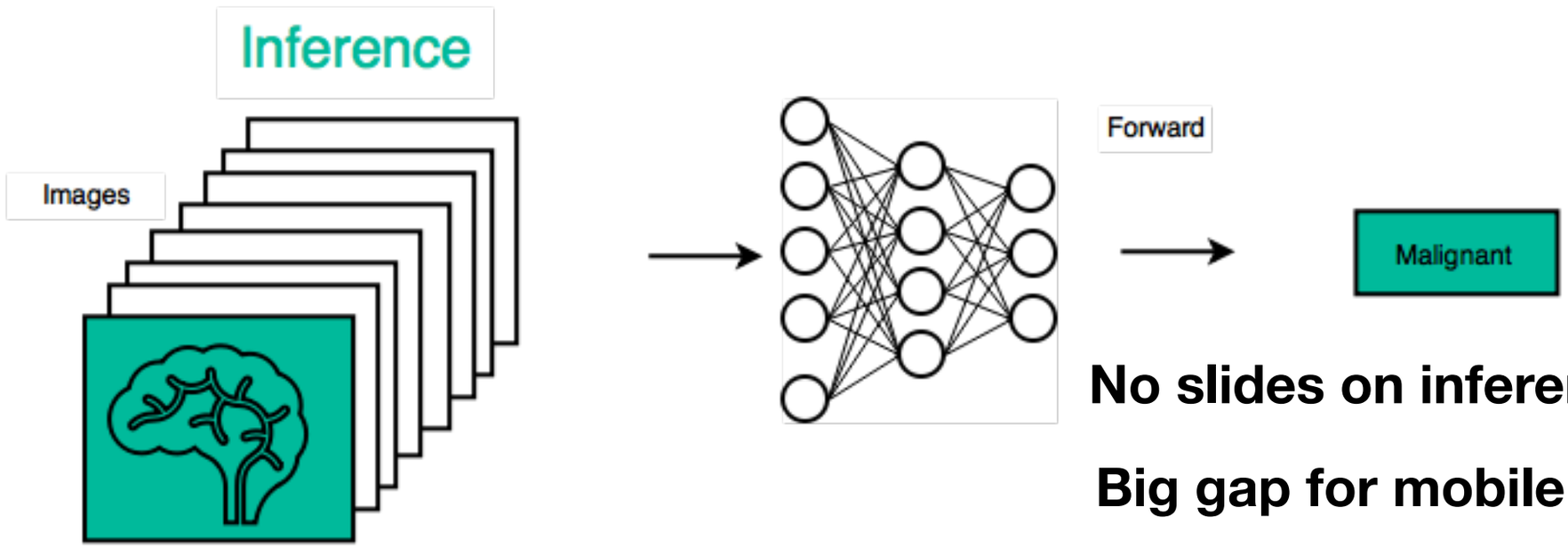
```
x = tf.Variable(10, name='x')
y = tf.Variable(20, name='y')
z = tf.add(x, y) # you create the node for add node before executing the graph

with tf.Session() as sess:
      sess.run(tf.global_variables_initializer())
      writer = tf.summary.FileWriter('./my_graph/l2', sess.graph)
      for _ in range(10):
            sess.run(z)
      writer.close()
```

```
x = tf.Variable(10, name='x')
y = tf.Variable(20, name='y')

with tf.Session() as sess:
      sess.run(tf.global_variables_initializer())
      writer = tf.summary.FileWriter('./my_graph/l2', sess.graph)
      for _ in range(10):
            sess.run(tf.add(x, y)) # someone decides to be clever to save one line of code
      writer.close()
```

Difference between Models/ Inference

Poll: mobile/targeting speeds





Poll, inference speed for mobile

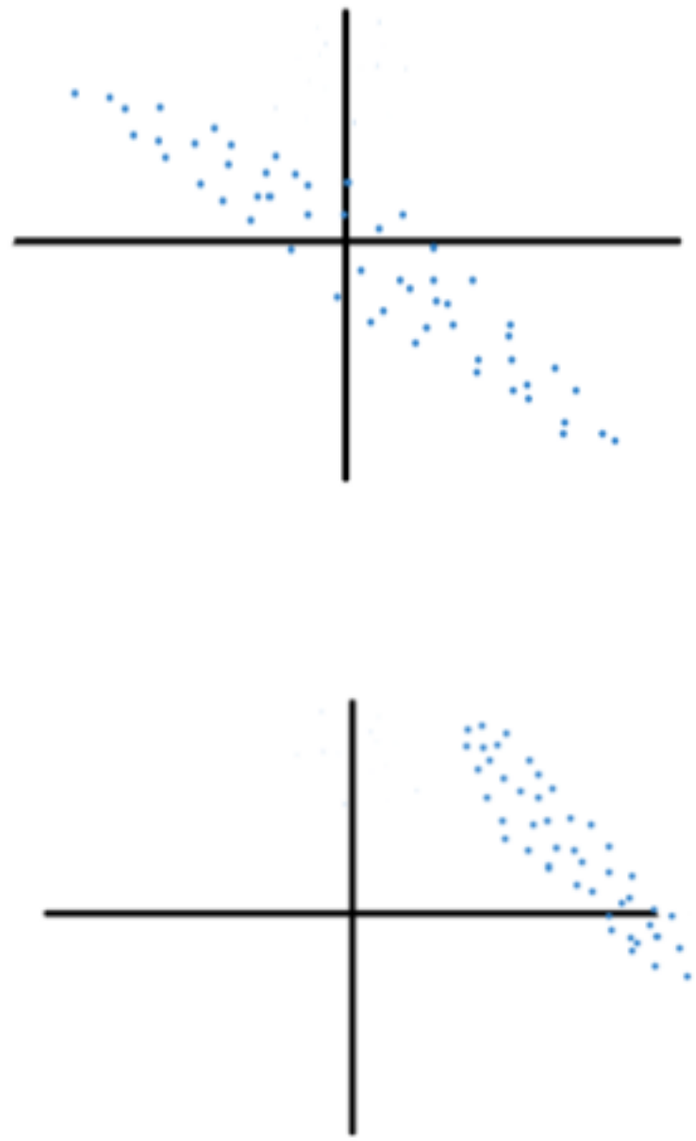
No slides on inference speed.

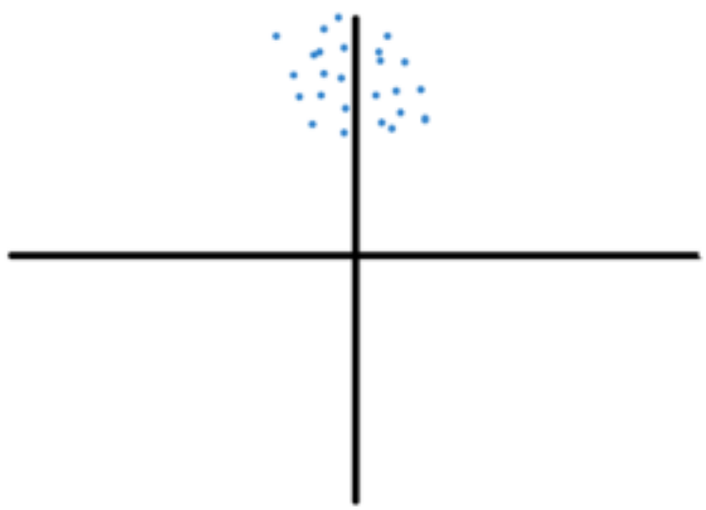
Big gap for mobile deployment

Modeling tips next

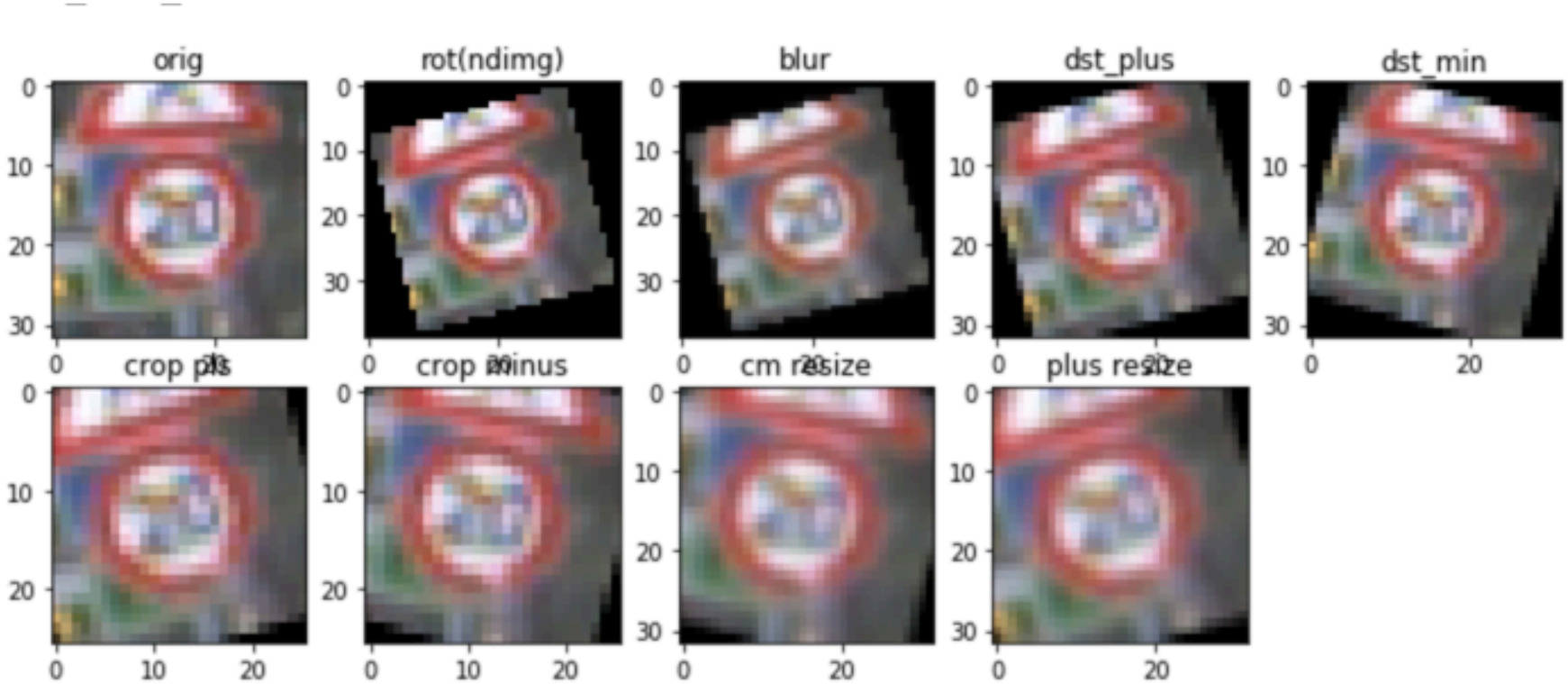
Prep for training

vs. batch normalization/gradient clipping





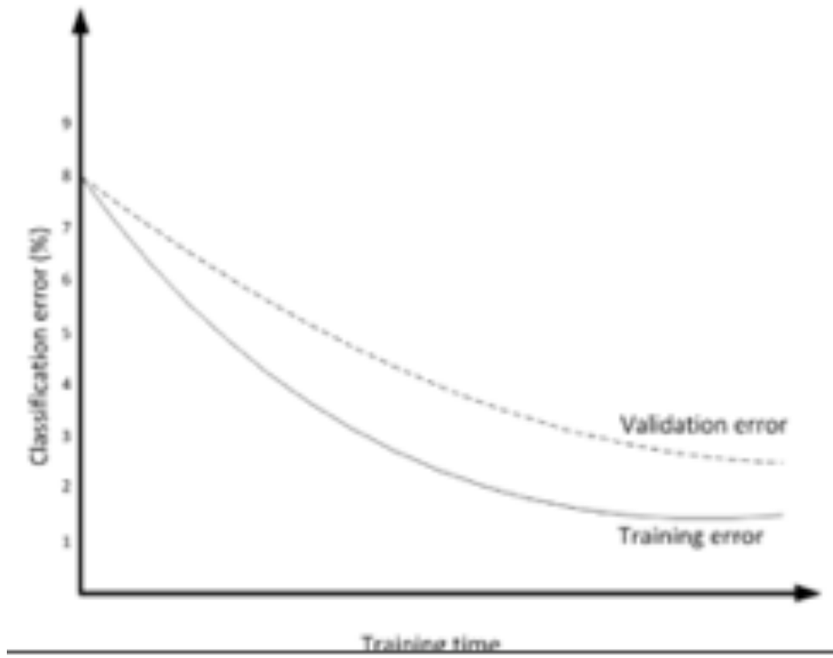
Augmentation



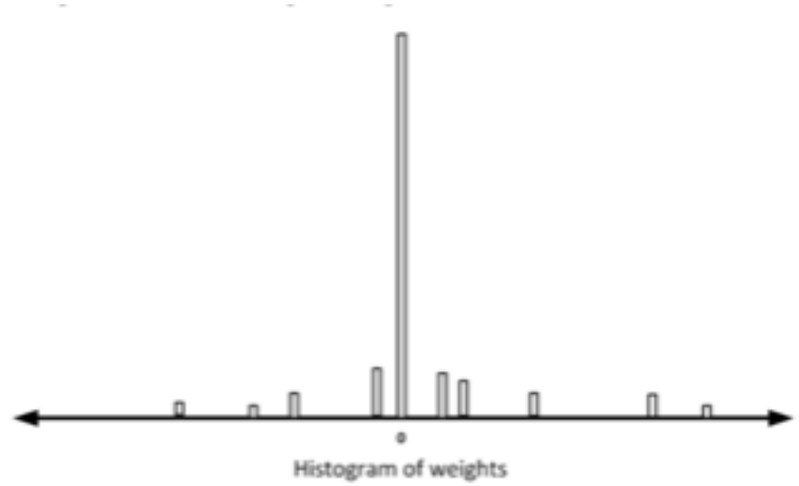
source: https://github.com/dougc333/DeepLearning

Overfitting/early stop

NN overfit



Weight histograms



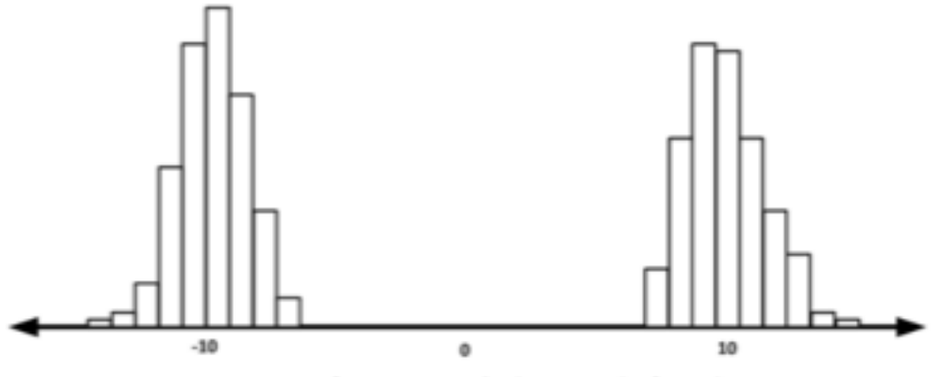
Histogram of weights

L2 regularization, get rid of large weights, center around origin

L1 regularization: center around 0

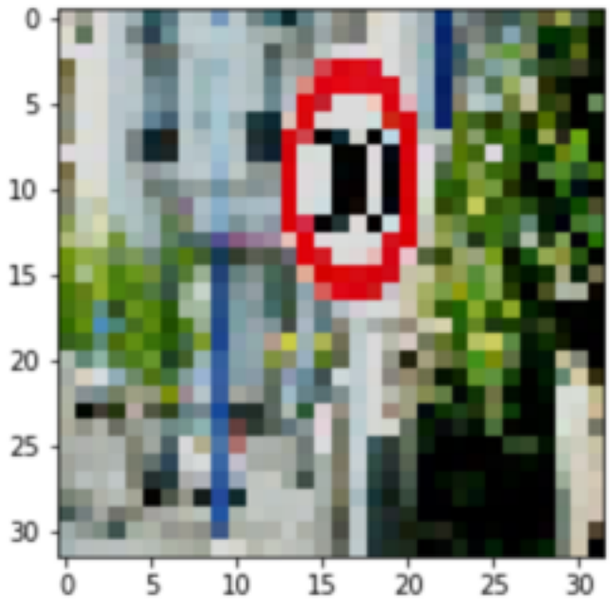
Additive noise: stops learning early

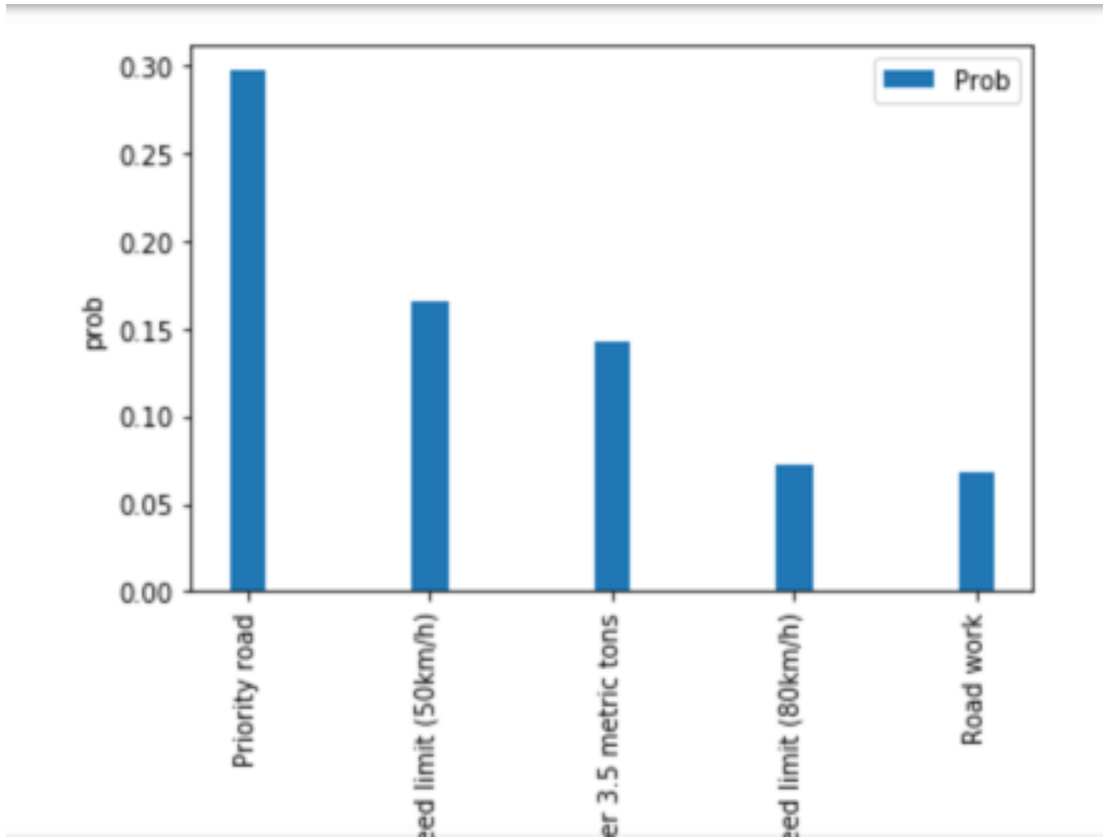
multiplicative noise, helps to push to origin/mean



Histogram of activations (values just before the logistic nonlinearity)

Generalization

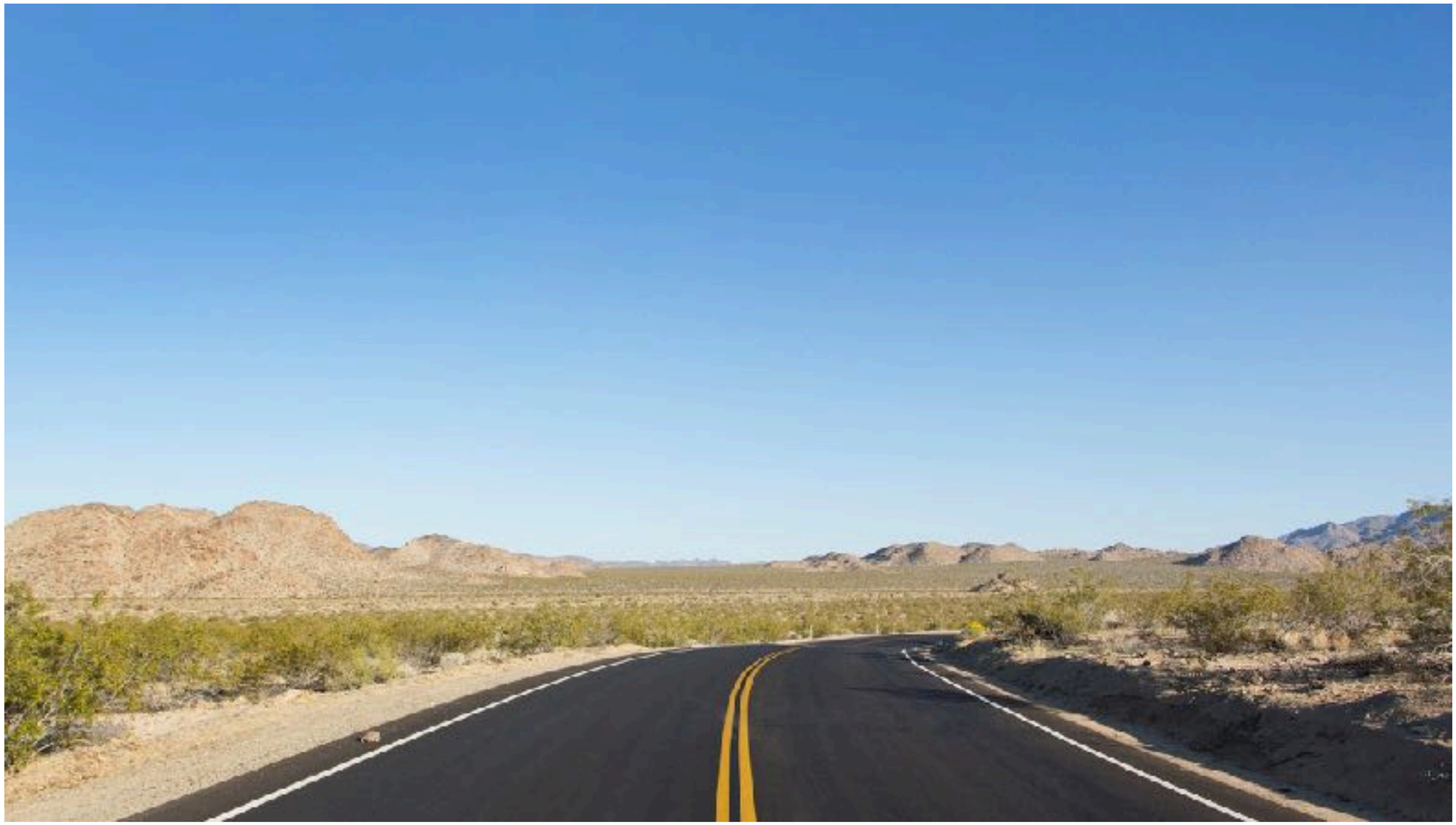




Medical Imaging

Can replicate the turtle ~ tumor

Need higher resolutions



Debug a higher resolution cnn > 224x224 input image size

Poll: who has/is working on higher resolution cnn?

Muffin or Chihuahua

fix w/PCA

Key to DL - Backprop

- leave out derivatives….
- none of the online classes emphasize this; matching manual gradient calculations to code.
    - need this skill to debug larger/deeper networks. Builds intuition on how networks fail/recover.
    - Tradeoff between batch normalization/L1/L2 regularization/adding noise/ reducing overfitting
    - Overfitting necessary for failed nodes during SGD. Sampling process not uniform.
    - practice derivatives/chain rule — implemented by sw packages tensorflow/**pytorch**/~~theano~~.

Derivation of backdrop

Key to Deep Learning

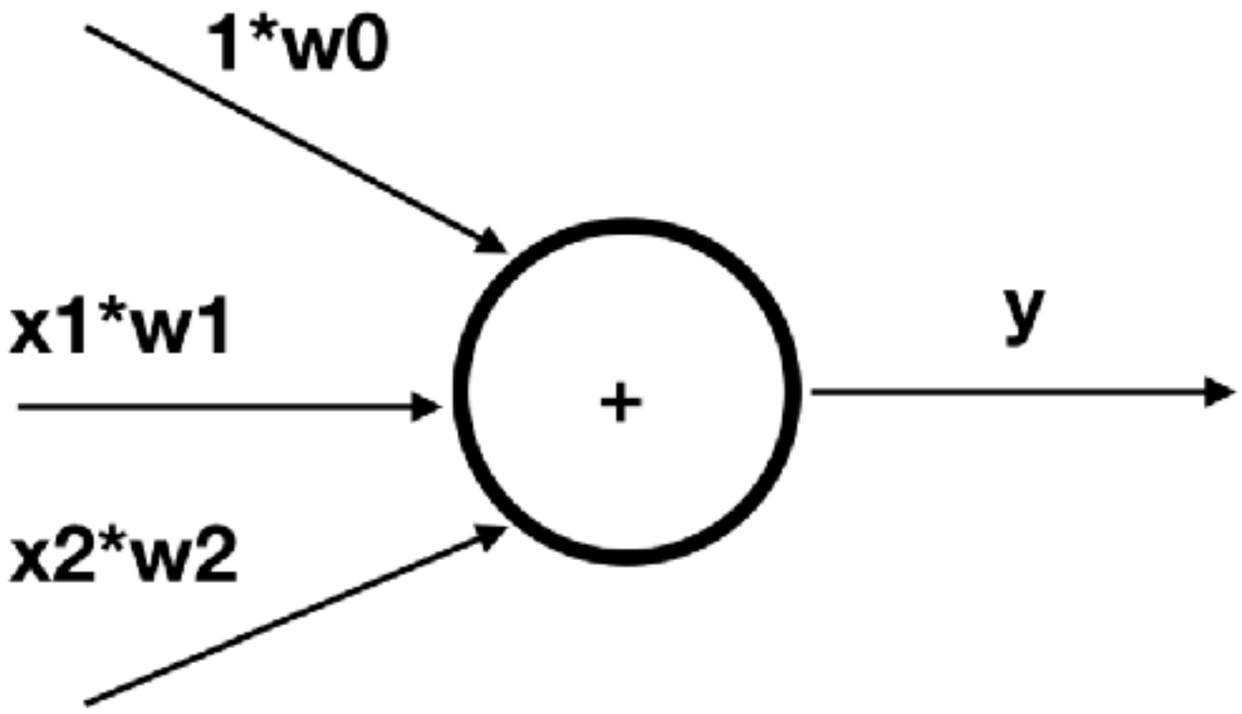
$$\frac{dy}{dx} = \frac{dg}{dx} = \frac{dg}{df}\ \frac{df}{dx}$$

Need this to replicate papers/debug networks

Derivation of backdrop — Linear function

Linear Neuron

Notation



$$z = \sum_{i=1}^{N} w_i x_i + b = \sum_{i=0}^{N} w_i x_i$$

Linear Result

step. $E = \frac{1}{2} \sum_{ninitraining} (t_n - y_n)^2$

verbatim Hinton Courser lecture3.pdf

or the total error over the entire training set of size N is: $E_{total} = \frac{1}{2} \sum_{i=0}^{N} (t_i - y_i)^2$

We can also define the error for a single training sample: $E_i = \frac{1}{2}(t_i - y_i)^2$

- Take the derivative of the total sum error loss over the entire training set

We can take the derivative of either the total loss or each step. It does not matter as long as we are consistent with the notation and clear which option we take.

- Chain rule (copied verbatim from Hinton Coursera lecture3.pdf)

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial E}{\partial y^n} \cdot \frac{\partial y^n}{\partial w_i}$$

We can calculate the 2 above parts separately

$$\frac{\partial E}{\partial y_n} = \frac{1}{2}(2)(t_n - y_n)(-1) = -(t_n - y_n)$$

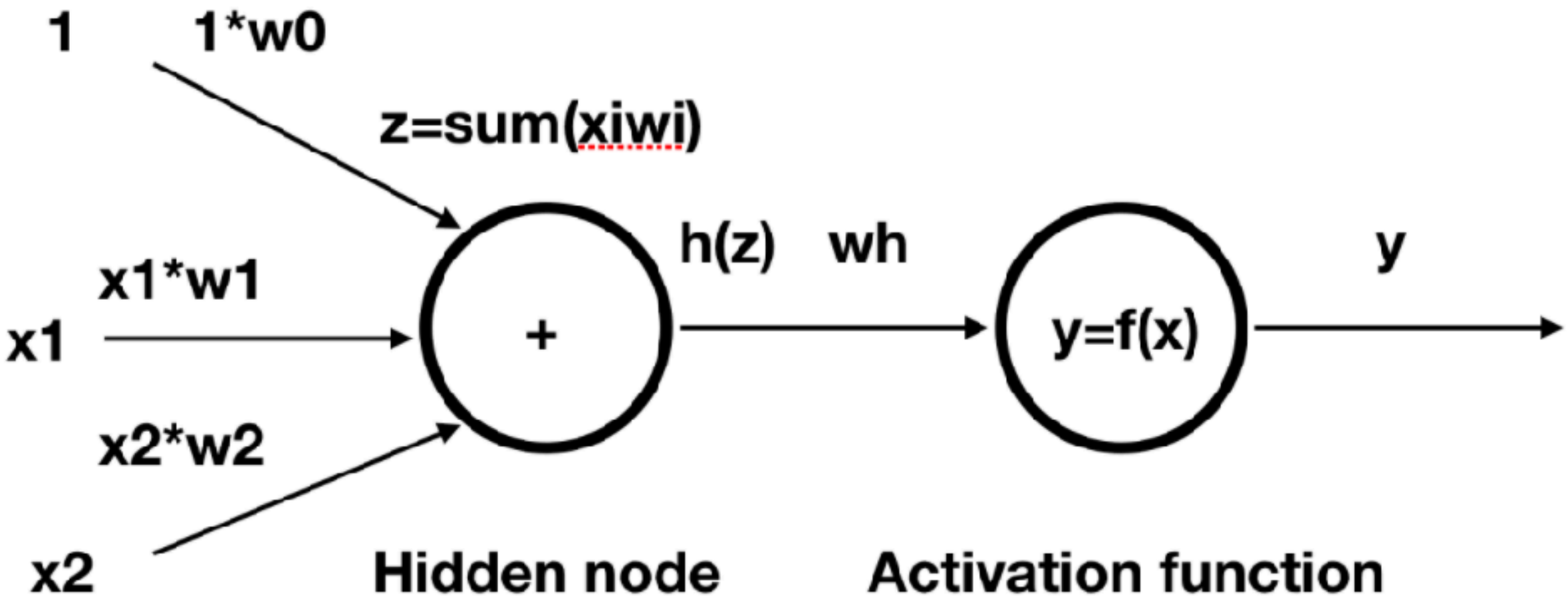
The above is Hinton notation. Many use the opposite with the training samples t_n reversed from the training set labels y_n. Will not matter can adjust when defining delta rule

$$\frac{\partial E}{\partial y_n} = -(t_n - y_n)$$

$$\frac{\partial y}{\partial w_i} = x_i$$

$$\frac{\partial E}{\partial w_i} = -x_i(t_n - y_n)$$

The delta rule takes the above and multiplies a learning rate, epsilon to the expression: $\delta w_i = w_i + \epsilon x_i(t_n - y_n)$


1
1*w0
z=sum(xiwi)
x1*w1
x1
+
h(z)
wh
y=f(x)
y
x2*w2
x2
Hidden node
Activation function

This is not the Udacity hw1 neuron. This is a simplified example with multiple inputs and input weights, 1 hidden node, one hidden node output and one output node y. This reduces the summations to only the ones at the input node to keep things simple.
This adds an additional set of weights w_h

$$E = \frac{1}{2}(t_n - y_n)^2$$

$$y = f(x)$$

This is crappy Notation practice from Udacity. They should leave the y=f(x) equation out as it is confusing and obvious. It does not need to be written down because it is confusing why it is a constructive statement and is not needed. It is not part of the equations used in the chain rule.

$$y = x$$

$$x = h(z) \cdot w_h$$

$$z = \sum_{i=0}^{N} w_i x_i$$

$$\frac{\partial E}{\partial w_h} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial x} \cdot \frac{\partial x}{\partial w_h} = -(t_n - y_n)(1)h(z)$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial x} \cdot \frac{\partial x}{\partial h} \cdot \frac{\partial h}{\partial w_i}$$

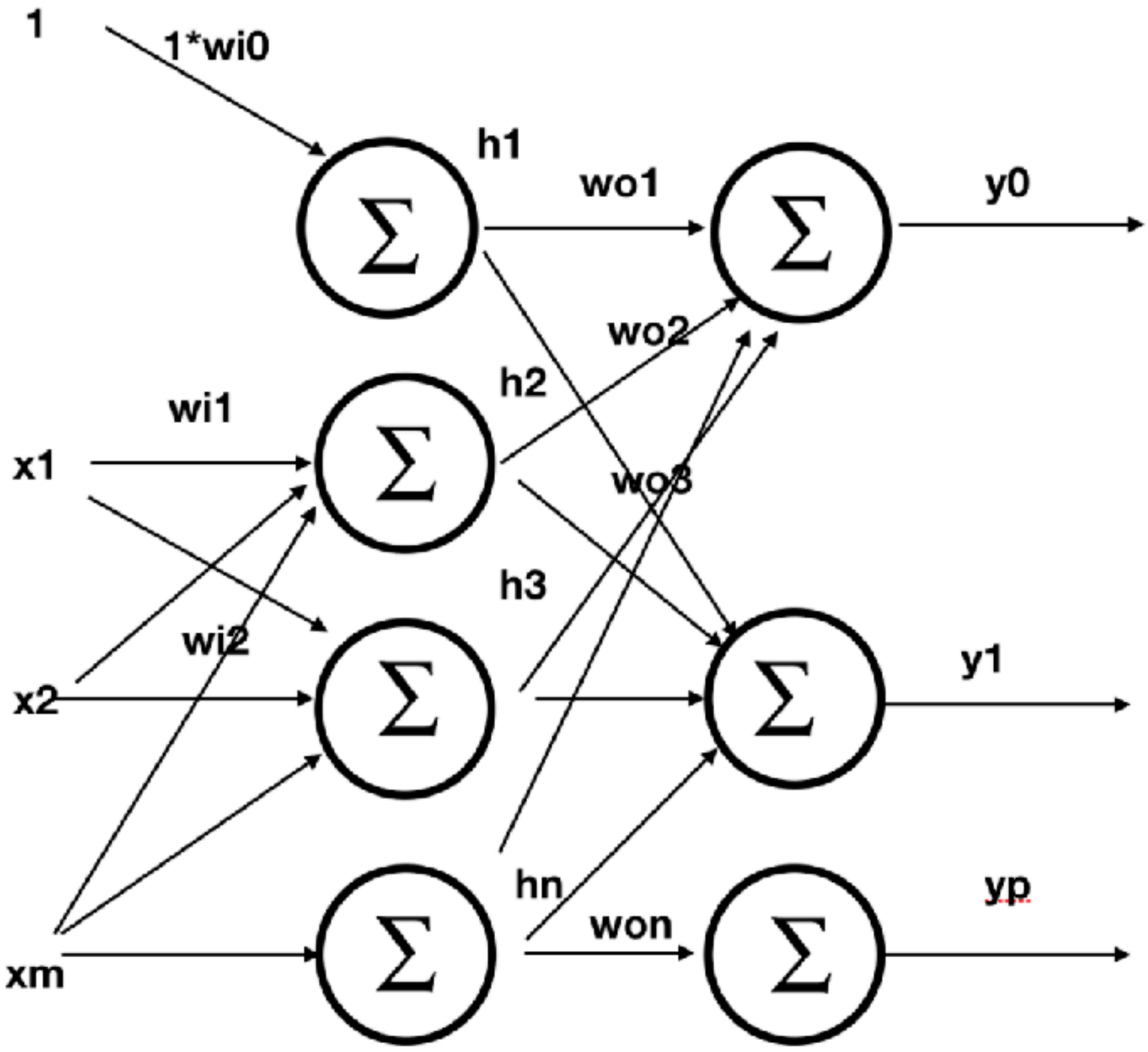
$$\frac{\partial y}{\partial x} = 1$$

$$\frac{\partial x}{\partial h} = w_h$$

$$\frac{\partial h}{\partial w_i} = x_i$$

$$\frac{\partial E}{\partial w_i} = -(t_n - y_n)(1) \cdot w_h \cdot x_i$$

Feedforward Network



To generalize for the case where there are n inputs, m hidden nodes and p outputs, we add summations to include the general case for a fully connected hidden layer with 2 linear neurons.

$$h(z) = \sum_{m=0}^{M} w_{im} x_{im}$$

$$y = x$$

$$x = \sum_{n=0}^{N} w_{on} h_{on}$$

$$E = \frac{1}{2}(t_n - y_n)^2$$

$$\frac{\partial E}{\partial w_{oh}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w_{on}}$$

$$\frac{\partial E}{\partial y} = -\sum_{m=0}^{M} (t_m - y_m)$$

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

$$\frac{\partial x}{\partial w_{on}} = \sum_{n=0}^{N} h_{on}$$

$$\frac{\partial E}{\partial w_{im}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial h} \frac{\partial h}{\partial w_{in}}$$

$$\frac{\partial x}{\partial h} = w_{on}$$

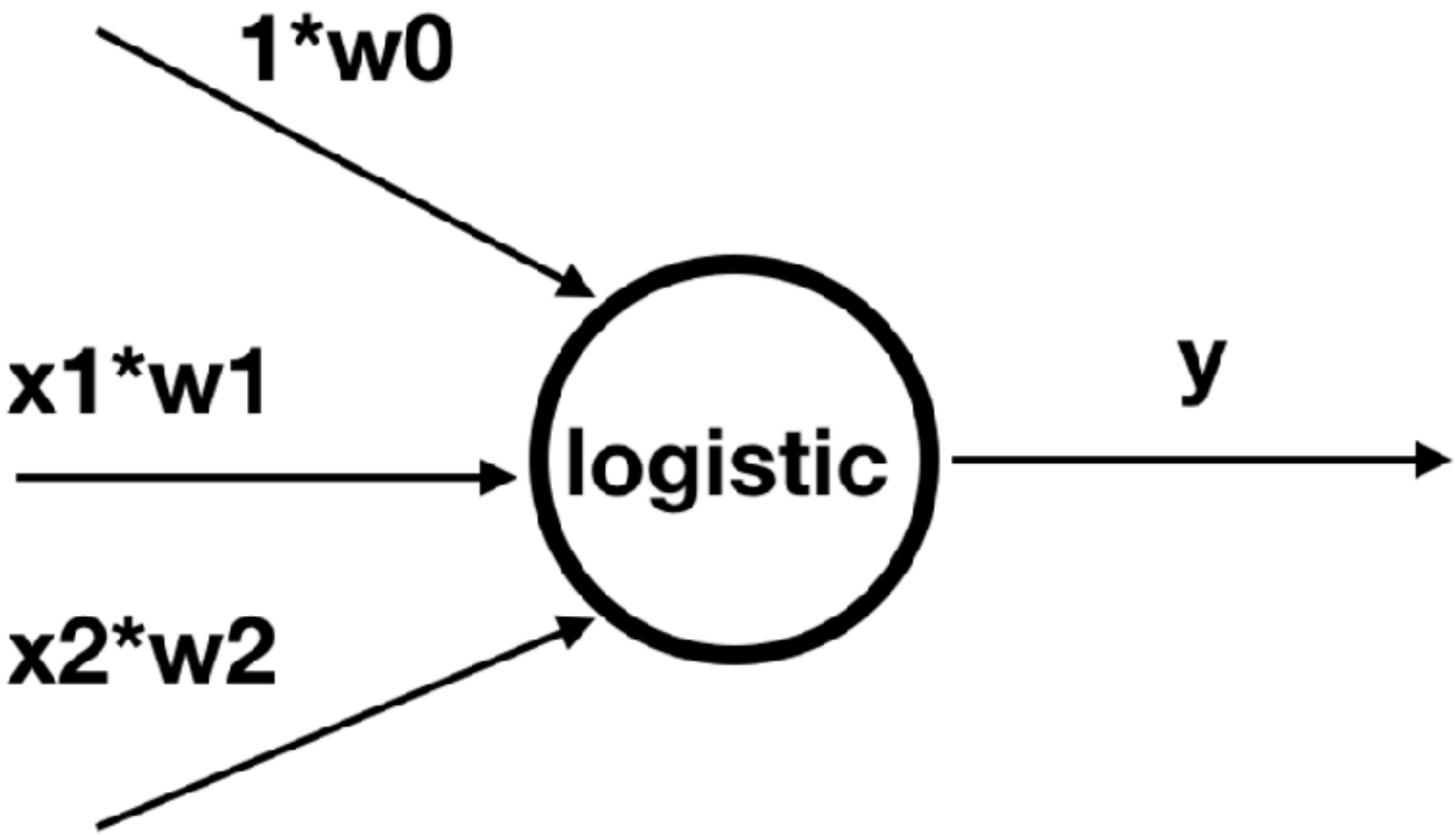
$$\frac{\partial h}{\partial w_{im}} = x_{in}$$

$$\frac{\partial E}{\partial w_{im}} = -(t_n - y_n) \cdot 1 \cdot w_{on} \cdot x_{in}$$

Logistic Function

Logistic function

A logistic neuron requires a set of weights and an Activation function at the output of y. Without an activation function this is not a neuron. The AF is not in the figure hence this is not a neuron and we first do back propagation with the logistic function first. Backpropagation refers to the method of determining the weights for a given training sample using gradient descent to compute the least loss or error with a set of weights per training sample.



Logistic Result

From the Hinton Notation the output of the logistic is y: $\sigma(z) = \frac{1}{1+exp(-z)}$ and the input to the logistic is z. For the diagram above we use $y = \frac{1}{1+exp(-z)}$ The input to the logistic is a summation of 2 terms, the bias b and the product of the input with the weight. $z = \sum_{i=1}^{N} x_i w_i + b_i$

From the Ng Notation the output of the logistic neuron is y and the bias terms are combined with the weight terms in θ : $y = \frac{1}{1+exp(-\theta_i \cdot x_i)}$

We will use the Hinton notation and start the index term from 0 to avoid writing a separate bias term. For the a single logistic function with 1 weight, multiple inputs and 1 output: We want to calculate the derivative of the output y as a function of the weights w_i. We use the derivative when set to 0 to determine the amount to change the state of the weights as we iterate using the training sets towards a converged value.

$$\frac{\partial y}{\partial w} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$$

The derivative of the logistic WRT z. $\frac{dy}{dz} = y(1-y)$

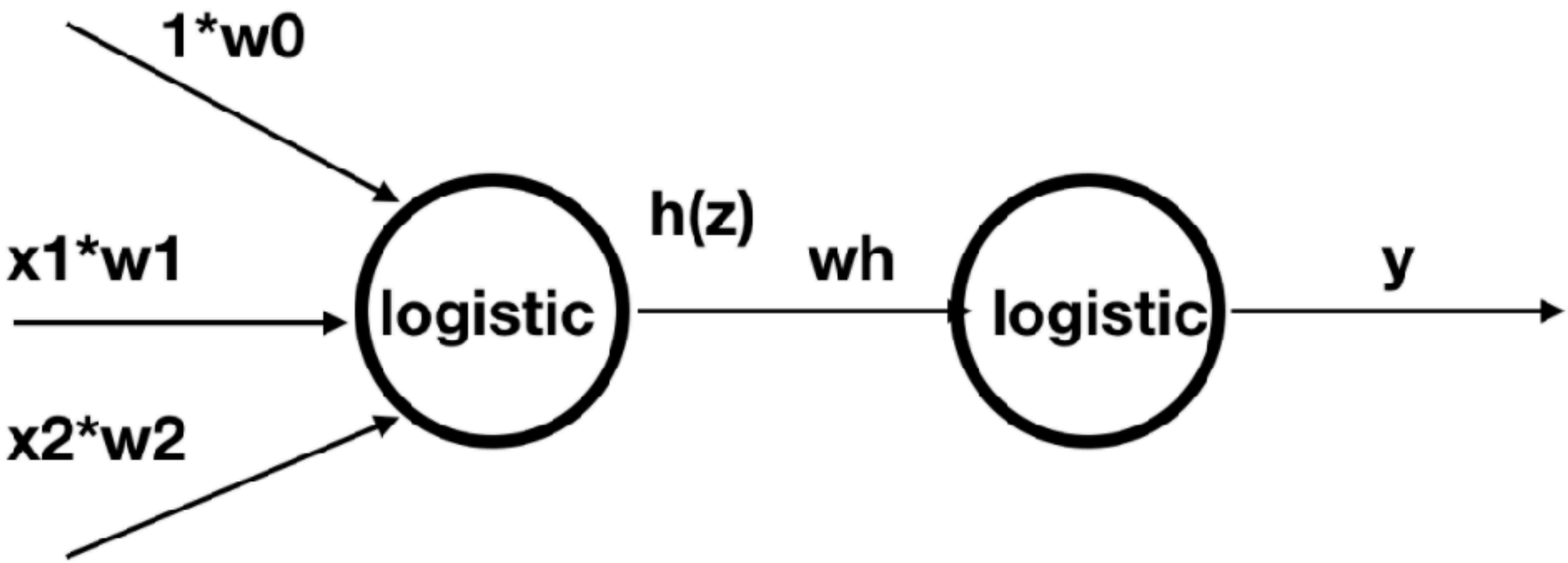
$$\frac{dz}{dw} = x_i$$

$$\frac{\partial y}{\partial w} = x_i \cdot y(1-y)$$

To derive the delta rule, add a multiply by the learning rate and add it to the existing weights $\delta w = w_i + \epsilon \cdot x_i \cdot y(1-y)$

Logistic Layer+ Logistic Activation

To mirror the linear derivation we add an activation function to the output. In this case we add a logistic activation function as seen below.



Logistic/Logistic Result

Add a hidden layer h(z) where $h(z) = \frac{1}{1+\exp[-z]}$ There are 2 logistic functions, the other one is at the output of the activation function, $y(h(z)) = \frac{1}{1+\exp[-h(z)]}$

$$z = b + \sum_{i=1}^{N} x_i w_i = \sum_{i=0}^{N} x_i w_i$$

Most use the notation start the index from 0 and do not bother to write out the b term. This is consistent with tensor notation used in statistical physics. There are 2 weights we need the update rule for, w_{ji} and w_i $\frac{\partial E}{\partial w_{ji}}$

$\frac{\partial E}{\partial w_i}$

Assign a temporary variable to the output of the hidden layer which is also teh input to the activation function, variable a

$$a = b + \sum_{i=1}^{M} h_i w_{ji} = \sum_{i=0}^{M} x_i w_{ji}$$

where M is the number of hidden nodes.

Using the chain rule, we expand the above to:

$\frac{\partial E}{\partial w_{ji}} = \frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$

$\frac{\partial a}{\partial w_{ji}} = \sum_{i=0}^{M} h_i$

$\frac{\partial E}{\partial b_n} = -(t_n - y_n)$ where n is over the training set data.

$\frac{\partial E}{\partial w_{ji}} = -(t_n - y_n) \cdot h_i$

$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial w_{ji}}$

$\frac{\partial E}{\partial y} = -(t_n - y_n)$

$\frac{\partial y}{\partial a} = y(1-y)$

$\frac{\partial a}{\partial h_i} = w_{ji}$

Add a temporary variable z, $z = \sum_{i=0}^{I} w_i x_i$

$h(z) = \frac{1}{1+exp(-z)}$ and

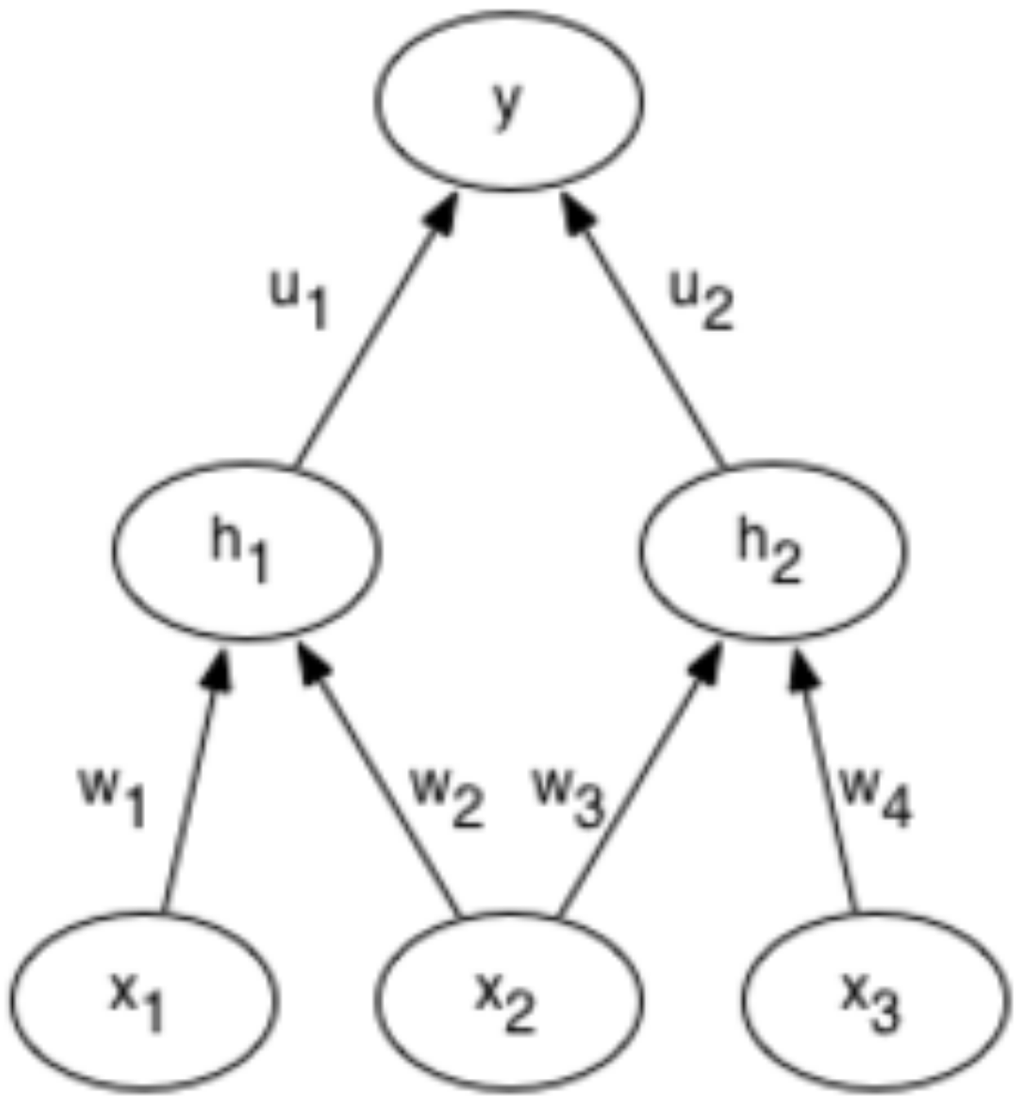
$\frac{\partial h}{\partial z} = h(1-h)$

$\frac{\partial z}{\partial w_{ji}} = x_i$

$weightupdate = e \cdot -(t_n - y_n) \cdot x_i \cdot y_j(1-y_j) \cdot h_i(1-h_i) \cdot w_{ji}$

Derivation of backdrop

Shared Weights **Increases generality and reduces computation**



Not just for CNNS

Derivation of backdrop

$$
\begin{aligned}
z_1 &= w_1 x_1 + w_2 x_2 \\
z_2 &= w_3 x_2 + w_4 x_3 \\
h_1 &= \sigma(z_1) \\
h_2 &= \sigma(z_2) \\
y &= u_1 h_1 + u_2 h_2 \\
E &= \frac{1}{2}(t - y)^2
\end{aligned}
$$

$$
\sigma(x) = \frac{1}{1+\exp(-x)}
$$

Derivation of backdrop

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial w_2}$$

$$\frac{\partial E}{\partial y} = -(t-y)$$

$$\frac{\partial y}{\partial h_1} = u_1$$

$$\frac{\partial h_1}{\partial z_1} = h_1(1-h_1)$$

$$\frac{\partial z_1}{\partial w_2} = x_2$$

So $\frac{\partial E}{\partial w_2} = -(t-y)u_1h_1(1-h_1)x_2$

We can similarly compute $\frac{\partial E}{\partial w_1} = -(t-y)u_1h_1(1-h_1)x_1$

To compute the derivative with tied weights, we now just need to add the derivatives:

$$\begin{aligned}\frac{\partial E}{\partial w_{\text{tied}}} &= \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \\ &= -(t-y)u_1h_1(1-h_1)x_1 - (t-y)u_1h_1(1-h_1)x_2 \\ &= -(t-y)(u_1h_1(1-h_1))(x_1+x_2)\end{aligned}$$

How did we know which sequence of derivatives we would need for backpropagation? One answer is to look at the equations, the other is to look at the picture. Take w_2 for example, and draw a path up the tree to the root. Notice how this path goes through h_1 and not h_2?

Derivation of backdrop

$$\frac{\partial E}{\partial w_{\text{tied}}} = -(t - y)u_1 h_1(1 - h_1)(x_1 + x_2)$$

Structured data with shared weights

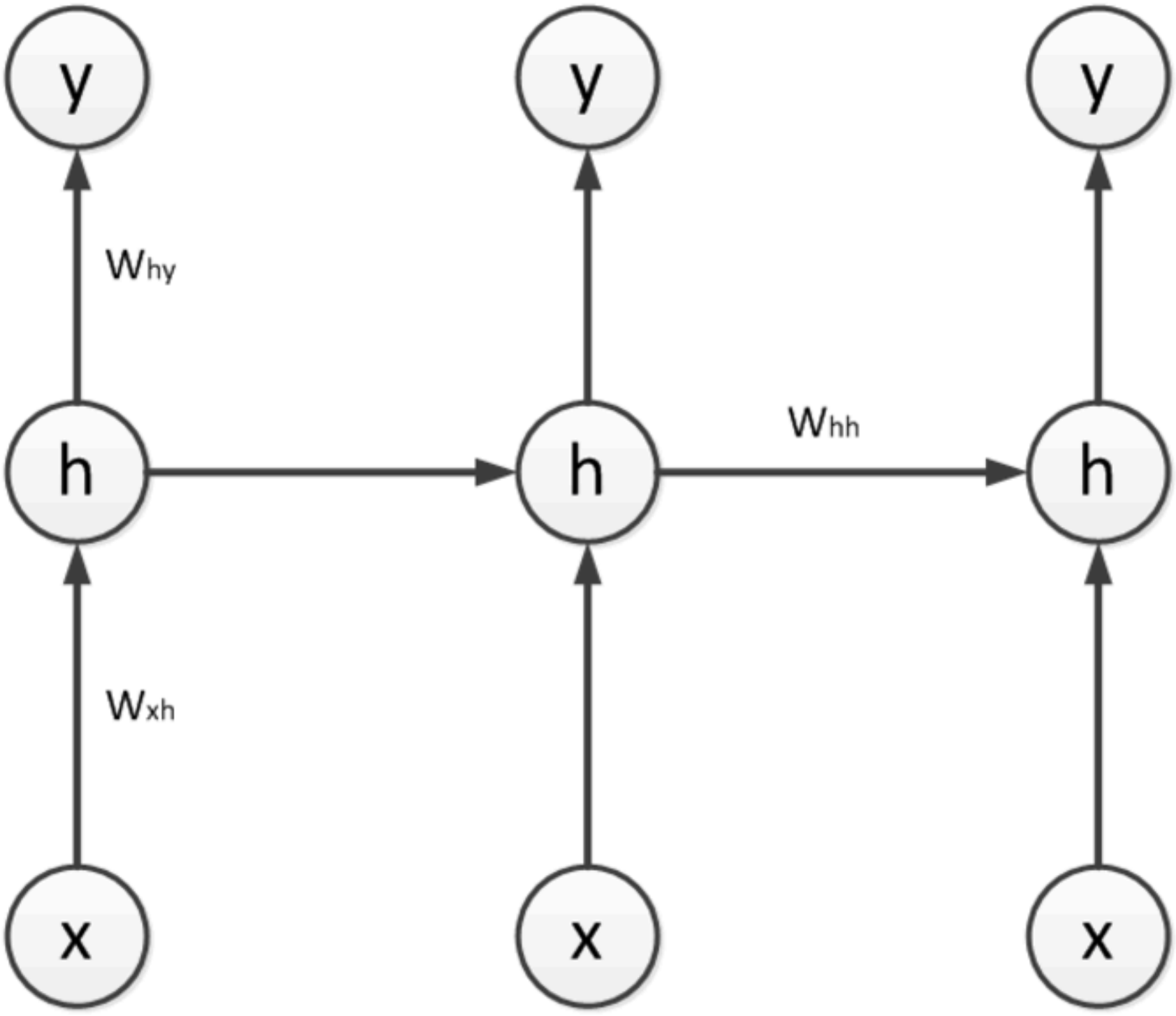
Accuracy comparisons

- **graphs w/# hidden neurons height, width**
- **accuracy w/hidden weights**
- **computation time savings w/hidden weights**

Keras may not be best tool for Architecture experiments

Do the work and have your own opinion

Derivation of backdrop



Derivation of backdrop

$\frac{\partial E}{\partial z_1} = \frac{\partial E_0}{\partial z_1} + \frac{\partial E_1}{\partial z_1} + \frac{\partial E_2}{\partial z_1}$ partial E0 wrt Z1 is 0 bc it is behind in time to z1 so it should not contain any dependent terms

$\frac{\partial E_1}{\partial z_1} = \frac{\partial E_1}{\partial y_1}\frac{\partial y_1}{\partial z_1} = \frac{\partial E_1}{\partial y_1}\frac{\partial y_1}{\partial h_1}\frac{\partial h_1}{\partial z_1}$

$\frac{\partial E_1}{\partial y_1} = -(t_1 - y_1) = -(-0.1 - .1) = -0.2$ $\frac{\partial y_1}{\partial h_1} = W_{hy} = 0.25$ $\frac{\partial h_1}{\partial z_1} = h_1(1 - h_1) = .4(1 - 0.4) = -.24$

$\frac{\partial E}{\partial z_1} = -.2 * .25 * -.24 = .012$

$\frac{\partial E_2}{\partial z_1} = \frac{\partial E_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial h_1}\frac{\partial h_1}{\partial z_1}$

$\frac{\partial E_2}{\partial y_2} = -(t_2 - y_2) = -(-0.2 - .2) = .4$ $\frac{\partial y_2}{\partial h_2} = W_{hy} = .25$ $\frac{\partial h_2}{\partial z_2} = h_2(1 - h_2) = .8 * (1 - .8) = -.16$ $\frac{\partial z_2}{\partial h_1} = W_{hh} = 0$
$\frac{\partial h_1}{\partial z_1} = h_1(1 - h_1) = .4(1 - .4) = -.24$

$\frac{\partial E_2}{\partial z_1} = .4 * .25 * -.16 * .5 * -.24 = .00192$

$.012 + .00192 = .01392$

Given

t_0, t_1, t_2=[0.1,-0.1,-0.2] x_0, x_1, x_2=[18,9,-8] h_0, h_1, h_2=[.2,.4,.8] y_0, y_1, y_2=[.05,.1,.2]

Gradient Based Methods

Udacity covers GD; Vincent covers SGD

SGD (Vincent's lecture)

Only 1 option w/BigData

SGD samples the input data

LGBFS/Ng's UFDL Stanford Notes

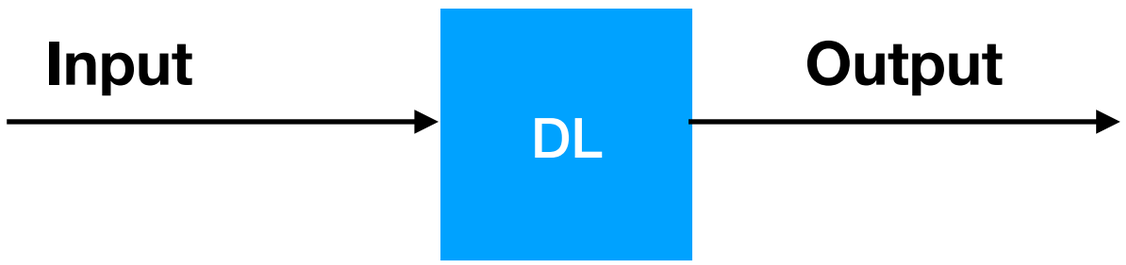
Conjugate Gradient Descent

Andrew Ng <-> Andrew Pop (word2vec)

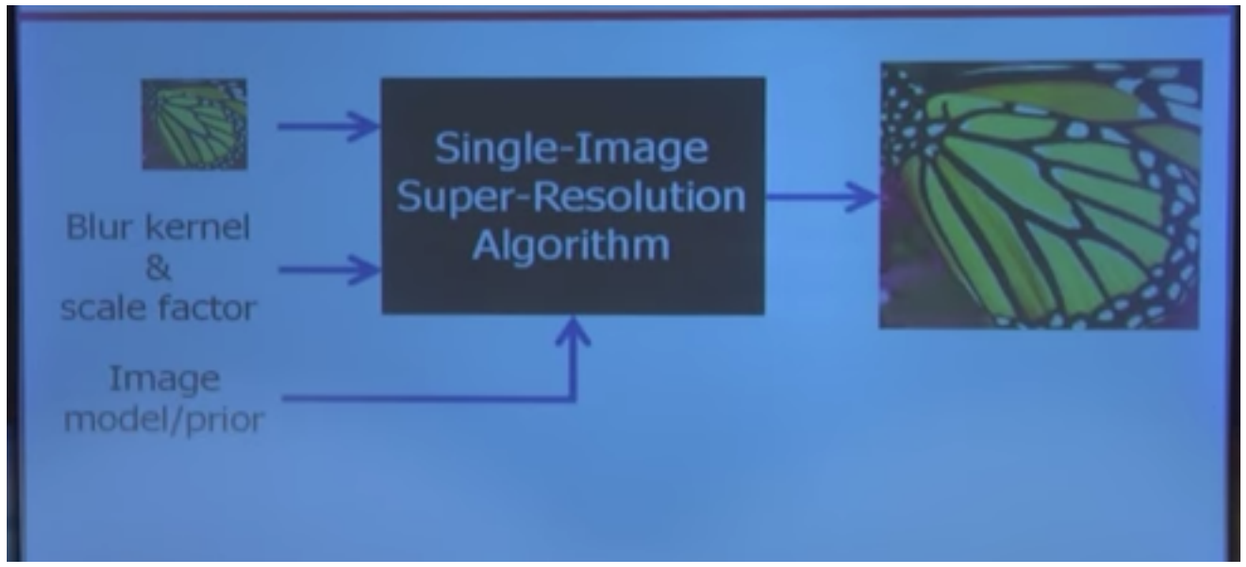
Poll: who is interested in hw accelerated momentum/python

Marshall /not telling him what to do

Deep Learning wo CNN



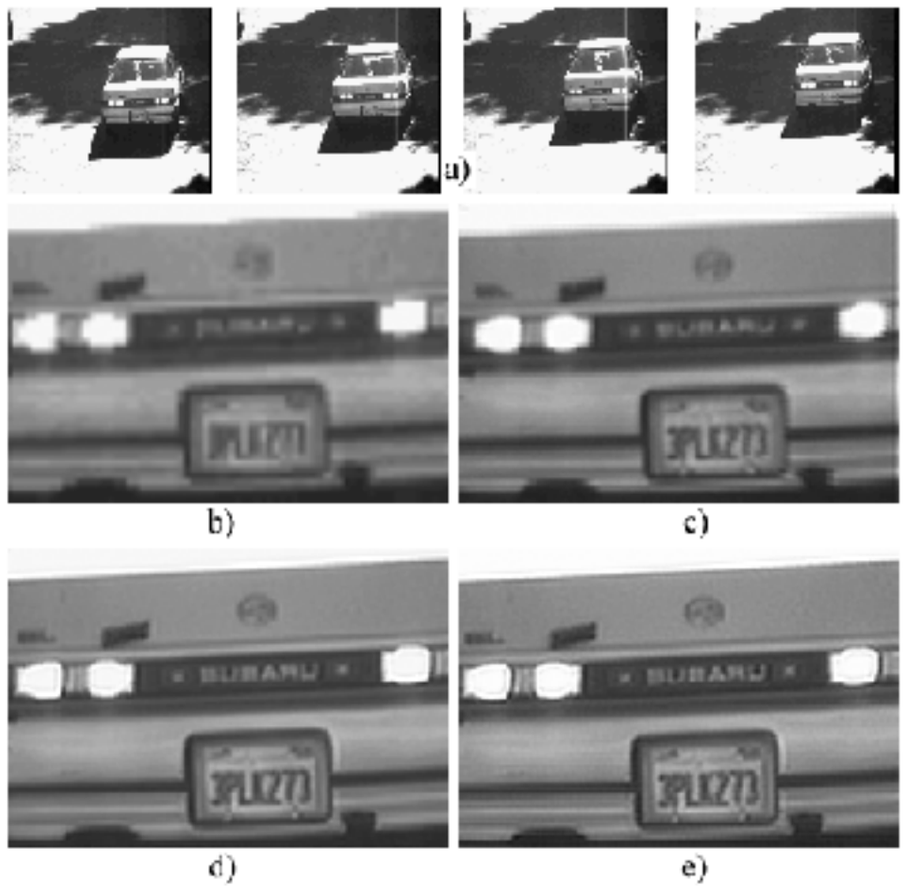
Super Resolution



not zoom, subpixel reconstruction

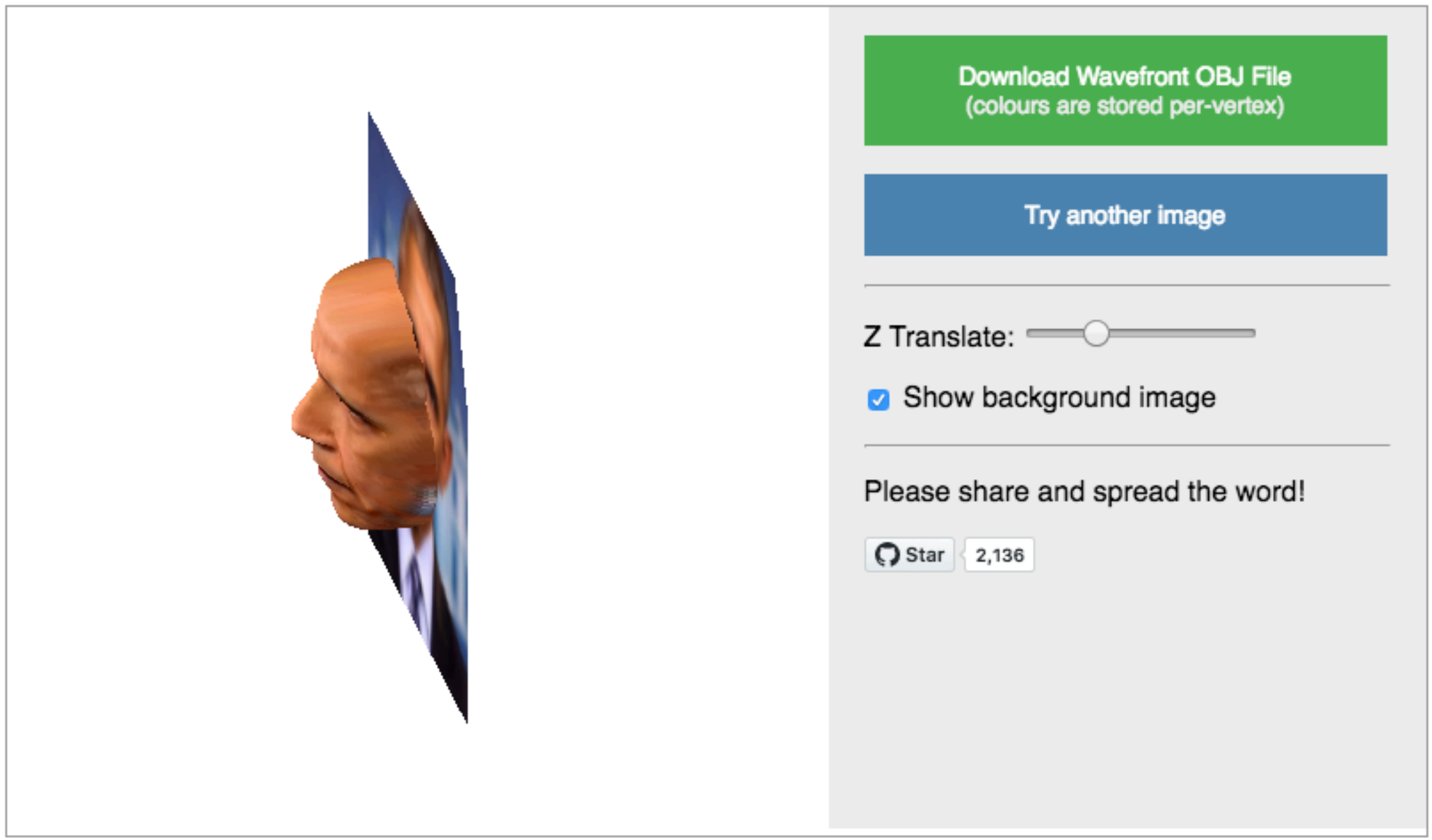
Trick: optimize for digits. leaf still blurry

Best algos not always DL



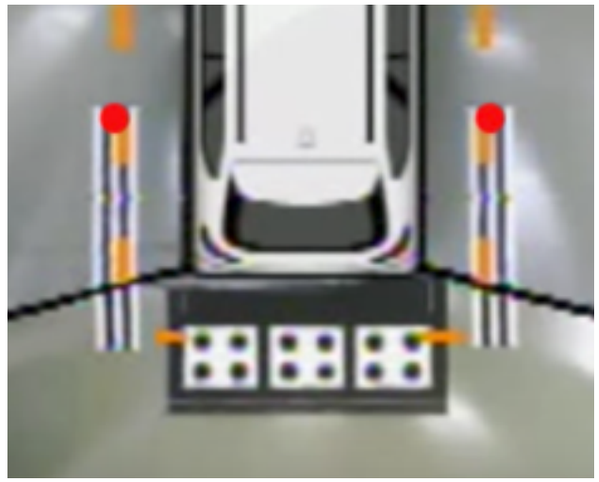
GANs

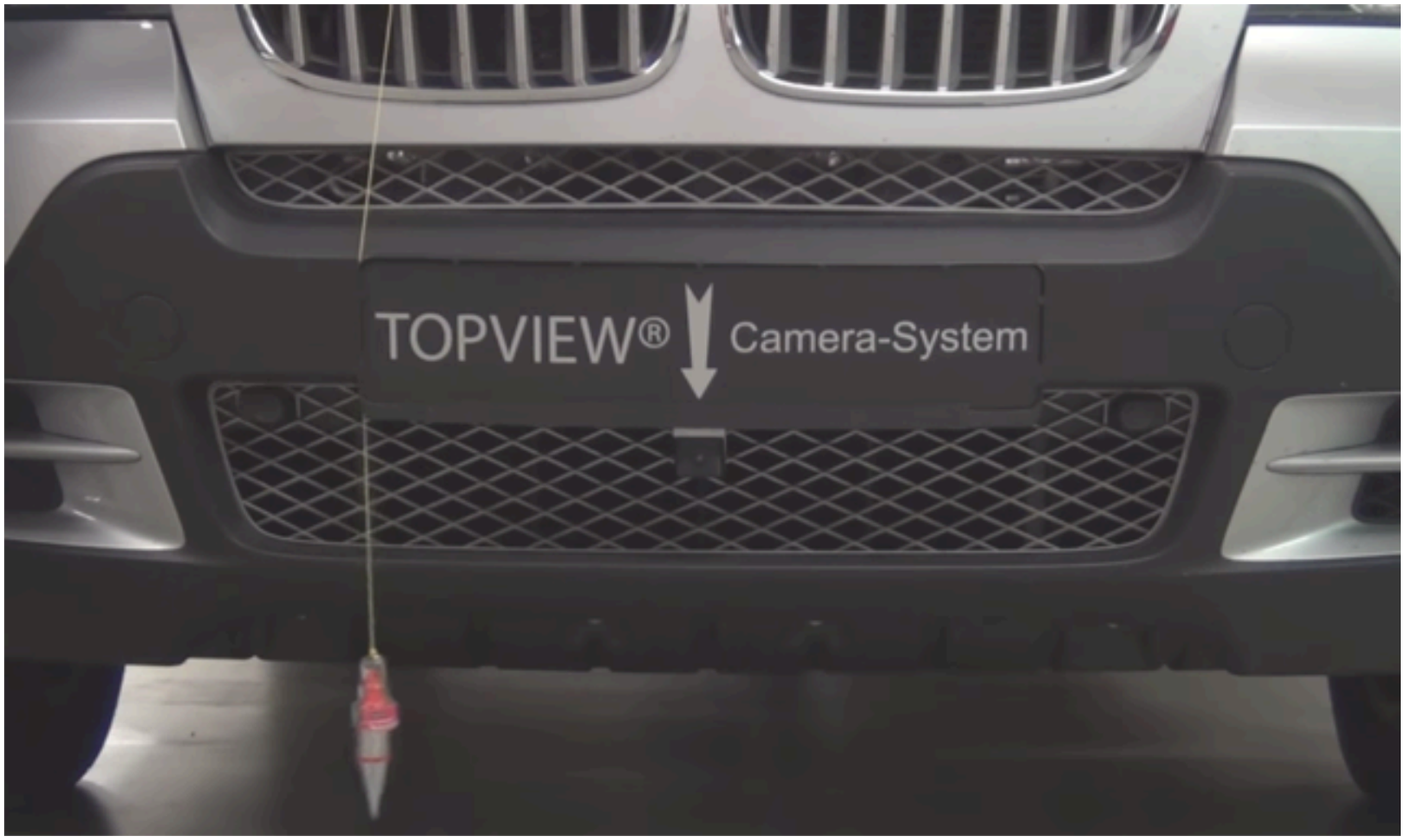
http://cvl-demos.cs.nott.ac.uk/vrn/view.php?name=../59b194ceed046



3D reconstruction from 2d image

BMW Car locator/parking





Perspective mapping

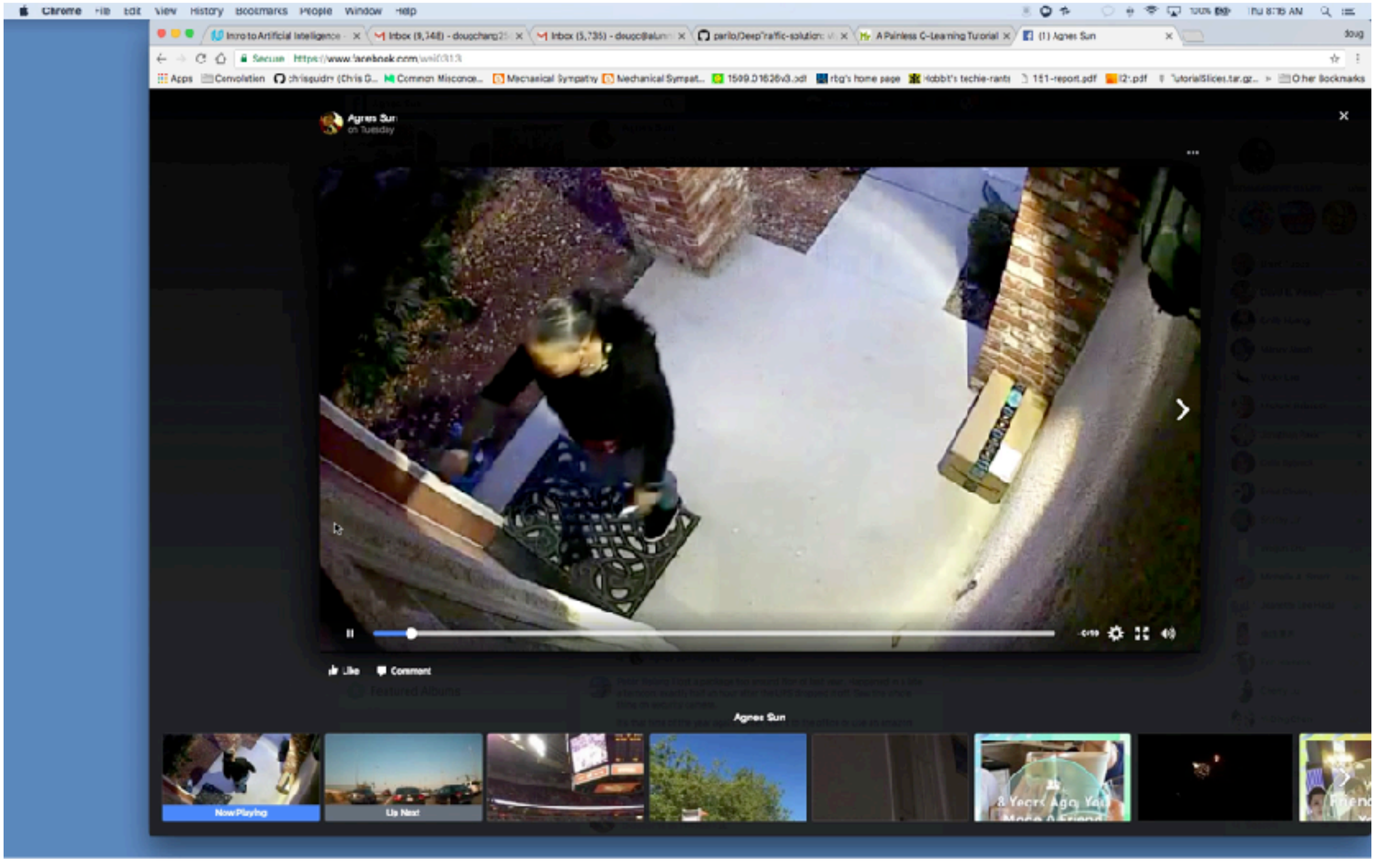
```
int board_w = atoi(argv[1]); //inner corners per row
int board_h = atoi(argv[2]); //inner corners per column
int board_n = board_w * board_h;
CvSize board_sz = cvSize( board_w, board_h );

//Hard coded intrinsics for the camera
Mat intrinsicMat = (Mat_<double>(3, 3) <<
    418.7490, 0., 236.8528,
    0.,558.6650,322.7346,
    0., 0., 1.);

//Hard coded distortions for the camera
CvMat* distortion = cvCreateMat(1, 4, CV_32F);
cvmSet(distortion, 0, 0, -0.0019);
cvmSet(distortion, 0, 1, 0.0161);
cvmSet(distortion, 0, 2, 0.0011);
cvmSet(distortion, 0, 3, -0.0016);

IplImage* image = 0;
IplImage* gray_image = 0;
```

ı inverse affine transformation represented by 2×3 matrix M :

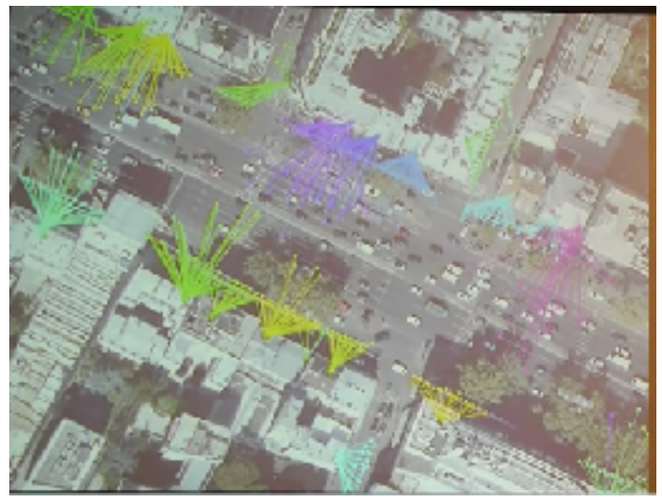
$$\begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \end{bmatrix}$$

matrix of the same type as M

Deep learning would find a better matrix/2 cameras to calibrate



RNNs



When **should** we worry about sequential decision making?

Limited supervision: you know **what** you want, but not **how** to get it
Actions have consequences

Common Applications

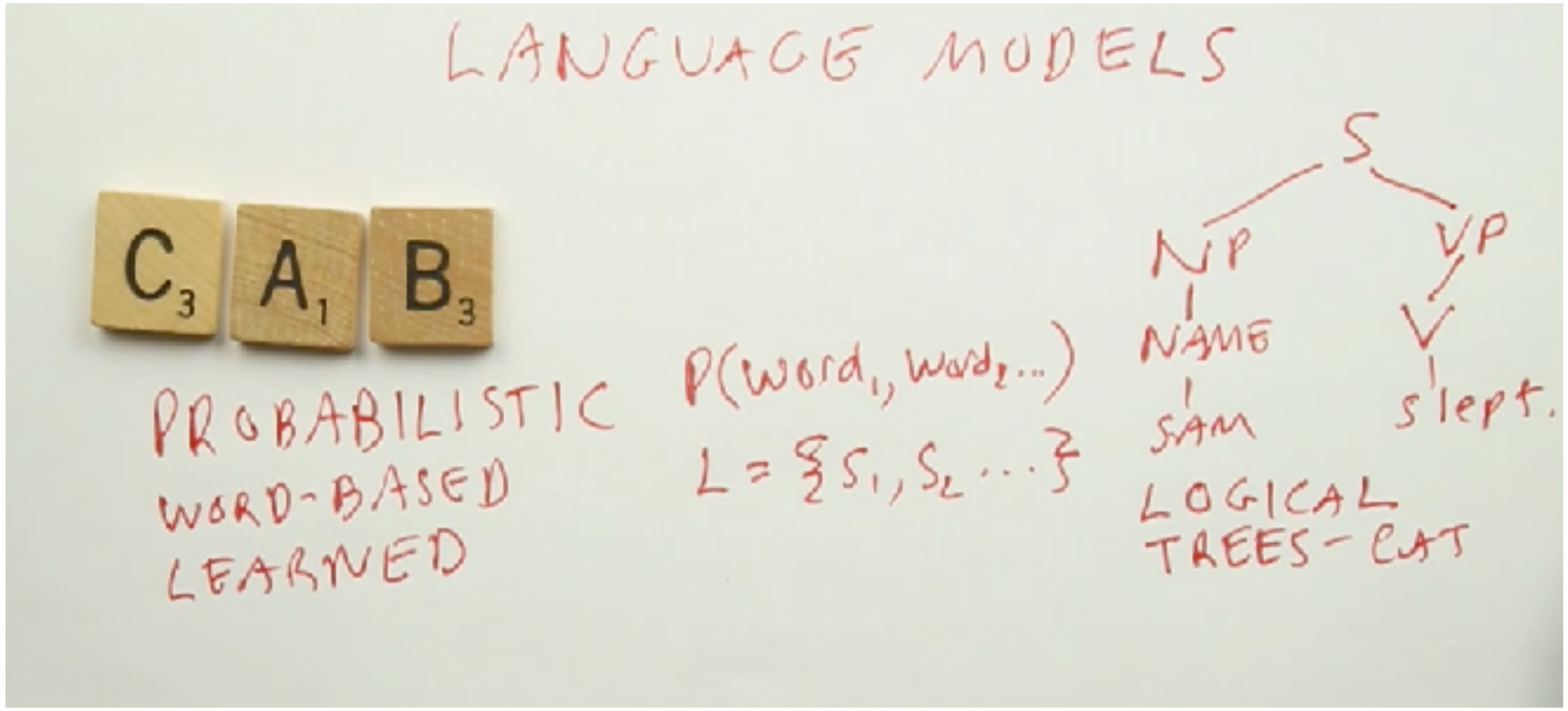
- Given a sentence, say if it is **{negative, neutral, positive}**
- Given the words in an email, predict if it is a **spam message**.
- Given "pieces" of an image, predict **what number** is in the image.

NLP - add more?

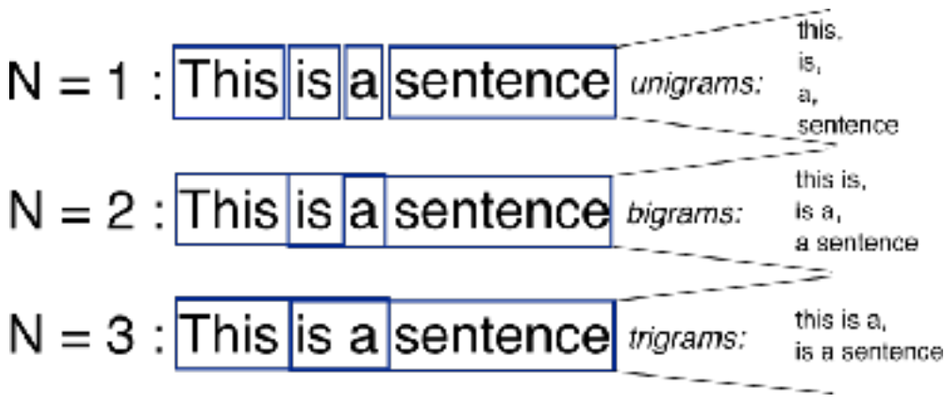
Poll: NLP tasks

Build a language model

Source: Peter Norvig(Google)/udacity



n-grams



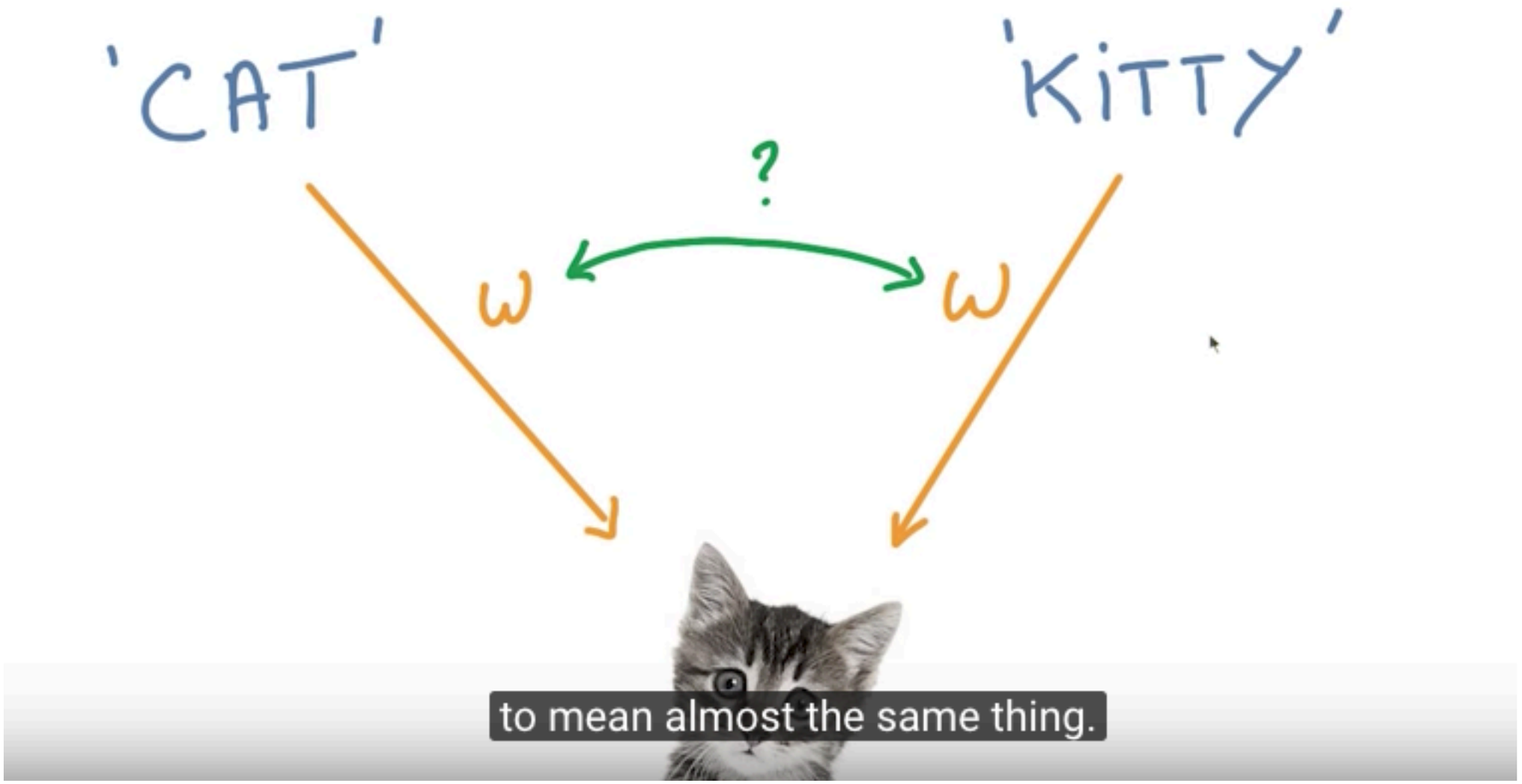
Next: word2vec
Need for embeddings

Semantic Ambuigity problem

We want to learn in a corpus cat similar to kitty and we want to share weights

This task using supervised learning would require too much labeled data which we don't have

Source: Vincent Vanhoucke(Google)/Udacity



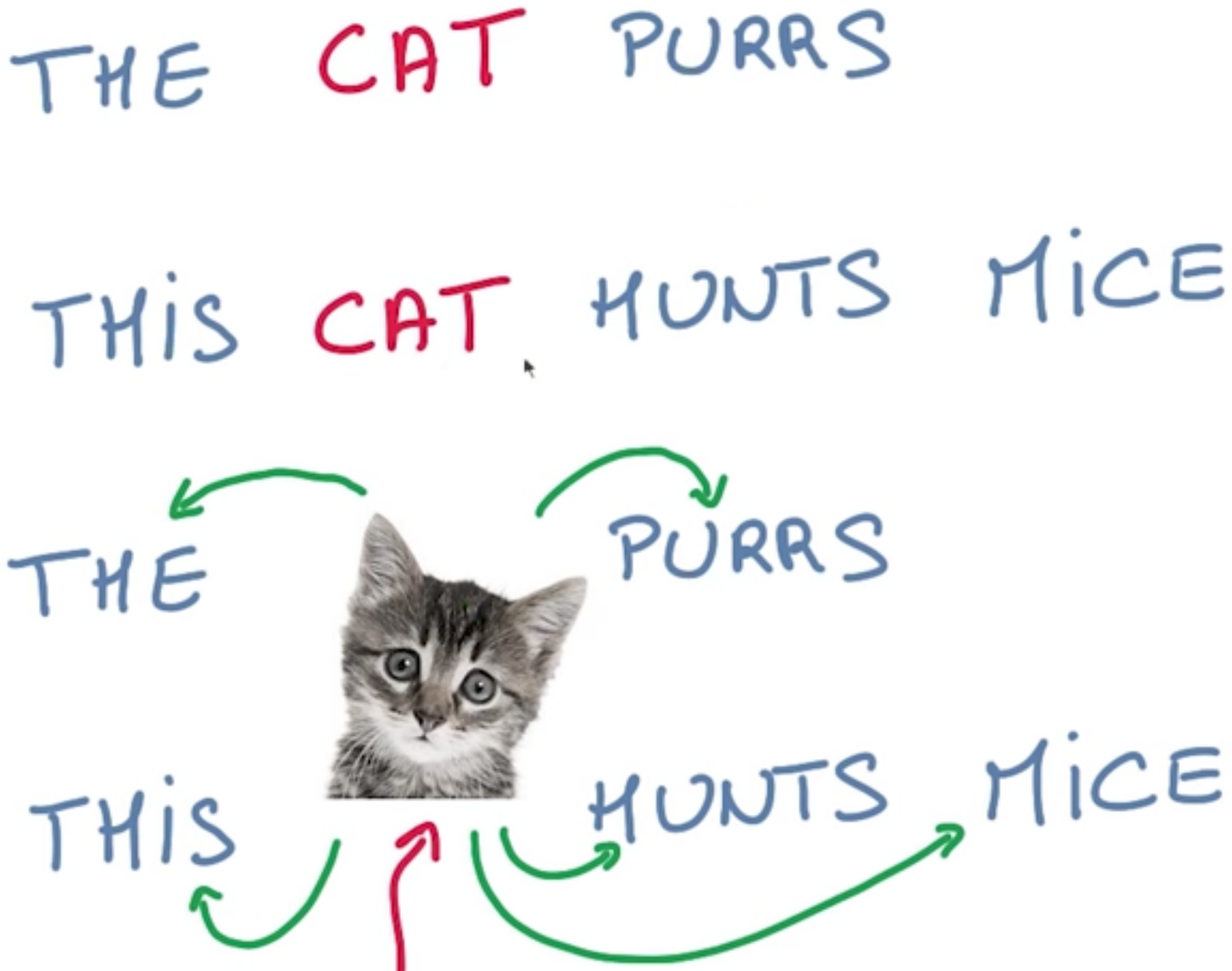
We are going to use unsupervised learning

Source: Vincent Vanhoucke(Google)/Udacity

Unsupervised learning

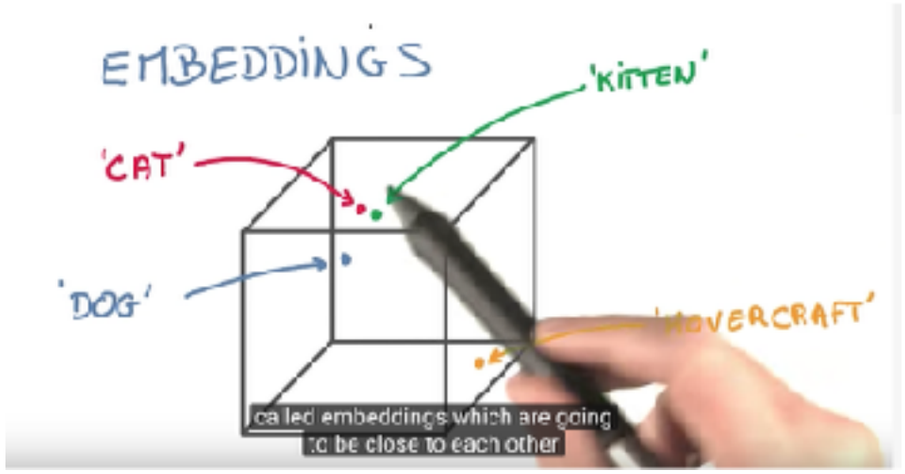
The trick we are going to use is similar words appear in similar context.

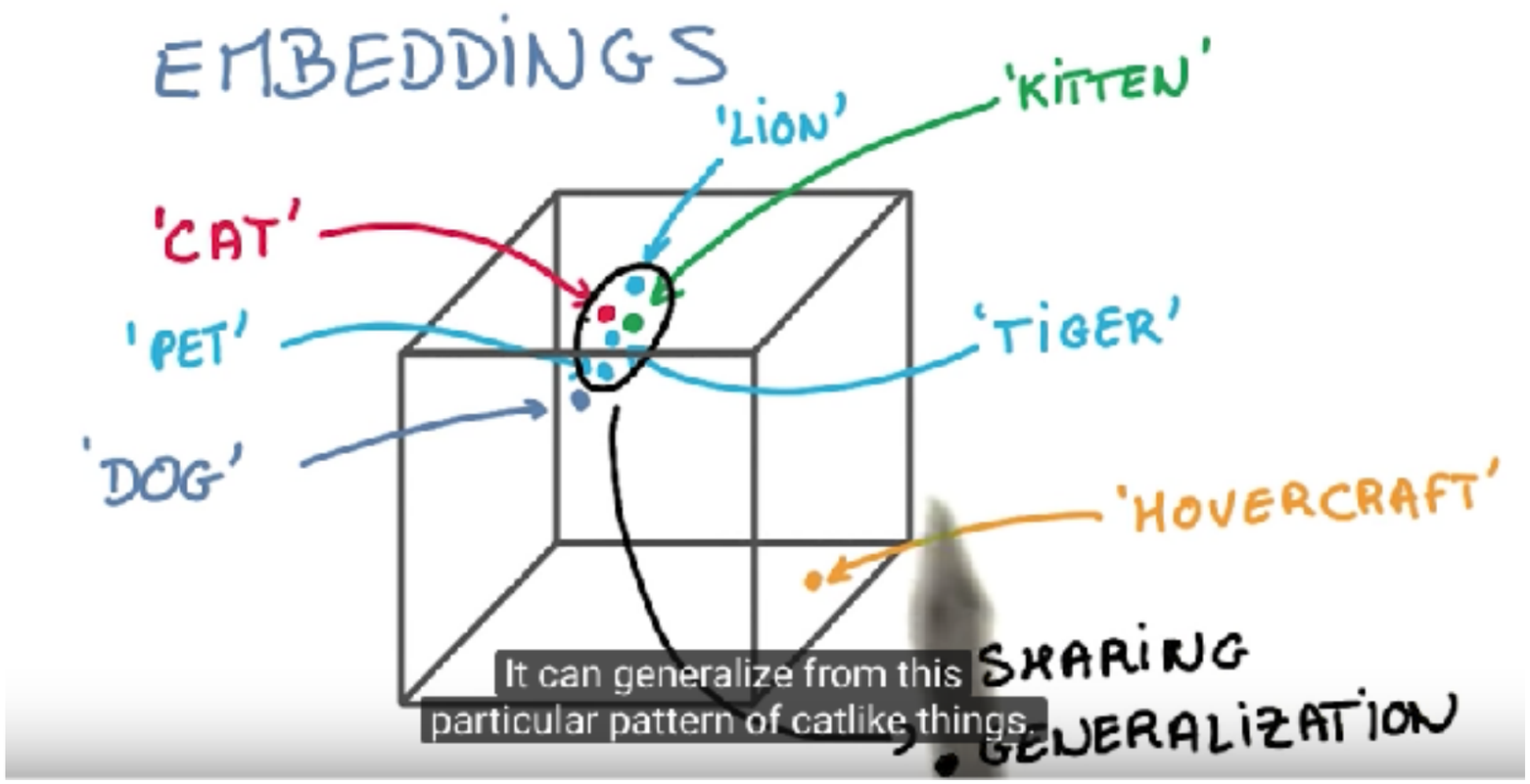
as we process more and more text we will notice we can use kitty and cat interchangably



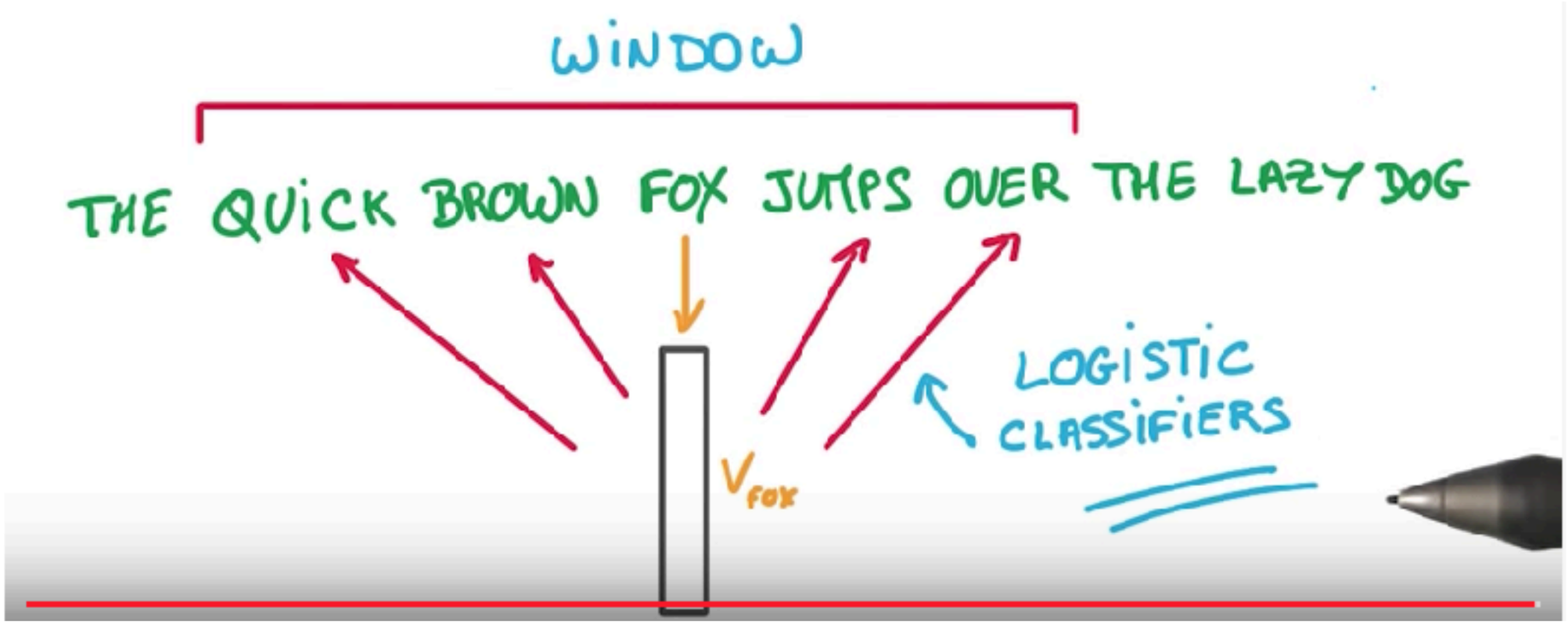
We are going to use embeddings which is a NN and will map similar words close to each other

Source: Vincent Vanhoucke(Google)/Udacity





Word2Vec finds these embeddings



1) map each word to a random embedding

2) use embedding to predict next word

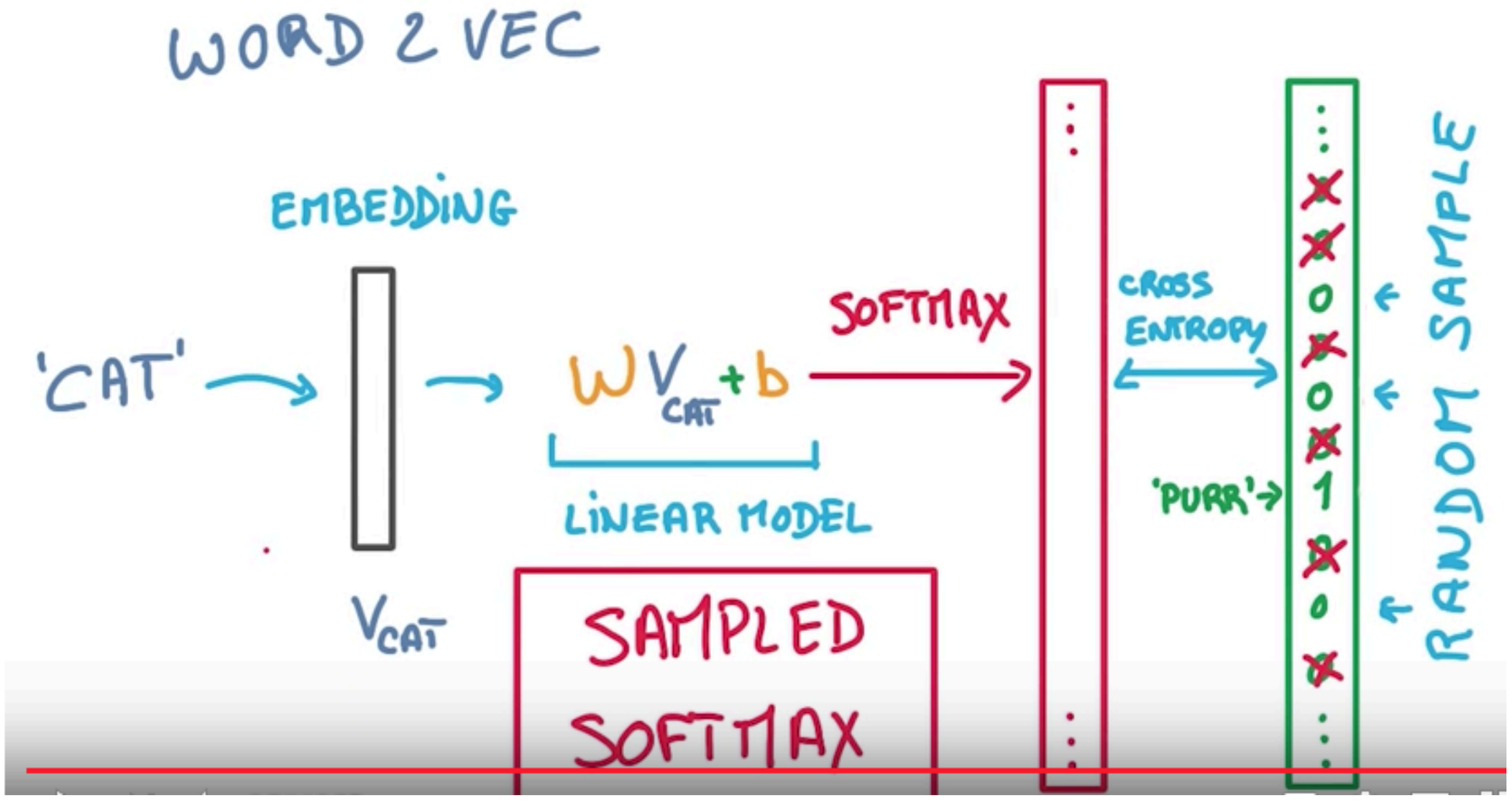
3) iterate over massive text and kitty purrs and cat purrs examples will cause cat/kitty to be next to a common word. This signal makes these words close together in the embedded cube

Source: Vincent Vanhoucke(Google)/Udacity

Source: Vincent Vanhoucke(Google)/Udacity

Embeddings/Word2Vec

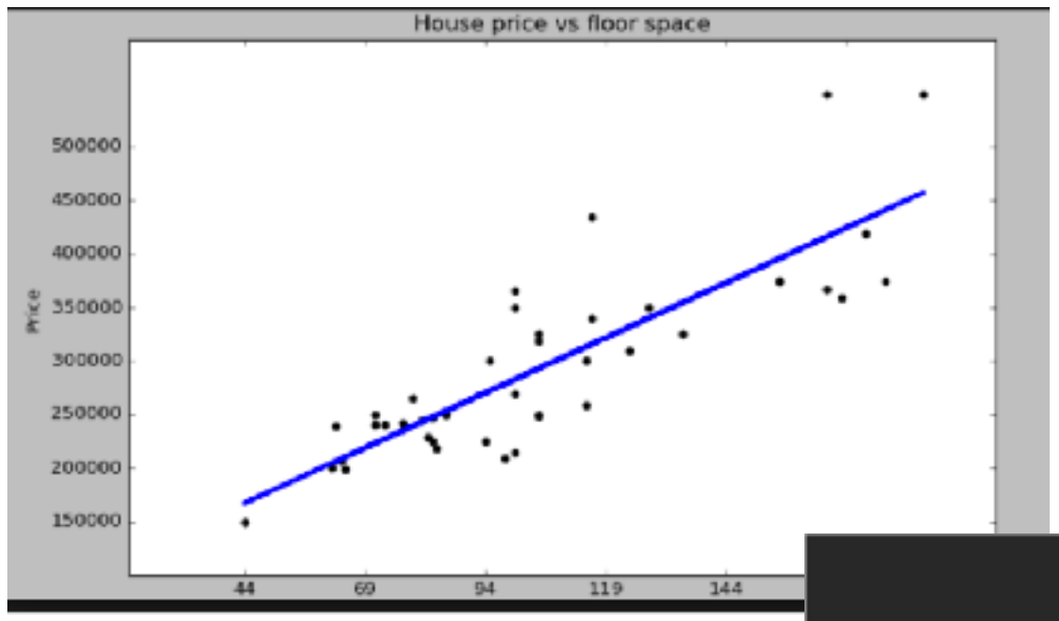
http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf



Outliers are good/ we need them to do feature engineering/on the outliers!!!!

Don't do this

cs20si youtube lecture3 18:39



Feature Engineering

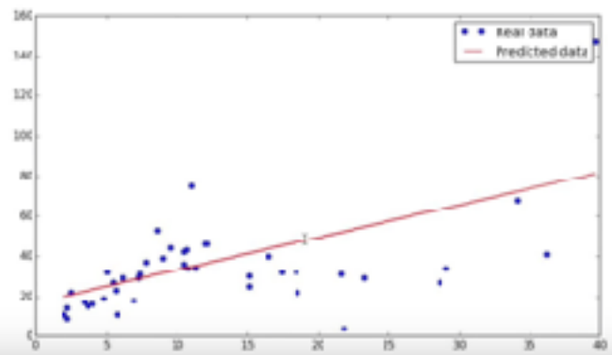
Create separate feature to detect outliers.
Increase profit

Do not hide it. Underfit

Never underfit

Example viable models

How to improve our model



Huber loss

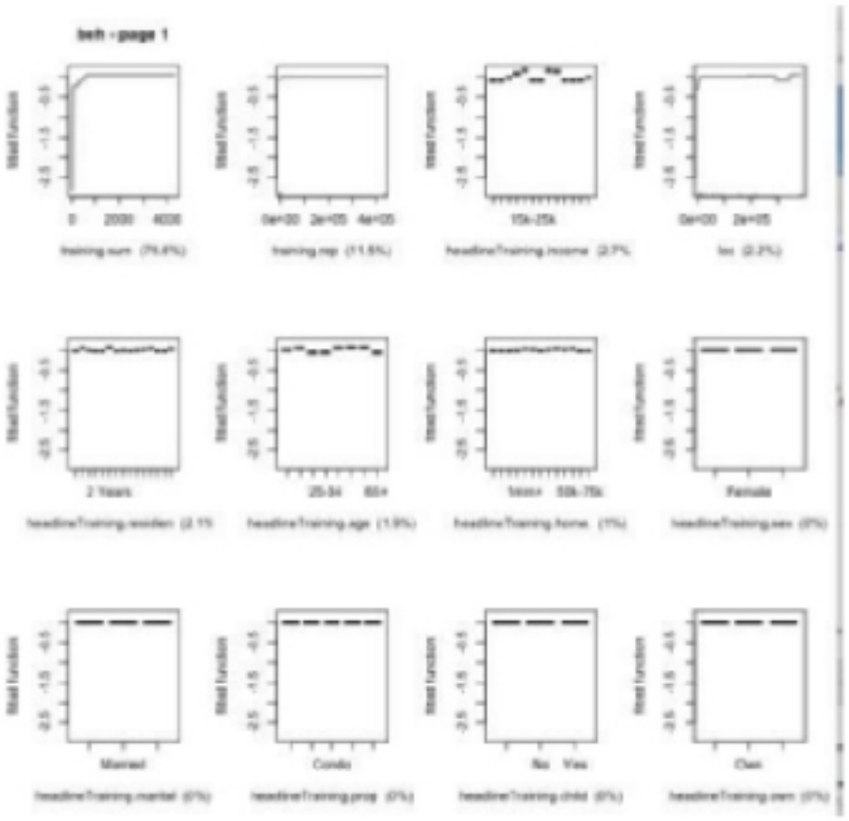
Robust to outliers

Intuition: if the difference between the predicted value and the real value is small, square it

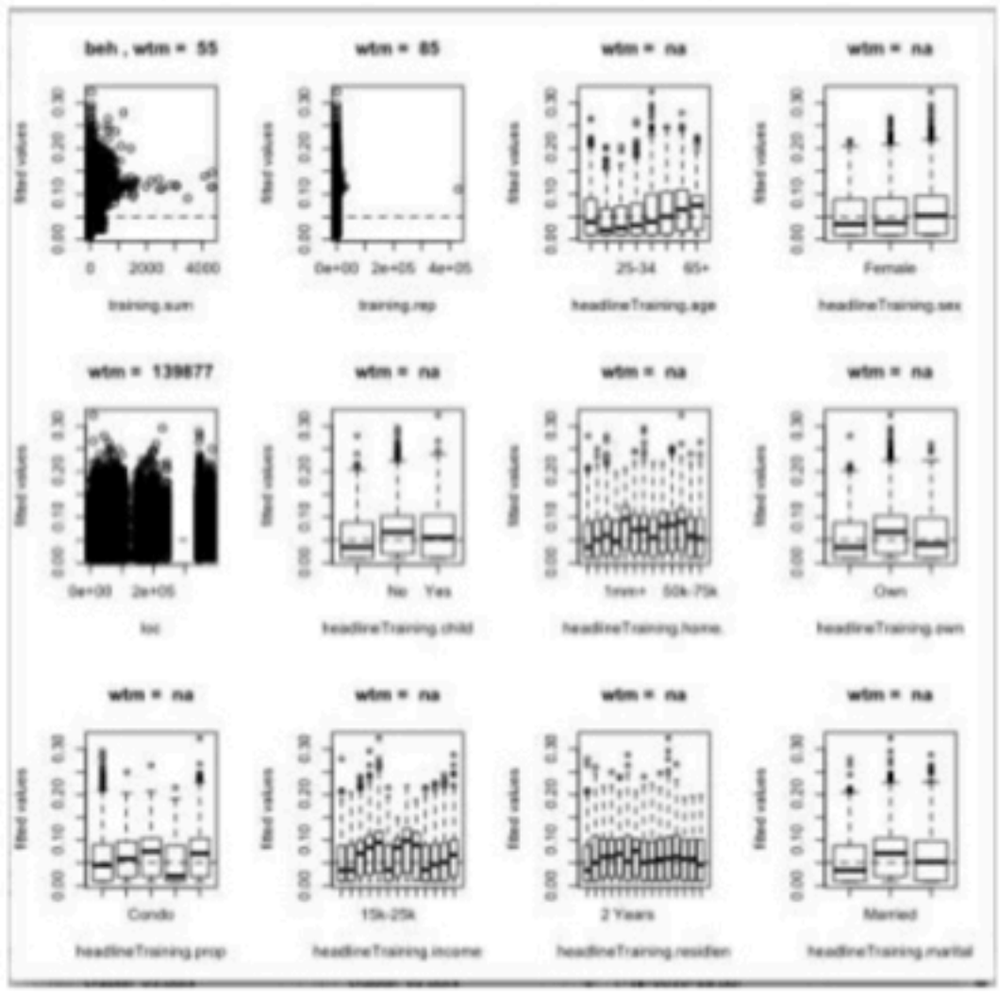
If it's large, take its absolute value

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Fitted Values (Crappy)

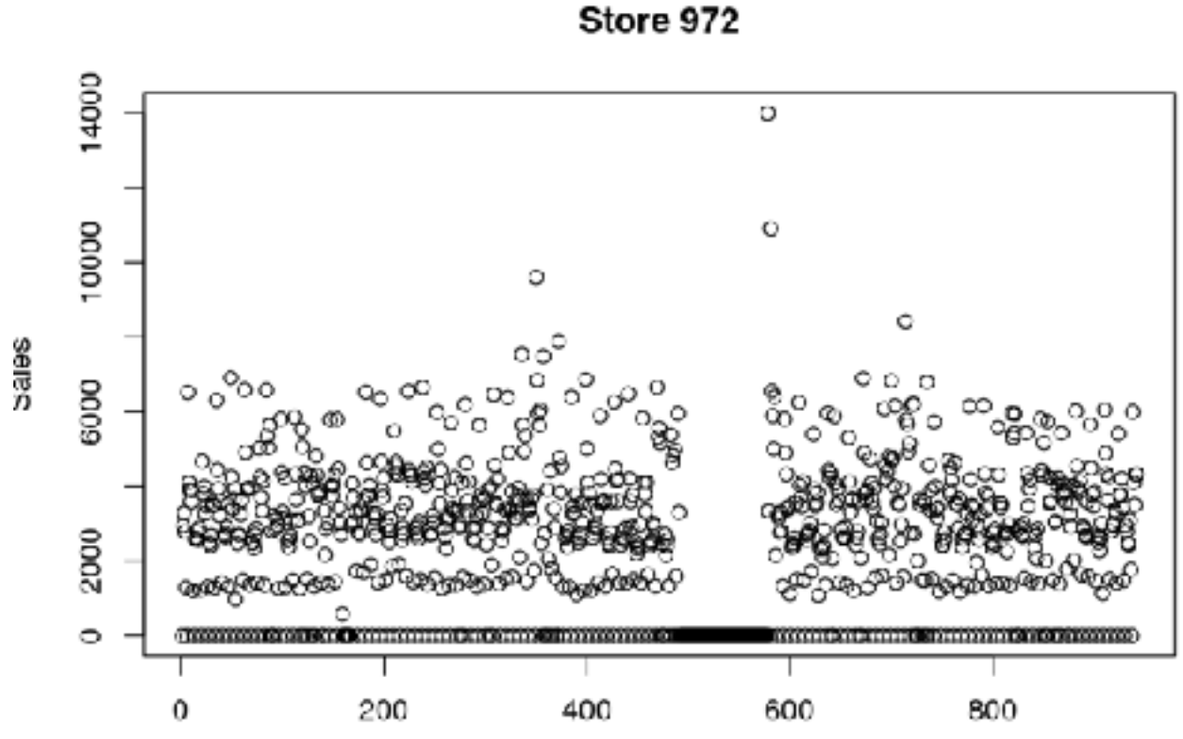


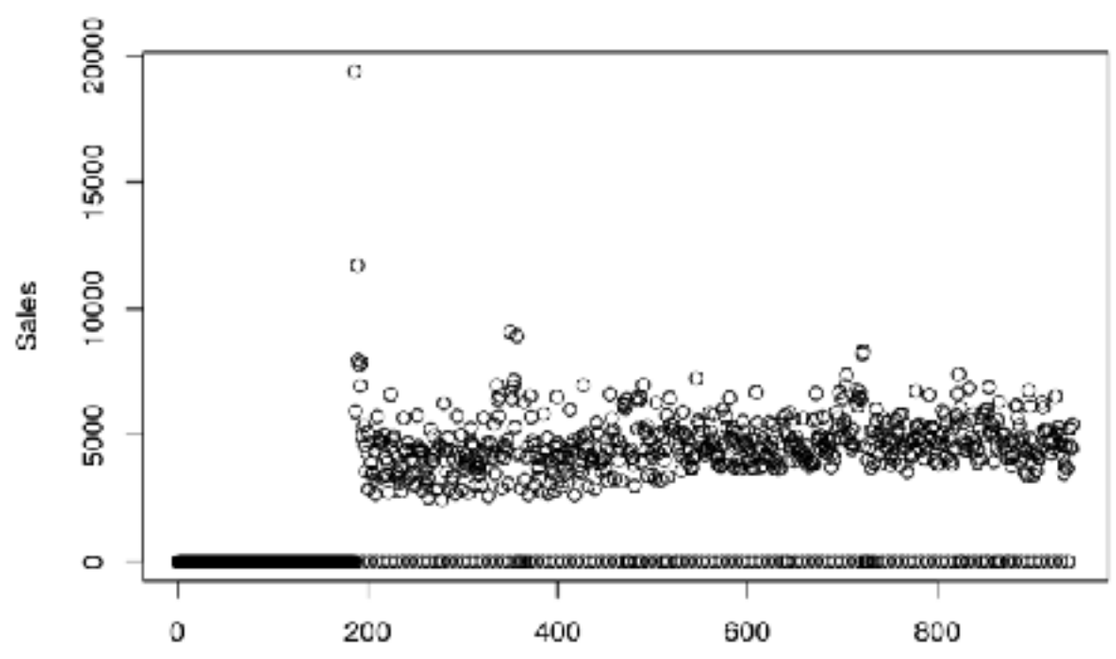
Fitted Values Better



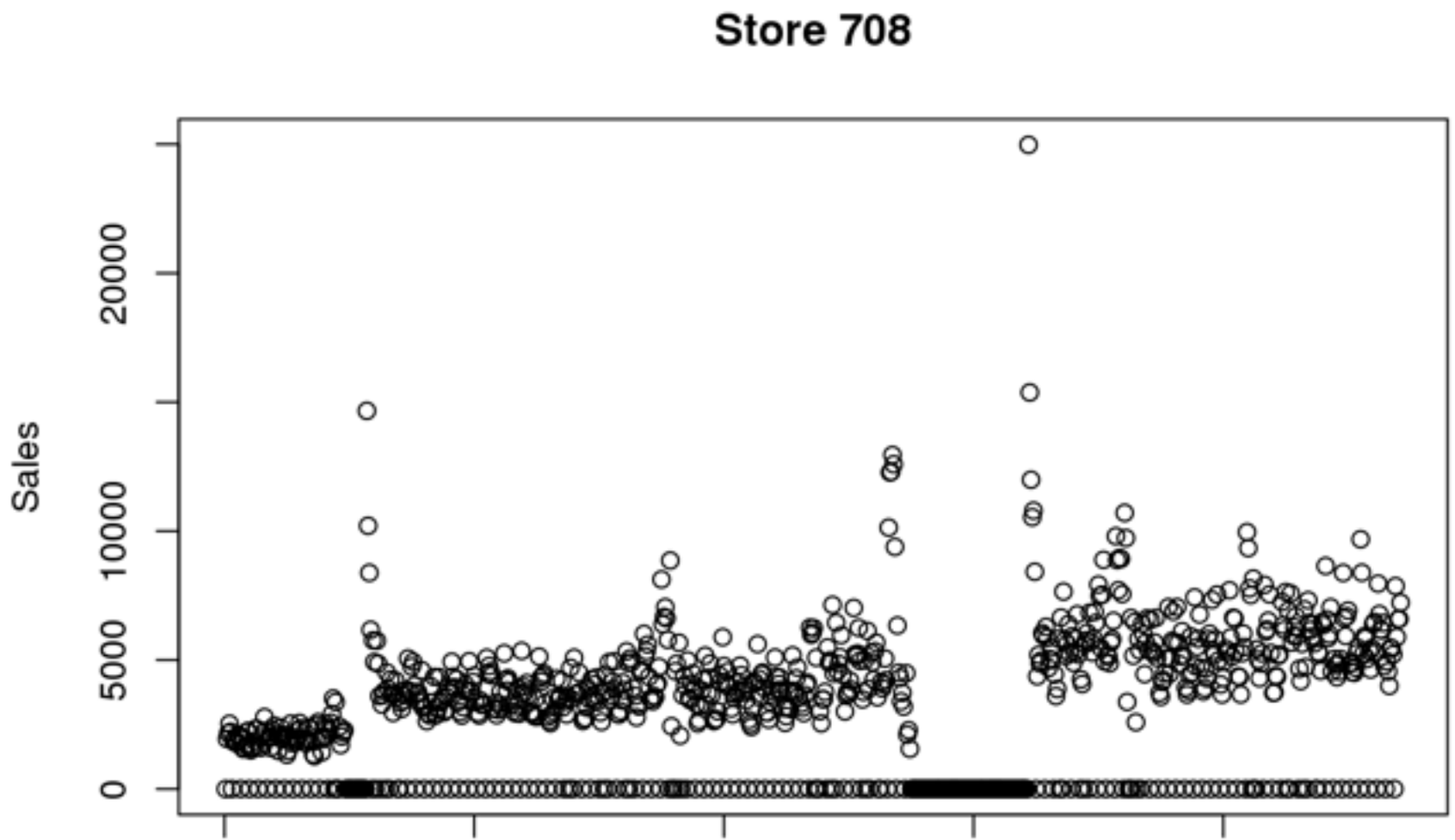
The crappy way/ can add numerical values to DL
make them more efficient/sparse w/embeddings

https://www.slideshare.net/DougChang1/demographics-andweblogtargeting-10757778





https://www.kaggle.com/thie1e/exploratory-analysis-rossmann



Entity Embeddings of Categorical Variables Cheng Guo/Felix Berkhahn

https://arxiv.org/pdf/1604.06737.pdf

- Convert categorical to embeddings. Related to Word2Vec

    - DL back propagation only holds for continuous functions. EE map them closer to continuous distributions

- Add additional data from weather and Google Trends

- Feature Engineering

- Bonus/makes existing algorithms Decision Trees/XGBoost orders of magnitude faster

- State of the art performance at 90-95%.

Reformat date split the strings into month/#, day/# etc…

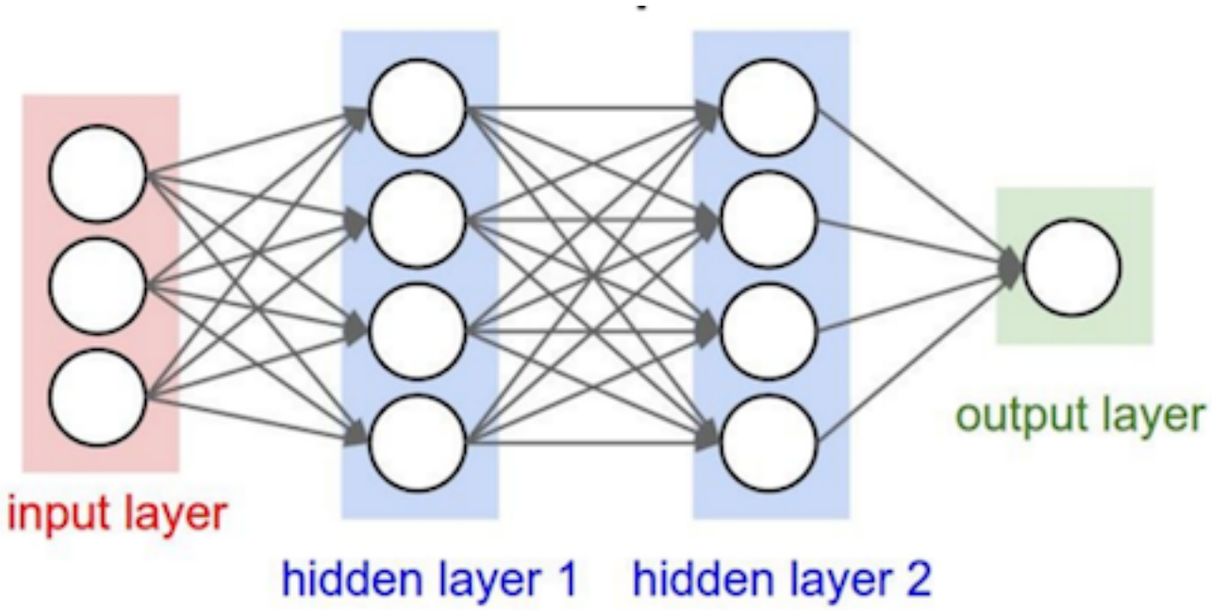
Instead of root mean square percentage error (RMSPE) used in the competition we use mean absolute percentage error (MAPE) as the criterion:

$$MAPE = \left\langle \left| \frac{Sales - Sales_{predict}}{Sales} \right| \right\rangle \qquad (24)$$

The reason is that we find MAPE is more stable with outliners, which may be caused by factors not included as features in the Rossmann dataset.

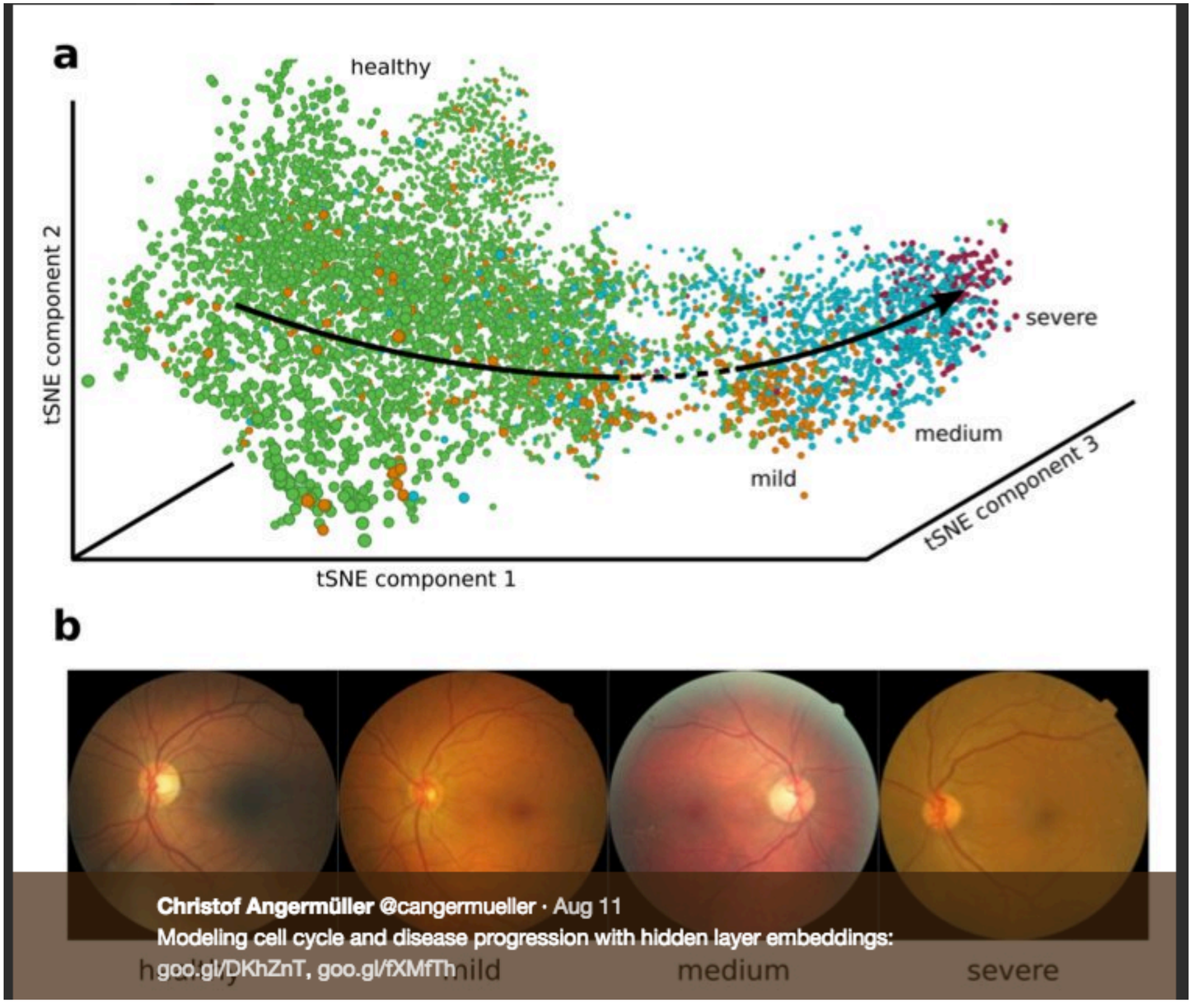
```
model.compile('adam', 'mean_absolute_error')
```

2 layer FF network

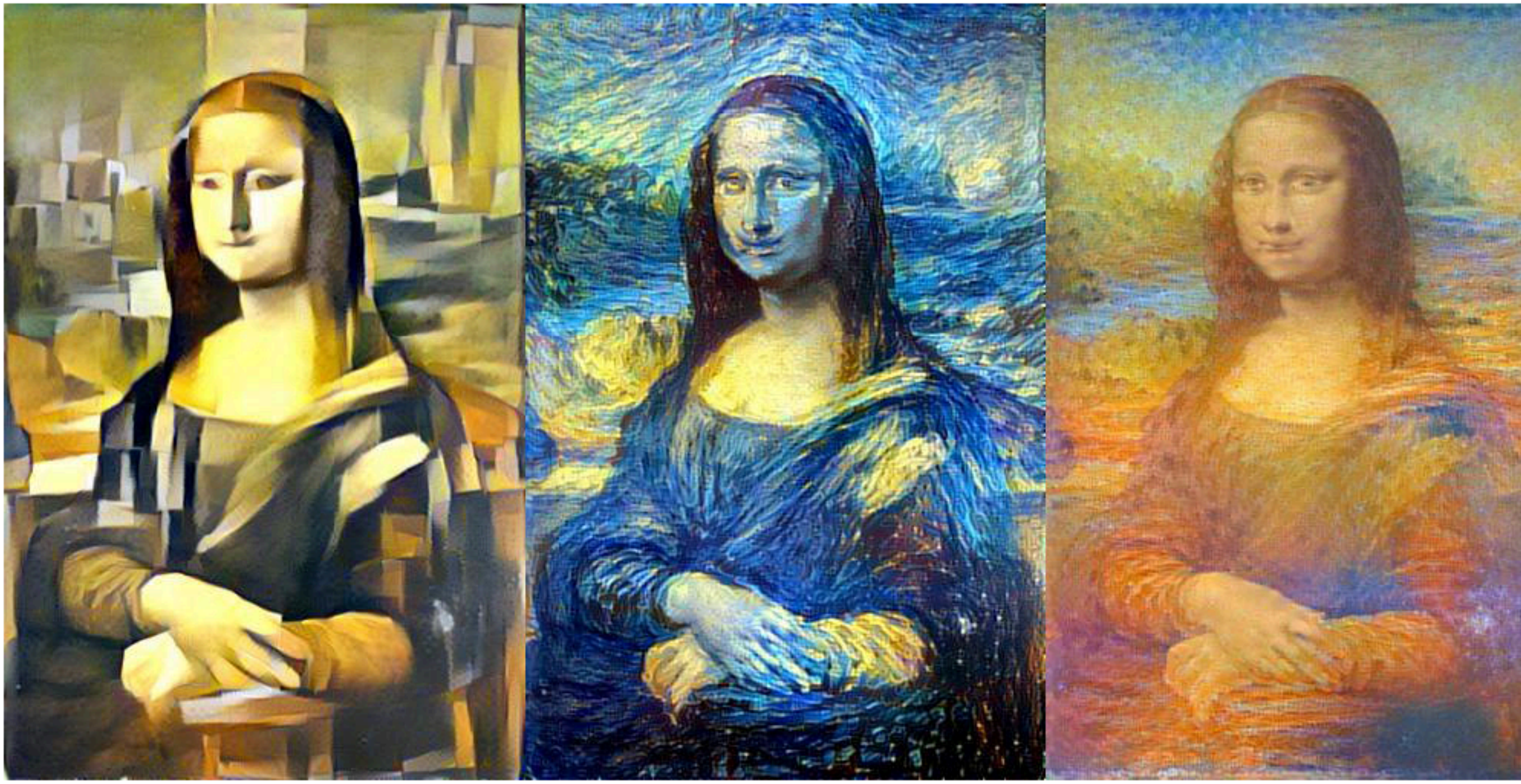



```
x = Dropout(0.02)(x)
first_layer = Dense(1000, activation='relu', name="first_layer")(x)
second_layer = Dense(500, activation='relu', init='uniform',name="second_layer")(first_layer)
out = Dense(1, activation='sigmoid')(second_layer)
```

Discrete values close enough to be continuous



Transfer Learning



Transfer Learning - data vs. new algorithms

https://medium.com/nf2-project

NF1 on chromosome 18

NF2 on chromosome 22

Discovered via hackathon

Sponsored by accel.ai/google

Expanded to health events

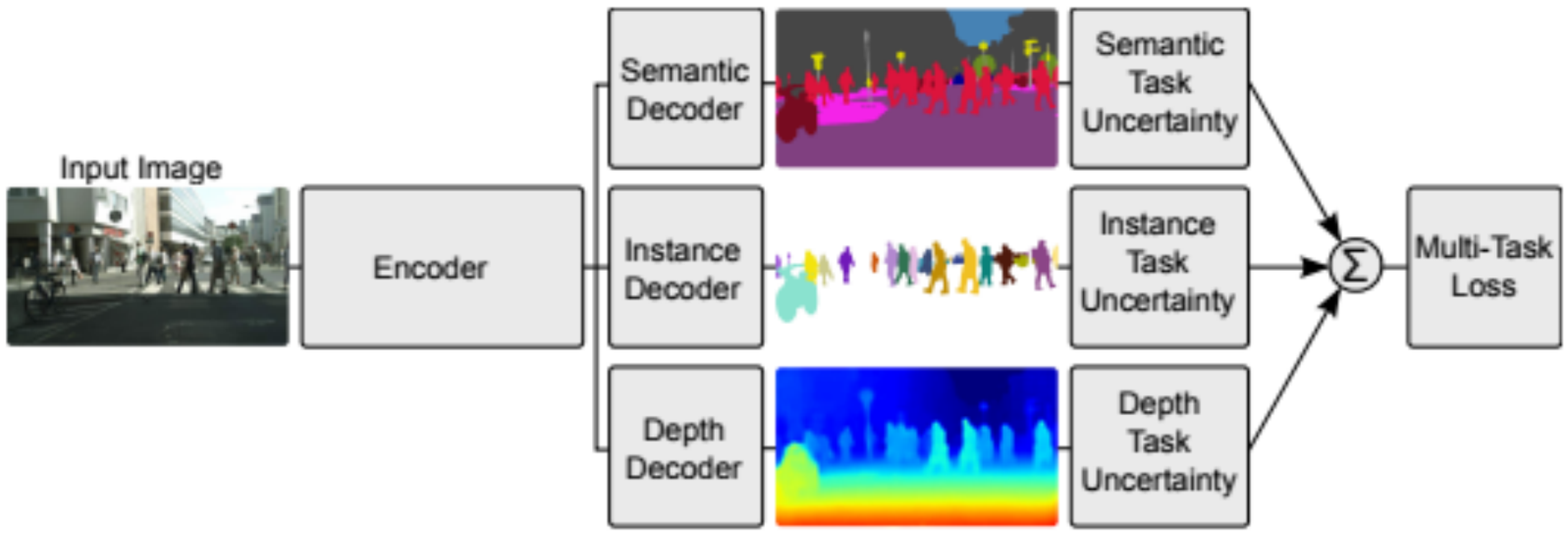
Multitask learning

- multimodal learning from Ng, using audio and video together;

- Multi-Task learning: https://arxiv.org/pdf/1705.07115.pdf

Poll: who has seen this/has application for this?

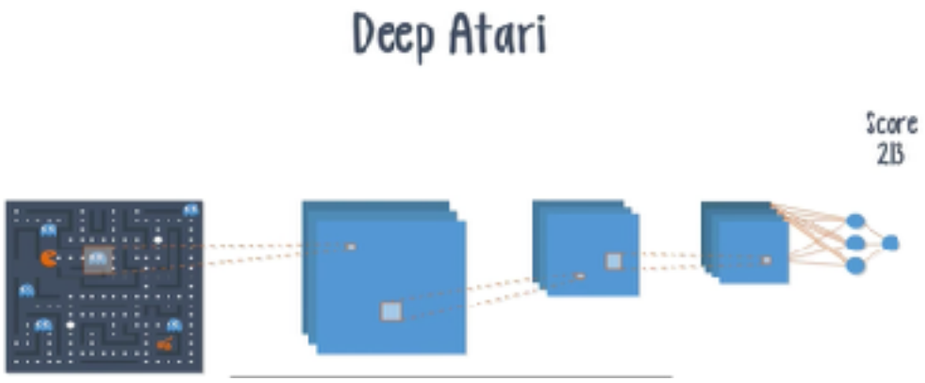
RNN type apps



Dec 2013 Deepmind Atari games/agents

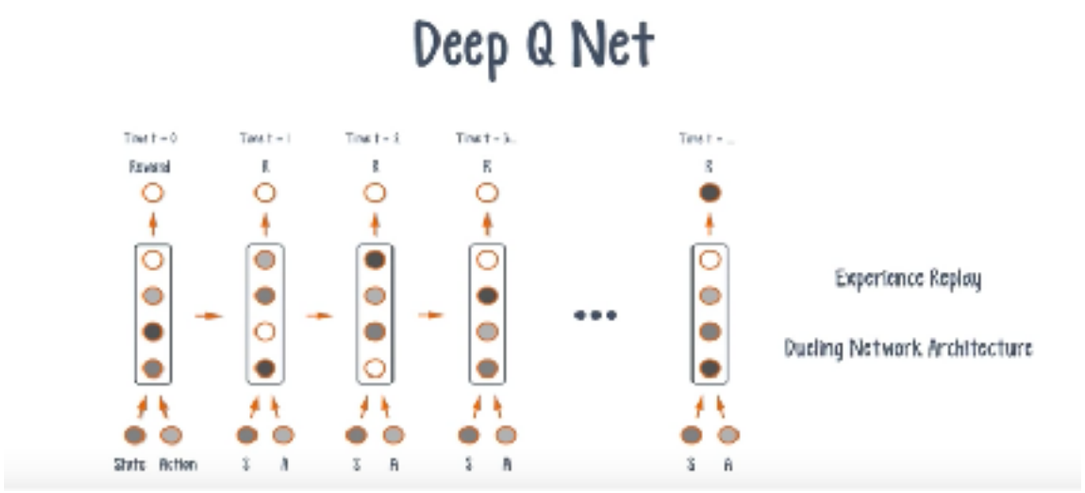
Reinforcement Learning

Design an agent to maximize a reward



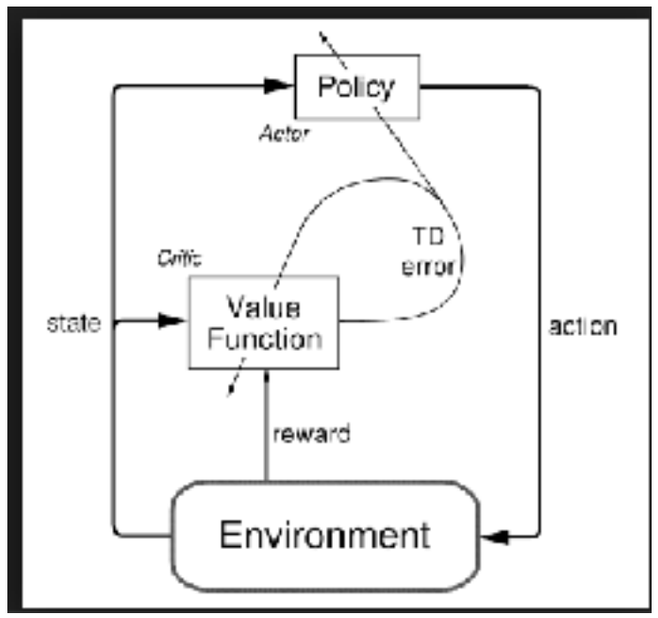
take screenshots of atari games, build a modified CNN to output a reward vs. a classification probability

reward: get points for staying in lane, etc.
lose points for collisions, etc…
actions: speed of car, steering angle
maximize reward



Q learning:
minimize temporal difference

Deterministic Policy Gradients



maximize expected reward

MDP-Markov Decision Process

- in current state s member of S
- with action a member of Actions(s)
- P(s'|s,a) probability of going to state s' when in state s and action a. We have a probability P because we don't have the same result every time on action a. (definition of stochastic)
- R(s,a,s') ; R is a reward state given current state s, next state s' and action a.

To solve a MDP

- find a PI(s) which is going to maximize the discounted total reward.

math… 10/19 4am.. I changed my mind.. schedule a deep dive like backprop