Applied Deep Learning

Lecture 2 • Jan 31, 2019

Today's agenda

Administrative stuff

- Office hours posted
- Assignment #1 released (see GitHub - **bit.ly/applied-dl**)

Topics

- Bias in data / incidents in DL (Important to cover these before image classification).
- Linear classifiers
- Softmax, loss
- Lab (This lecture won't take the whole class, we'll start on the homework too).

Office hours

Who	When	Where
Pratik	Mon: 4pm - 5pm	Mudd TA Room
Prerna	Tues: 4:30pm - 6:30pm	Mudd TA Room
Josh	Thurs: 5:30pm - 6:30pm	Mudd 417
Terry	Fri: 10am - 11am	Mudd TA Room
Seungmin	Fri: 1pm - 2pm	Mudd TA Room

The TA room is on the 1st floor of Mudd. I'm also available right after class.

Assignment 1 walkthrough

Uploaded to GitHub

Goal: write neural networks in two different styles in TensorFlow.

- Due Feb 14th
- Optional EC section

The assignment includes links to helpful code you can base your work on.

Questions from last time

We mentioned the expanding space of languages for DL. But we didn't cover why Python is so popular.

Discussion: Why?

Machine Learning

happens in Python,

right?

```python
import numpy as np
from timeit import default_timer as timer

# Create a random ndarray
size = int(10e7)
data = np.random.random(size)

start = timer()
data *= 2
end = timer()
print(end - start) # 0.12
```

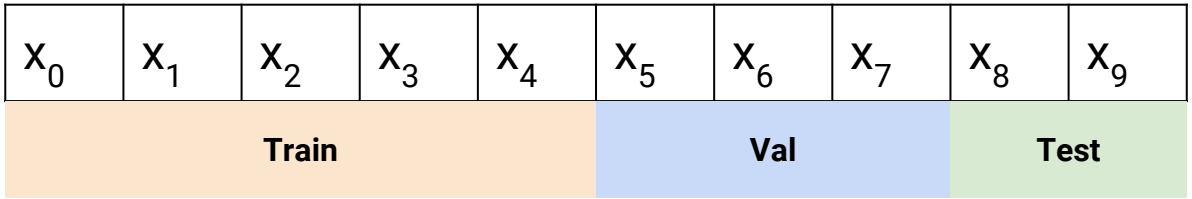
```python
import random
from timeit import default_timer as timer

# Create a random list
size = int(10e7)
data = [random.random() for _ in range(size)]

start = timer()
for i in range(len(data)):
    data[i] = data[i] * 2
end = timer()
print(end - start) # 13.33
```

A toy benchmark. Naively, this gives a +/- 100x speedup.
That's the difference between 6 seconds and 10 minutes.

Bias in data, famous incidents in DL.

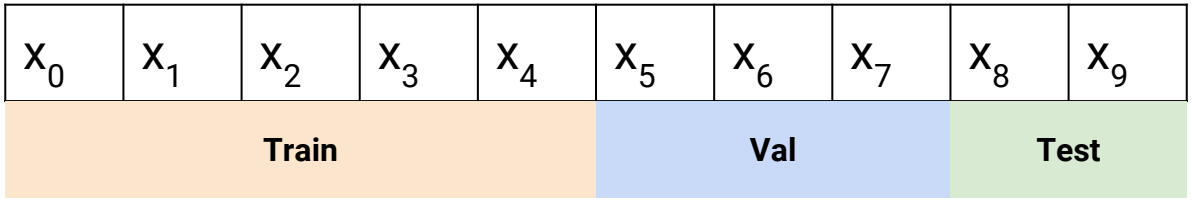
Warm up: One simple reason why mistakes happen.

What life looks like when working with a UCI or Kaggle dataset



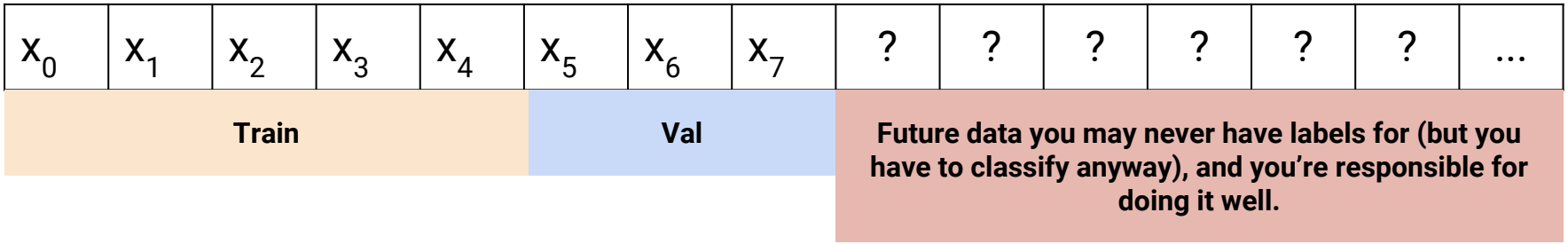
This is the easy case (you have access to all the data your system will ever see).

Reality



This is the easy case (you have access to all the data your system will ever see).

You build a system and deploy it to make predictions in production - will users be happy?



The way it's supposed to work

1. Download a dataset.
2. Divide into train / validation / test.
3. Train your model and tune the parameters.
4. Evaluate on test once; report your accuracy.

What actually happens

1. Download a dataset.
2. Divide into train / validation / test.
3. Train your model and tune the parameters.
4. Evaluate on test ~~once~~; ~~report your accuracy~~.

What actually happens

1. Download a dataset.
2. Divide into train / validation / test.
3. Train your model and tune the parameters.
4. Evaluate on test ~~once~~; ~~report your accuracy~~.
5. Tune your parameters a bit more.
6. Evaluate on test again.
7. ….
8. Report your accuracy.

Discussion: What does this cause?

What actually happens

1. Download a dataset.
2. Divide into train / validation / test.
3. Train your model and tune the parameters.
4. Evaluate on test ~~once~~; ~~report your accuracy~~.
5. Tune your parameters a bit more.
6. Evaluate on test again.
7. ….
8. Report your accuracy.
9. Overfit.

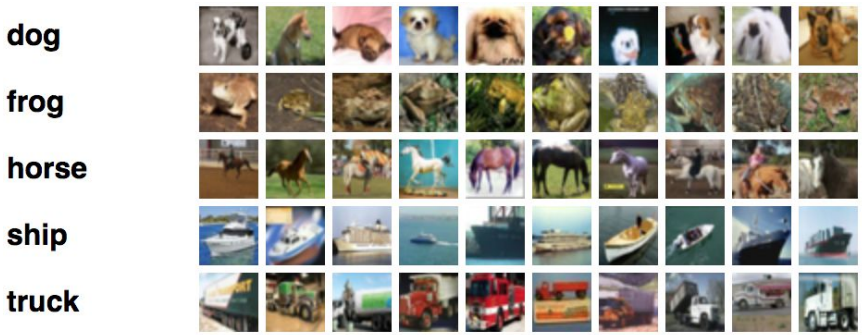
Your data is rarely as diverse as you think

And, why it’s important to carefully look at your data

Why it's important to look at your data

(Hint: it can be biased in **non-obvious** ways).

Image: you have developed an image classifier. It performs well in practice on your train / validation / and test sets, with 95% accuracy on every class… but!



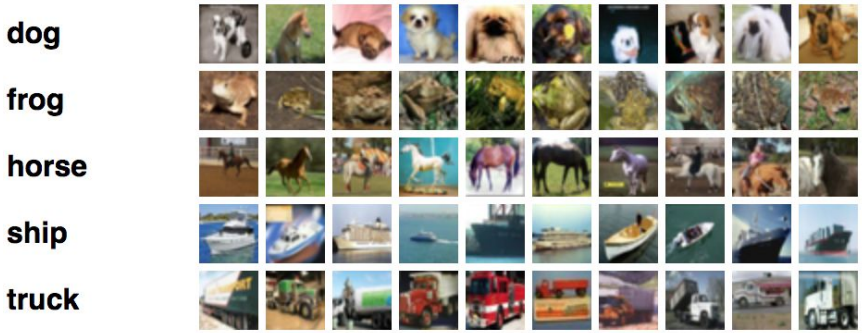
Train / test / val accuracy

Dogs: 95%
Frogs: 95%
Horse: 95%
Ship: 95%
Truck: 95%

When you launch your service -- suddenly -- it doesn't work on **horses**.

(Everything else works well, as before...)

Discussion: what on earth could cause this bug? Things looked good with train / val / test...



Accuracy as reported by users

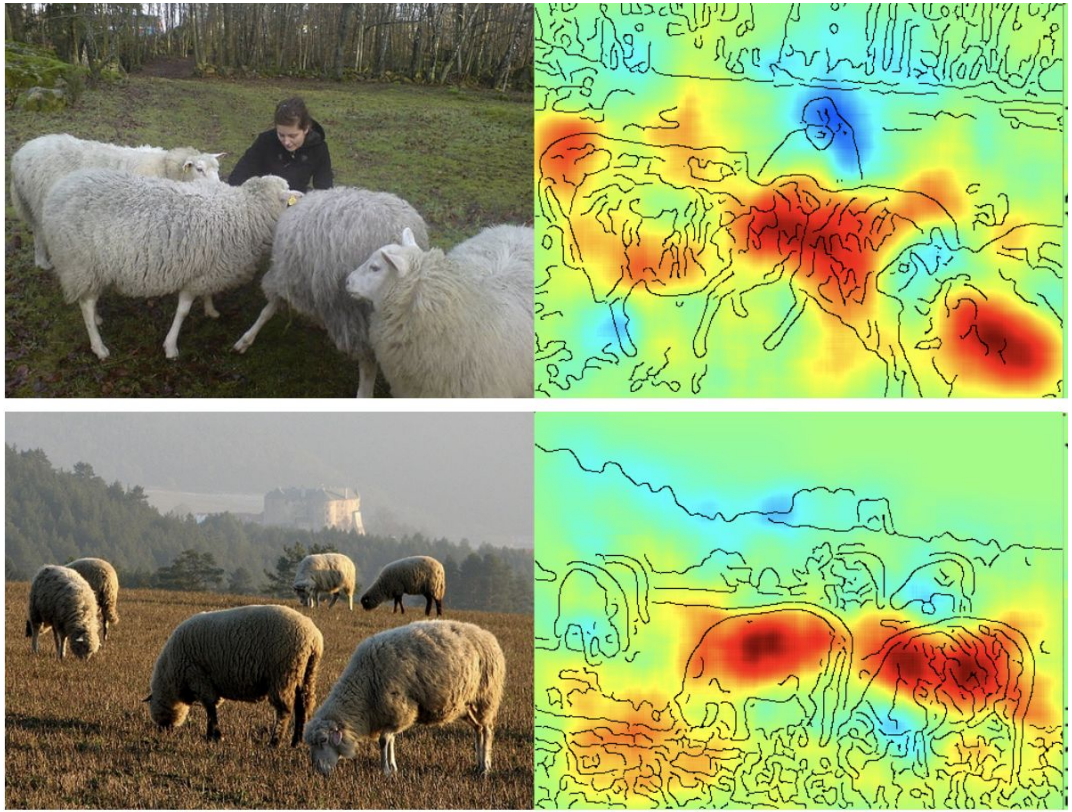
Dogs: 95%

Frogs: 95%

Horse: 0%

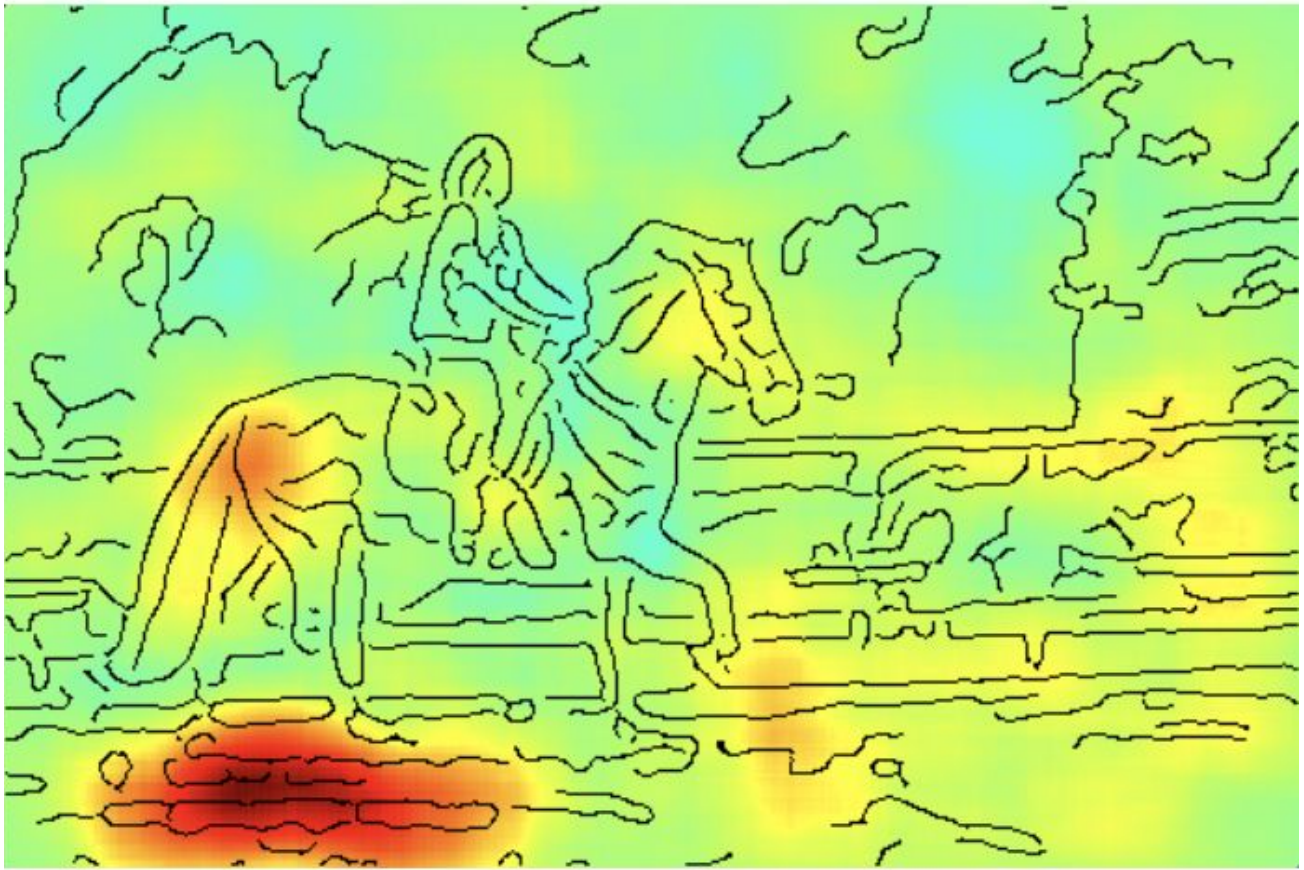
Ship: 95%

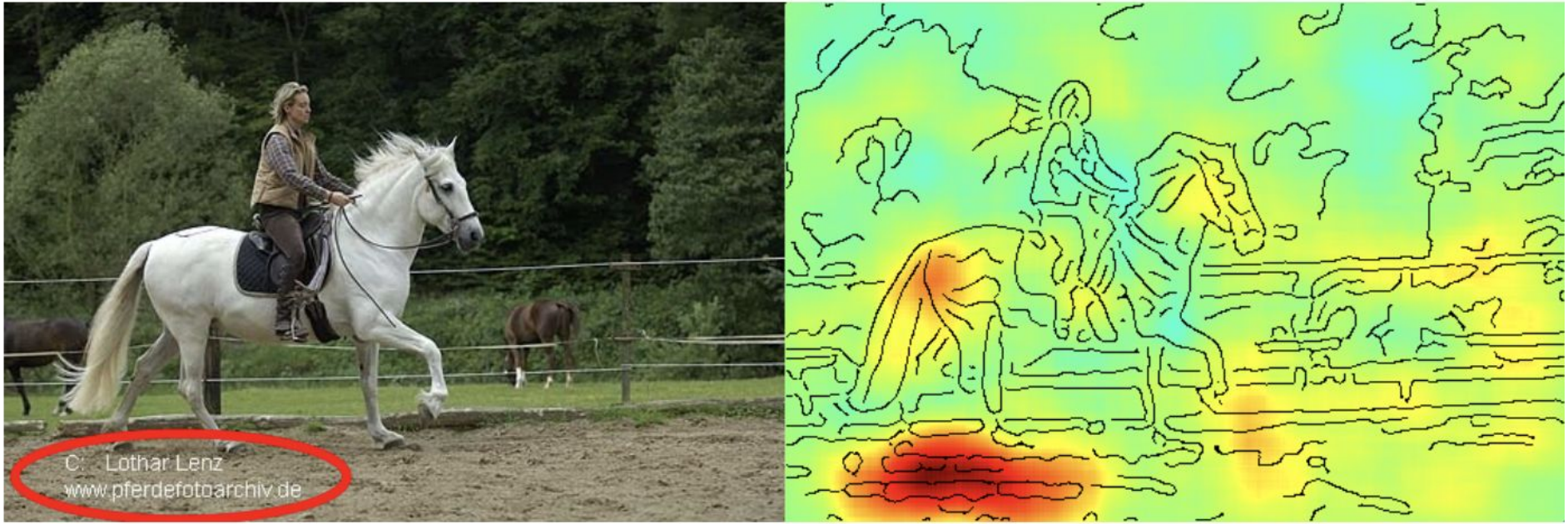
Truck: 95%



[Analyzing Classifiers: Fisher Vectors and Deep Neural Networks] (not suggesting you read this one right now, just an FYI with a great example).

Why might this region be important?!





All the images in your dataset were collected by the same photographer: causing the model to condition on the copyright.

Following examples are more serious.

Why?

Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
Microsoft	94.0%	79.2%	100%	98.3%	20.8%
FACE++	99.3%	65.5%	99.2%	94.0%	33.8%
IBM	88.0%	65.3%	99.7%	92.9%	34.4%

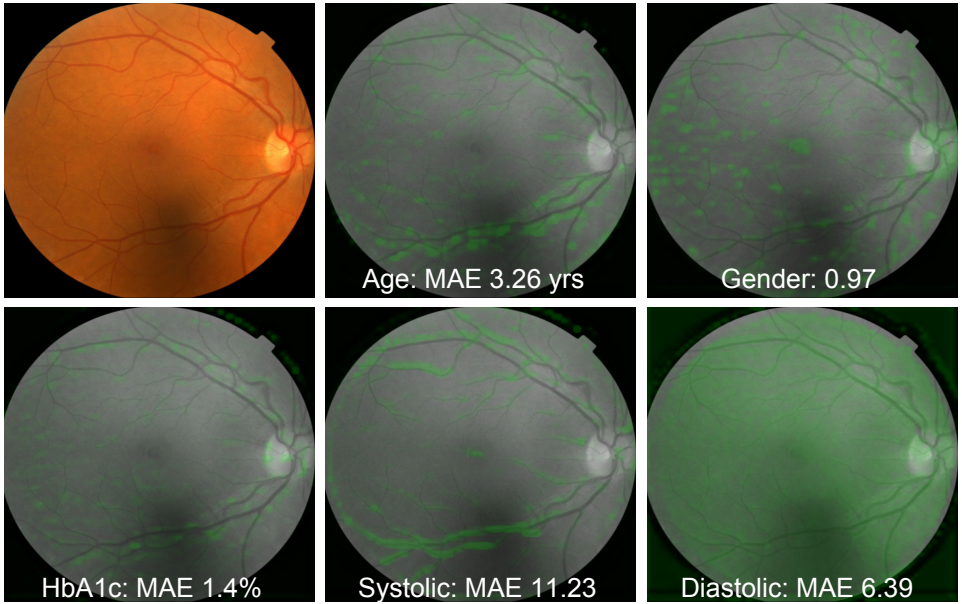
Quick discussion: why might this be happening?

Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification

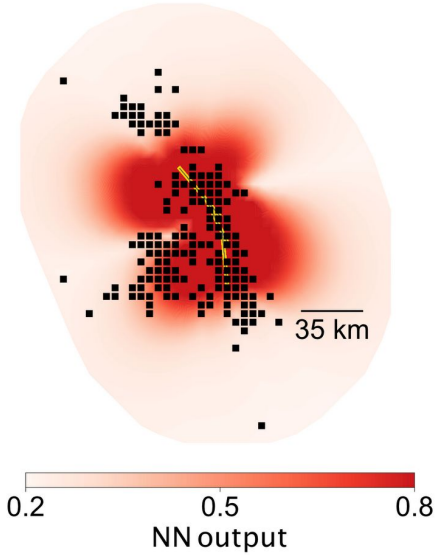
Misunderstanding DL can lead to unsupported & harmful claims

Last week we saw models supporting basic science. Here's the reverse.

Last time: Supporting basic science



R. Poplin, A. Varadarajan *et al.* Predicting Cardiovascular Risk Factors from Retinal Fundus Photographs using Deep Learning. *Nature Biomedical Engineering, 2018.*



Forecasted distribution of aftershock location probabilities for the Landers earthquake. Dark red colors indicate regions predicted to experience aftershocks. Black dots are the locations of observed aftershocks. Yellow line shows the faults that ruptured during the mainshock.

[Forecasting earthquake aftershock locations with AI-assisted science]

Cover of the economist from **Sep 9th 2017** (about one year ago)

Echos of physiognomy and phrenology are sadly back, due to (in my opinion) a basic misunderstanding of what classifiers are / how they work / what their limitations are / and how to carefully design & evaluate an experiment.

Cover of the economist from **Sep 9th 2017** (about one year ago)

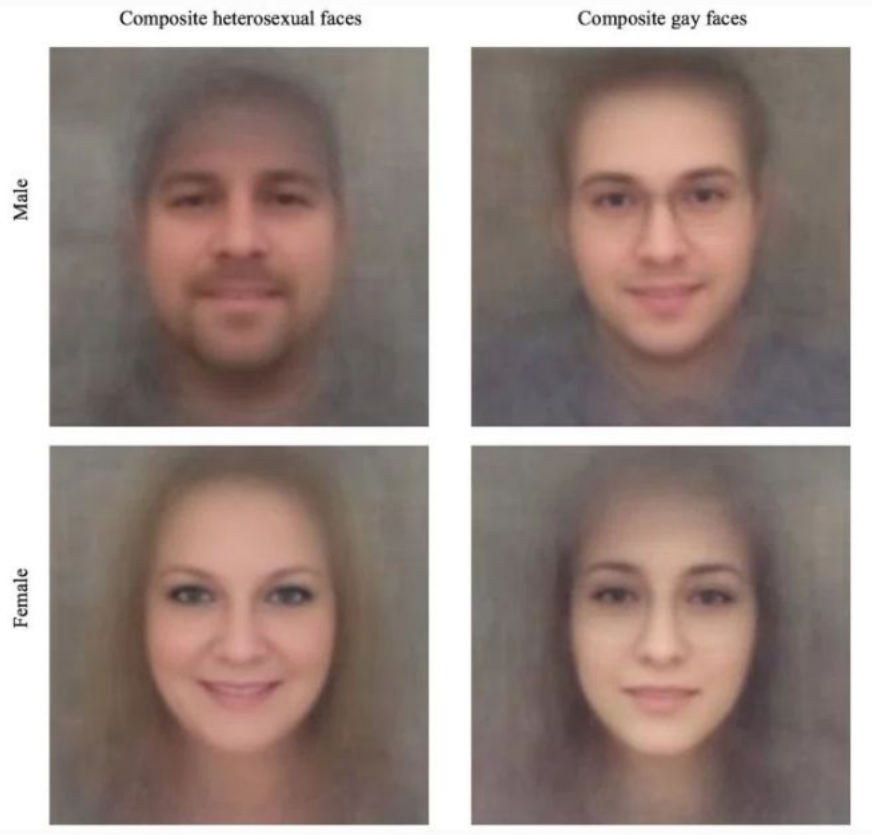
From the paper below

“We show that faces contain much more information about sexual orientation than can be perceived and interpreted by the human brain.

We used deep neural networks to extract features from 35,326 facial images [...] these features were entered into a logistic regression aimed at classifying sexual orientation [...]

Those findings **advance our understanding of the origins of sexual orientation** and the limits of human perception.” ???!

Deep neural networks are more accurate than humans at detecting sexual orientation from facial images.



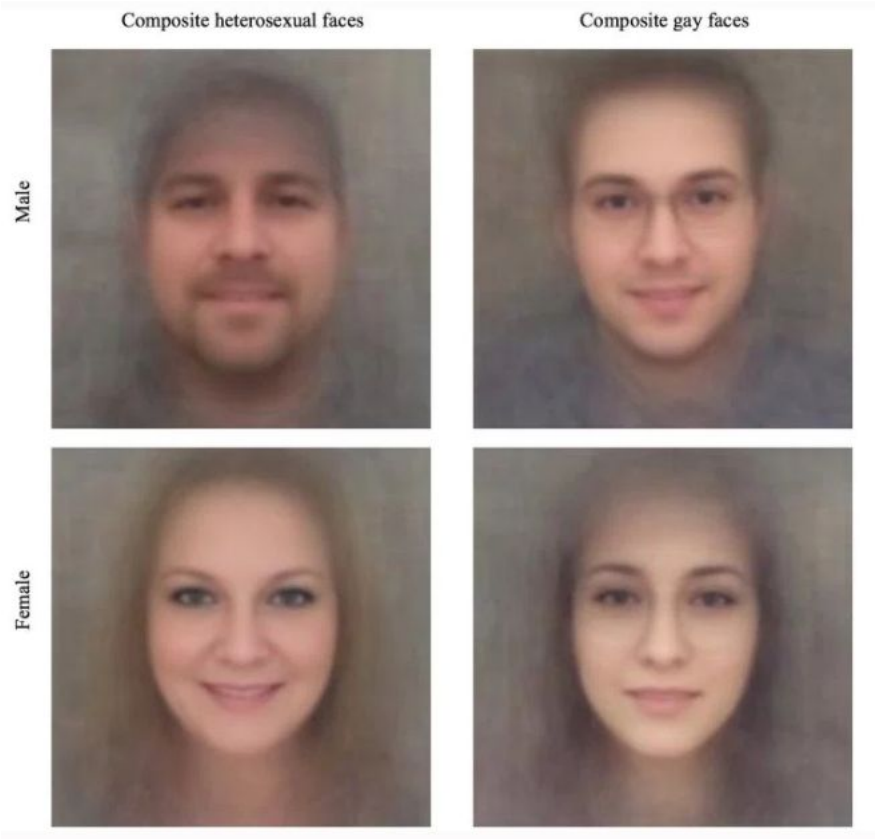
From a rebuttal

"The authors assert that the key differences are in physiognomy, meaning that a sexual orientation tends to go along with a characteristic facial structure.

However, we can immediately see that some of these differences are more superficial…"

Quick discussion: What do you see?

[Do algorithms reveal sexual orientation or just expose our stereotypes?]



Important variables include:

- **Makeup**
- **Eyeshadow**
- **Facial hair**
- **Glasses**
- **Selfie angle**
- **Amount of sun exposure**

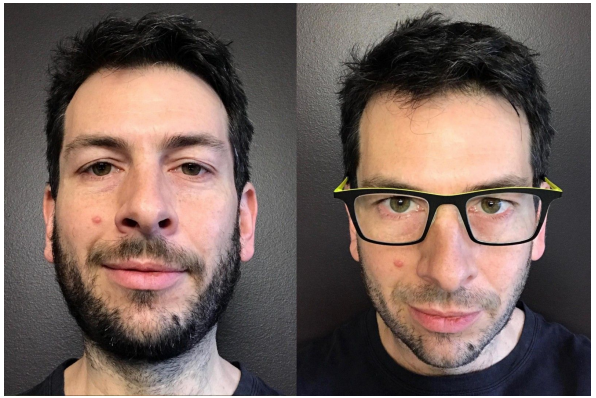
Yes/no questions about these do nearly as well at guessing orientation as the supposedly sophisticated software.

[Do algorithms reveal sexual orientation or just expose our stereotypes?]

Those findings ~~advance our understanding of the origins of sexual orientation and the limits of human perception.~~”

Or maybe they just tell us if someone is wearing glasses...

Do algorithms reveal sexual orientation or just expose our stereotypes?
Deep neural networks are more accurate than humans at detecting sexual orientation from facial images.



[Do algorithms reveal sexual orientation or just expose our stereotypes?]

Always a bad sign...

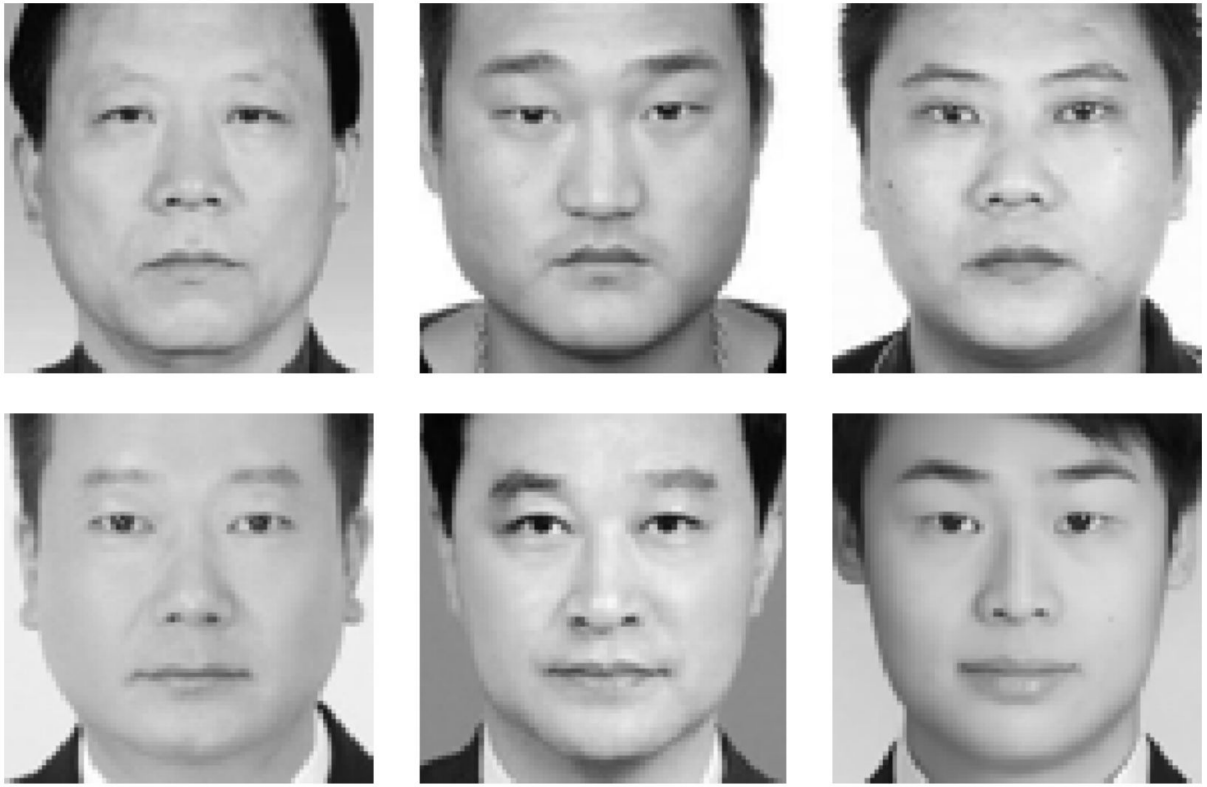
Quick discussion

Another example.

The authors only shared six images...

Do you see anything they may not have controlled for?

Criminals on the top row.



Automated Inference on Criminality using Face Images

Physiognomy's New Clothes

Always a bad sign...

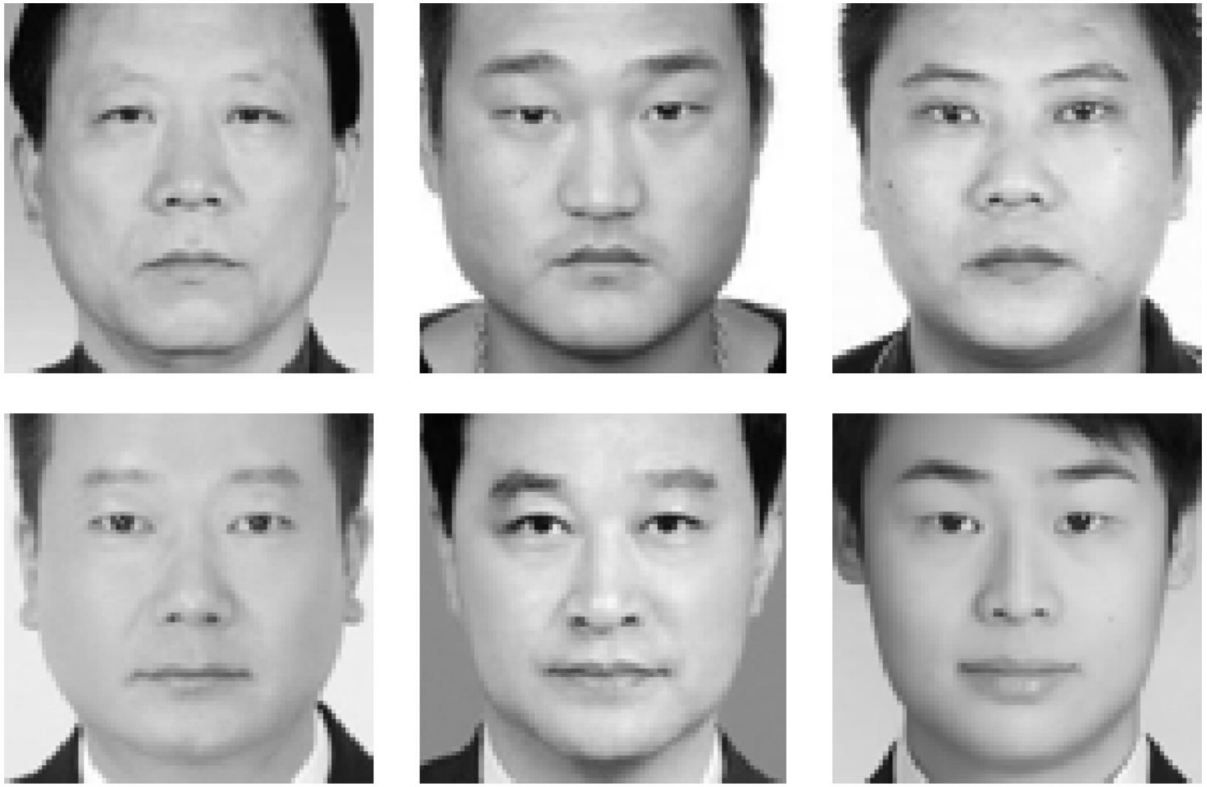
Quick discussion

Another example.

The authors only shared six images...

Do you see anything they may not have controlled for?

Criminals on the top row.



Frowning

Smiling

[Automated Inference on Criminality using Face Images]
[Physiognomy's New Clothes]

Incidents

Good intentions with bad results, some of which could have been avoided with careful design.

Tae

Quick discussion: what could go wrong? How long might it take?

Canary

A simple and useful technique

Google photos incident

- **Concretely**, can you suggest one reason why this may have happened?

- Can you suggest one course of action to **prevent it**? What are the costs involved? How likely is it to work?

Google apologises for Photos app's racist blunder

Google photos incident

- **Concretely**, can you suggest one reason why this may have happened?

All these are hypothetical

  - Insufficient training data; training data not sufficiently diverse; noisy training data (say images were scrapped from web and labels were determined automatically by capturing surrounding text); etc.

- Can you suggest one course of action to **prevent it**? What are the costs involved? How likely is it to work?

  - Larger test set, manually validate all training data, ?

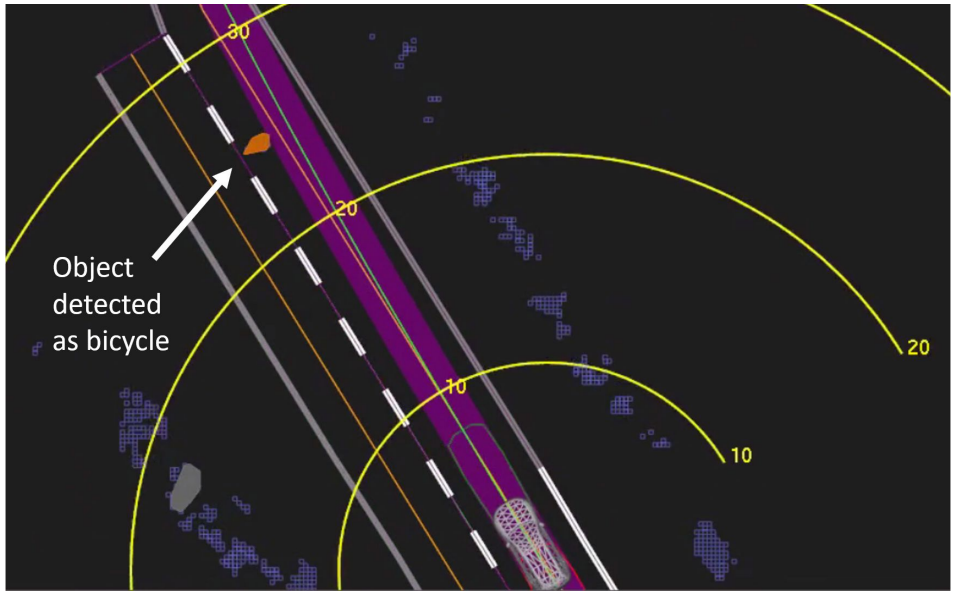
Google apologises for Photos app's racist blunder

March 18, last year.

The first self-driving car fatality, Uber.

As the vehicle and pedestrian paths converged, the self-driving system software classified the pedestrian as an unknown object, as a vehicle, and then as a bicycle.

NTSB Preliminary Report



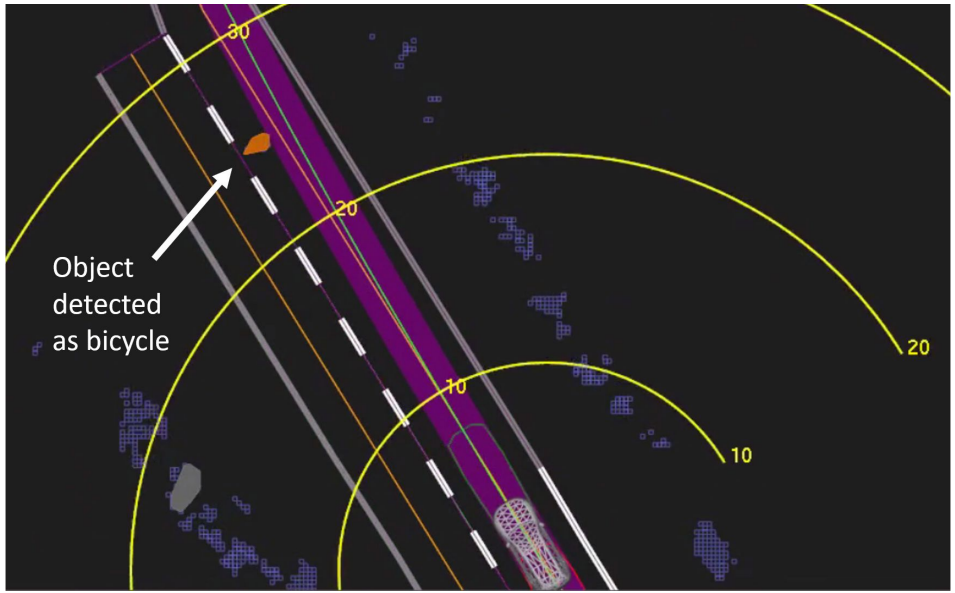
This Uber self-driving system data playback from the fatal, March 18, 2018, crash of an Uber Technologies, Inc., test vehicle in Tempe, Arizona

When you start playing with image classifiers on the web, you'll see erratic results you move the camera around.

The first self-driving car fatality, Uber.

NTSB Preliminary Report

- System determined that emergency braking was needed 1.3 seconds before impact.
- But, emergency braking maneuvers were **disabled** while vehicle was under computer control (to reduce potential for erratic behavior).
- Vehicle operator was relied on to take action. System wasn't designed to alert the operator.



This Uber self-driving system data playback from the fatal, March 18, 2018, crash of an Uber Technologies, Inc., test vehicle in Tempe, Arizona

Modeling is the “easy” part. It’s everything surrounding it that’s hard.

Workflow

1. Concretely, what you will predict, and how will you predict it?

2. What is the simplest non-ml baseline (rules or “if statements”)? (This may stronger than you think if working with structured data!)

3. Where will the data come from? How much will you need? How do you know? How will you collect it?

4. How will you evaluate your experiment? What metrics do you care about? What scores are good enough?

Modeling is the “easy” part. It’s everything surrounding it that’s hard.

5. How will your model be used in practice?

6. Begin by using a simple interpretable model (a decision tree or logistic regression).

7. Consider a more sophisticated but less interpretable model.

Obviously, data changes with time

Quick discussion: Say you've developed a model to recommend movies. It works well on your test set. Will it work well a week from now? A year? What could go wrong?

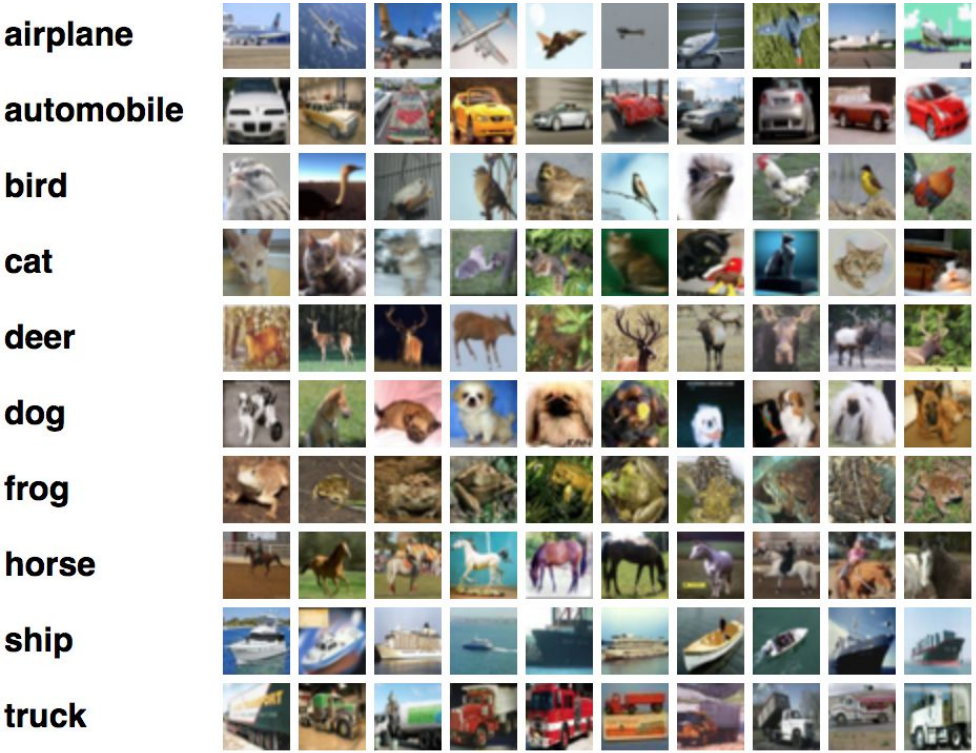
Obviously, data changes with time

Quick discussion: Say you've developed a model to recommend movies. It works well on your test set. Will it work well a week from now? A year? What could go wrong?

The one I would have missed: user preferences change, too, as time goes by.

CIFAR-10

CIFAR-10



- 60,000 32x32 color images
- 10 classes
- Balanced (6k images / class).
- Single label per image.
- 50k train, 10k test.

The CIFAR-10 dataset

A closer look at an few images

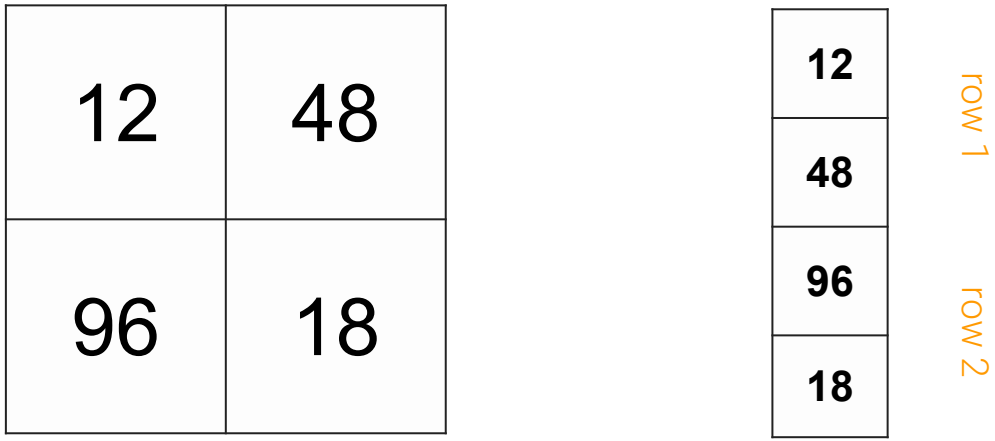
plane

bird

car

Unrolled into an array

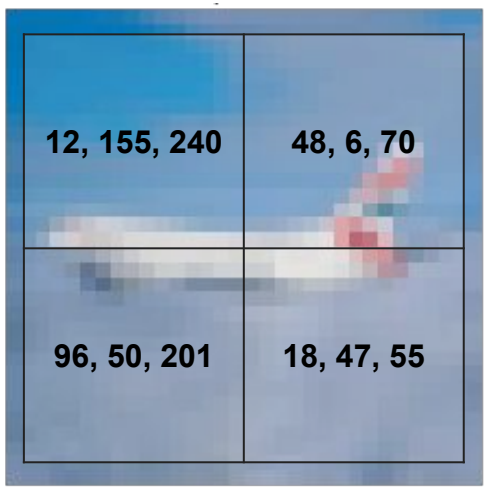
Later, techniques like convolution will allow us to work with 2d structure directly. For now, we'll unroll.



Quick discussion: how many elements in an array would you need to represent a 32x32 pixel color image from CIFAR-10?

Three color channels (R,G,B)

An image may be unrolled by channel, or by pixels.



12	155	240	48	6	70	...

Unrolling by pixels

12	48	96	18	155	6	...

Unrolling by channels

An image from CIFAR-10 is 32x32x3 (three color channels: R,G,B), so the complete unrolled array would have 3072 elements.

Quick discussion: Does it matter which way you unroll when working with a linear model?

Linear classifiers

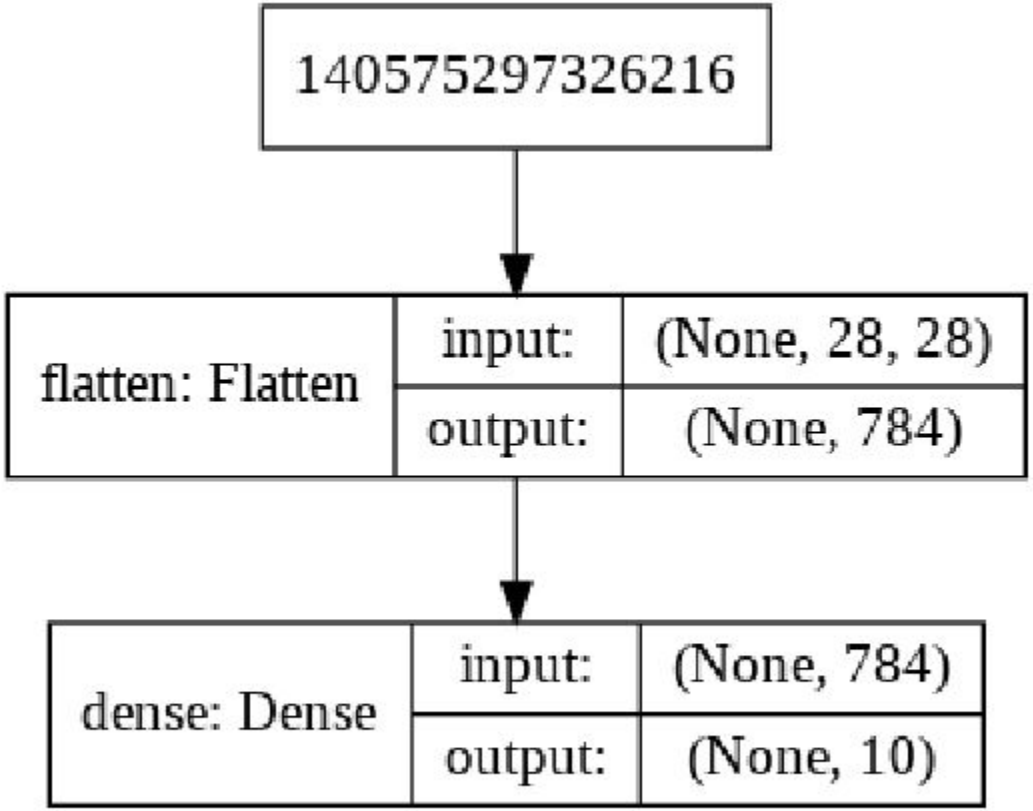
```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

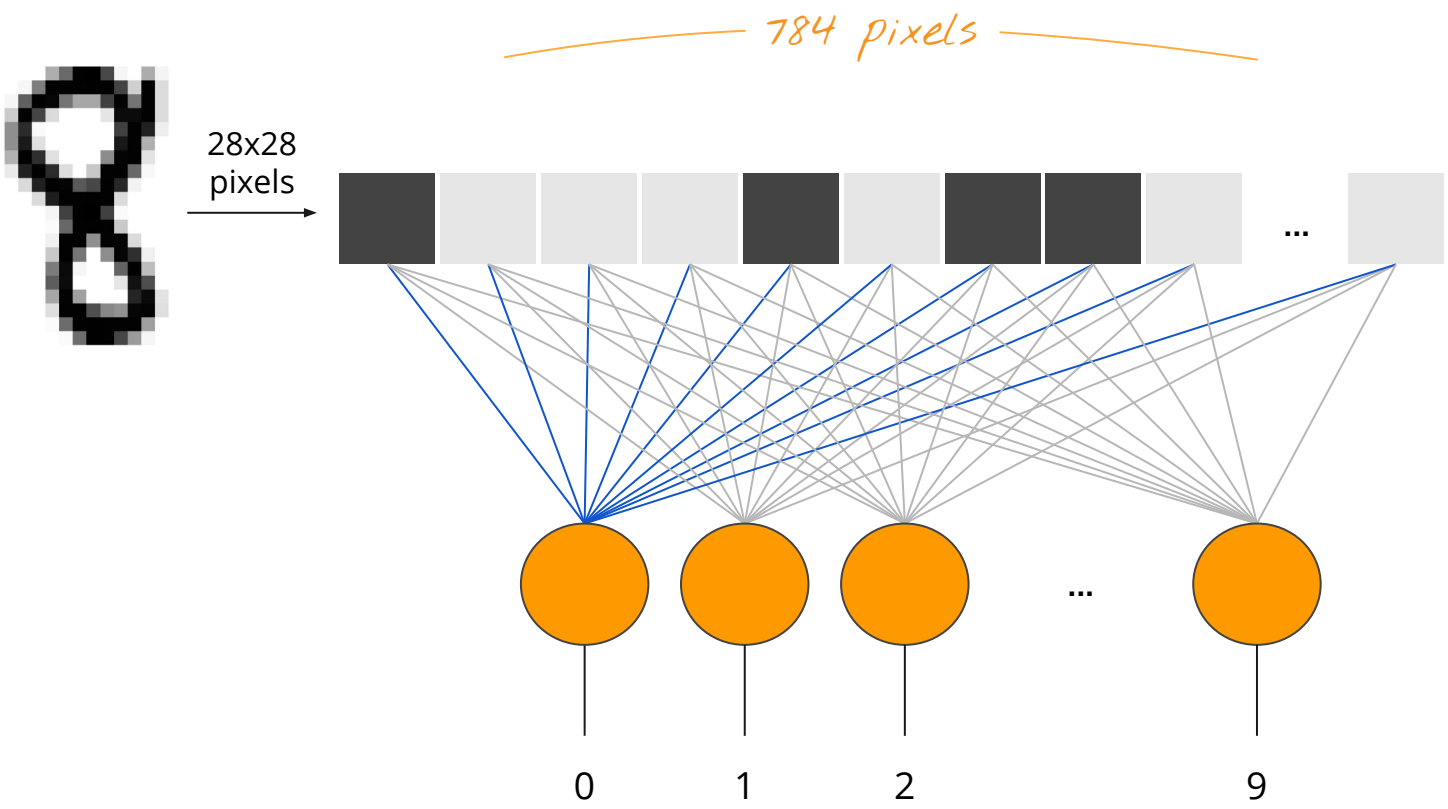
```
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
```

Code linked from the HW

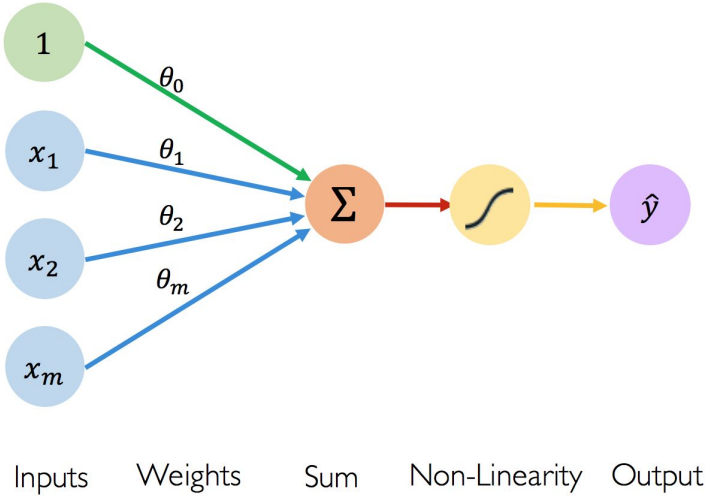
Should also work copy/pasting to Colab along w/ previous example.

```
# You need to install pydot and graphviz for this utility to work.

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
image_path = 'model.png'
tf.keras.utils.plot_model(model, show_shapes=True, to_file=image_path)
plt.figure(dpi=100)
imgplot = plt.imshow(mpimg.imread(image_path))
plt.axis('off')
plt.show()
```


140575297326216
flatten: Flatten
input:
(None, 28, 28)
output:
(None, 784)
dense: Dense
input:
(None, 784)
output:
(None, 10)


784 pixels
28x28
pixels
...
...
0
1
2
9

One neuron (basically, a logistic regression unit)



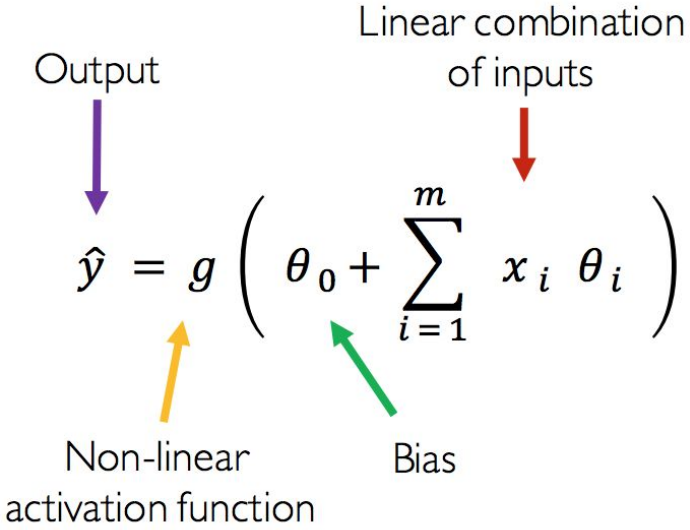


Diagram borrowed from MIT with permission.

Ingredients to train a DNN

Today

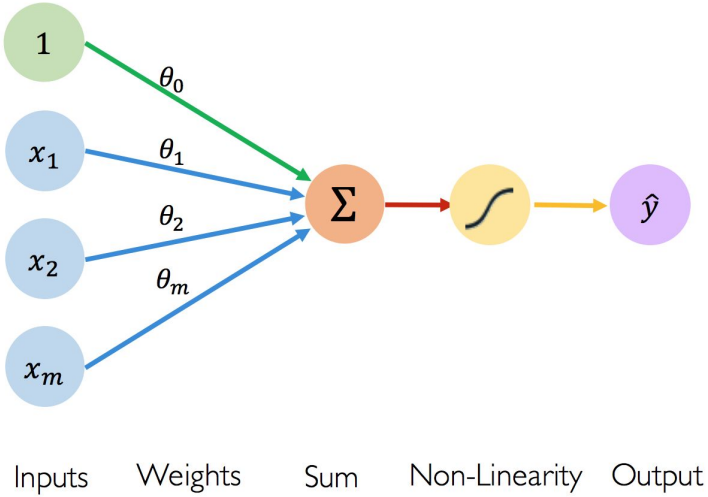
- A score function (forward pass)
  - Given an image, predict whether it's a plane, bird, or car.

Today

- A loss function
  - Given your prediction, quantify how wrong it is.

- An update function (backward pass)
  - Update your model's weights to reduce the loss.

Forward pass



$$\hat{y} = g\left(x^T\theta\right)$$

Forward propagation is a **dot product** followed a **non-linearity.**

Diagram borrowed from MIT with permission.

We know that linear models are building blocks for DNNs

$$f = Wx$$ Linear model

$$f = W_2(g(Wx))$$ Neural network

$$f = W_3(g(W_2(g(Wx))))$$ Deep neural network

We know that linear models are building blocks for DNNs

$$f = Wx$$

Let's start with this

Linear model

$$f = W_2(g(Wx))$$

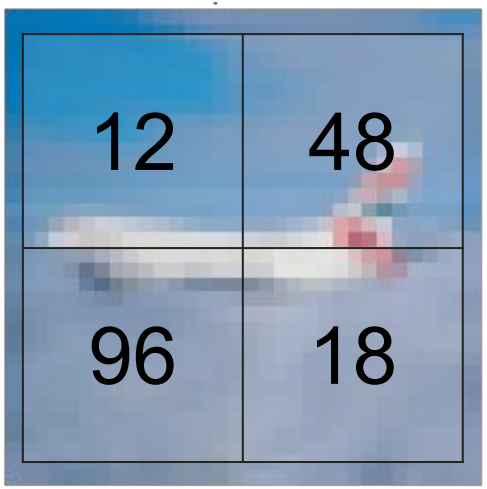
Neural network

$$f = W_3(g(W_2(g(Wx))))$$

Deep neural network

Example: one image and one class

Interpret as “how strongly do you think this image is a plane?”



1.4	0.5	0.7	1.2

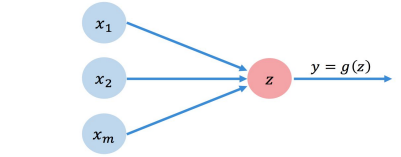
12
48
96
18

+

0.5

=

130.1	Plane



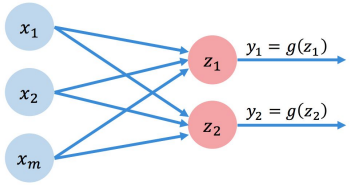
w Weights

x Inputs

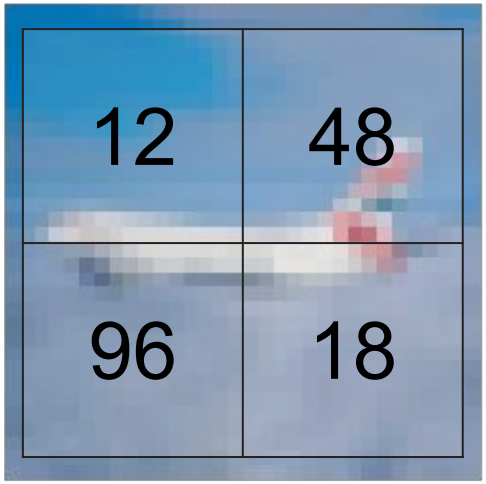
b Bias

Output Scores

Example: one image and two classes



Multiple inputs, multiple outputs



1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7

12
48
96
18

+

0.5
1.2

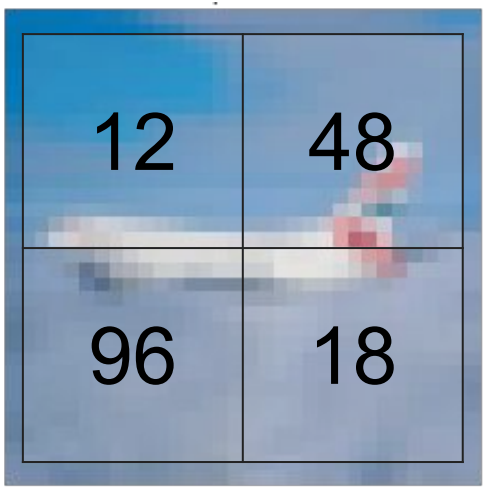
=

130.1	Plane
-11.4	Car

W is now a matrix

W	x	b	Output
Weights	Inputs	Bias	Scores

Example: one image and three classes



1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7
0.2	0.9	-0.2	0.5

W

Weights

12
48
96
18

x

Inputs

+

0.5
1.2
0.2

b

Bias

=

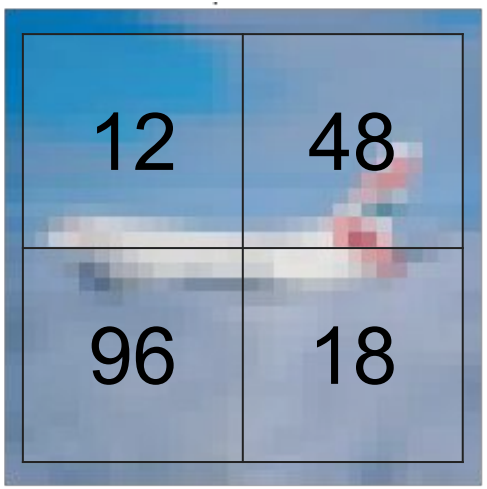
130.1	Plane
-11.4	Car
12.8	Truck

Output

Scores

np.matmul(W,x) + b # b is added by broadcasting

Example: one image and three classes



3x4

1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7
0.2	0.9	-0.2	0.5

W

Weights

4x1

12
48
96
18

x

Inputs

+

3x1

0.5
1.2
0.2

b

Bias

=

3x1

130.1	Plane
-11.4	Car
12.8	Truck

Output

Scores

Example: two images and three classes

Good news! Math doesn't change.

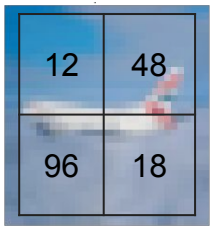


Image 1

Image 2

N x D

1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7
0.2	0.9	-0.2	0.5

W

Weights

D x batch_size

12	4
48	18
96	2
18	96

x

Inputs

+

N x 1

0.5
1.2
0.2

b

Bias

=

N x batch_size

Image 1	Image 2	
130.1	131.7	Plane
-11.4	-71.7	Car
12.8	64.8	Truck

Output

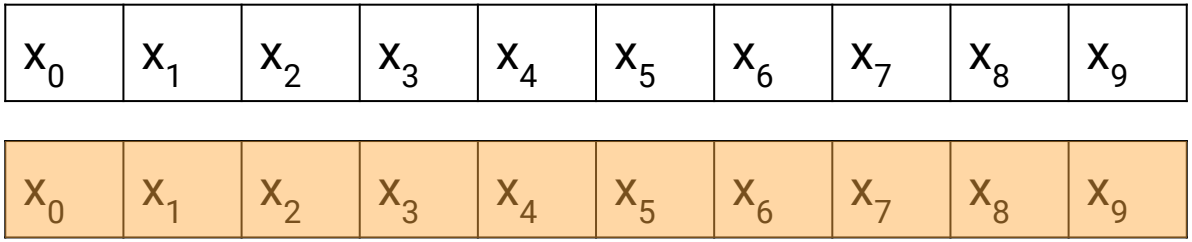
Scores

Epochs and batches

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

Say this is all of our training data

Epochs and batches

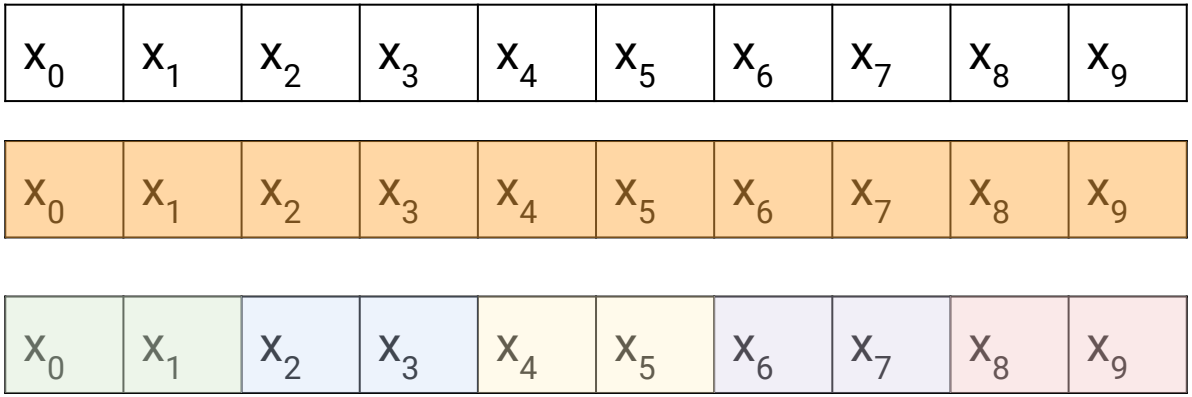


Say this is all of our training data

An 'epoch'

An 'epoch' is one complete pass, or sweep over the training data. It means we've used each example once, say, in the forward and backward pass.

Epochs and batches



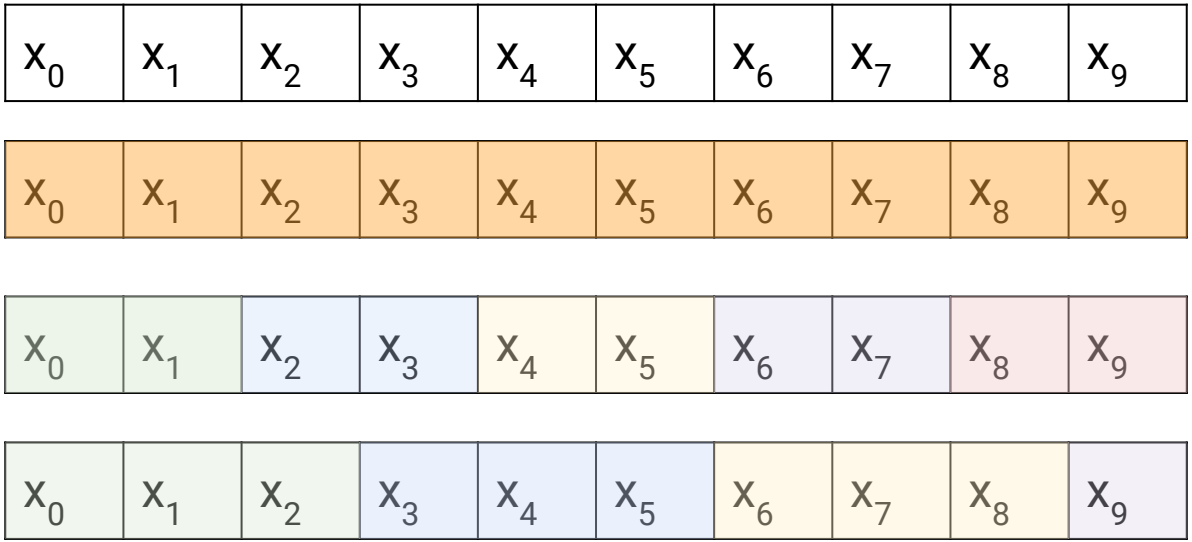
Say this is all of our training data

An 'epoch'

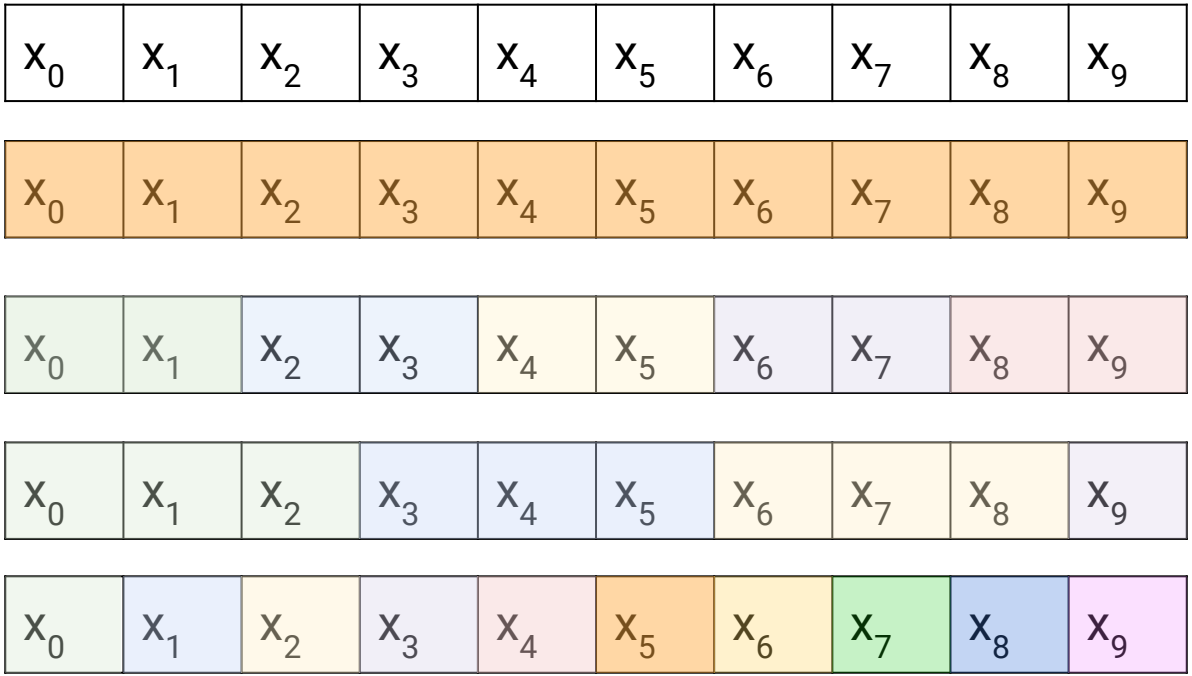
Batch size of 2

Batches, each of a non-overlapping chunk of the data.

Epochs and batches



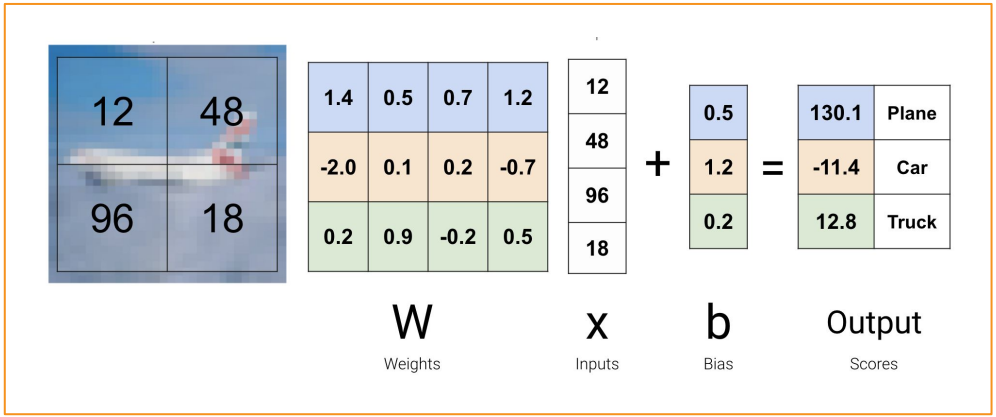
Epochs and batches



This is a Dense layer. It's the most common type.

$$f = W_2\, g(\boxed{W x})$$

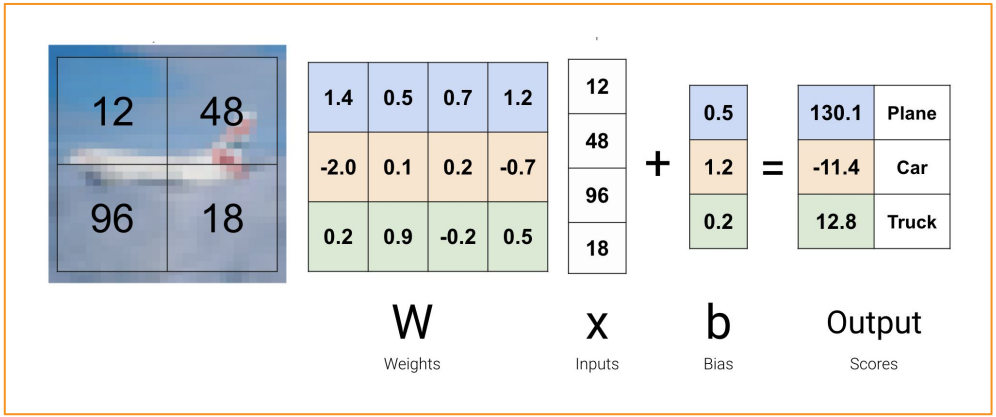
A neural network (a dense layer, followed by a non-linear activation function, followed by a dense layer.



This is a Dense layer. It's the most common type neural networks.

$$f = W_2 g(Wx)$$

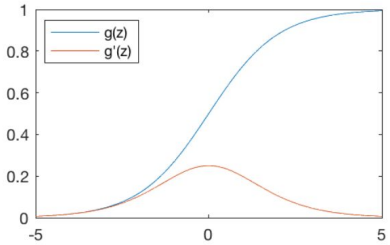
After we apply the nonlinearity, the result becomes the input to the next layer.



Many activation functions

$$f = W_2 g(Wx)$$

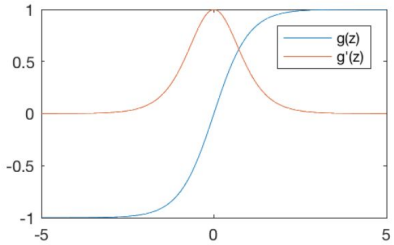
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

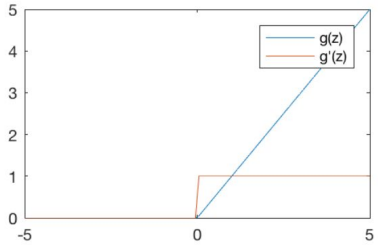
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

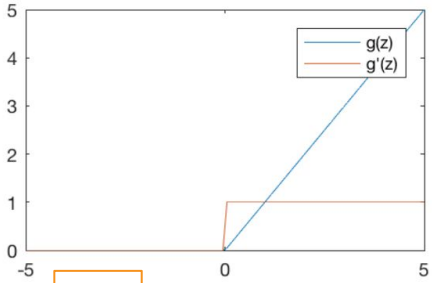
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

All are applied piecewise.

Some (like softmax) compute statistics about the distribution of their inputs, first

130.1	Plane
-11.4	Car
12.8	Truck

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

z here refers to our Wx

g(130.1)	Plane
g(-11.4)	Car
g(12.8)	Truck

=

?	Plane
?	Car
?	Truck

Output

Scores

$$f = W_2 g(Wx)$$

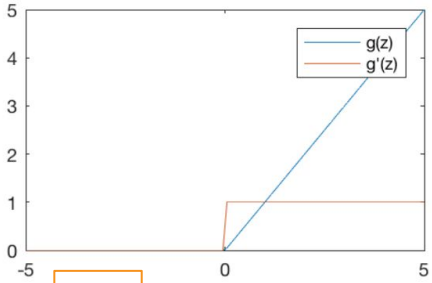
Why look! We're halfway there on the forward pass our of NN

All are applied piecewise

Some (like softmax) compute statistics about the distribution of their inputs, first

130.1	Plane
-11.4	Car
12.8	Truck

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

z here refers to our Wx

g(130.1)	Plane
g(-11.4)	Car
g(12.8)	Truck

=

130.1	Plane
0	Car
12.8	Truck

Output

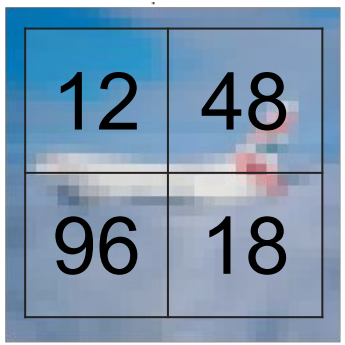
Scores

$$f = W_2 g(Wx)$$

Why look! We're halfway there on the forward pass our of NN

If this was an intermediate layer, instead of producing scores for classes, we'd be producing scores for learned features. Otherwise, it's the same.

Features (or representations) computed by the layer



1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7
0.2	0.9	-0.2	0.5
...	...	...	...
...	...	...	...
...	...	...	...

W

Weights

12
48
96
18

x

Inputs

+

0.5
1.2
0.2
...
...
...

b

Bias

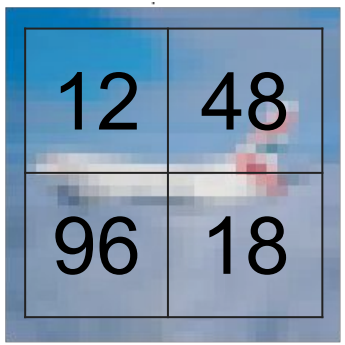
=

130.1	f1
-11.4	f2
12.8	f3
...	...
...	...
...	...

Output

Scores

The math is the same (it's still just a matmul followed by an additional). You can probably guess already that wrangling with dimensions is important in debugging.



6x4

1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7
0.2	0.9	-0.2	0.5
...	...	...	...
...	...	...	...
...	...	...	...

W

Weights

4x1

12
48
96
18

x

Inputs

+

6x1

0.5
1.2
0.2
...
...
...

b

Bias

=

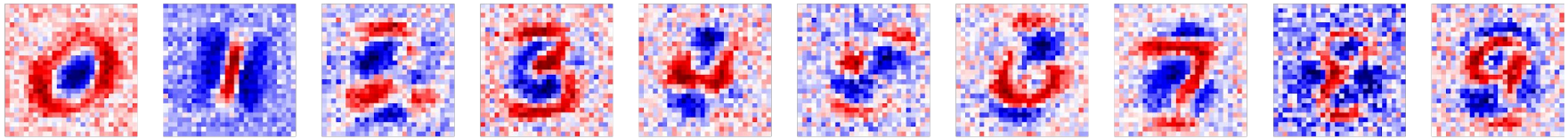
6x1

130.1	f1
-11.4	f2
12.8	f3
...	...
...	...
...	...

Output

Scores

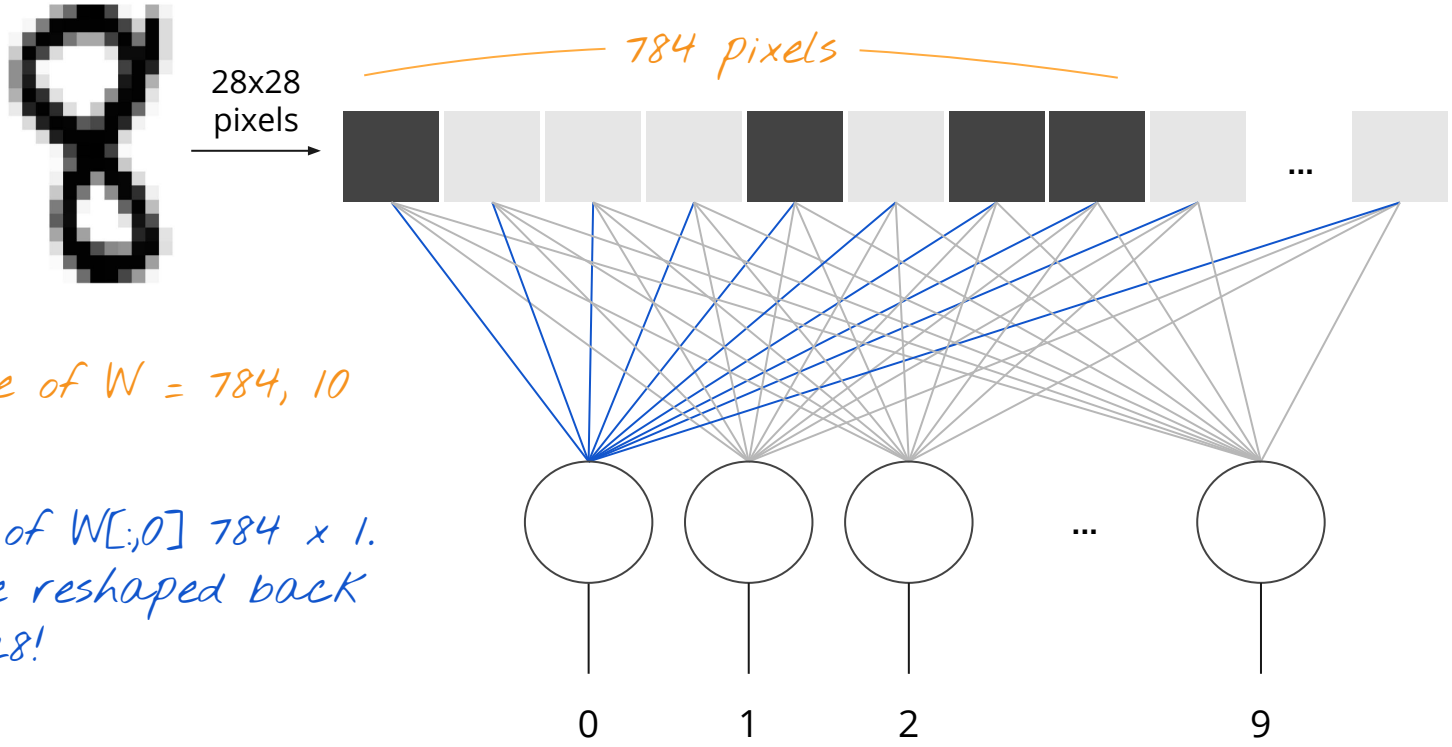
EC3 :Visualizing the weights learned by a linear classifier



Discussion:

- What do you see here?
- Why is this behavior occurring?

Trivia: the original TensorFlow MNIST tutorial gave intuitution for this, but didn't quite show how to reproduce the diagrams. We're now pushing for completely **reproducible** experiments and examples, with **end-to-end** code, data, and settings shared.



Shape of W = 784, 10

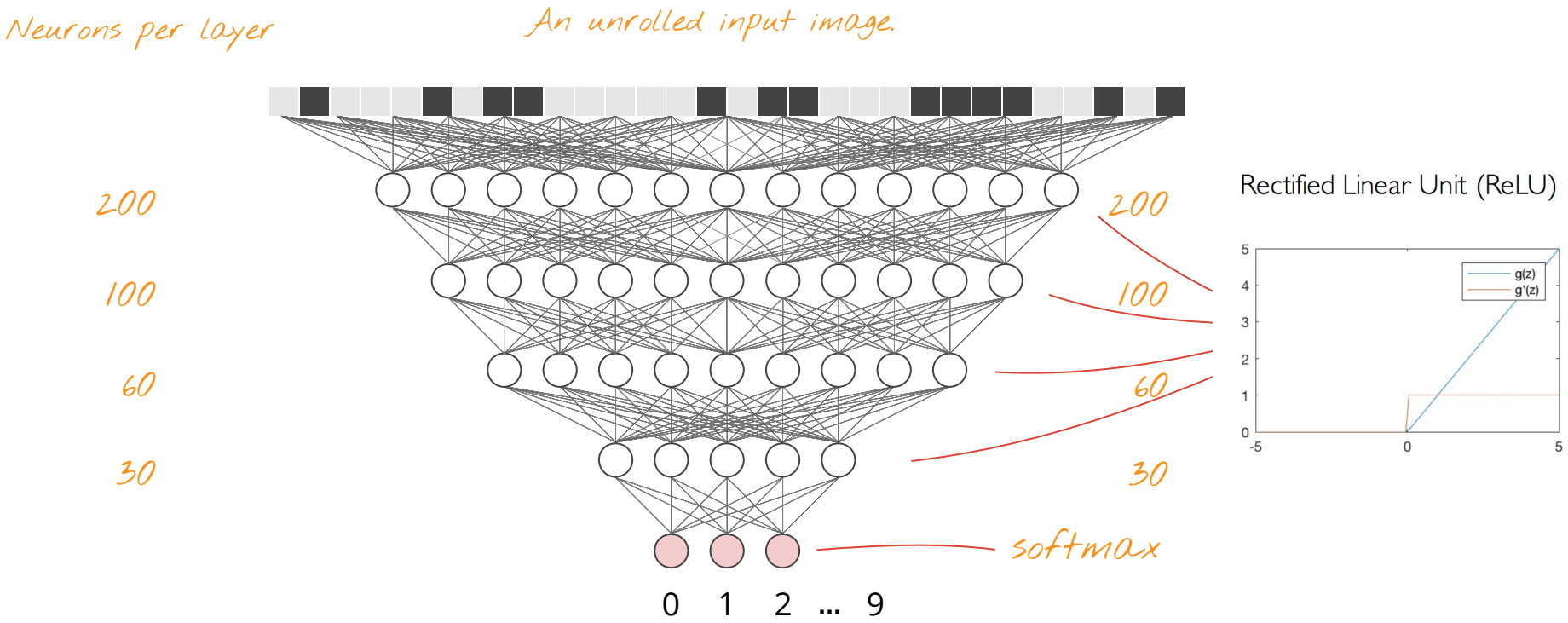
Shape of W[:,0] 784 x 1.
Can be reshaped back
to 28x28!

Idea

```
# Initialize weights to small values
# 784 (number of pixels on unrolled 28x28 images)
# 10 (number of classes, one for each digit)
W = 0.01 * np.random.randn(784, 10)

# Train the model
# ...

# Select the weights connected to the '0' output
# Reshape them into an 28x28 image
im = W[:,0].reshape(28,28)
plt.imshow(im, cmap = plt.get_cmap('seismic'))
```

Seismic is a 'diverging' colormap, low values will be in blue, high values in red

Graphical view



Shapes at each layer

Batch size of 1

Input shape 1 x 784

W1 = 784 x 200

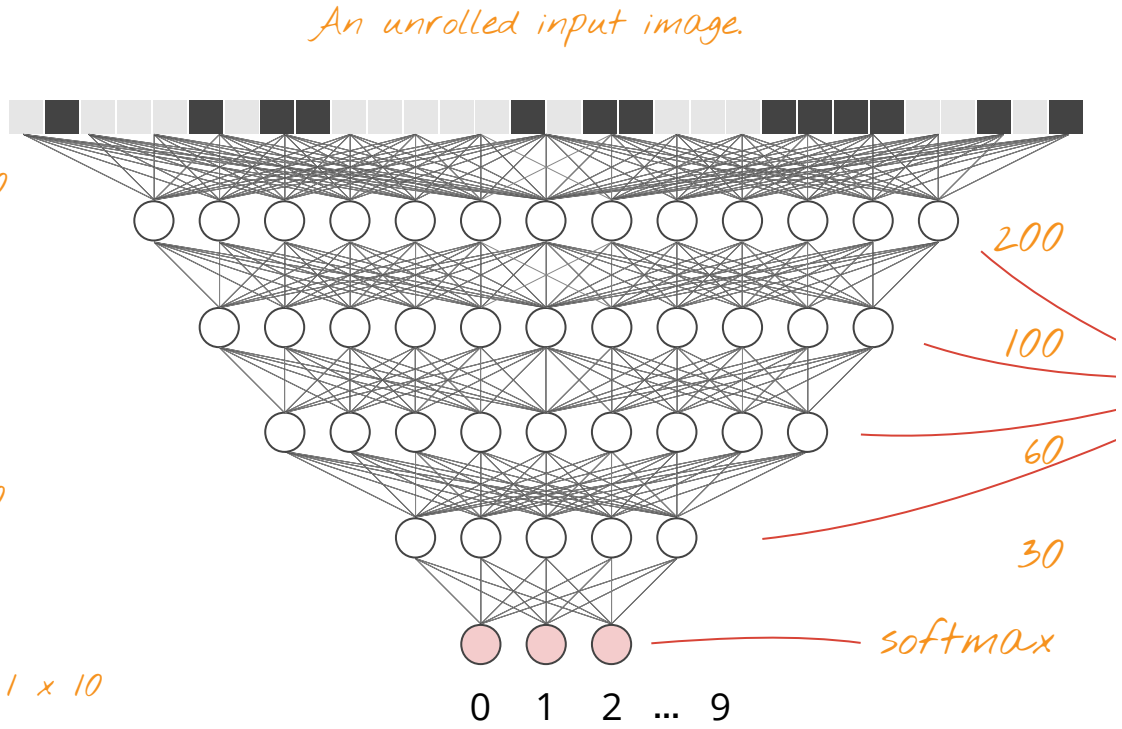
W2 = 200 x 100

W3 = 100 x 60

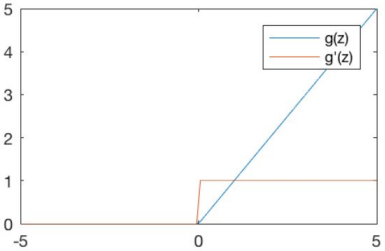
W4 = 60 x 30

W5 = 30 x 10

Output shape 1 x 10



Rectified Linear Unit (ReLU)



Number of parameters

Batch size of 1

Input shape 1 x 784

W1 = 784 x 200

W2 = 200 x 100

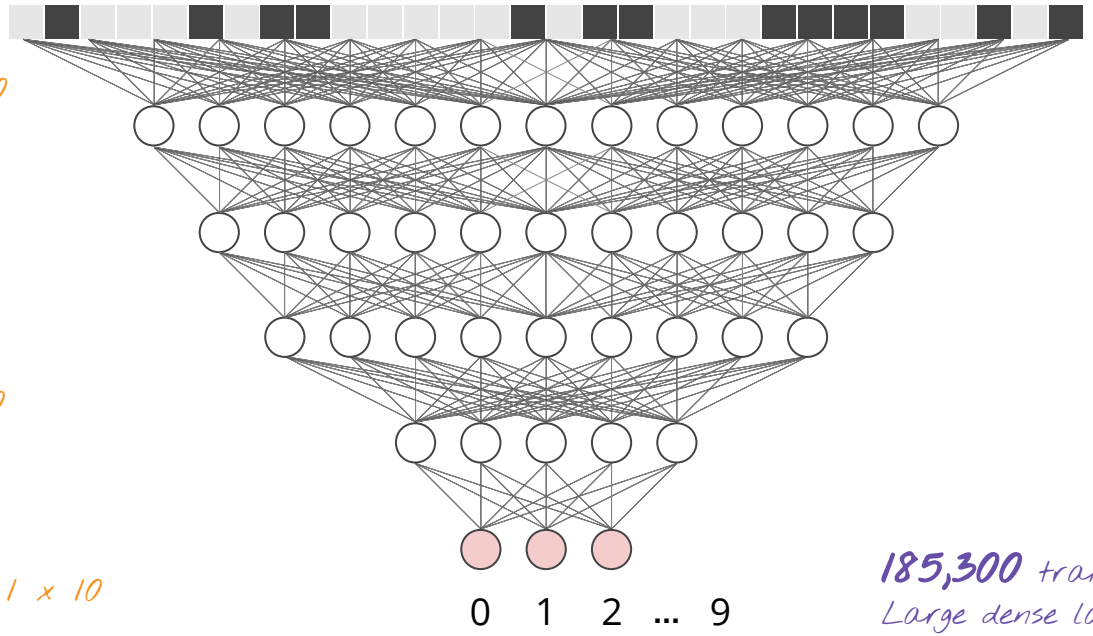
W3 = 100 x 60

W4 = 60 x 30

W5 = 30 x 10

Output shape 1 x 10

An unrolled input image.



784 * 200 + 200 (for the bias) = 157000

20100

6060

1830

310

185,300 trainable parameters

Large dense layers are expensive. Notice the first contains ~85% of the total).

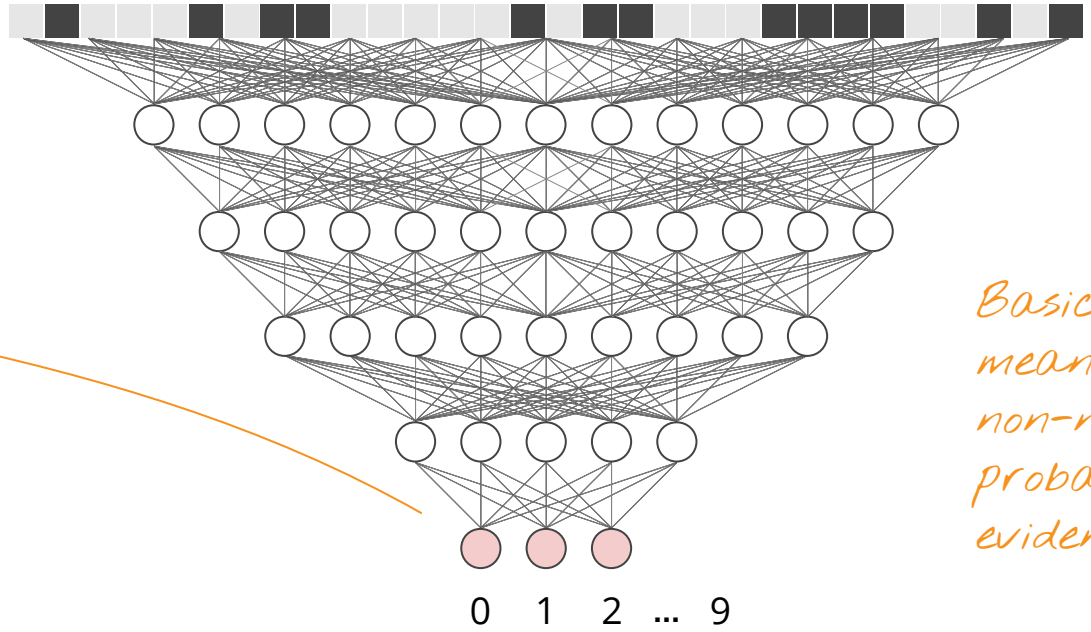
Comparison

y = mx + b. 2 trainable parameters

Modern NNs may have +/- 10^8

Terminology: Logits

The final output of the network just before applying the softmax function



Basically, we mean a non-normalize probability, or evidence score

A DNN (equation view). *The onion.*

$$y = softmax(W_5(g(W_4(g(W_3(g(W_2(g(Wx)))))))))$$

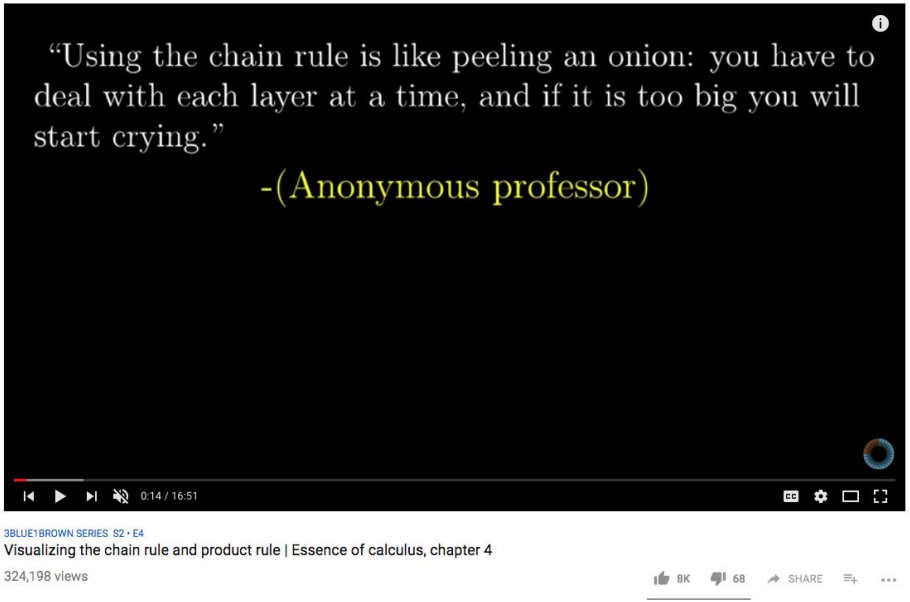
So far, we know how to compute:

$$y = g(Wx)$$

This becomes next input to the next layer, and you can iterate from there for the full forward pass.

If you need a **calculus refresher** (say, to understand the math for backprop)

Recommend the Essence of Calculus series. It includes thoughtful, animated explanations (which may feel more intuitive than hacking away on derivatives symbolically).

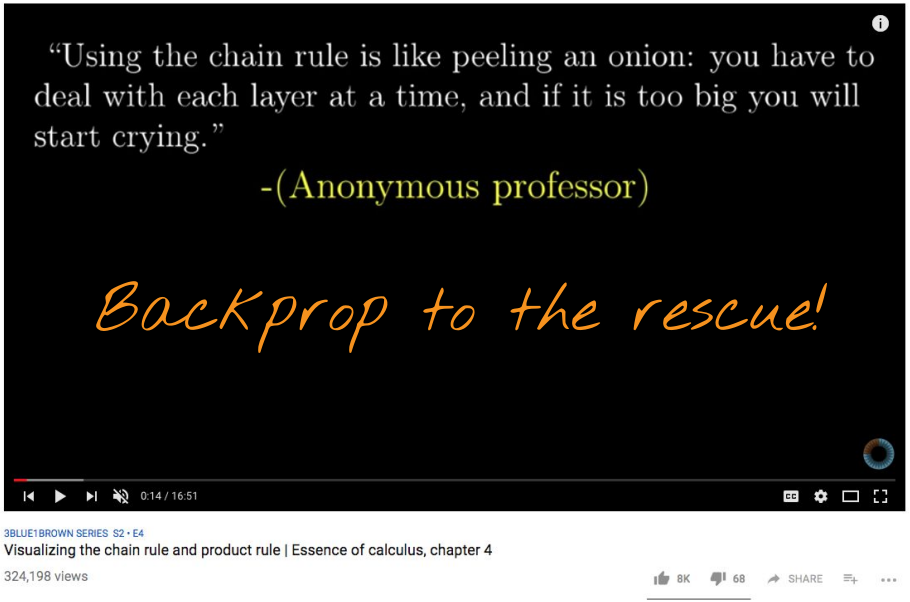


The chain rule is important to peel this onion.

$$y = softmax(W_5(g(W_4(g(W_3(g(W_2(g(Wx))))))))$$

Concepts to review:

- *Derivatives*
- *Partial derivatives*
- *Gradients*
- *Chain rule*



[Essence of Calculus]

Code view

```
model = Sequential([
  Dense(200, activation=tf.nn.relu),
  Dense(100, activation=tf.nn.relu),
  Dense(60, activation=tf.nn.relu),
  Dense(30, activation=tf.nn.relu),
  Dense(10, activation=tf.nn.softmax)
])
```

Use model.summary() for a nice count of the parameters.

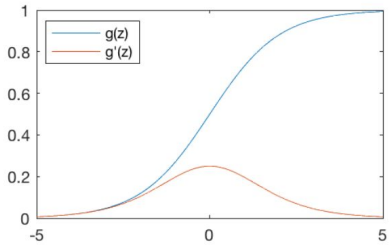
More on ReLU later, but TLDR

If ReLU feels unsatisfying to you (as compared to using a Sigmoid activation) you are not alone. **Why do we use them?**

Sigmoid has a shortcoming in gradient based learning. They "saturate" (the derivative approaches zero) near the extremes. This can stall learning, especially in DNNs (when we're multiplying a series of small gradients together).

ReLU is much faster to compute - but that's **not** why the model converges faster.

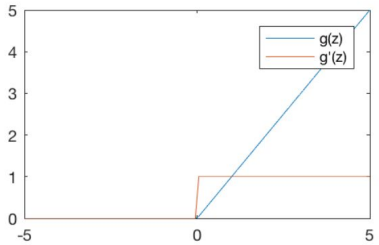
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

Rectified Linear Unit (ReLU)

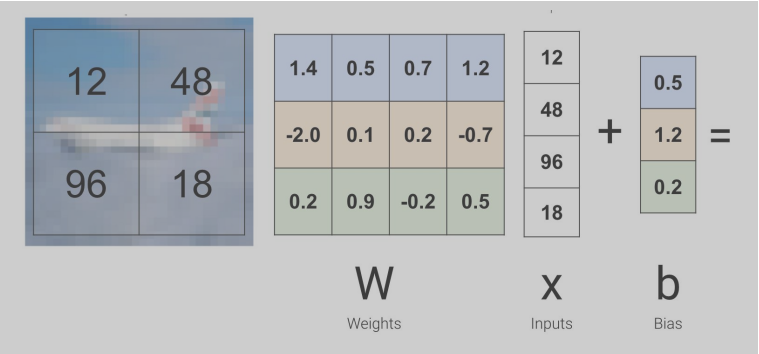


$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Softmax

At this point we have scores. What we want are probabilities (or at least, to interpret them as probabilities).



130.1	Plane
-11.4	Car
12.8	Truck

Output

Scores

We left off with scores

For classification problems, the last activation function is most often softmax.

$$y = softmax(W_2\ g(Wx))$$

A neural network

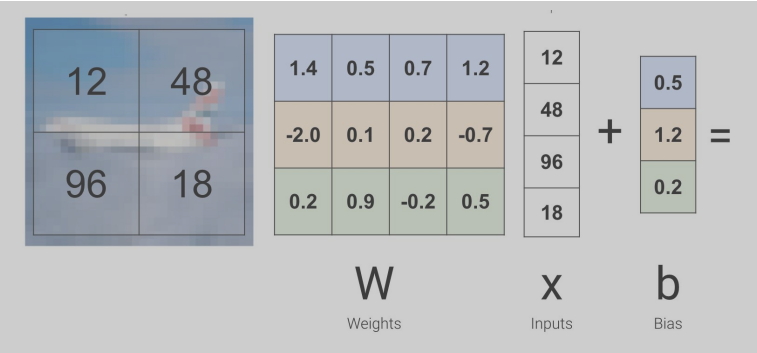
In this case, we have:

$$y = \boxed{softmax}(Wx)$$

A linear classifier.

Softmax converts scores to “probabilities”

So far, we’ve produced a score for each image. Softmax is a function to convert those scores into probabilities.



We left off with scores

130.1	Plane
-11.4	Car
12.8	Truck

Scores

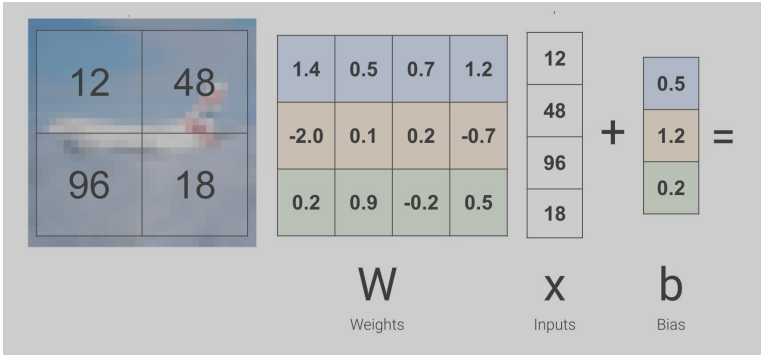
Only input to the softmax function is the class scores.

```
softmax([130.1, -11.4, 12.8])
>>> 0.999, 0.001, 0.001
```

Probabilities

Softmax converts scores to “probabilities”

So far, we’ve produced a score for each image. Softmax is a function to convert those scores into probabilities.



We left off with scores

130.1	Plane
-11.4	Car
12.8	Truck

Scores

Only input to the softmax function is the class scores

```
softmax([130.1, -11.4, 12.8])
>>> 0.999, 0.001, 0.001
```

Probabilities

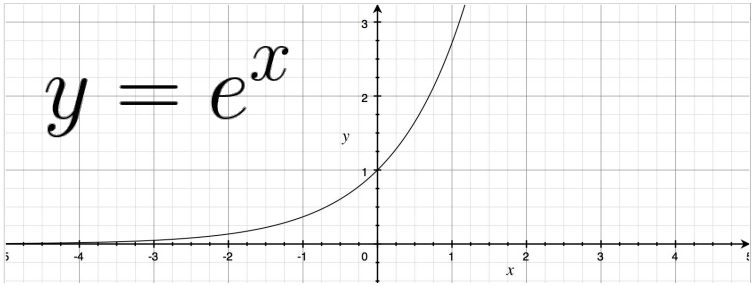
Note: these are ‘probability like’ numbers (do not go to vegas and bet in this ratio).

Math view

The last activation function in a network used for classification. Normalizes each output to 0 < x < 1, and such that they sum to 1.

$$softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

Why exp? No prediction will have zero or negative probability.



Reminder

Notice, no parameters to learn - just a function to convert scores to probabilities.

Code view and a few examples

```
import numpy as np
softmax = np.exp(scores) / np.sum(np.exp(scores))
```

Could be written more efficiently (we're computing that exp(scores) twice.

```
>>> softmax([1,2,3])
# 0.09, 0.24, 0.66
```

Higher scores increase output multiplicatively

```
>>> softmax([0, 0, 10])
# 4.53e-05 4.53-05 9.99e-01
```

Outputs may approach 1 (but will always be less than 1, rounding errors aside)

```
>>> softmax([-10, -5, -8])
# 0.006 0.946 0.047
```

No output ever has zero or negative probability

Loss

Loss

- Just a number that ranges between 0 (perfect) and +inf (maximally bad).
- Lower is better.

Cross Entropy Loss

- The most common (and standard) loss for classification problems.

- Compares two distributions: The probabilities you predicted, and the probabilities you wanted.

One-hot encodings

In CIFAR-10 we have 10 classes, airplane, automobile, etc

Each example has a label in a one-hot format

This is a bird

0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	0	0	0

Interpret as the probability distribution we want our model to predict

Each example has a label in a one-hot format

This is a bird

0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	0	0	0

True probabilities

Softmax output is predicted distribution

Each example has a label in a one-hot format

This is a bird

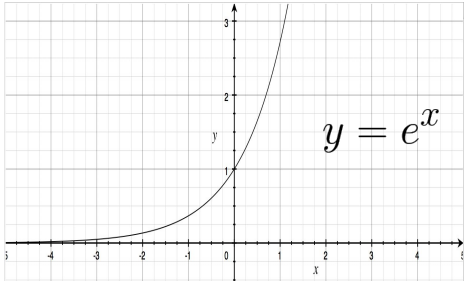
0	1	2	3	4	5	6	7	8	9	
0	0	1	0	0	0	0	0	0	0	True probabilities
0.1	0.2	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	Predicted probabilities

Rounded! Softmax output is always $0 < x < 1$

Recall: Softmax output 0 < x < 1

Each example has a label in a one-hot format

This is a bird



$$softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	0	0	0

True probabilities

0.1	0.2	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0

Predicted probabilities

Rounded! Softmax output is always 0 < x < 1

A loss function compares these distributions

Each example has a label in a one-hot format

This is a bird

0	1	2	3	4	5	6	7	8	9	
0	0	1	0	0	0	0	0	0	0	True probabilities
0.1	0.2	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	Predicted probabilities

Rounded! Softmax output is always $0 < x < 1$

Cross entropy loss for a batch of examples

True prob (either 1 or 0) in our case!

$$L = -\sum \hat{y}_i \log(y_i)$$

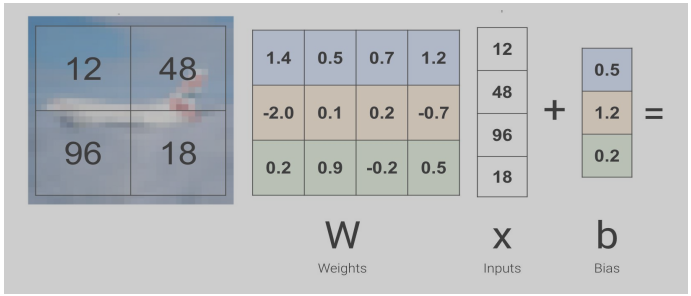
Sum over all examples

Predicted prob (between 0-1)

Let's look at an example

For these examples, correct class will always be 'plane'.

Rounded! Not actually 1, or 0 (but close).



130.1	Plane
-11.4	Car
12.8	Truck

Scores

1.0	Plane
0.0	Car
0.0	Truck

Softmax

?

Loss

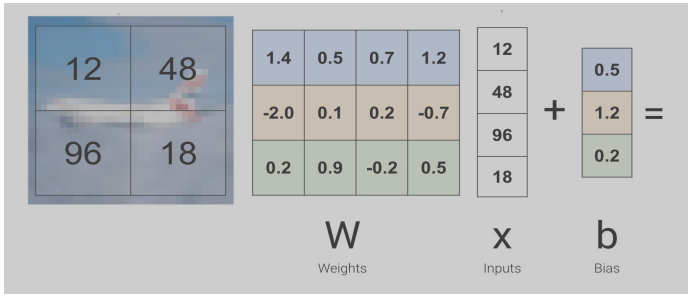
Our friend the dense layer. (Ignore the weights as written. Let's see how cross-entropy loss looks for a few example outputs.)

We want to interpret these as probabilities

Let's look at an example

For these examples, correct class will always be 'plane'.

Rounded! Not actually 1, or 0 (but close).



130.1	Plane
-11.4	Car
12.8	Truck

Scores

1.0	Plane	-log(1.0)
0.0	Car	
0.0	Truck	

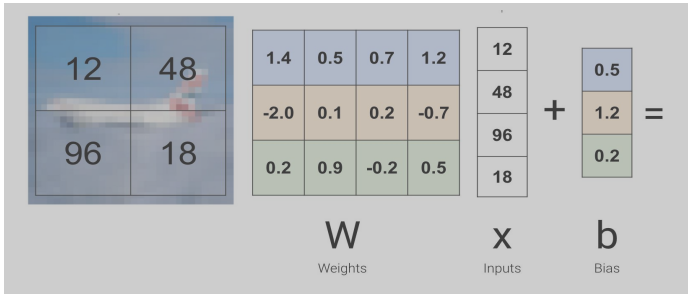
Softmax

Loss

Our friend the dense layer. (Ignore the weights as written. Let's see how cross-entropy loss looks for a few example outputs.)

We want to interpret these as probabilities

Let's look at an example



Our friend the dense layer. (Ignore the weights as written. Let's see how cross-entropy loss looks for a few example outputs.)

For these examples, correct class will always be 'plane'.

130.1	Plane
-11.4	Car
12.8	Truck

Scores

We want to interpret these as probabilities

Rounded! Not actually 1, or 0 (but close).

1.0	Plane
0.0	Car
0.0	Truck

Softmax

Rounded! Not actually 0 (but close).

Loss = 0

Loss

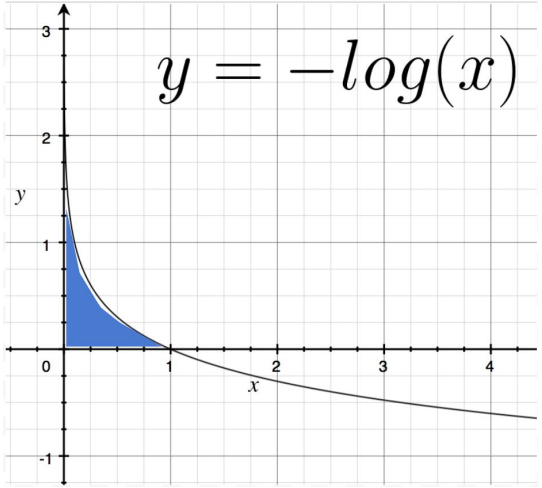
Minimum and maximum possible loss?

When does loss reach a **minimum**?

-

When does loss reach a **maximum**?

-



CE loss will be in this region

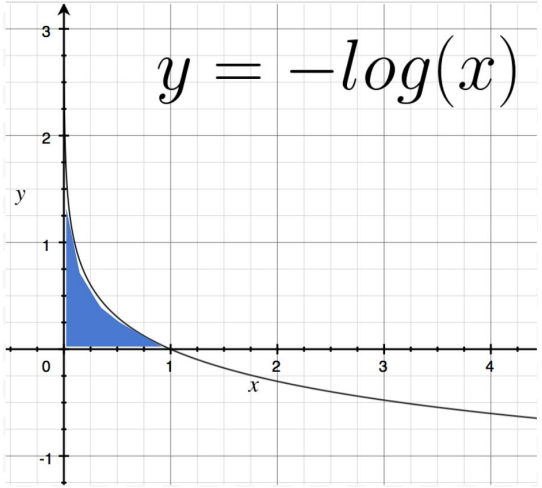
Minimum and maximum possible loss?

When does loss reach a **minimum**?

- Softmax output on the correct example approaches 1, loss approaches 0.

When does loss reach a **maximum**?

- Softmax output on the correct example approaches 0, loss approaches +inf.



CE loss will be in this region

What initial loss do we expect?

Imagine we have 10 classes, weights initialized randomly. What's the average loss over a large balanced training set?

Predicted probabilities

0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

What initial loss do we expect?

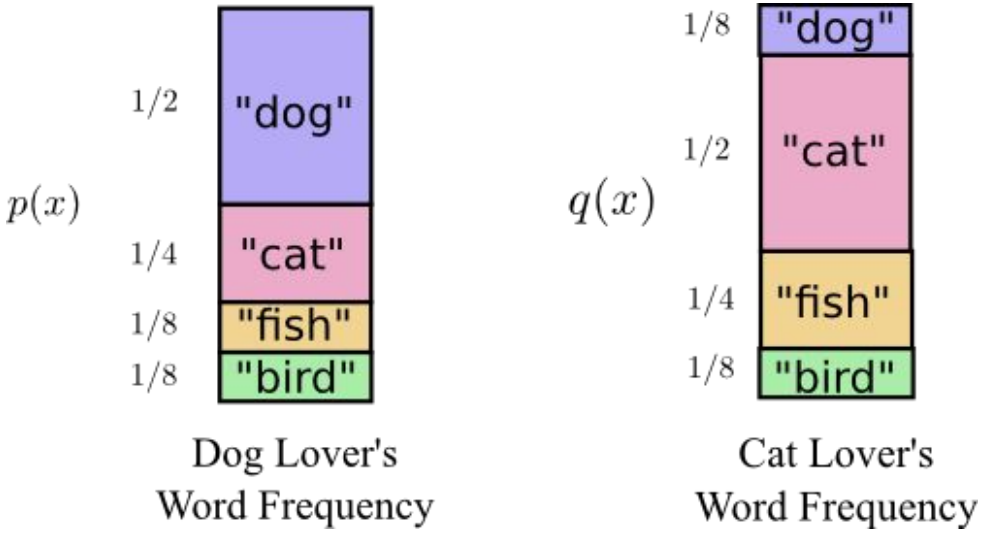
Imagine we have 10 classes, weights initialized randomly. What's the average loss over a large balanced training set?

loss = -ln(1 / n_classes) = ~2.3

Predicted probabilities

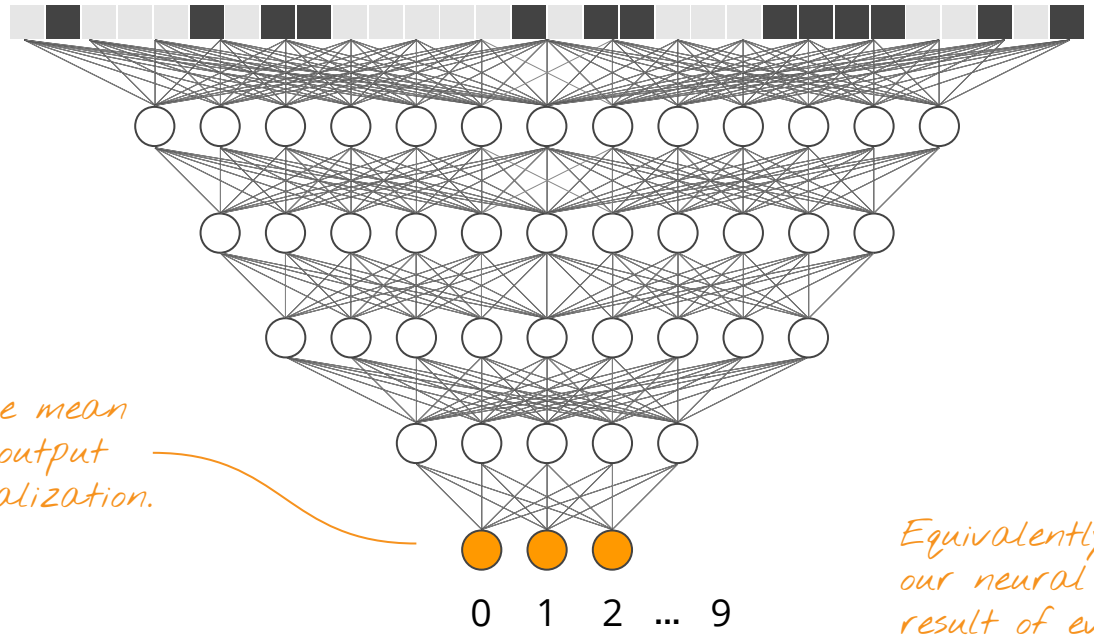
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

If you'd like to learn much more about cross entropy.



colah.github.io/posts/2015-09-Visual-Information/

Softmax and CE loss are common / you'll often see them combined by frameworks into a single utility function.



In DL, by logits, we mean the scores on the output nodes before normalization.

Equivalently, this quantity in our neural network onion. The result of everything right before the softmax.

$$y = softmax(W_5(g(W_4(g(W_3(g(W_2(g(Wx)))))))))$$

Why use cross entropy instead of classification error?

Scenario: consider the case in which you're training a neural network using softmax activation on the output units. There are three class you can predict (*car*, *plane*, *truck*). Let's see how these different loss functions look.

Classification error?

Targets (true label)			Predicted class probabilities			Correct?
Car	Plane	Truck	Car	Plane	Truck	
1	0	0	0.4	0.3	0.3	Yes
0	1	0	0.3	0.4	0.3	Yes
0	0	1	0.1	0.8	0.1	No

Classification error?

Targets (true label)			Predicted class probabilities			Correct?
Car	Plane	Truck	Car	Plane	Truck	
1	0	0	0.4	0.3	0.3	Yes
0	1	0	0.3	0.4	0.3	Yes
0	0	1	0.1	0.8	0.1	No

Accuracy: ⅔ = 0.66. Classification error: ⅓ = 0.33

Does classification error describe how well this model does?

Targets (true label)			Predicted class probabilities			Correct?
Car	Plane	Truck	Car	Plane	Truck	
1	0	0	0.4	0.3	0.3	Yes
0	1	0	0.3	0.4	0.3	Yes
0	0	1	0.1	0.8	0.1	No

Are all errors equivalent?

Targets (true label)			Predicted class probabilities			Correct?
Car	Plane	Truck	Car	Plane	Truck	
1	0	0	0.4	0.3	0.3	Yes
0	1	0	0.3	0.4	0.3	Yes
0	0	1	0.1	0.8	0.1	No

Confidently incorrect predictions should be penalized more

Are all errors equivalent?

Targets (true label)			Predicted class probabilities			Correct?
Car	Plane	Truck	Car	Plane	Truck	
1	0	0	0.4	0.3	0.3	Yes
0	1	0	0.3	0.4	0.3	Yes
0	0	1	0.1	0.8	0.1	No

These predictions are correct, but barely

The same classification error, but a much better classifier.

Targets (true label)			Predicted class probabilities			Correct?
Car	Plane	Truck	Car	Plane	Truck	
1	0	0	0.8	0.1	0.1	Yes
0	1	0	0.1	0.8	0.1	Yes
0	0	1	0.4	0.4	0.3	No

```
# Training data
X = load_MNIST() # [60,000, 784]

# Initialize weights and bias
hidden_size = 100

# Weights from input layer -> hidden layer
W = 0.01 * np.random.randn(input_dim, hidden_size)  # [784, 100]
b = np.zeros((1, h))  # [1, 100]

# Weights from hidden layer -> output layer
W2 = 0.01 * np.random.randn(hidden_size, output_classes)  # [784, 10]
b2 = np.zeros((1, output_classes))  # [1, 10]
```

```
# Begin our gradient descent loop
for i in range(epochs):

  # 1. Forward propagate
  hidden_layer = np.maximum(0, np.dot(X, W) + b) # ReLU activation. (60,000, 100)
  scores = np.dot(hidden_layer, W2) + b2 # Unnormalized scores. (60,000, 10)
```

```
# Begin our gradient descent loop
for i in range(epochs):

  # 1. Forward propagate
  hidden_layer = np.maximum(0, np.dot(X, W) + b) # ReLU activation. (60,000, 100)
  scores = np.dot(hidden_layer, W2) + b2 # Unnormalized scores. (60,000, 10)

  # 2. Softmax
  exp_scores = np.exp(scores)
  probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # (60,000, 10)
```

```
# Begin our gradient descent loop
for i in range(epochs):

  # 1. Forward propagate
  hidden_layer = np.maximum(0, np.dot(X, W) + b) # ReLU activation. (60,000, 100)
  scores = np.dot(hidden_layer, W2) + b2 # Unnormalized scores. (60,000, 10)

  # 2. Softmax
  exp_scores = np.exp(scores)
  probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # (60,000, 10)

  # 3. Loss
  losses = -np.log(probs[range(num_examples),y]) # (60,000, 1)
  loss = np.sum(losses) / n_examples # (1,)
```

For next time

Next time

Code

- Deep Learning basics (from our friends at MIT)
- These notebooks from Aurélien Geron look promising

Reading

- Deep Learning with Python: 3.
- Deep Learning: 4.1, 4.2, 4.3, 5.1, 5.2
- Essence of Calculus (okay, it's video, not reading, but incredibly helpful if you need a refresher)

Assignments

- A1 released (see GitHub). Due Feb 14th.