Logistic Regression

Intro to ML 466/566

Fall 2033

Administrivia

- Virtual office hours
  - (**Oct.3, 17, Nov. 14, Dec. 5 only**):
  - https://meet.google.com/ptm-znnz-kbb
  - Meet link is also on eclass

Homework typos 1D

c) **(3 marks)** The perceptron algorithm will stop when it arrives at any weight vector that properly separates training data. Are all possible weight vector solutions (that separate the training data) equally likely to generalize to a larger dataset? Please explain your answer.

d) **(7 marks)** Now consider the dataset={([-1,-1],1),([1,1],1),([1,-1],-1),([-1,1],-1)}, and the weight is initialized as $w_0 = [0, 1]^T$.

i. As above, plot out the update direction of the weight using point $([-1, -1], 1)$ on the right of Figure 1. [Plot the update direction starting from the current weight point.]

ii. Then, plot out the new weight vector and the separation plane in a new figure. Again, plot out the update direction of the weight using point $([1, 1], 1)$.

iii. After these updates, are all training data points correctly predicted? If the algorithm uses other data points to update, will it yield a separation line that correctly predicts for all points? Why or why not?

Homework clarification 2 b

- “Use Cauchy Schwartz inequality on the result of 1 to find a lower bound of *w^{k* +1} .”

- “Use Cauchy Schwartz inequality on the result of 1 to find a lower bound of **||*w^{k* +1} ||**.”

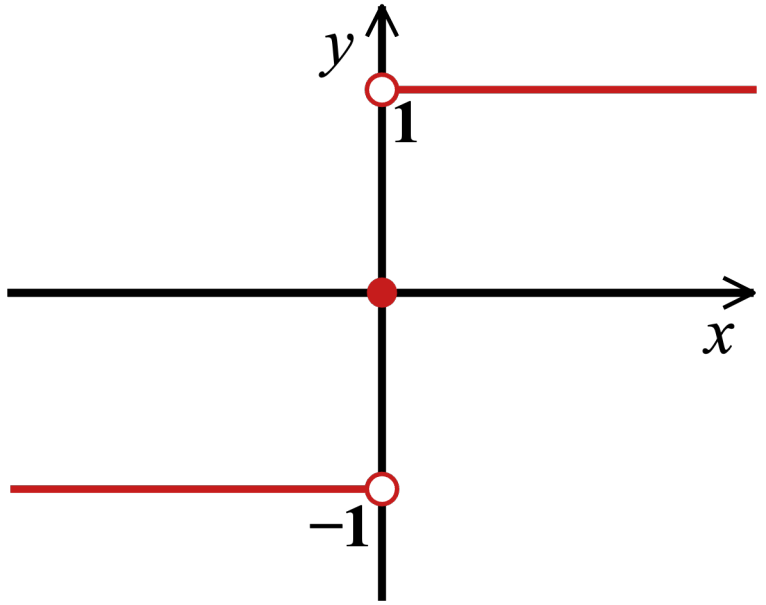
Homework clarification 2 c

- “Find an upper bound for the **inner product of the weights at (*k* + 1)-th iteration** using the perceptron algorithm update.”
- **inner product of the weight vector at the (*k* + 1)-th iteration with itself**

||w^{k+1}||^2 (squared L2 norm)

Perceptron

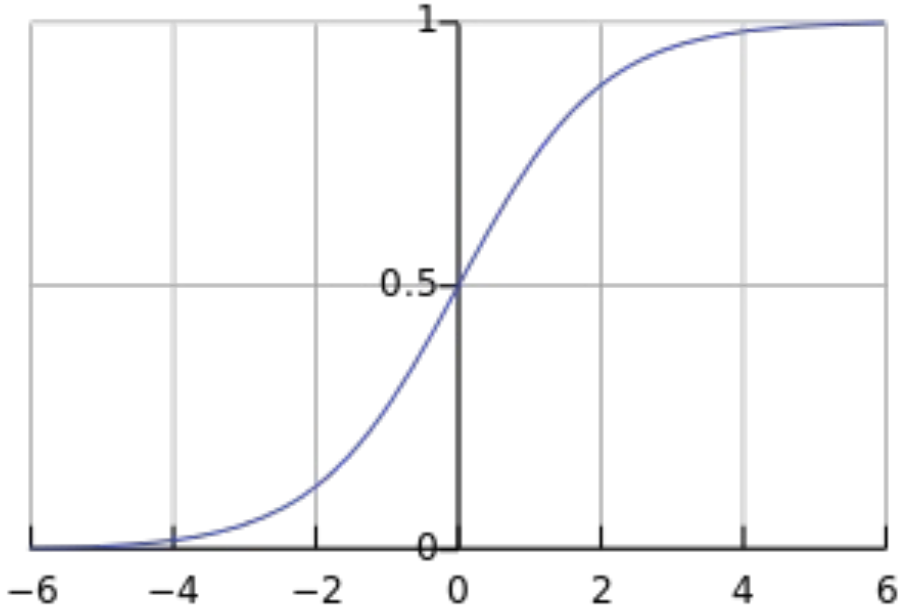
- Recall: perceptron uses the *sign* function



- Outputs either -1 or +1

Logistic Regression

- Similar to perceptron, but sigmoid function applied to linearity.



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

- Error function is based on Max Likelihood →

  as a result we get genuine probabilities as output of prediction (not just 1,0 discrete values).

Models so far, and the new one

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

Perceptron

$$h(\mathbf{x}, \mathbf{w}) = \text{sign}(z)$$

Linear regression

$$h(\mathbf{x}, \mathbf{w}) = z$$

Logistic regression

$$h(\mathbf{x}, \mathbf{w}) = \text{S}(z)$$

Output *h*(x,w) will be interpreted as probability.

Why? Because of the range of S, and particular cost function we'll optimize.

Probability Interpretation

- Assume tuples (**x**, *y*), where *y* is binary, are generated from some noisy data source according to some distribution *f*.

$$p(y \mid \mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = 0 \\ 1 - f(\mathbf{x}) & \text{for } y = 1 \end{cases}$$

For mathematical convenience we will use [0,1]
as our label set instead of [-1,1]

- We will learn *f*(x) by approximating it with the sigmoid $S(z) = \frac{1}{1+e^{-z}}$

  i.e. S(w.x)

$$\mathbf{w} = \left[w_0 = b, w_1, .., w_m \right] \qquad \mathbf{x} = \left[x_0 = 1, x_1, .., x_m \right]$$

- Makes sense as S(*z*) is a function from 0 to 1.

Approximation

- What is the probability of *y*=0?

  (according to our approximation)

$$p(y=0 \mid \mathbf{x}) = \frac{1}{1+e^{\mathbf{w}'\mathbf{x}}}$$

- What is the probability of *y*=1?

  (according to our approximation)

$$p(y=1 \mid \mathbf{x}) = 1 - \frac{1}{1+e^{\mathbf{w}'\mathbf{x}}}$$

$$= \frac{1+e^{\mathbf{w}'\mathbf{x}}-1}{1+e^{\mathbf{w}'\mathbf{x}}}$$

$$= \frac{e^{\mathbf{w}'\mathbf{x}}}{1+e^{\mathbf{w}'\mathbf{x}}}$$

Training Logistic Regression: MCLE

- Choose parameters W=<w_0, ... w_n> to <u>maximize conditional likelihood</u> of training data

where $p(y=0 \mid \mathbf{x}) = \frac{1}{1+e^{\mathbf{w}'\mathbf{x}}}$

$p(y=1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}'\mathbf{x}}}{1+e^{\mathbf{w}'\mathbf{x}}}$

- Training data D = $\{< X^1, Y^1 >, \ldots < X^N, Y^N >\}$

- Data <u>conditional</u> likelihood = $\prod_{i=1}^{N} P(Y^i|X^i, W)$

$$W_{MCLE} = \arg\max_{w} \prod_{i=1}^{N} P(Y^i|X^i, W)$$

Training Logistic Regression: MCLE

- We would like to find the w that maximizes the probability of our data

$$W_{MCLE} = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{N} P(Y^i | X^i, W)$$

Expressing Conditional Log Likelihood

$$\mathcal{L}(W) \equiv ln\left(\prod_i P(Y^i|X^i,W)\right)$$

$$= \sum_i ln\ P(Y^i|X^i,W)$$

$$p(y=0\,|\,\mathbf{x}) = \frac{1}{1+e^{\mathbf{w}'\mathbf{x}}}$$

$$p(y=1\,|\,\mathbf{x}) = \frac{e^{\mathbf{w}'\mathbf{x}}}{1+e^{\mathbf{w}'\mathbf{x}}}$$

$$\mathcal{L}(W) = \sum_i Y^i\ ln\ P(Y^i=1|X^i,W) + \sum_i (1-Y^i)\ ln\ P(Y^i=0|X^i,W)$$

For the samples with Y^l=1

For the samples with Y^l=0

Expressing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \; ln \; P(Y^i = 1|X^i, W) + \sum_i (1 - Y^i) \; ln \; P(Y^i = 0|X^i, W)$$

$$= \sum_i Y^i \; ln \; \frac{P(Y^i = 1|X^i, W)}{P(Y^i = 0|X^i, W)} + \sum_i ln \; P(Y^i = 0|X^i, W)$$

Expressing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \; ln \; P(Y^i = 1|X^i, W) + \sum_i (1 - Y^i) \; ln \; P(Y^i = 0|X^i, W)$$

$$= \sum_i Y^i \; ln \; \frac{P(Y^i = 1|X^i, W)}{P(Y^i = 0|X^i, W)} + \sum_i ln \; P(Y^i = 0|X^i, W)$$

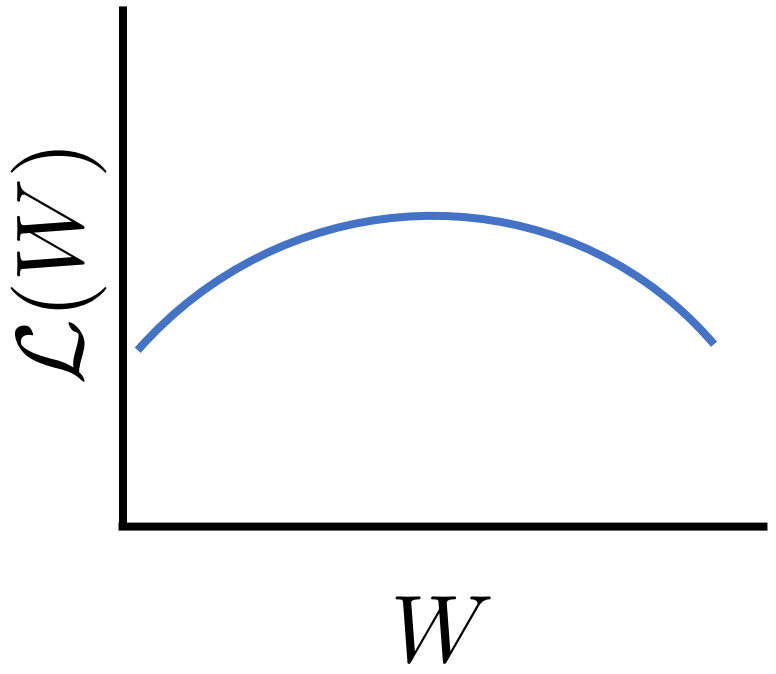
$$= \sum_i Y^i \; (w' x^i) - \sum_i ln(1 + e^{w' x^i})$$

$$p(y = 0 \mid \mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}'\mathbf{x}}}$$

$$p(y = 1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}'\mathbf{x}}}{1 + e^{\mathbf{w}'\mathbf{x}}}$$

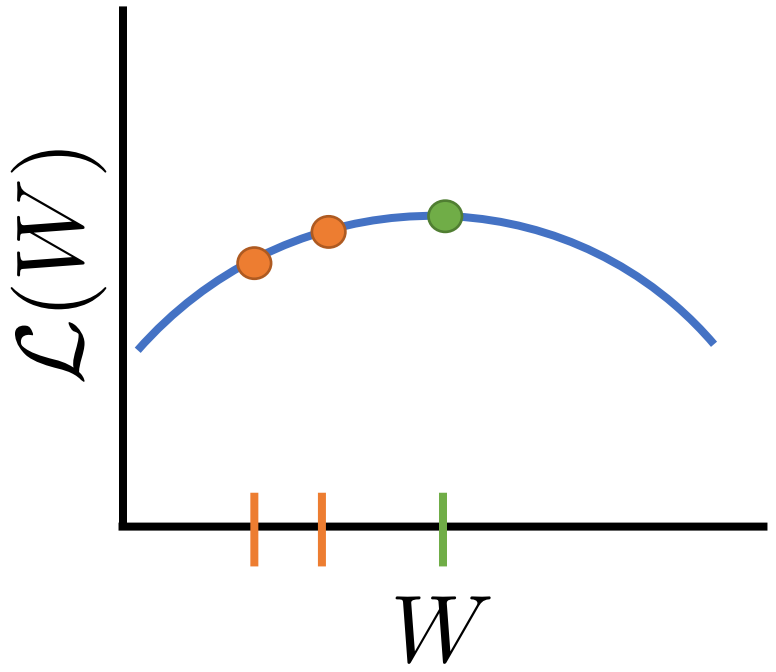
Maximizing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \ (w' x^i) - \sum_i ln(1 + e^{w' x^i})$$



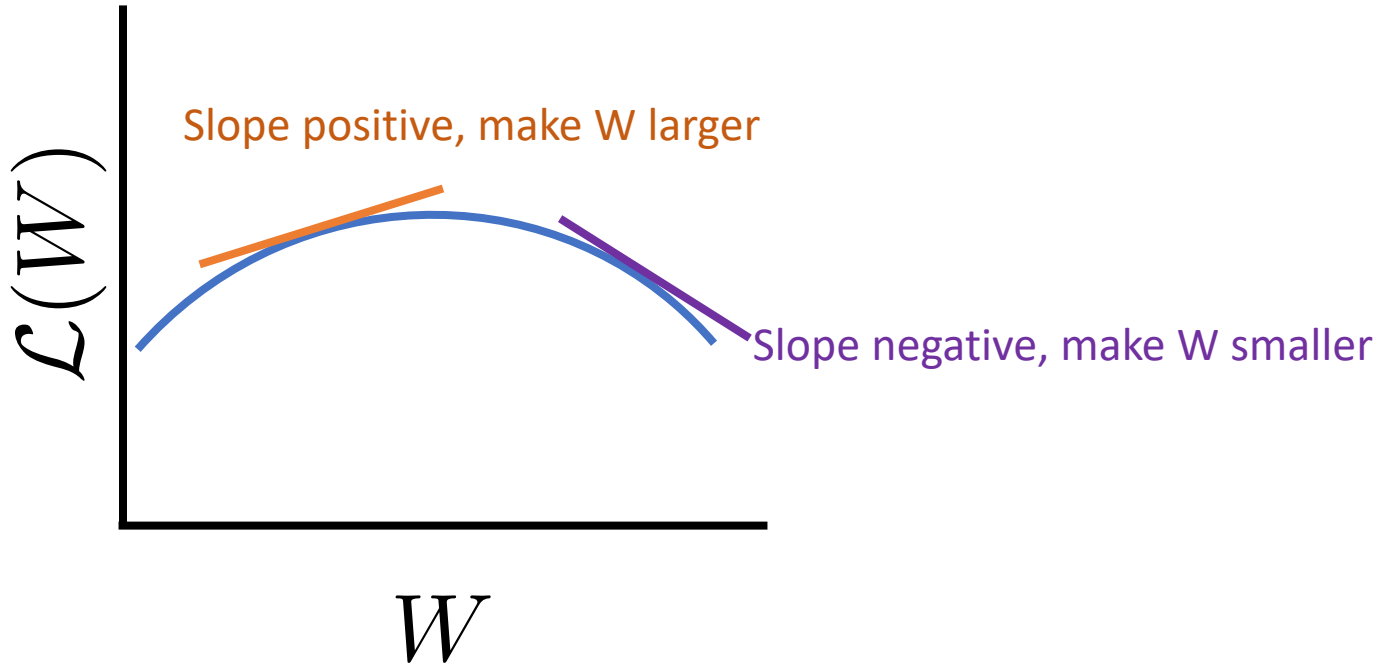
Maximizing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \ (w'x^i) - \sum_i ln(1 + e^{w'x^i})$$

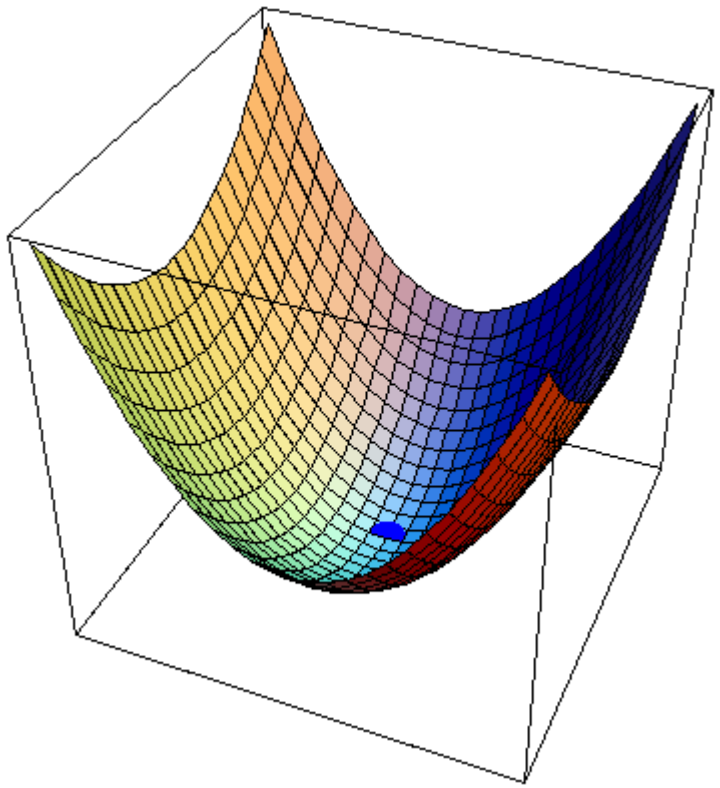


Gradient Descent/Ascent

How do we know *how* to change W?
We want to move along the blue line in a way that makes L(W) larger

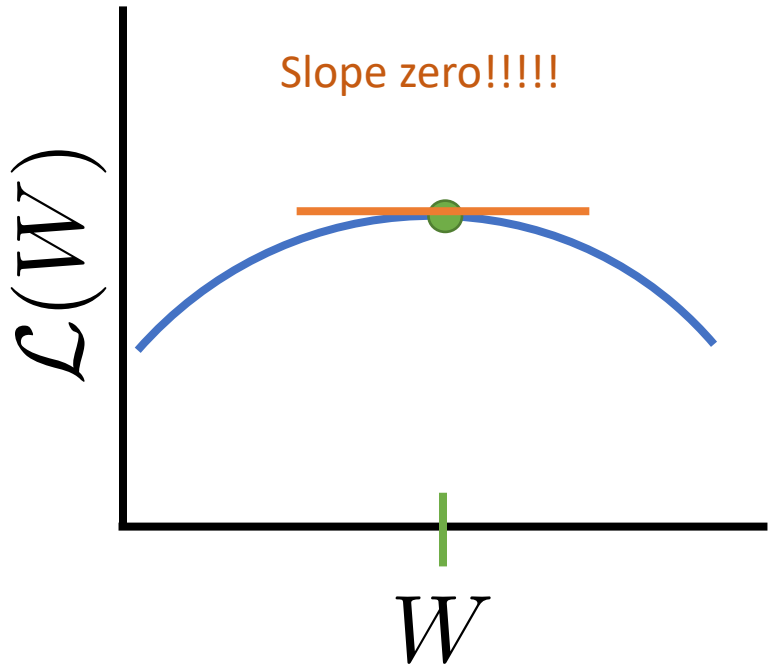


Gradient Descent/Ascent



Maximizing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \ (w' x^i) - \sum_i ln(1 + e^{w' x^i})$$



Maximizing Conditional Log Likelihood

$$\mathcal{L}(W) = \sum_i Y^i \ (w' x^i) - \sum_i ln(1 + e^{w' x^i})$$

$$\frac{d}{dw}\mathcal{L}(W) = \sum_i Y^i x^i - \sum_i \frac{x^i e^{w' x^i}}{1 + e^{w' x^i}}$$

$$= \sum_i x^i (Y^i - P(Y = 1|W, x^i))$$

Iterative Method

- Start at $\mathbf{w}_0$=**1**; take a step along **steepest slope**
- Fixed step size:

$$\mathbf{w}_1 = \mathbf{w}_0 + \eta\mathbf{v}$$

- **v** is a **vector** in the direction of the **steepest slope**.
  - What's the steepest slope?

Gradient Descent Algorithm

Kappa is the learning rate

Initialize $\mathbf{w}_0=\mathbf{0}$

For t=0,1,2,…do

Compute the gradient and update the weights

$$\frac{d}{dw}\mathcal{L}(W) = \sum_i x^i(Y^i - P(Y = 1|W, x^i)$$

$$w \leftarrow w + \kappa\left(\sum_i x^i(Y^i - P(Y = 1|W, x^i)\right)$$

Iterate with the next step until **w** doesn't change too much

(or for a fixed number of iterations)

We may not find the exact point where the slope is 0, but we will get "close enough"

Return final **w**.

Making predictions

- A new tuple comes: (**x**,?)

$$p(y=0 \mid \mathbf{x}) = \frac{1}{1+e^{\mathbf{w}'\mathbf{x}}}$$

$$p(y=1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}'\mathbf{x}}}{1+e^{\mathbf{w}'\mathbf{x}}}$$

- Fix a threshold in [0,1] to make predictions.

$p(y=1 \mid \mathbf{x}) > \text{threshold}$ Predict y=1

$p(y=1 \mid \mathbf{x}) \leq \text{threshold}$ Predict y=0

Example

GPA, GRE, and success.

100, 800, 1

90, 800, 1

90, 700, 1

70, 600, 0

60, 700, 0

60, 700, 1

50, 600, 0

50, 650, 01

50, 800, 1

50, 700, 0

50, 700, 1

Dummy	GPA	GRE	y
1	1.0	1.0	1
1	0.9	1.0	1
1	0.9	0.875	1
1	0.7	0.75	0
1	0.6	0.875	0
1	0.6	0.875	1
1	0.5	0.75	0
1	0.5	0.8125	0
1	0.5	1.0	1
1	0.5	0.875	0
1	0.5	0.875	1

Scaled so that max GPA/GRE is 1

It's a good idea to scale your features!

Example

Logistic regression:

-7.44 + 4.4056*GPA + 5.57 * GRE

$$p(y = 0 \mid gpa, gre) = \frac{1}{1 + e^{(\text{-7.44 + 4.4056*GPA + 5.57 * GRE})}}$$

Fix a threshold in [0,1] to make predictions.

$p(y = 1 \mid gpa, gre) >$ threshold Predict y=1

$p(y = 1 \mid gpa, gre) \leq$ threshold Predict y=0

Odds

Definition: $$odds(0 \text{ vs. } 1 \text{ given } \mathbf{x}) = \frac{p(y=0 \mid \mathbf{x})}{p(y=1 \mid \mathbf{x})}$$

Formula: $$odds(0 \text{ vs. } 1 \text{ given } \mathbf{x}) = \frac{\frac{1}{1+e^{\mathbf{w}'\mathbf{x}}}}{\frac{e^{\mathbf{w}'\mathbf{x}}}{1+e^{\mathbf{w}'\mathbf{x}}}}$$

$$= \frac{1}{e^{\mathbf{w}'\mathbf{x}}}$$

$$= e^{-\mathbf{w}'\mathbf{x}}$$

We predict ? if this number is less than ?

Interpretation

$$odds(\text{successful vs. unsuccessful given } gpa \text{ and } gre) = e^{-7.44 + 4.4056*\text{GPA} + 5.57 * \text{GRE}}$$

If GPA increases by .1 then the odds of success will increase by 55%

$$e^{0.441} \approx 1.55$$

If GRE increases by .1 then the odds of success will increase by 75%

$$e^{0.557} \approx 1.75$$

Regularization

- Recall: What's overfitting?
- Can that happen in logistic regression?
- How can we avoid overfitting?

Regularization

- If I have two models, one with 10 parameters and one with 100, which is more complex?

- So we would like to encourage the model to “have” fewer parameters
  - In practice, this boils down to making some of the parameters close to 0

- This is what regularization can do

Regularization

- Two typical regularization forms
  - L1 (produces sparsity: many w set to 0)

$$\sum_i |w_i|$$

  - L2 (weights small, but not always 0)

$$\sum_i w_i^2$$

So for L1 our likelihood becomes…

$$\mathcal{L}(W) = \sum_i Y^i \ (w' x^i) - \sum_i ln(1 + e^{w' x^i}) - \lambda \sum_i |w_i|$$

- Lambda is a *hyperparameter*
- Tune lambda using nested cross validation, or the validation set.

Multi-class

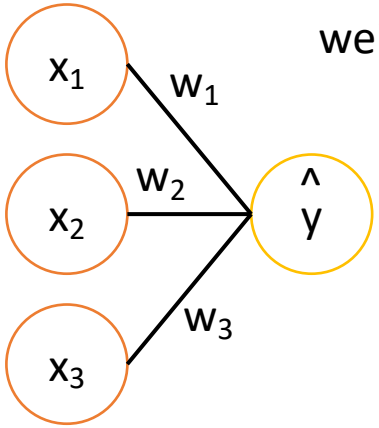
- What if there are more than two classes?
  - E.g. positive, negative or neutral sentiment

- Multinomial logistic regression
  - Essentially builds a logistic regression model for every class
  - Predict the highest probability class
  - In practice this can be done with the softmax function

Logistic regression is actually a simple neural net

Neural Nets

input features (x)



weights (w) associated with each edge

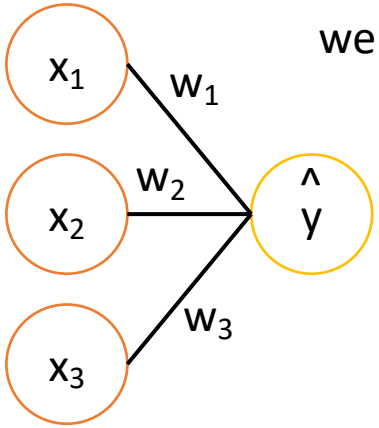
prediction $\hat{y}$

$$\hat{y} = \frac{1}{1 + e^{-\sum_i x_i w_i}}$$

output
(in the range [0…1])

Neural Nets

input features (x)



weights (w) associated with each edge

prediction $\hat{y}$

$$\hat{y} = \frac{1}{1 + e^{-\sum_i x_i w_i}}$$

output
(in the range [0…1])

Logistic regression is actually a simple neural net

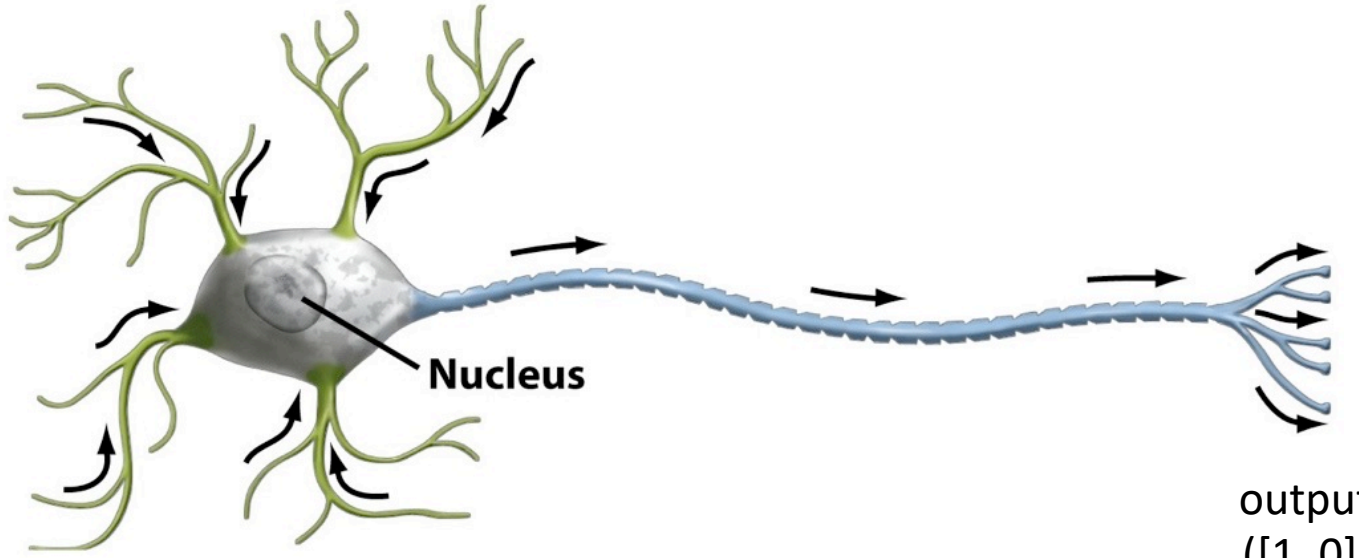
Let’s play!

https://playground.tensorflow.org/

Neural Nets

- Have been around for decades
  - See http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/ for a brief history lesson
- Recently re-popularized due to
  - ability to process more data more quickly
    - GPUs
  - availability of large datasets (particularly in computer vision)

Inspiration

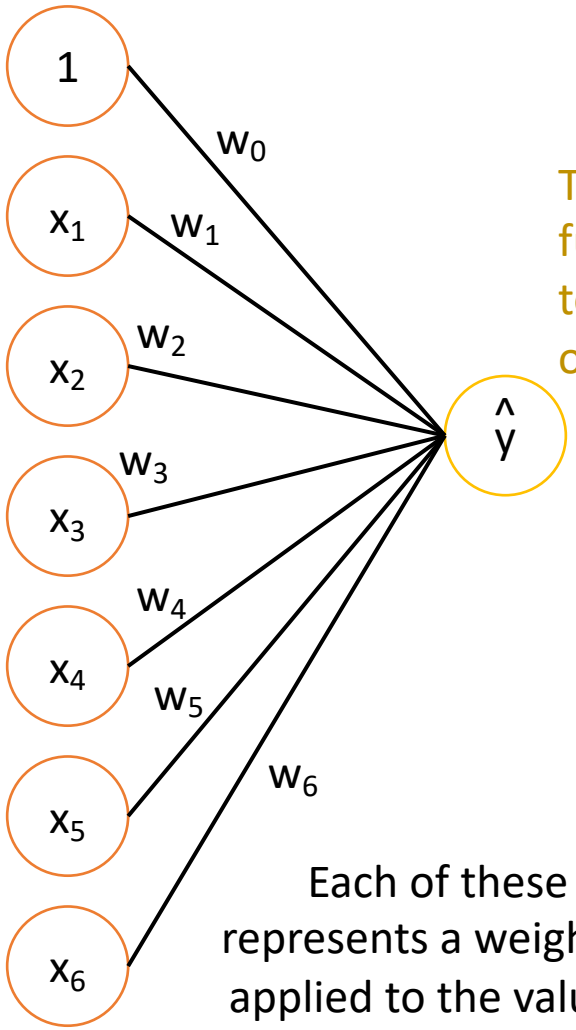


input
(many inputs
from many cells,
weighted with
different
importance)

output
([1, 0]
fire, don’t fire)

Logistic Regression Models are Neural Networks

x is our input feature vector

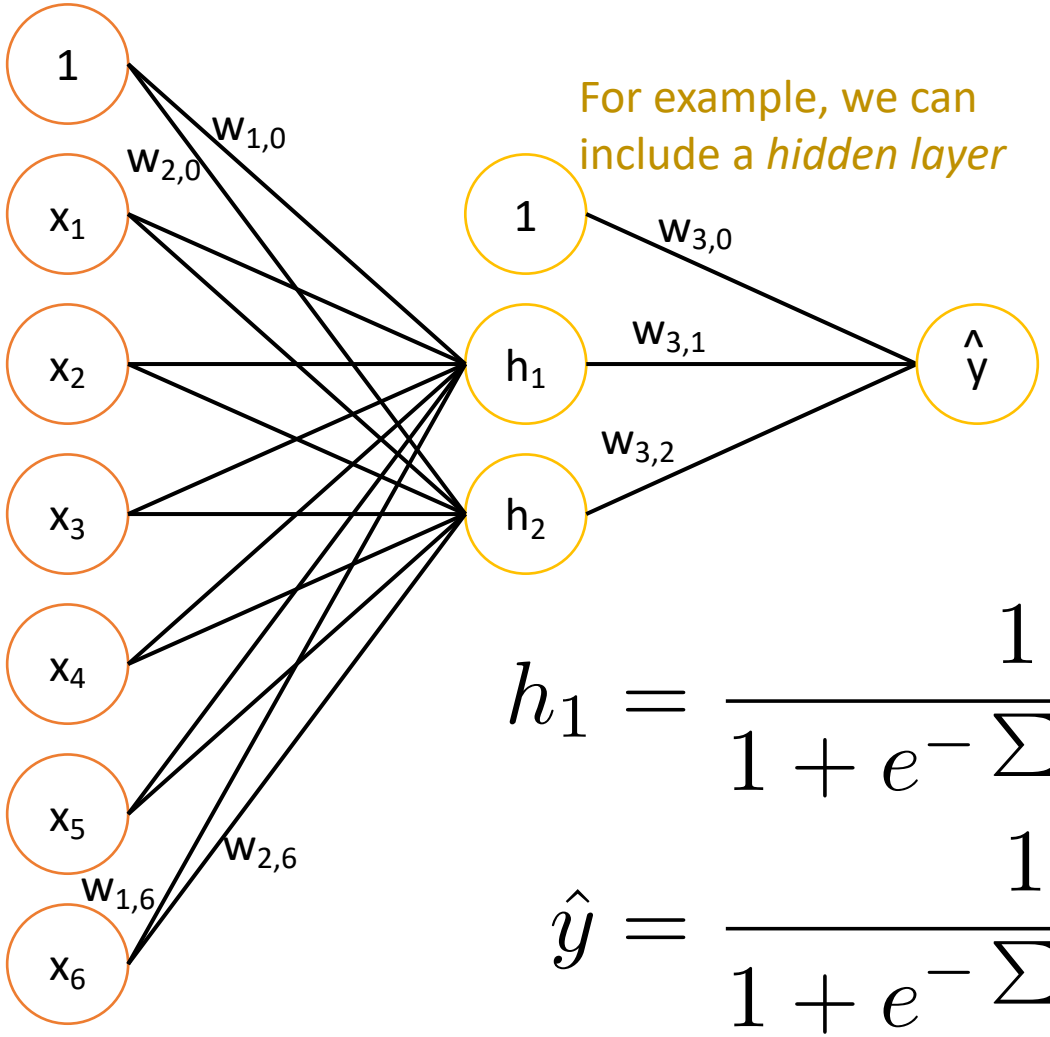


This circle (node) represents a function of the edges connected to it from the left, and the nodes on the other end of that edge

$$\hat{y} = \frac{1}{1 + e^{-\sum_i x_i w_i}}$$

Each of these lines (edges) represents a weight (*w*) that will be applied to the value in the node of the left

Neural nets are a generalization that includes logistic regression, but also other models

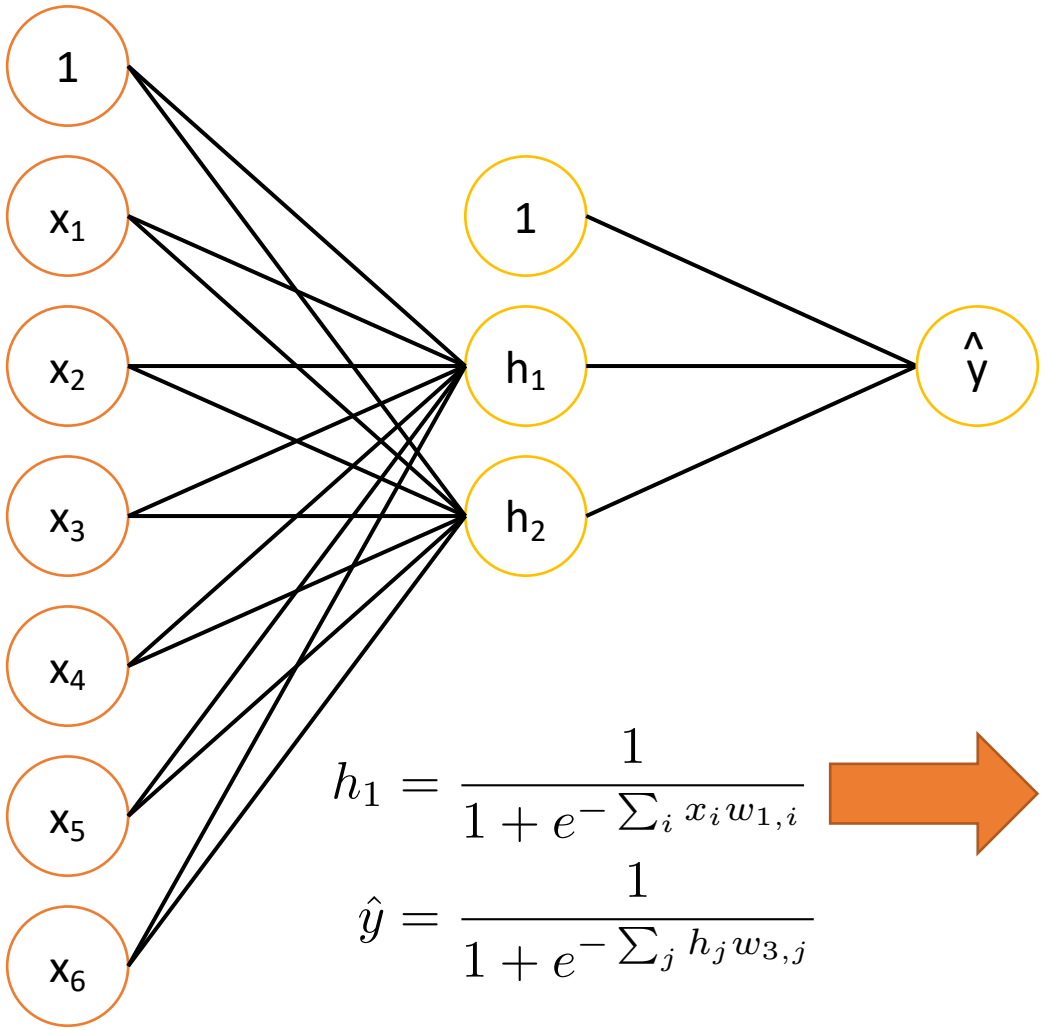


$$h_1 = \frac{1}{1 + e^{-\sum_i x_i w_{1,i}}}$$

$$\hat{y} = \frac{1}{1 + e^{-\sum_j h_j w_{3,j}}}$$

(For simplicity, only some edges are labeled here)

We can also change the activation function



$$h_1 = \frac{1}{1 + e^{-\sum_i x_i w_{1,i}}}$$

$$\hat{y} = \frac{1}{1 + e^{-\sum_j h_j w_{3,j}}}$$

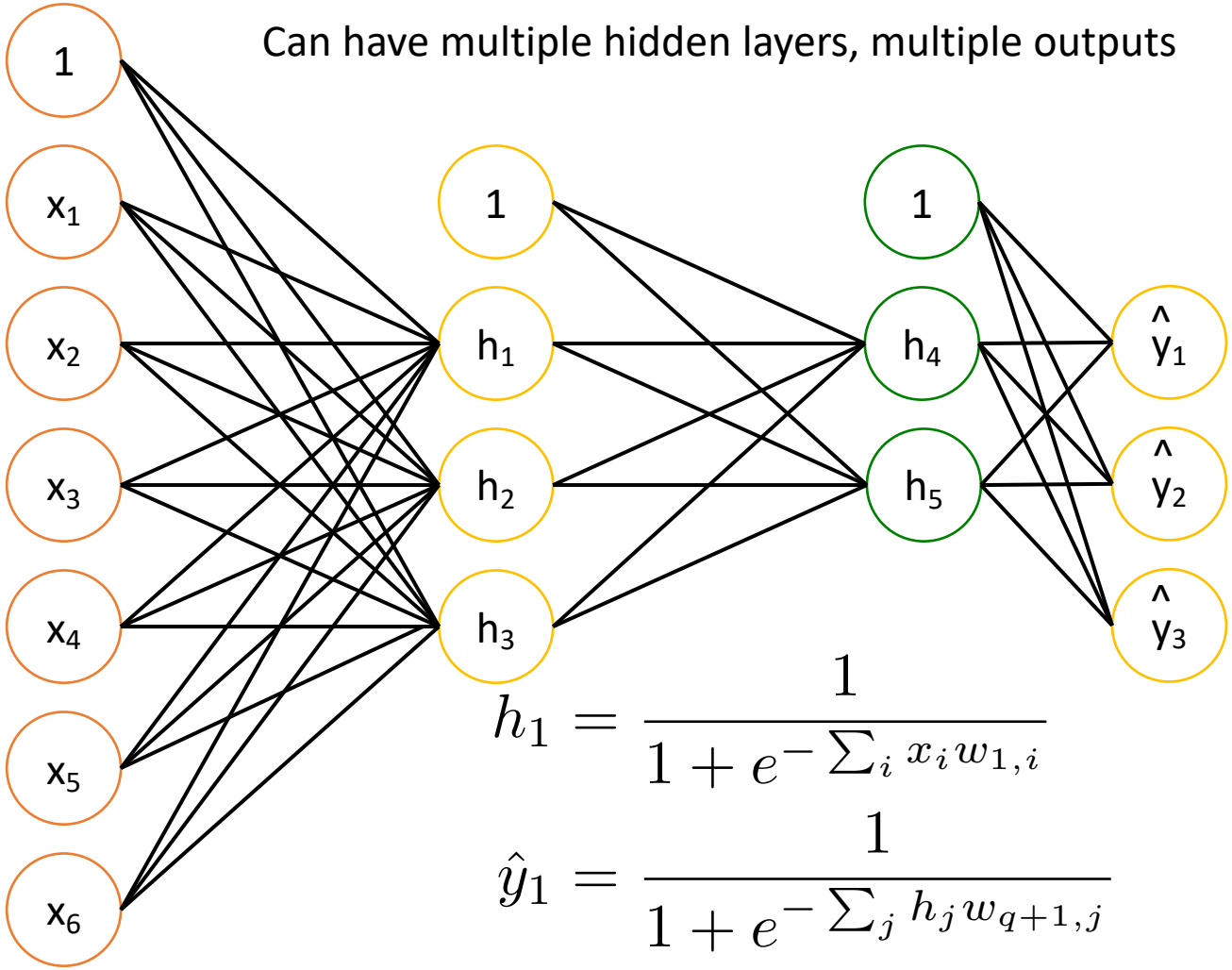
This is the ReLU

$$h_1 = max(0, \sum_i x_i w_{1,i})$$

$$\hat{y} = \frac{1}{1 + e^{-\sum_j h_j w_{3,j}}}$$

Neural Nets

Can have multiple hidden layers, multiple outputs



$$h_1 = \frac{1}{1 + e^{-\sum_i x_i w_{1,i}}}$$

$$\hat{y}_1 = \frac{1}{1 + e^{-\sum_j h_j w_{q+1,j}}}$$

q is total # of hidden nodes across all layers

Deep Learning

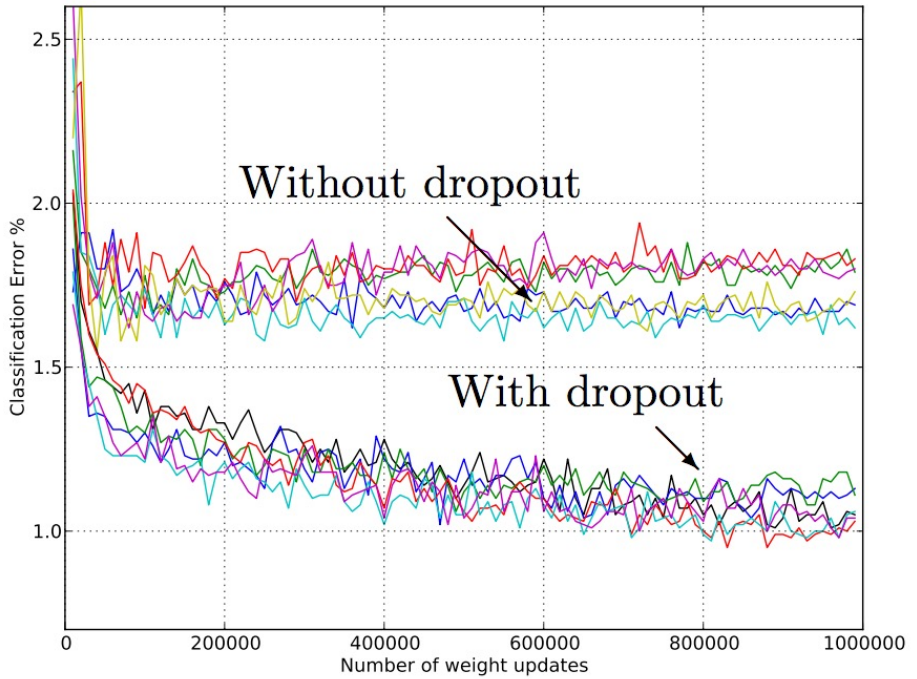
- A model with more than two hidden layers can be called “deep”
- State of the art models can have 50, 100 or more hidden layers

Regularization

- We are learning a much larger number of parameters
- We need to regularize
- Can use L1 or L2 regularizers as in logistic regression & linear regression

Regularization at the Network Level

- Dropout
  - randomly remove nodes from the network @ train
    - i.e. set to 0
  - encourages redundant connections
  - reduces overfitting



https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Deep Learning for Computer Vision

Convolutional neural nets (CNNs)

- Built to capture the invariances we see in images
  - objects can appear at any place in the image

Convolutional neural nets (CNNs)

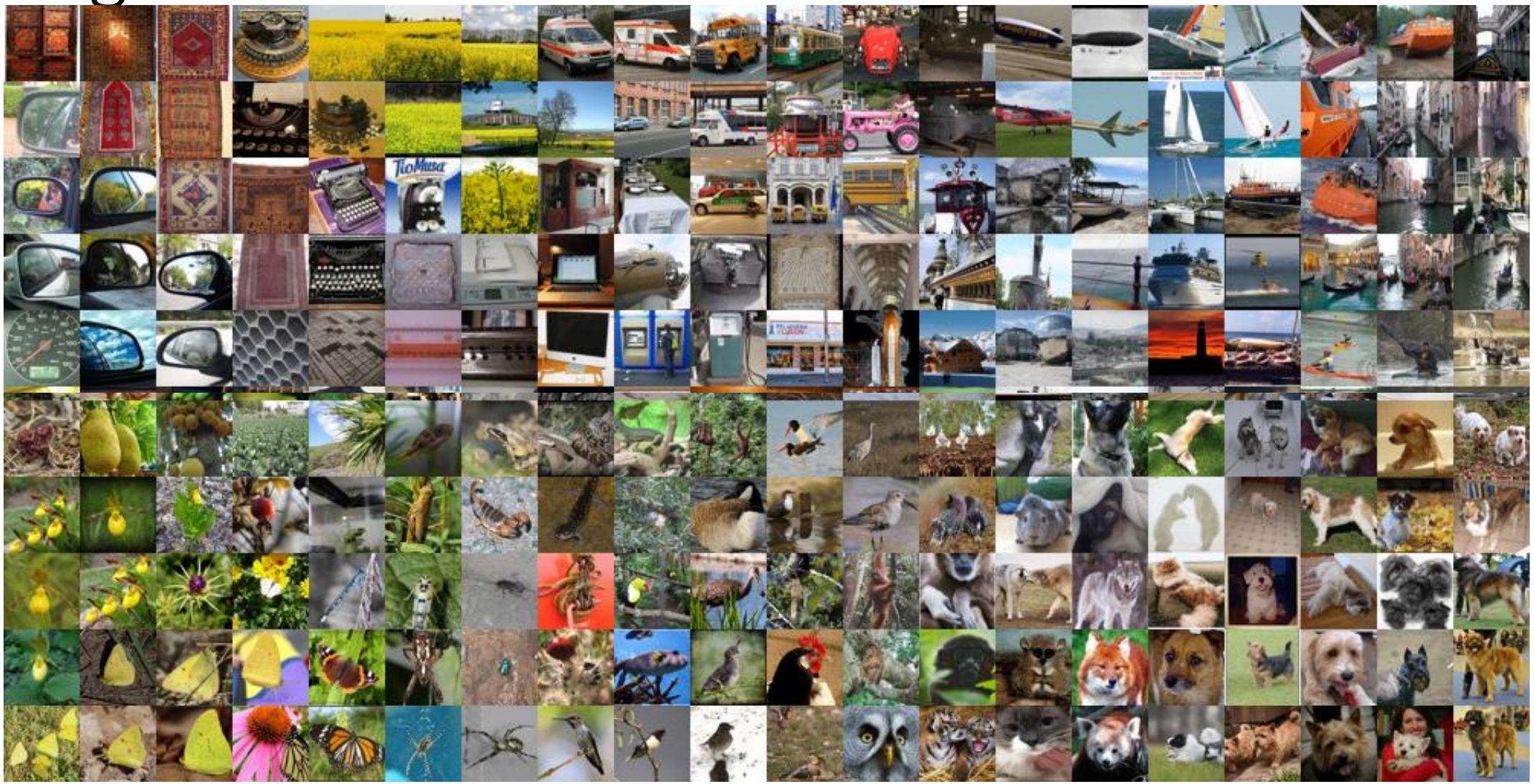
- Learn filters that respond to particular visual features
  - e.g. edges, curves, etc
- Look for those visual features anywhere in the image
  - i.e. convolve the *same* filter with many patches of the image
- Hidden layers combine these filters to create more complex shapes
  - e.g. straight edges combined to form curves, combined to form the handle on a coffee mug

CNNs

- Amazing advances in computer vision
  - ImageNet
    - First released 2009
    - 1.2 million images
    - more than 1000 concepts
      - e.g. cup, oil filter, ptarmigan
  - Deep learning
    - CNNs

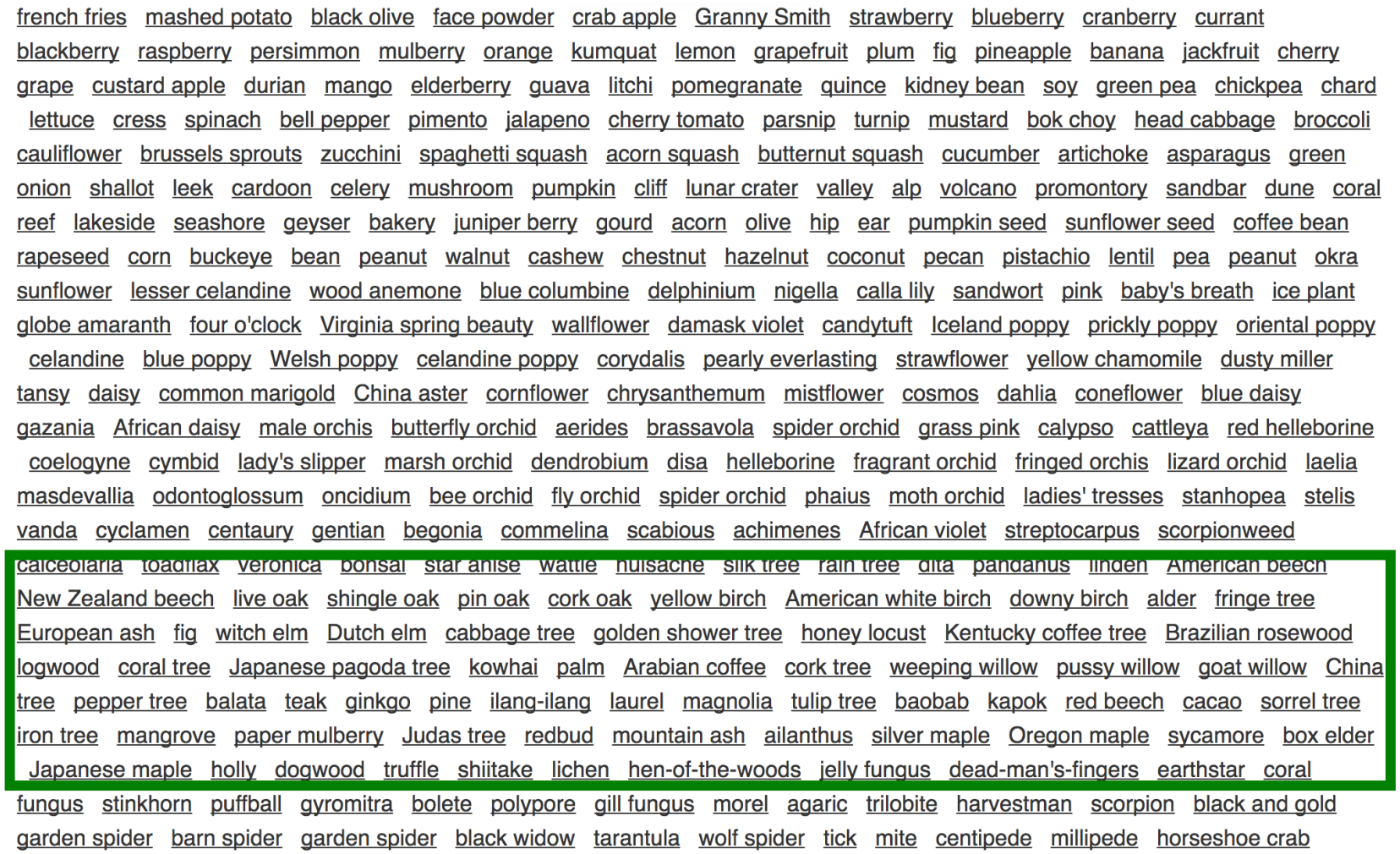
http://image-net.org/

ImageNet

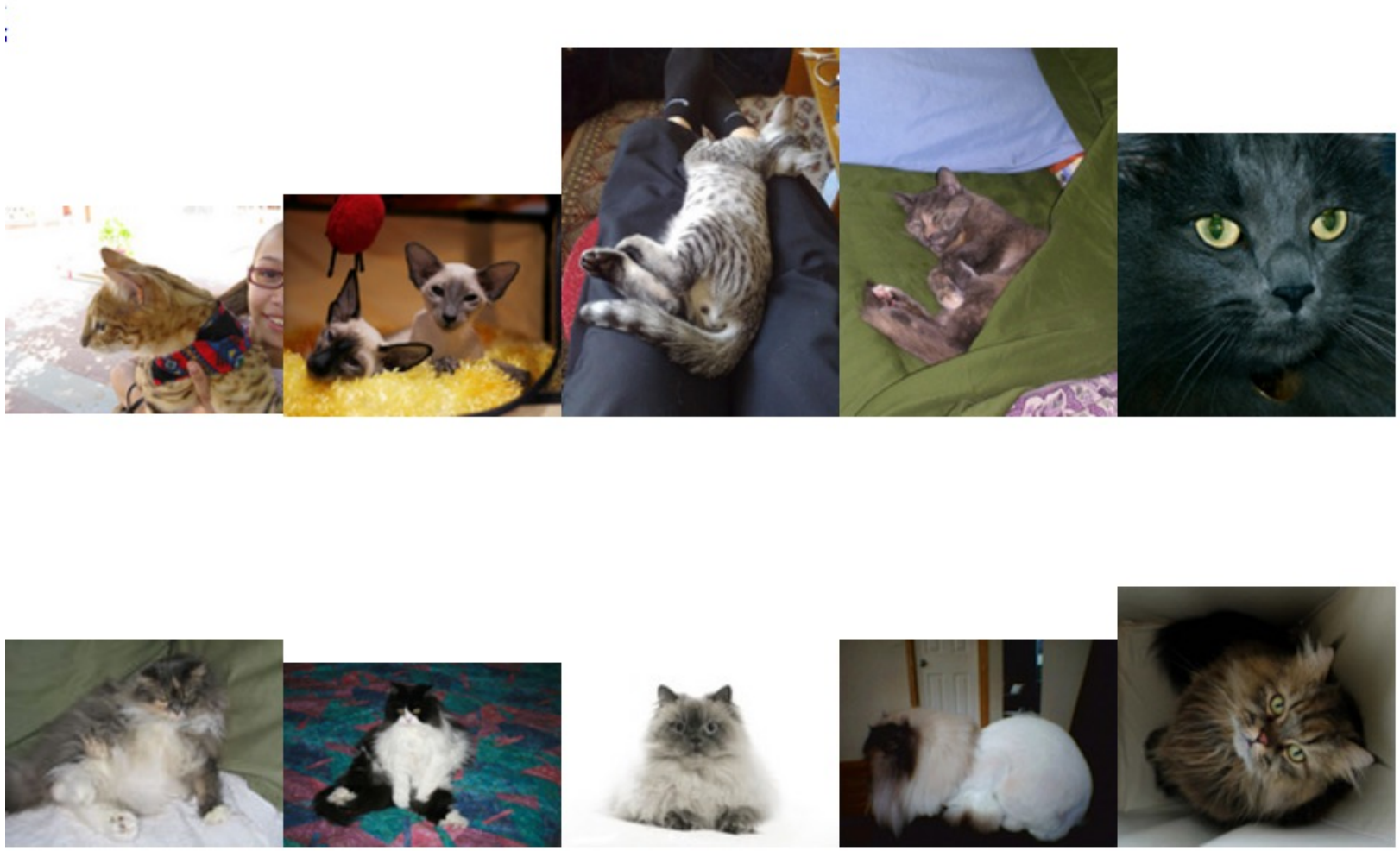


http://cs.stanford.edu/people/karpathy/cnnembed/

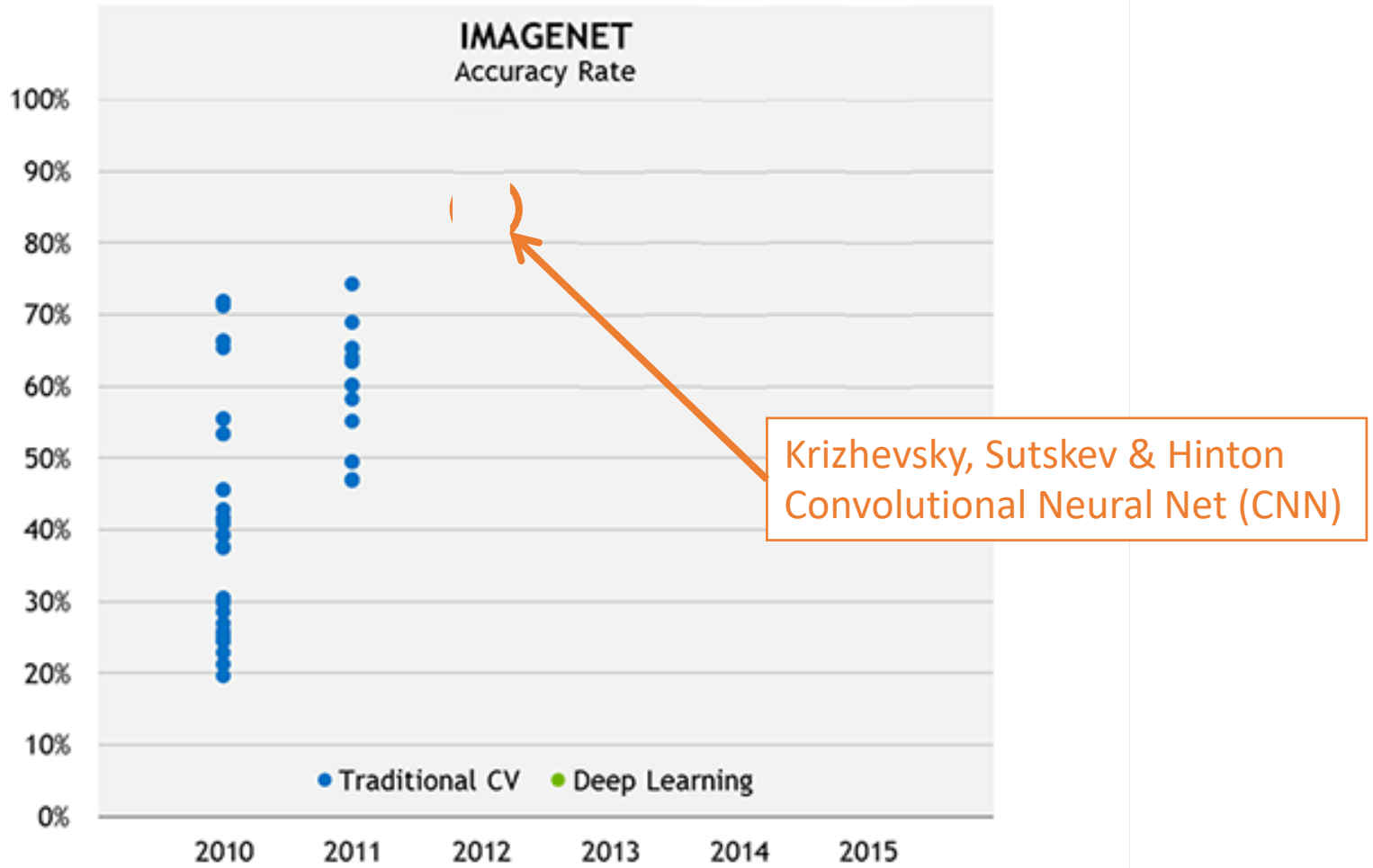
ImageNet



http://image-net.org/challenges/LSVRC/2010/browse-synsets



http://image-net.org/challenges/LSVRC/2012/analysis/



https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/

Self Driving Cars

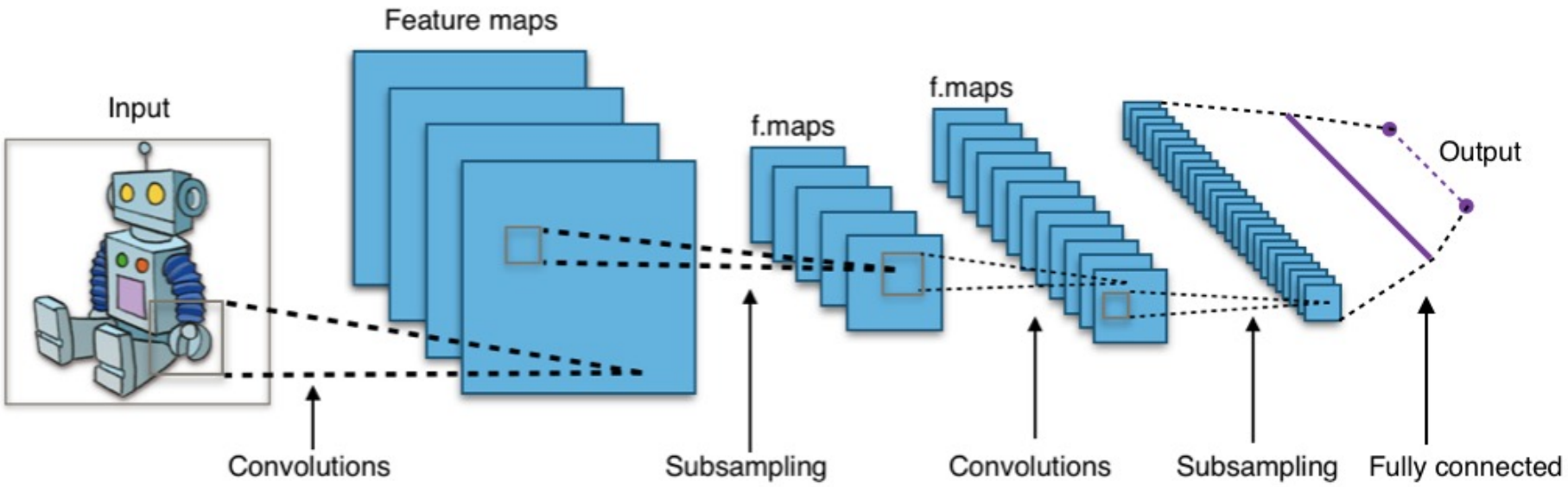
Self Driving Cars

Convolutional Neural Nets

Two additional operations:

- Convolution
- Pooling/Subsampling

Convolutional Neural Nets (CNNS)



https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png

CNNs: Convolution

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

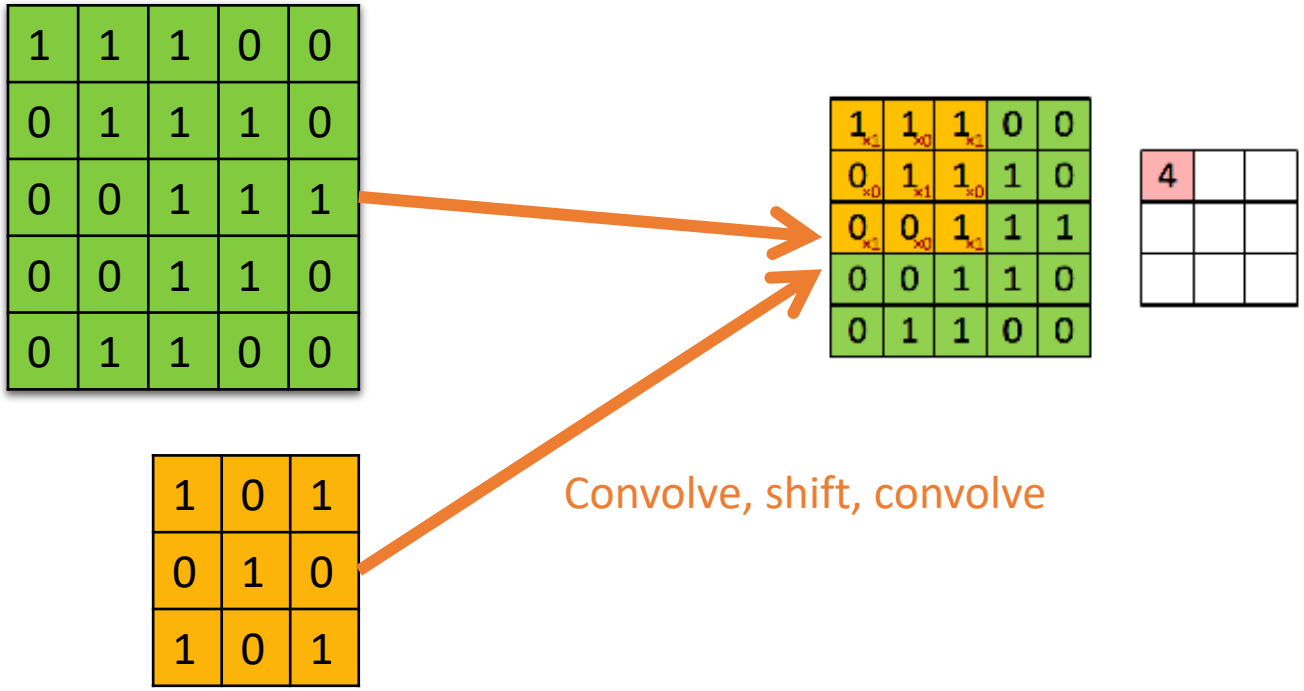
Input image (here binary, RGB typical)

1	0	1
0	1	0
1	0	1

Filter (here binary, but typically continuous values)

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

CNNs: Convolution



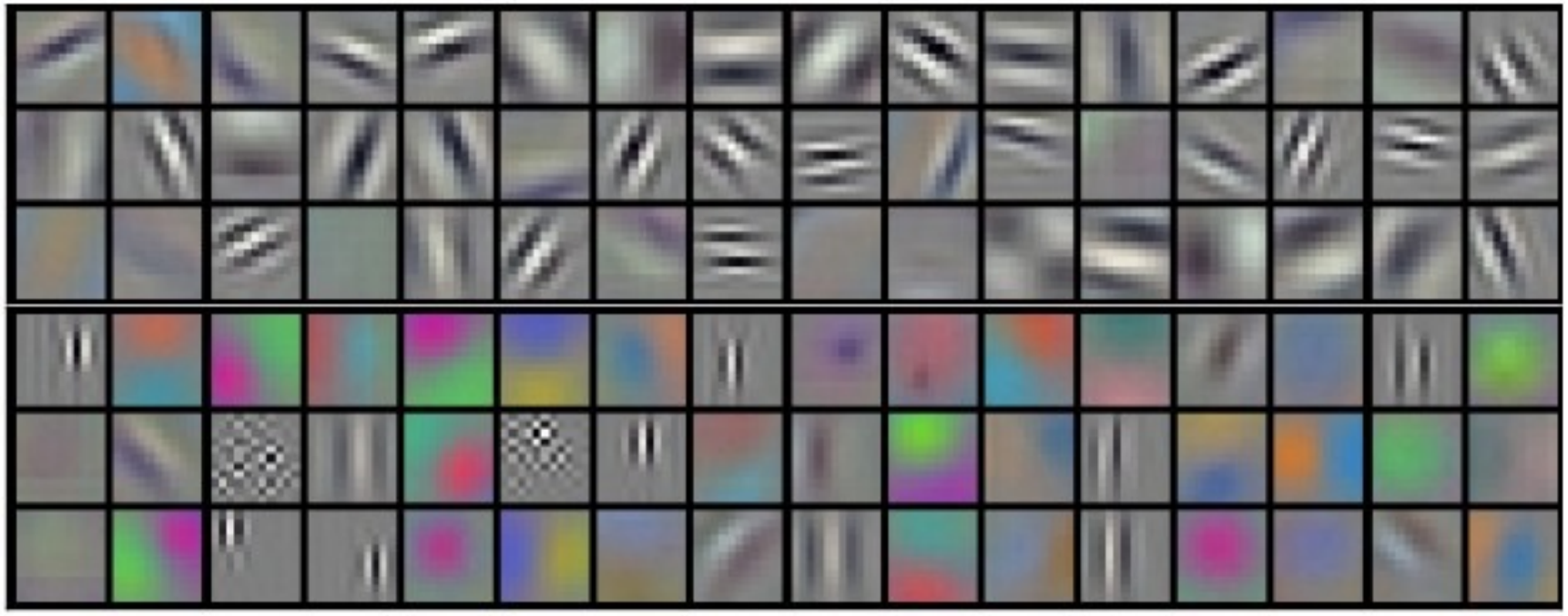
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

CNNs: Convolution

- Typical Filters ( )

1	0	1
0	1	0
1	0	1

Filters are learned

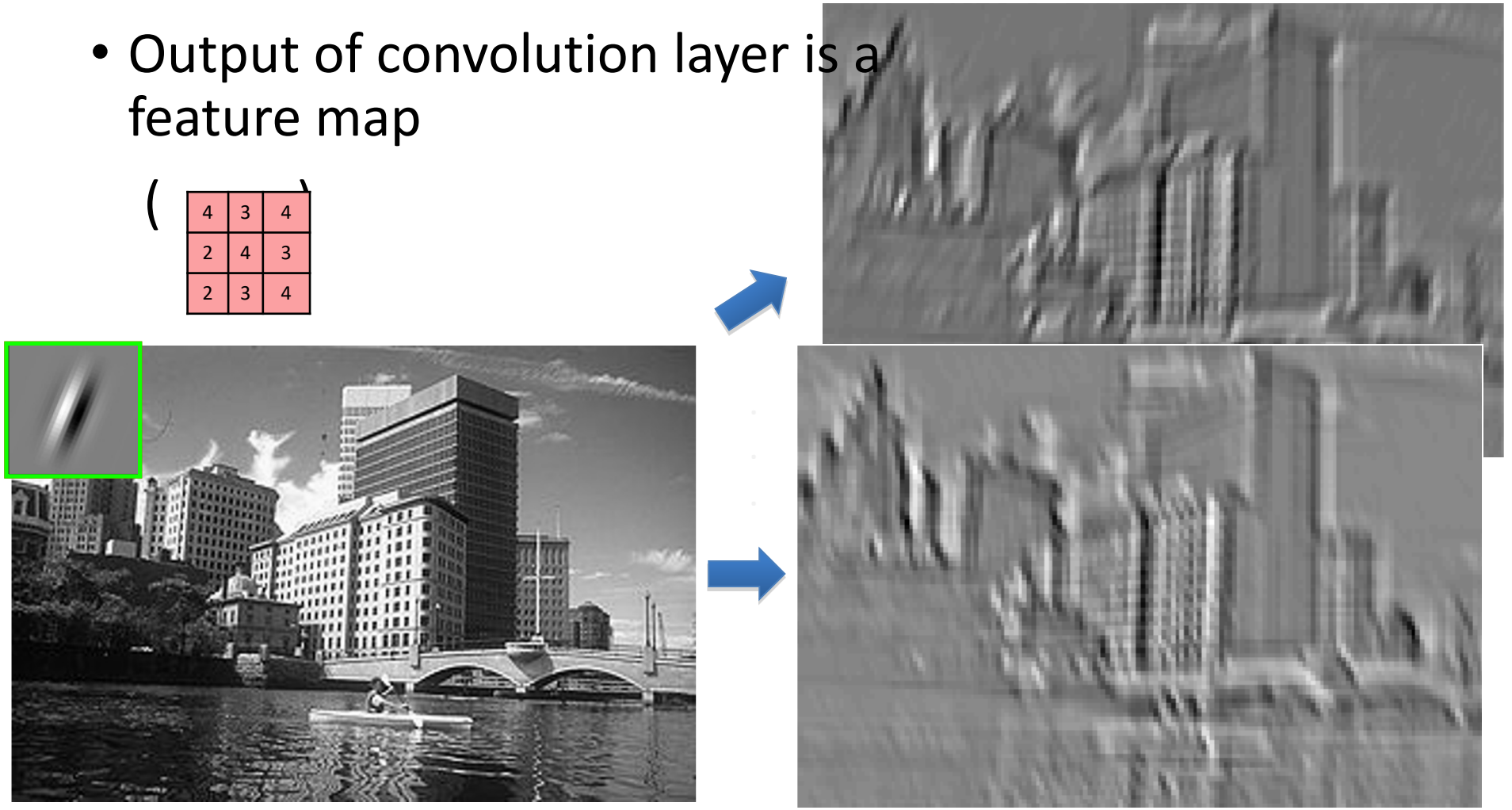


Krizhevsky et al.

CNNs: Convolution

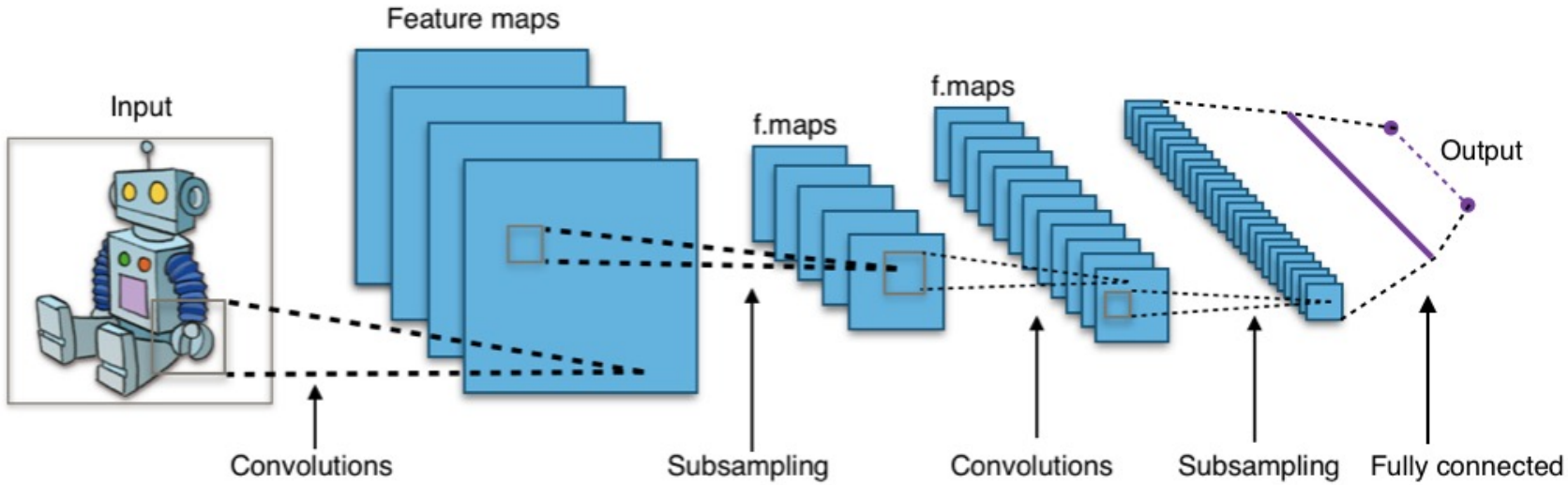
- Output of convolution layer is a feature map

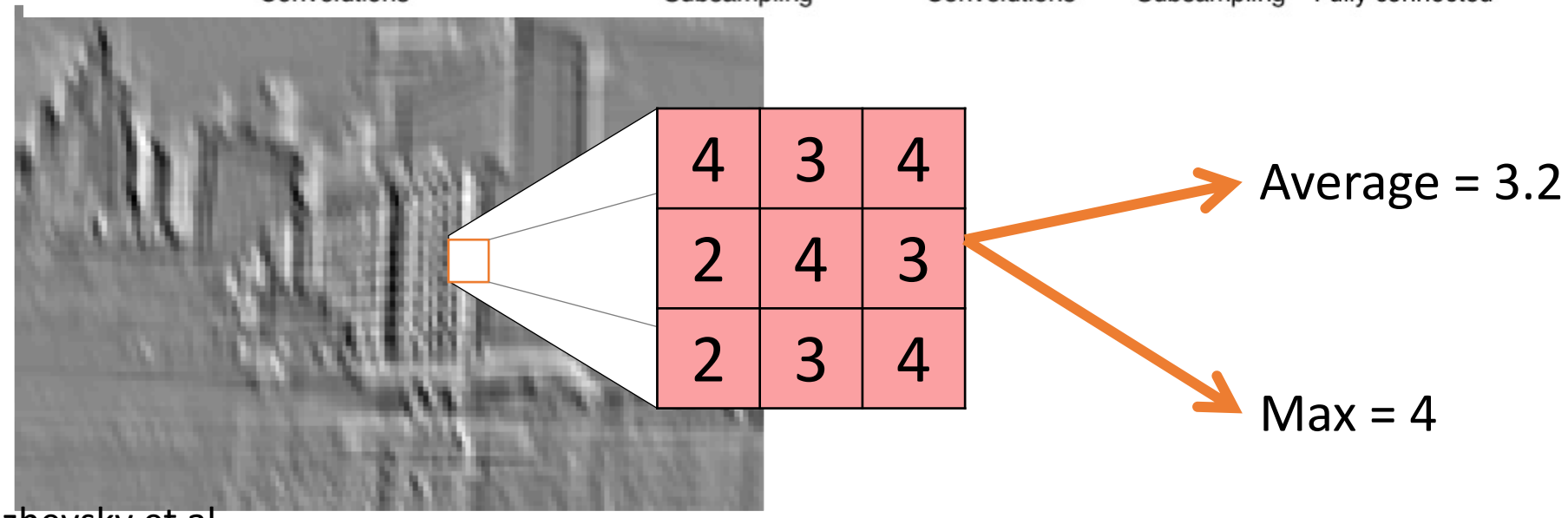
$$\begin{pmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{pmatrix}$$



http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/ (Fergus intro)

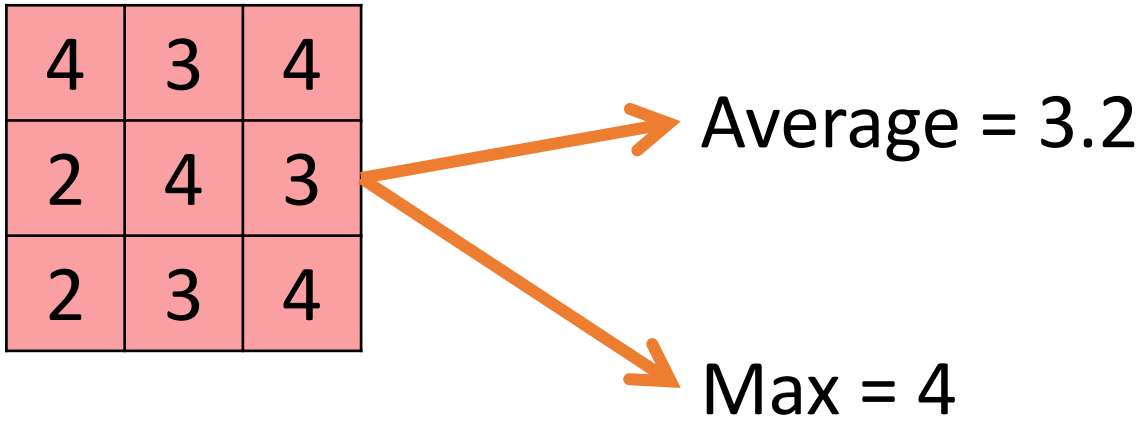
CNNs: Pool/Subsample





Krizhevsky et al.

CNNs: Pool/Subsample

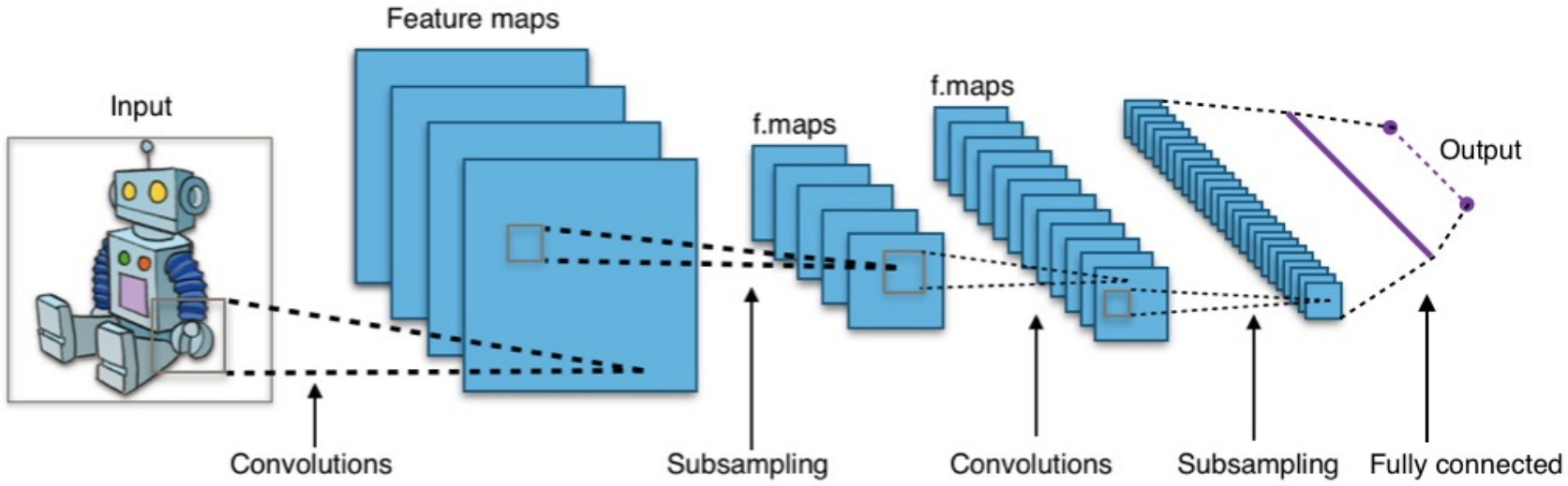


- Supplies some **invariance** to local translations
- Reduces the dimension of subsequent layers

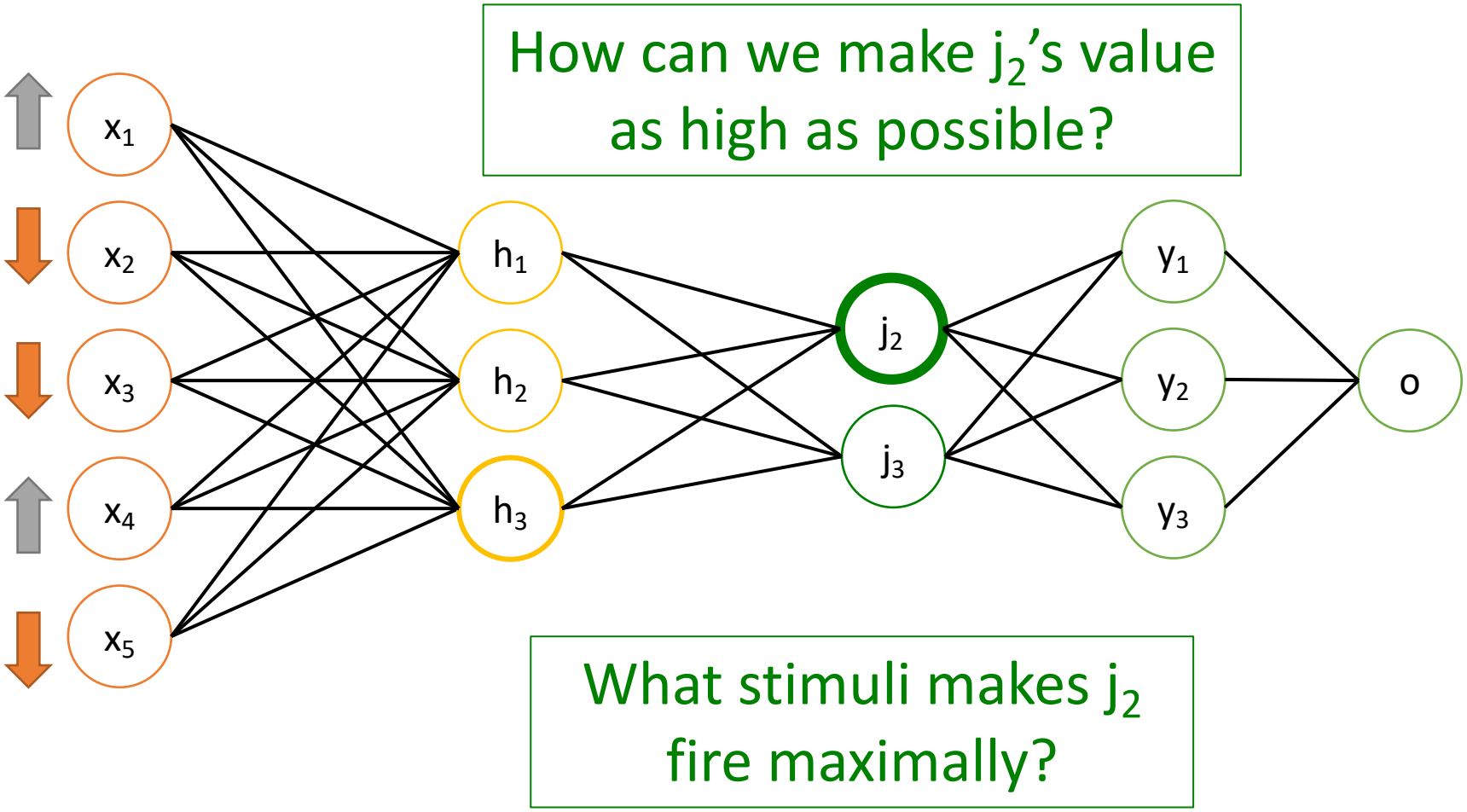
Krizhevsky et al.

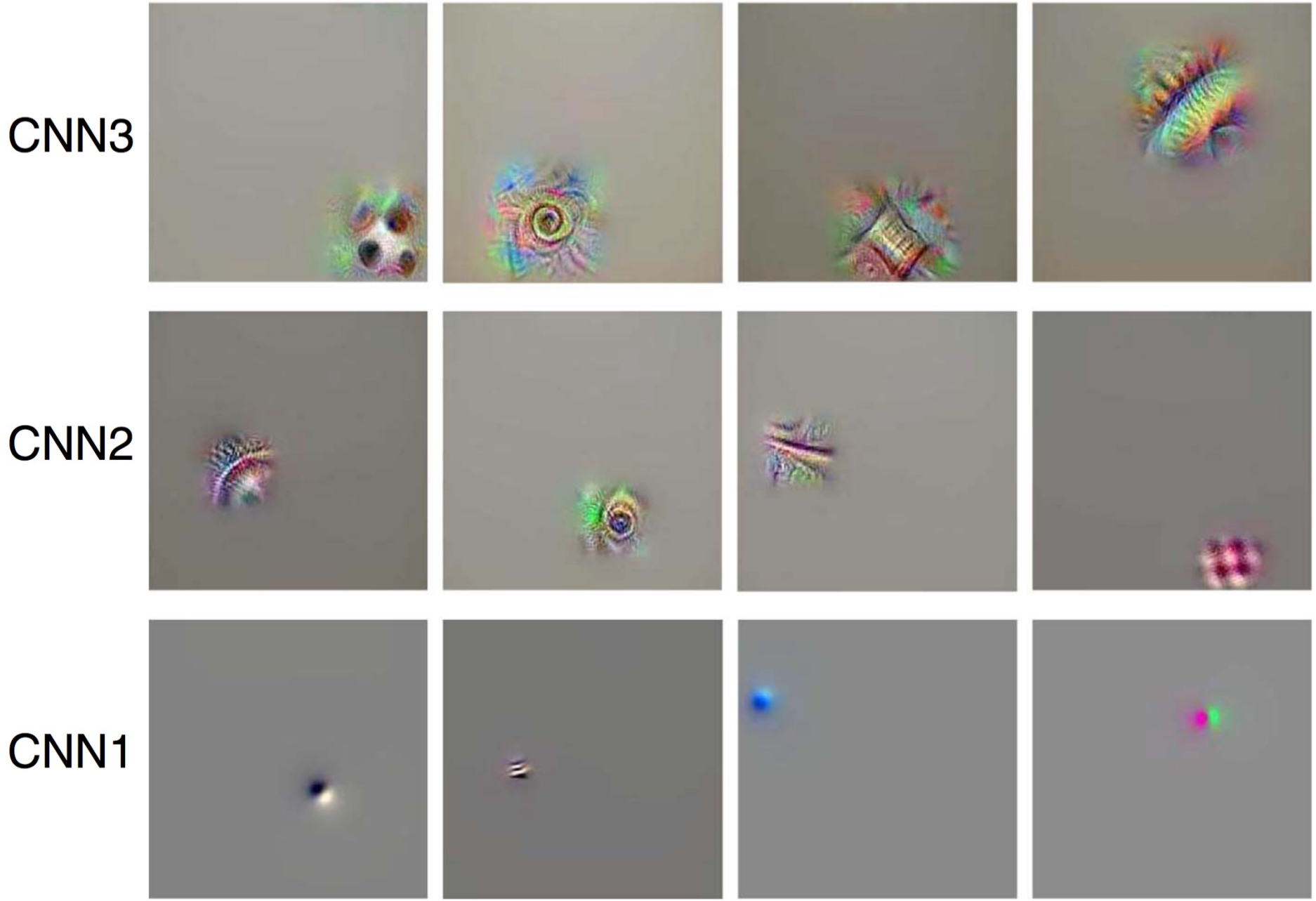
CNNs: Hidden representations

- Activations (feature maps) at each layer are a “hidden representation” of the image
- A compression of the information in image
  - not unlike PCA/SVD



What do the neurons represent?



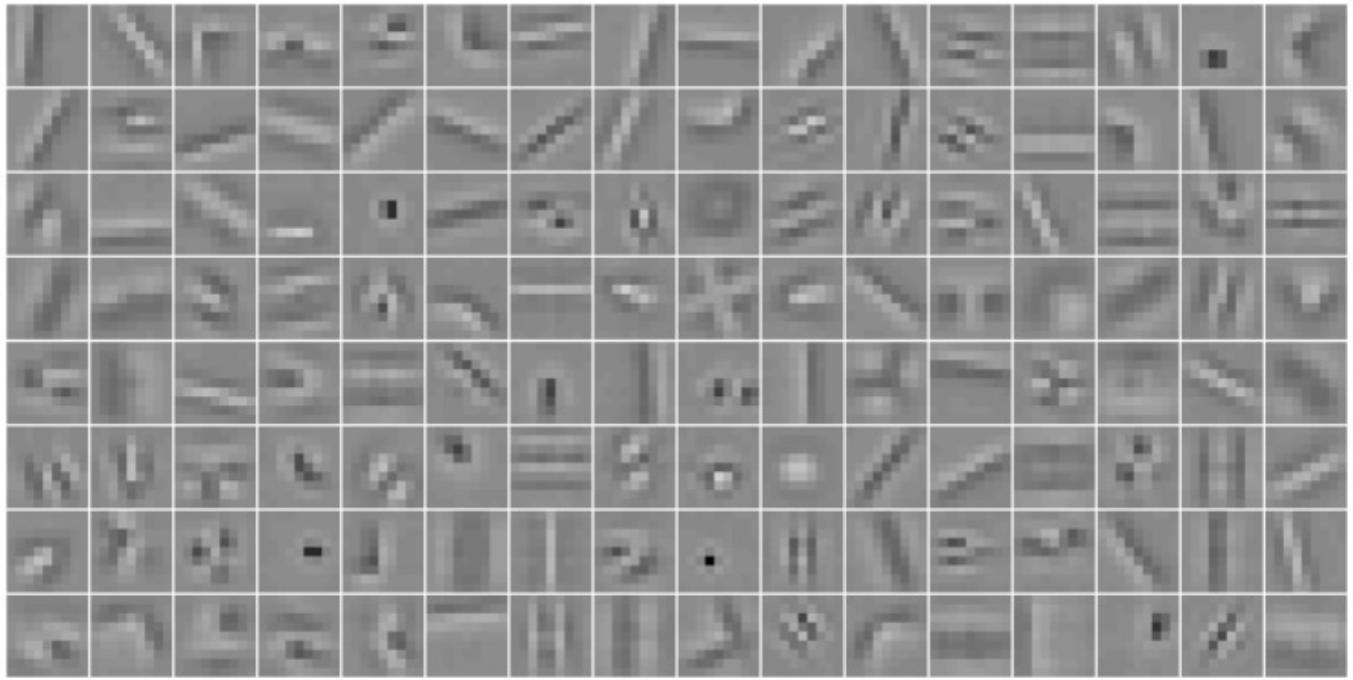


Horikawa & Kamitani (2017)

Horikawa & Kamitani (2017)

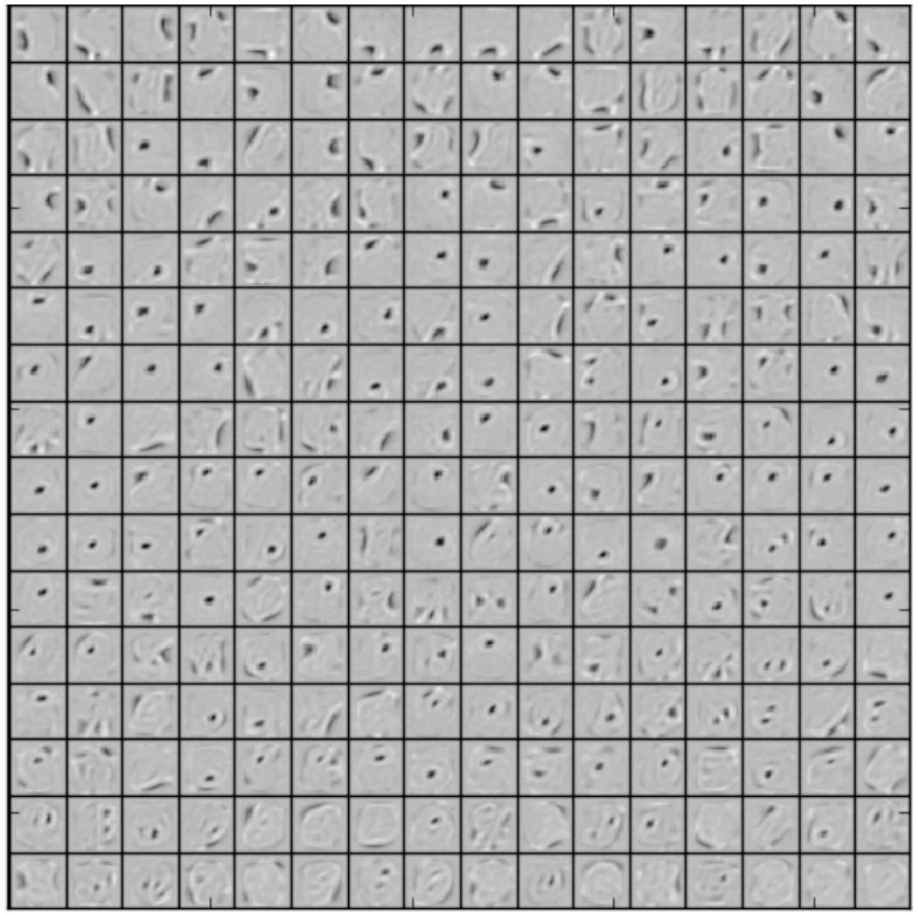
What do CNNs learn?

- When trained on images



http://yann.lecun.com/exdb/publis/pdf/lecun-eccv-12.pdf

What do CNNs Learn?



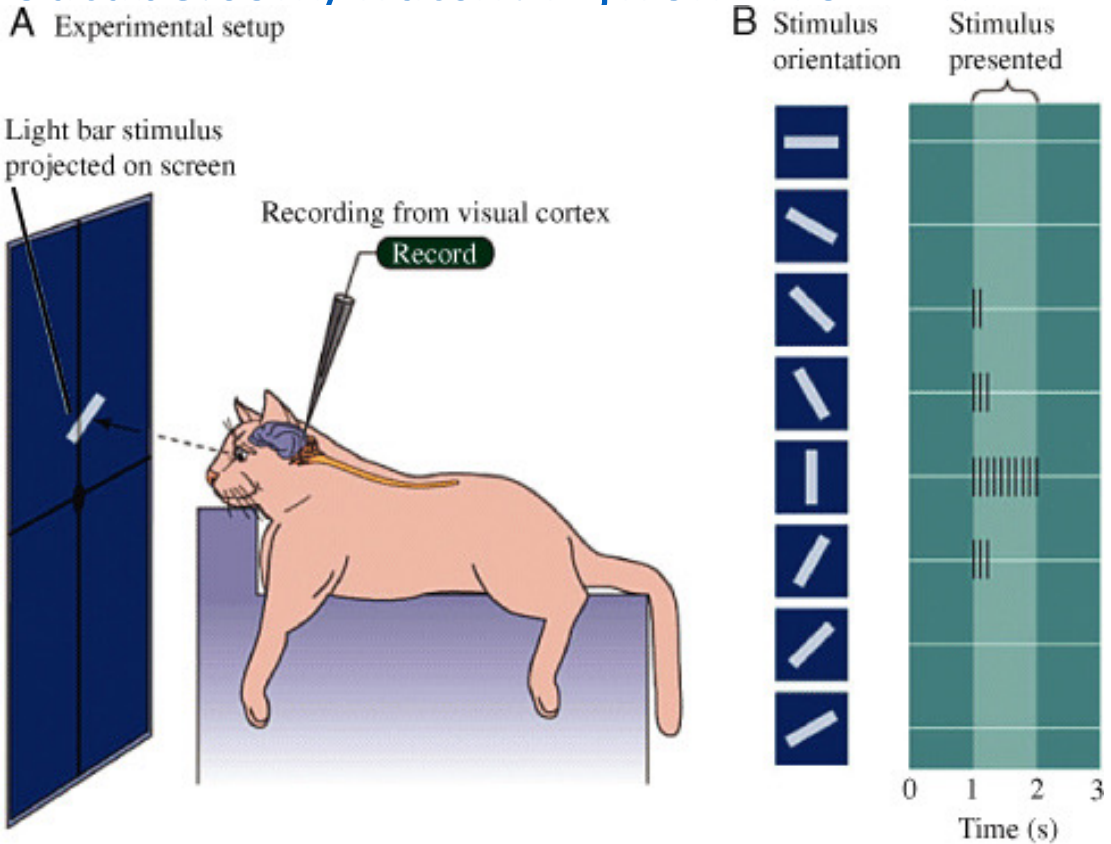
https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Interpretability

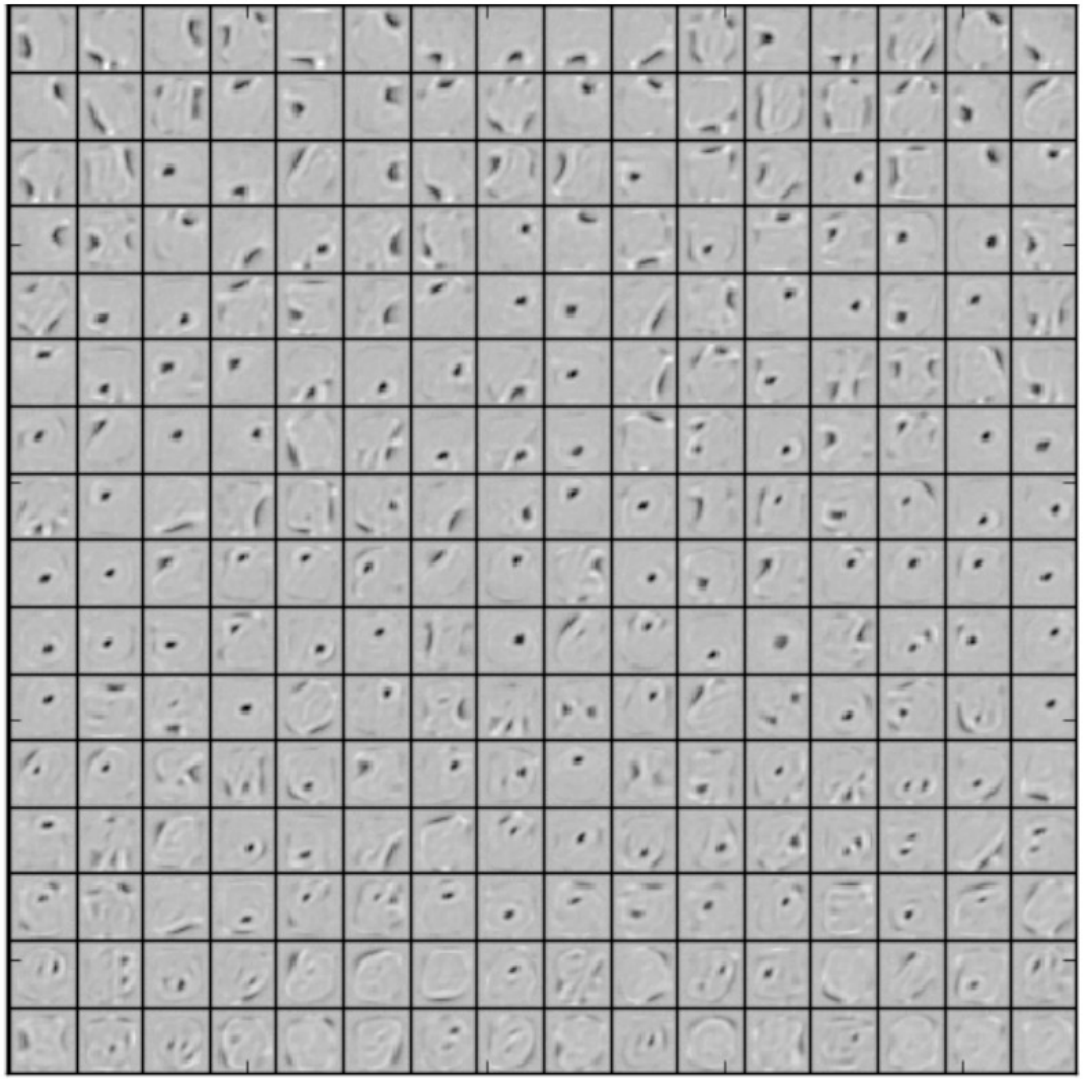
- Chris Olah has a great series on interpretability (the distill ones are particularly good)
  - https://colah.github.io/

An Aside

- In this video, the static noise you hear is a representation of the neurons firing in response to the visual stimulus
  - https://www.youtube.com/watch?v=jw6nBWo21Zk



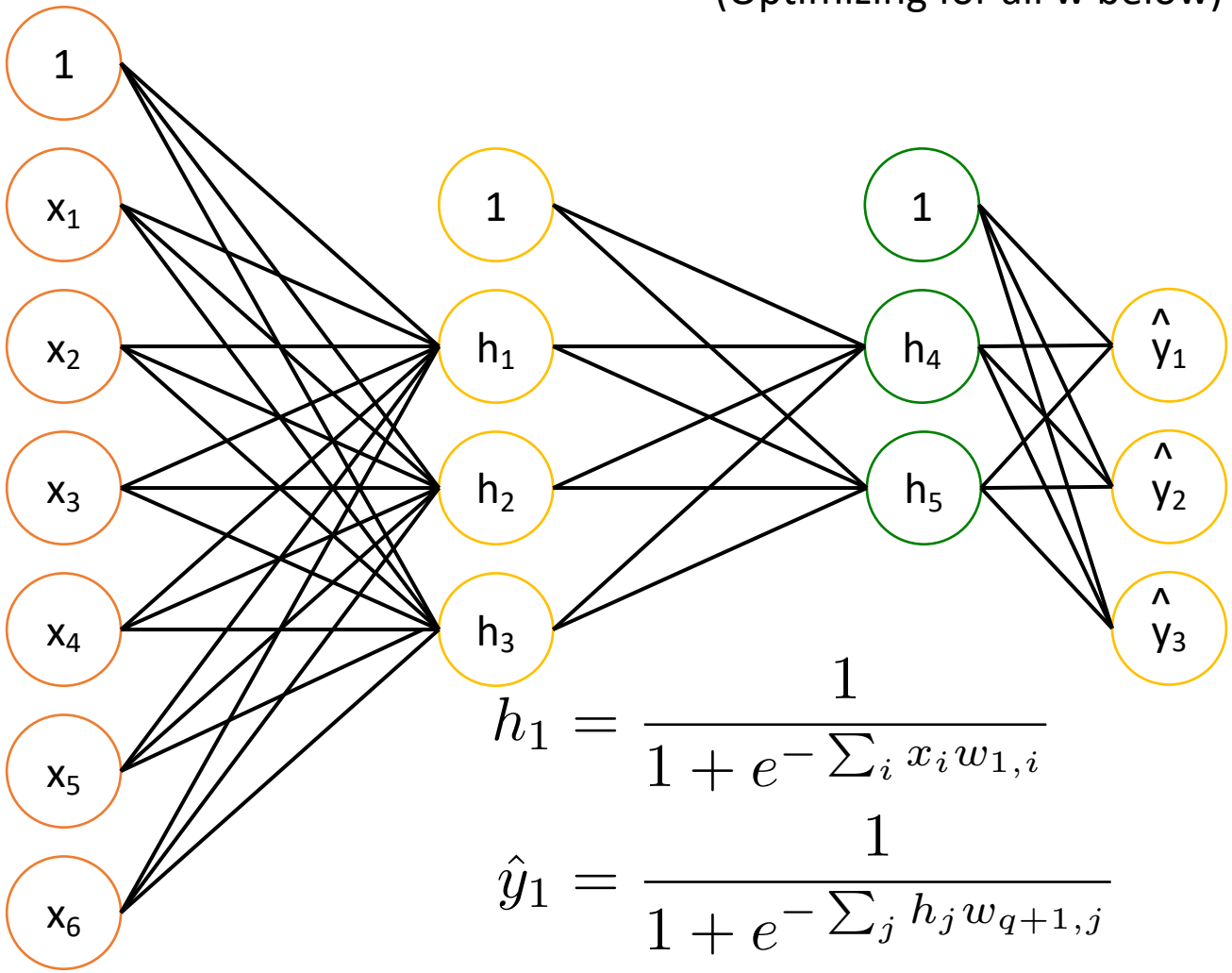
NNs learn something similar



https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Next time: Back Prop

(Optimizing for all w below)



$$h_1 = \frac{1}{1 + e^{-\sum_i x_i w_{1,i}}}$$

$$\hat{y}_1 = \frac{1}{1 + e^{-\sum_j h_j w_{q+1,j}}}$$

q is total # of hidden nodes across all layers

Resources

- Train very simple multi-layer perceptron for classification
  - https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- Programming resources for training your own NNs
  - Tensorflow https://www.tensorflow.org/
  - Keras https://keras.io/
  - Pytorch https://pytorch.org/
  - For intuition: http://playground.tensorflow.org/
- Short course on deep learning (Nando De Freitas)
  - https://www.youtube.com/playlist?list=PLjK8ddCbDMphIMSXn-w1IjyYpHU3DaUYw
- Commentary on AlphaGo
  - https://www.youtube.com/watch?v=UMm0XaCFTJQ
  - https://www.youtube.com/watch?v=g-dKXOlsf98
- Other fun videos
  - Geoff Hinton is in this one! Neural Net stuff is towards the end
    - https://www.youtube.com/watch?v=yxxRAHVtafI
  - Fei Fei Li's Ted Talk (Creator of ImageNet)
    - https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures?language=en