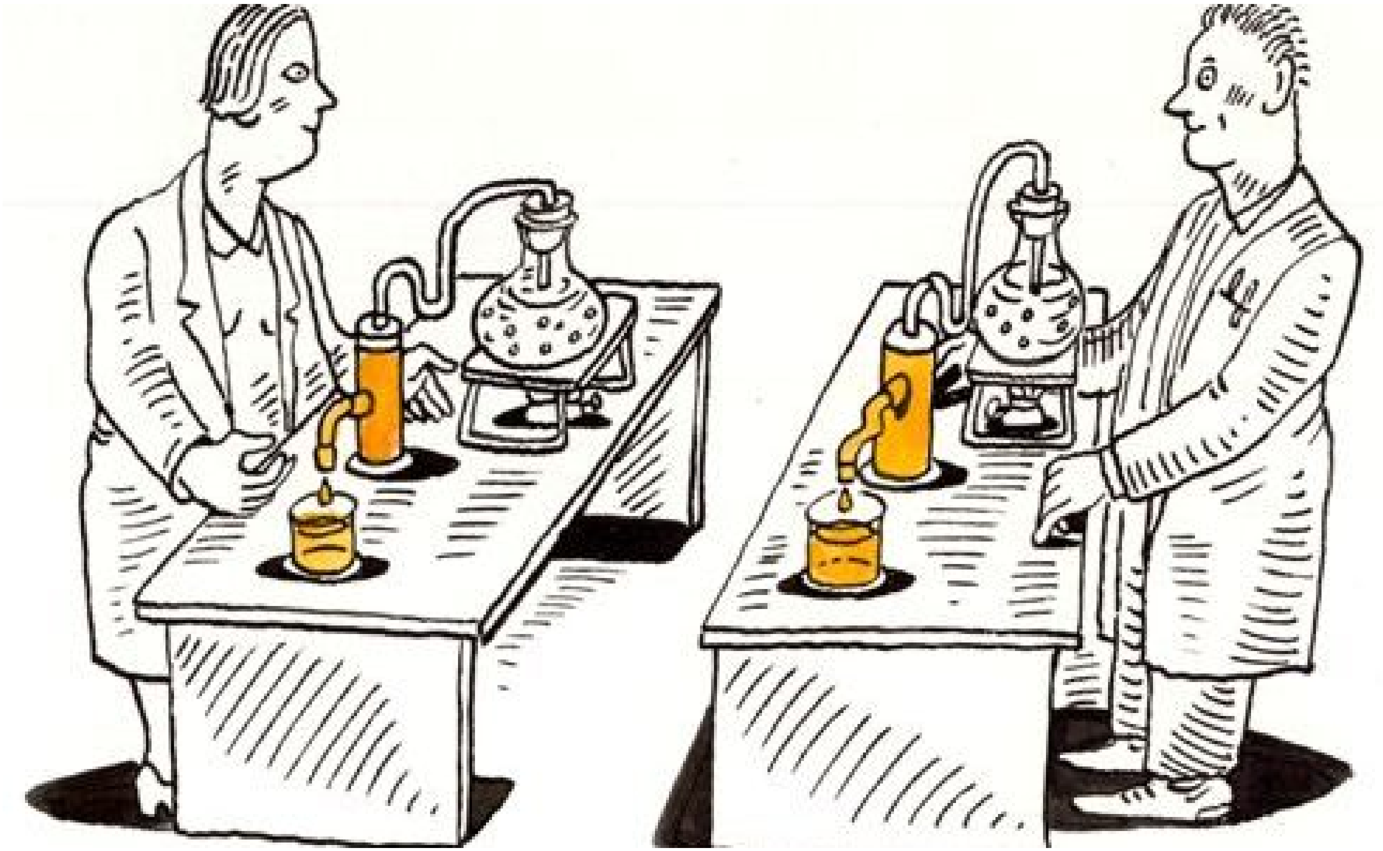
https://www.youtube.com/watch?v=flzGjnJLyS0

Lecture 9: **Logistic Regression** ***& Deep Neural Networks***

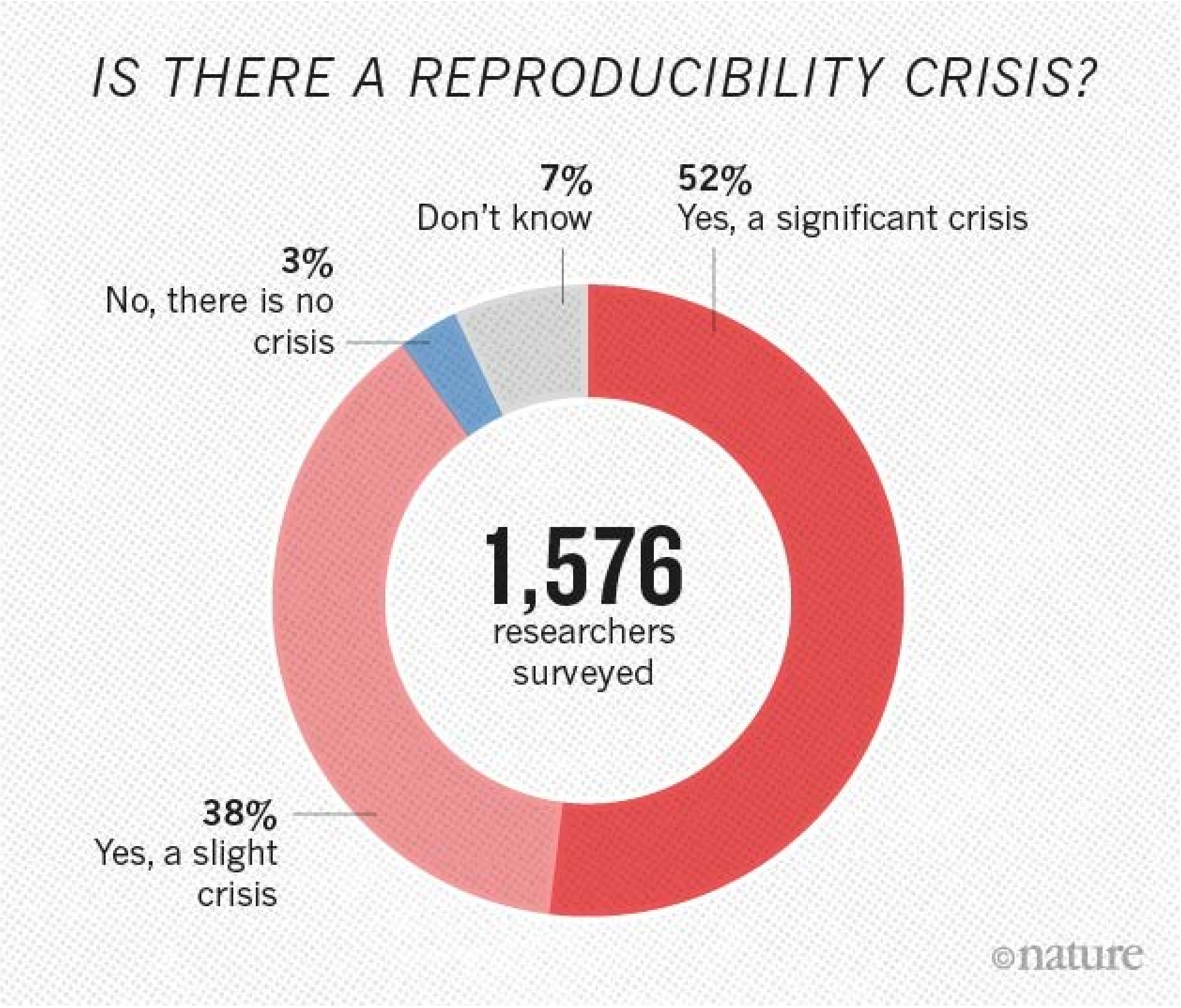
Haiping Lu’s ***last*** lecture @ MLAI19

Reproducibility



https://www.ucl.ac.uk/pals/research/experimental-psychology/wp-content/uploads/2016/03/reproducibility-small-496x300.jpg

Reproducibility



https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970

PCA: Eigenvalue → Variance

- First PC: **maximise the variance** in the **projected** space, i.e., the variance of $y = \mathbf{u}^\top \mathbf{x}$

$$\mathrm{var}(y) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \mu_y\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\mathbf{u}^\top\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}_{\mathbf{x}}\right)\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}_{\mathbf{x}}\right)^\top \mathbf{u} = \mathbf{u}^\top \mathbf{S}\mathbf{u}$$

- To learn/estimate the *projection vector* **u**, we incorporate the unit norm constraint and take derivative w.r.t. **u**

$$L\left(\mathbf{u},\lambda\right) = \mathbf{u}^\top\mathbf{S}\mathbf{u} + \lambda\left(1 - \mathbf{u}^\top\mathbf{u}\right)$$
$$\mathrm{d}L\left(\mathbf{u},\lambda\right)/\mathrm{d}\mathbf{u} \implies \mathbf{S}\mathbf{u} = \lambda\mathbf{u}$$

- **Question**: many solutions (eigen-pairs), which to choose?

$$\mathrm{var}(y) = \mathbf{u}^\top\mathbf{S}\mathbf{u} = \lambda\mathbf{u}^\top\mathbf{u} = \lambda$$

PCA: Max Variance ←→ Min MSE

Consider the first PC: Here the projection vector is denoted as **v**.
We assume **zero-mean**, i.e., *centered* data

Maximum Variance Direction: 1st PC = a vector **v** such that projection on it captures max variance in the data

$$\frac{1}{n}\sum_{i=1}^{n}(\mathbf{v}^T\mathbf{x}_i)^2 = \mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v}$$

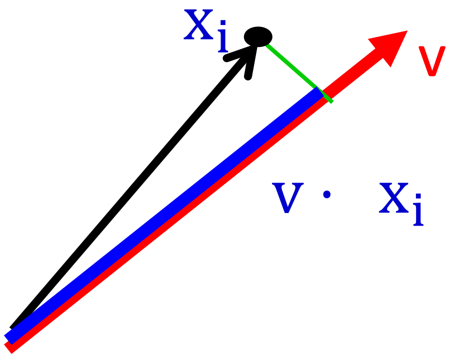
Minimum Reconstruction Error: 1st PC = a vector **v** such that projection on it yields minimum MSE reconstruction

$$\frac{1}{n}\sum_{i=1}^{n}\|\mathbf{x}_i - (\mathbf{v}^T\mathbf{x}_i)\mathbf{v}\|^2$$

blue^2 + green^2 = black^2

black^2 is fixed (it's just the data)

So, maximizing blue^2 is equivalent to minimizing green^2



Week 9 Contents / Objectives

Part A

- Motivation for Logistic Regression
- Logistic Regression

Part B

- Neural Networks
- Convolutional Neural Networks Unboxing

Week 9 Contents / Objectives

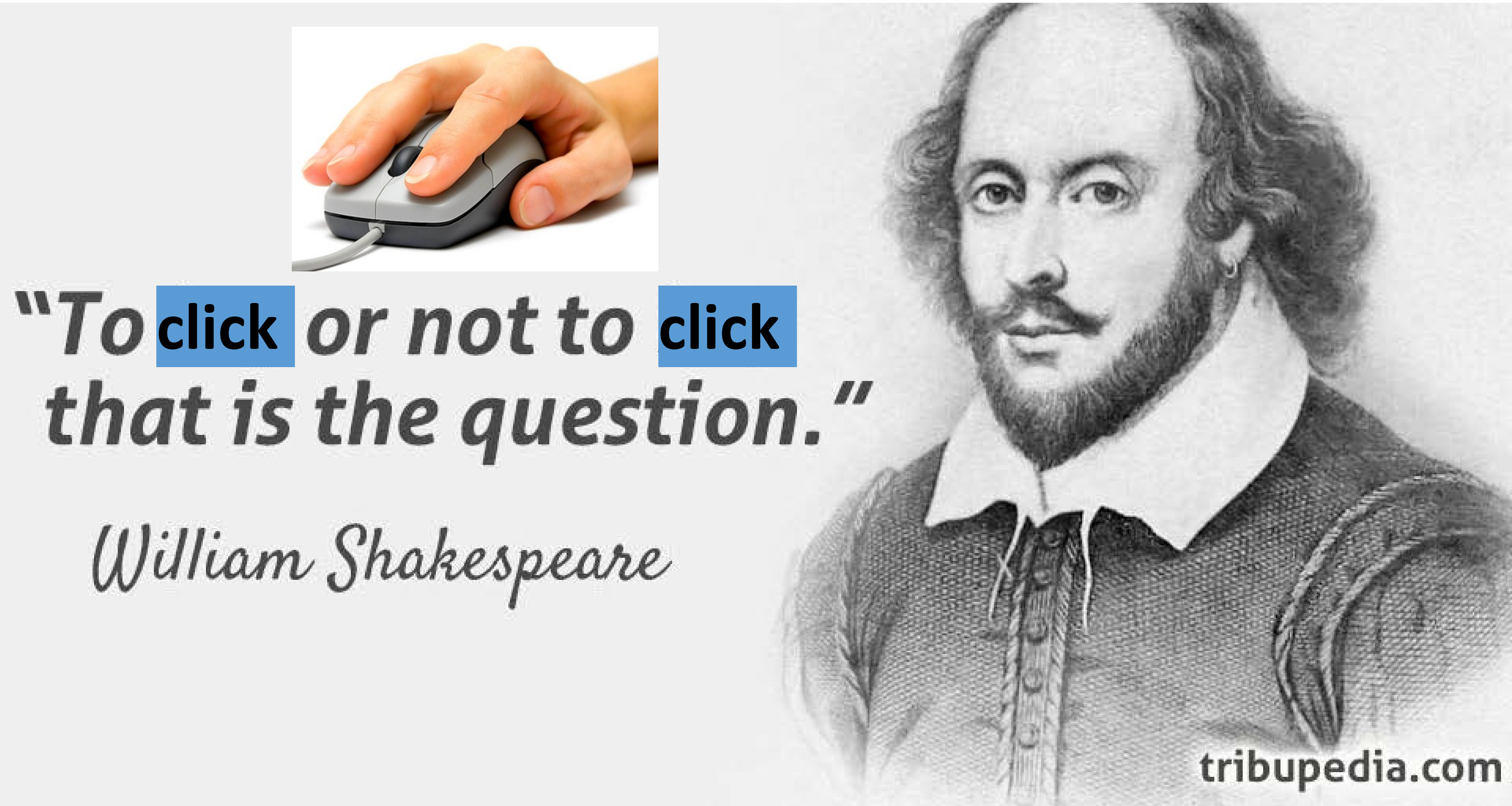
Part A

- **Motivation for Logistic Regression**
- Logistic Regression

Part B

- Neural Networks
- Convolutional Neural Networks Unboxing

The Question



Click-Through Rate (CTR) Prediction

- Estimating click probabilities: What is the probability that user *i* **will click** on ad *j*
  - Not important just for ads:
    - Optimize search results
    - Suggest news articles
    - Recommend products

- Logistic regression is used by many internet companies, making lots of money for them
  - E.g., Facebook ad matching

A Binary Classification Problem

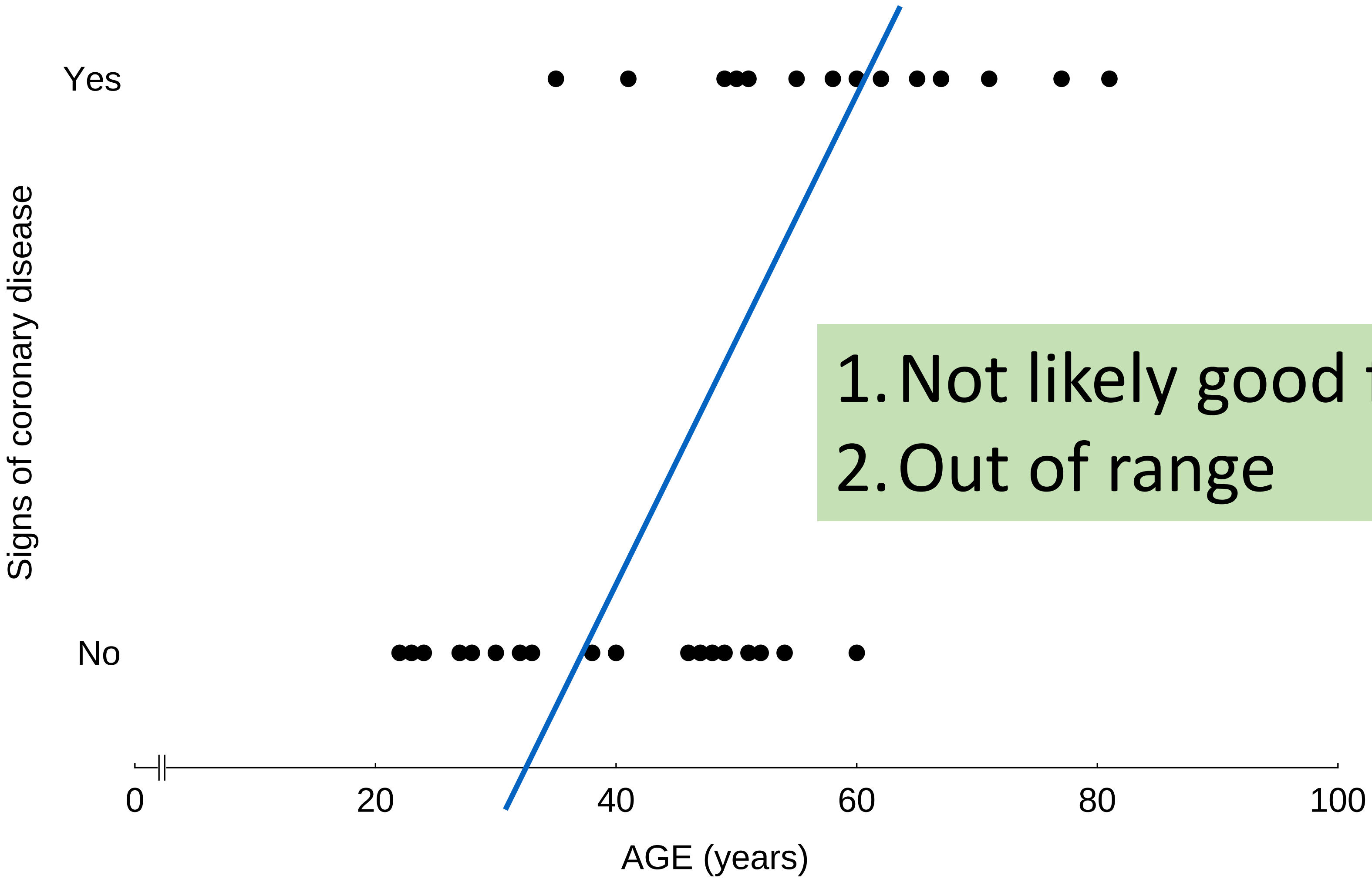
Table 1: Age and signs of coronary heart disease (CD)

Age	CD	Age	CD	Age	CD
22	0	40	0	54	0
23	0	41	1	55	1
24	0	46	0	58	1
27	0	47	0	60	1
28	0	48	0	60	0
30	0	49	1	62	1
30	0	49	0	65	1
32	0	50	1	67	1
33	0	51	0	71	1
35	1	51	1	77	1
38	0	52	0	81	1

Prediction question: a particular age → CD

Linear regression?

Dot-plot: Data from Table 1



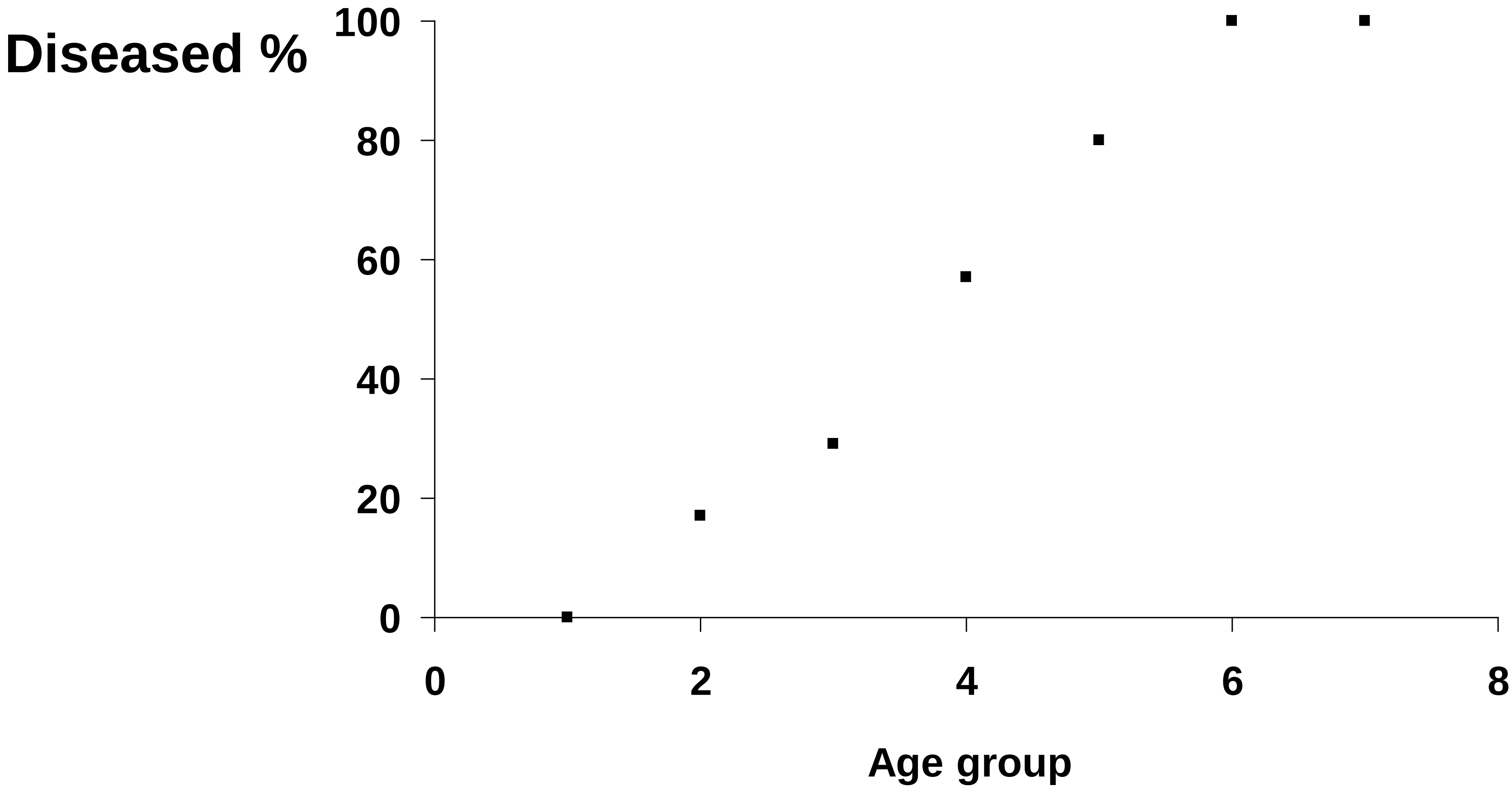
1. Not likely good fit
2. Out of range

Transform the Data→ Probability

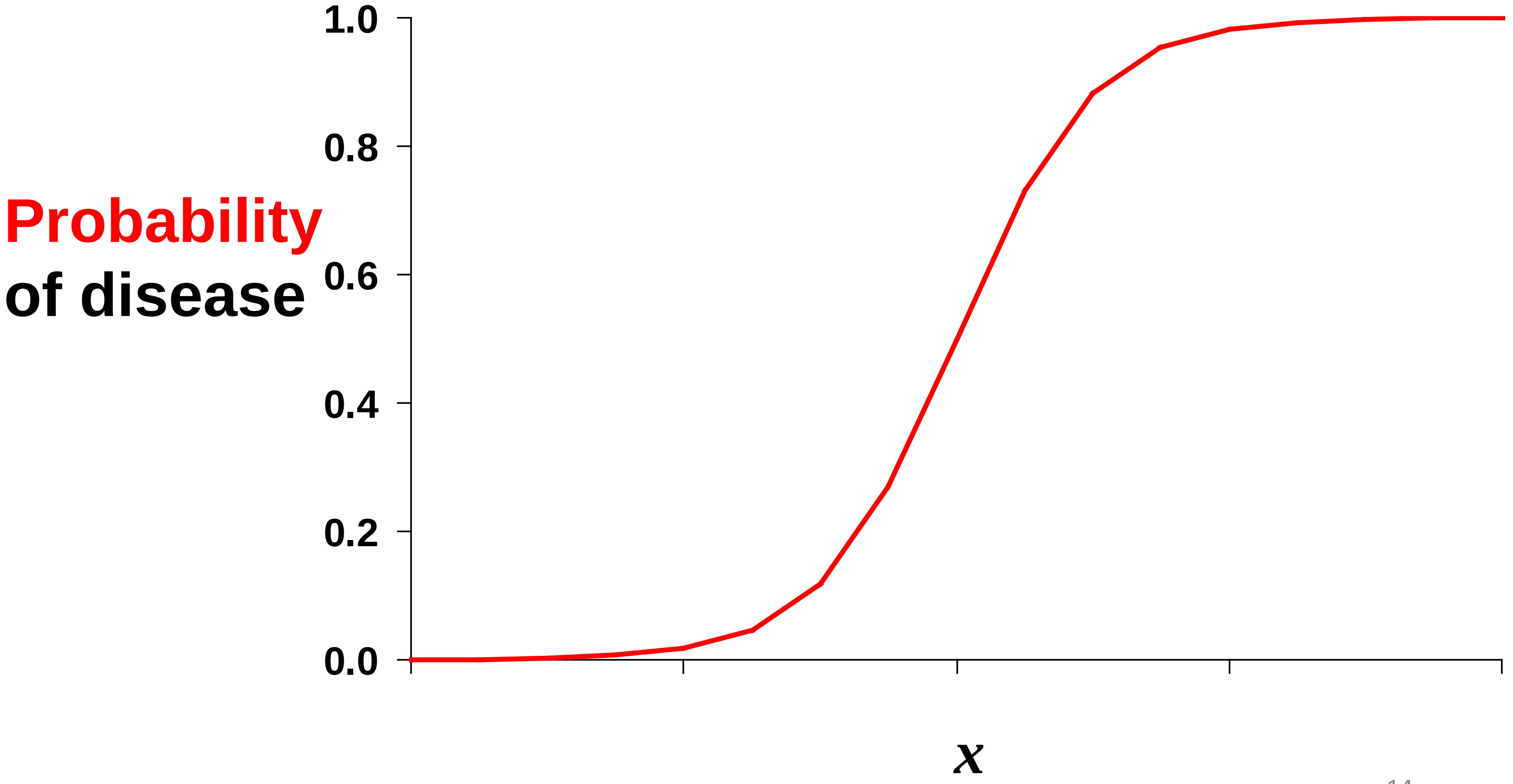
Table 2   Prevalence (%) of signs of CD according to age group

		Diseased	
Age group	**# in group**	**#**	**%**
20 - 29	5	0	0
30 - 39	6	1	17
40 - 49	7	2	29
50 - 59	7	4	57
60 - 69	5	4	80
70 - 79	2	2	100
80 - 89	1	1	100

Dot-plot: Data from Table 2



Logistic Function



Week 9 Contents / Objectives

Part A

- Motivation for Logistic Regression
- **Logistic Regression**

Part B

- Neural Networks
- Convolutional Neural Networks Unboxing

Probabilistic Classification

- Training classifiers: estimating f: X → Y, or P(Y|X)
- **Discriminative** classifiers
  - Assume some functional form for P(Y|X)
  - Estimate parameters of P(Y|X) directly from training data
- **Generative** classifiers
  - Assume some functional form for P(X|Y), P(X)
  - Estimate parameters of P(X|Y), P(X) directly from training data
  - Use Bayes rule to calculate $P(Y|X = x_i)$

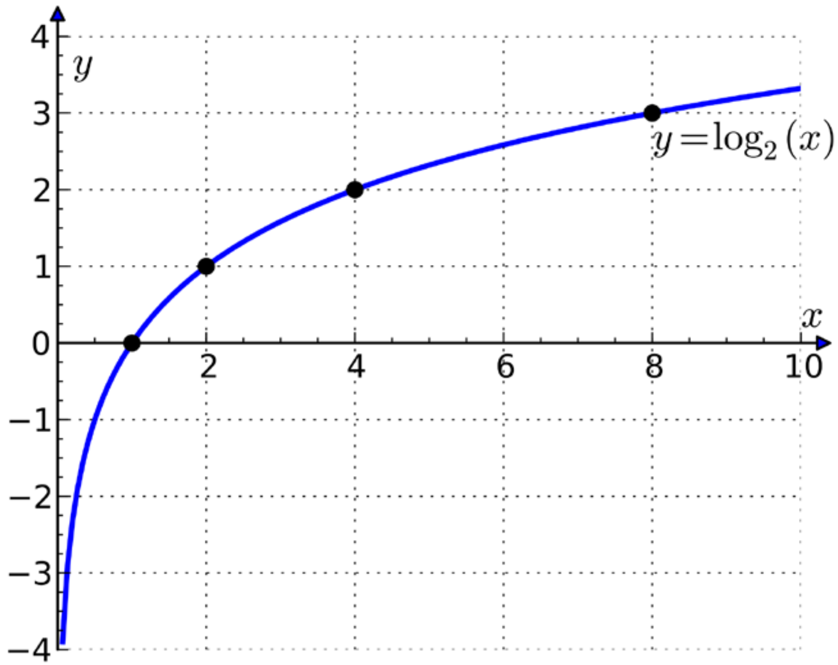
Log Odds

- **Odds**: the ratio of π, the probability of a positive outcome *P*(*y=1*|**x*)***, to *(1- π)*, the probability of a negative outcome *P*(*y=0*|**x*)***.

$$\frac{\pi}{1-\pi}$$

- Probability: [0, 1]
- → Odds: [0, ∞]
- → Log odds: **[-∞, ∞]**

$$\log \frac{\pi}{1-\pi}$$



Logit Function→ Logistic Function

- Linear **regression** on **logit** function = logistic *regression*

$$\text{logit}(\pi) = \log \frac{\pi}{1-\pi} = \mathbf{w}^\top \mathbf{x} = w_0 + w_1 x_1 + \cdots$$

- More generally, we can use **basis function** as

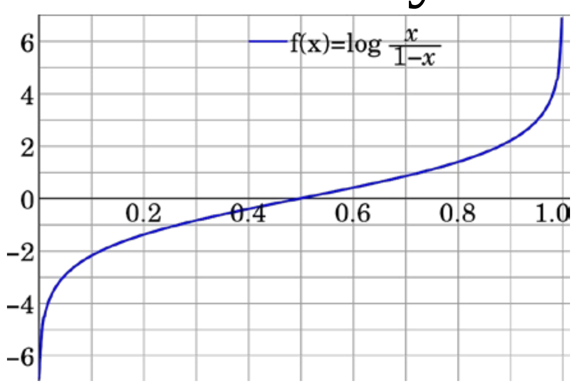
$$\text{logit}(\pi) = \log \frac{\pi}{1-\pi} = \mathbf{w}^\top \phi(\mathbf{x}) = w_0 + w_1 \phi(x_1) + \cdots$$

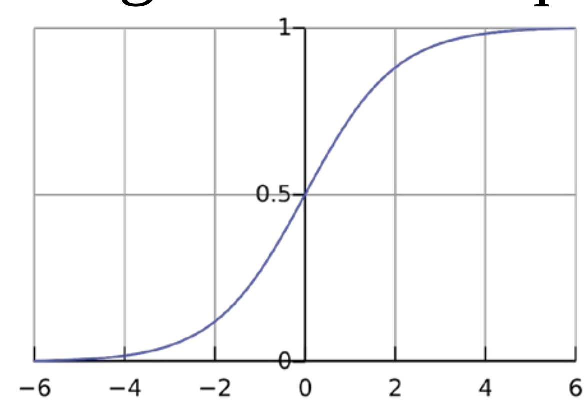
In the following, we use the simpler first form above

- Logistic function (sigmoid)= inverse of logit

$$P(y=1|\mathbf{x}) = \text{logit}^{-1}(\mathbf{w}^\top \mathbf{x}) = \text{logistic}(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$$

- **Exercise**: verify the odds using the above equation





How to Estimate **w**? (Learning algo)

- Assumption: Conditional independence of data
- → Likelihood: $$P(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}_i)$$
- **Bernoulli distribution** for binary classification
  - $P(y=1) = \pi$; $P(y=0) = 1 - \pi$ (coin flipping)
  - Write the above as a single equation: using *y* as a switch

$$P(y) = \pi^y (1-\pi)^{(1-y)} \qquad \pi_i = P(y_i = 1|\mathbf{x}_i)$$

- Log likelihood (*cross entropy*)

$$\log P(\mathbf{y}|\mathbf{X}) = \sum_{i=1}^{n} \log P(y_i|\mathbf{x}_i) = \sum_{i=1}^{n} y_i \log \pi_i + \sum_{i=1}^{n} (1-y_i) \log(1-\pi_i)$$

- MLE: no closed form solution

  → SGD on negative log likelihood

Summary on Logistic Regression

- **Discriminative** classifiers directly model the likelihood $P(Y|X)$

- Logistic regression is a simple **linear** classifier, that retains a **probabilistic** semantics (see lab)

- Parameters in LR are learned by **iterative** optimization (e.g. SGD), no closed-form solution

Week 9 Contents / Objectives

Part A

- Motivation for Logistic Regression
- Logistic Regression

Part B

- **Neural Networks**
- Convolutional Neural Networks Unboxing

ImageNet I 010 (

Fancy feature extraction **/representation learning**

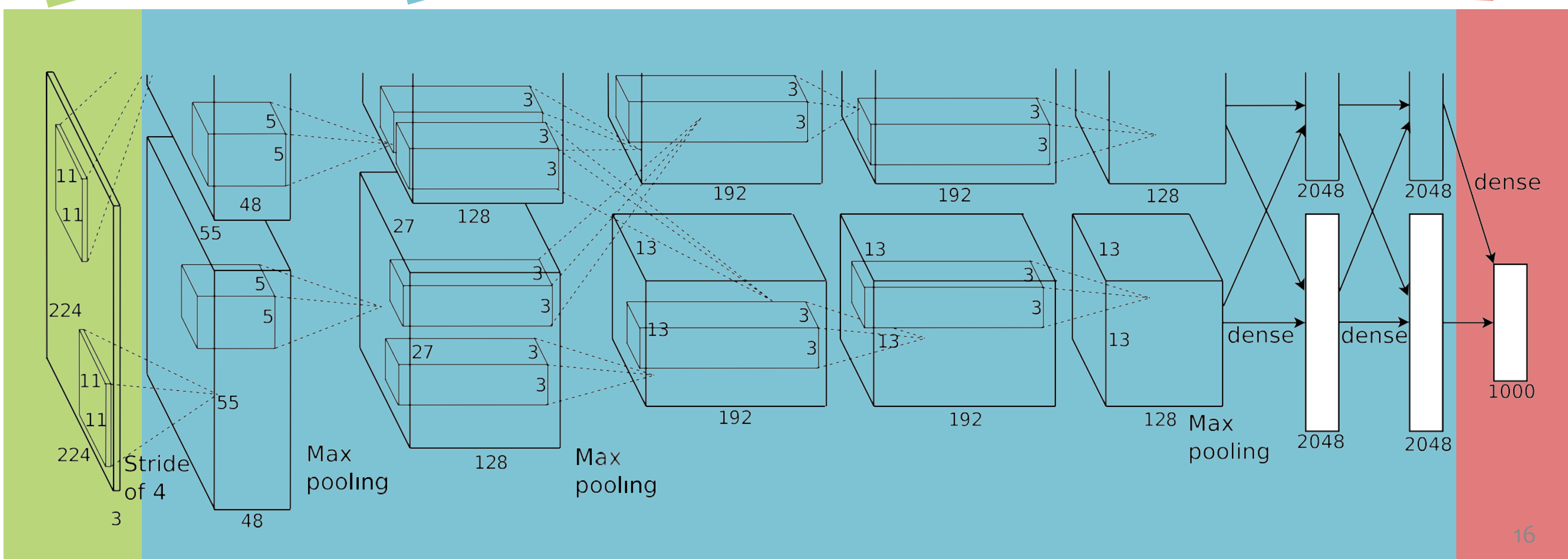
Logistic Regression!
Softmax: sigmoid for multiclass

Dataset: 1.2 million 1000 cla
CNN for Image Clas evsky, S
Hinton, 2011) → 17.5% error

Input image (pixels)

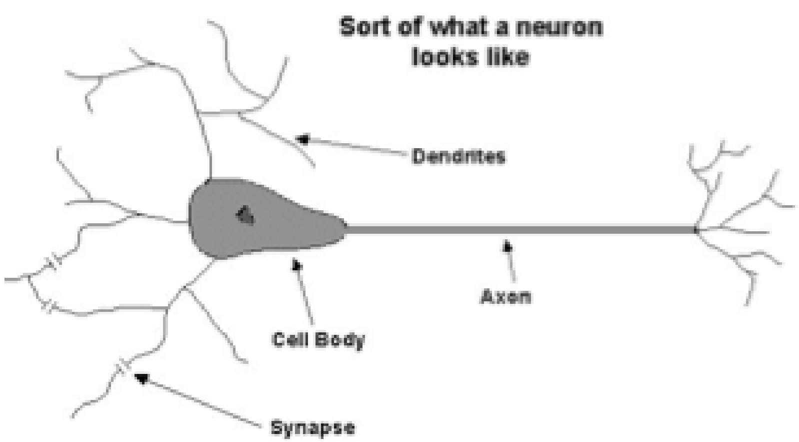
- Five convolutional layers (w/max-pooling)
- Three fully connected layers

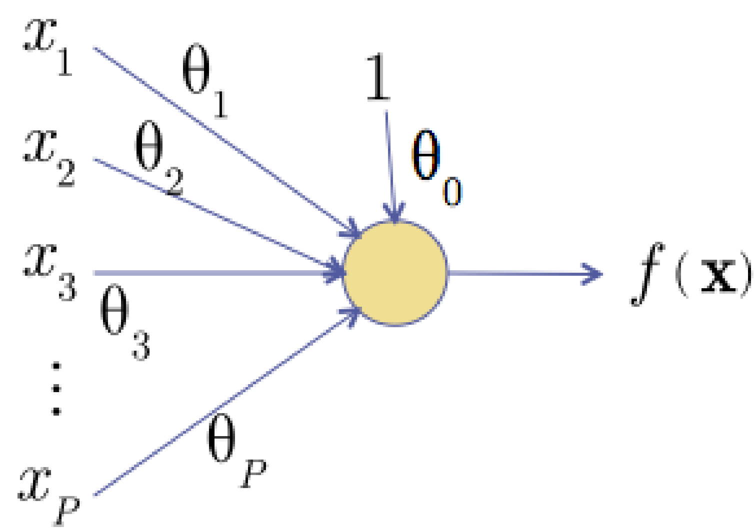
1000-way softmax



Pigeon? The Neuron Metaphor

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node





Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

Face

Face

Not a face

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

Examples: Linear regression, Logistic regression, Neural Network

Examples: Mean-squared error, Cross Entropy

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Background

A Recipe for

Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode *automatic differentiation* that can compute the gradient of any differentiable function efficiently!

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of th

– Decision functio

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

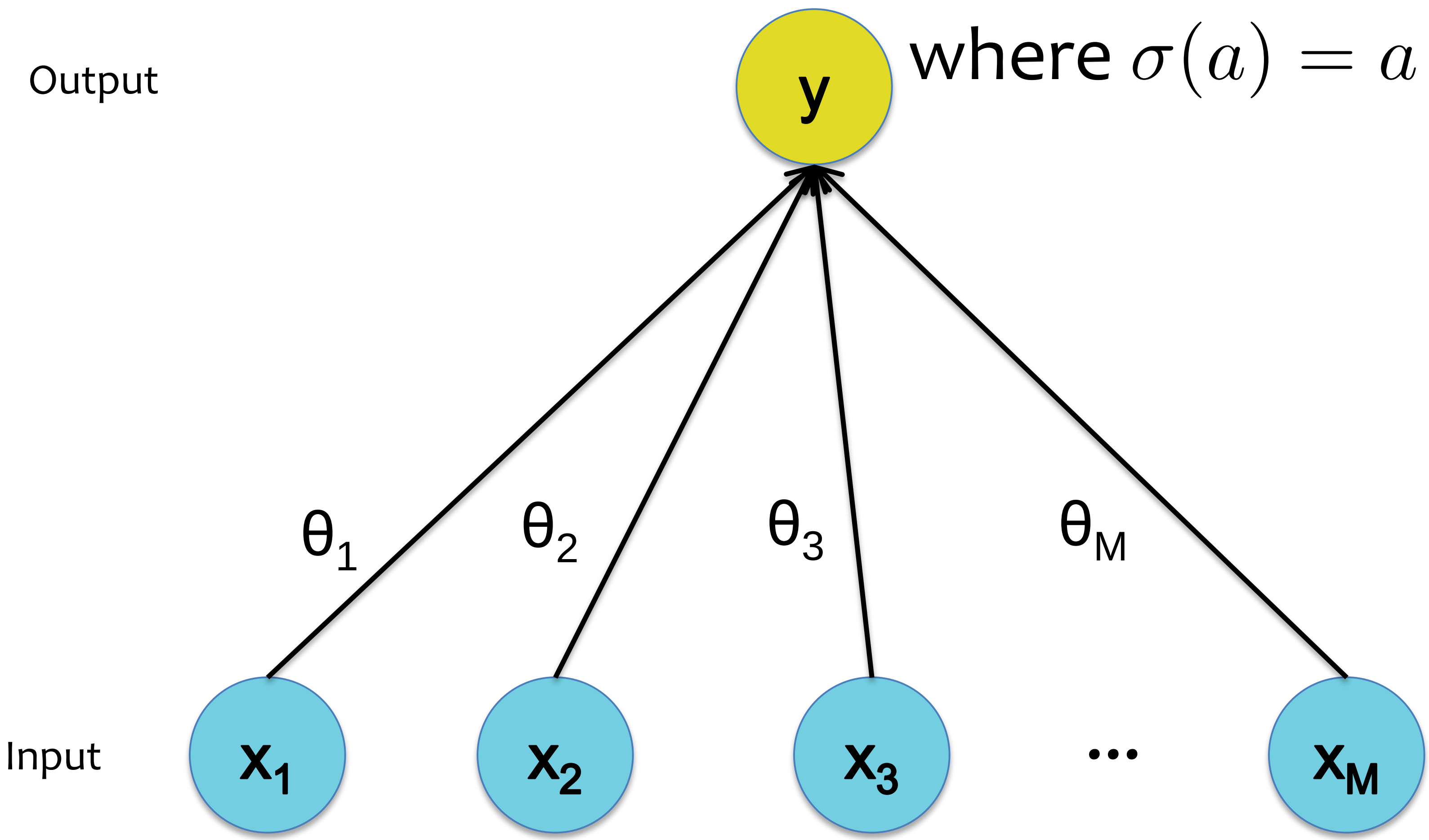
opposite the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Linear Regression

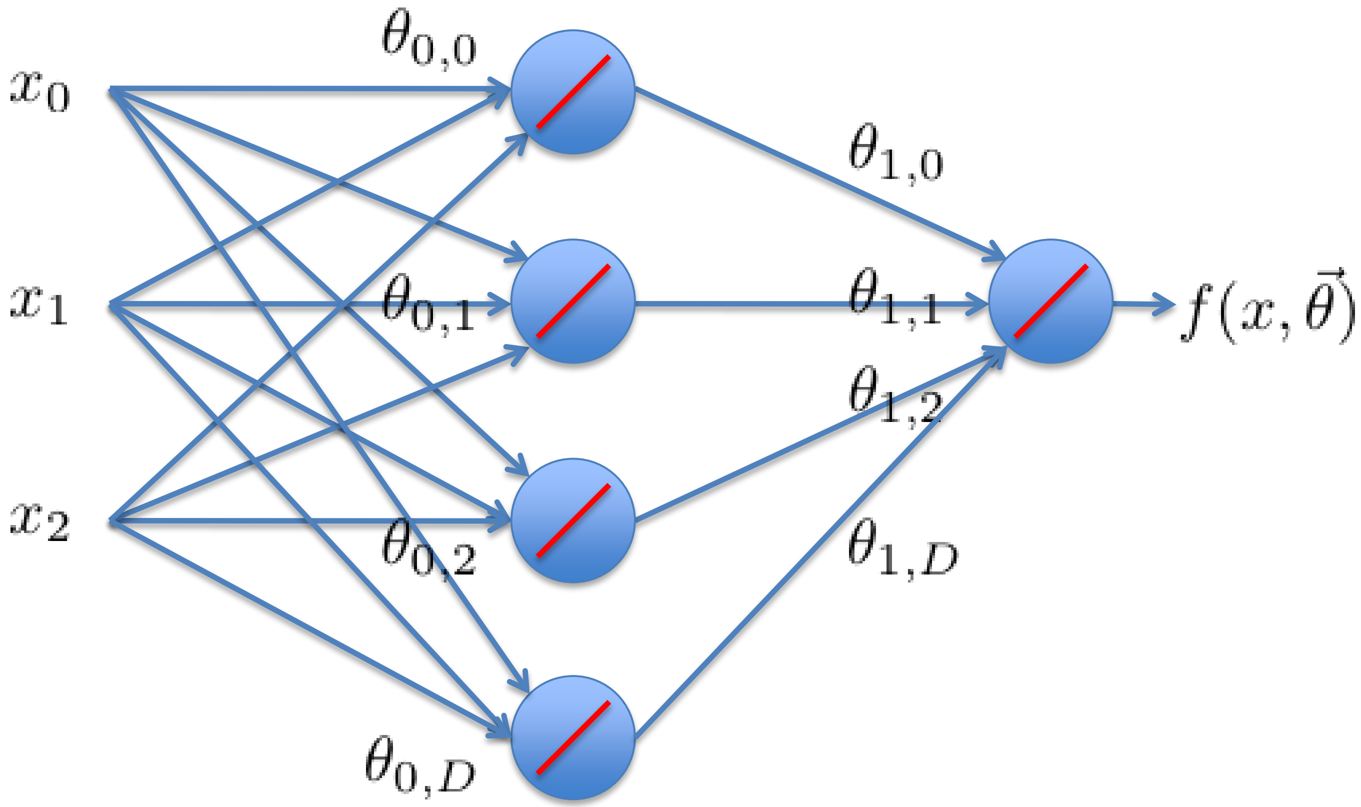
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = a$



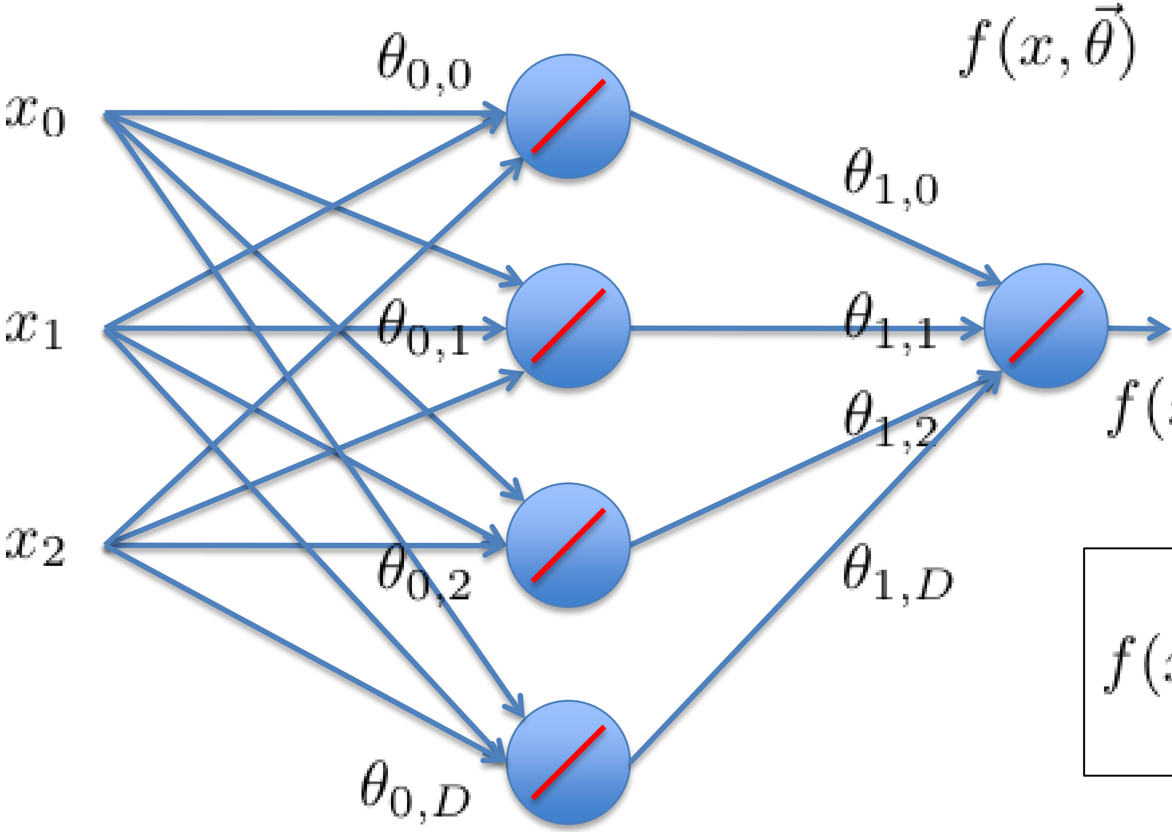
Linear Regression Neural Networks

- **Question:** What happens when we arrange **linear neurons** in a multilayer network?



Linear Regression Neural Networks

- Nothing special happens.
  - The product of two linear transformations is itself a linear transformation.



$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} \sum_{n=0}^{N-1} \theta_{0,i,n} x_n$$

$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} [\theta_{0,i}^T \vec{x}]$$

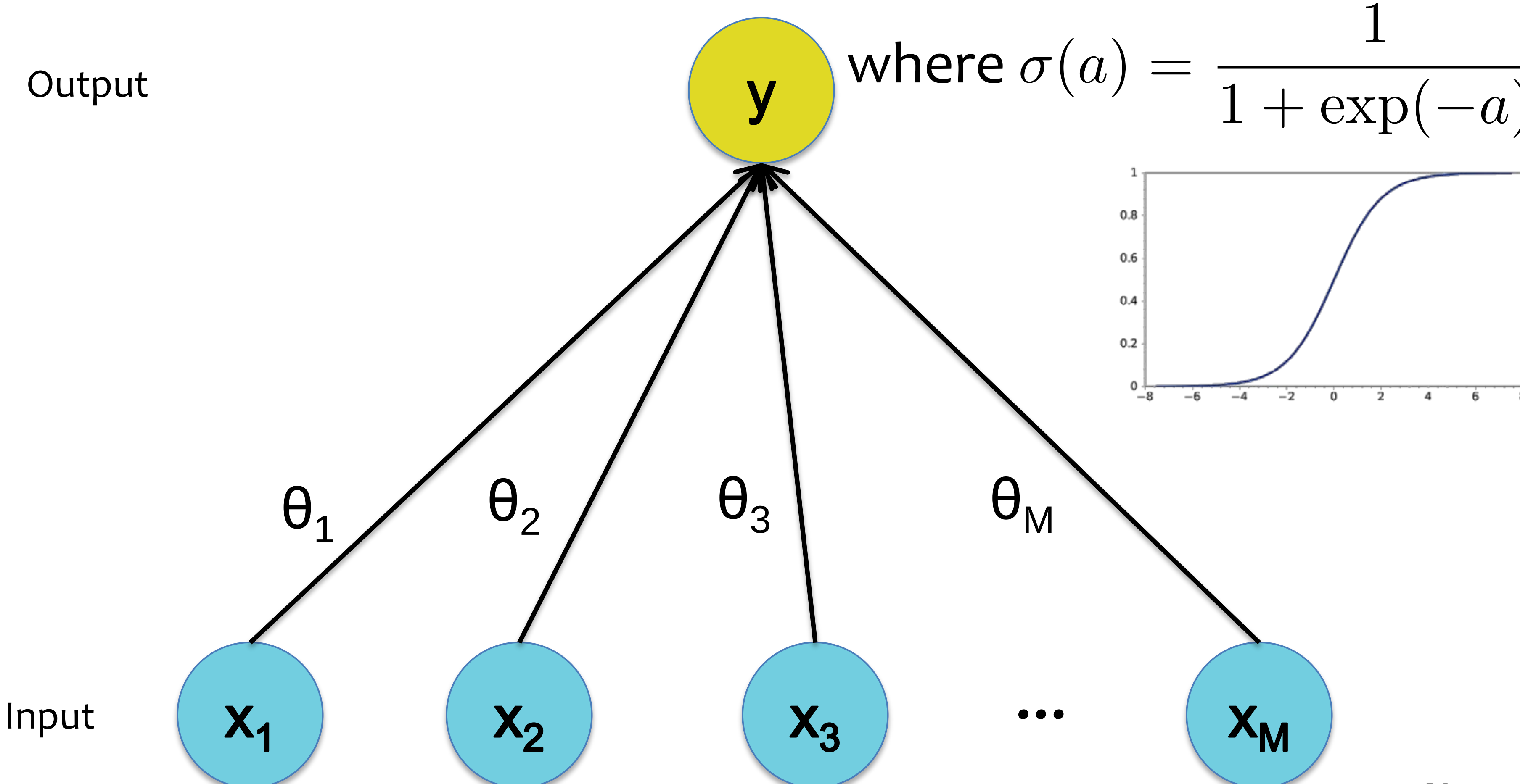
$$\boxed{f(x,\vec{\theta}) = \sum_{i=0}^{D} [\hat{\theta}_i^T \vec{x}]}$$

Decision Functions

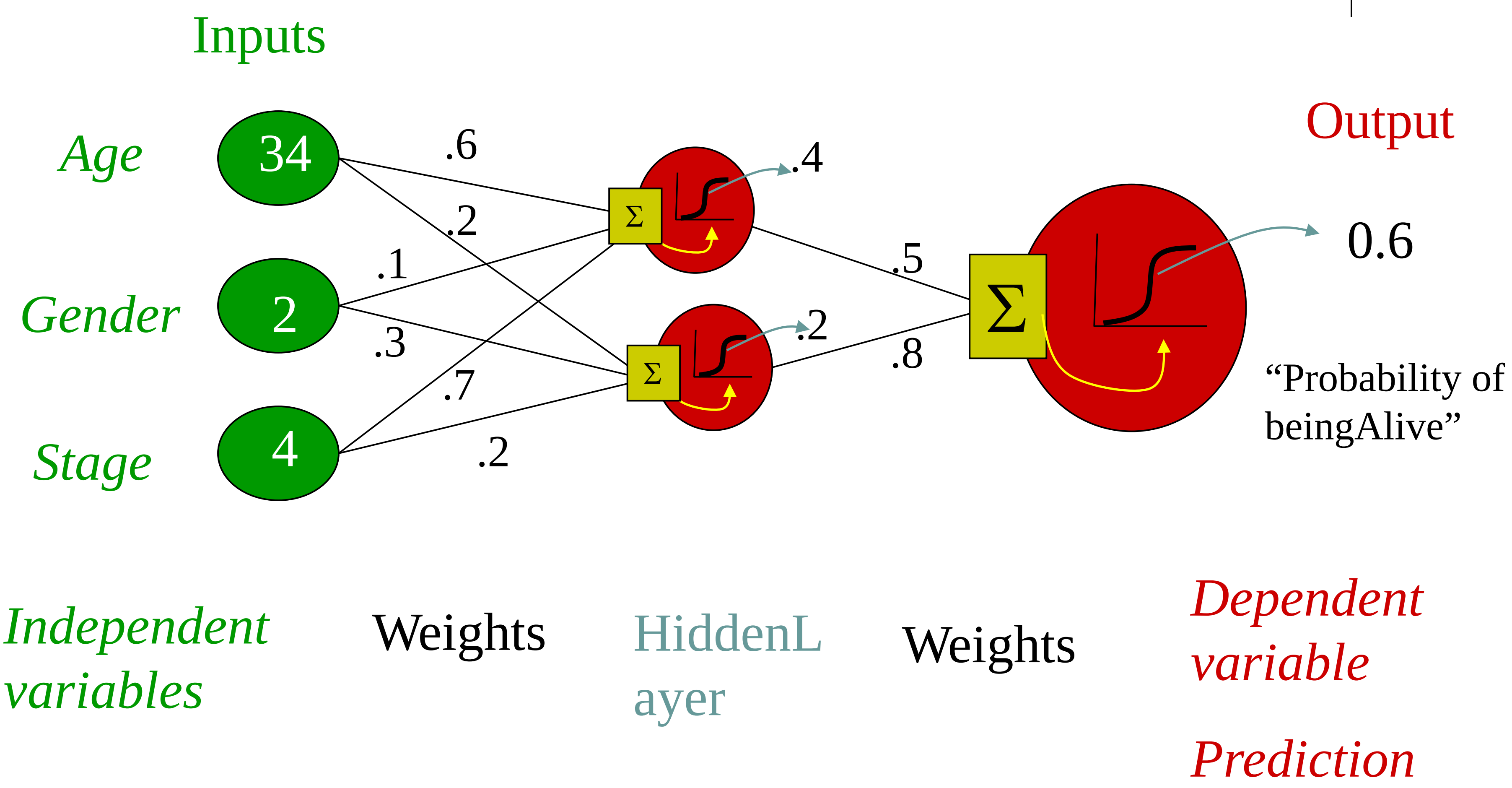
Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

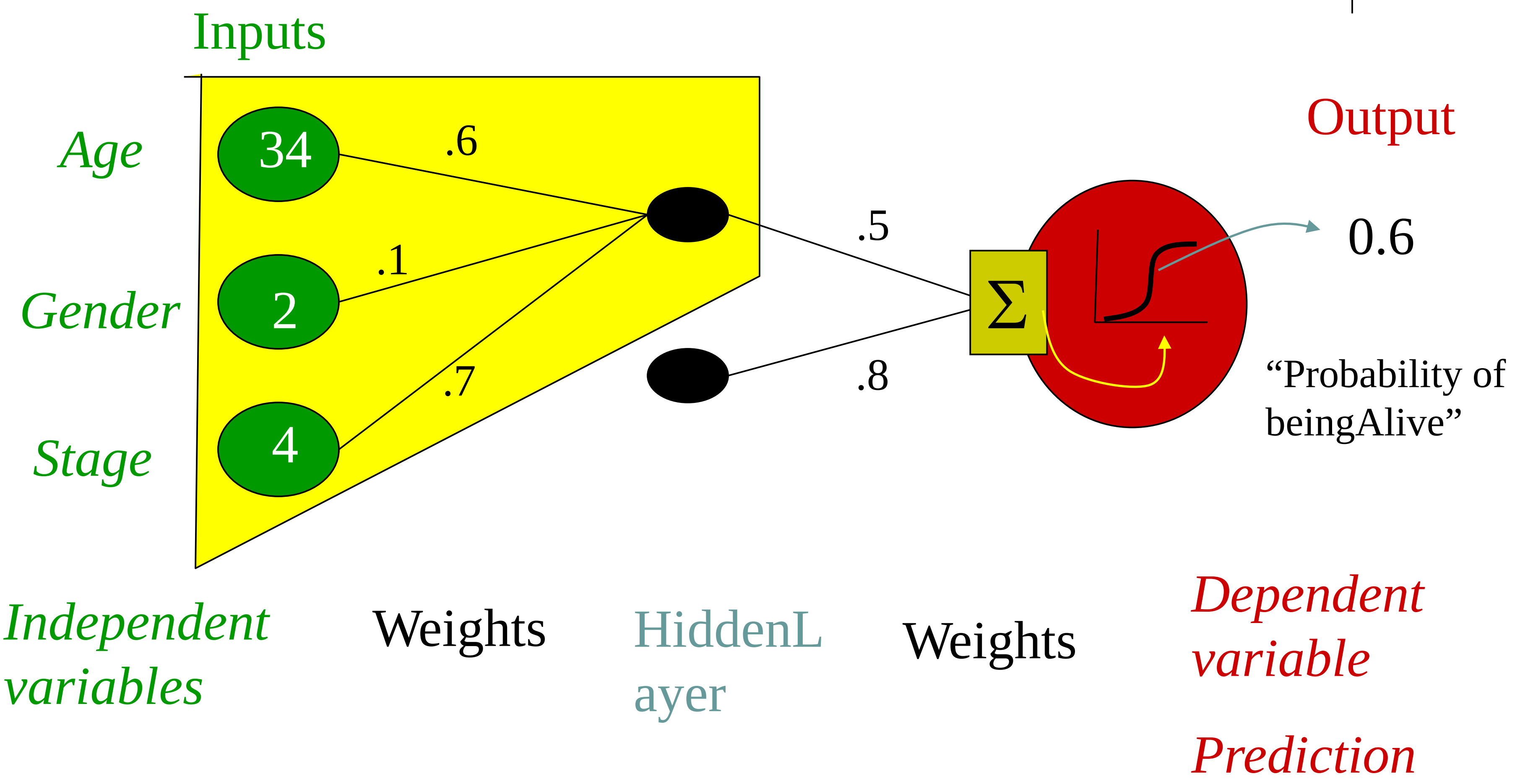
$$\text{where } \sigma(a) = \frac{1}{1 + \exp(-a)}$$

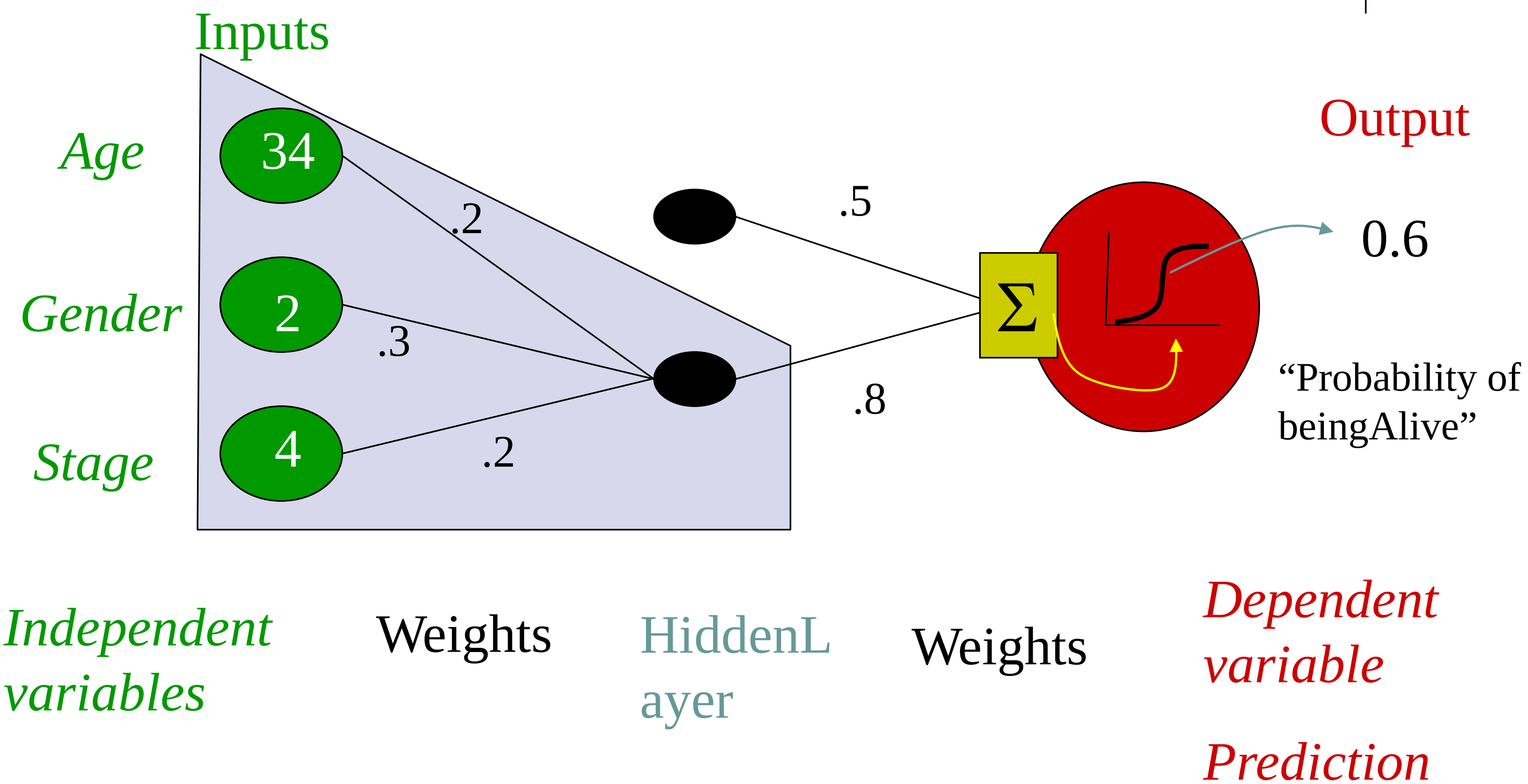


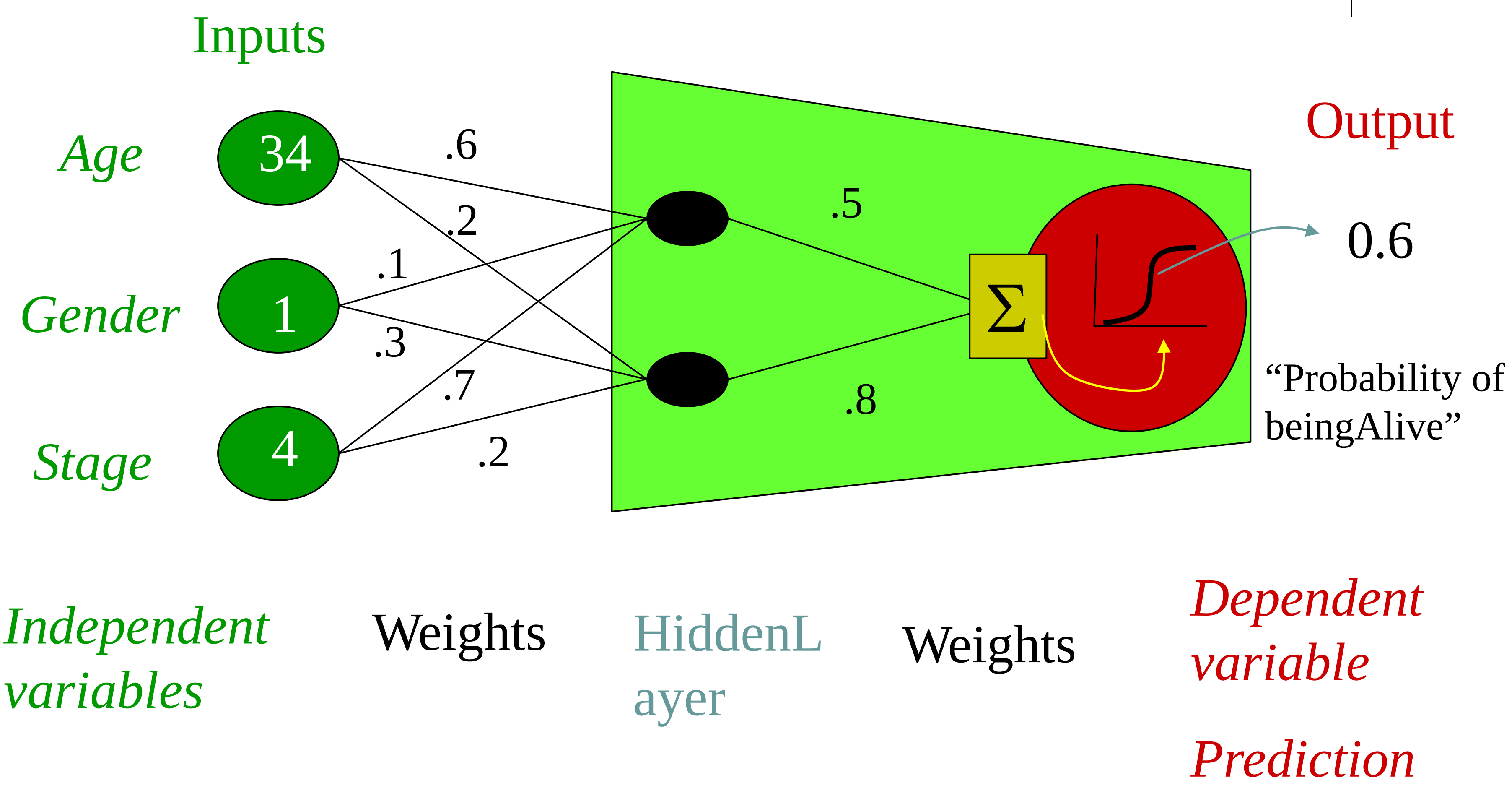
Neural Network Model



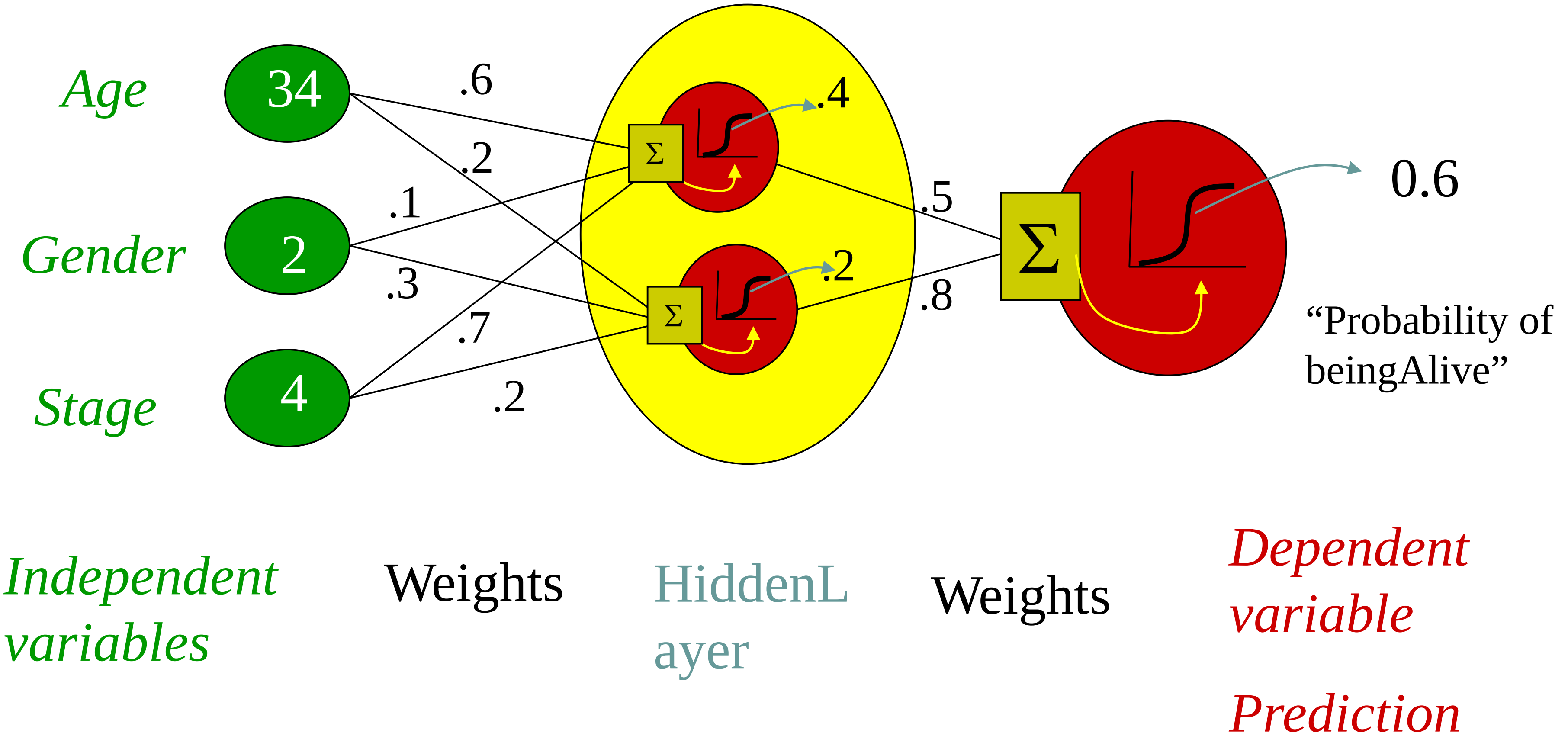
"Combined logistic models"




Inputs
Age
34
Gender
2
Stage
4
.2
.3
.2
.5
.8
Σ
Output
0.6
"Probability of beingAlive"
Independent variables
Weights
HiddenLayer
Weights
Dependent variable
Prediction


Inputs
Age
34
Gender
1
Stage
4
.6
.2
.1
.3
.7
.2
.5
.8
Σ
Output
0.6
"Probability of beingAlive"
Independent variables
Weights
HiddenLayer
Weights
Dependent variable
Prediction

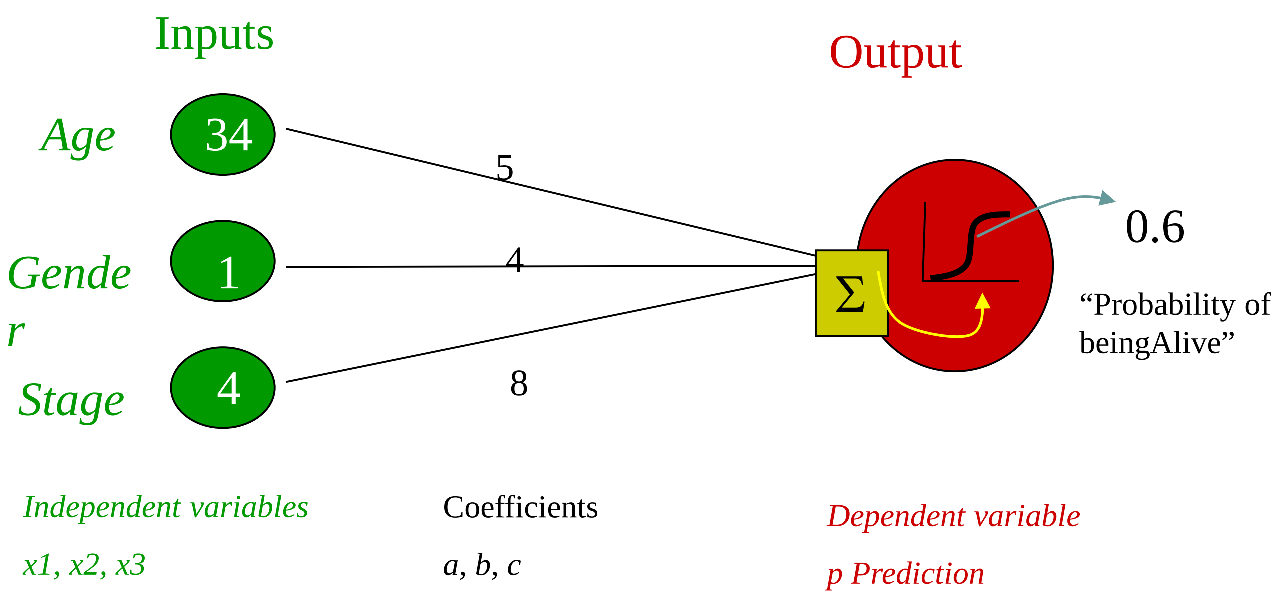
Not really, no target for hidden units...



Jargon Pseudo-Correspondence

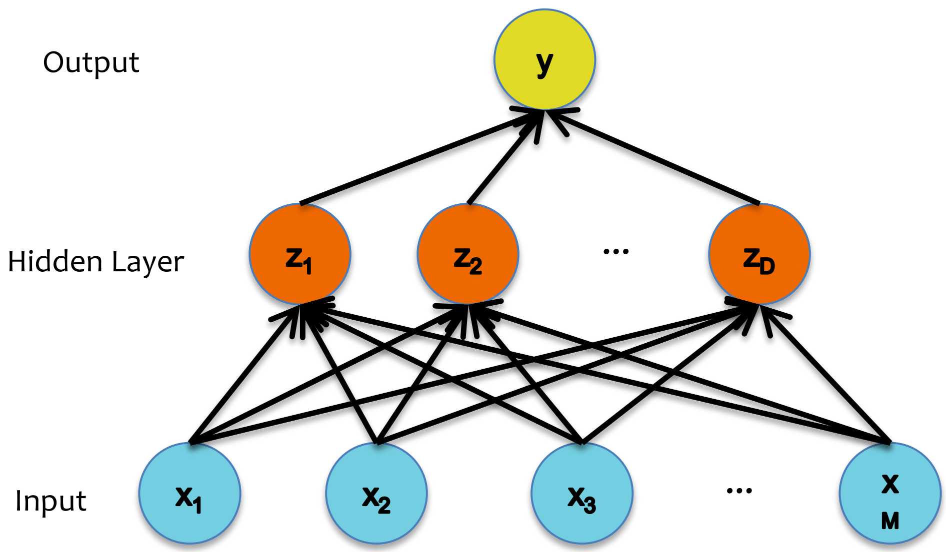
- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”
- Estimates = “targets”

Logistic Regression Model (the sigmoid unit)



Decision Functions

Neural Network



(E) **Output (sigmoid)**

$$y = \frac{1}{1+\exp(-b)}$$

(D) **Output (linear)**

$$b = \sum_{j=0}^{D} \beta_j z_j$$

(C) **Hidden (sigmoid)**

$$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$$

(B) **Hidden (linear)**

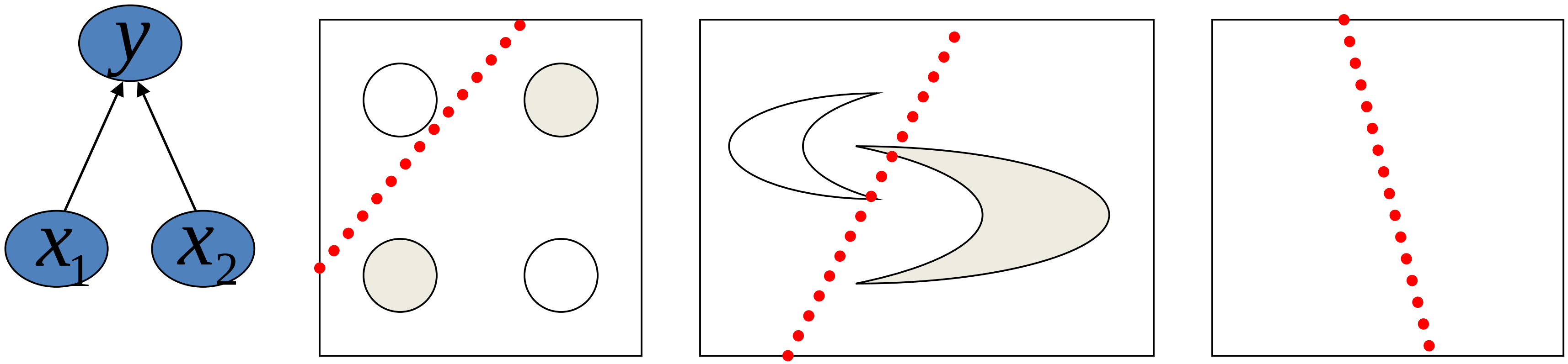
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$$

(A) **Input**

Given $x_i, \ \forall i$

Decision Boundary

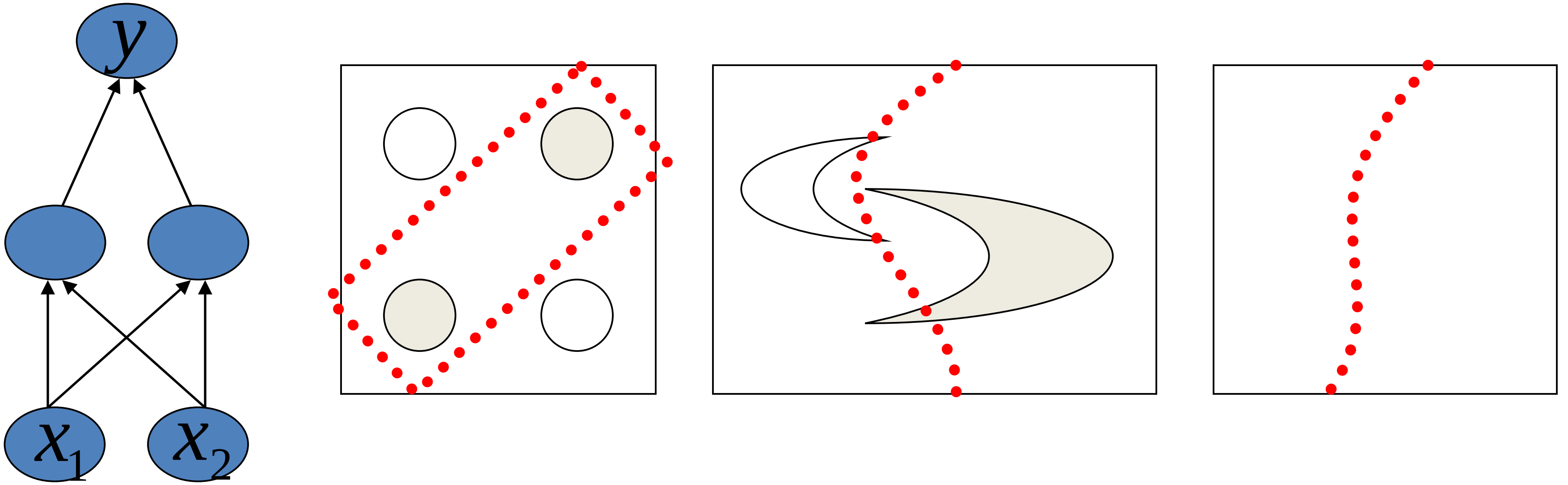
- 0 hidden layers: linear classifier
  - Hyperplanes



Example from to Eric Postma via Jason Eisner

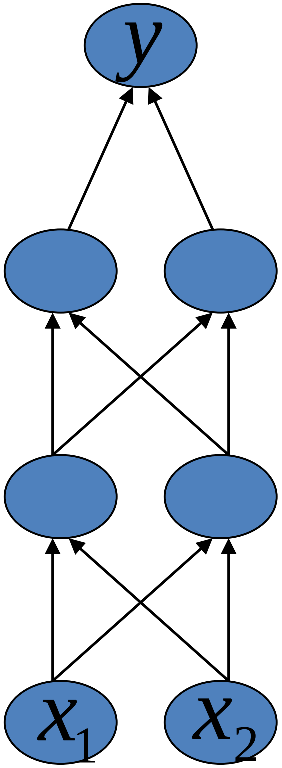
Decision Boundary

- 1 hidden layer
  - Boundary of convex region (open or closed)

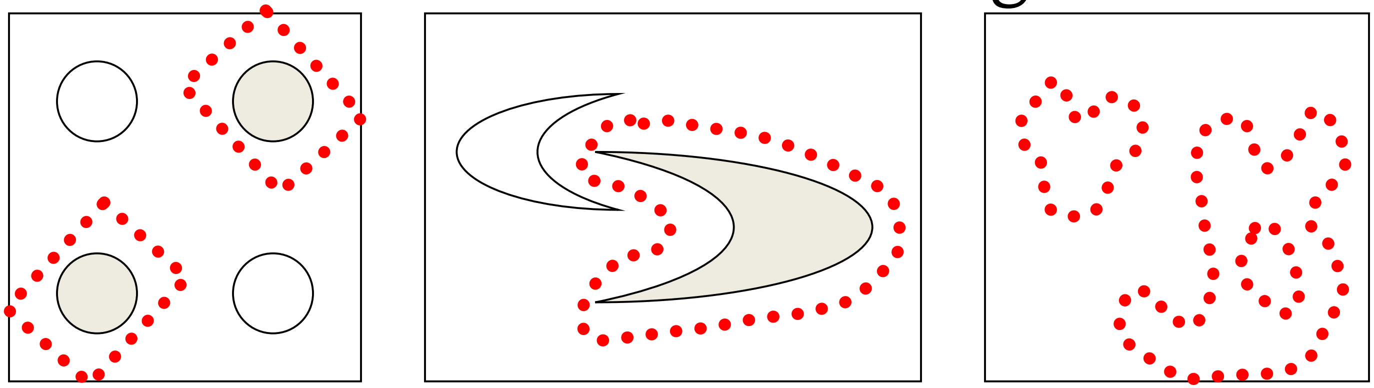


Example from to Eric Postma via Jason Eisner

Decision Boundary

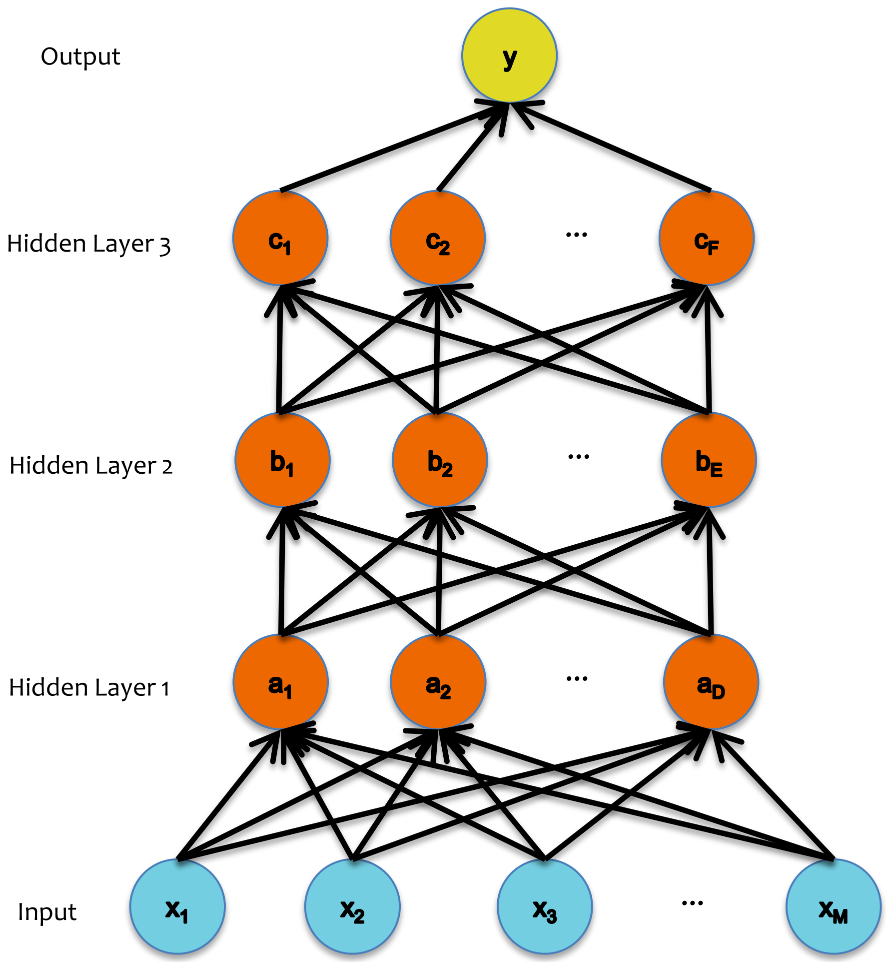


- 2 hidden layers
  - Combinations of convex regions



Example from to Eric Postma via Jason Eisner

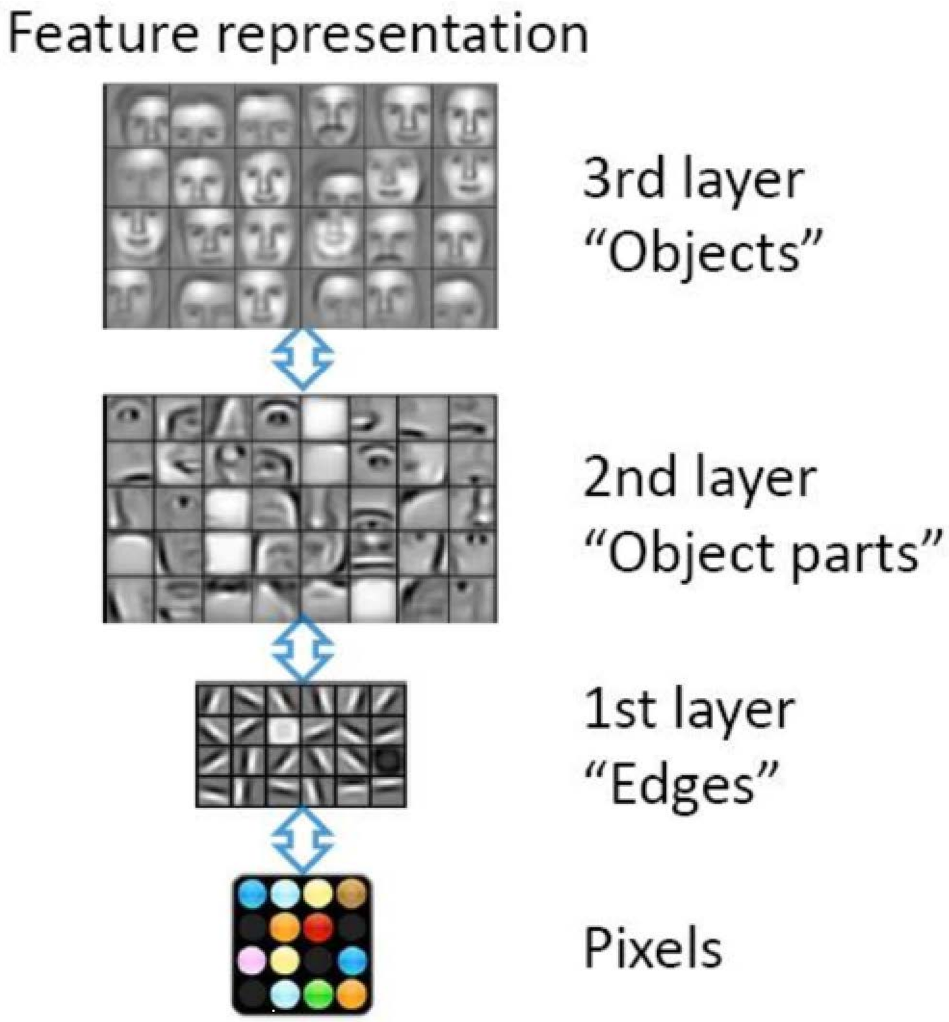
Deeper Networks



Decision Functions

Different Levels of Abstraction

- We don't know the "right" levels of abstraction
- So let the model figure it out!



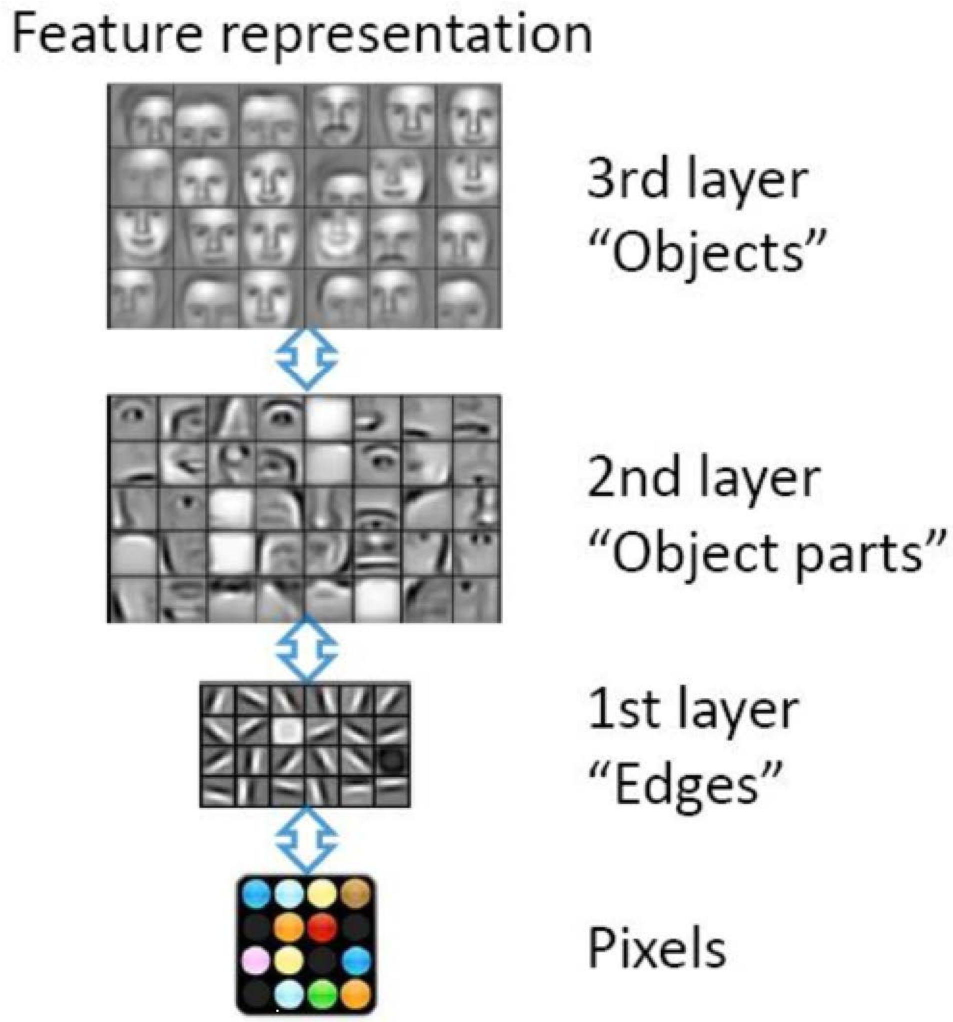
Example from Honglak Lee (NIPS 2010)

Decision Functions

Different Levels of Abstraction

Face Recognition:

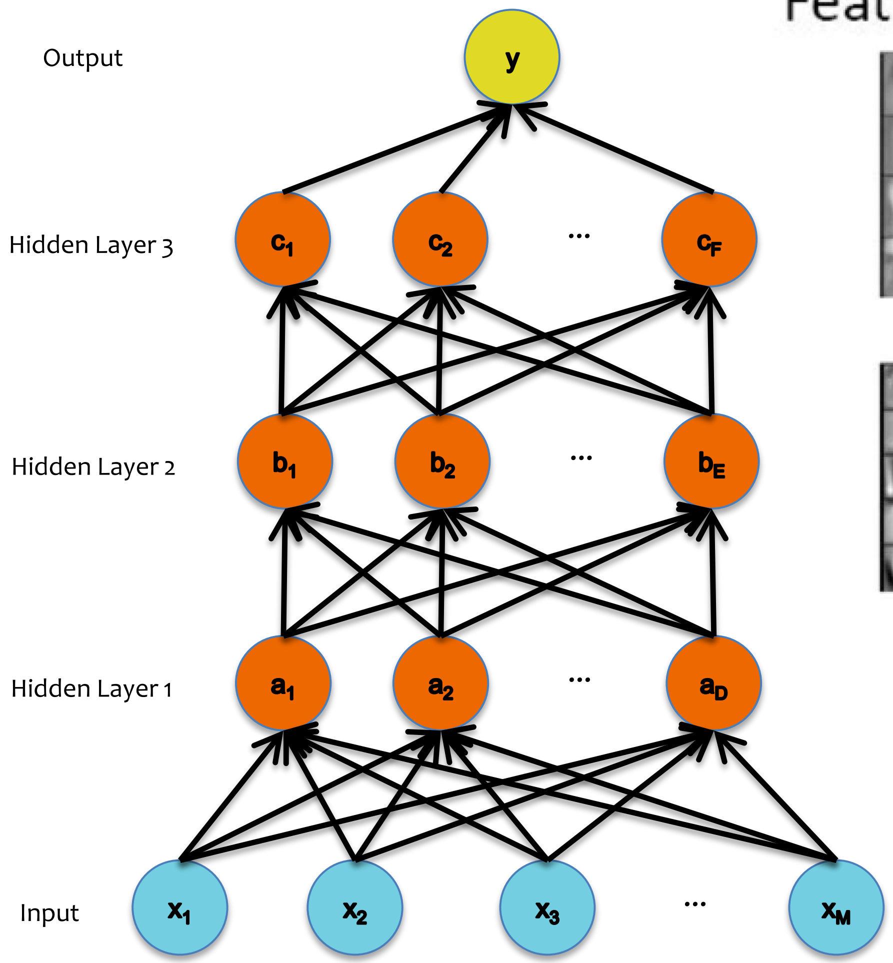
- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions

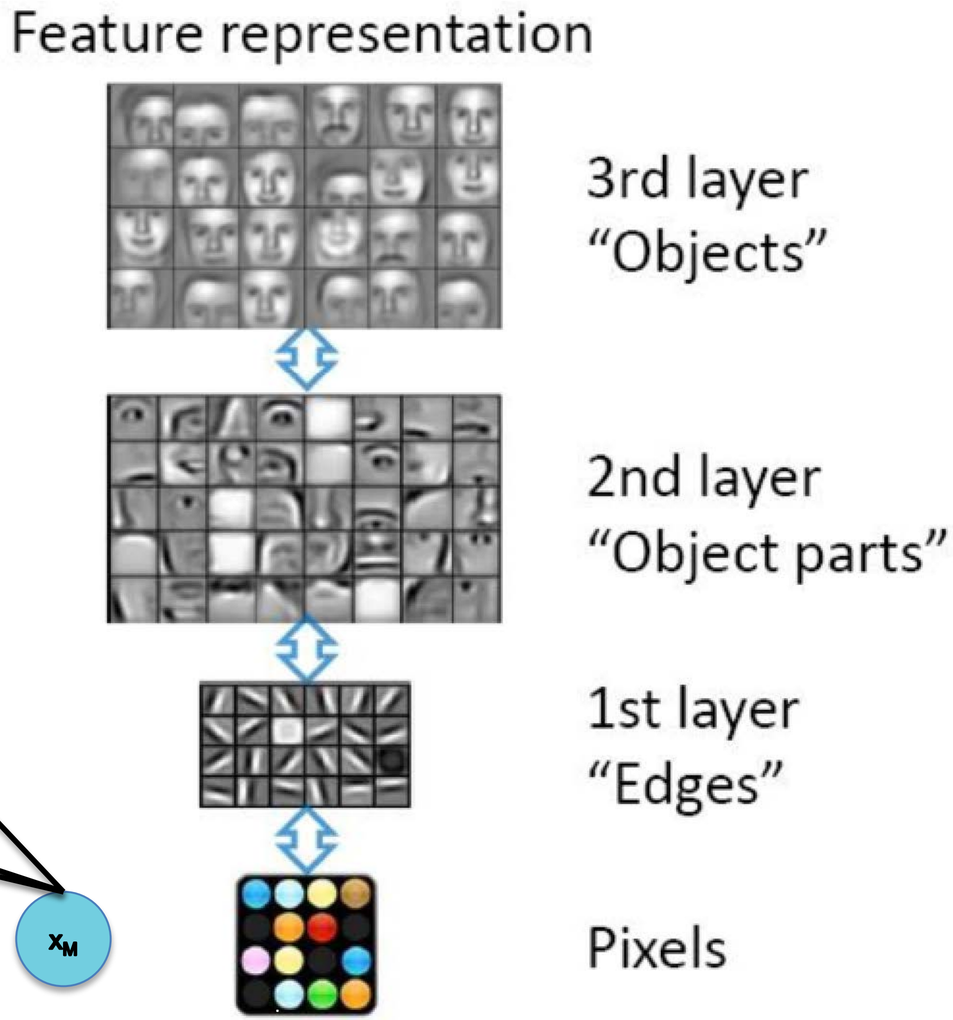


Example from Honglak Lee (NIPS 2010)

Decision Functions

Different Levels of Abstraction





Example from Honglak Lee (NIPS 2010)

Week 9 Contents / Objectives

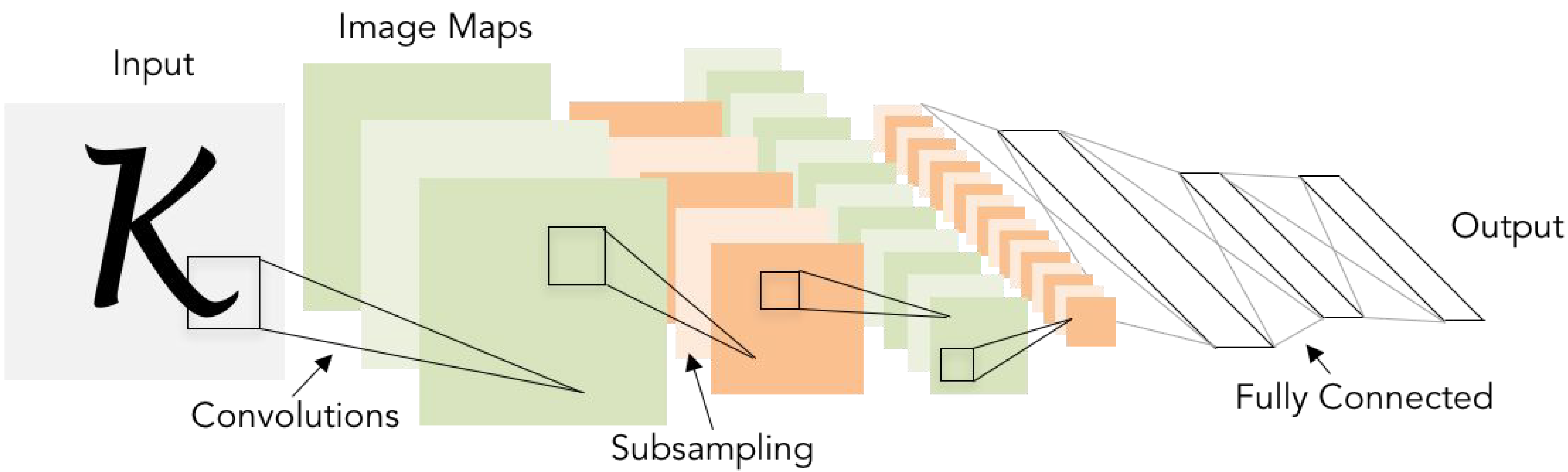
Part A

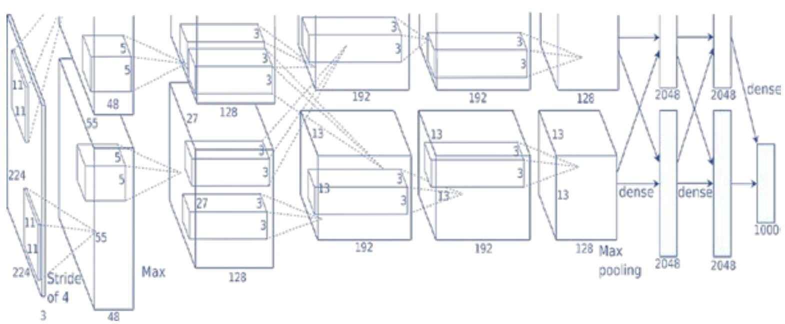
- Motivation for Logistic Regression
- Logistic Regression

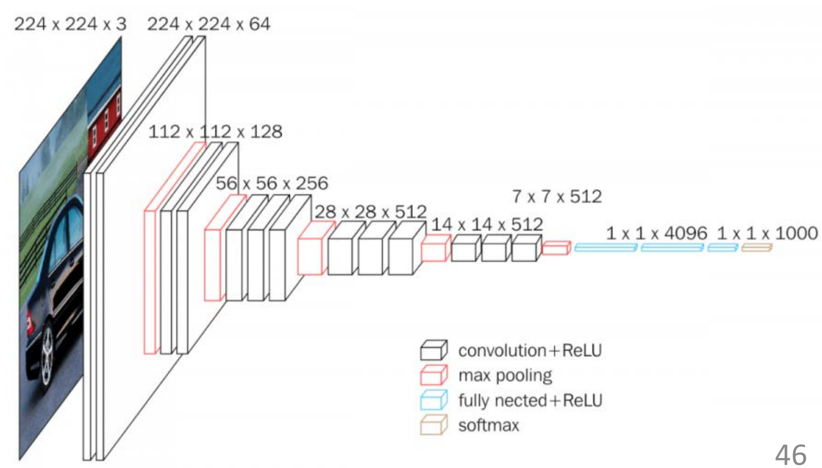
Part B

- Neural Networks
- **Convolutional Neural Networks Unboxing**

Convolutional Neural Networks

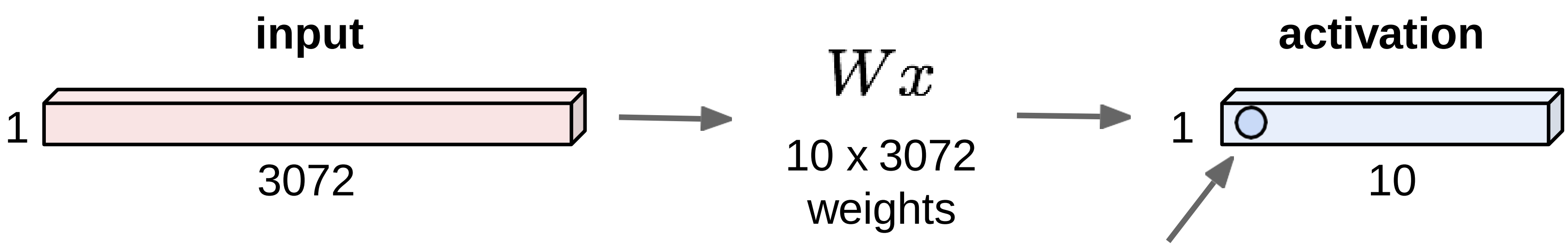






Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



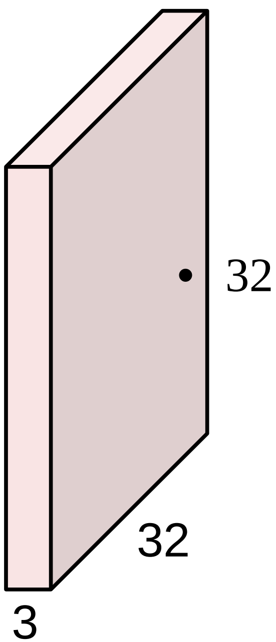
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Convolution Layer

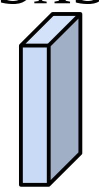
Tensor: Preserve spatial structure

Filters always extend the full depth of the input volume

- 32x32x3 image



- 5x5x3 filter

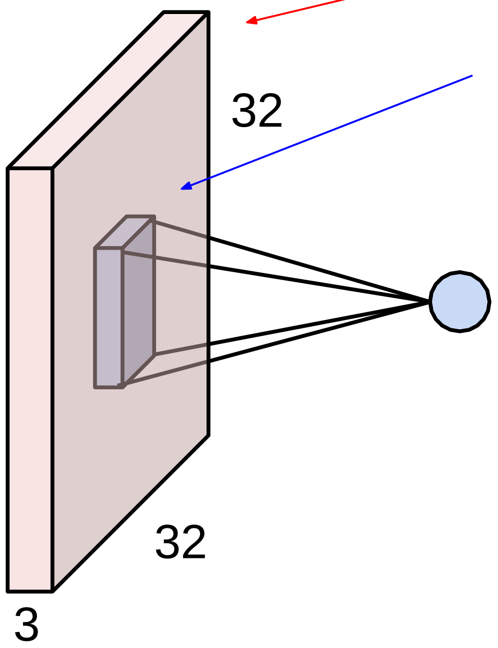


- **Convolve** the filter with the image
- i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

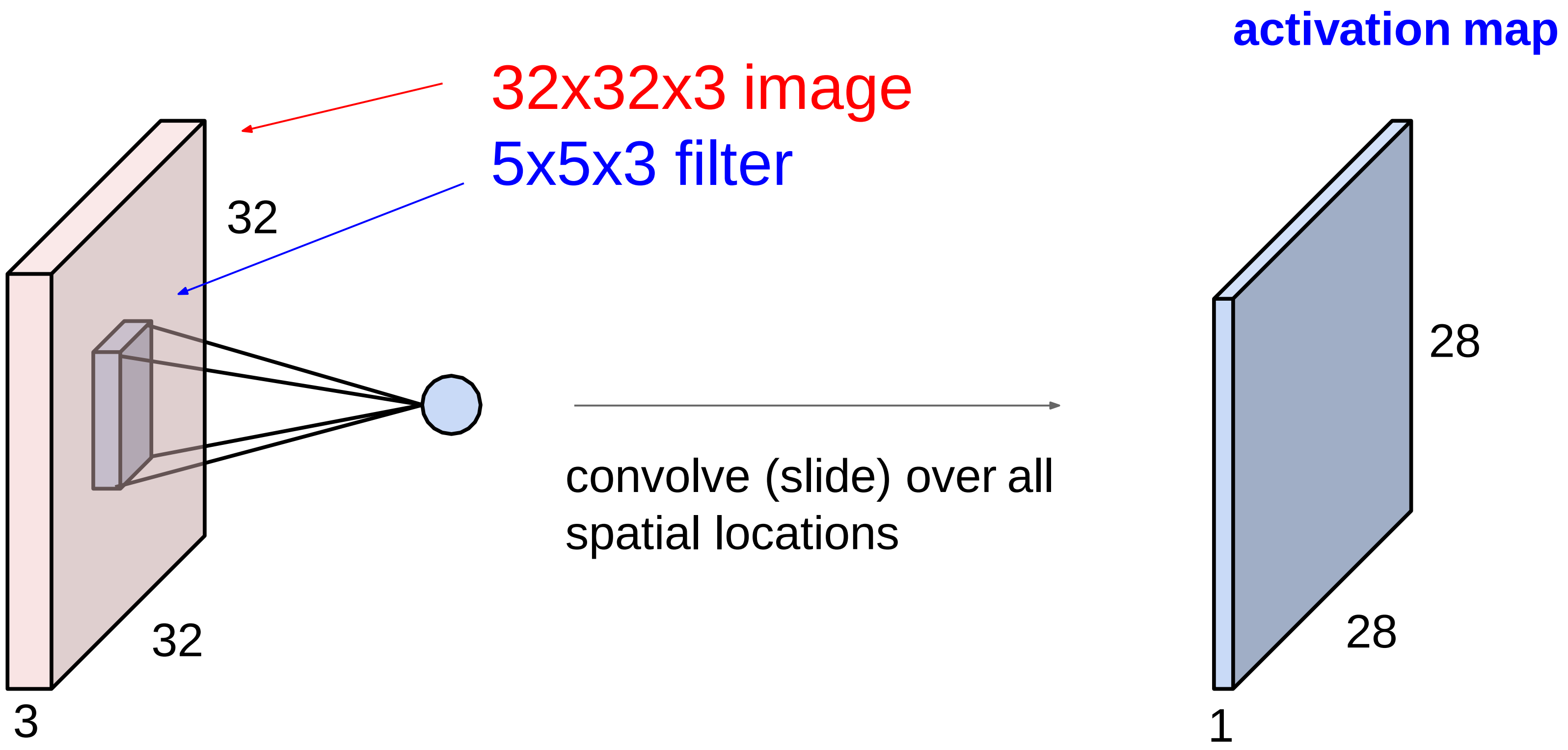
5x5x3 filter w



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

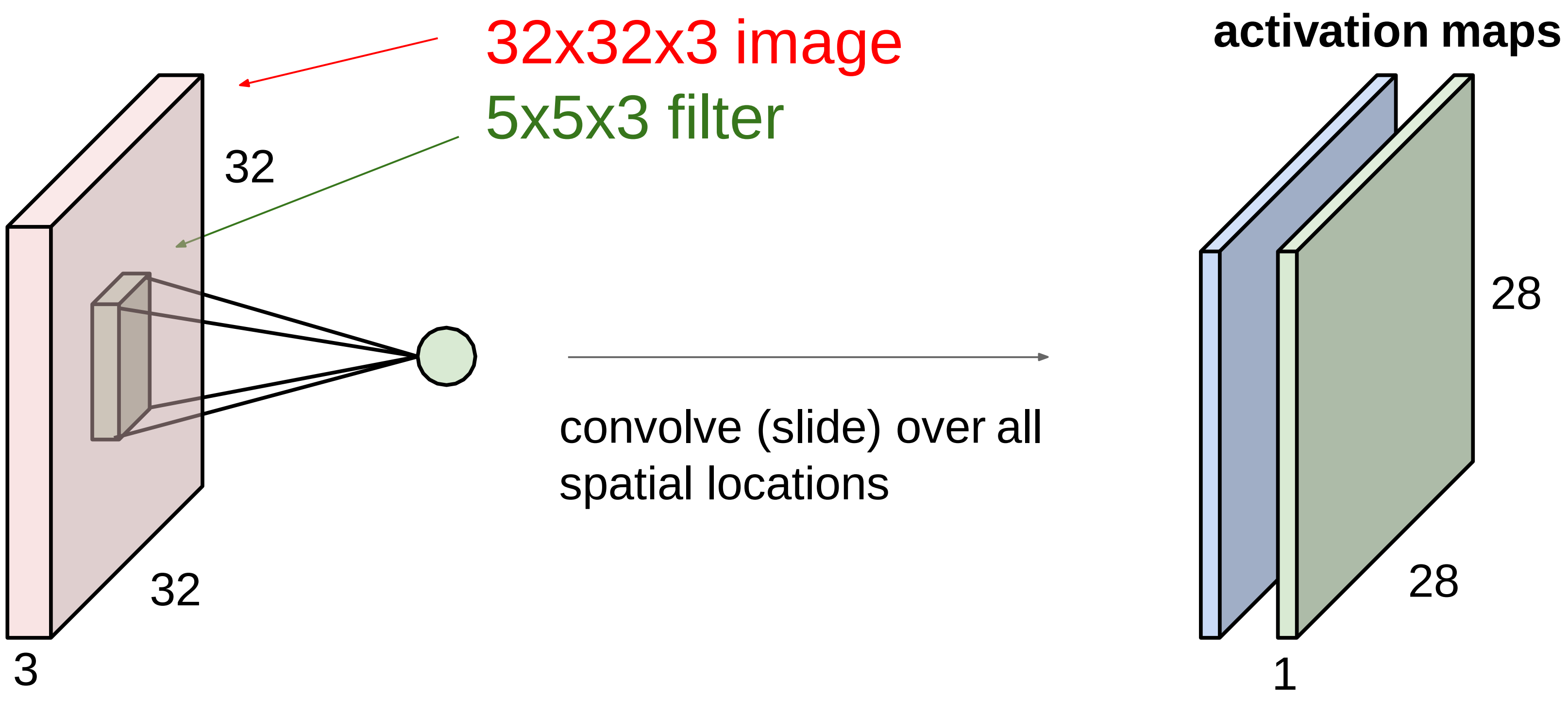
$$w^T x + b$$

Convolution Layer



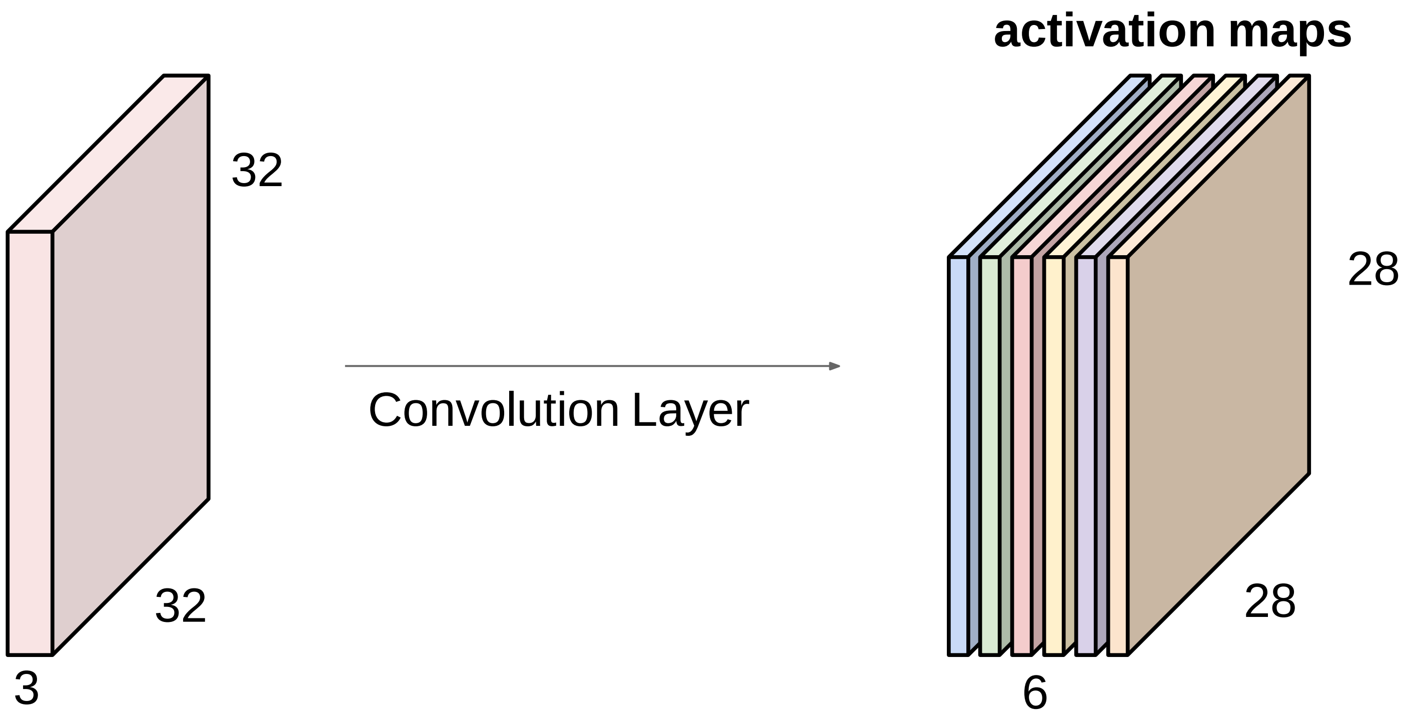
Convolution Layer

consider a second, green filter



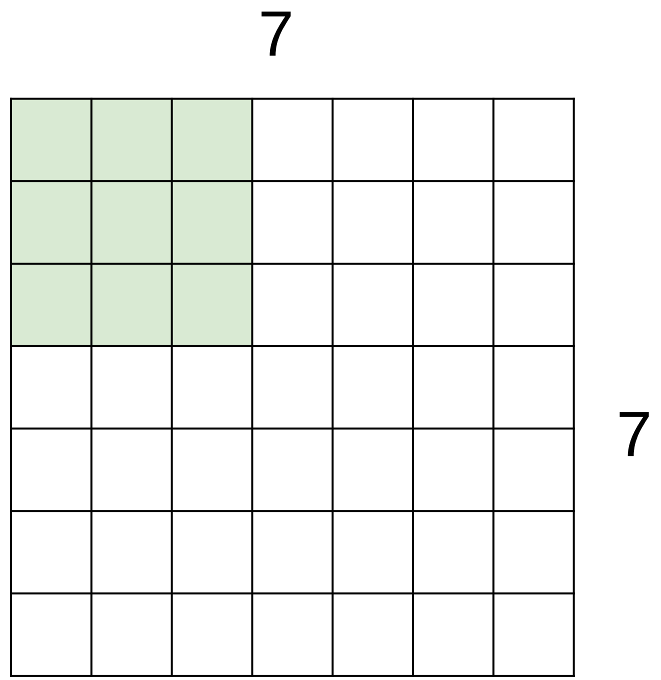
Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



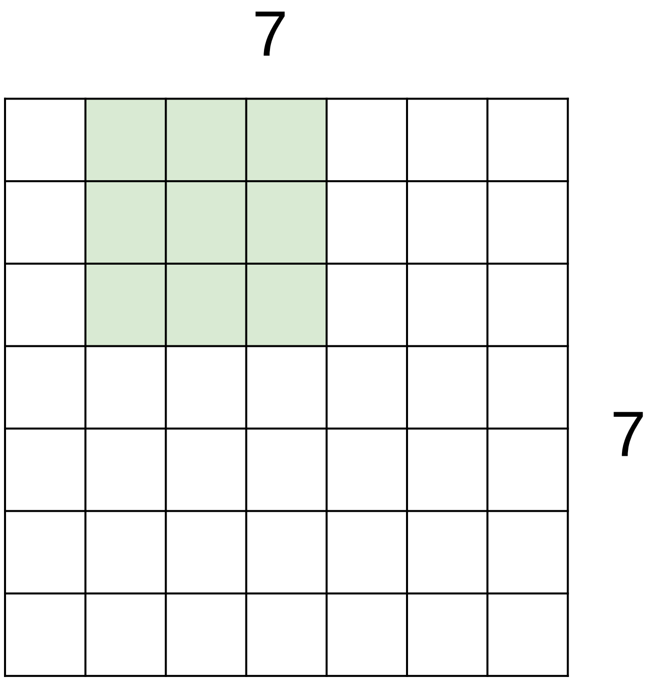
We stack these up to get a "new image" of size 28x28x6!

Convolution Operation



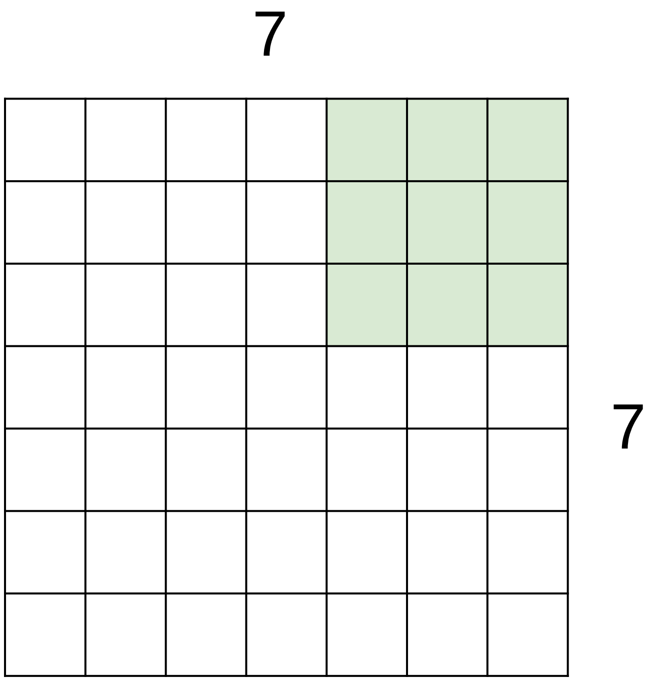
7x7 input (spatially)
assume 3x3 filter

Convolution Operation



7x7 input (spatially)
assume 3x3 filter

Convolution Operation

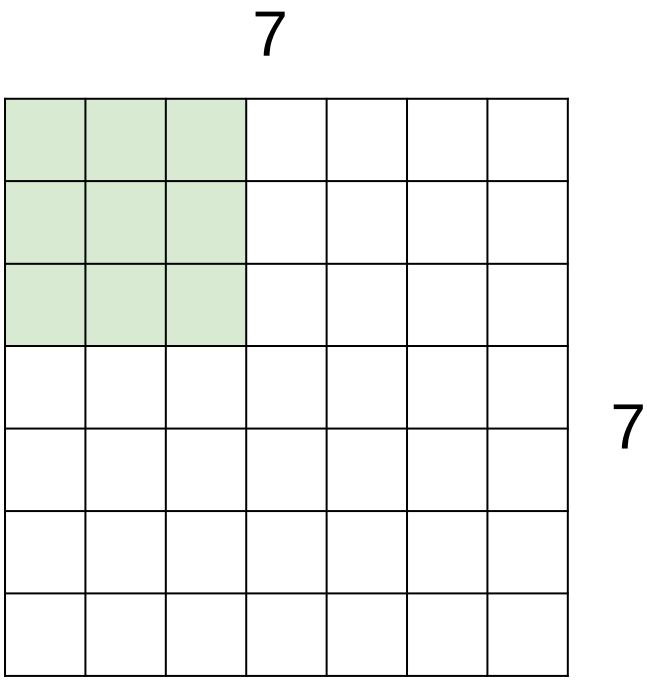


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

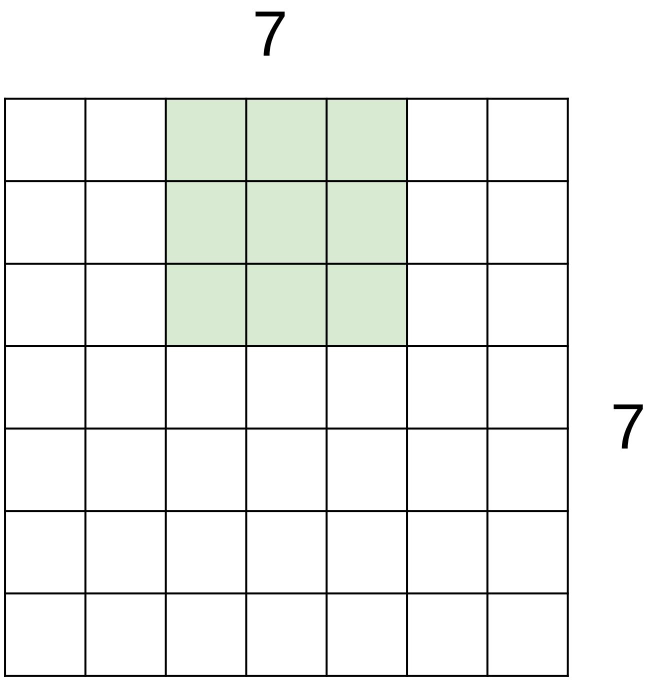
After three more sliding and dot products

Convolution with **Stride**



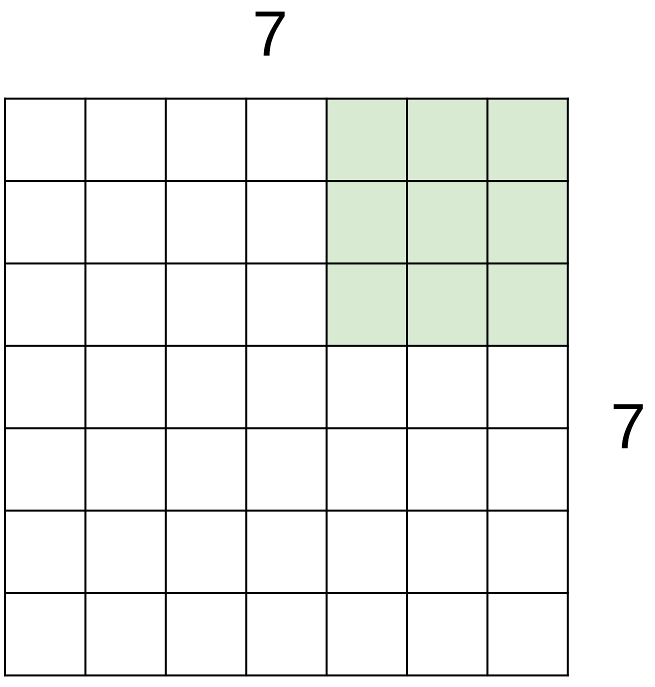
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution with **Stride**



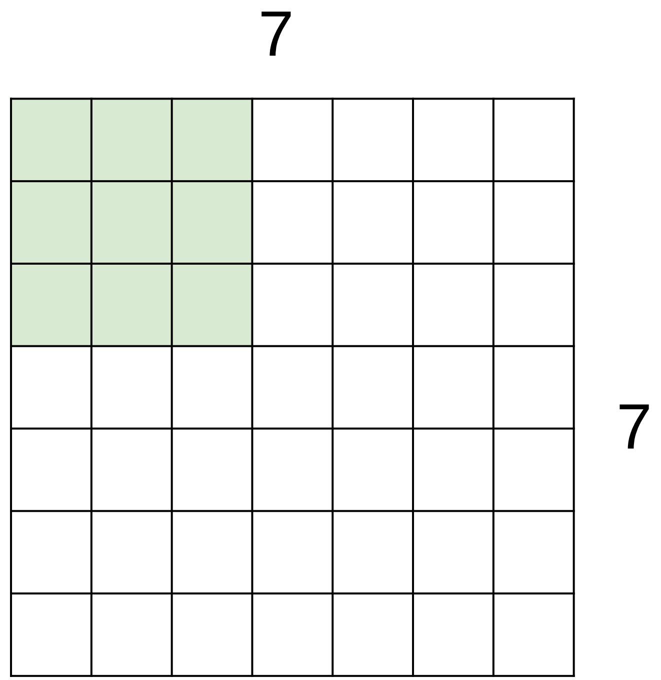
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution with **Stride**



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

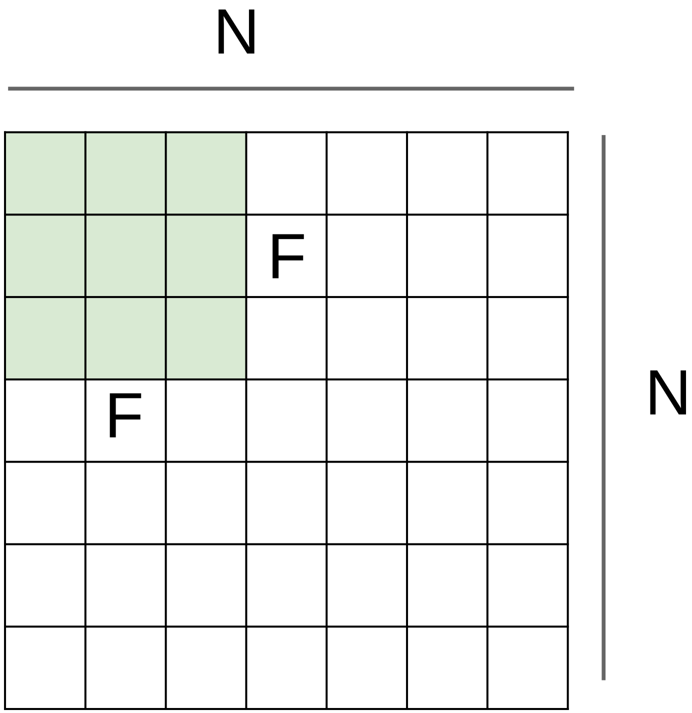
Convolution with **Stride**



Question: 7x7 input (spatially) assume 3x3 filter applied **with stride 3?**

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.

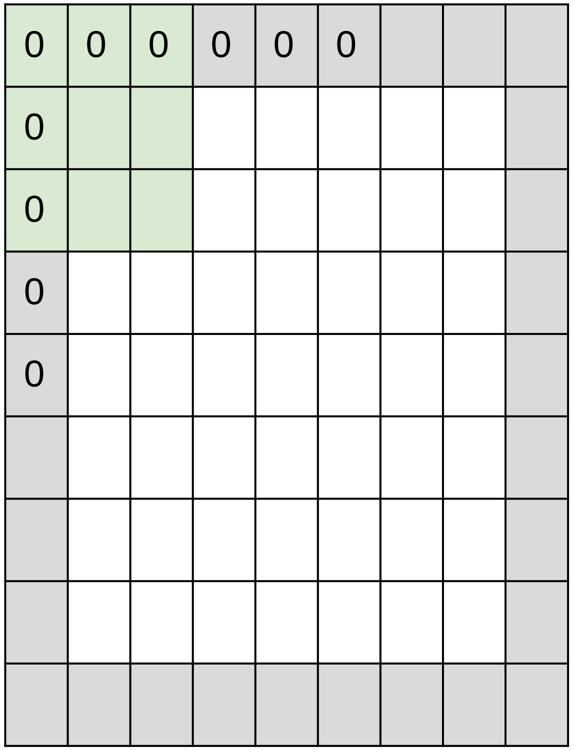
Convolution – Size of output



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Zero Padding

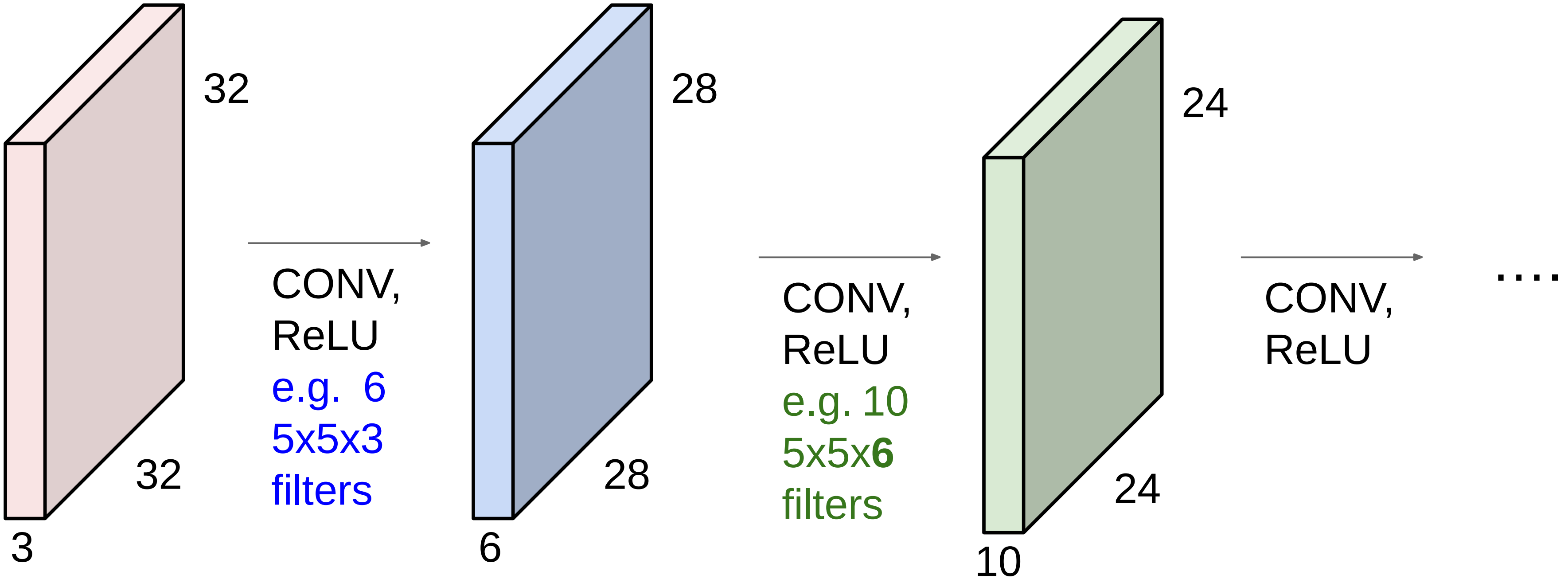


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Convolution Shrinks

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

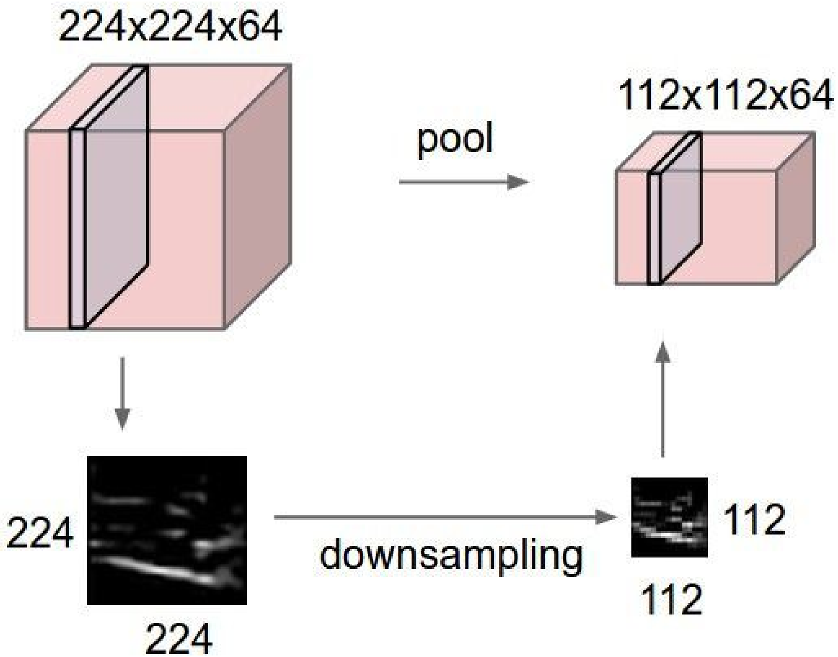


Exercises

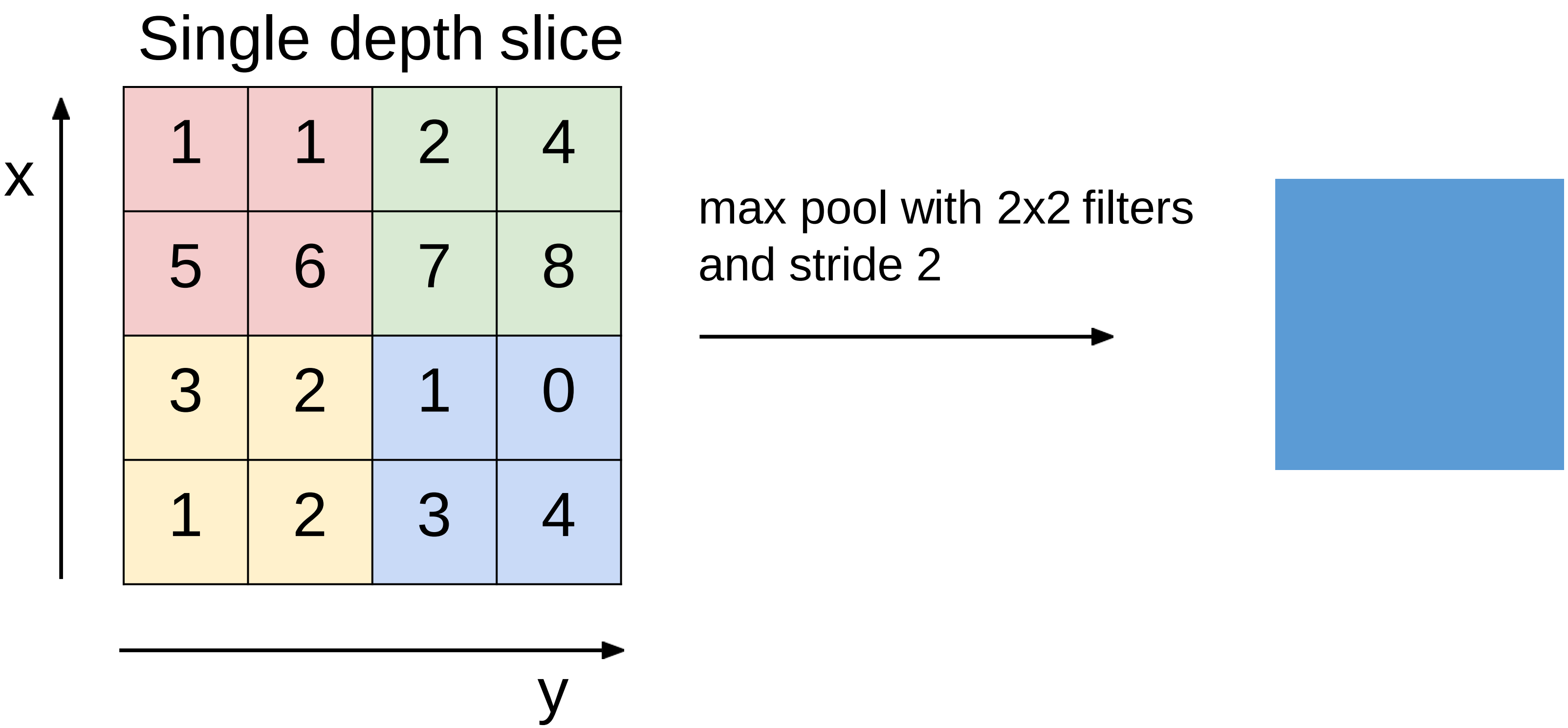
- Input volume: **32x32x3** 10 5x5 filters with stride 1, pad 2
  - Output volume size?
  - Number of parameters in this layer?

Pooling Layer: Downsampling

- makes the representations smaller and more manageable → dimensionality reduction
- operates over each activation map independently:



Max Pooling



Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2

Lab 9 CNN

```
__init__(self):
super(Net, self).__init__()
self.conv1 = nn.Conv2d(3, 6, 5) #3: #
self.pool = nn.MaxPool2d(2, 2)
self.conv2 = nn.Conv2d(6, 16, 5)
self.fc1 = nn.Linear(16 * 5 * 5, 120)
self.fc2 = nn.Linear(120, 84)
self.fc3 = nn.Linear(84, 10)

forward(self, x):
x = self.pool(F.relu(self.conv1(x)
x = self.pool(F.relu(self.conv2(x)
x = x.view(-1, 16 * 5 * 5)
x = F.relu(self.fc1(x))
x = F.relu(self.fc2(x))
x = self.fc3(x)
return x
```

- Initial Image Size: $3 \times 32 \times 32$
- After `conv1` : $6 \times 28 \times 28$ (32
- After Pooling: $6 \times 14 \times 14$ (ima
- After `conv2` : $16 \times 10 \times 10$ (14
- After Pooling: $16 \times 5 \times 5$ (halve
- After `fc1` : 120
- After `fc2` : 84
- After `fc3` : 10 (**= number of cl**

Acknowledgement

- Part A used materials from: Matt Gormley, Rachid Salmi, Jean-Claude Desenclos, Thomas Grein, Alain Moren, Christophe Giraud-Carrier, Bart Selman, Sham Kakade, Neil Lawrence
- Part B used materials from: Matt Gormley, Eric Xing, Nina Balcan, Fei-Fei Li & Justin Johnson & Serena Yeung

Recommended Reading

- The lab notebook and references there