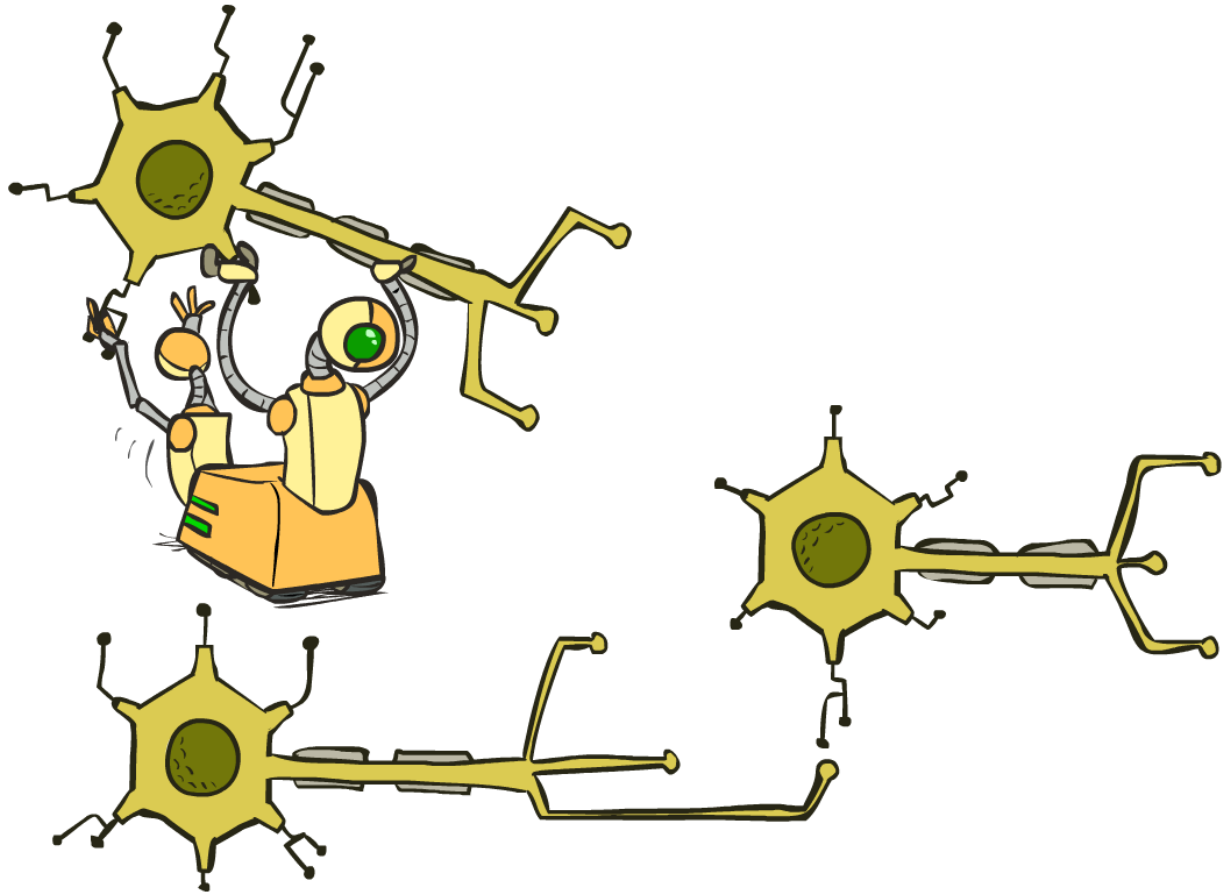
Announcements

- HW10
  - Due **tonight!**
- That is all.

CS 188: Artificial Intelligence

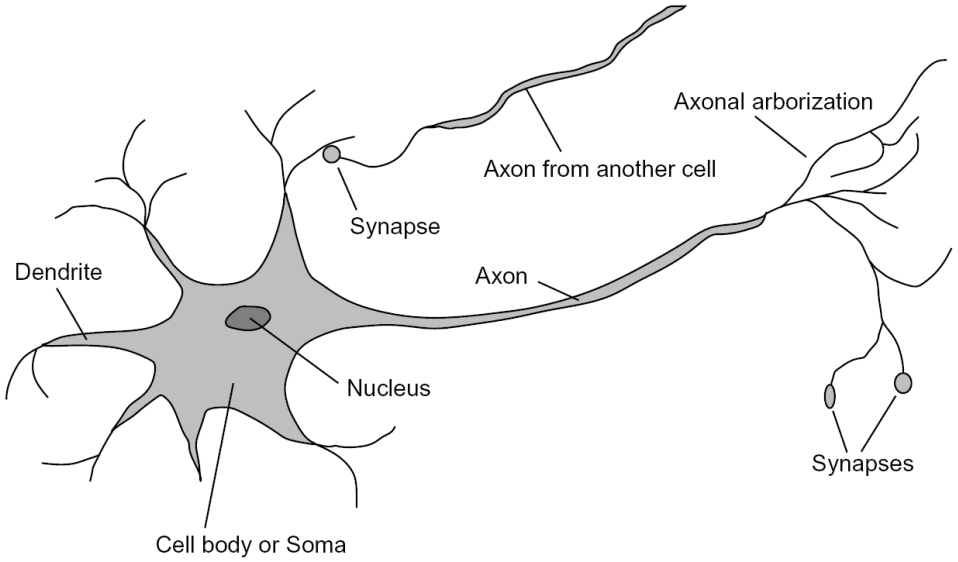
Optimization and Neural Nets

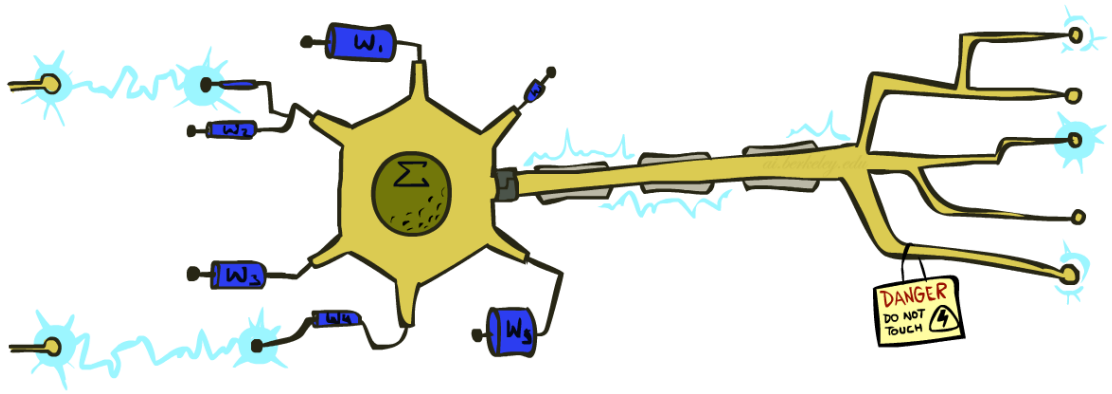


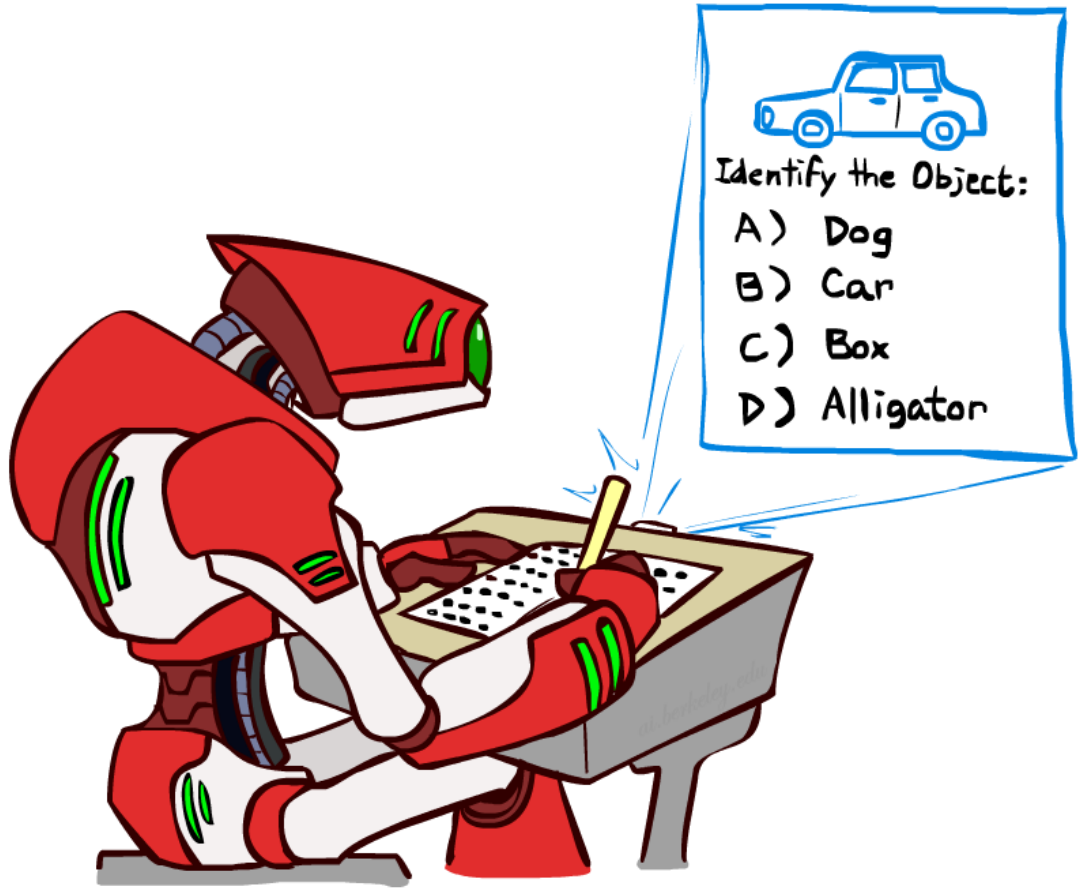
Instructors: Sergey Levine and Stuart Russell --- University of California, Berkeley

[These slides were created by Dan Klein, Pieter Abbeel, Sergey Levine. All CS188 materials are at http://ai.berkeley.edu.]

Last Time

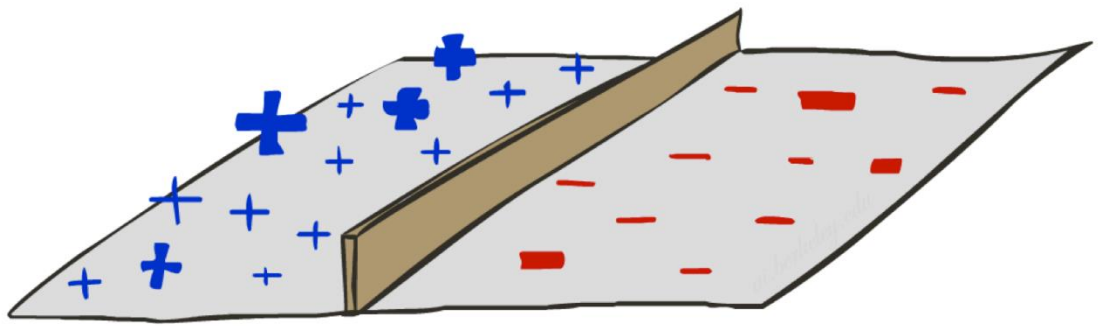

Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

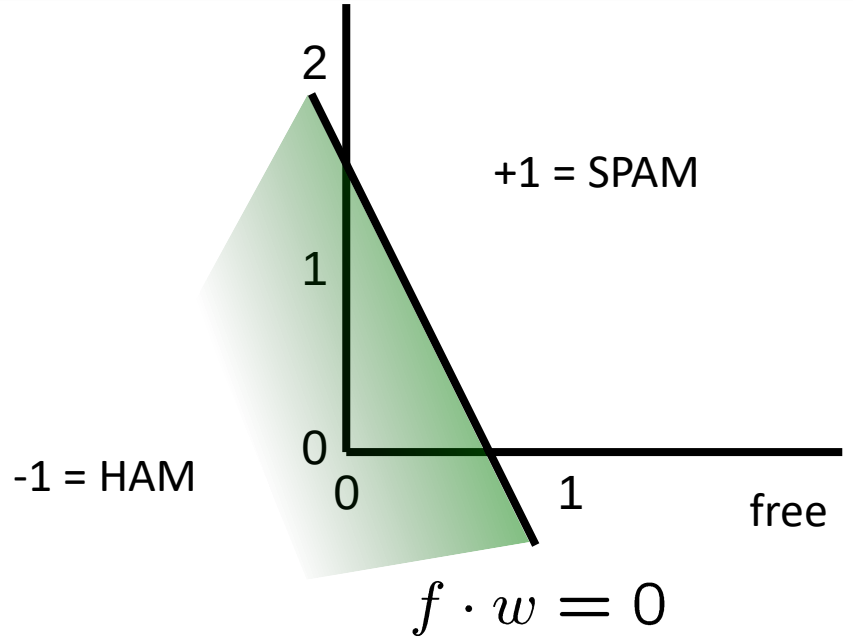

DANGER
DO NOT TOUCH


Identify the Object:
A) Dog
B) Car
C) Box
D) Alligator

Last Time

- Linear classifier
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1
- Perceptron
  - Algorithm for learning decision boundary for **linearly separable** data





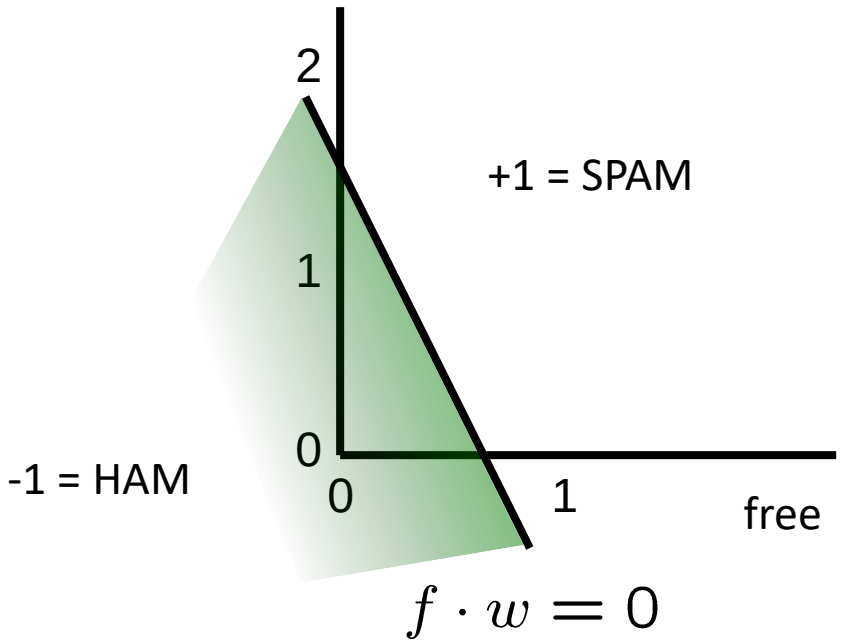
Quick Aside: Bias Terms

$\mathrm{f}(\mathrm{x})$

```
BIAS   : 1
free   : 0
money  : 1
...
```

w

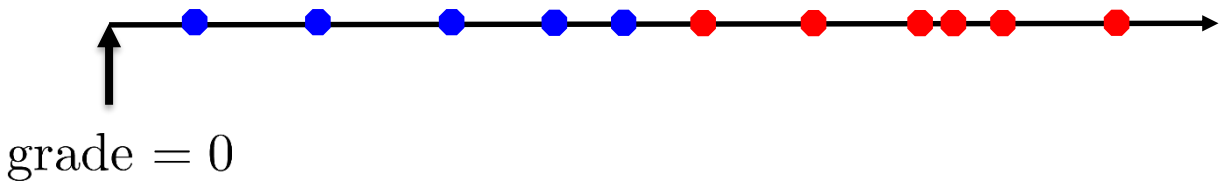
```
BIAS   : -3.6
free   :  4.2
money  :  2.1
...
```



- Why???

Quick Aside: Bias Terms

Imagine 1D features, without bias term:



f(x)	w
grade: 1	grade: 3.7

for which $f(x)$ is $f(x) \cdot w > 0$?

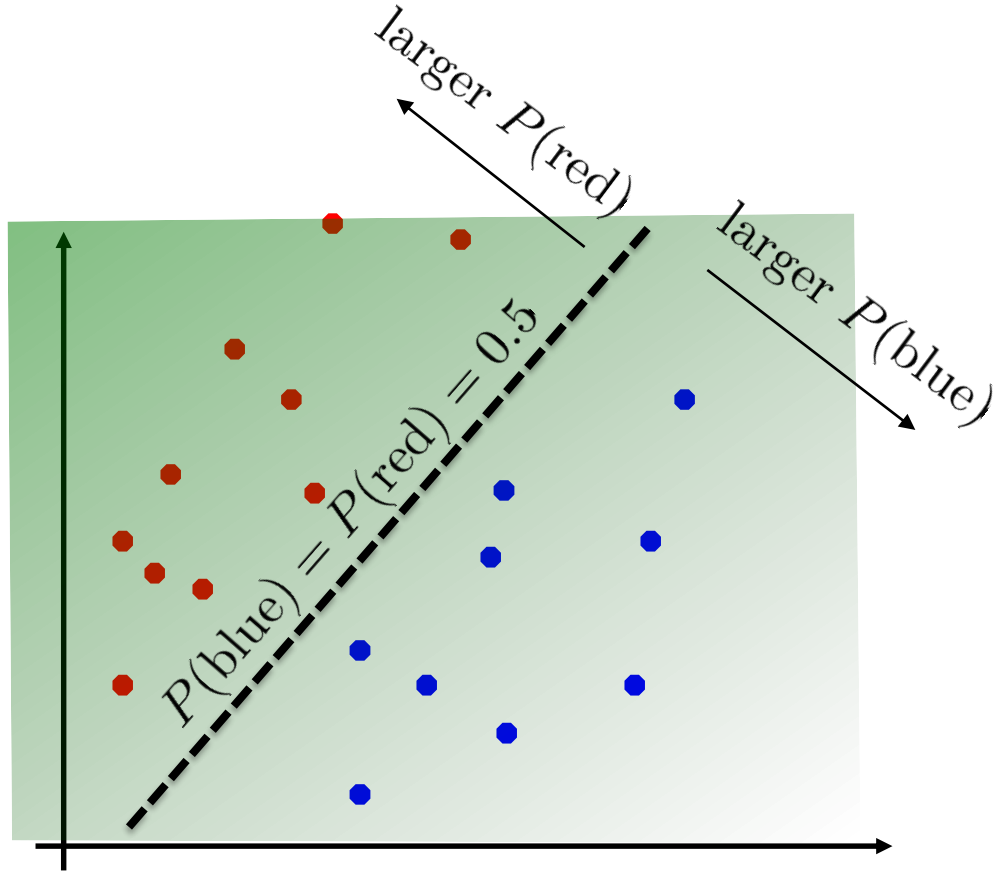
With bias term:

f(x)	w
BIAS : 1 grade : 1	BIAS : -1.5 grade : 1.0

for which $f(x)$ is $f(x) \cdot w > 0$?

$$f(x) \cdot w = w_0 + \text{grade} \times w_1$$

A *Probabilistic* Perceptron



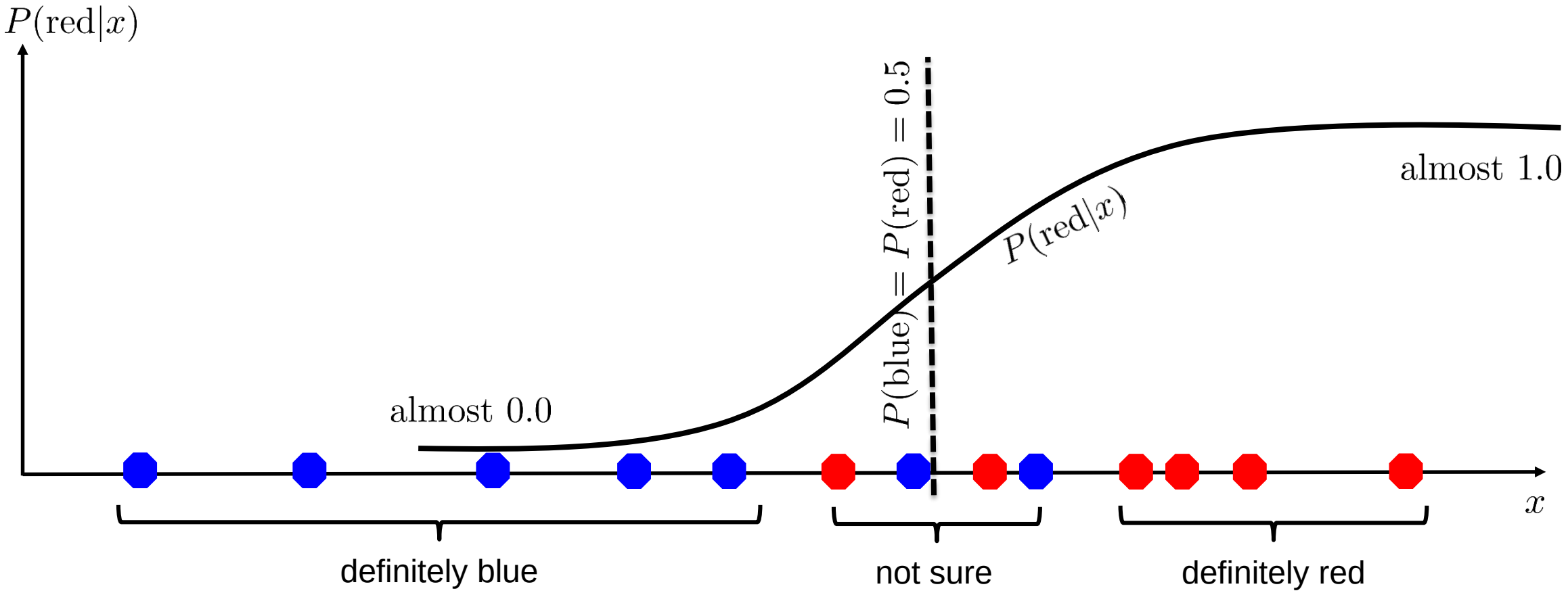
As $w_y \cdot x$ gets bigger, $P(y|x)$ gets bigger

A 1D Example

$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$

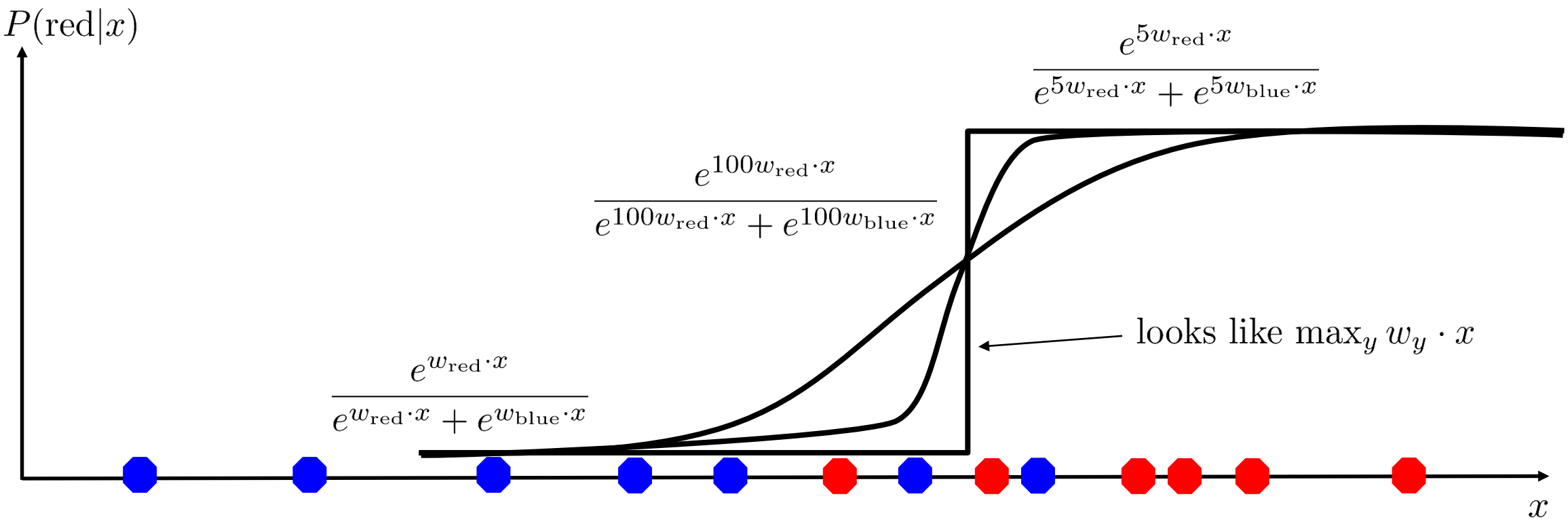
probability increases exponentially as we move away from boundary

normalizer



The *Soft* Max

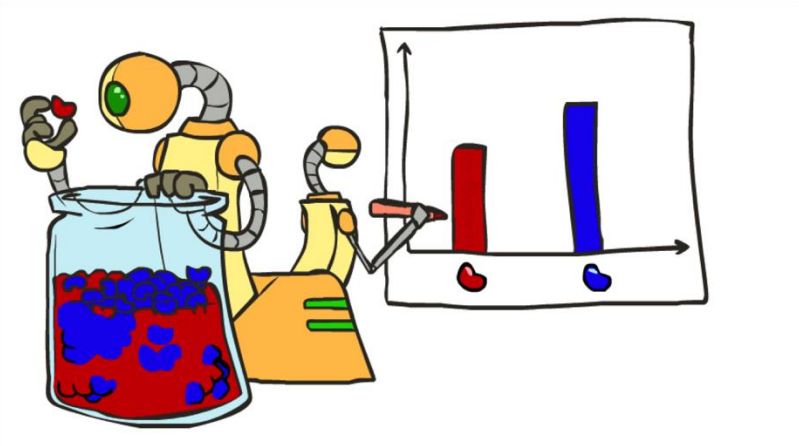
$$P(\text{red}|x) = \frac{e^{w_{\text{red}} \cdot x}}{e^{w_{\text{red}} \cdot x} + e^{w_{\text{blue}} \cdot x}}$$



How to Learn?

- Maximum likelihood estimation

$$\theta_{ML} = \arg\max_{\theta} P(\mathbf{X}|\theta)$$
$$= \arg\max_{\theta} \prod_i P_\theta(X_i)$$



- Maximum *conditional* likelihood estimation

$$\theta^\star = \arg\max_{\theta} P(\mathbf{Y}|\mathbf{X},\theta)$$
$$= \arg\max_{\theta} \prod_i \underbrace{P_\theta(y_i|x_i)}$$
$$\ell(w) = \prod_i \frac{e^{w_{y_i}\cdot x_i}}{\sum_y e^{w_y \cdot x_i}}$$

$$\ell\ell(w) = \sum_i \log P_w(y_i|x_i)$$
$$= \sum_i w_{y_i}\cdot x_i - \log \sum_y e^{w_y \cdot x_i}$$

Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with:

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression

Logistic Regression Demo!

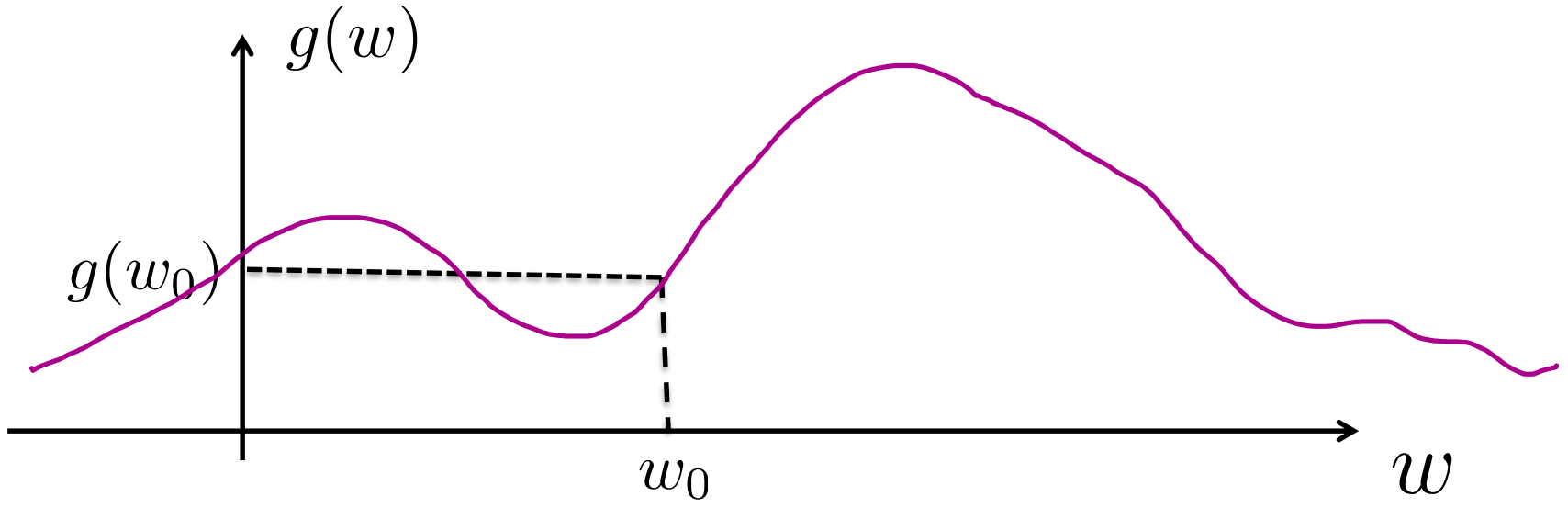
https://playground.tensorflow.org/

Hill Climbing

- Recall from CSPs lecture: simple, general idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What's particularly tricky when hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?

1-D Optimization

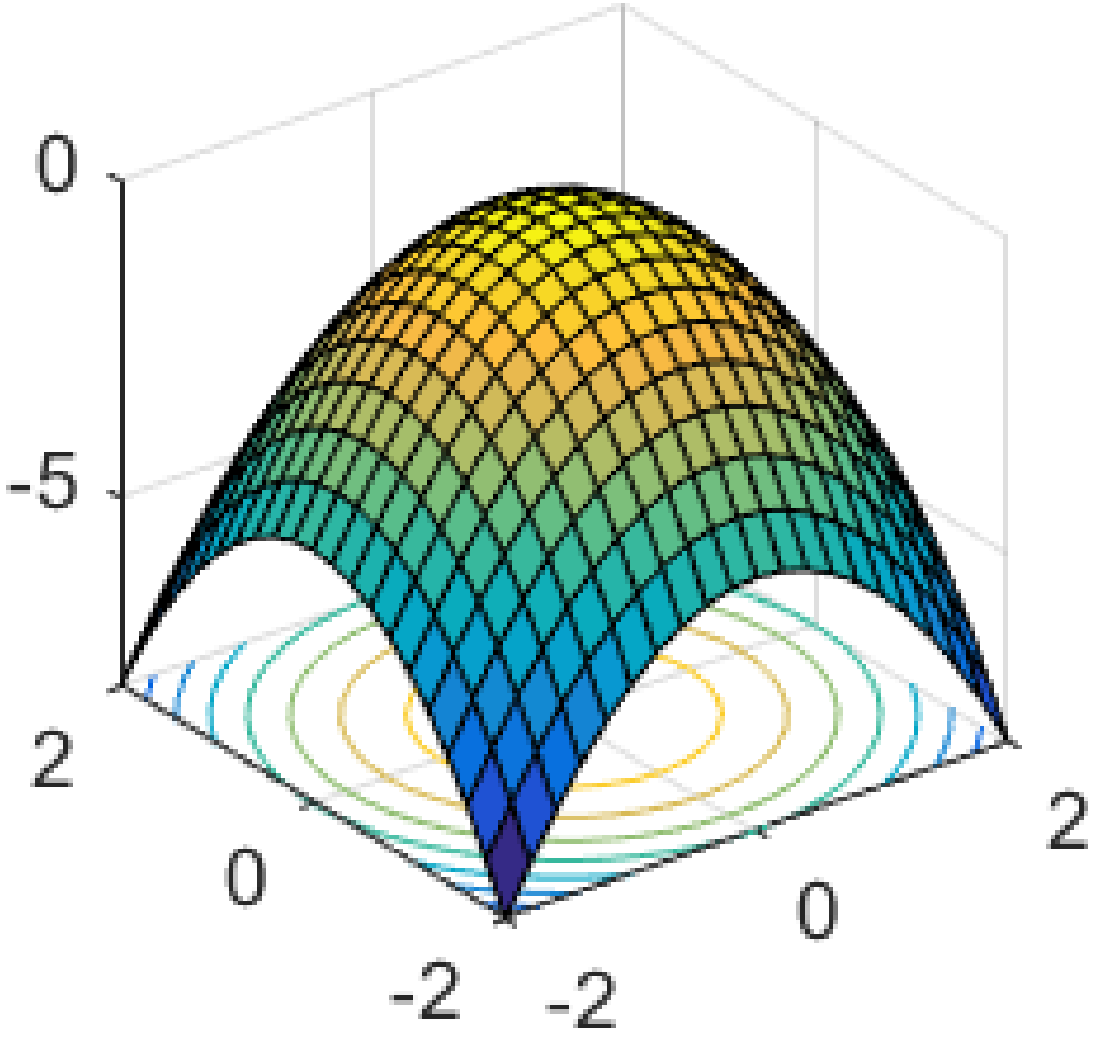


- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction
- Or, evaluate derivative:

$$\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$

  - Tells which direction to step into

2-D Optimization



Source: offconvex.org

Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$
  - Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

  - Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ **= gradient**

Gradient Ascent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction

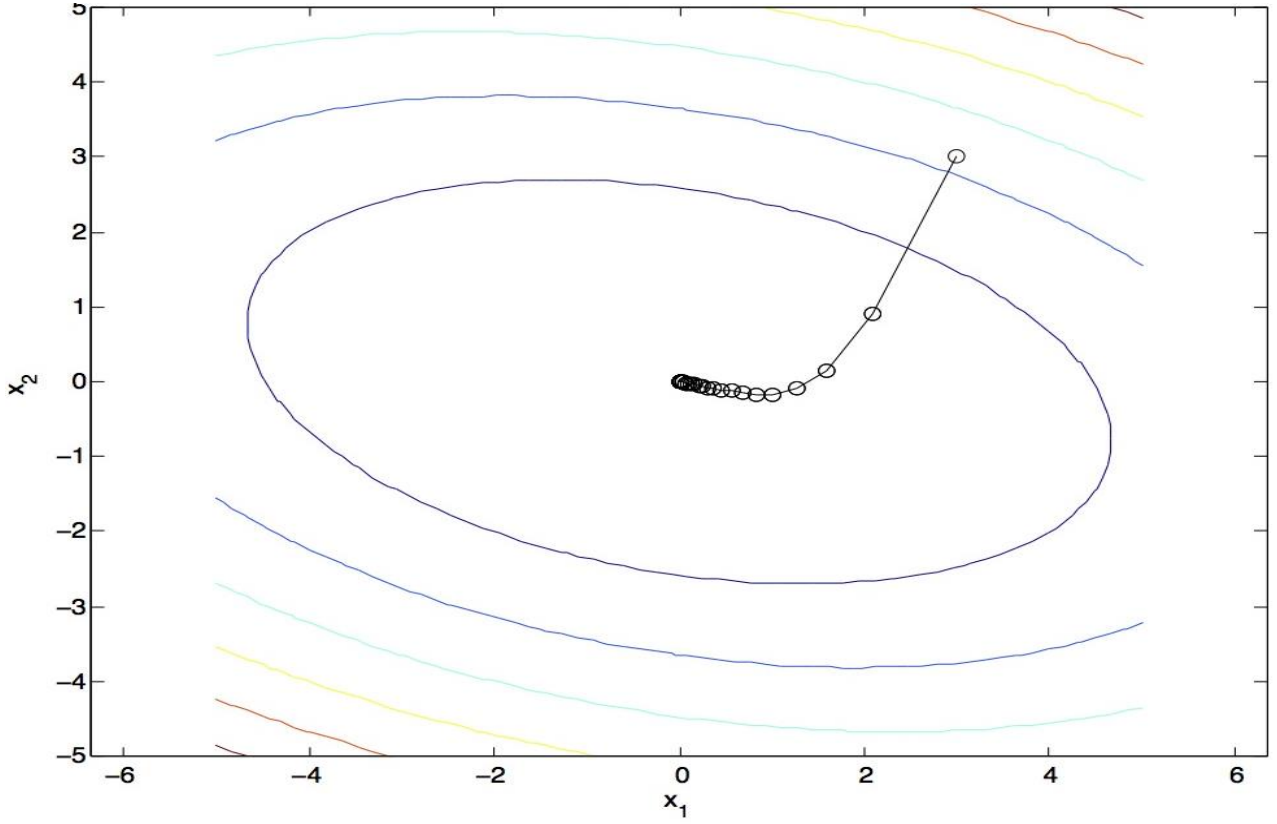
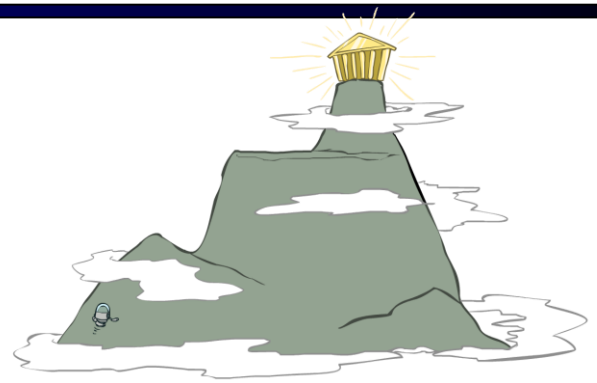


Figure source: Mathworks

What is the Steepest Direction?

$$\max_{\Delta:\Delta_1^2+\Delta_2^2\leq\varepsilon} \quad g(w+\Delta)$$



- First-Order Taylor Expansion: $g(w+\Delta) \approx g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$
- Steepest Descent Direction: $\max_{\Delta:\Delta_1^2+\Delta_2^2\leq\varepsilon} \quad g(w) + \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$
- Recall: $\max_{\Delta:\|\Delta\|\leq\varepsilon} \Delta^\top a \quad \rightarrow \quad \Delta = \varepsilon\frac{a}{\|a\|}$
- Hence, solution: $\Delta = \varepsilon\frac{\nabla g}{\|\nabla g\|}$

Gradient direction = steepest direction!

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \end{bmatrix}$$

Gradient in n dimensions

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \cdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

Optimization Procedure: Gradient Ascent

- `init` w
- `for iter = 1, 2, …`

$$w \leftarrow w + \alpha * \nabla g(w)$$

- α : learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes w about 0.1 – 1 %

Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \underbrace{\sum_i \log P(y^{(i)}|x^{(i)};w)}_{g(w)}$$

- init w
- for iter = 1, 2, …

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)};w)$$

Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: once gradient on one training example has been computed, might as well incorporate before computing next one

- `init` w
- `for iter = 1, 2, …`
  - `pick random j`

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Mini-Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- init w
- for iter = 1, 2, …
  - pick random subset of training examples J

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Gradient for Logistic Regression

$$\ell\ell(w) = \sum_i \log P_w(y_i|x_i)$$

$$= \sum_i w_{y_i} \cdot f(x_i) - \log \sum_y e^{w_y \cdot f(x_i)}$$

$$\frac{d}{dw_y} \log P_w(y_i|x_i) = \begin{cases} f(x_i) - f(x_i) \frac{e^{w_y \cdot f(x_i)}}{\sum_{y'} e^{w_{y'} \cdot f(x_i)}} & \text{if } y = y_i \\ -f(x_i) \frac{e^{w_y \cdot f(x_i)}}{\sum_{y'} e^{w_{y'} \cdot f(x_i)}} & \text{otherwise} \end{cases}$$

$$= f(x_i)(I(y = y_i) - P(y|x_i))$$

- Recall perceptron:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

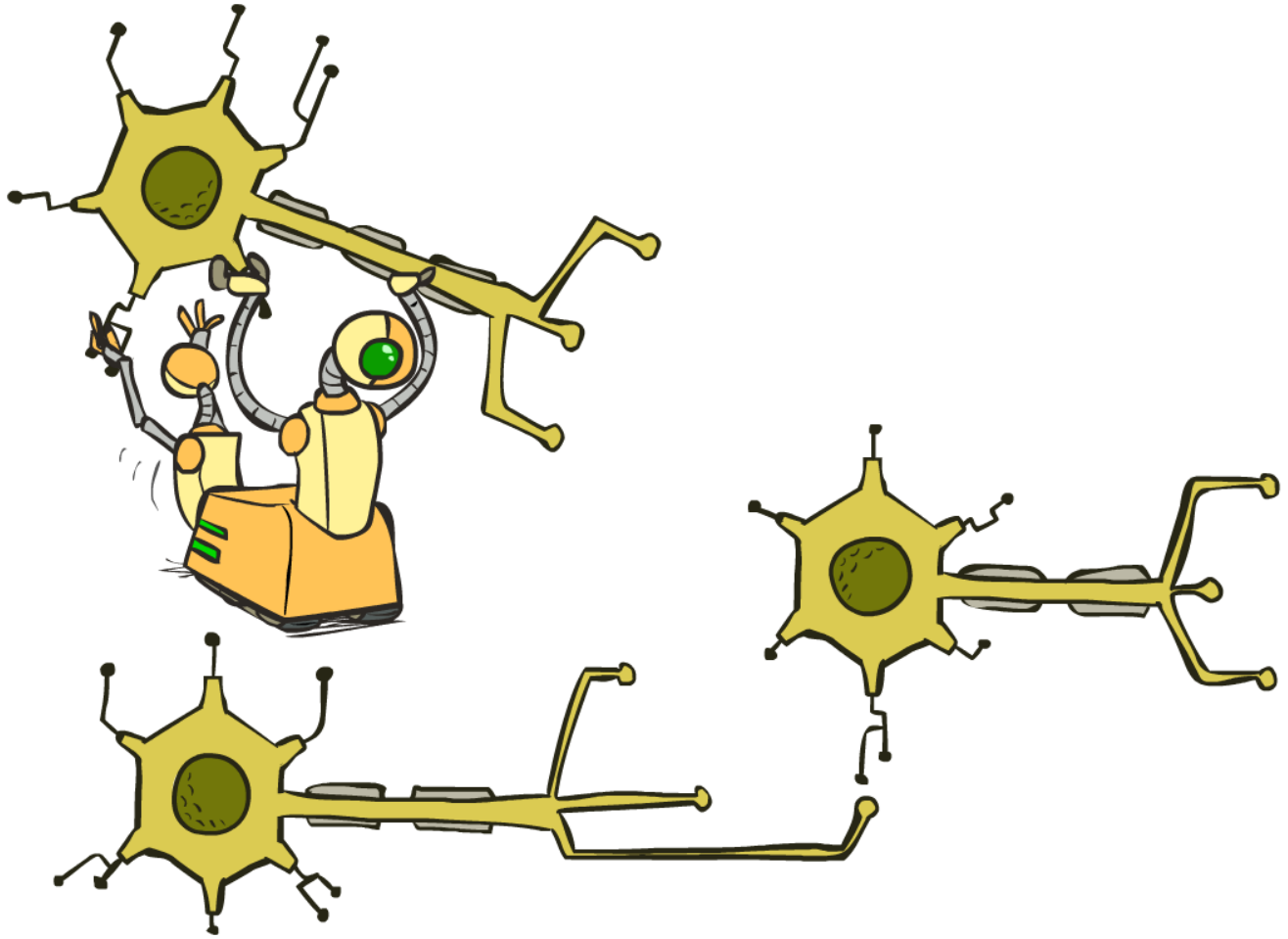
  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$

How about computing all the derivatives?

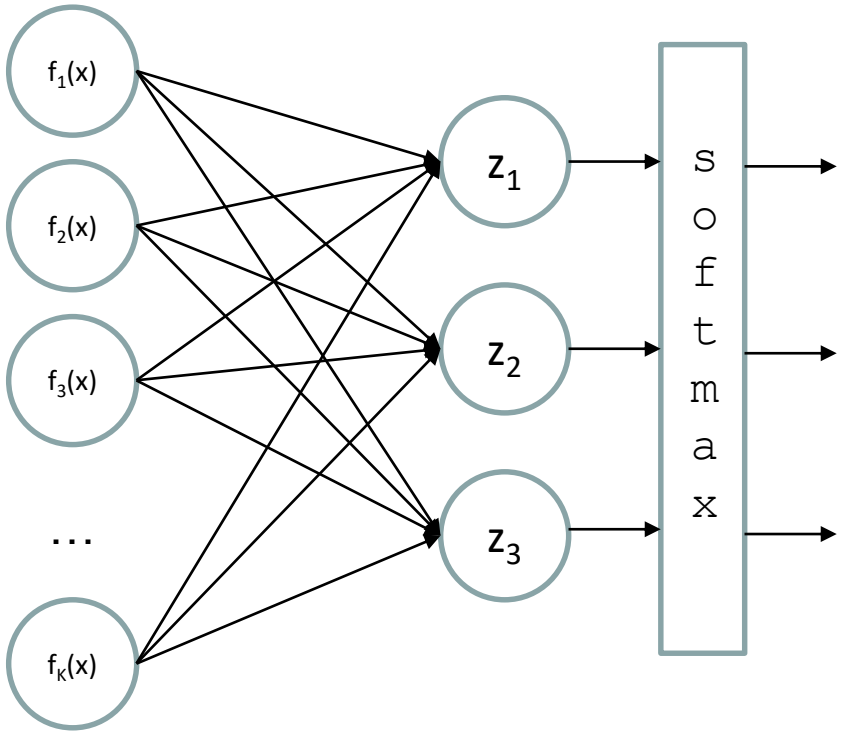
- We'll talk about that once we covered neural networks, which are a generalization of logistic regression

Neural Networks



Multi-class Logistic Regression

- = special case of neural network

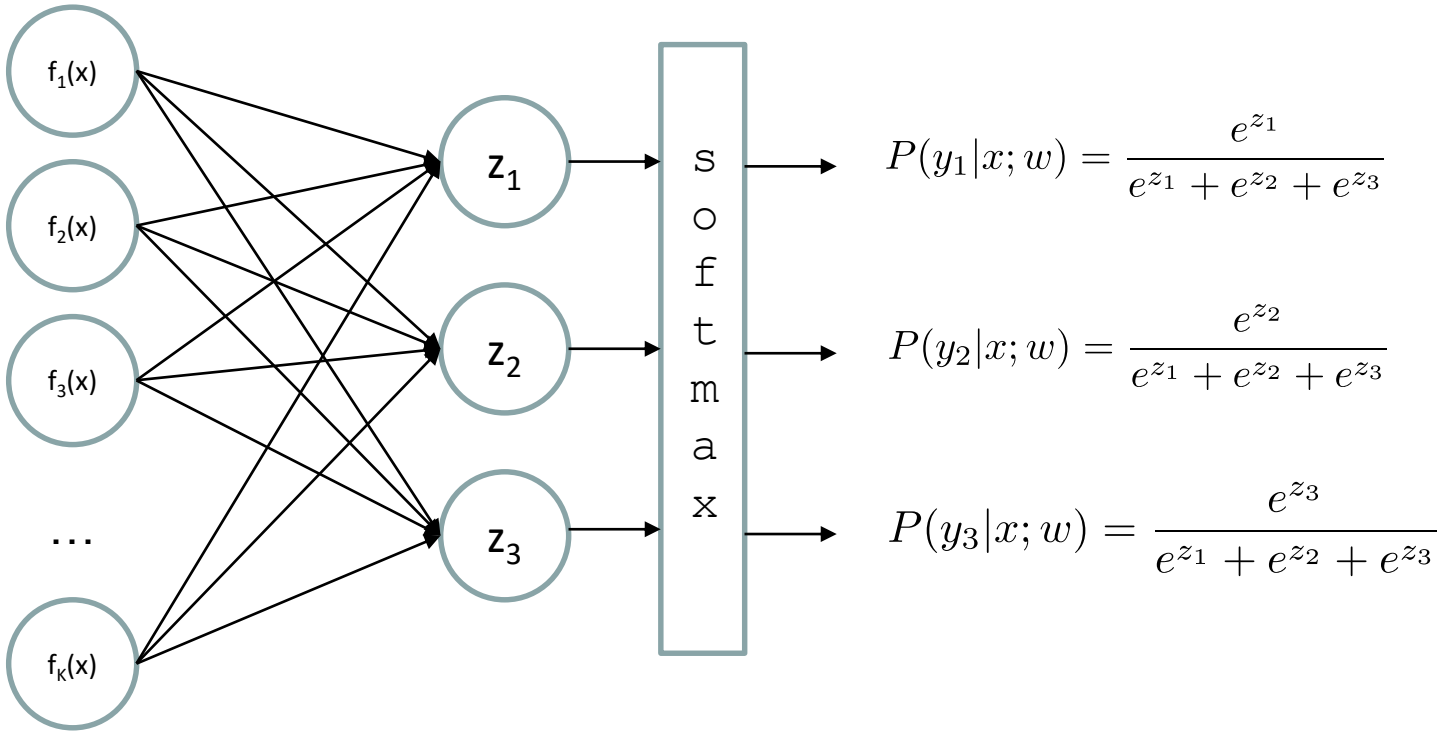


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

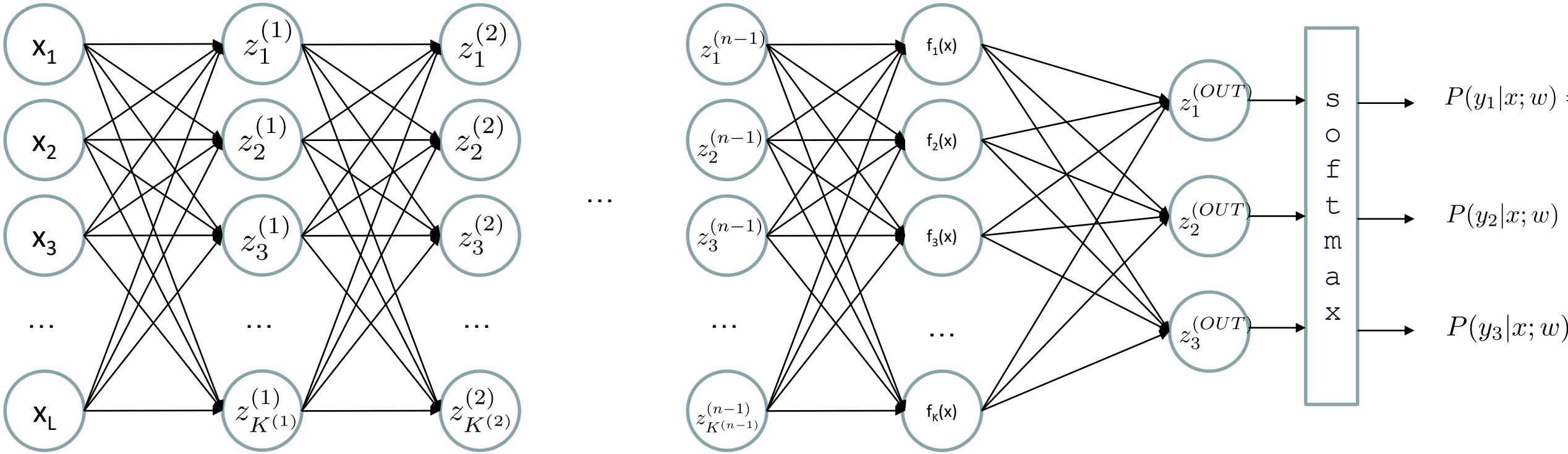
$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Deep Neural Network = Also learn the features!



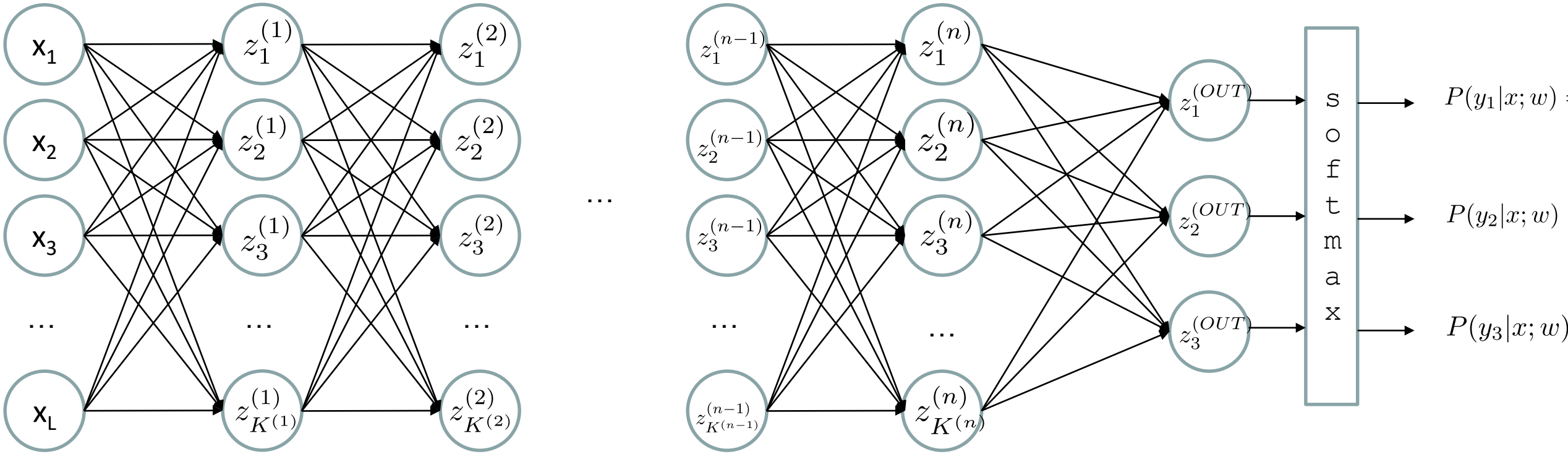
Deep Neural Network = Also learn the features!



$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Deep Neural Network = Also learn the features!

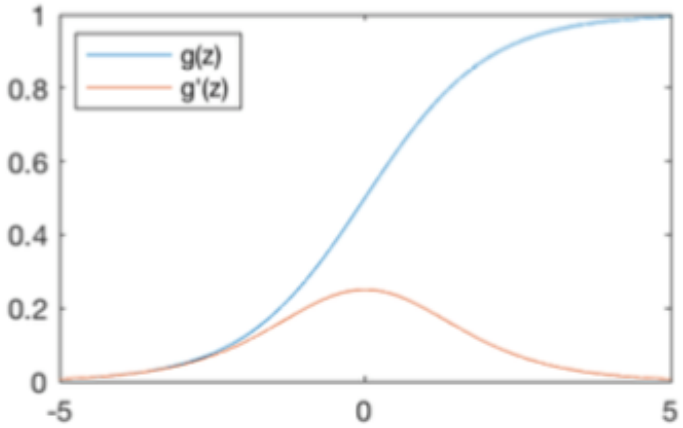


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Common Activation Functions

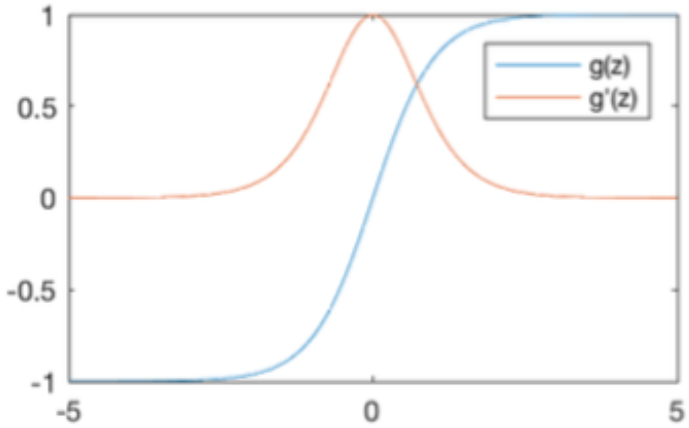
Sigmoid Function



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

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

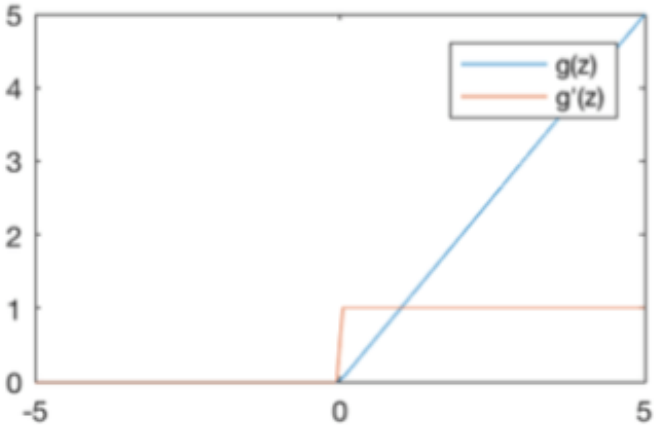
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)



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

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

[source: MIT 6.S191 introtodeeplearning.com]

Deep Neural Network: Also Learn the Features!

- Training the deep neural network is just like logistic regression:

$$\max_w \quad ll(w) = \max_w \quad \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

just w tends to be a much, much larger vector ☺

→just run gradient ascent

+ stop when log likelihood of hold-out data starts to decrease

Neural Networks Properties

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

- Practical considerations
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
    - (hence early stopping!)

Neural Net Demo!

https://playground.tensorflow.org/

How about computing all the derivatives?

- Derivatives tables:

$$\frac{d}{dx}(a) = 0$$

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(au) = a\frac{du}{dx}$$

$$\frac{d}{dx}(u + v - w) = \frac{du}{dx} + \frac{dv}{dx} - \frac{dw}{dx}$$

$$\frac{d}{dx}(uv) = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{1}{v}\frac{du}{dx} - \frac{u}{v^2}\frac{dv}{dx}$$

$$\frac{d}{dx}(u^n) = nu^{n-1}\frac{du}{dx}$$

$$\frac{d}{dx}(\sqrt{u}) = \frac{1}{2\sqrt{u}}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u}\right) = -\frac{1}{u^2}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u^n}\right) = -\frac{n}{u^{n+1}}\frac{du}{dx}$$

$$\frac{d}{dx}[f(u)] = \frac{d}{du}[f(u)]\frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{d}{dx}[\log_e u] = \frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}[\log_a u] = \log_a e\frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}e^u = e^u\frac{du}{dx}$$

$$\frac{d}{dx}a^u = a^u \ln a\frac{du}{dx}$$

$$\frac{d}{dx}\left(u^v\right) = vu^{v-1}\frac{du}{dx} + \ln u \;\; u^v\frac{dv}{dx}$$

$$\frac{d}{dx}\sin u = \cos u\frac{du}{dx}$$

$$\frac{d}{dx}\cos u = -\sin u\frac{du}{dx}$$

$$\frac{d}{dx}\tan u = \sec^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\cot u = -\csc^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\sec u = \sec u \tan u\frac{du}{dx}$$

$$\frac{d}{dx}\csc u = -\csc u \cot u\frac{du}{dx}$$

[source: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/derfunc.html

How about computing all the derivatives?

- But neural net f is never one of those?
  - No problem: CHAIN RULE:

If $$f(x) = g(h(x))$$

Then $$f'(x) = g'(h(x))h'(x)$$

→ **Derivatives can be computed by following well-defined procedures**

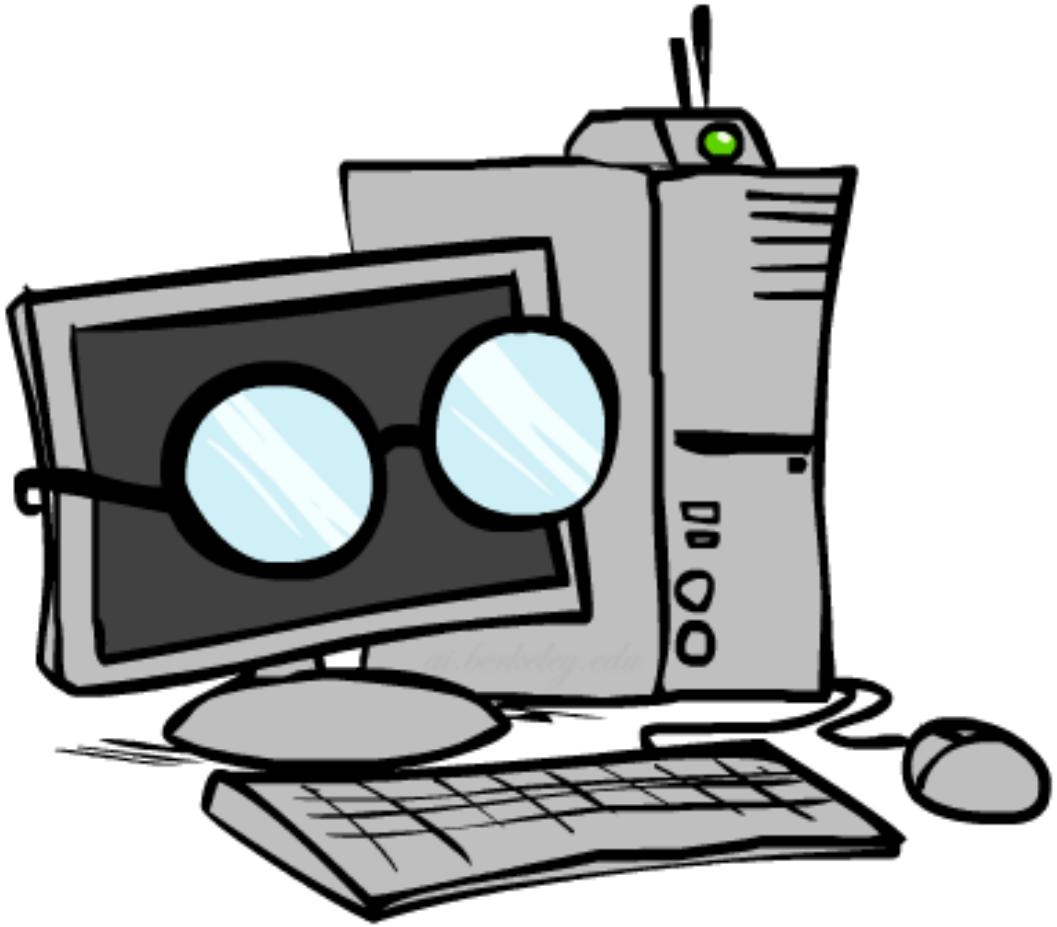
Automatic Differentiation

- Automatic differentiation software
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function g(x,y,w)
  - Can automatically compute all derivatives w.r.t. all entries in w
  - This is typically done by caching info during forward computation pass of f, and then doing a backward pass = “backpropagation”
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass
- Need to know this exists
- How this is done? -- outside of scope of CS188

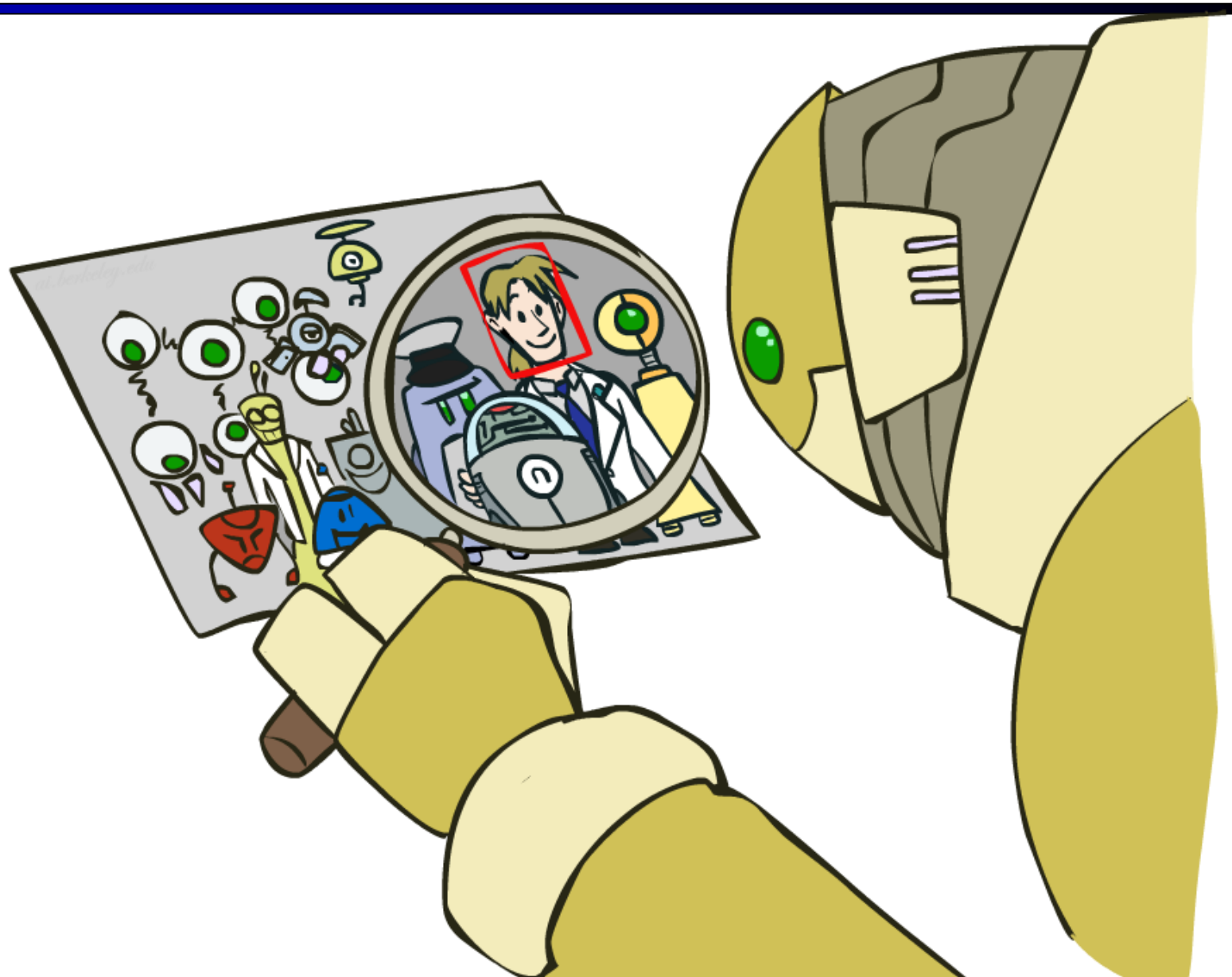
Summary of Key Ideas

- Optimize probability of label given input $\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)};w)$
- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = "early stopping")
- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - → the features are learned rather than hand-designed
  - Universal function approximation theorem
    - `If` neural net is large enough
    - `Then` neural net can represent any continuous mapping from input to output with arbitrary accuracy
    - But remember: need to avoid overfitting / memorizing the training data → early stopping!
  - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of 188)

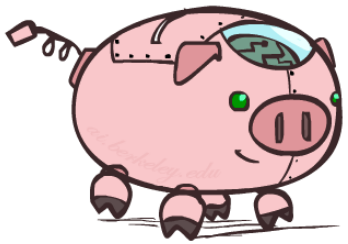
Computer Vision

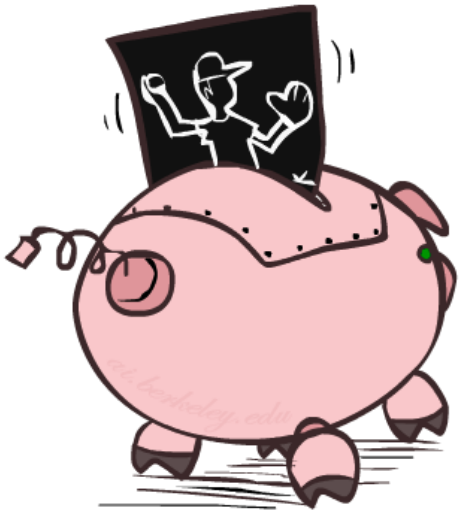


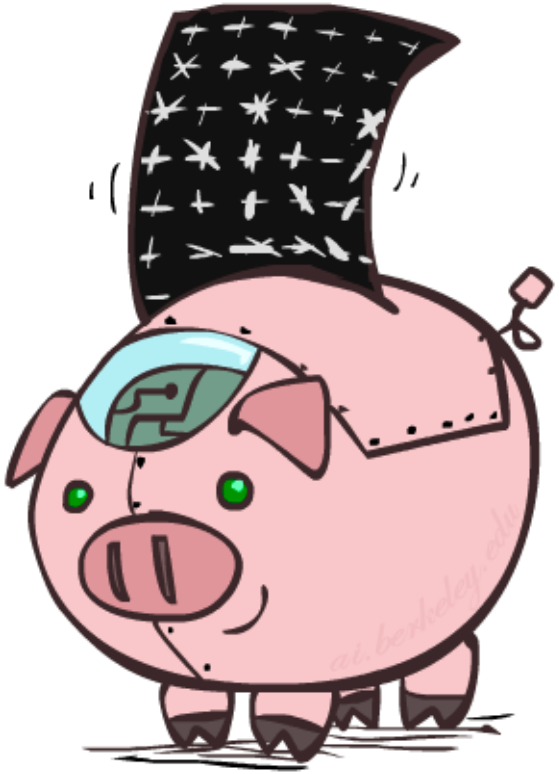
Object Detection



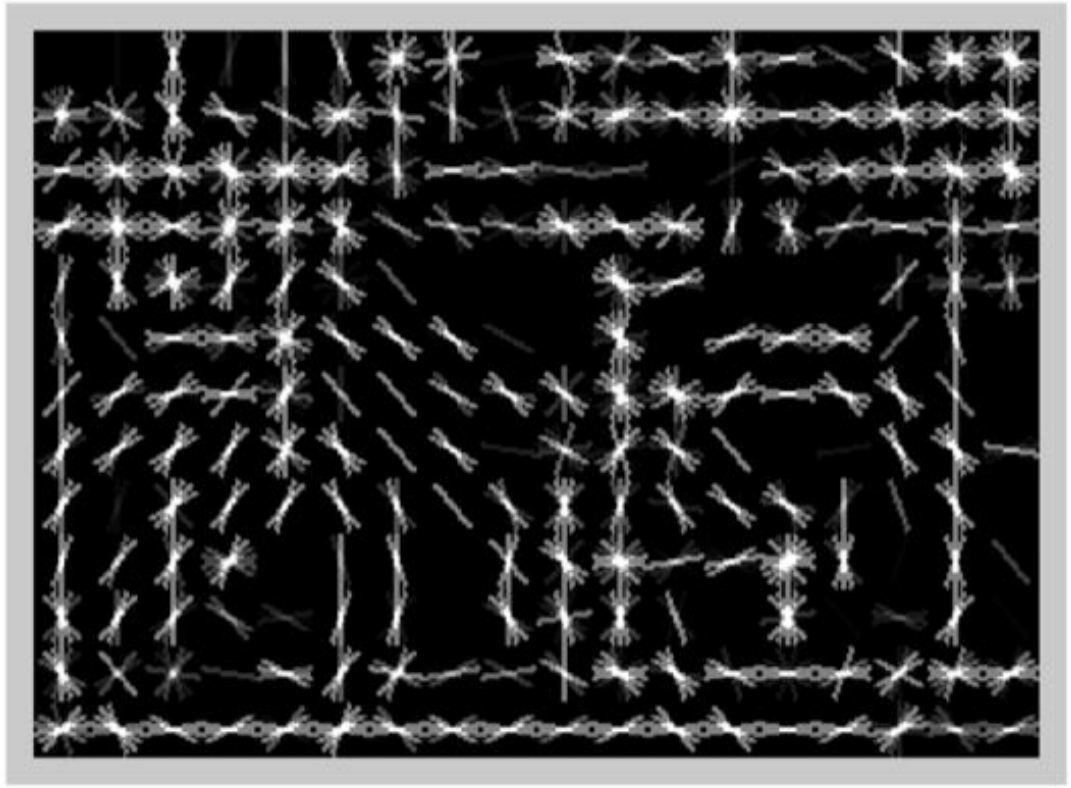
Manual Feature Design







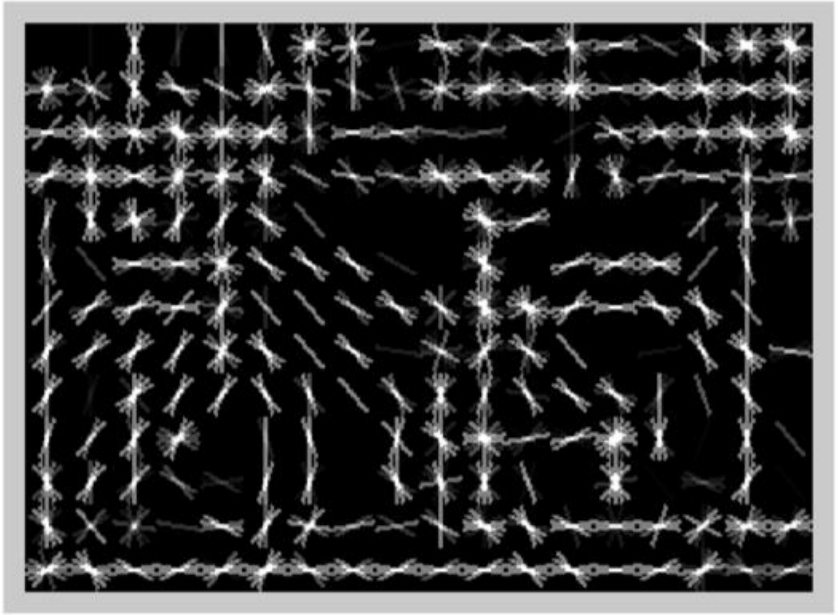
Features and Generalization



[HoG: Dalal and Triggs, 2005]

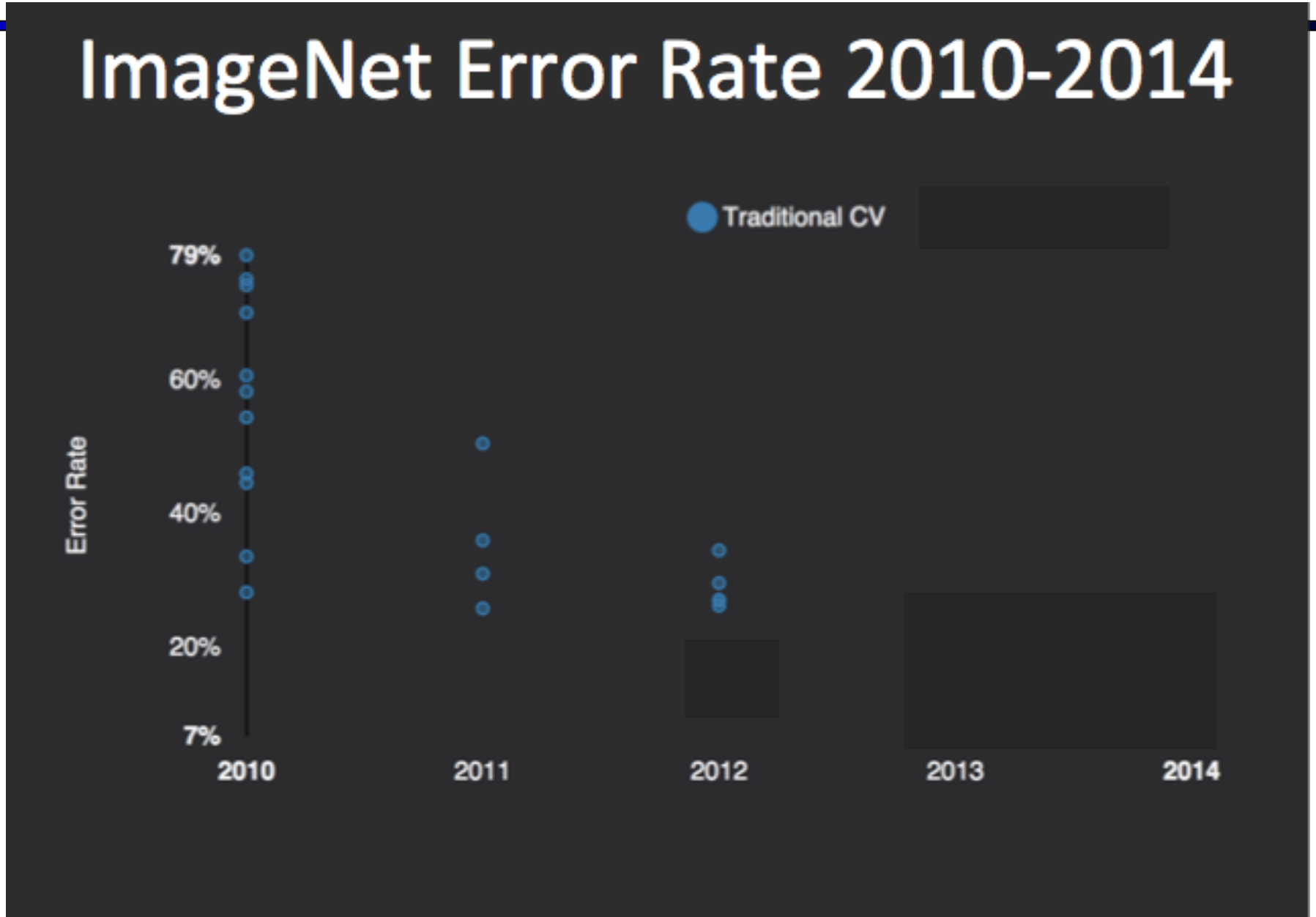
Features and Generalization

Image



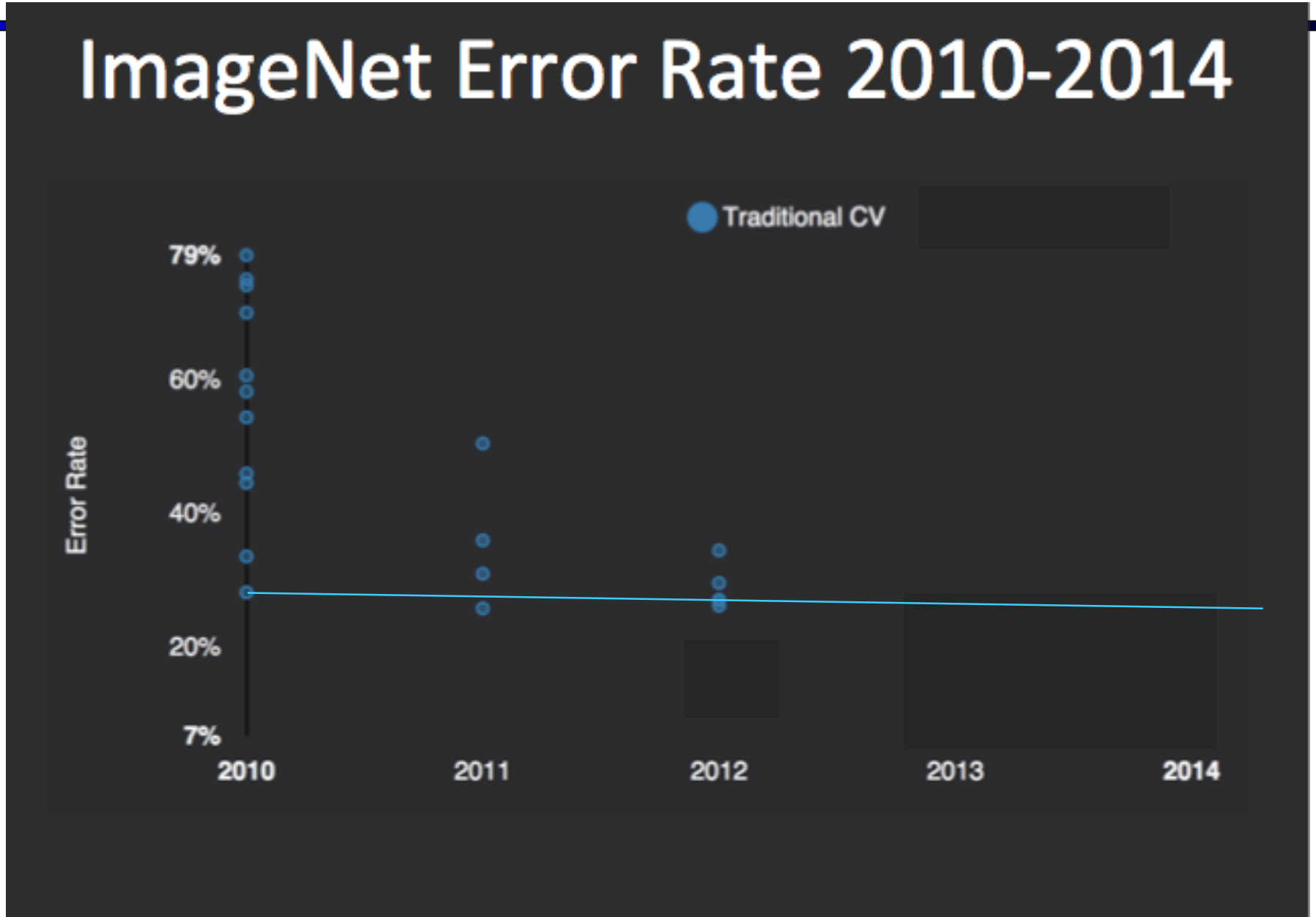
HoG

Performance



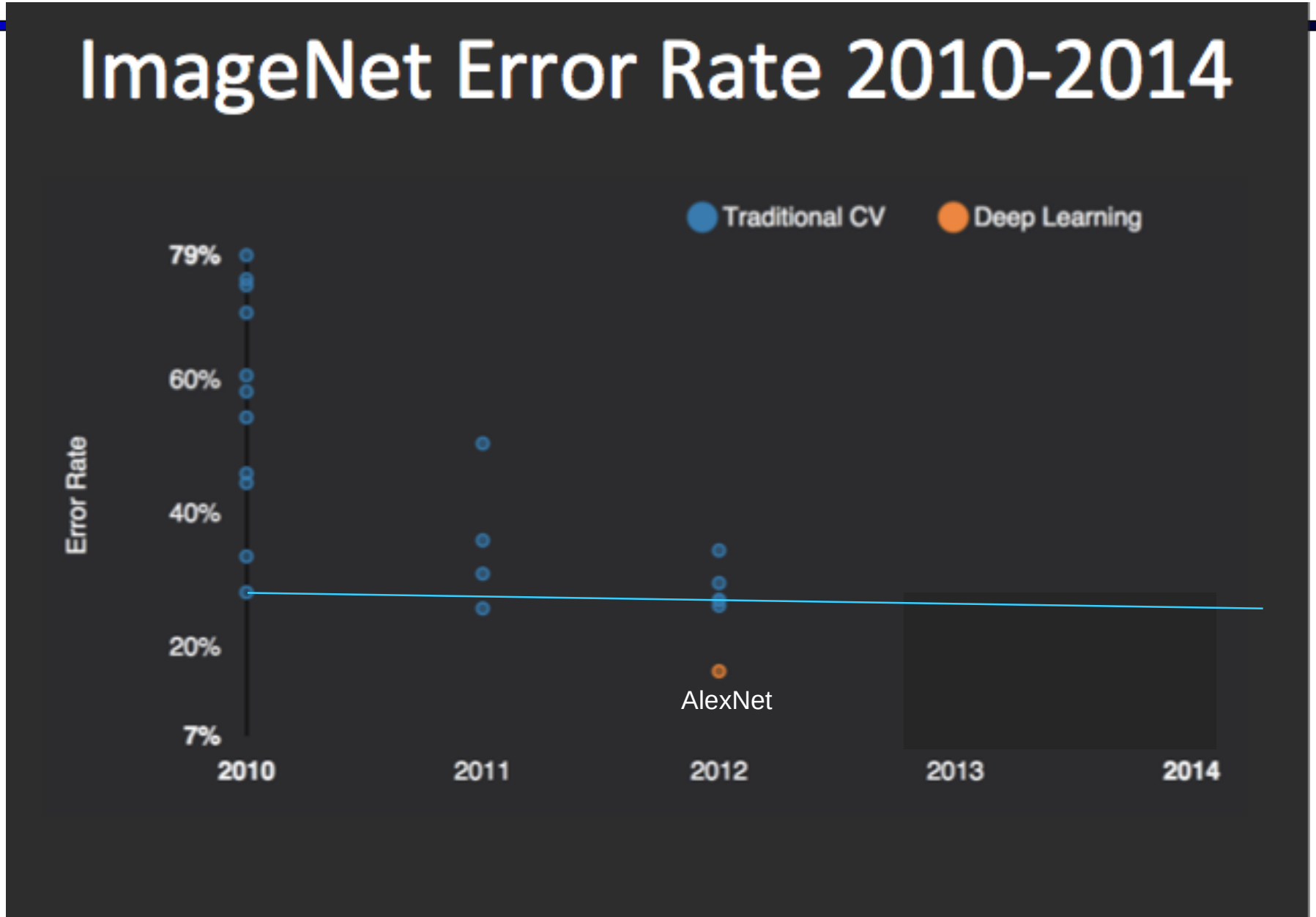
graph credit Matt Zeiler, Clarifai

Performance



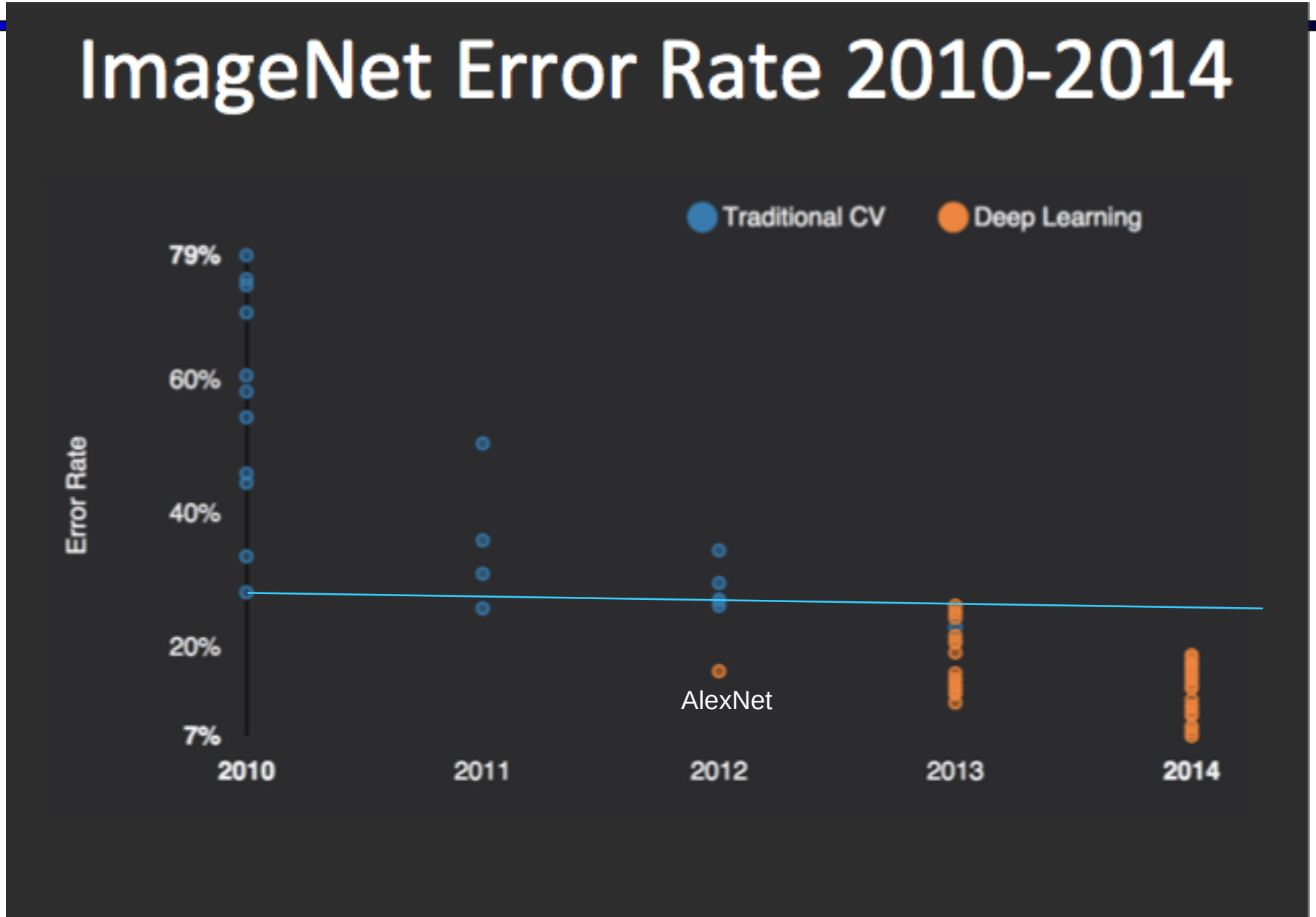
graph credit Matt Zeiler, Clarifai

Performance



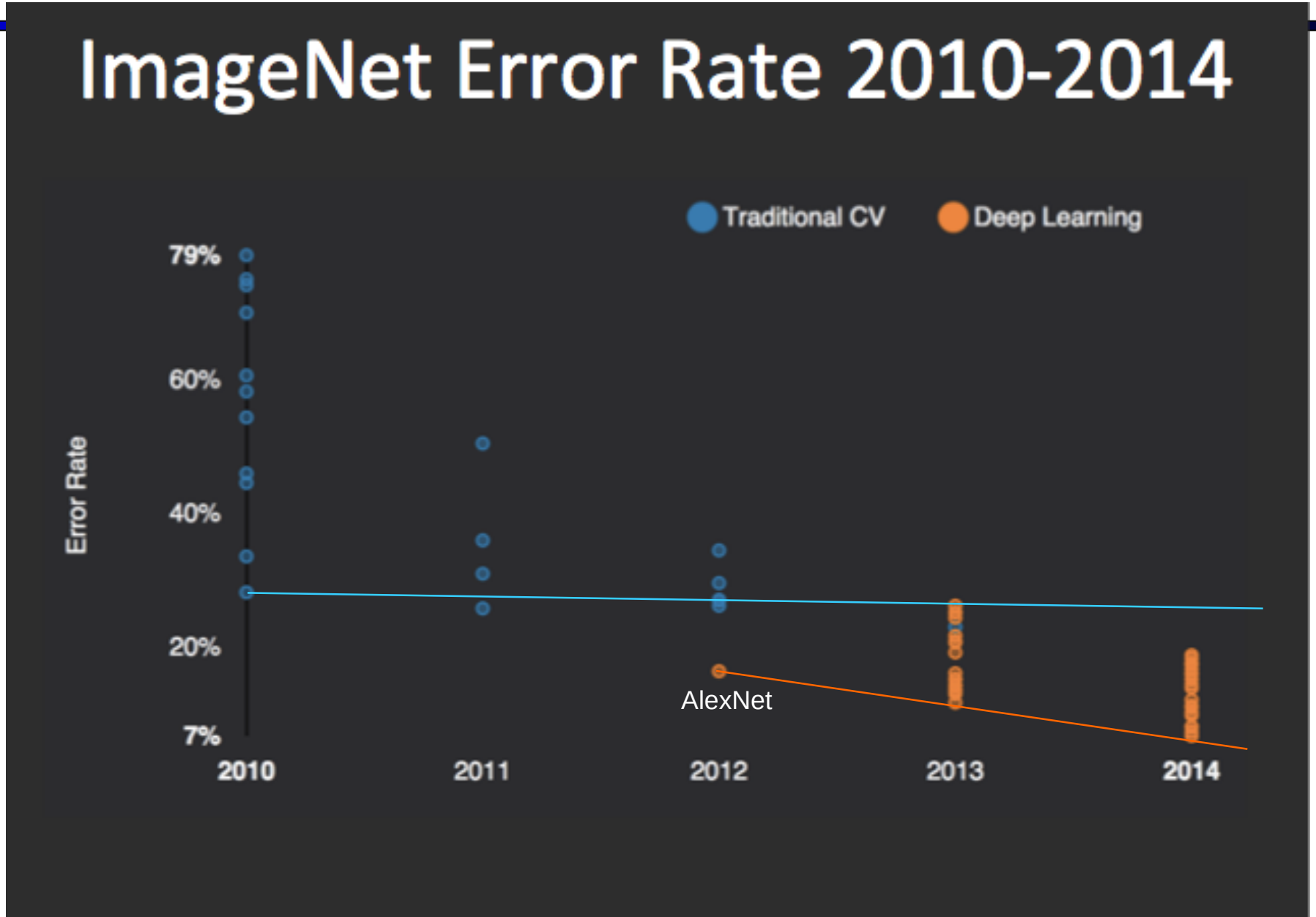
graph credit Matt Zeiler, Clarifai

Performance



graph credit Matt Zeiler, Clarifai

Performance



graph credit Matt Zeiler, Clarifai

MS COCO Image Captioning Challenge

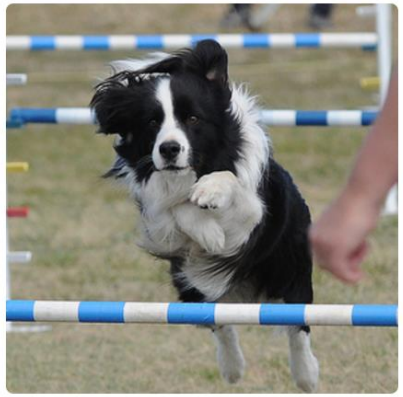
"man in black shirt is playing guitar."

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

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."



"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Karpathy & Fei-Fei, 2015; Donahue et al., 2015; Xu et al, 2015; many more

Visual QA Challenge

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh

What vegetable is on the plate?
Neural Net: broccoli
Ground Truth: broccoli

What color are the shoes on the person's feet ?
Neural Net: brown
Ground Truth: brown

How many school busses are there?
Neural Net: 2
Ground Truth: 2

What sport is this?
Neural Net: baseball
Ground Truth: baseball

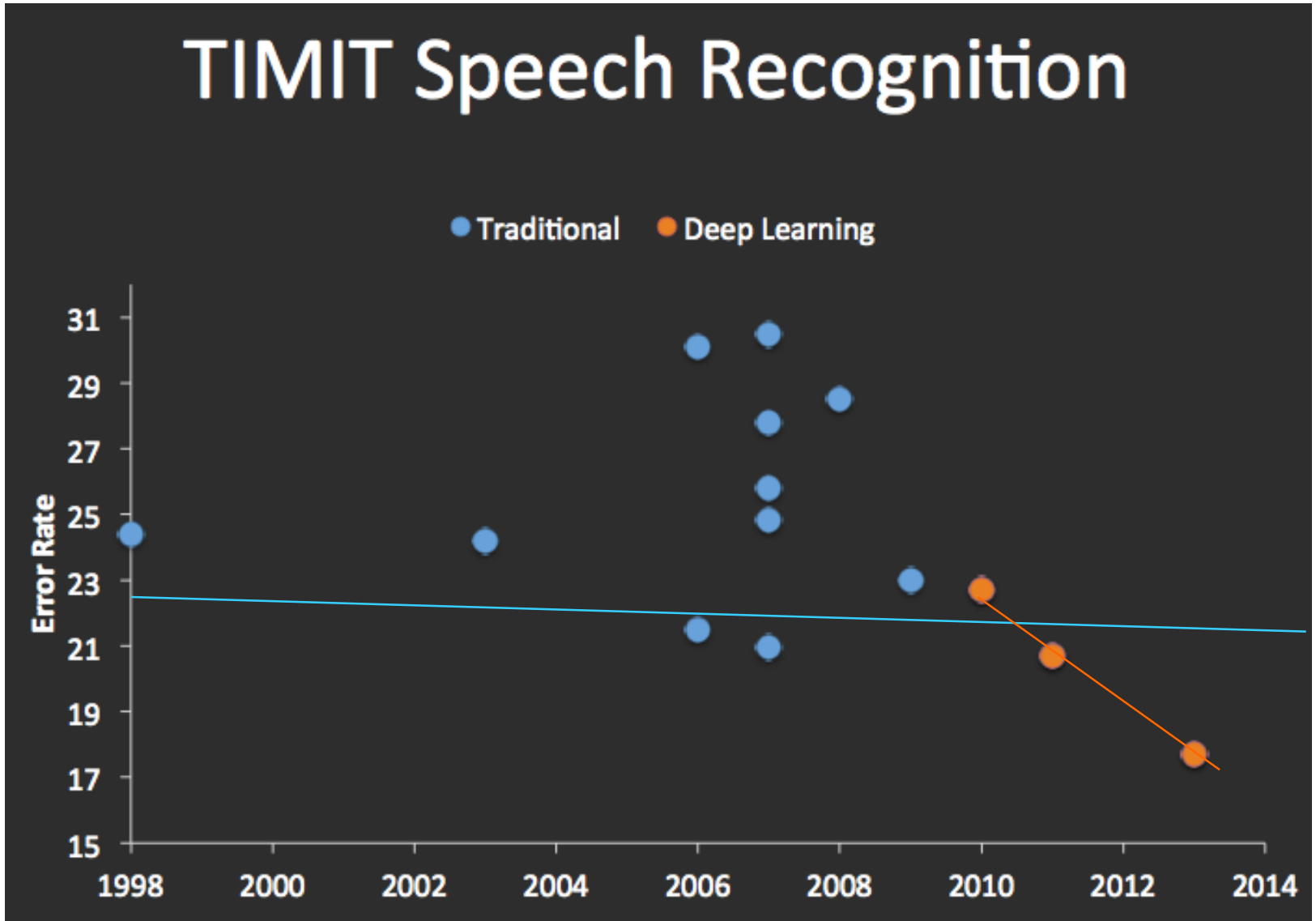
What is on top of the refrigerator?
Neural Net: magnets
Ground Truth: cereal

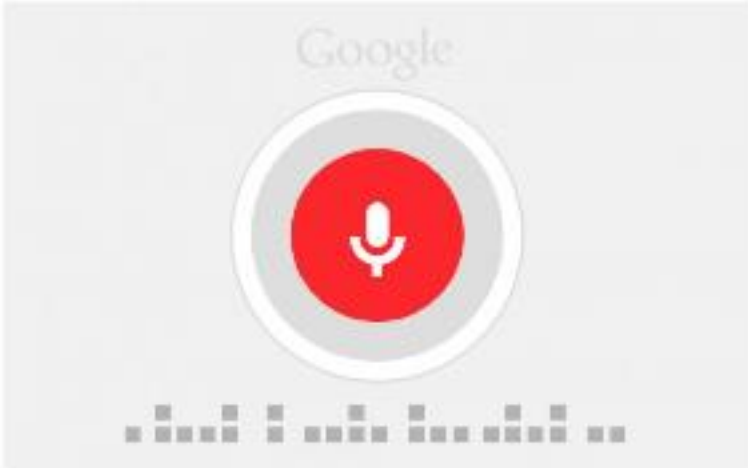
What uniform is she wearing?
Neural Net: shorts
Ground Truth: girl scout

What is the table number?
Neural Net: 4
Ground Truth:40

What are people sitting under in the back?
Neural Net: bench
Ground Truth: tent

Speech Recognition





graph credit Matt Zeiler, Clarifai

Machine Translation

Google Neural Machine Translation (in production)