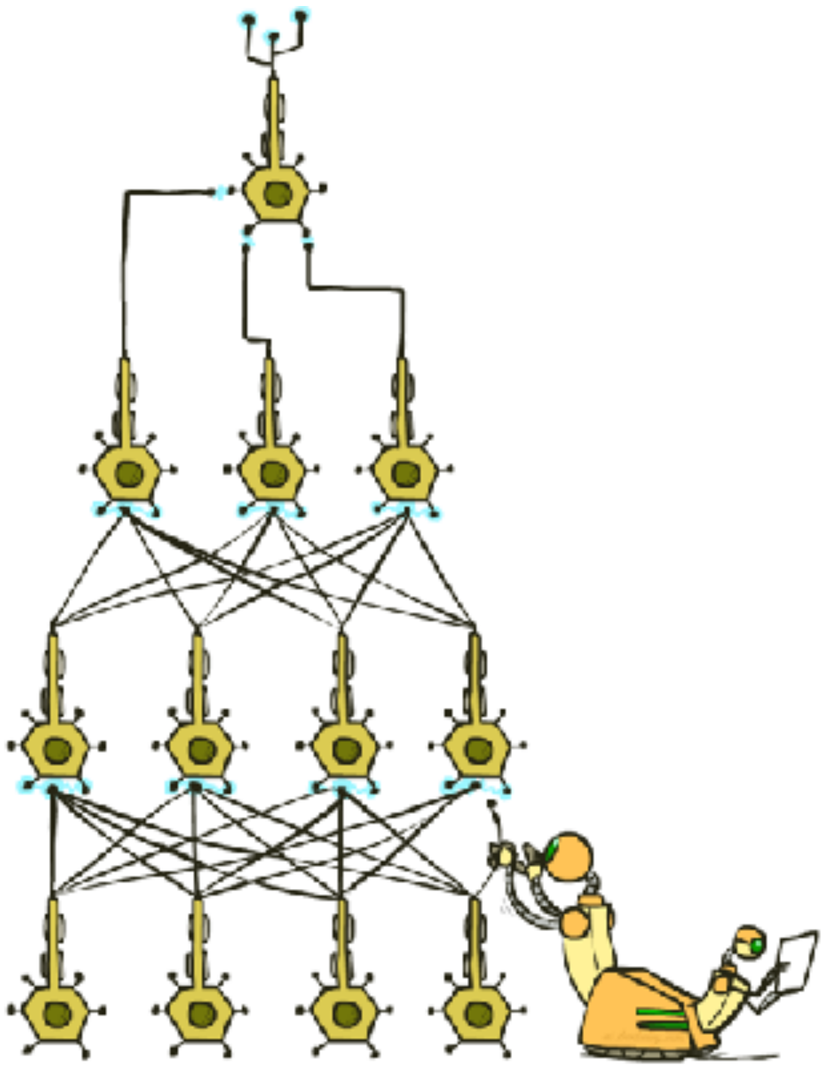


CS 383: Artificial Intelligence

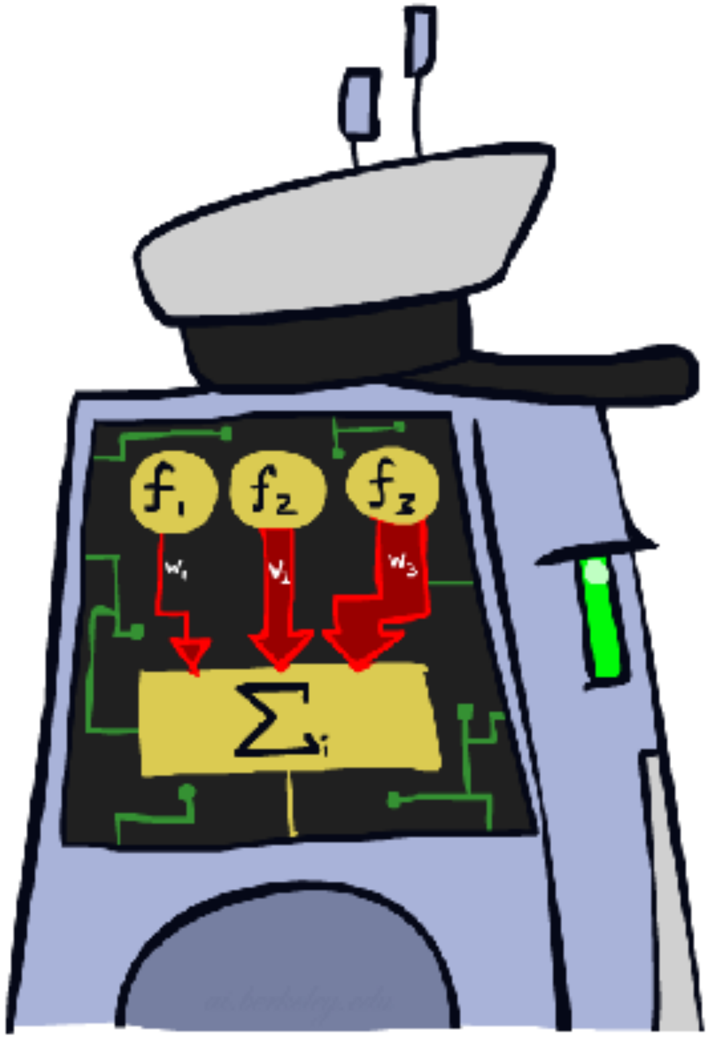
Deep Learning

Prof. Scott Niekum — UMass Amherst

[These slides based on those of Dan Klein, Pieter Abbeel, Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Please fill out course evals online!

Review: Linear Classifiers



Feature Vectors

x $\quad f(x)$ $\quad y$

```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

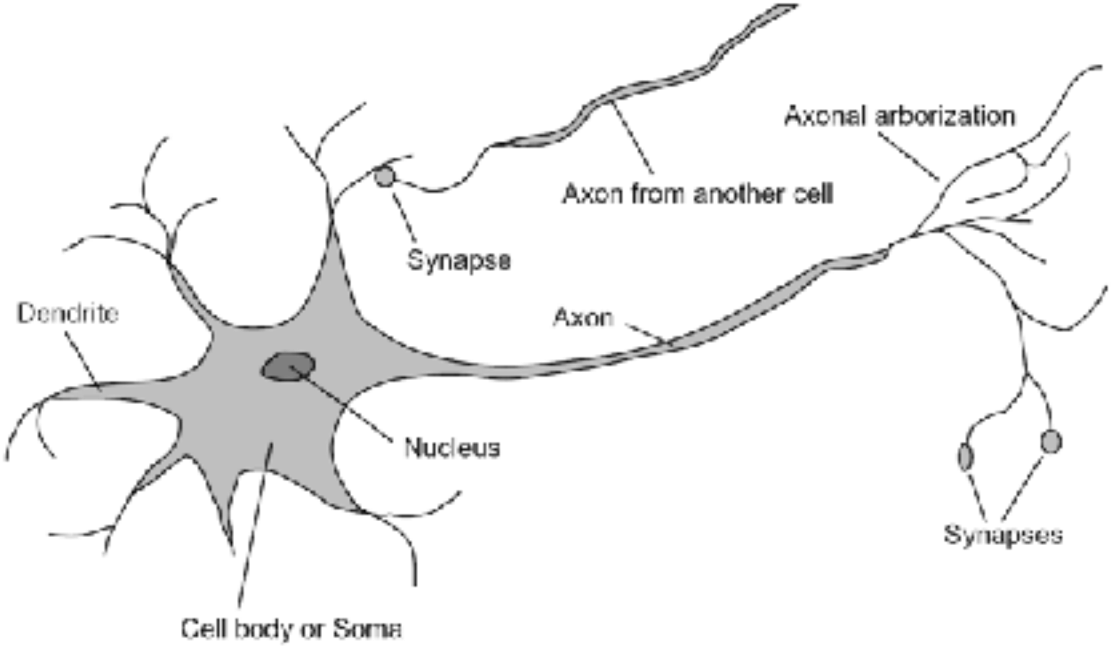
SPAM
or
+

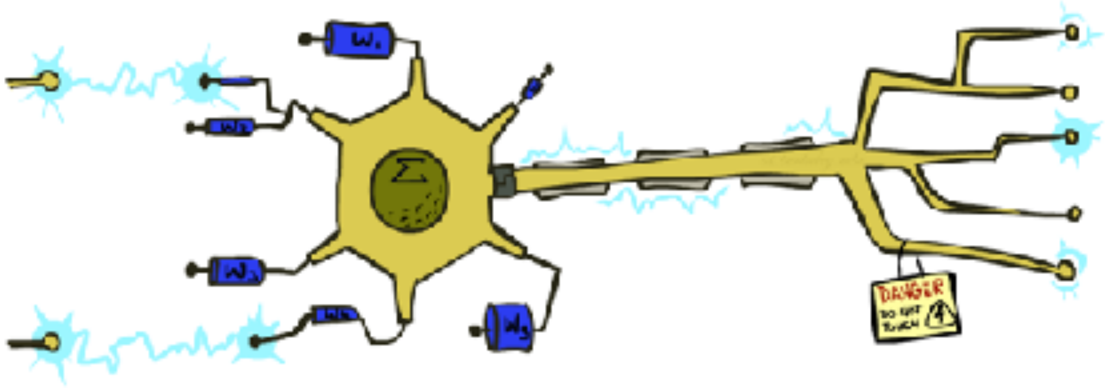
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

“2”

Some (Simplified) Biology

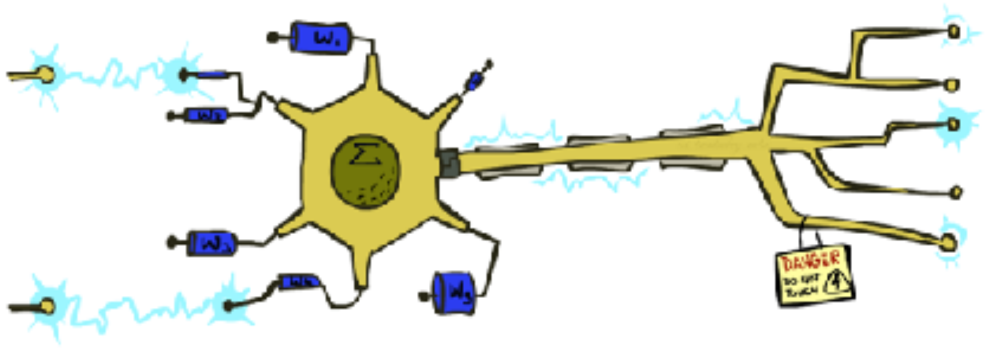
- Very loose inspiration: human neurons





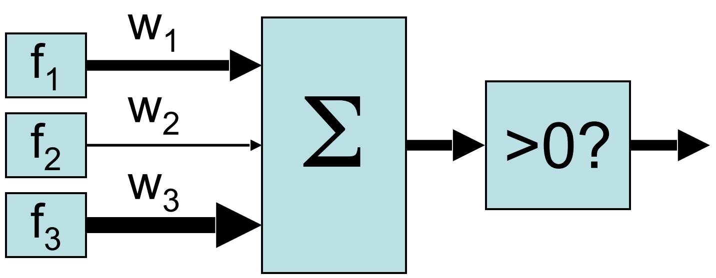
Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

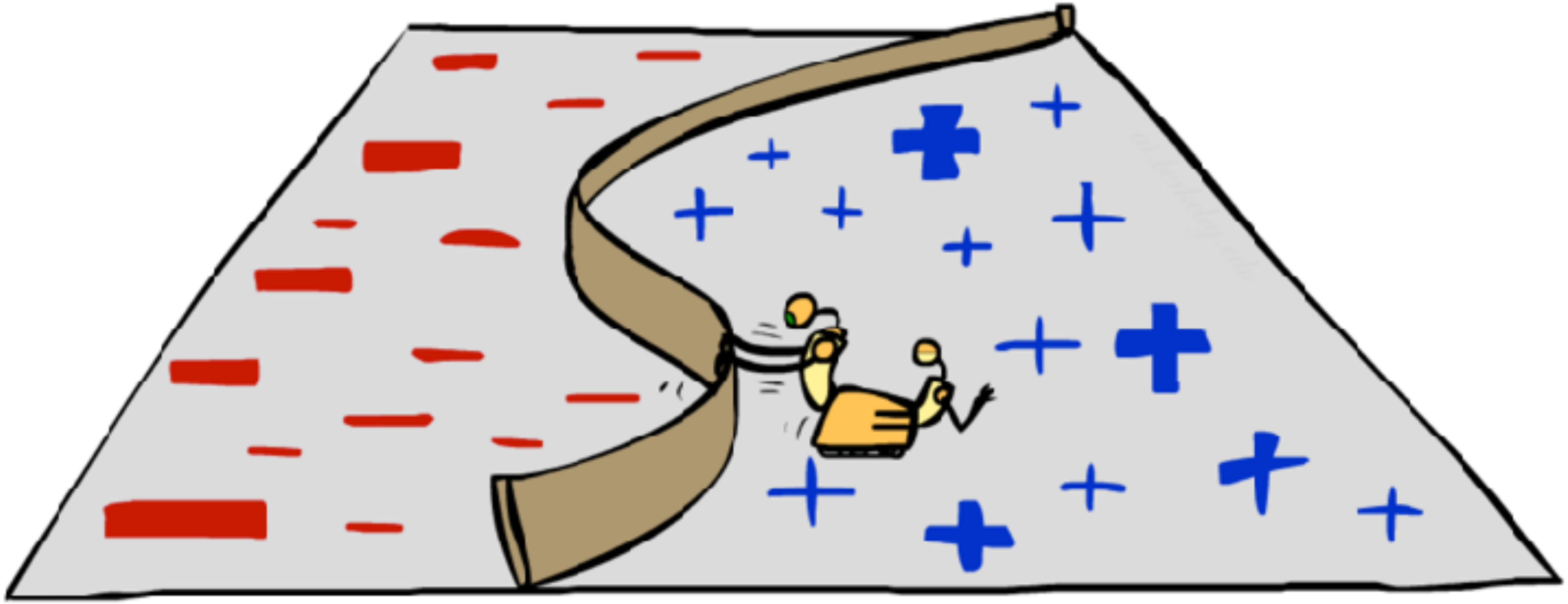


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output +1
  - Negative, output -1

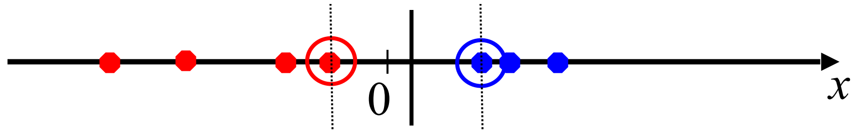


Non-Linearity

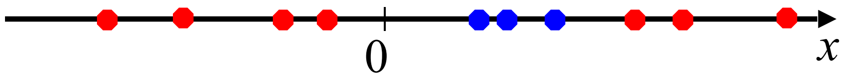


Non-Linear Separators

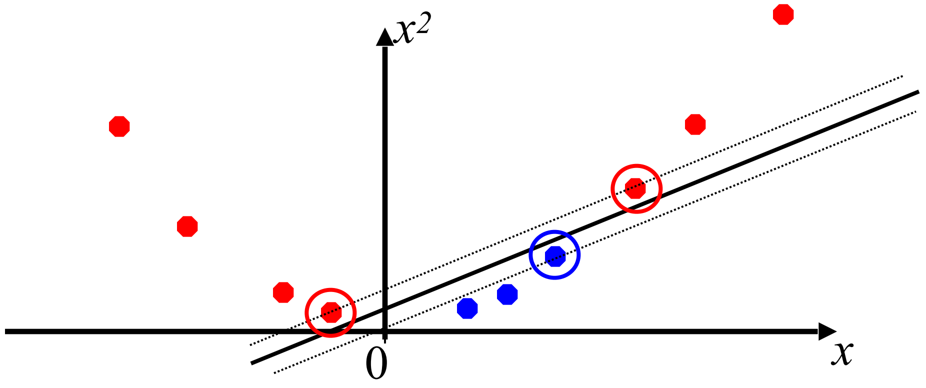
- Data that is linearly separable works out great for linear decision rules:



- But what are we going to do if the dataset is just too hard?



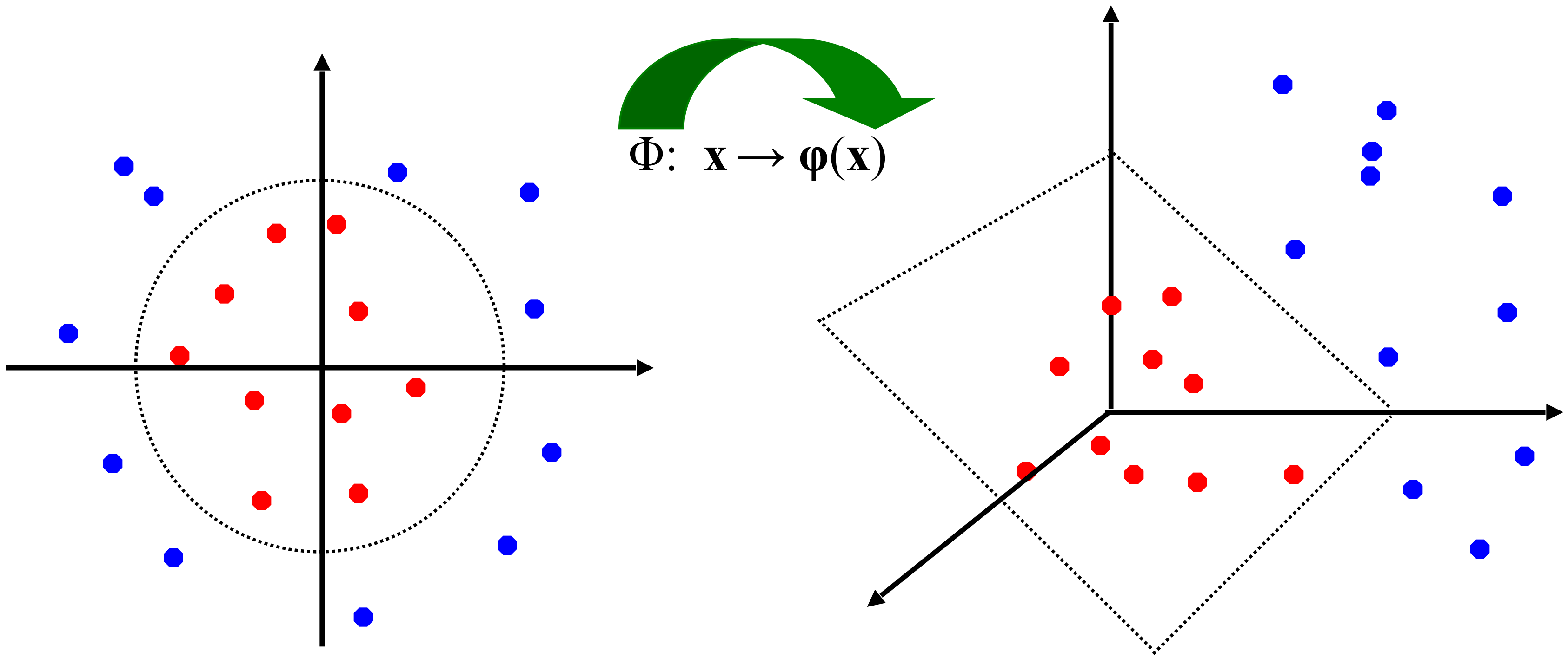
- How about… mapping data to a higher-dimensional space:



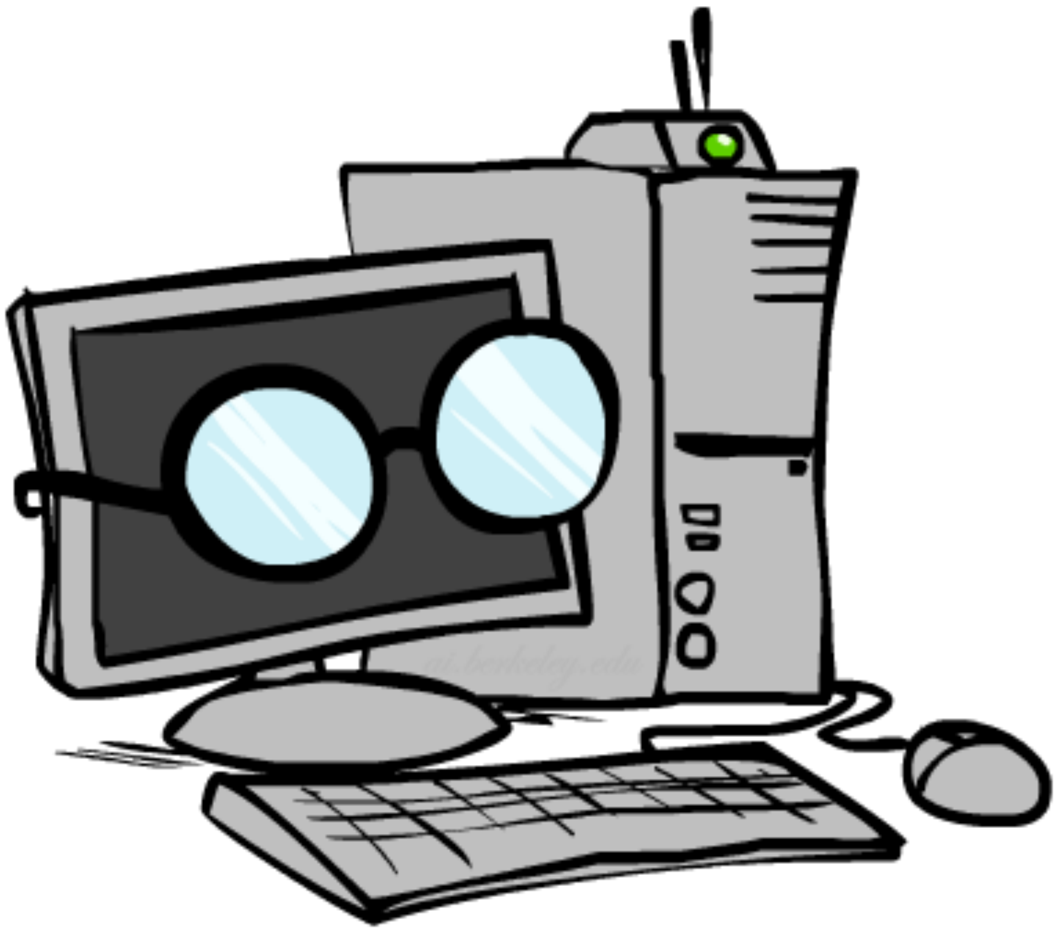
This and next slide adapted from Ray Mooney, UT

Non-Linear Separators

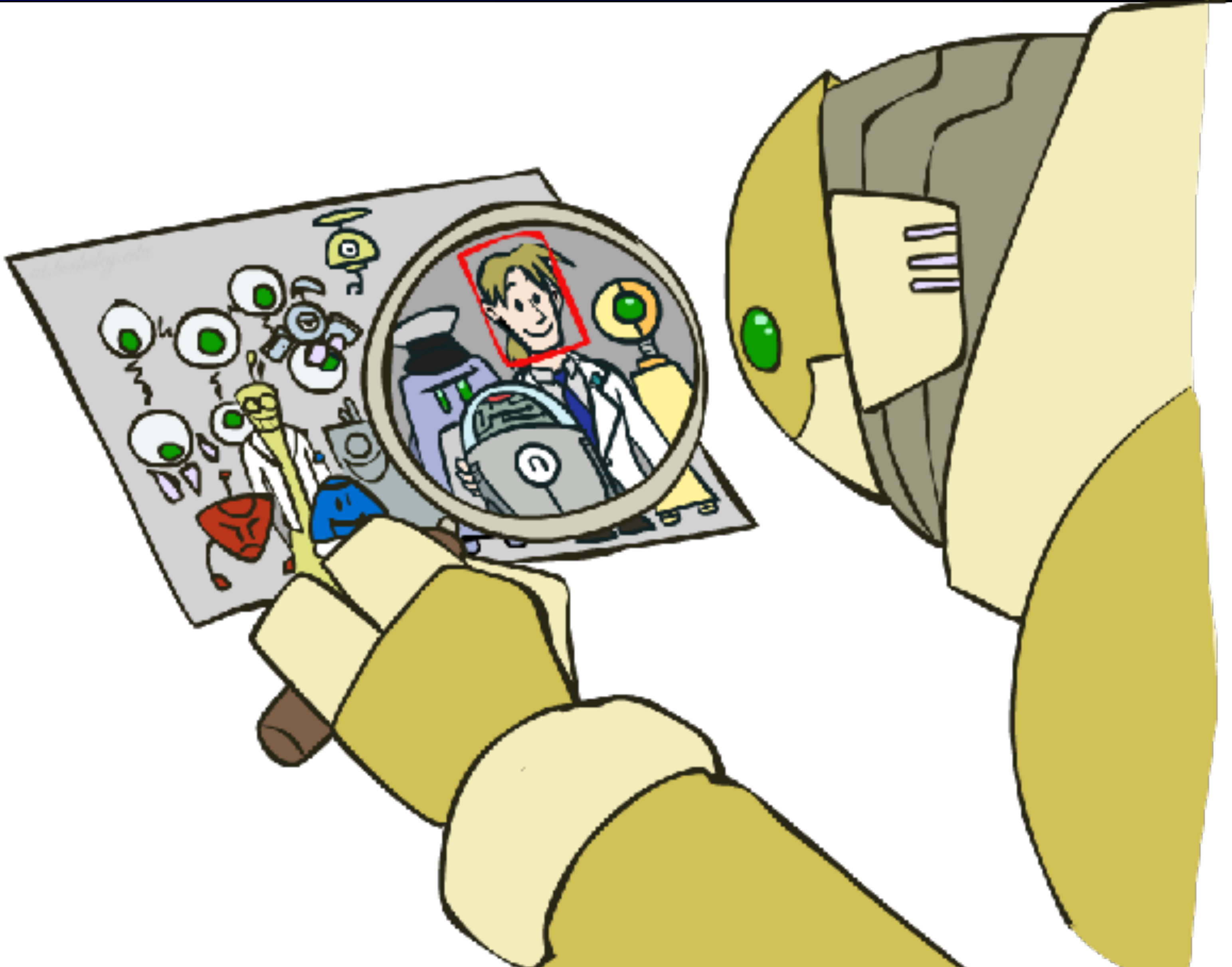
- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



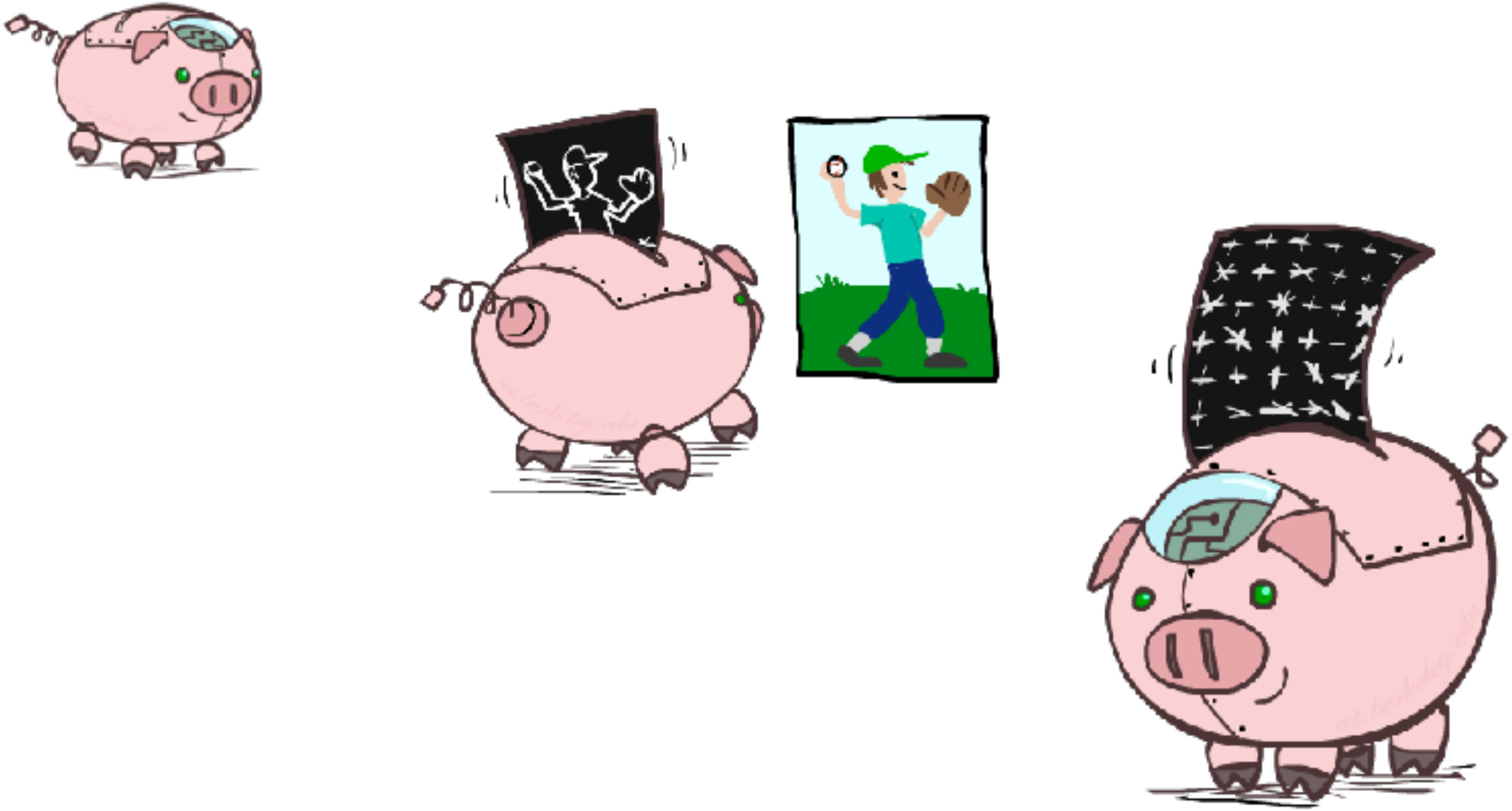
Computer Vision



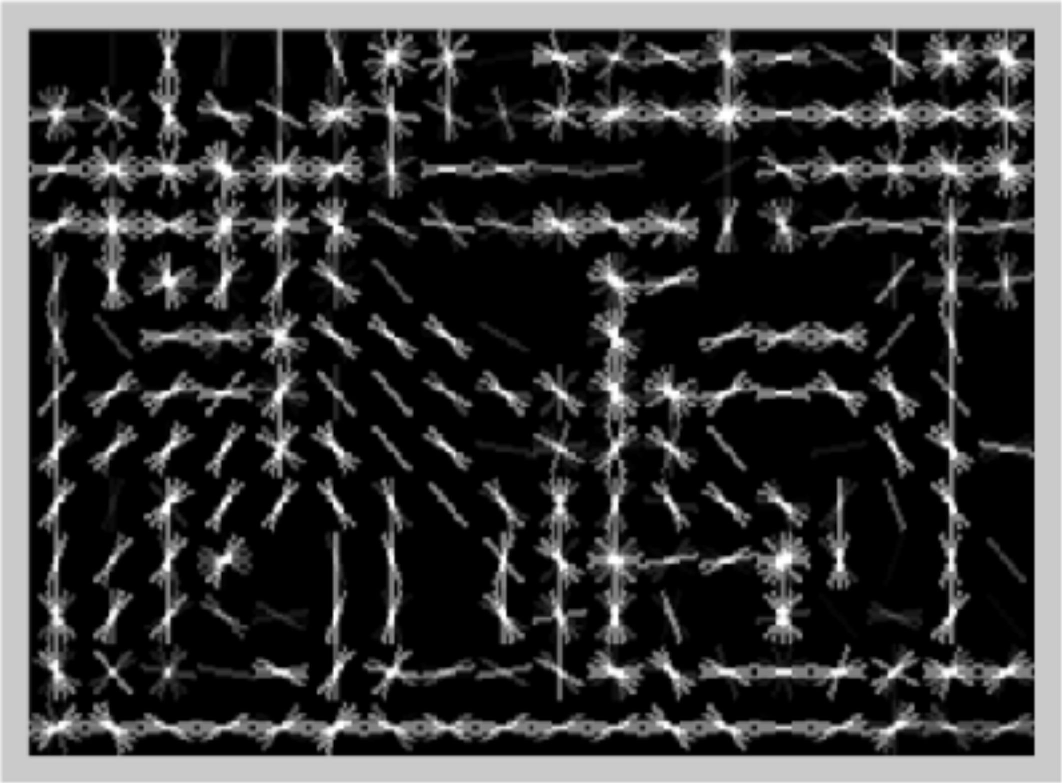
Object Detection



Manual Feature Design



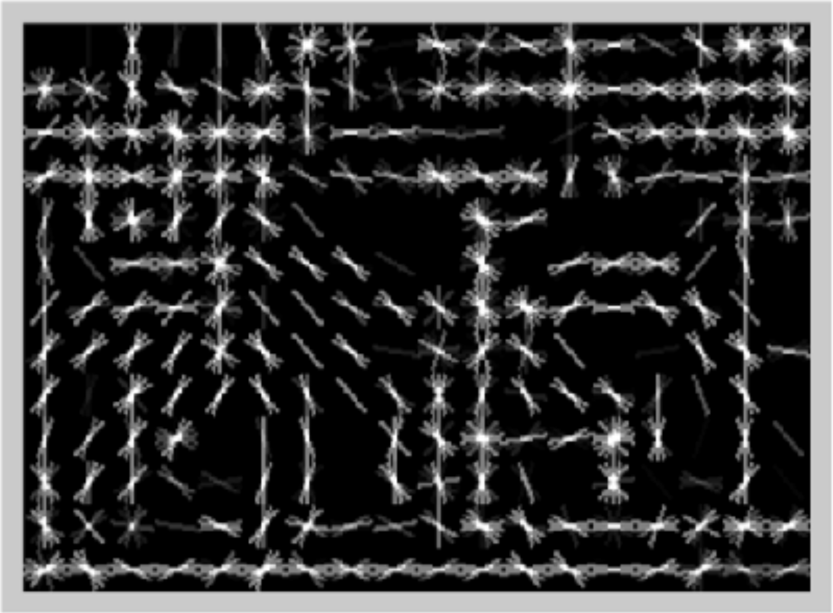
Features and Generalization



[Dalal and Triggs, 2005]

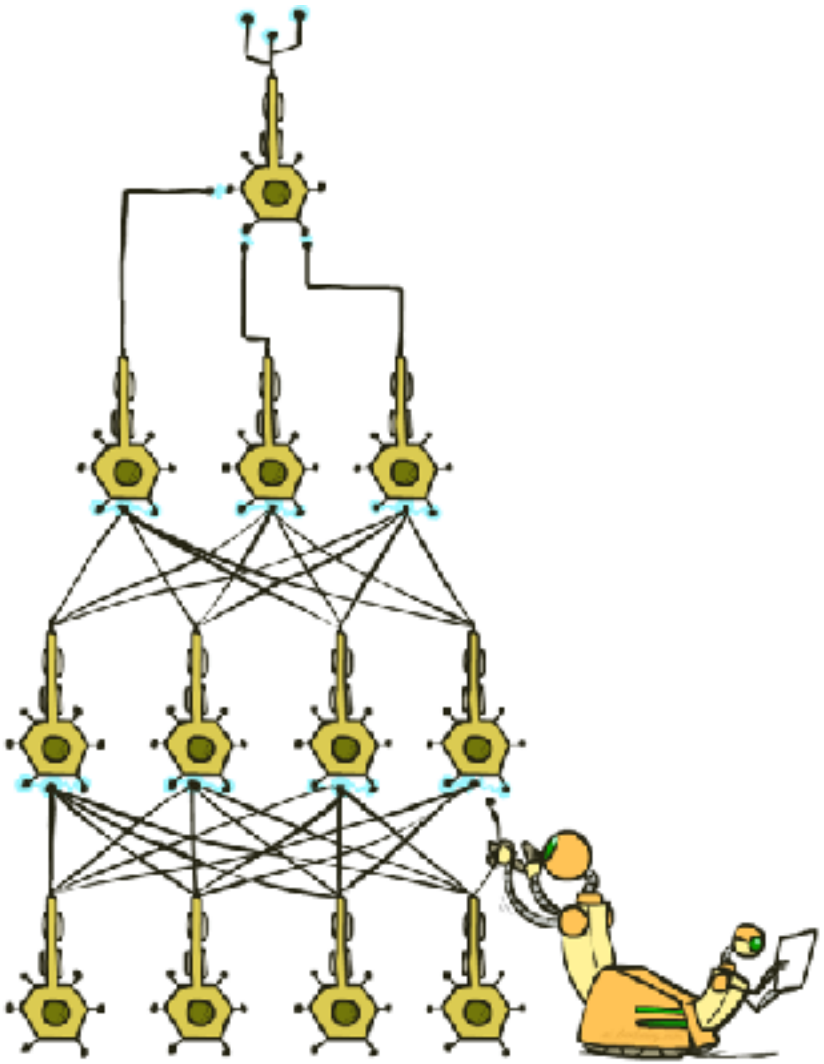
Features and Generalization

Image



HoG

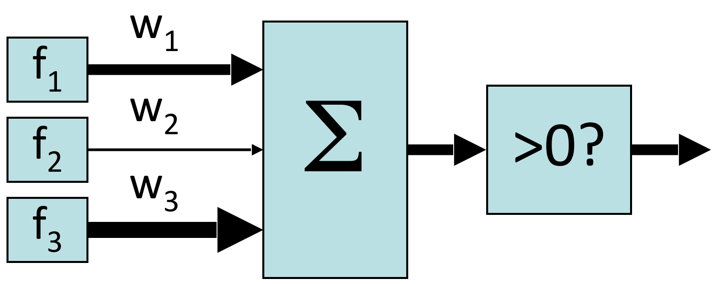
Manual Feature Design →Deep Learning



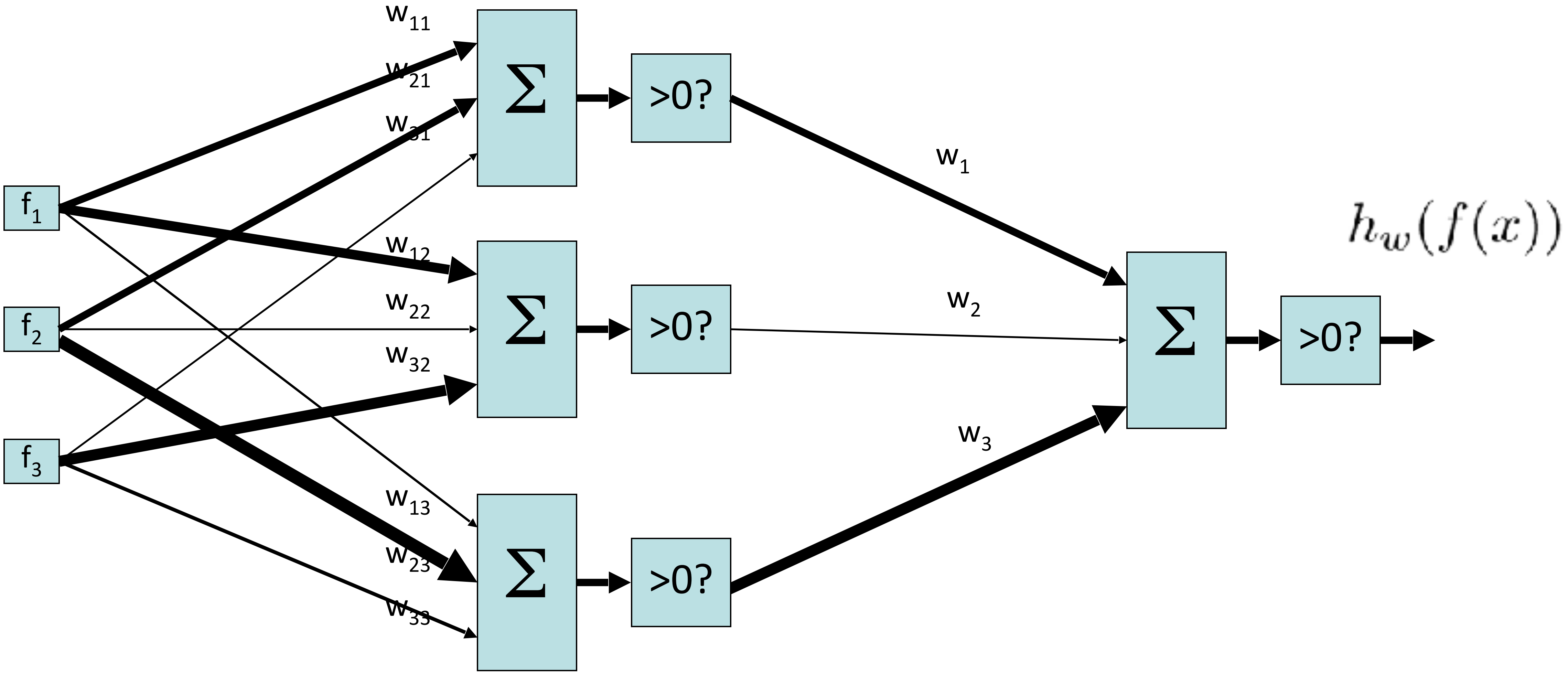
- Manual feature design requires:
  - Domain-specific expertise
  - Domain-specific effort

- What if we could learn the features, too?
  - **Deep Learning**

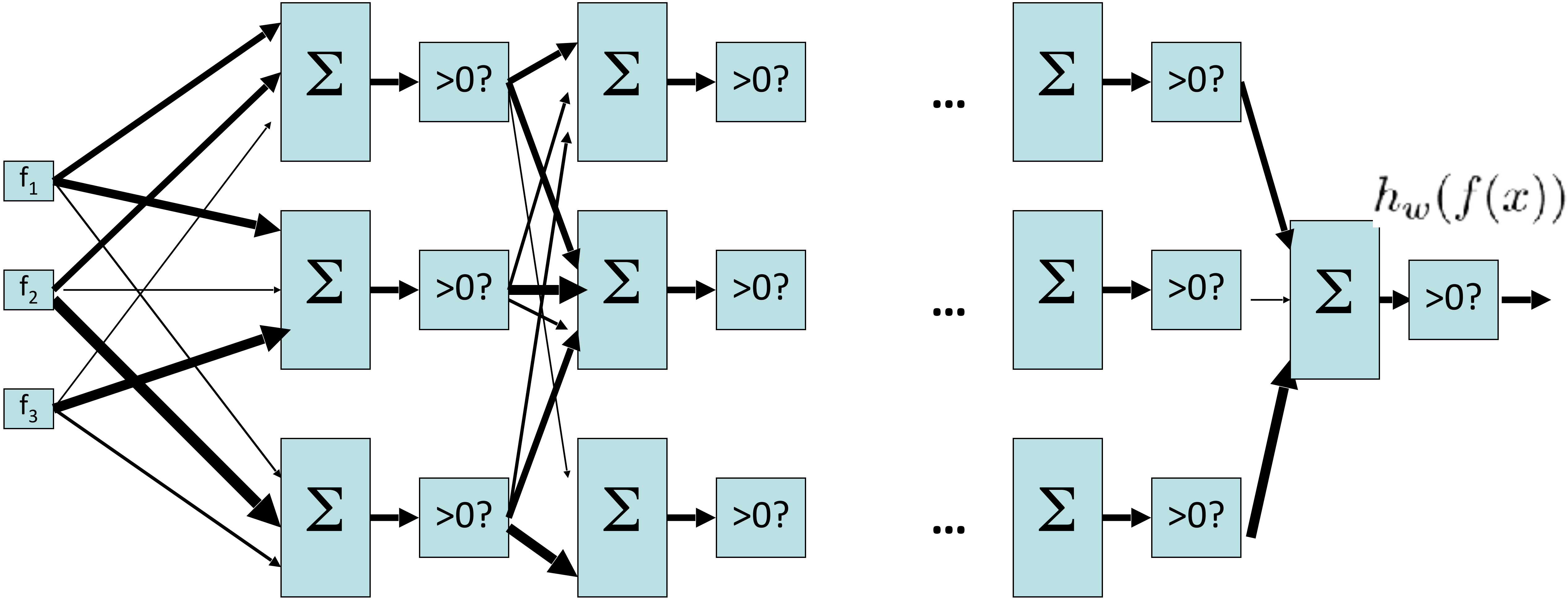
Perceptron



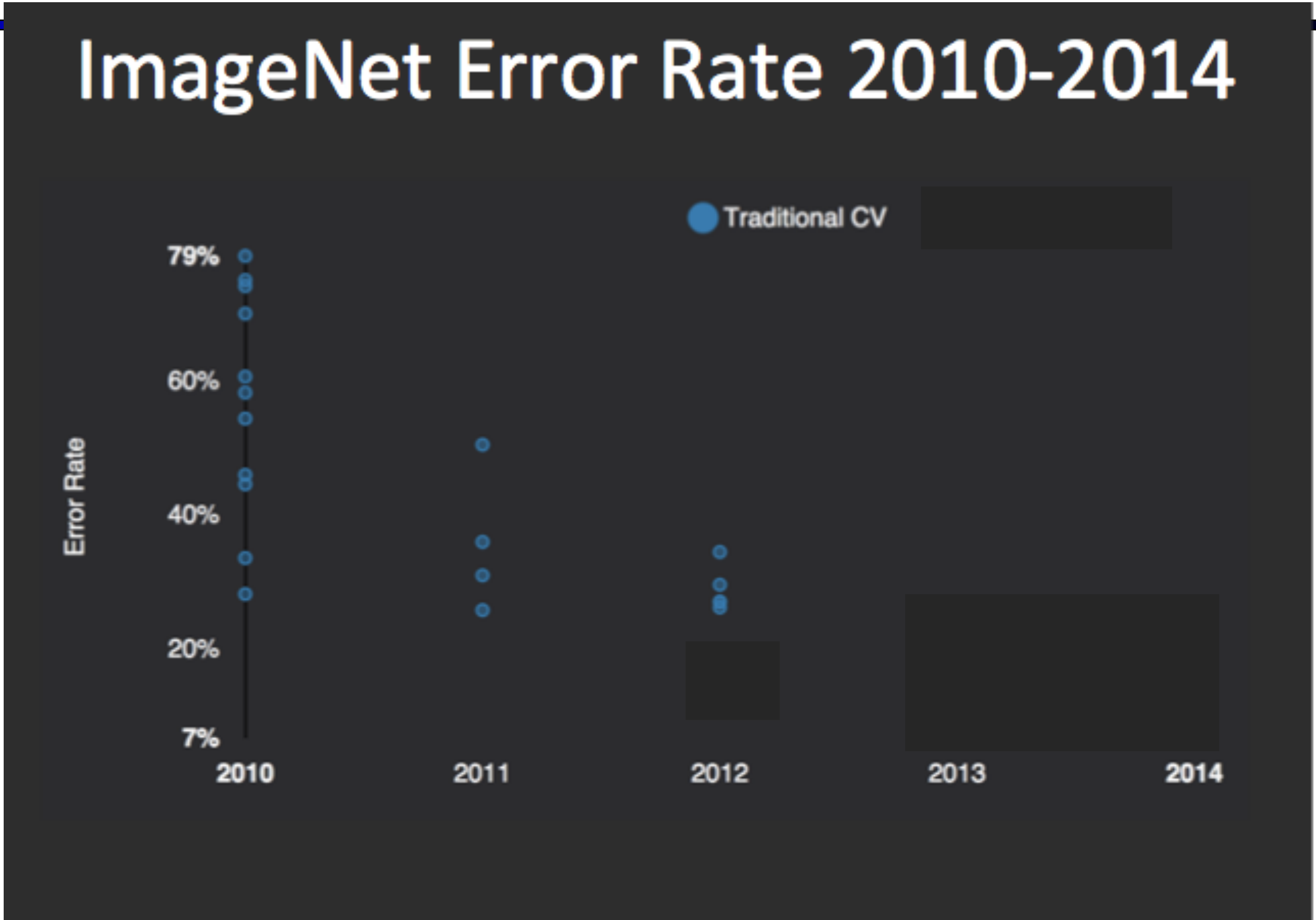
Two-Layer Perceptron Network



N-Layer Perceptron Network

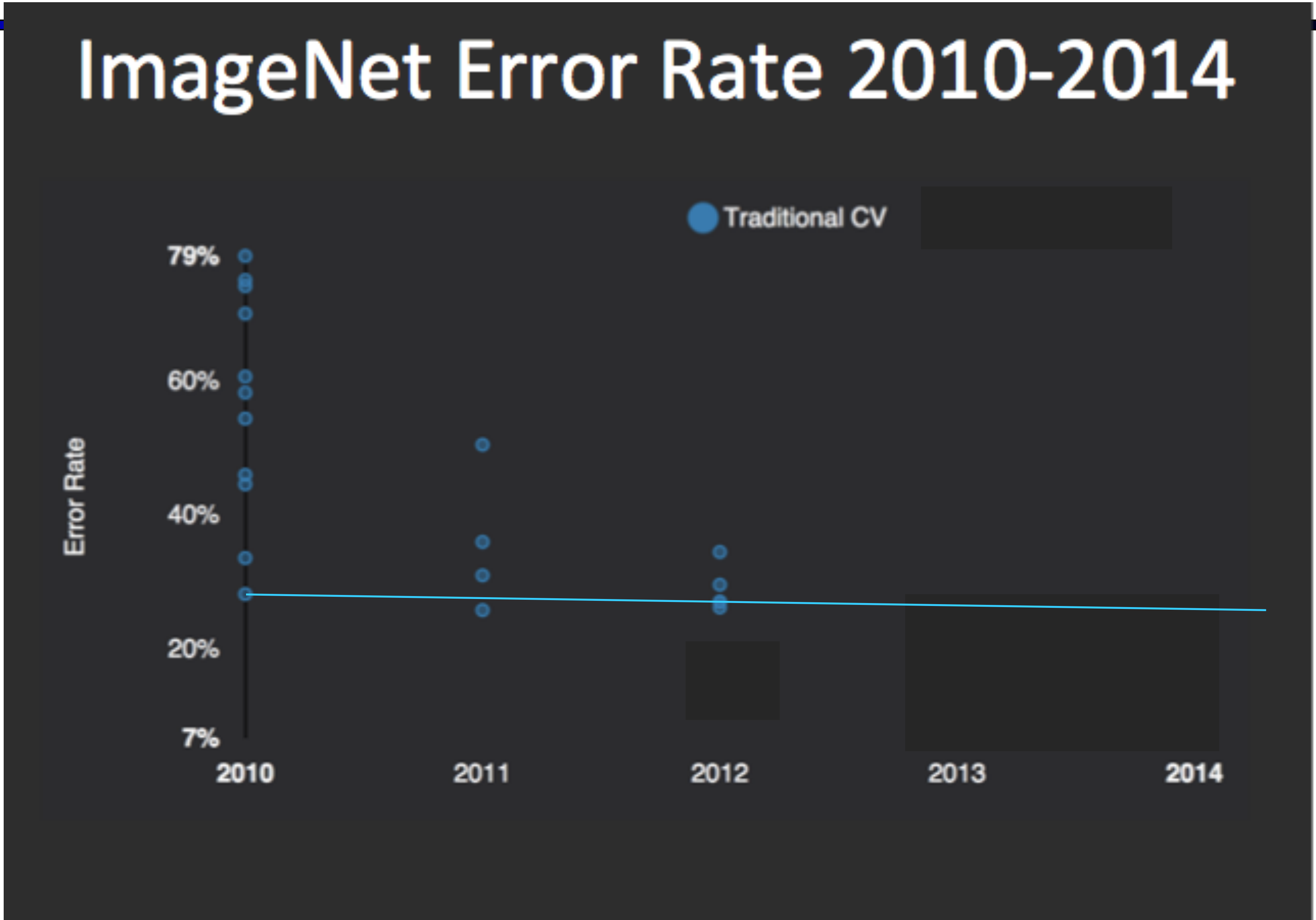


Performance



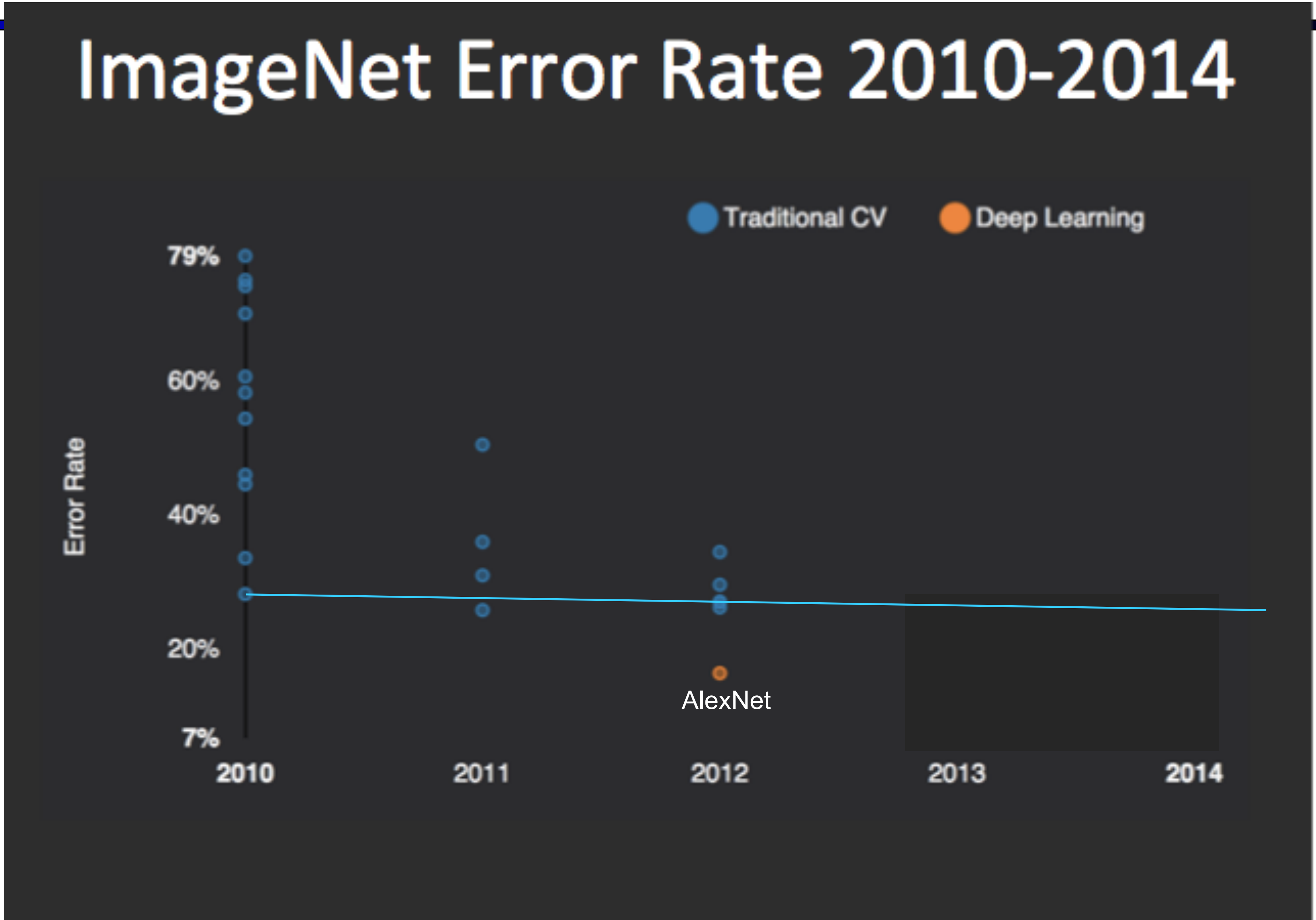
graph credit Matt Zeiler, Clarifai

Performance



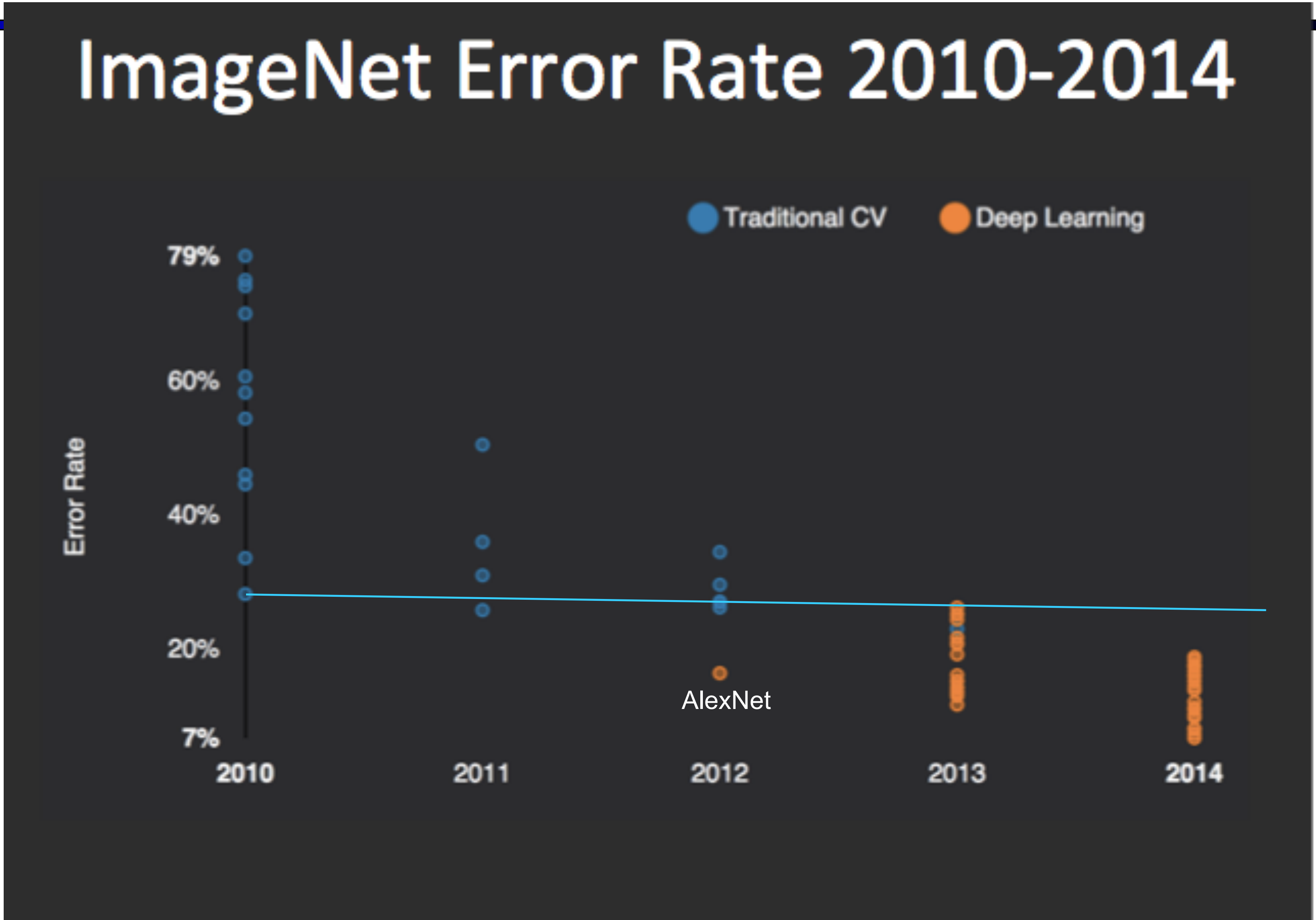
graph credit Matt Zeiler, Clarifai

Performance



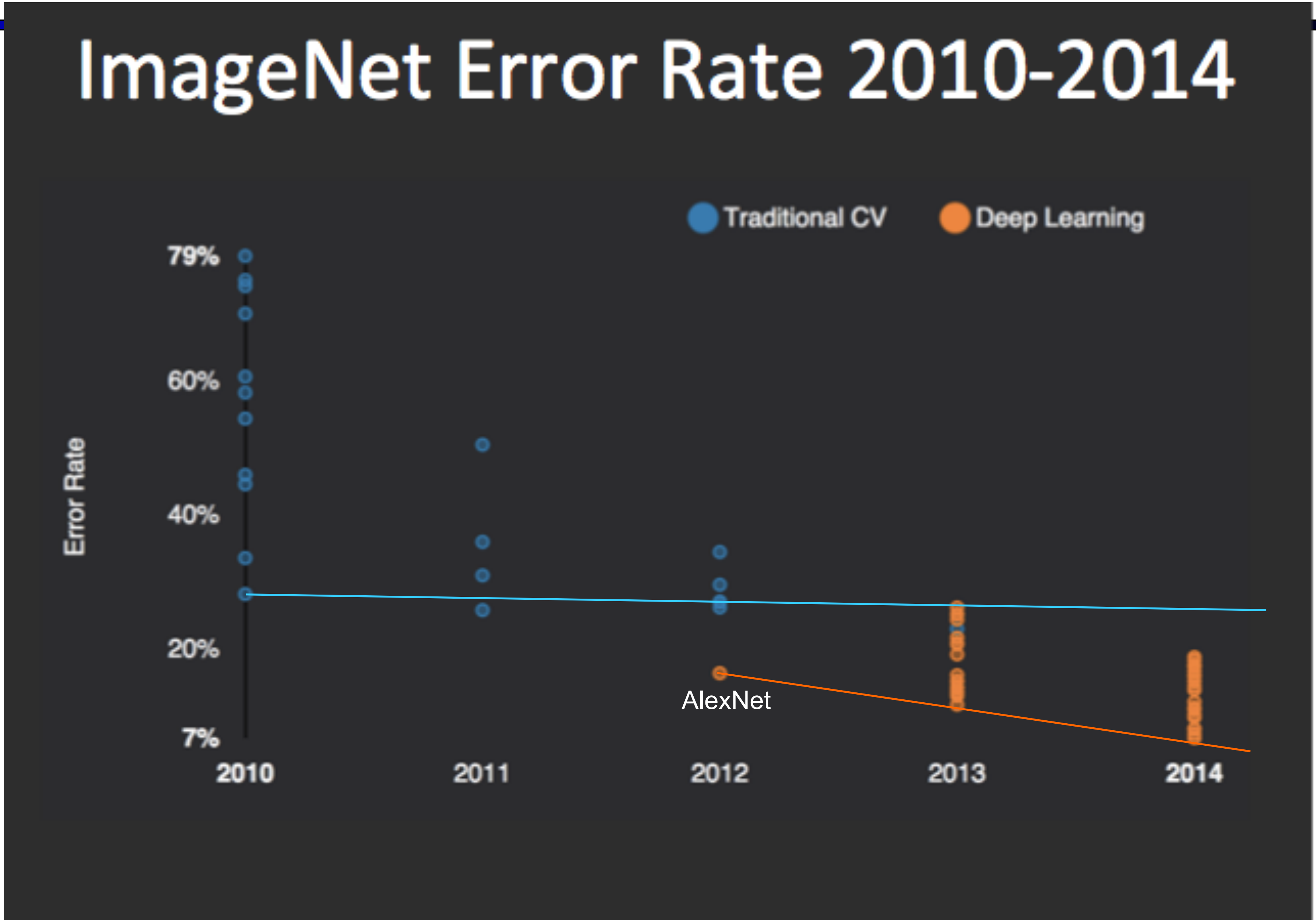
graph credit Matt Zeiler, Clarifai

Performance



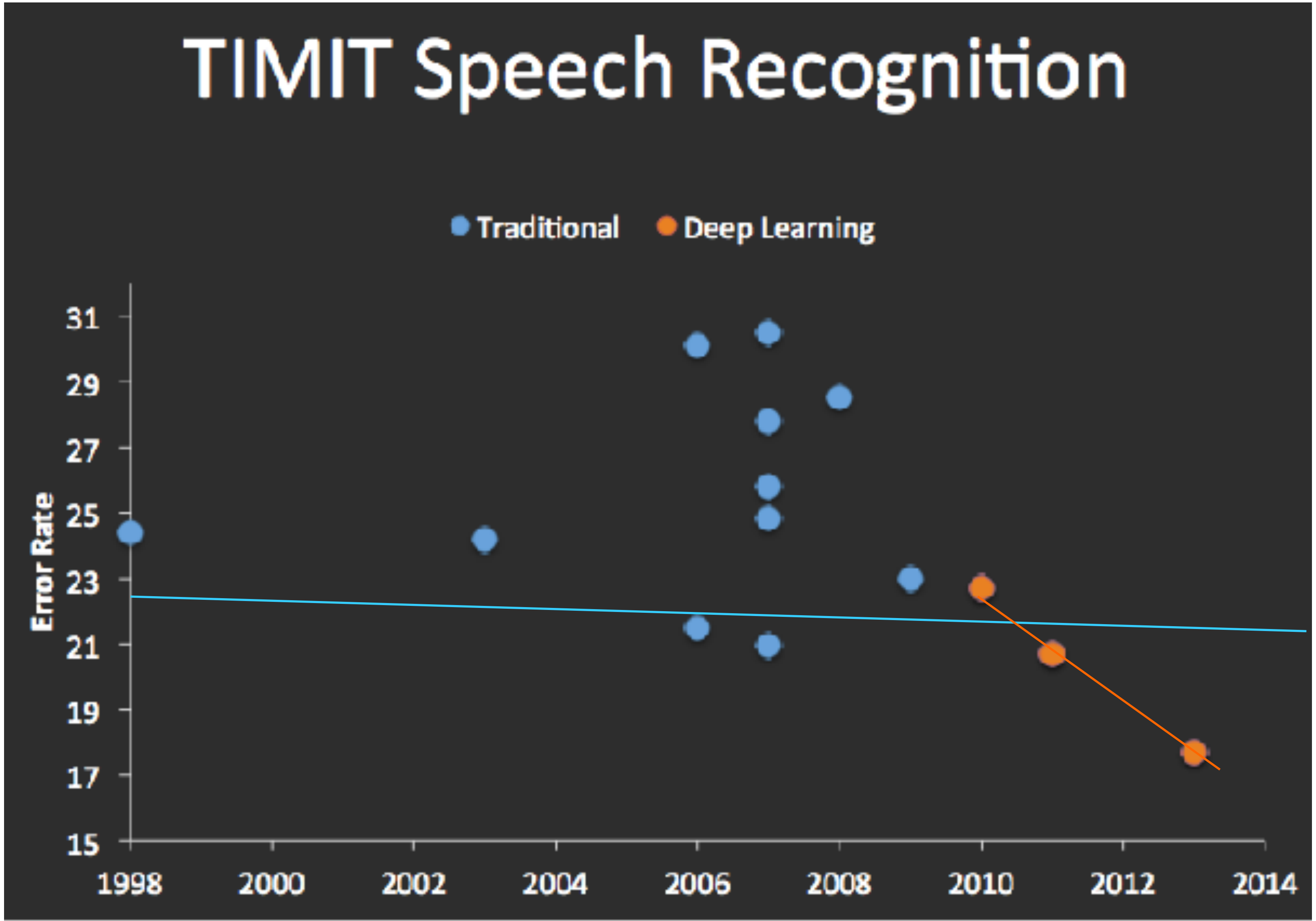
graph credit Matt Zeiler, Clarifai

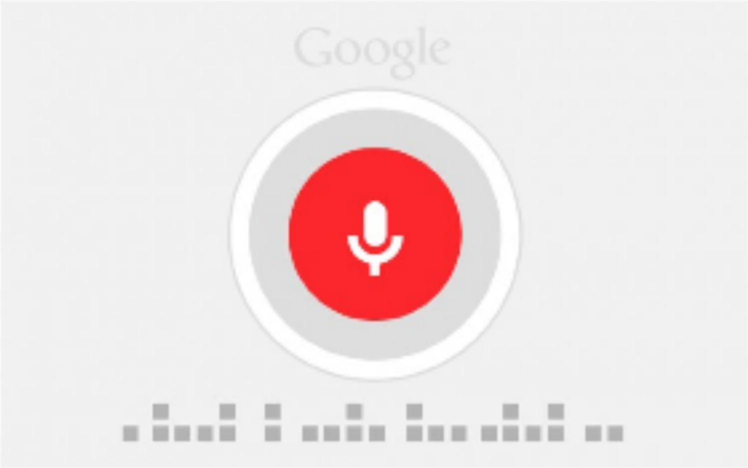
Performance



graph credit Matt Zeiler, Clarifai

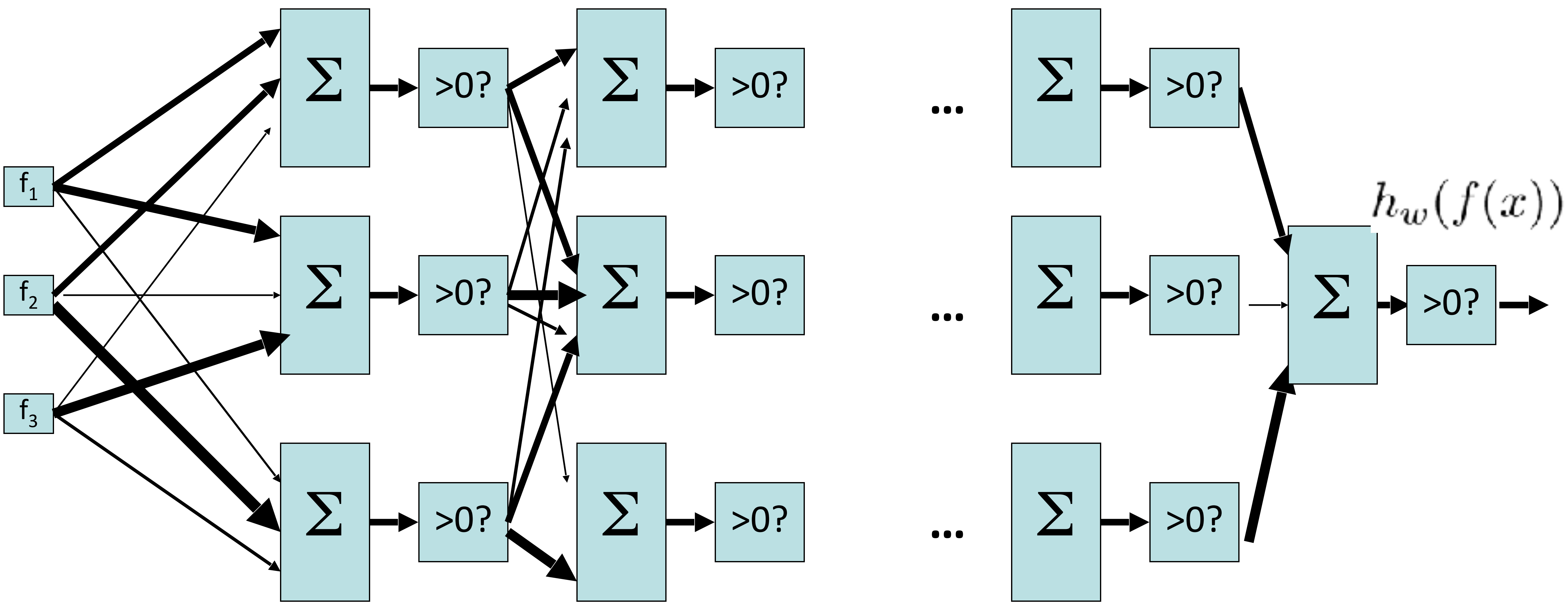
Speech Recognition





graph credit Matt Zeiler, Clarifai

N-Layer Perceptron Network



Local Search

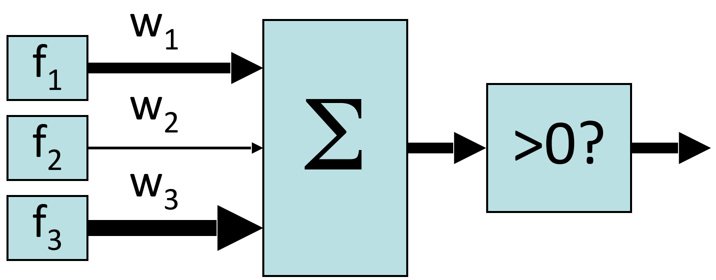
- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
  - Neighbors = small perturbations of w

- Properties
  - Plateaus and local optima

How to escape plateaus and find a good local optimum?

How to deal with very large parameter vectors? E.g., $w \in \mathbb{R}^{1billion}$

Perceptron



- Objective: Classification Accuracy

$$l^{\text{acc}}(w) = \frac{1}{m}\sum_{i=1}^{m}\left(\text{sign}(w^\top f(x^{(i)})) == y^{(i)}\right)$$

  - Issue: many plateaus → how to measure incremental progress toward a correct label?

Soft-Max

- Score for y=1: $w^\top f(x)$ Score for y=-1: $-w^\top f(x)$

- Probability of label:

$$p(y=1|f(x);w) = \frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

$$p(y=-1|f(x);w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

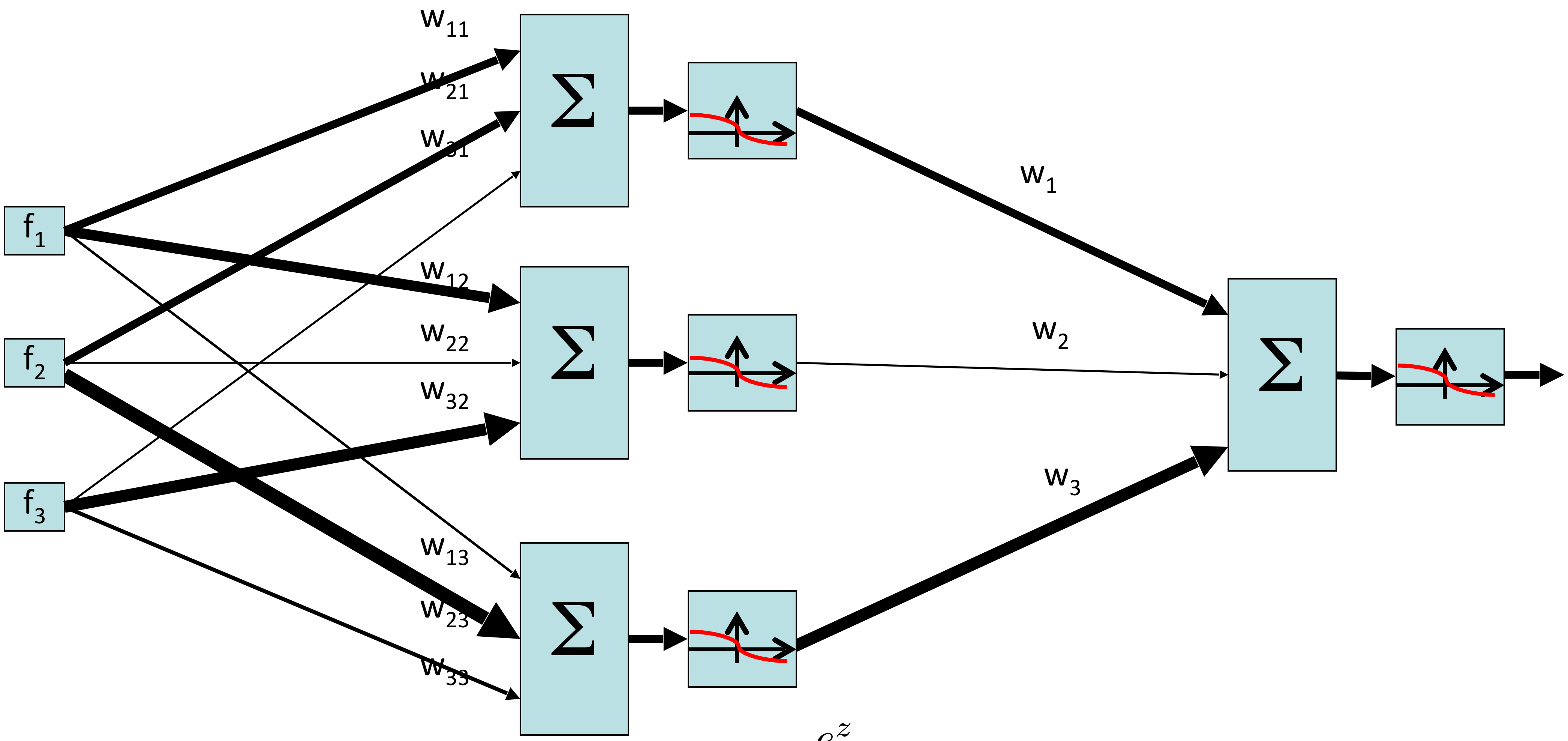
- Objective:

$$l(w) = \prod_{i=1}^{m} p(y = y^{(i)}|f(x^{(i)});w)$$

- Log:

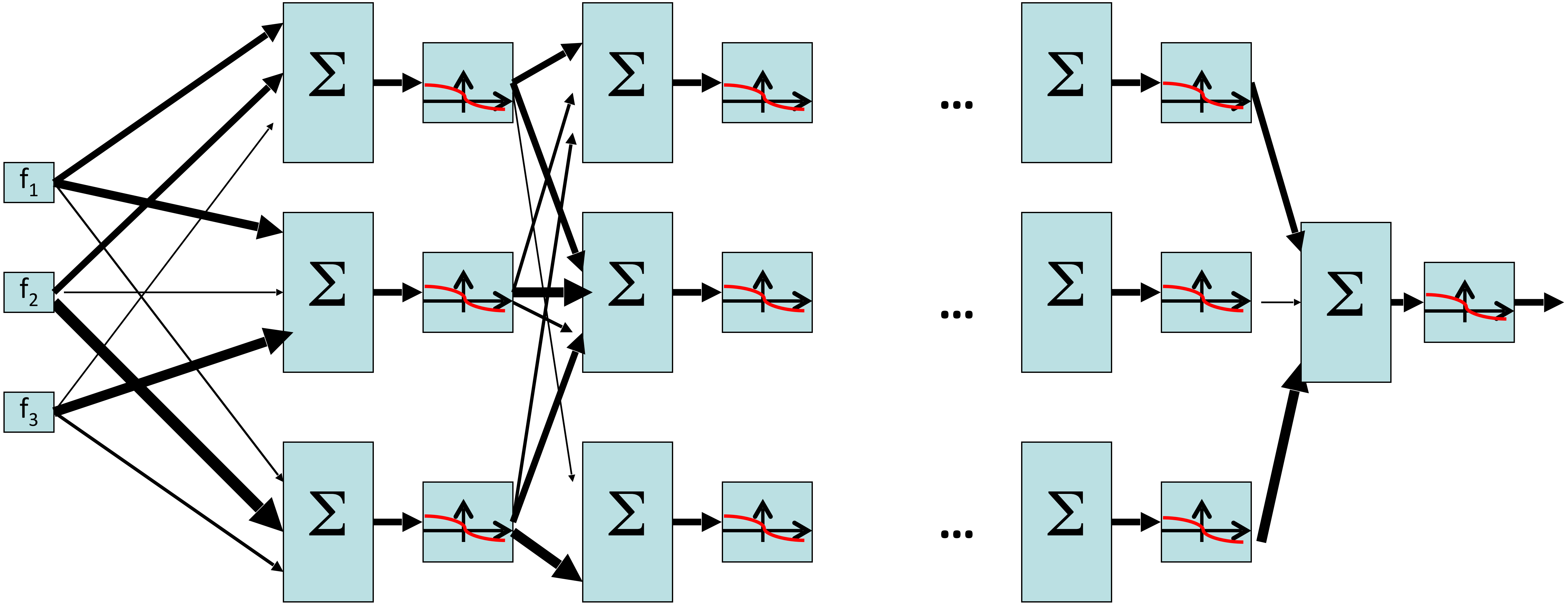
$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)}|f(x^{(i)});w)$$

Two-Layer Neural Network



$$z \rightarrow \frac{e^z}{e^z + e^{-z}}$$

N-Layer Neural Network



Our Status

- Our objective $ll(w)$
  - Changes smoothly with changes in w
  - Doesn't suffer from the same plateaus as the perceptron network

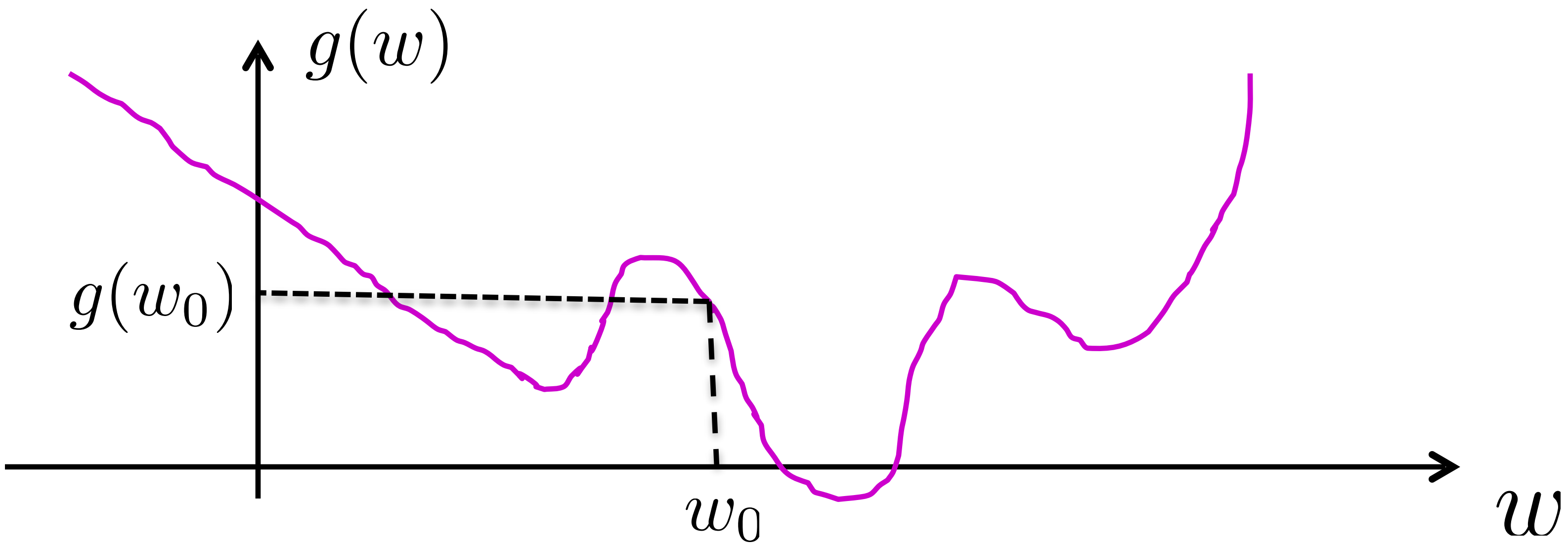
- Challenge: how to find a good w ?

$$\max_w ll(w)$$

- Equivalently:

$$\min_w -ll(w)$$

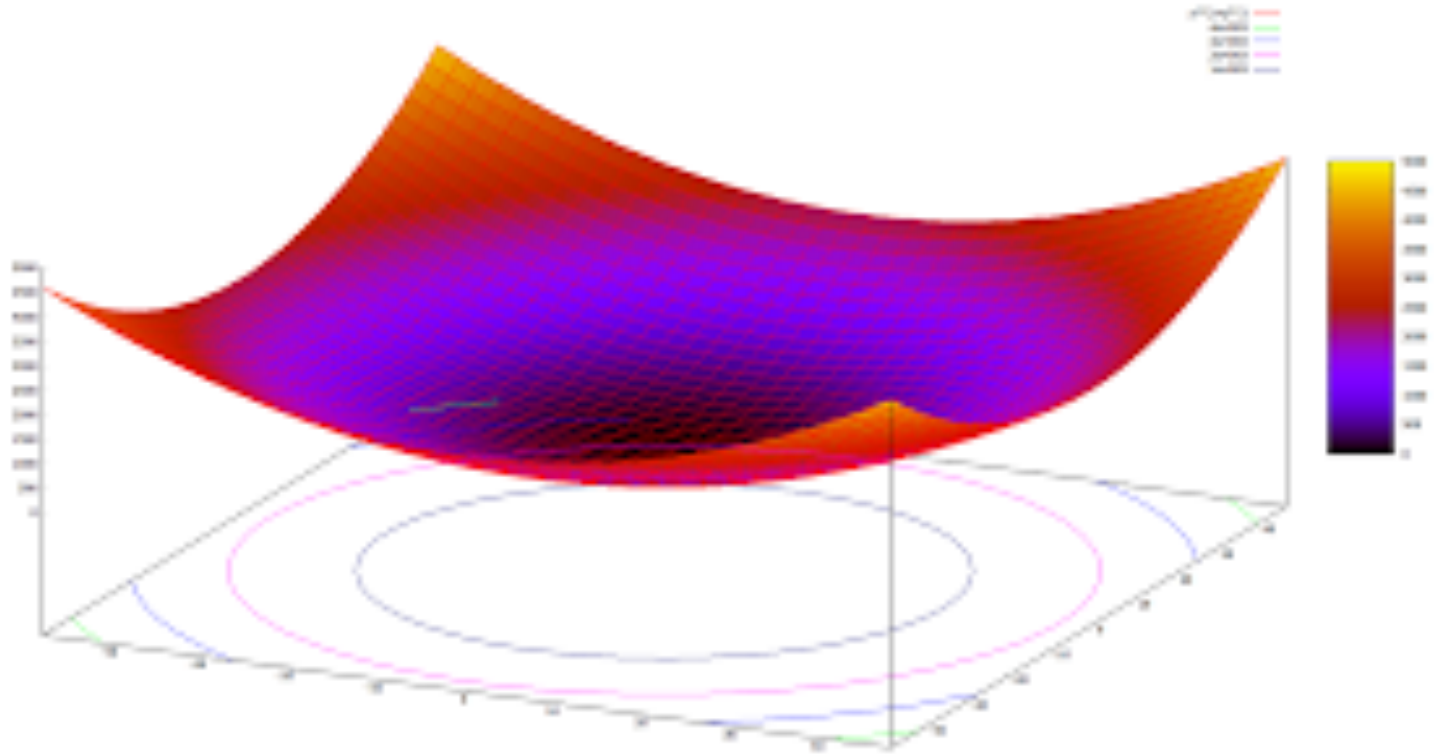
1-d optimization



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

- Or, evaluate derivative: $\dfrac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \dfrac{g(w_0 + h) - g(w_0 - h)}{2h}$
  - Tells which direction to step in

2-D Optimization



Source: Thomas Jungblut's Blog

Steepest Descent

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

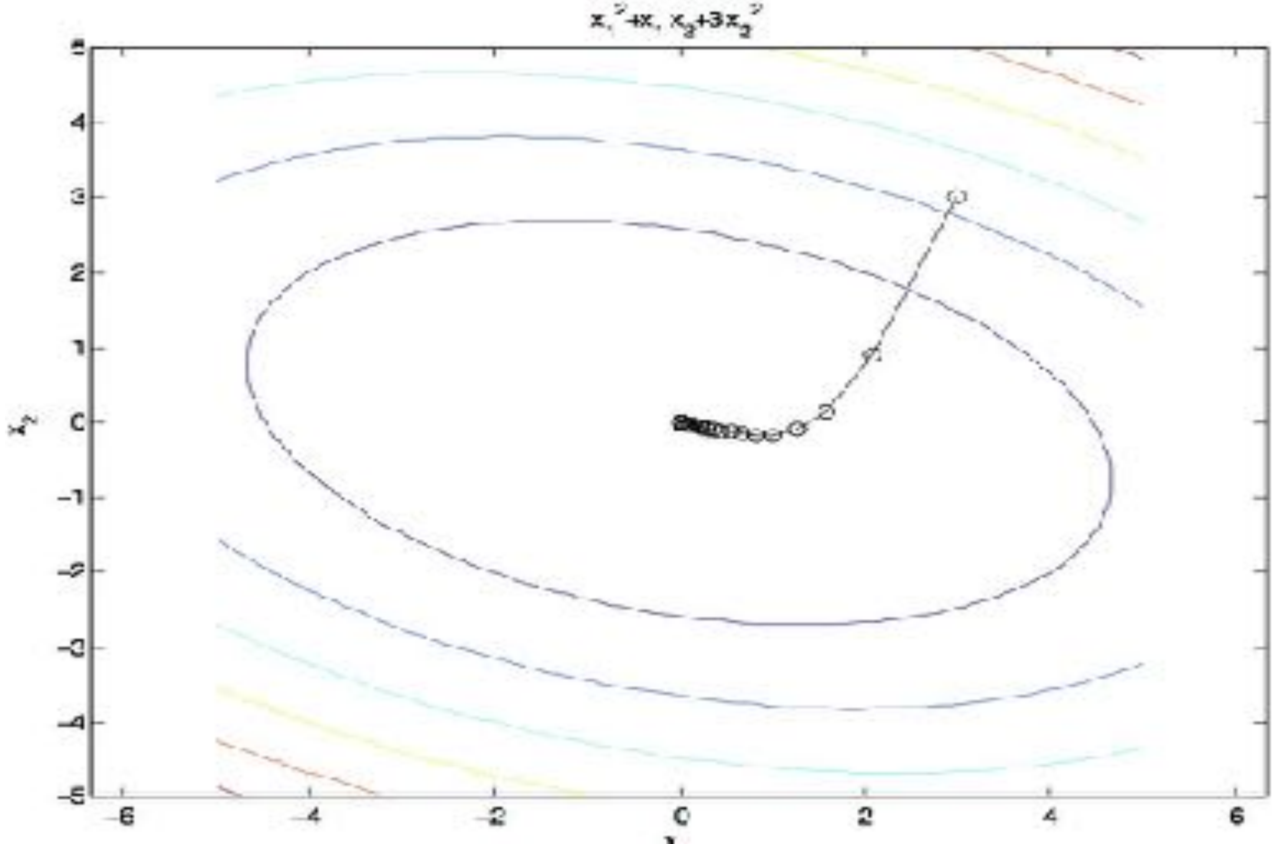


Figure source: Mathworks

What is the Steepest Descent Direction?

What is the Steepest Descent Direction?

- Steepest Direction = direction of the gradient

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

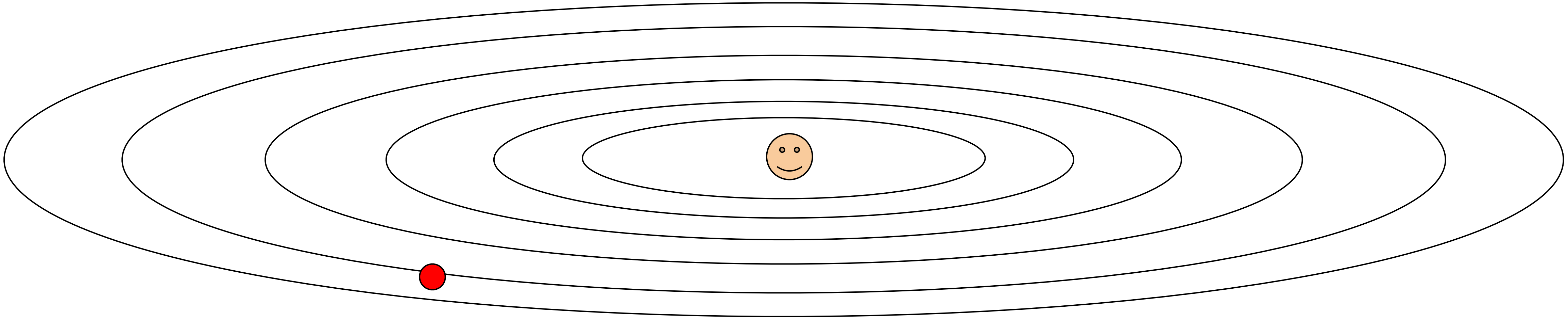
Optimization Procedure 1: Gradient Descent

- Init: w
- For i = 1, 2, …

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

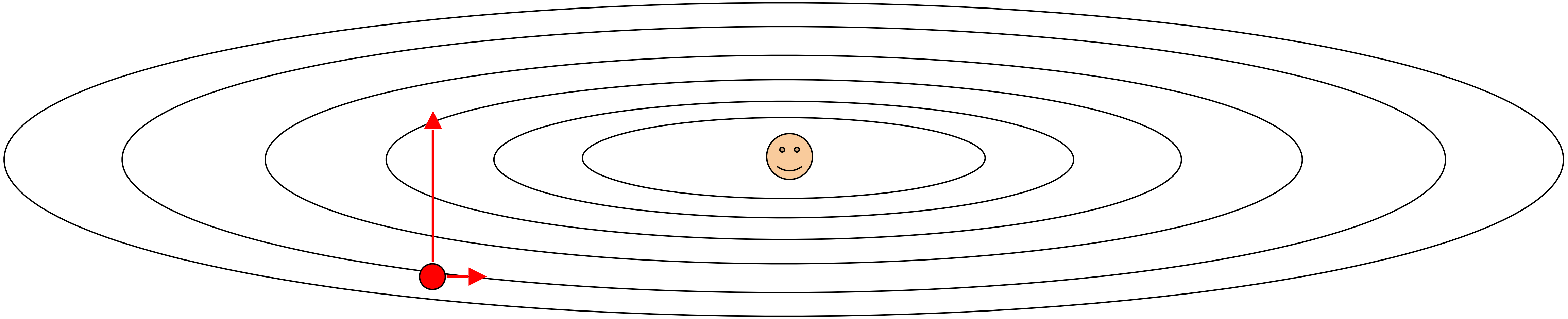
- α: learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes w about 0.1 – 1 %

Suppose loss function is steep vertically but shallow horizontally:



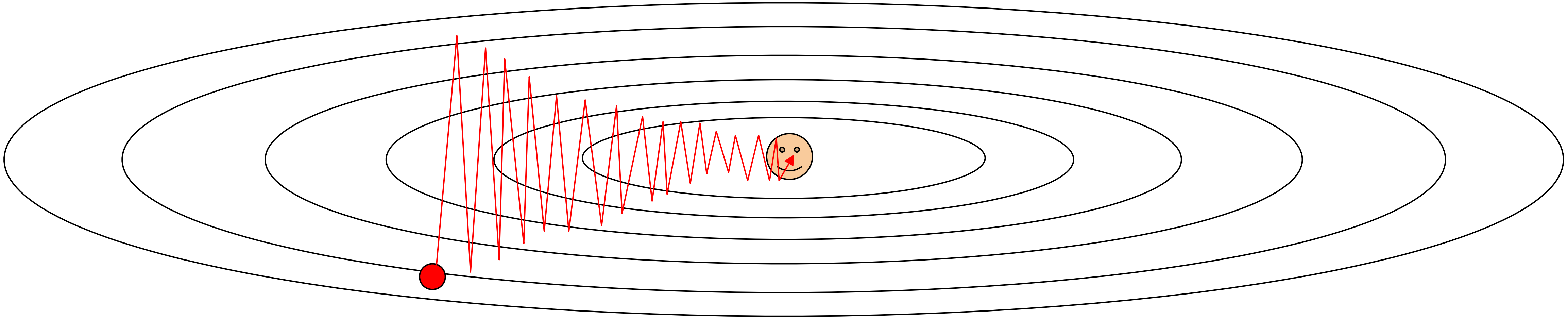
Q: What is the trajectory along which we converge towards the minimum with Gradient Descent?

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with Gradient Descent?

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with Gradient Descent? very slow progress along flat direction, jitter along steep one

Optimization Procedure 2: Momentum

- Gradient Descent

  - Init: w
  - For i = 1, 2, …

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

- Momentum

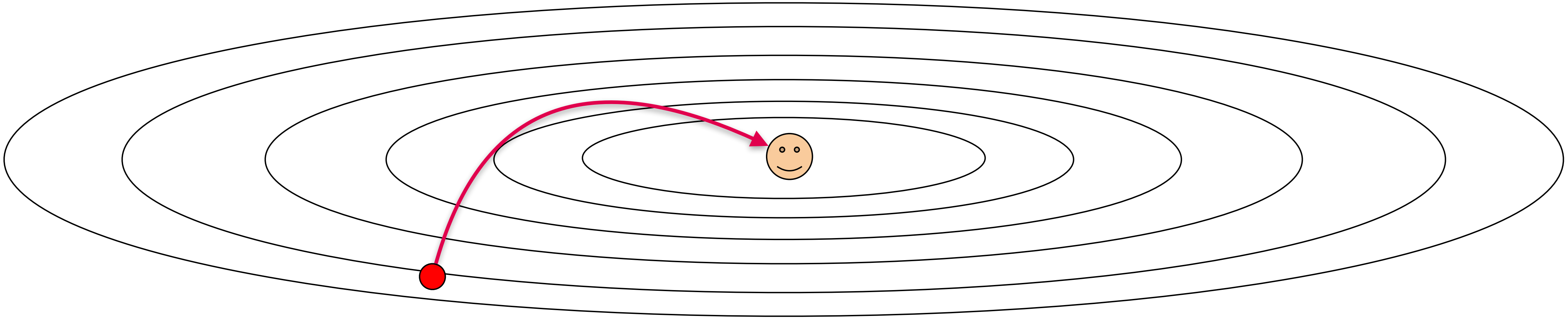
  - Init: w
  - For i = 1, 2, …

$$v \leftarrow \mu * v - \alpha * \nabla g(w)$$

$$w \leftarrow w + v$$

- Physical interpretation as ball rolling down the loss function + friction (mu coefficient).
- mu = usually ~0.5, 0.9, or 0.99 (Sometimes annealed over time, e.g. from 0.5 -> 0.99)

Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with Momentum?

How do we actually compute gradient w.r.t. weights?

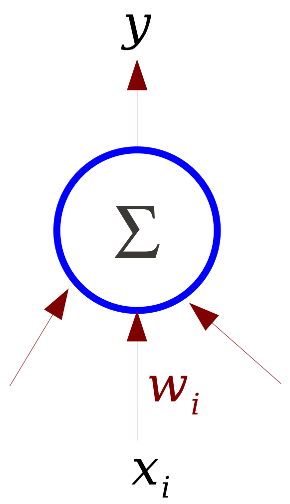
Backpropagation!

Backpropagation Learning

15-486/782: Artificial Neural Networks
David S. Touretzky

Fall 2006

LMS / Widrow-Hoff Rule

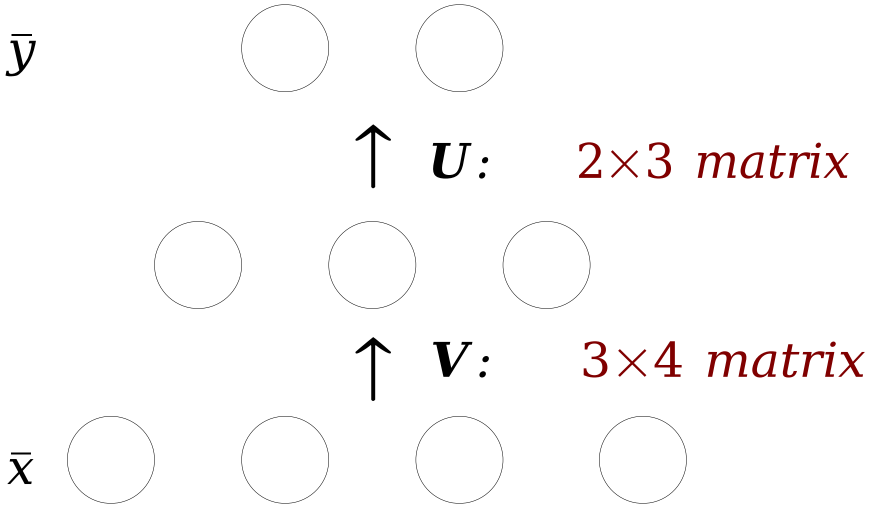


$$\Delta w_i = -\eta(y-d)x_i$$

Works fine for a single layer of trainable weights.

What about multi-layer networks?

With Linear Units, Multiple Layers Don't Add Anything



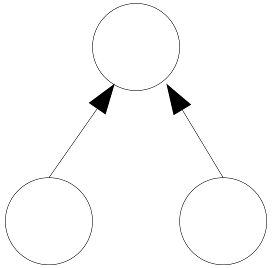
$$\bar{y} = \boldsymbol{U}\times(\boldsymbol{V}\bar{x}) = \underbrace{(\boldsymbol{U}\times\boldsymbol{V})}_{2\times 4}\bar{x}$$

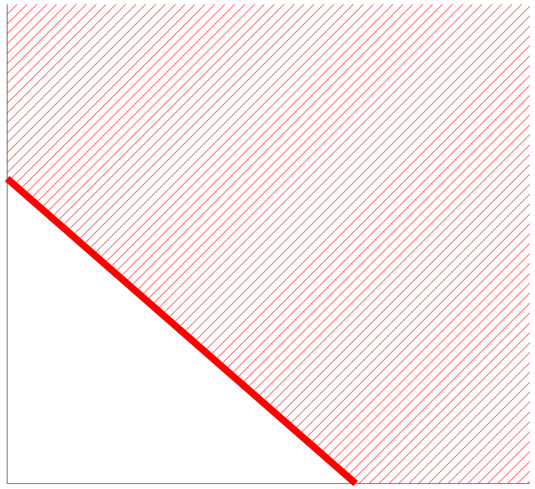
Linear operators are closed under composition.
Equivalent to a single layer of weights $\boldsymbol{W}=\boldsymbol{U}\times\boldsymbol{V}$

But with non-linear units, extra layers add computational power.

What Can be Done with Non-Linear (e.g., Threshold) Units?

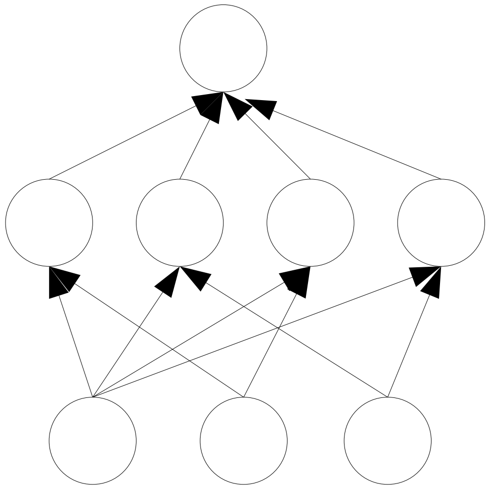
1 layer of trainable weights

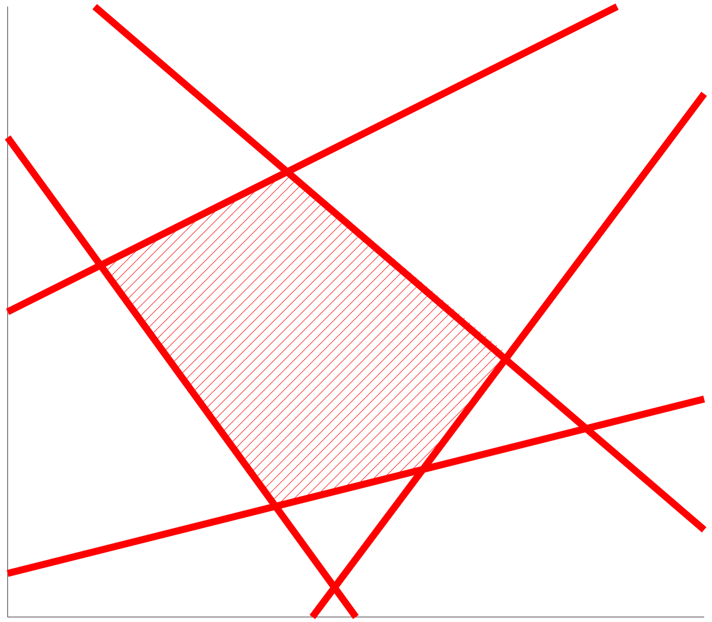




separating hyperplane

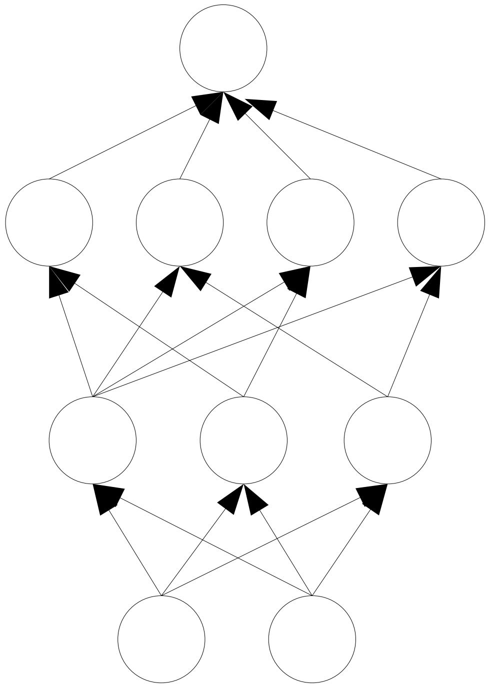
2 layers of trainable weights

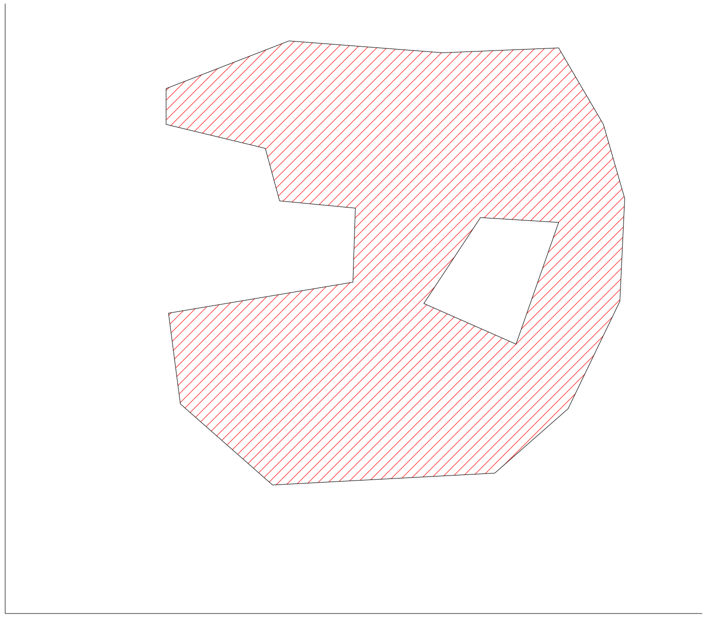




convex polygon region

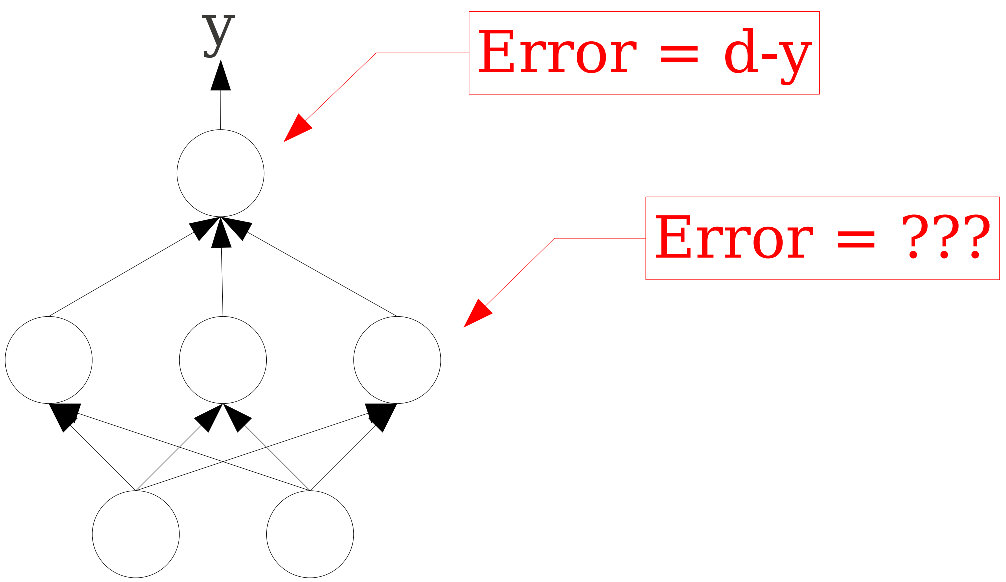
3 layers of trainable weights





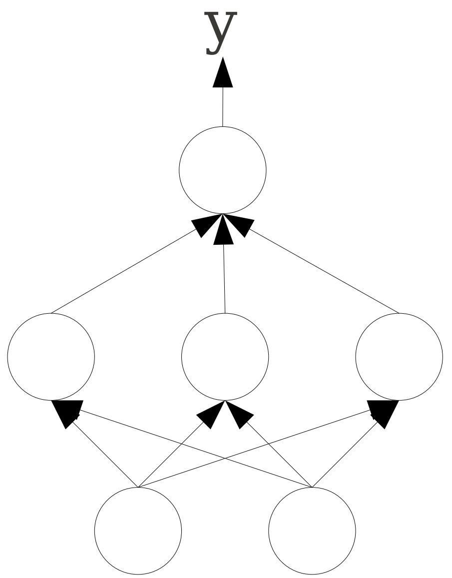
composition of polygons:
non convex regions

How Do We Train A Multi-Layer Network?



Can't use perceptron training algorithm because
we don't know the 'correct' outputs for hidden units.

How Do We Train A Multi-Layer Network?



Define sum-squared error:

$$E = \frac{1}{2}\sum_{p}(d^p - y^p)^2$$

Use gradient descent error minimization:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

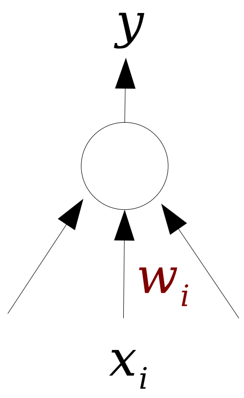
Works if the nonlinear transfer function is differentiable.

Deriving the LMS or "Delta" Rule As Gradient Descent Learning

$$y = \sum_i w_i x_i$$

$$E = \frac{1}{2}\sum_p (d^p - y^p)^2$$

$$\frac{dE}{dy} = y - d$$



$$\frac{\partial E}{\partial w_i} = \frac{dE}{dy} \cdot \frac{\partial y}{\partial w_i} = (y - d) x_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta (y - d) x_i$$

How do we extend this to two layers?

Switch to Smooth Nonlinear Units

$$\text{net}_j = \sum_i w_{ij} y_i$$

$$y_j = \text{g}(\text{net}_j)$$ *g must be differentiable*

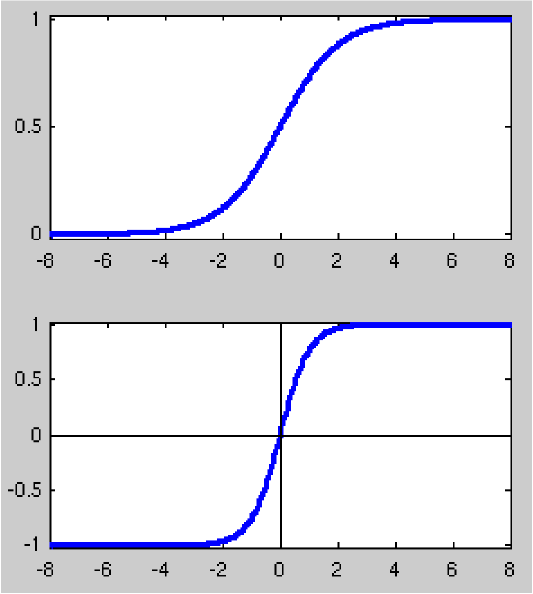
Common choices for g:

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

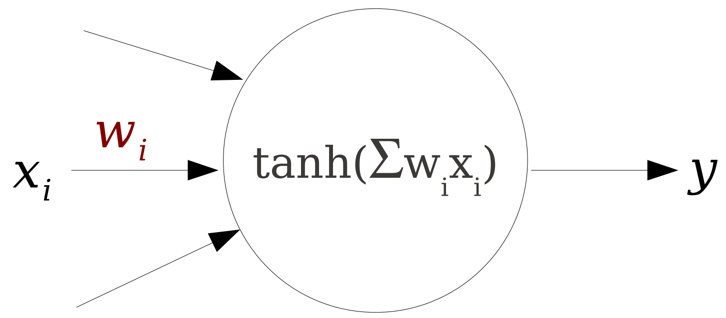
$$g'(x) = g(x)\cdot(1-g(x))$$

$$g(x)=\tanh(x)$$

$$g'(x)=1/\cosh^2(x)$$



Gradient Descent with Nonlinear Units

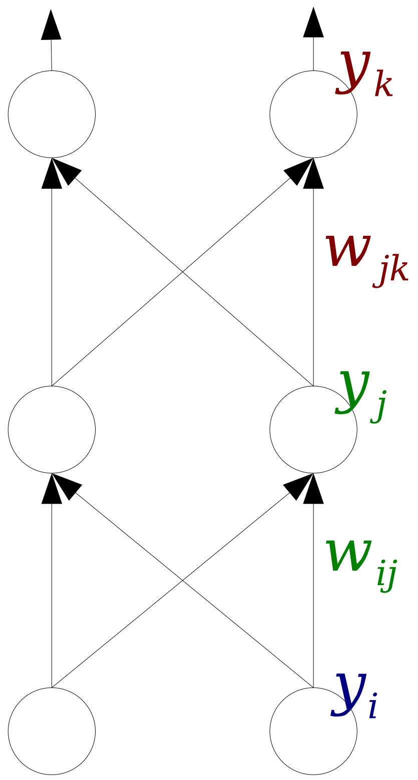


$$y = g(net) = \tanh\left(\sum_i w_i x_i\right)$$

$$\frac{dE}{dy} = (y - d), \qquad \frac{dy}{d\,net} = 1/\cosh^2(net), \qquad \frac{\partial\,net}{\partial w_i} = x_i$$

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{dE}{dy} \cdot \frac{dy}{d\,net} \cdot \frac{\partial\,net}{\partial w_i} \\ &= (y-d)/\cosh^2\left(\sum_i w_i x_i\right) \cdot x_i\end{aligned}$$

Now We Can Use The Chain Rule



$$\frac{\partial E}{\partial y_k} = (y_k - d_k)$$

$$\delta_k = \frac{\partial E}{\partial net_k} = (y_k - d_k) \cdot g'(net_k)$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}} = \delta_k \cdot y_j$$

$$\frac{\partial E}{\partial y_j} = \sum_k \left(\frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \right)$$

$$\delta_j = \frac{\partial E}{\partial net_j} = \frac{\partial E}{\partial y_j} \cdot g'(net_j)$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j \cdot y_i$$

Weight Updates

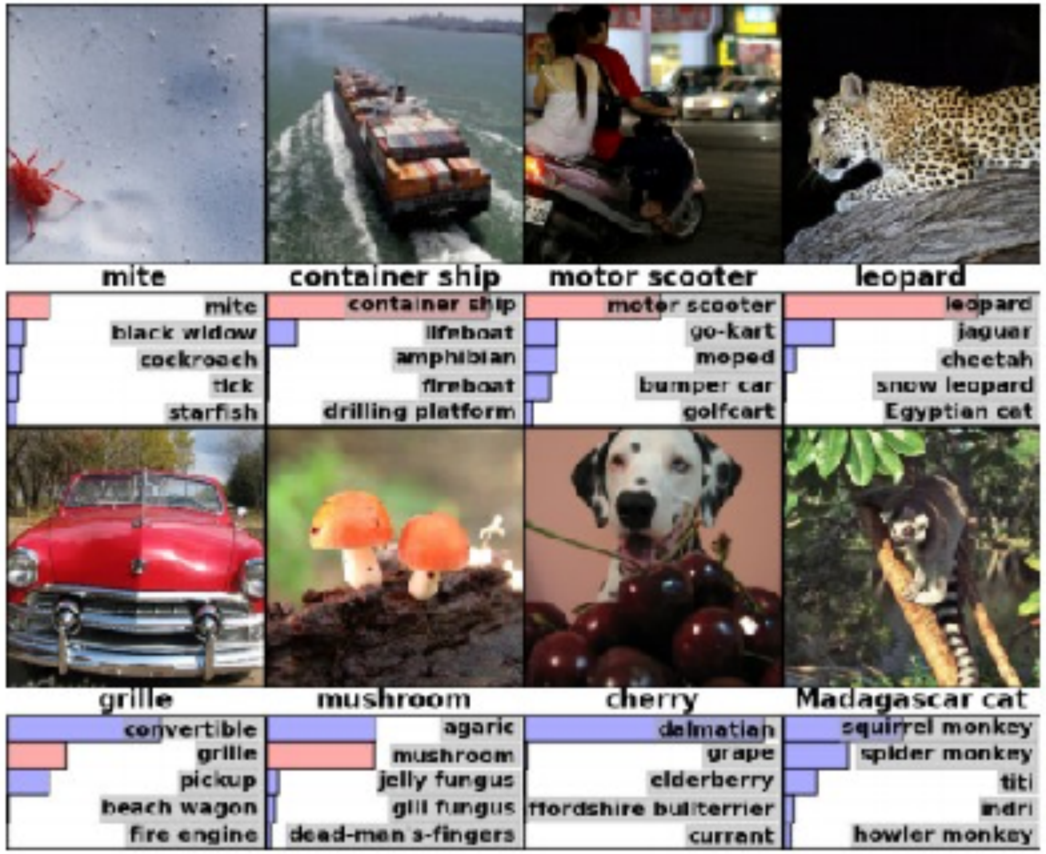
$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{jk}} = \delta_k \cdot y_j$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} = \delta_j \cdot y_i$$

$$\Delta w_{jk} = -\eta \cdot \frac{\partial E}{\partial w_{jk}} \qquad \Delta w_{ij} = -\eta \cdot \frac{\partial E}{\partial w_{ij}}$$

Deep learning is everywhere

Classification

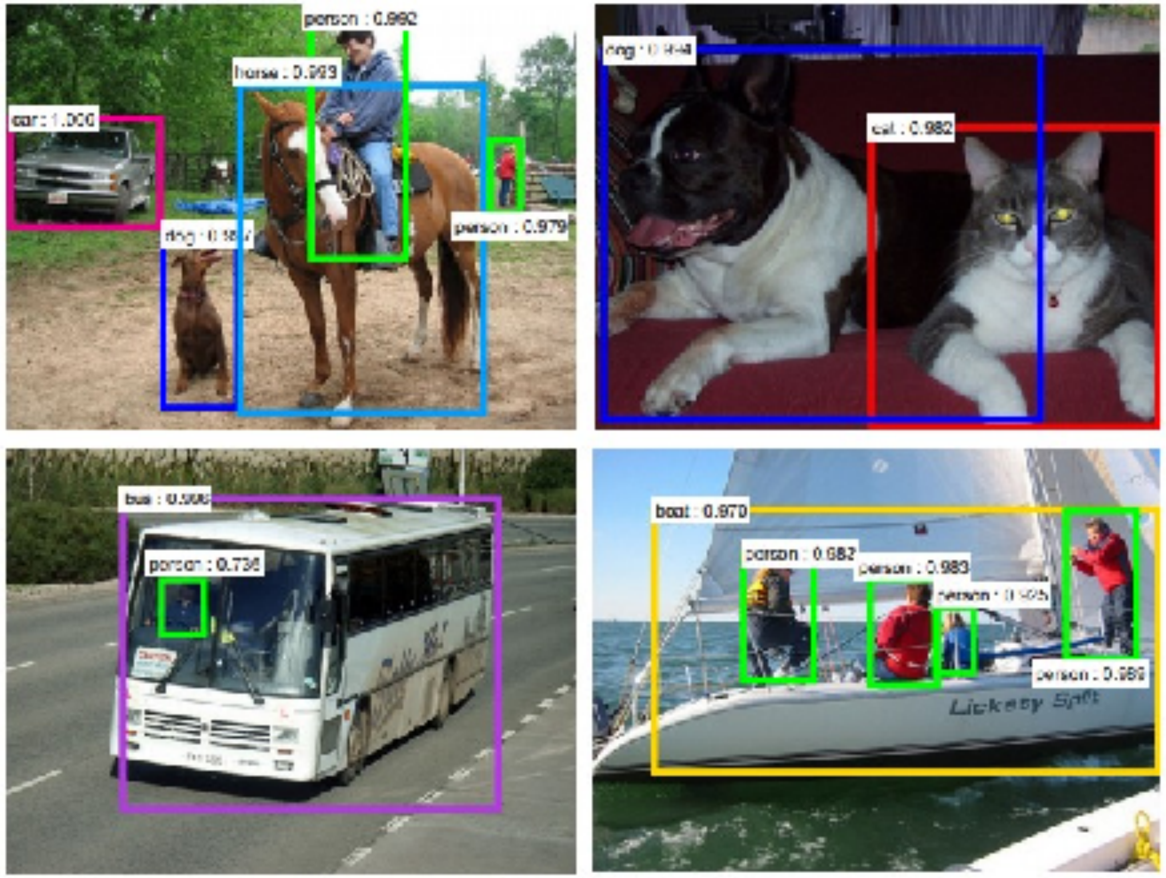


Retrieval

[Krizhevsky 2012]

Deep learning is everywhere

Detection



Segmentation

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

[Farabet et al., 2012]

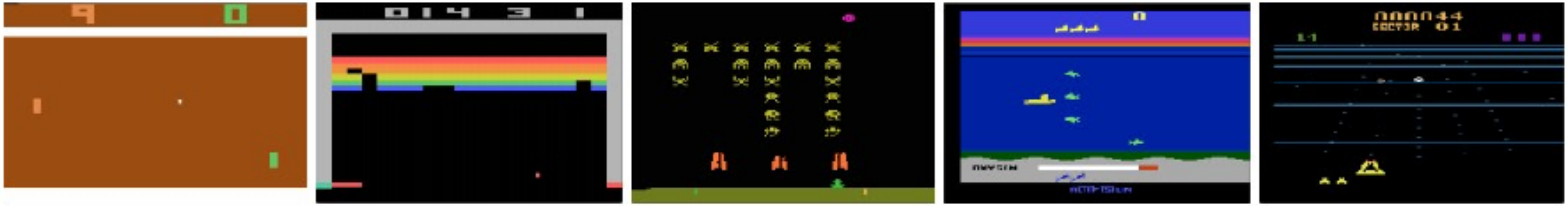
Deep learning is everywhere

self-driving cars

NVIDIA Tegra X1

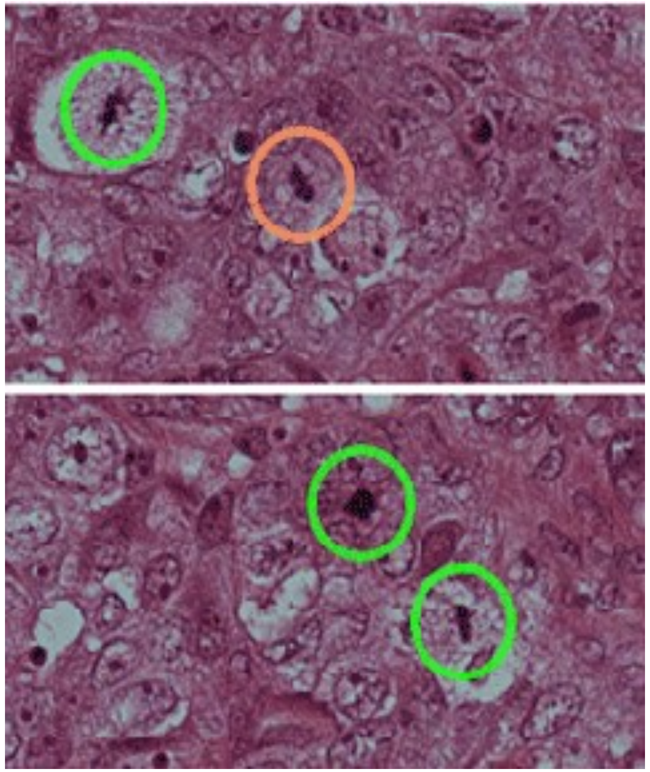
Deep learning is everywhere

[Toshev, Szegedy 2014]



[Mnih 2013]

Deep learning is everywhere



[Ciresan et al. 2013]

[Sermanet et al. 2011]
[Ciresan et al.]

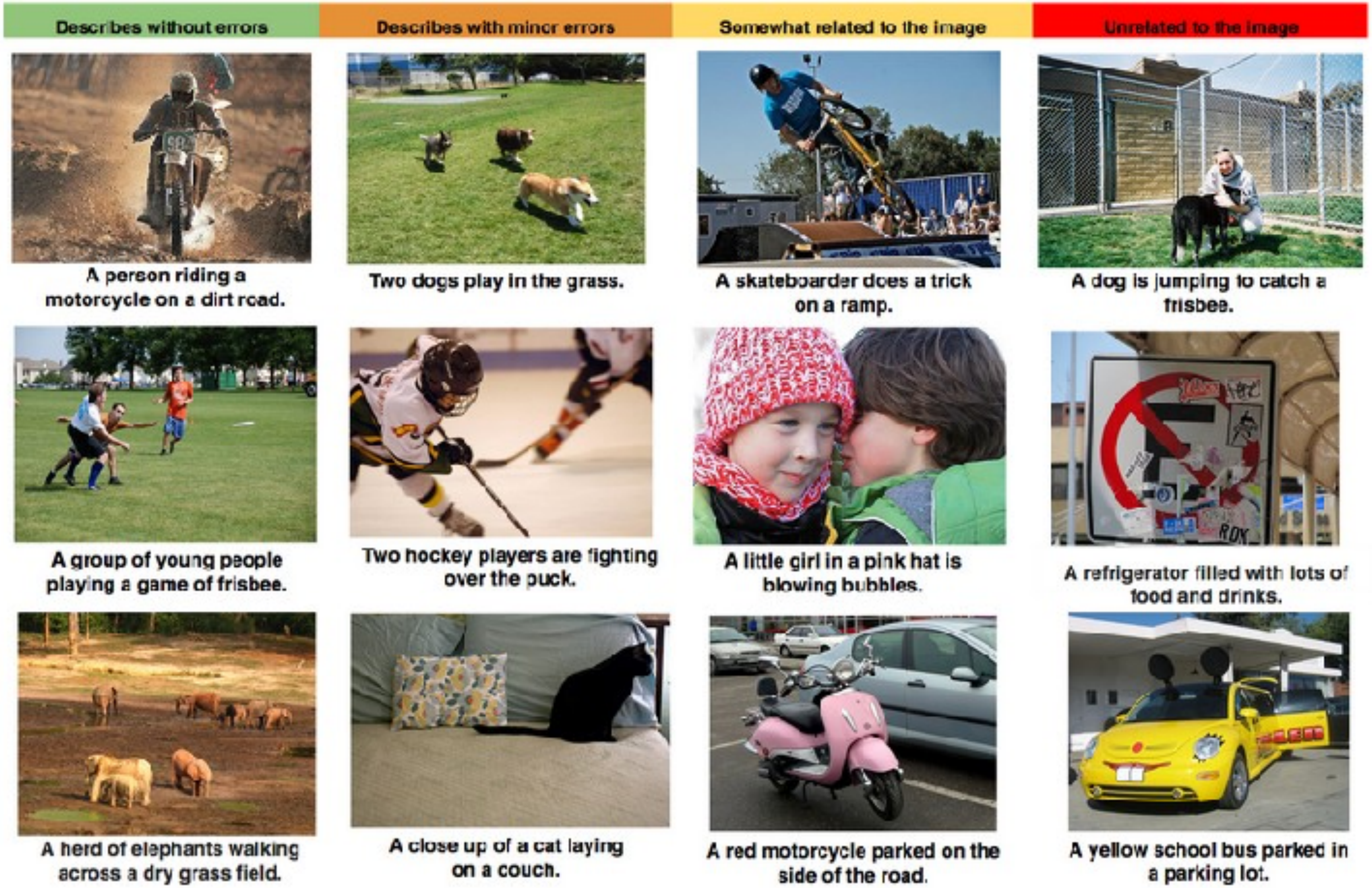


Image Captioning

[Vinyals et al., 2015]