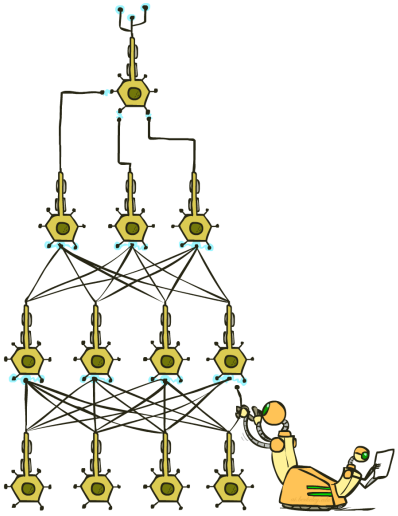
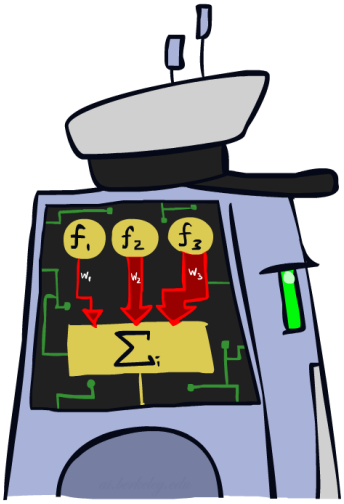


CS 188: Artificial Intelligence

Deep Learning I

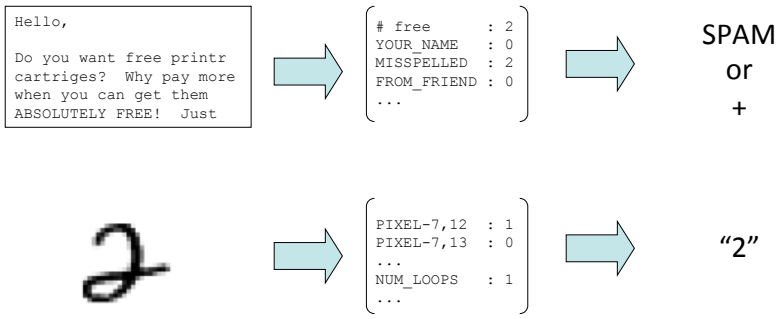
Instructors: Pieter Abbeel & Anca Dragan --- University of California, Berkeley

But First: Linear Classifiers



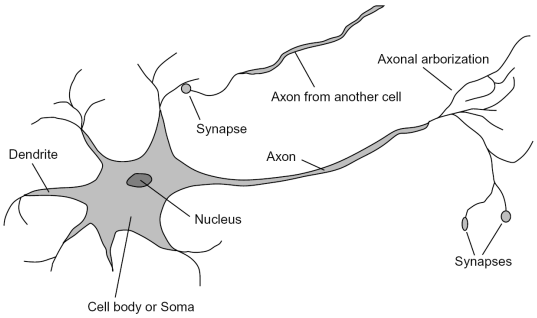
Feature Vectors

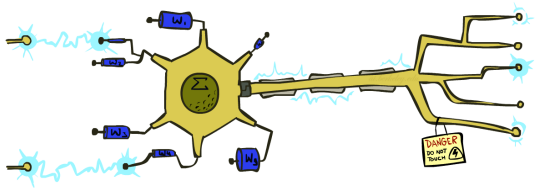
x $f(x)$ y



Some (Simplified) Biology

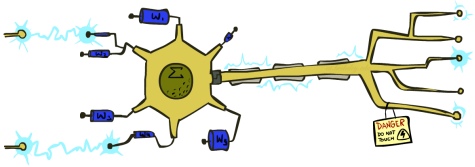
- Very loose inspiration: human neurons





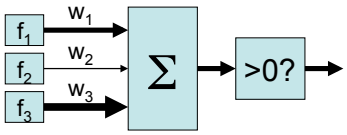
Linear Classifiers

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

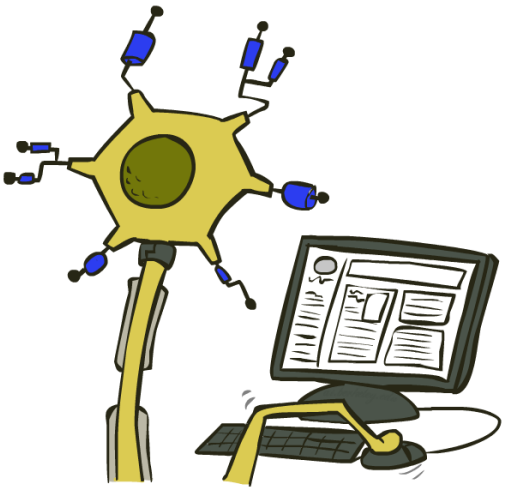


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

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



Web Search



Extension: Web Search

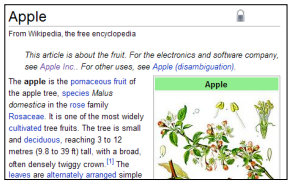
- Information retrieval:
  - Given information needs, produce information
  - Includes, e.g. web search, question answering, and classic IR

- Web search: not exactly classification, but rather ranking

x = “Apple Computers”

Feature-Based Ranking

x = “Apple Computer”

$f(x,$  $) = [0.3\ 5\ 0\ 0\ \ldots]$

$f(x,$  $) = [0.8\ 4\ 2\ 1\ \ldots]$

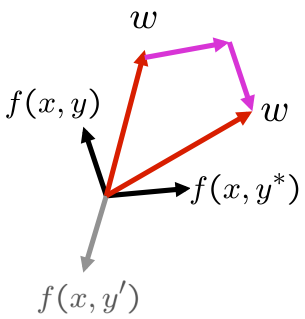
Perceptron for Ranking

- Inputs x
- Candidates y
- Many feature vectors: $f(x, y)$
- One weight vector: w
  - Prediction:

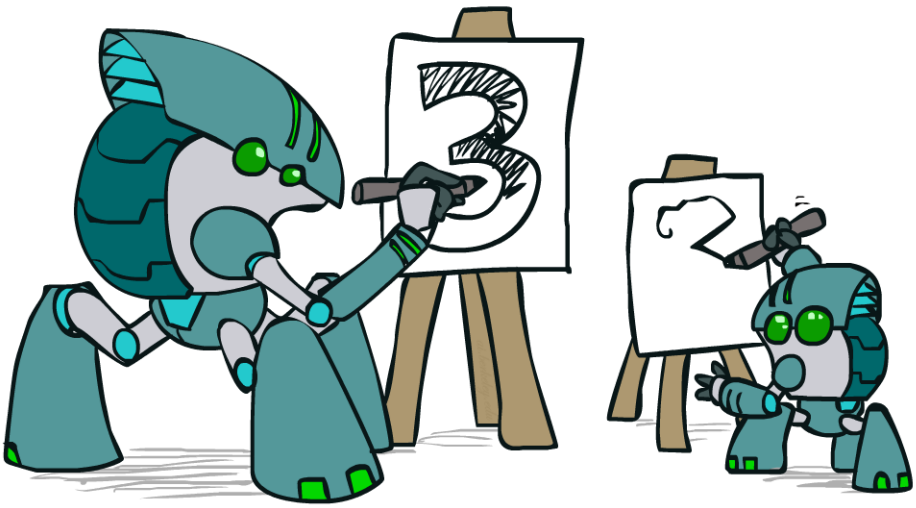
$$y = \arg\max_y \; w \cdot f(x, y)$$

  - Update (if wrong):

$$w = w + f(x, y^*) - f(x, y)$$

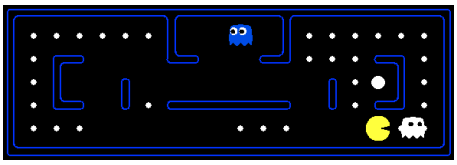


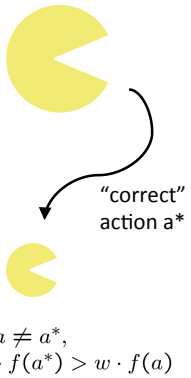
Apprenticeship



Pacman Apprenticeship!

- Examples are states s





- Candidates are pairs (s,a)
- "Correct" actions: those taken by expert
- Features defined over (s,a) pairs: f(s,a)
- Score of a q-state (s,a) given by:

$$w \cdot f(s,a)$$

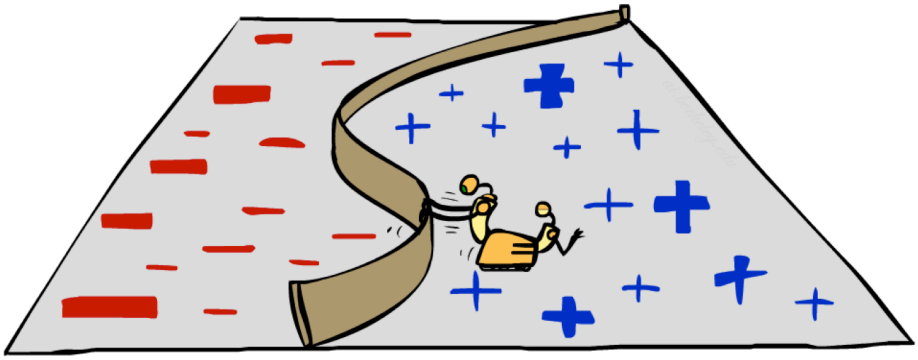
$$\forall a \neq a^*, \\ w \cdot f(a^*) > w \cdot f(a)$$

- How is this VERY different from reinforcement learning?

[Demo: Pacman Apprentice (L22D1,2,3)]

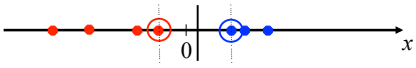
Video of Demo Pacman Apprentice

Non-Linearity

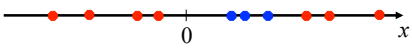


Non-Linear Separators

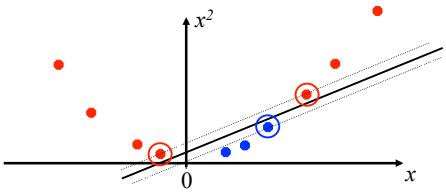
- Data that is linearly separable works out great for linear decision rules:



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



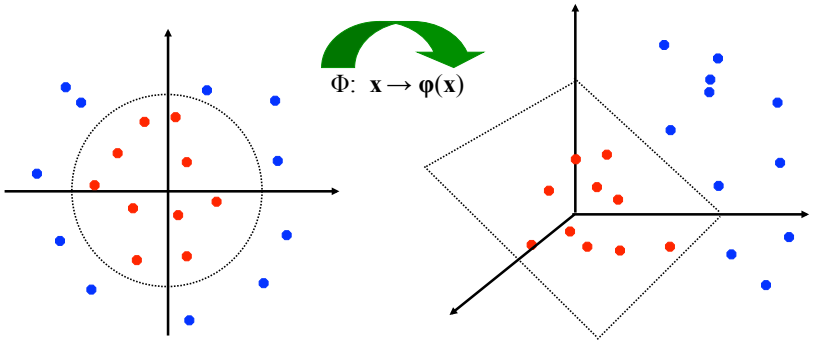
- How about… mapping data to a higher-dimensional space:



This and next slide adapted from Ray Mooney, UT

Non-Linear Separators

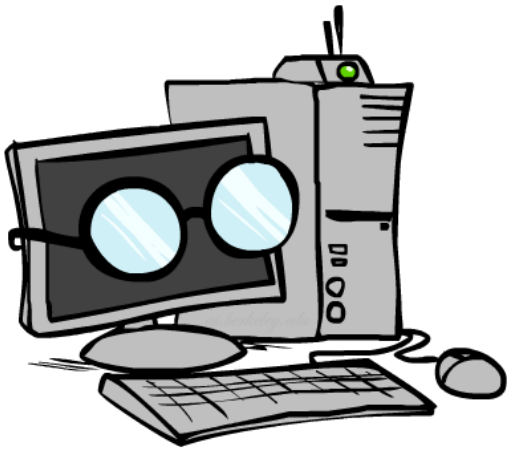
- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



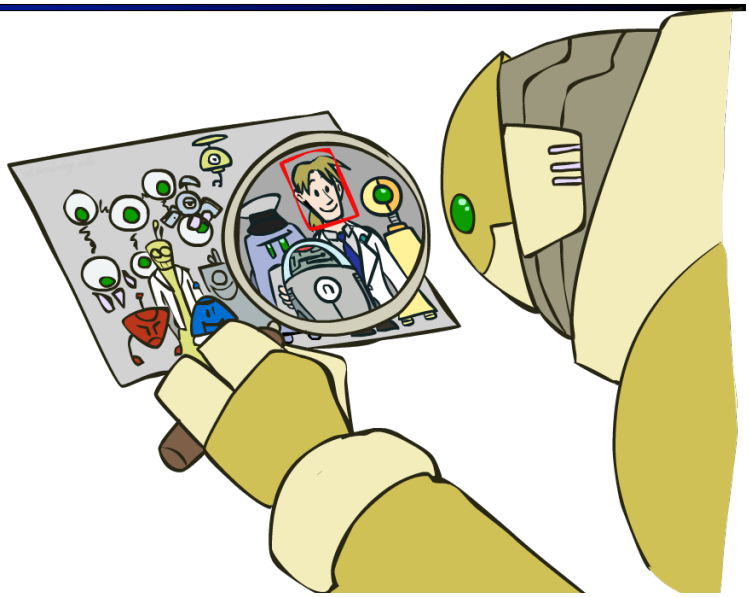
Feature Selection

- To choose between two feature sets:
  - For feature set 1: train perceptron on training data -> Classifier 1
  - For feature set 2: train perceptron on training data -> Classifier 2
  - Evaluate performance of Classifier 1 and Classifier 2 on hold-out data
    - Select the one performing best on the hold-out data

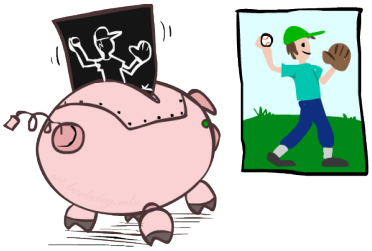
Computer Vision

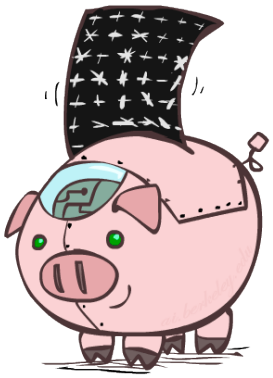


Object Detection

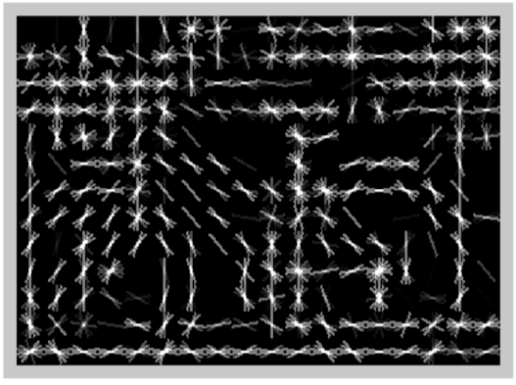


Manual Feature Design





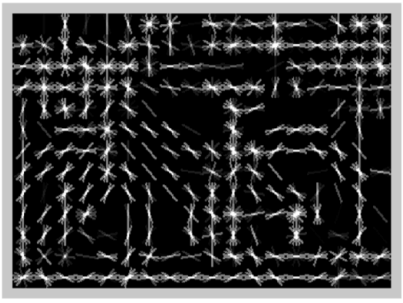
Features and Generalization



[Dalal and Triggs, 2005]

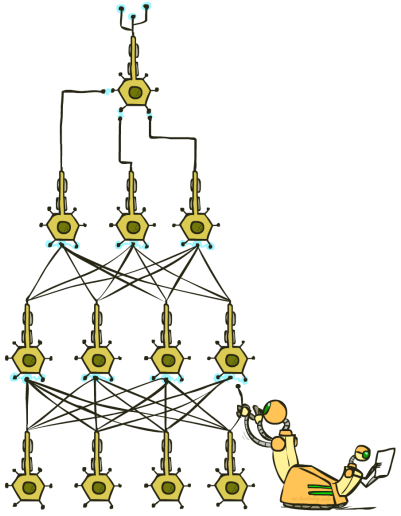
Features and Generalization

Image



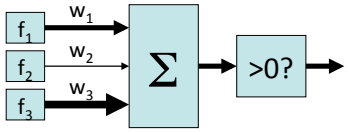
HoG

Manual Feature Design → Deep Learning

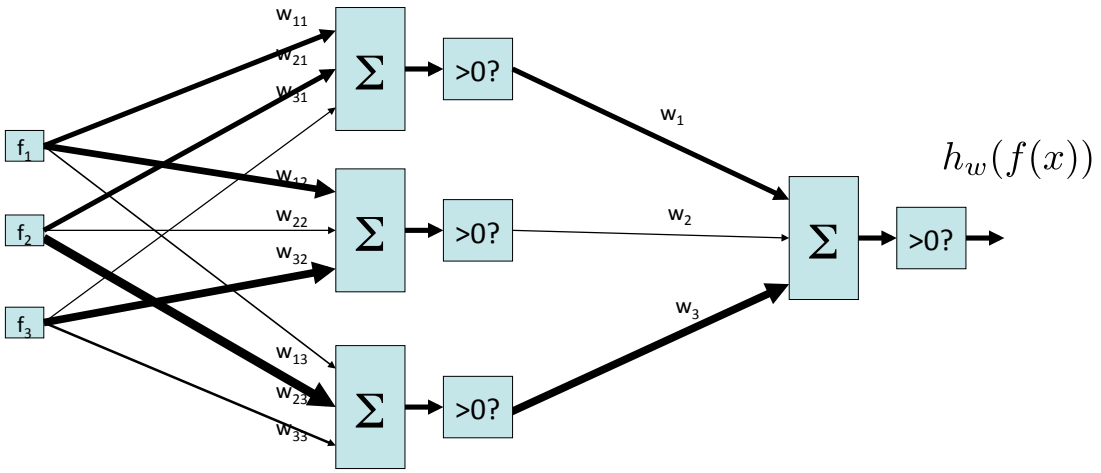


- Manual feature design requires:
  - Domain-specific expertise
  - Domain-specific effort
- What if we could learn the features, too?
  - **-> Deep Learning**

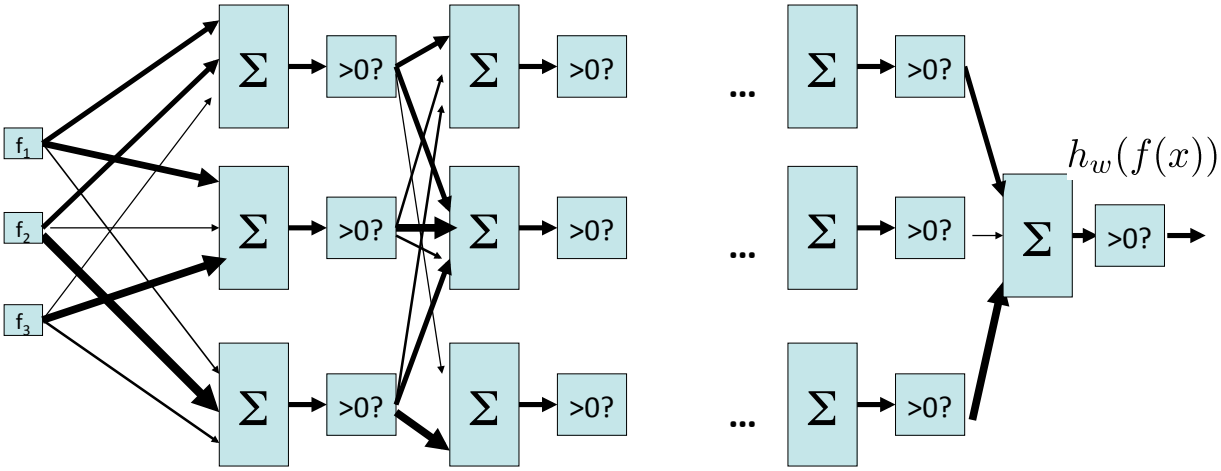
Perceptron



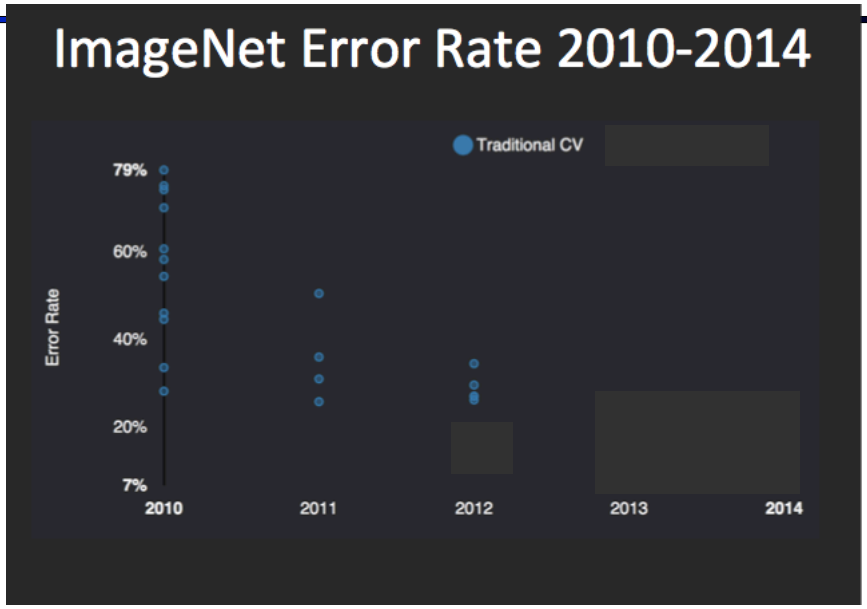
Two-Layer Perceptron Network



N-Layer Perceptron Network

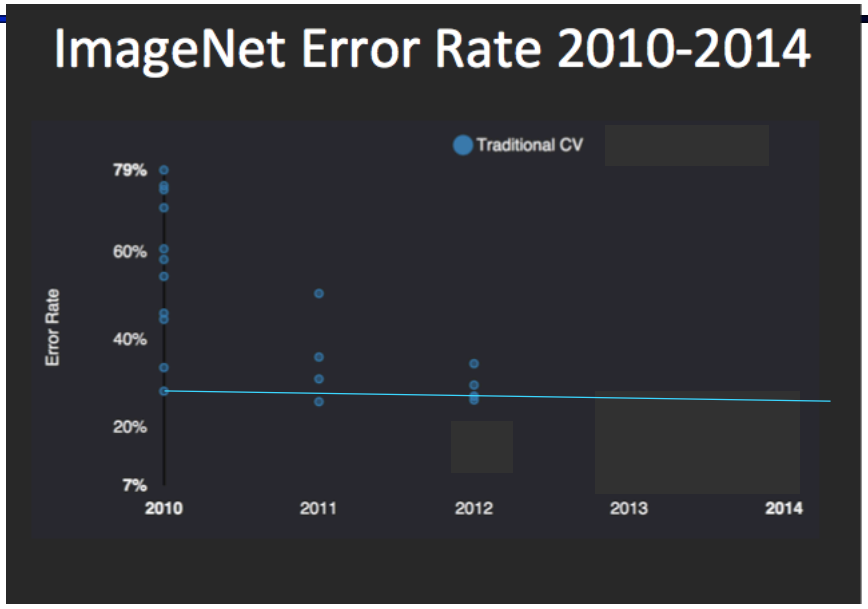


Performance



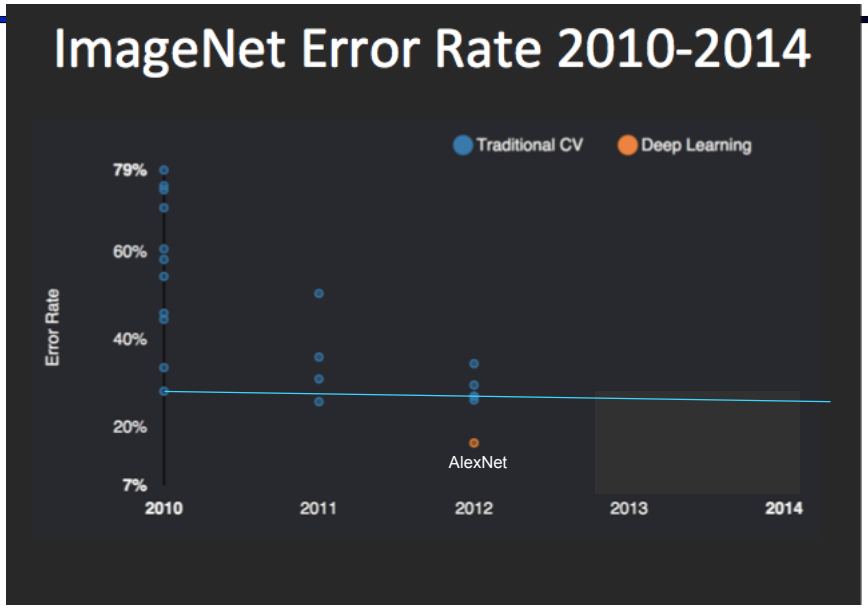
graph credit Matt Zeiler, Clarifai

Performance



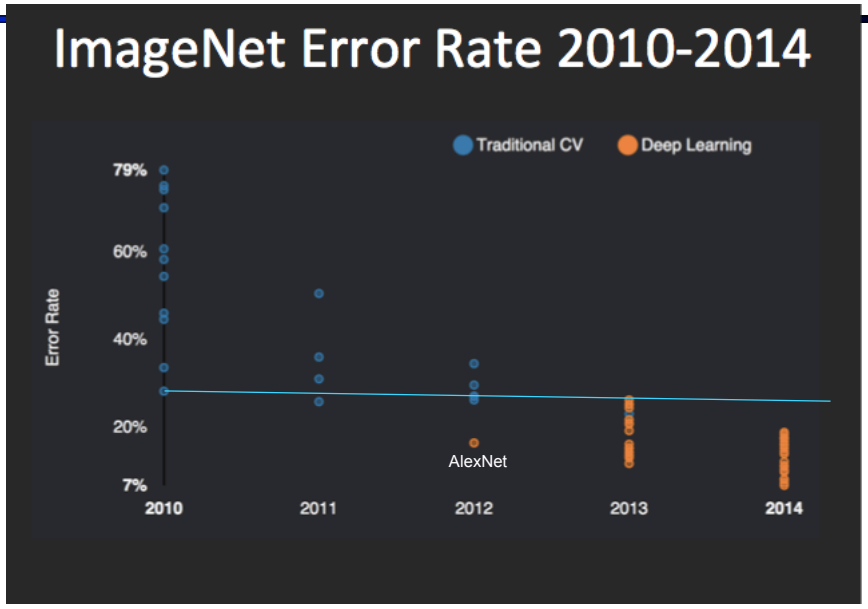
graph credit Matt Zeiler, Clarifai

Performance



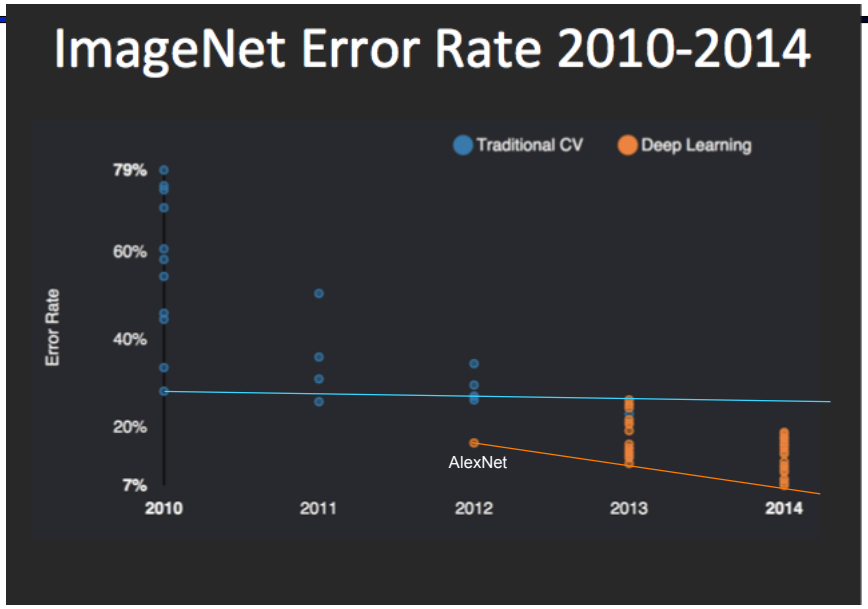
graph credit Matt Zeiler, Clarifai

Performance



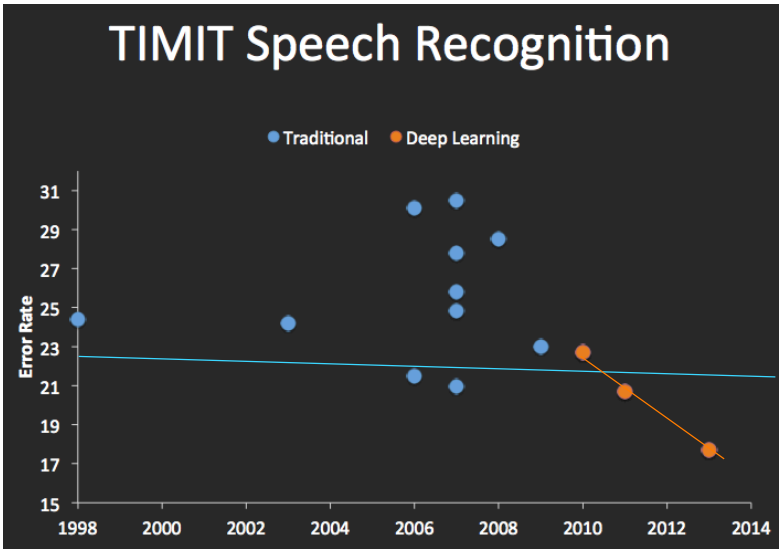
graph credit Matt Zeiler, Clarifai

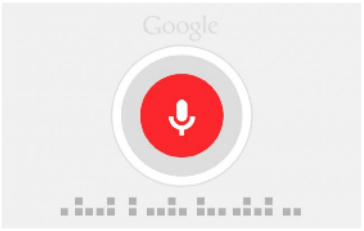
Performance



graph credit Matt Zeiler, Clarifai

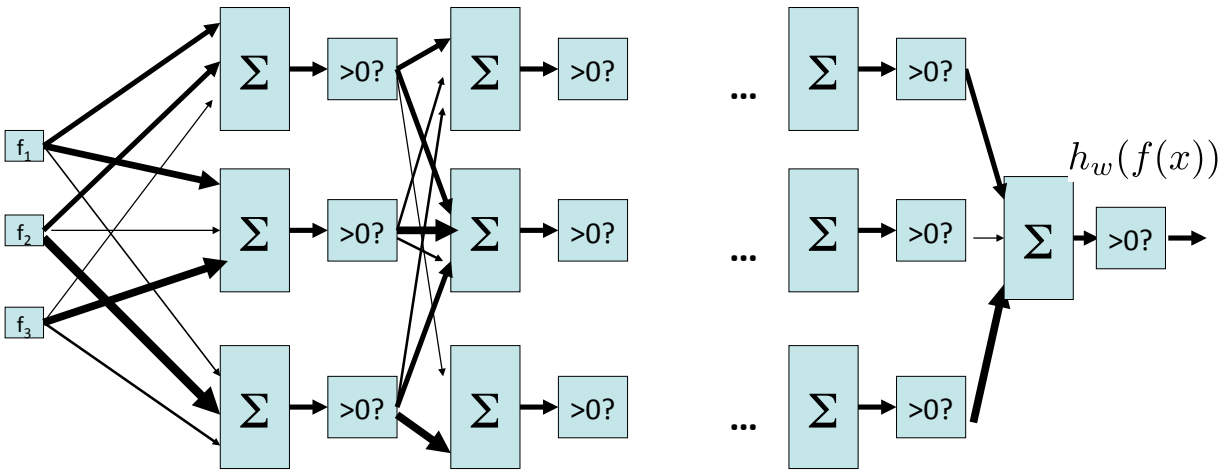
Speech Recognition





graph credit Matt Zeiler, Clarifai

N-Layer Perceptron Network



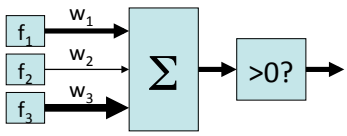
Local Search

- 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?

Soft-Max

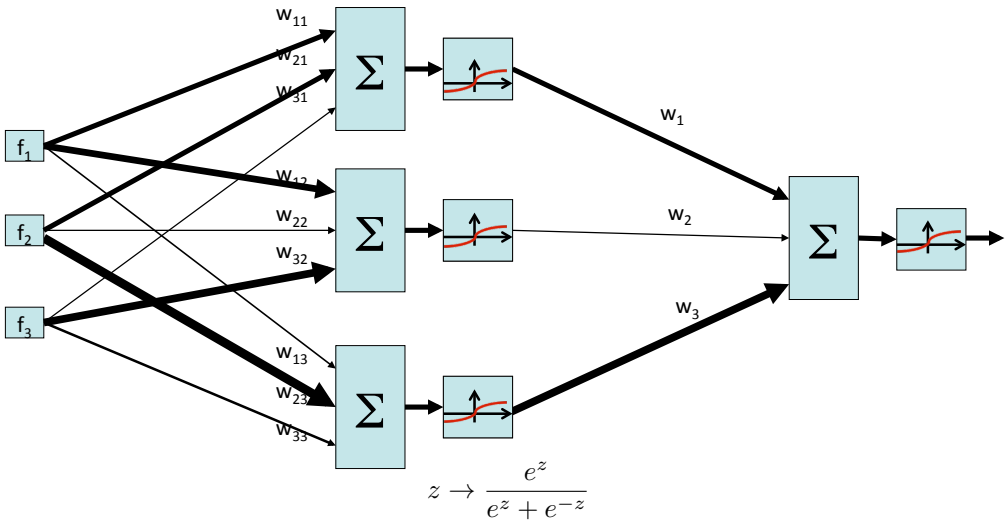
- Score for y=1: $w^\top f(x)$ Score for y=-1: $-w^\top f(x)$
- Probability of label:

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

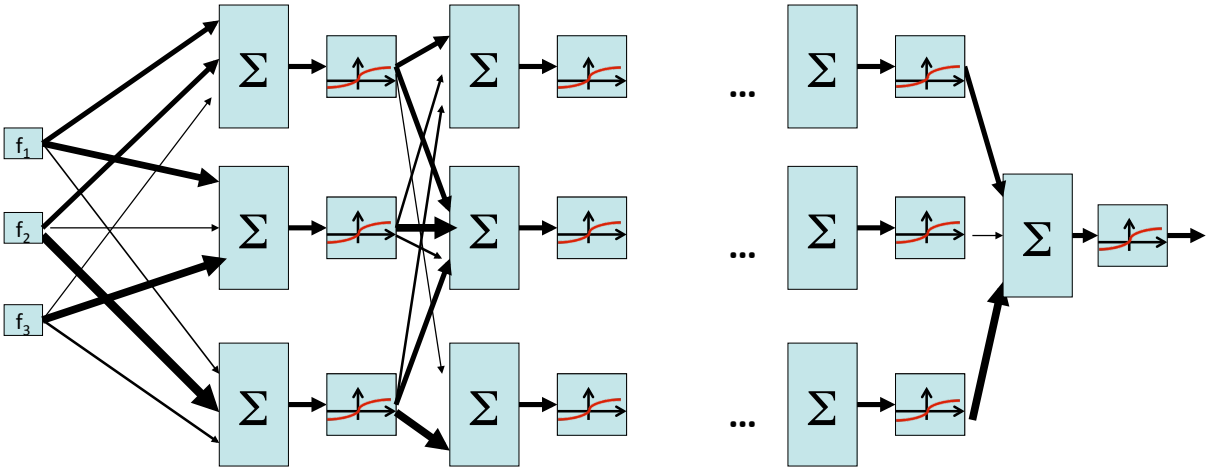
$$p(y=-1|f(x);w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

- 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



N-Layer Neural Network



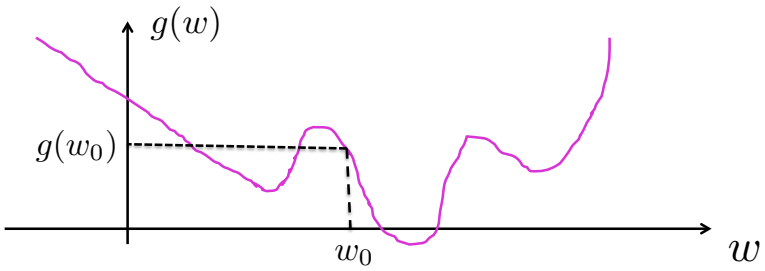
Our Status

- 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 ?

$$\max_w ll(w)$$

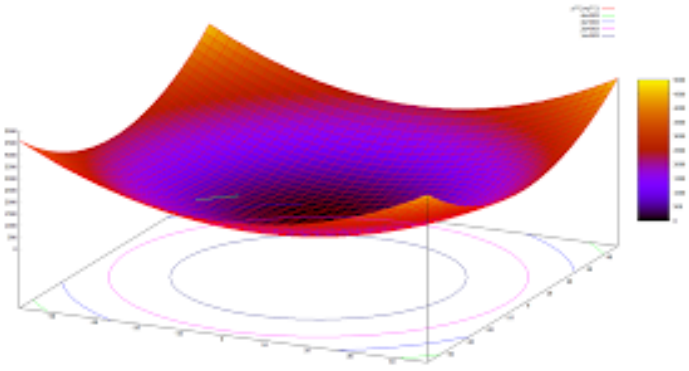
- Equivalently: $\min_w -ll(w)$

1-d optimization



- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction
- Or, evaluate derivative: $\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$
  - Which tells which direction to step into

2-D Optimization



Source: Thomas Jungblut's Blog

Steepest Descent

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

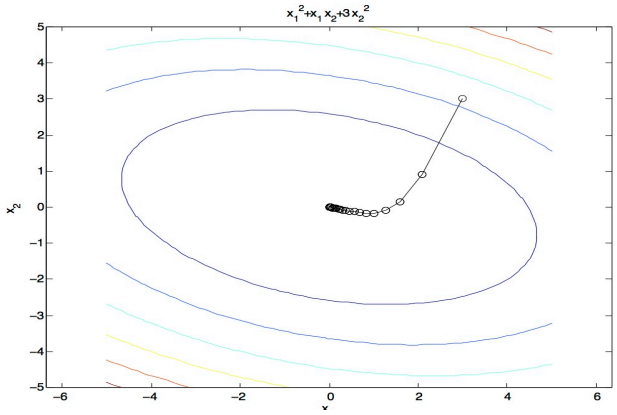


Figure source: Mathworks

What is the Steepest Descent Direction?

$$\min_{\Delta:\Delta_1^2+\Delta_2^2\leq\epsilon} g(w+\Delta)$$

- 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: $\min_{\Delta:\Delta_1^2+\Delta_2^2\leq\epsilon} \frac{\partial g}{\partial w_1}\Delta_1 + \frac{\partial g}{\partial w_2}\Delta_2$
- Recall: $\min_{a:\|a\|\leq\epsilon} a^\top b \quad \rightarrow \quad a = -b\frac{\epsilon}{\|b\|}$
- Hence, solution: $-\nabla g \frac{\epsilon}{\|\nabla g\|}$ $\qquad \nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \end{bmatrix}$

Generally, Steepest 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 %

Try Many Learning Rates

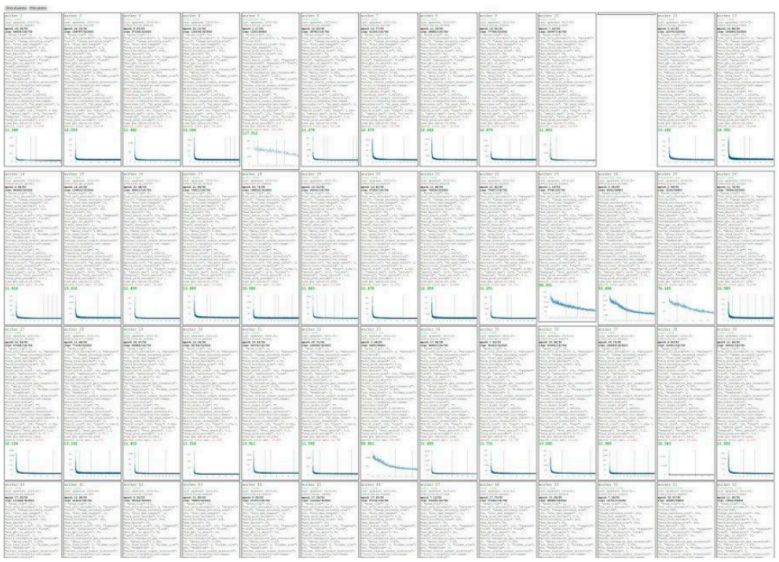
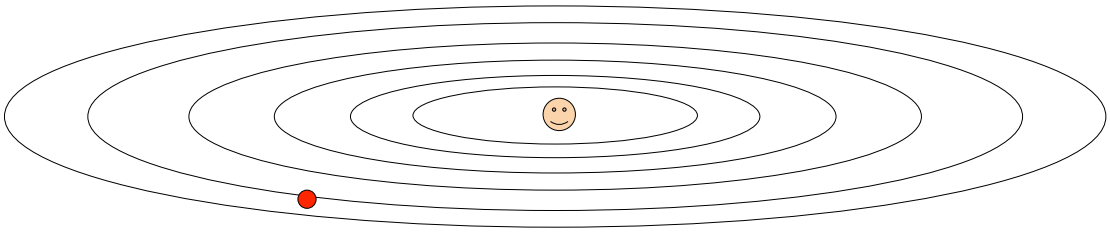


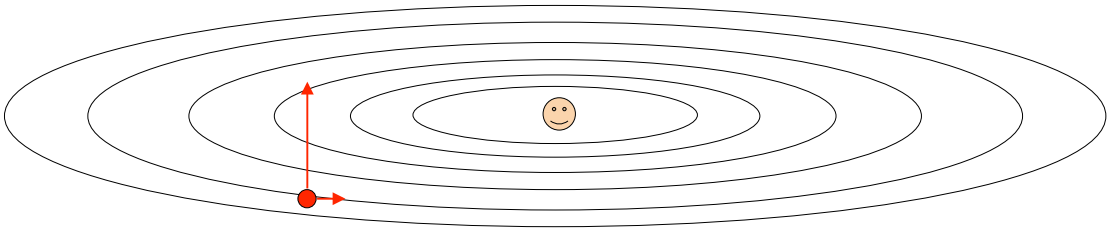
Figure source:
Andrej Karpathy

Suppose loss function is steep vertically but shallow horizontally:



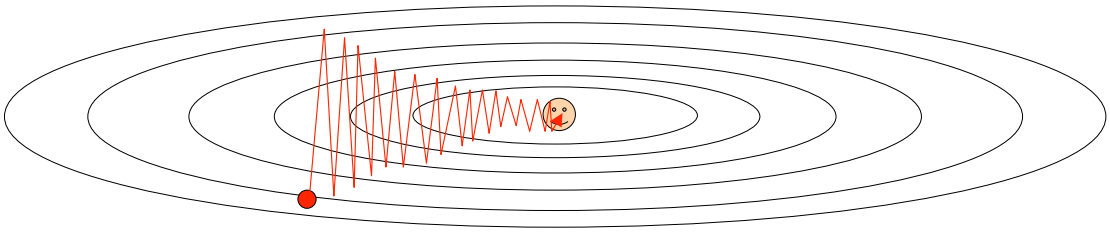
Q: What is the trajectory along which we converge towards the minimum with SGD?

Suppose loss function is steep vertically but shallow horizontally:



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

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

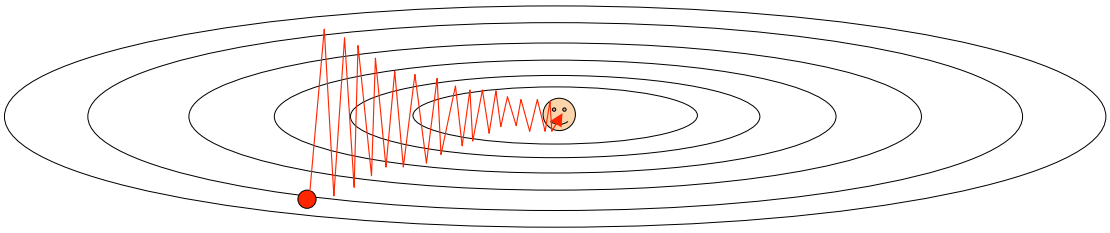
- Init: w
- For i = 1, 2, …

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

$$w \leftarrow w + v$$

- 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?

Some More Advanced Optimization Methods*

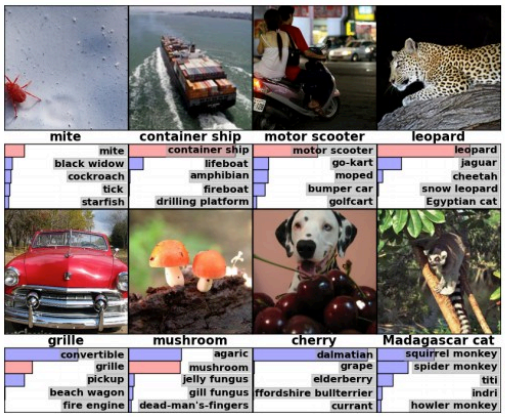
- Nesterov Momentum Update
- AdaGrad
- RMSProp
- ADAM
- L-BFGS

Remaining Pieces

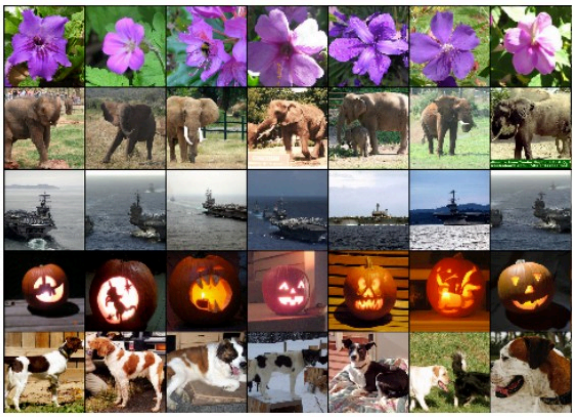
- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization
- Renormalization
- Computing the gradient $\nabla g(w)$
  - Backprop
  - Gradient checking

ConvNets are everywhere

Classification



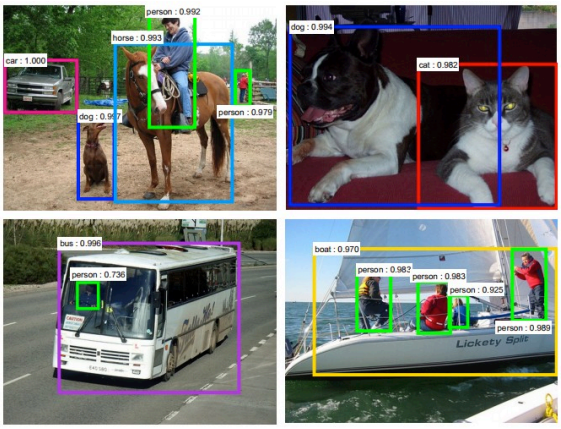
Retrieval



[Krizhevsky 2012]

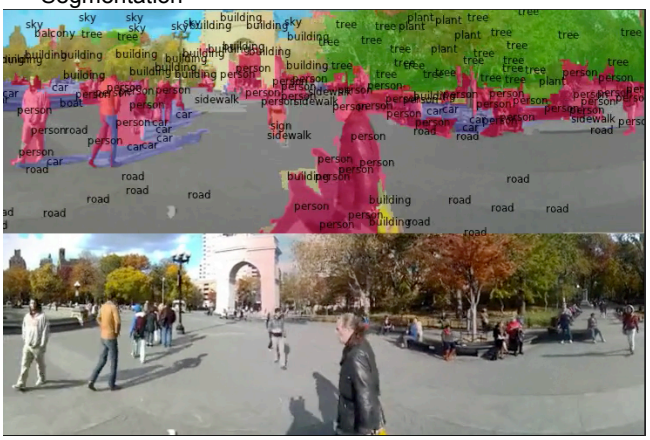
ConvNets are everywhere

Detection



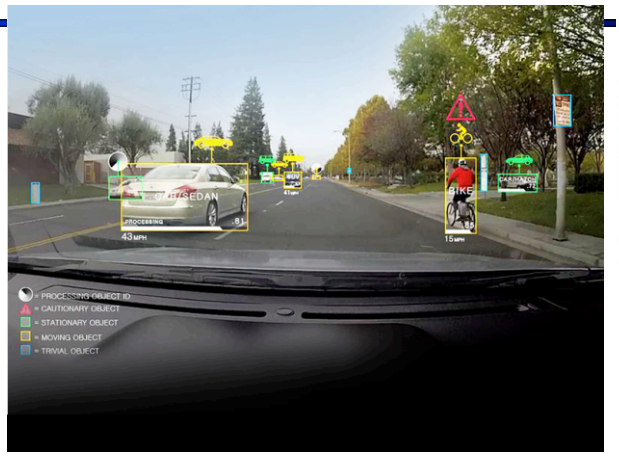
[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



[Farabet et al., 2012]

ConvNets are everywhere

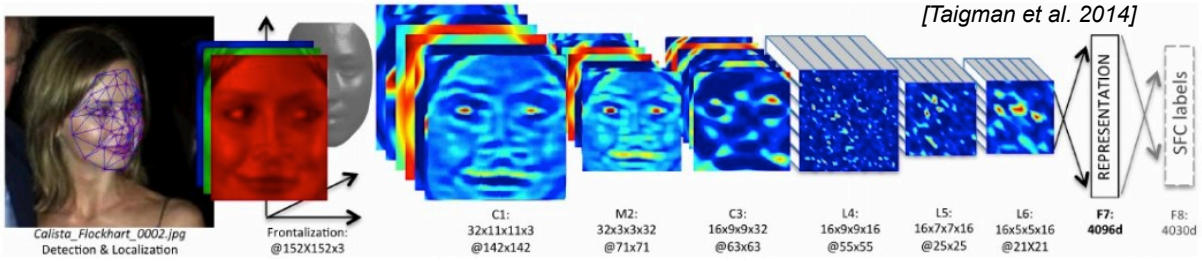


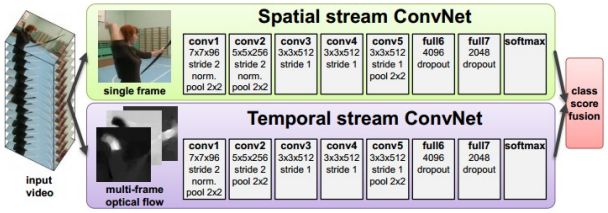
self-driving cars

NVIDIA Tegra X1

ConvNets are everywhere

[Taigman et al. 2014]





[Simonyan et al. 2014]

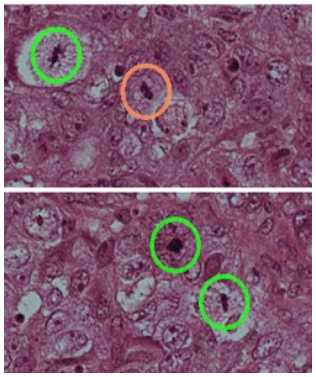
[Goodfellow 2014]

ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]

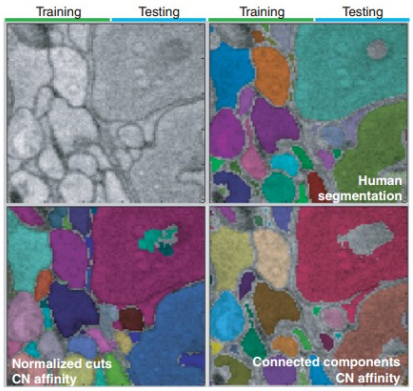
ConvNets are everywhere



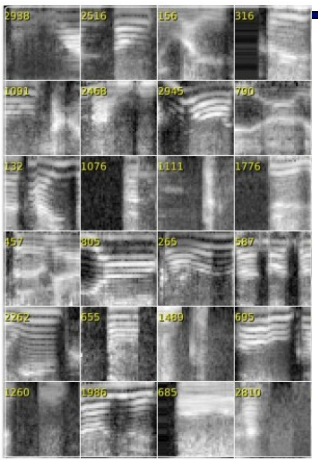
[Ciresan et al. 2013]

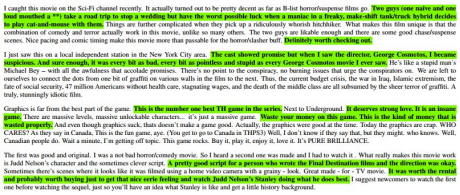
[Sermanet et al. 2011]
[Ciresan et al.]

ConvNets are everywhere

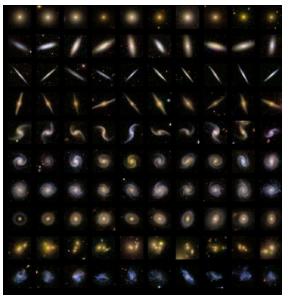


[Turaga et al., 2010]





[Denil et al. 2014]



ConvNets are everywhere

Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010

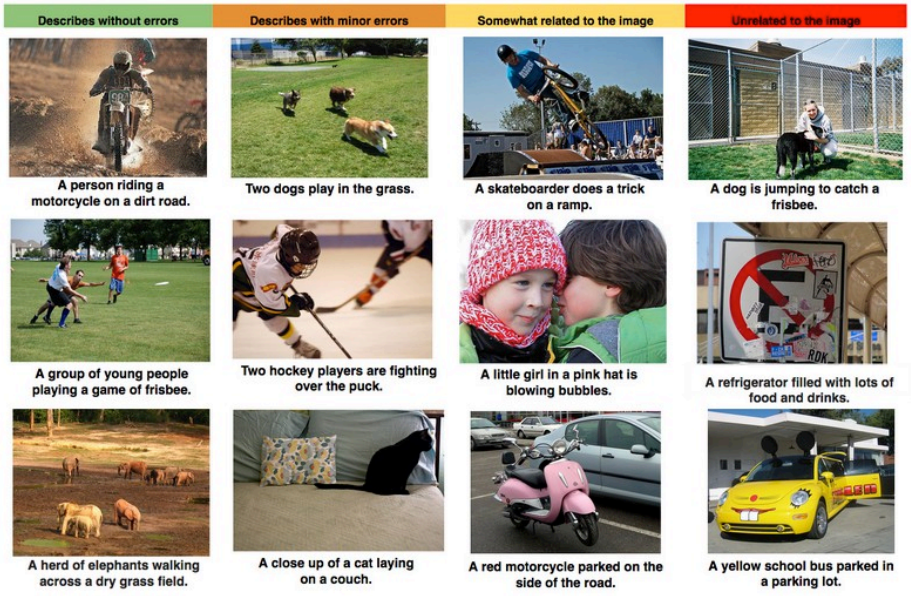


Image Captioning

[Vinyals et al., 2015]

reddit.com/r/deepdream

Next Time

- Final Contest results
- Robot butlers
- Where to go next to learn more about AI