
The Ancient
Secrets
Computer Vision

Logistics:

- Homework 3 is out!
  - Optical flow
  - Won’t work on newest OpenCV 3.4
  - Should work on OpenCV 2, maybe early versions of 3
  - demo!


Previously
On
the
Ancient Secrets
of Computer Vision

What is machine learning?

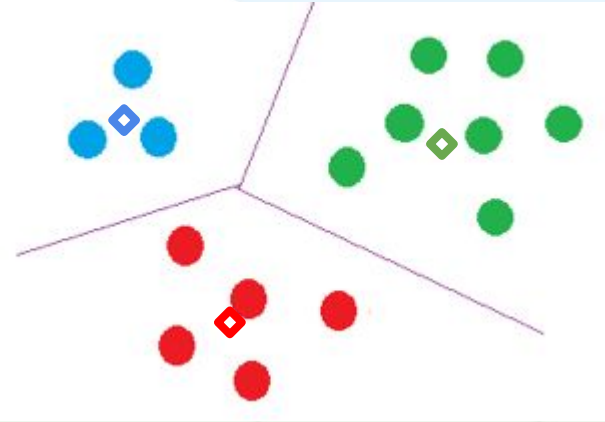
- Algorithms to approximate functions
  - Usually use lots of statistics
  - Usually minimize some form of *loss function*
- Supervised learning
  - Given inputs to a function, try to predict the output
  - Have lots of labelled examples
- Semi-supervised learning
  - Same but number of labelled examples < number of examples
- Unsupervised learning
  - Want to model unlabelled data
  - Find similarities and differences between subgroups of data
  - Learn functions to generate new data

K-means clustering

Assume points are close to other points in group, far from points out of group

Algorithm:

Randomly initialize cluster centers
Loop until converged:
    Calculate distance points <-> centers
    Assign each point to closest center
    Update cluster centers: avg of points



https://healthcare.ai/step-step-k-means-clustering/

Clustering on images

Group together pixels by color, automatic segmentation: k-means, k = 2

Supervised Learning (typically)

Have data with labels, want to take new data and predict correct label

Pick a model

Do you know anything *a priori* about your data? Is it linear, polynomial, etc?

Pick a loss function

Easy to calculate function of model parameters given training data

Should give an approximation of how "good" your model is

E.g. Sum squared error (L_2 loss), sum absolute differences (L_1 loss), etc…

Pick model parameters to minimize loss

Find local or global minimum of loss.

How? Set derivative = 0, closed form solution, gradient descent, etc

Regression

Predict real-valued output

What temperature will it be tomorrow?

Loss function: squared error

Evaluation: squared error

Classification

Predict category output

Will it be sunny tomorrow?

Often predicts continuous values which are probabilities of different labels.

Loss function: Log likelihood, cross-entropy

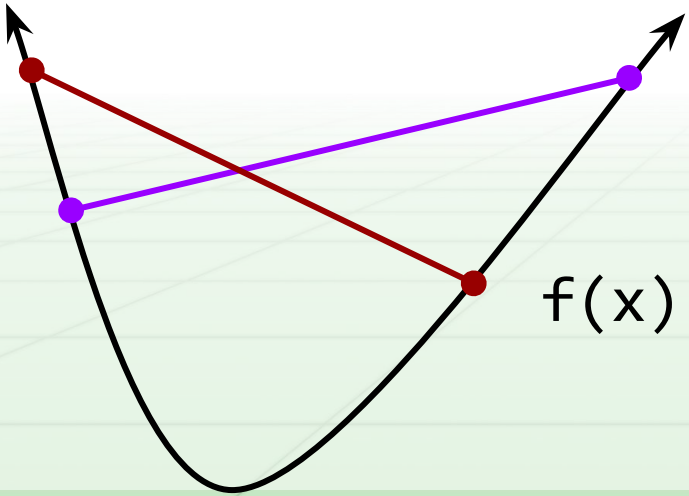
Evaluation: accuracy

Loss	Likelihood
Measure of how wrong our model is	How probable our model thinks our training data is
Want a smaller loss	Want high likelihood, model that explains the data well
Try to find local or global minima of our loss function	Find local or global maxima of likelihood function
Derivative = 0, gradient descent	Derivative = 0, gradient ascent

Convex

Connect any two points on graph with a line, that line lies above function everywhere.

Any local extrema is global extrema!



Non-convex

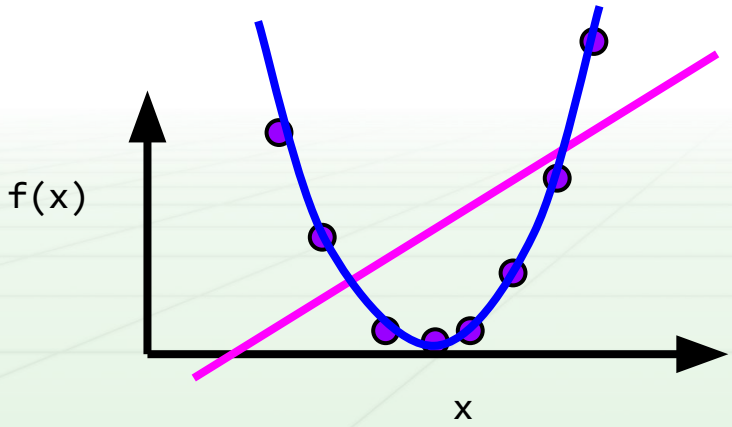
Local optima are not global optima.

Harder to optimize, especially to find global optima.

Bias

Error from assumptions model makes about data

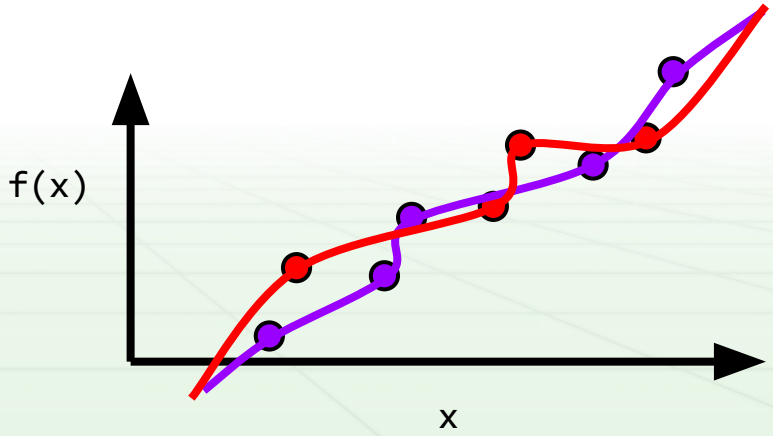
Less complex algorithms -> more assumptions about data



Variance

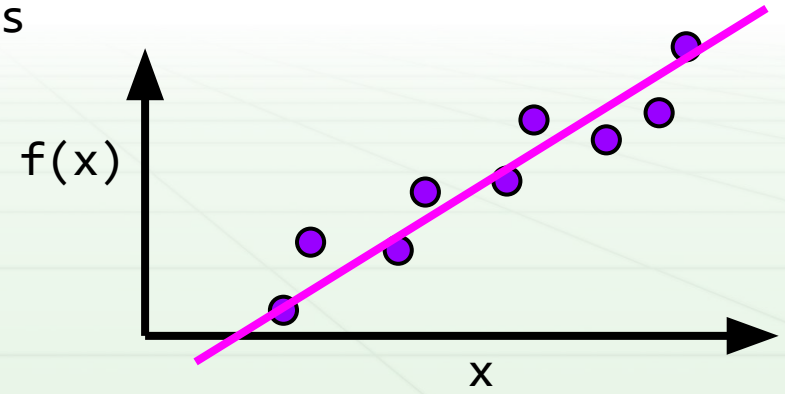
Algorithm’s sensitivity to noise

More complex algorithms are more sensitive!



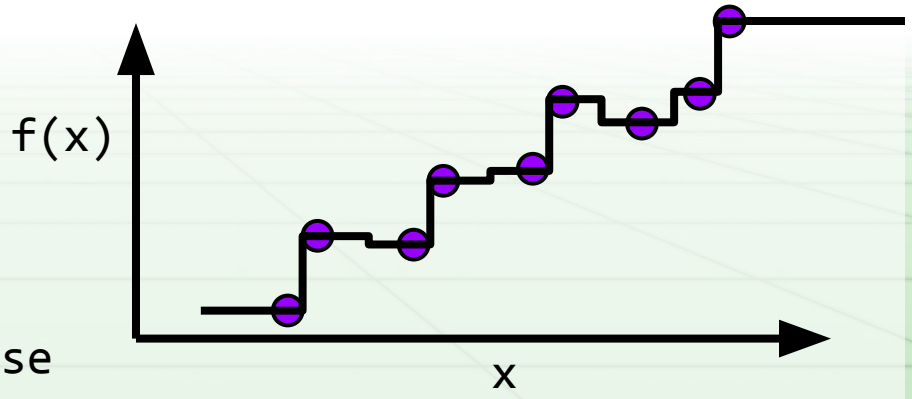
Linear regression

- $f^*(x) = ax + b$
- Learn a and b from data (how?)
  - Minimize squared error!
  - Loss function $L(f^*) = \Sigma_i \, ||f(x_i) - f^*(x_i)||^2$
  - Want $\text{argmin}_{a,b}[L(f^*)]$
  - Extrema when derivative = 0
  - Solve linear system of equations
    - **Ma = b**
  - Already did this!



Nearest neighbor

- f*(x) = f(x') for nearest x' in training set
- Low bias: no assumptions about data
- High variance: very sensitive to training set
- Benefits:
  - Super easy to implement
  - Easy to understand
  - Arbitrarily powerful, esp with lots of data
- Weaknesses:
  - Hard to scale
  - Prone to overfitting to noise



Classification approach: partitions

- Find best split or splits to data along one variable
- One possibility, Pr(flu) = (temp > 99.5)
  - Pretty accurate on our training data

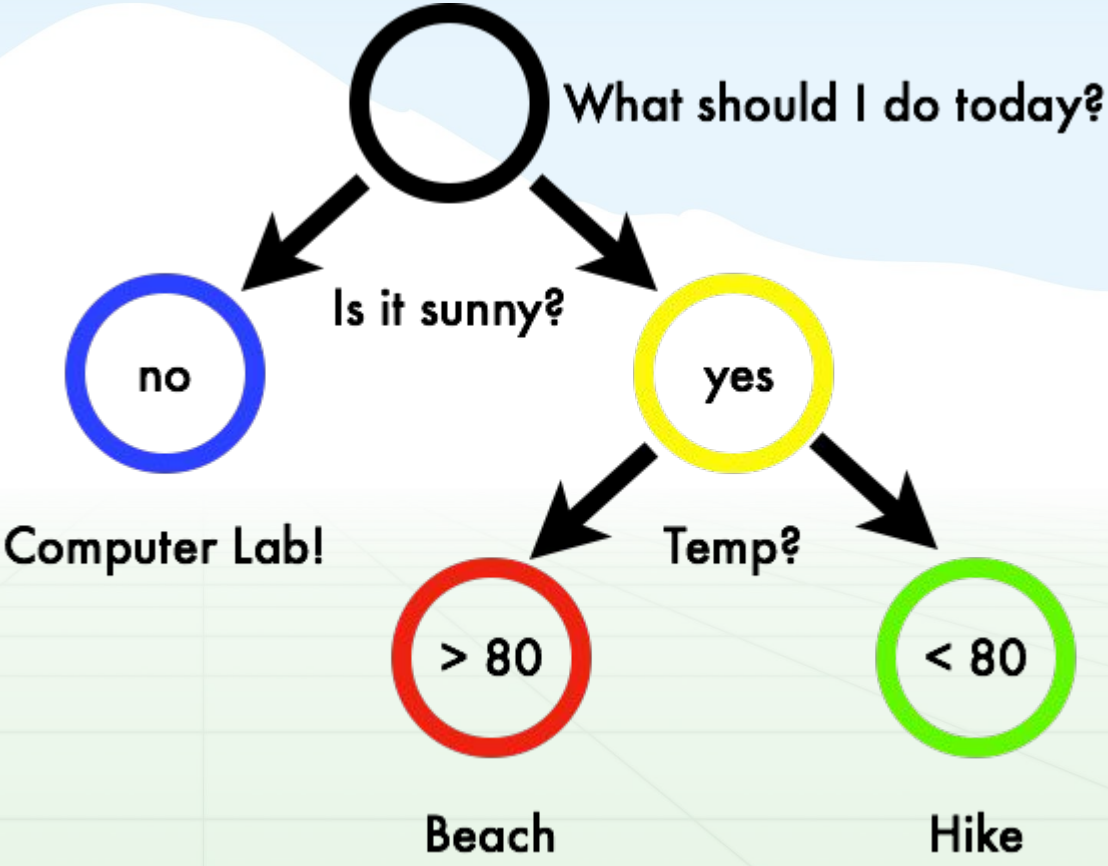
sore throat	runny nose	nausea	temp	chills	pain	age	days	diagnosis
no	yes	yes	101.3	yes	7	15	5	flu
yes	yes	no	98.8	no	3	74	3	not flu
yes	yes	no	100.1	yes	4	46	4	flu
yes	yes	yes	99.8	yes	6	27	1	flu
yes	no	no	98.4	yes	5	35	2	not flu
yes	yes	yes	99.9	no	3	42	4	not flu

Trees: layers of partitions

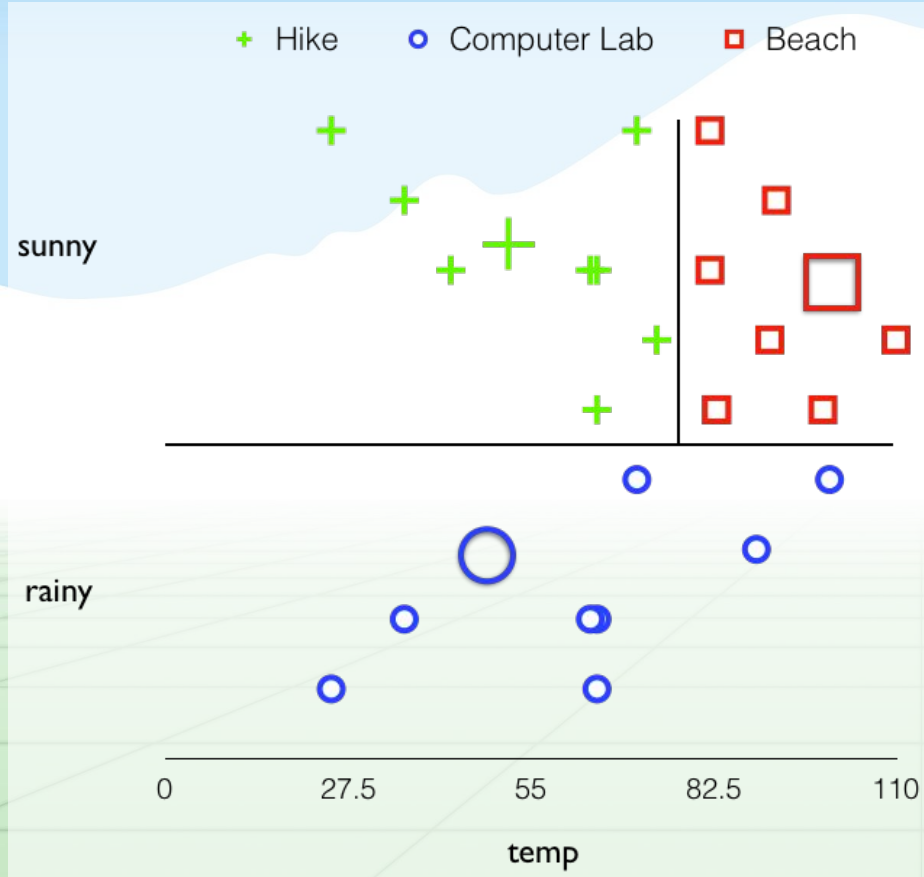
Very simple models

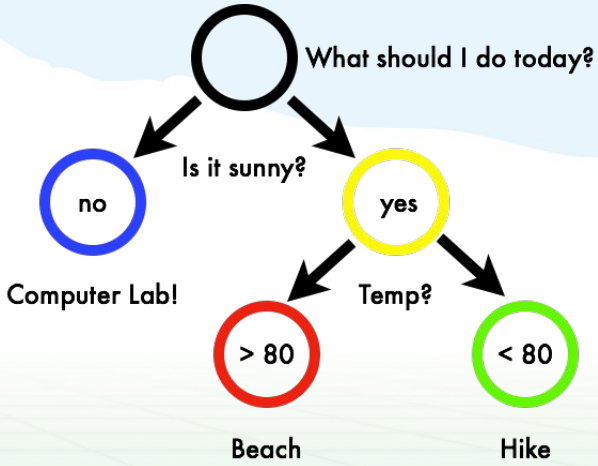
Benefits:

- Interpretable
- Easy to use
- Good for applications
  - E.g. medicine

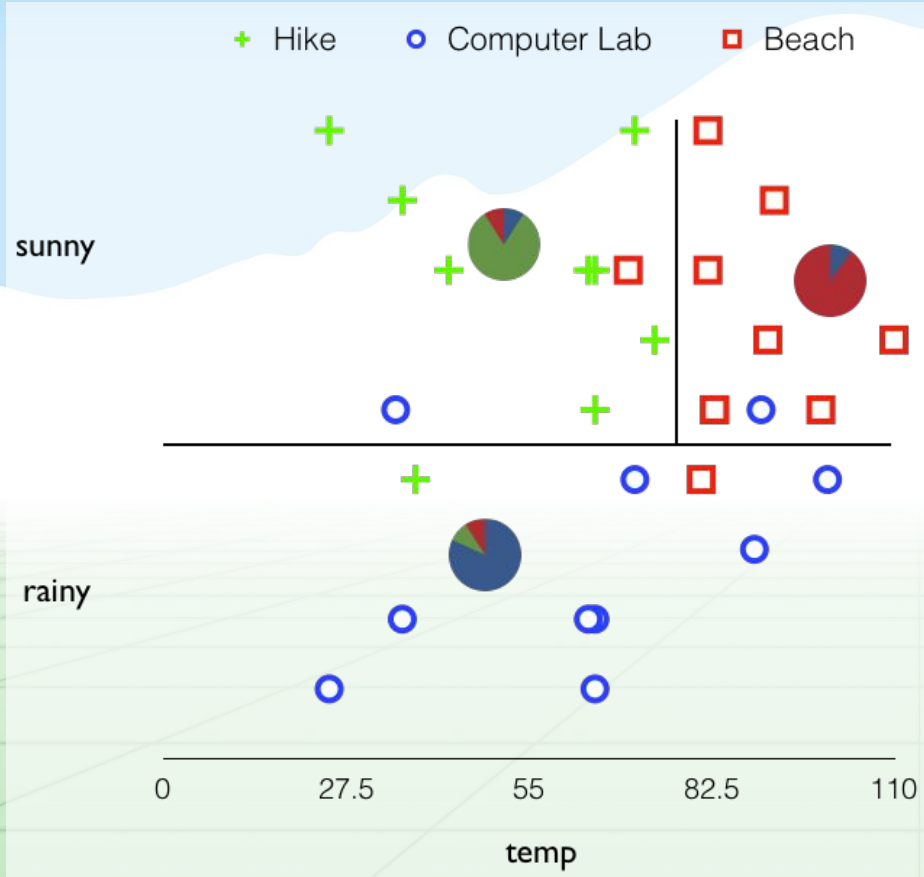


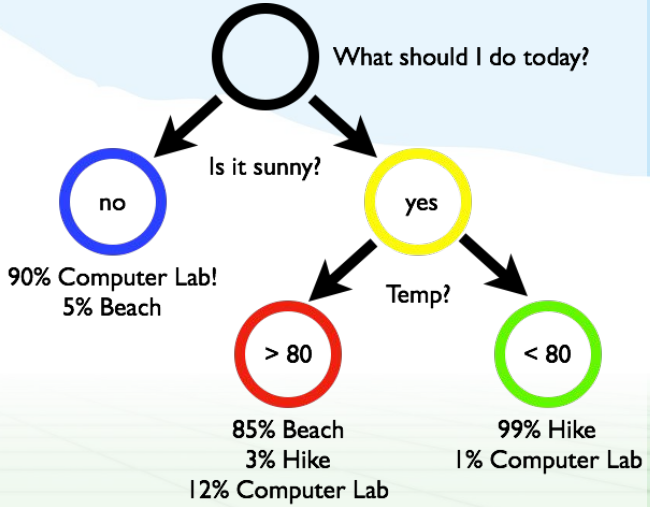
Predict new data based on what region it falls into





Data might be noisy, use soft assignments



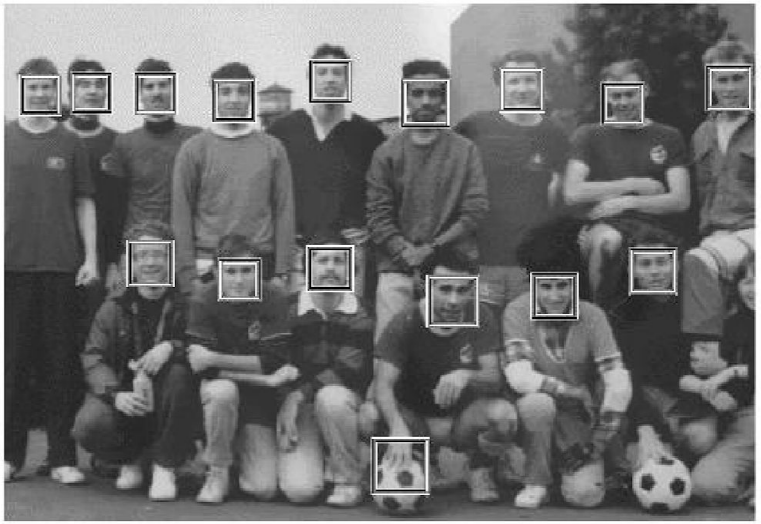


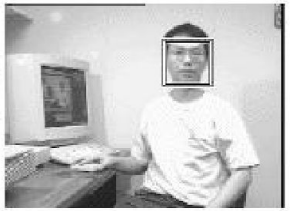
Case study: Viola-Jones Face detection

Want very fast detector for embedded systems

Haar features

Cascade of boosted classifiers





Haar features

Haar features:

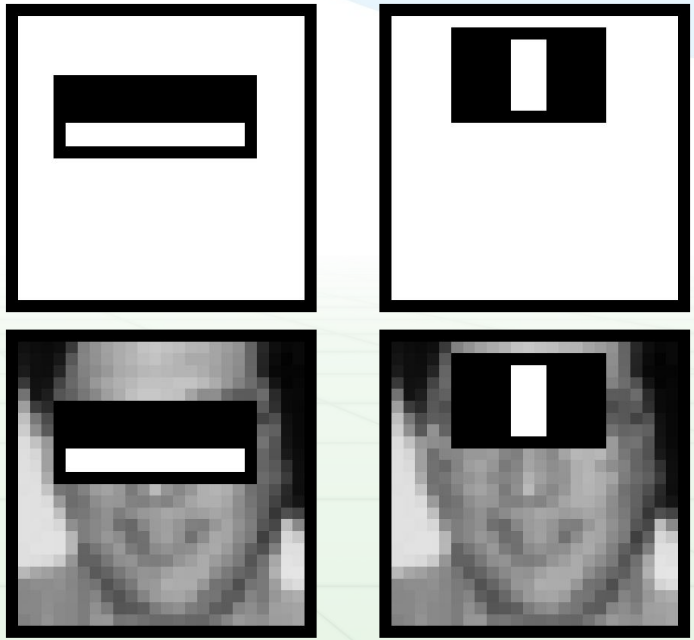
Response = Σ pix in black region - Σ pix in white region

Why do Haar features work?

Eyes are generally darker than cheeks

Bridge of nose lighter than eyes

Etc.



Also, fast to compute!

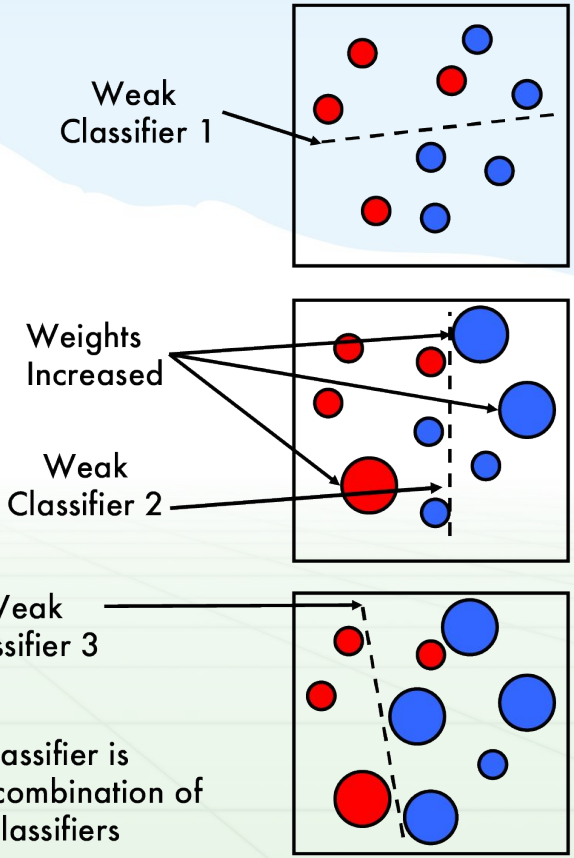
Integral images - fast sums over regions.

https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

Boosting: combination of weak classifiers

Way to make weak classifiers better
Train a weak classifier
Reweight data we got wrong, train again
...and again
Until you feel like stopping

Final classifier is combination of these



https://www.cs.ubc.ca/~lowe/425/slides/13-ViolaJones.pdf

Cascade: series of cheap -> expensive models

1st classifier

Very fast, throws out easy negatives

2nd classifier

Fast, throws out harder negatives

3rd classifier

Slower, throws out hard negatives

Only run slow, good classifiers on hard examples

Fast classifier that is still very accurate

Linear classifiers

Given dataset, learn weights $\mathbf{w}$

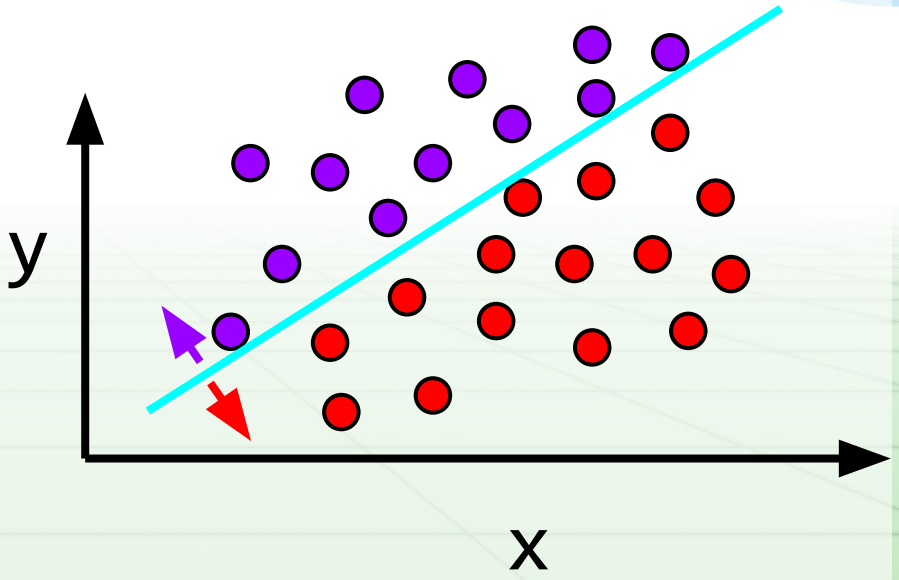
Output of model is weighted sum of inputs

$P(\text{purple} \mid \mathbf{x}) = f(\mathbf{w} \cdot \mathbf{x}) = f(\Sigma_i(w_i x_i))$

Where f is some function (a few options)

Typically a bias term:

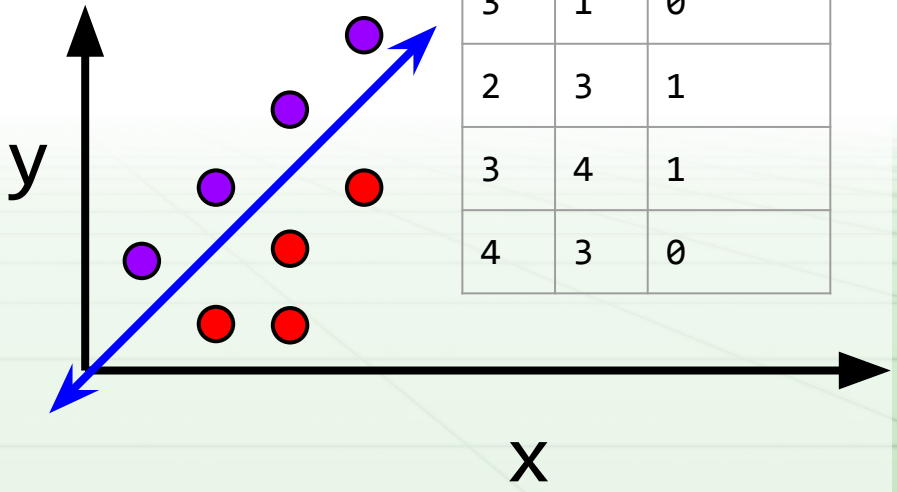
$$f(\Sigma_i(w_i x_i) + w_{bias})$$



Classification in two dimensions

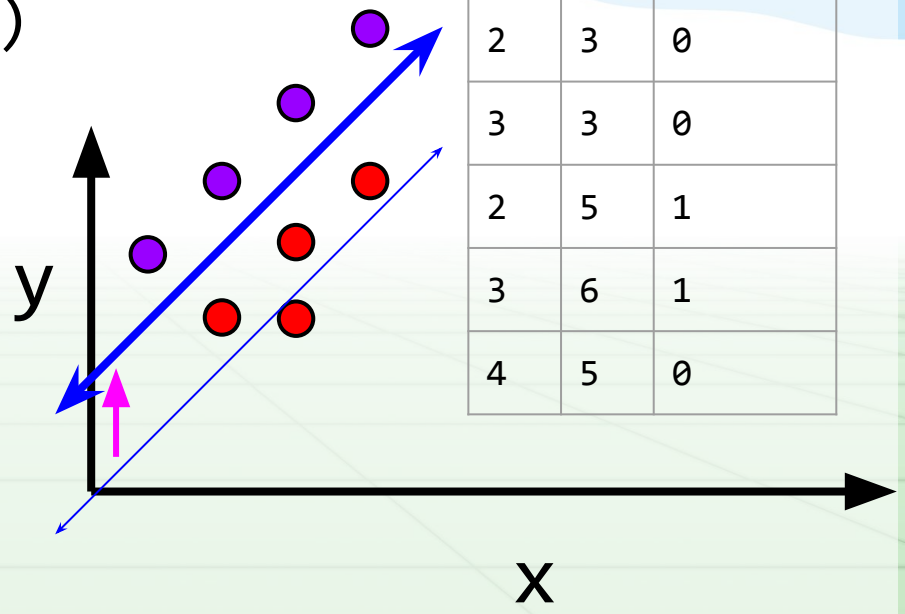
- Simple example:
  - Learned weights: [-1, 1]
  - f is threshold at 0
- New data point (4, 5)
  - f(**w**·**x**) = 1
- New data point (3, 2)
  - f(**w**·**x**) = 0
- Decision boundary: x=y

x	y	purple
1	2	1
2	1	0
3	1	0
2	3	1
3	4	1
4	3	0



What if data is shifted up by two?

- Need a bias!:
  - Learned weights: [-1, 1, -2]
  - f is threshold at 0
- New data point (4, 7, 1)
  - (**w**·**x**) = (4,7,1)·(-1,1,-2)
    = 4*-1 + 7*1 + 1*-2 = 1
  - f(**w**·**x**) = f(1) = 1



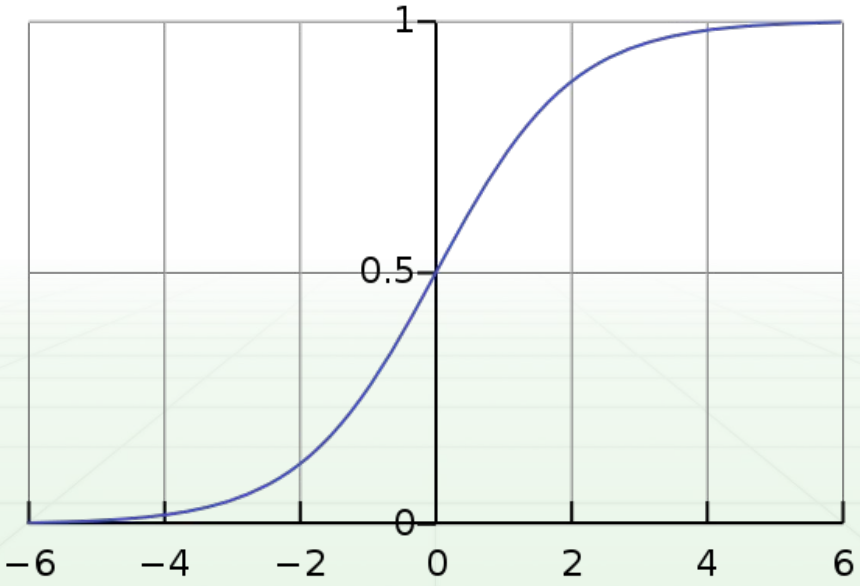
x	y	purple
1	4	1
2	3	0
3	3	0
2	5	1
3	6	1
4	5	0

Logistic regression

Linear classifier, f is logistic function

$\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$

Maps all reals -> [0,1], probabilities!



Logistic regression

Linear classifier, f is logistic function

$\sigma(x) = 1/(1 + e^{-x}) = e^{x}/(1 + e^{x})$

Want something to optimize!

Good choice: how well our model fits the data, likelihood

X: training input variables, **Y**: training dependant variables

Want to learn **w** that models training data well

$L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = Pr(\mathbf{Y} \mid \mathbf{X}, \mathbf{w}) = \mathbf{\Pi}_i Pr(\mathbf{Y}_i \mid \mathbf{X}_i, \mathbf{w})$

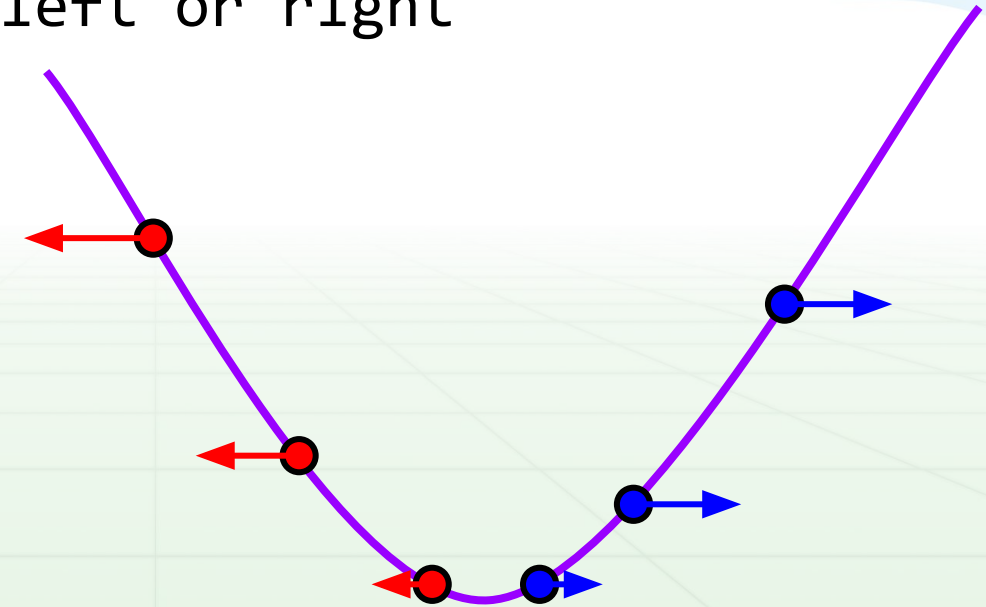
$\text{Log } L(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \Sigma_i \, [\mathbf{Y}_i \log(\sigma(\mathbf{w} \cdot \mathbf{X}_i)) + (1-\mathbf{Y}_i)\log(1 - \sigma(\mathbf{w} \cdot \mathbf{X}_i))]$

Derivative, set to 0? No closed form solution :-(

Gradient descent

For some loss function $L_{data}(\mathbf{w})$, gradient $\nabla L_{data}(\mathbf{w})$ points towards in direction of steepest ascent.

In 1d, either points left or right



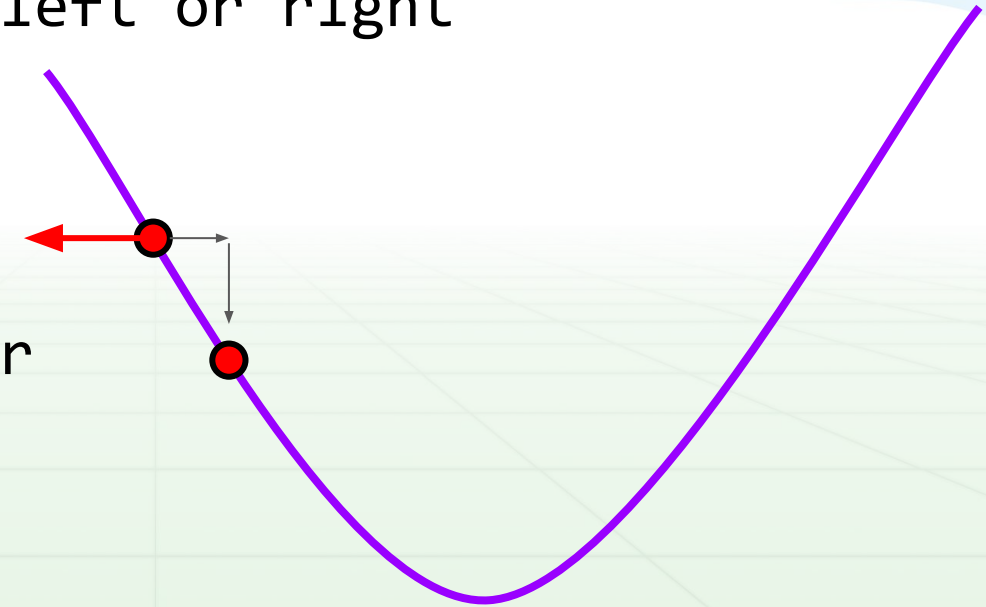
Gradient descent

For some loss function $L_{data}(\mathbf{w})$, gradient $\nabla L_{data}(\mathbf{w})$ points towards in direction of steepest ascent.

In 1d, either points left or right

Algorithm:

Take derivative
Move slightly in other direction
Repeat



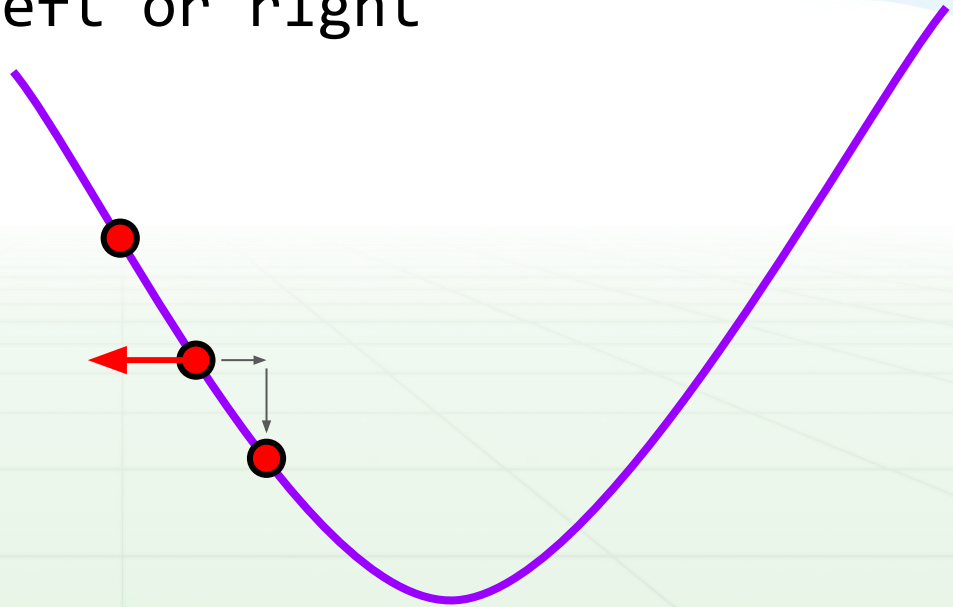
Gradient descent

For some loss function $L_{data}(\mathbf{w})$, gradient $\nabla L_{data}(\mathbf{w})$ points towards in direction of steepest ascent.

In 1d, either points left or right

Algorithm:

Take derivative
Move slightly in other direction
Repeat



Gradient descent

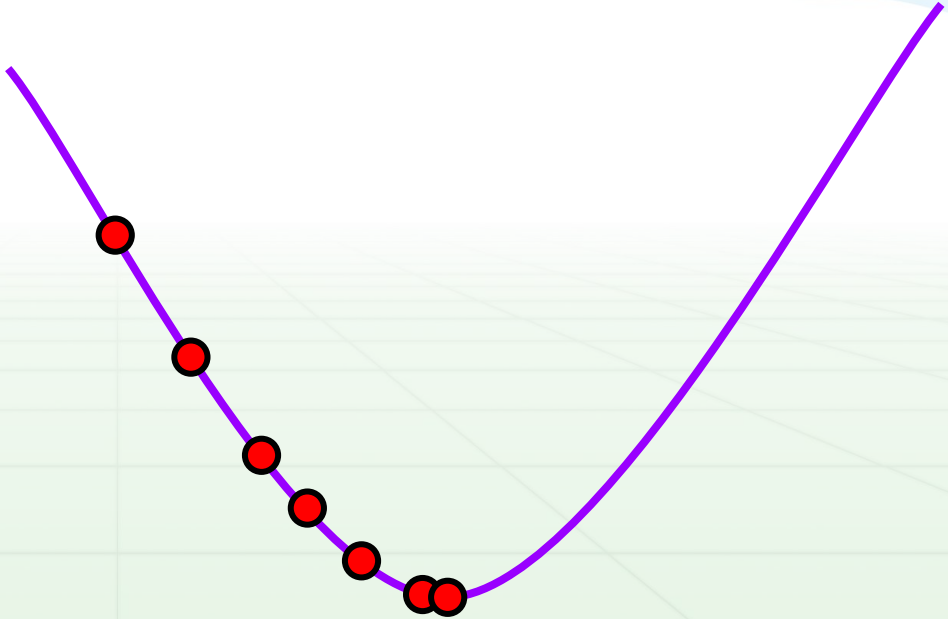
Algorithm:

Take derivative

Move slightly in other direction

Repeat

End up at local optima

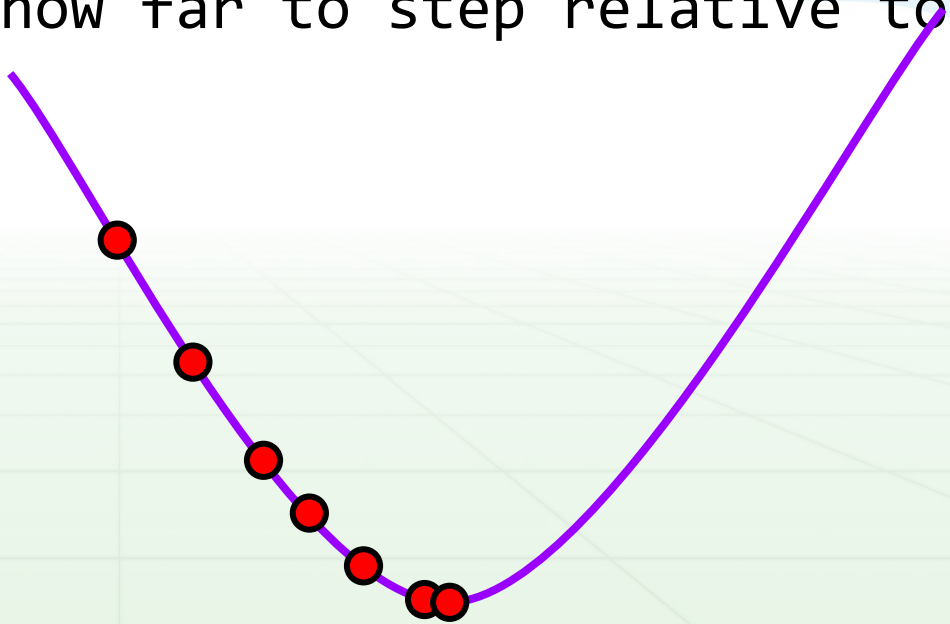


Gradient descent

Formally:

$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla L(\mathbf{w})$

Where η is *step size*, how far to step relative to the gradient



Stochastic gradient descent (SGD)

Estimate $\nabla L_{data}(\mathbf{w})$ with only some of the data

Before:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_i \nabla L_i(\mathbf{w}), \text{ for all } i \text{ in data}$$

Now:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_j \nabla L_j(\mathbf{w}), \text{ for some subset } j$$

Maybe even:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla L_k(\mathbf{w}), \text{ for some random } k$$

# of points used for update is called *batch size*

SGD with logistic regression

Say we want to do SGD with batch size = 1

Want to maximize $\log L(\mathbf{w} \mid \mathbf{X}_i, \mathbf{Y}_i)$ for random i

$= \mathbf{Y}_i \log(\sigma(\mathbf{w} \cdot \mathbf{X}_i)) + (1-\mathbf{Y}_i)\log(1 - \sigma(\mathbf{w} \cdot \mathbf{X}_i))$

$=$ … bunch of math …

$= \mathbf{Y}_i(\mathbf{w} \cdot \mathbf{X}_i) - \log(1+e^{(\mathbf{w} \cdot \mathbf{Xi})})$

Take derivatives:

$$\nabla \log L_i(\mathbf{w}) = \nabla \mathbf{Y}_i(\mathbf{w} \cdot \mathbf{X}_i) - \nabla \log(1+e^{(\mathbf{w} \cdot \mathbf{Xi})})$$

$$= \mathbf{Y}_i\mathbf{X}_i - 1/(1+e^{(\mathbf{w} \cdot \mathbf{Xi})}) * e^{(\mathbf{w} \cdot \mathbf{Xi})} * \mathbf{X}_i = \mathbf{X}_i[\mathbf{Y}_i - e^{(\mathbf{w} \cdot \mathbf{Xi})}/(1+e^{(\mathbf{w} \cdot \mathbf{Xi})})]$$

$$= \mathbf{X}_i[\mathbf{Y}_i - \sigma(\mathbf{w} \cdot \mathbf{X}_i)]$$

https://math.stackexchange.com/questions/477207/derivative-of-cost-function-for-logistic-regression

SGD with logistic regression

$\nabla \log L_i(\mathbf{w}) = \mathbf{X}_i[\mathbf{Y}_i - \sigma(\mathbf{w} \cdot \mathbf{X}_i)]$

So our weight update rule is:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta\, \mathbf{X}_i[\mathbf{Y}_i - \sigma(\mathbf{w}_t \cdot \mathbf{X}_i)]$$

Why plus?

Doing gradient ascent up the likelihood function


Chapter
Eleven
More ML for Vision

What if we have multiple classes?

Use an extension of logistic regression to multiple classes

For each class k we have weights $\mathbf{w}_k$

Want to predict probability distribution over classes, what's wrong with:

$\Pr(Y_i=1) = \sigma(\mathbf{w}_1 \cdot \mathbf{X}_i)$, $\Pr(Y_i=2) = \sigma(\mathbf{w}_1 \cdot \mathbf{X}_i)$, … etc

What if we have multiple classes?

Use an extension of logistic regression to multiple classes

For each class k we have weights $\mathbf{w}_k$

Want to predict probability distribution over classes, what's wrong with:

$Pr(Y_i=1) = \sigma(\mathbf{w}_1 \cdot \mathbf{X}_i)$, $Pr(Y_i=2) = \sigma(\mathbf{w}_1 \cdot \mathbf{X}_i)$, … etc

No normalization! Might sum to <> 1.

What if we have multiple classes?

What if we normalized logistic regression across classes?

Softmax!

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

If we have 2 classes and we assume $z_0 = 1$, $z_1 = \mathbf{w} \cdot \mathbf{X}$ then this is normal logistic regression.

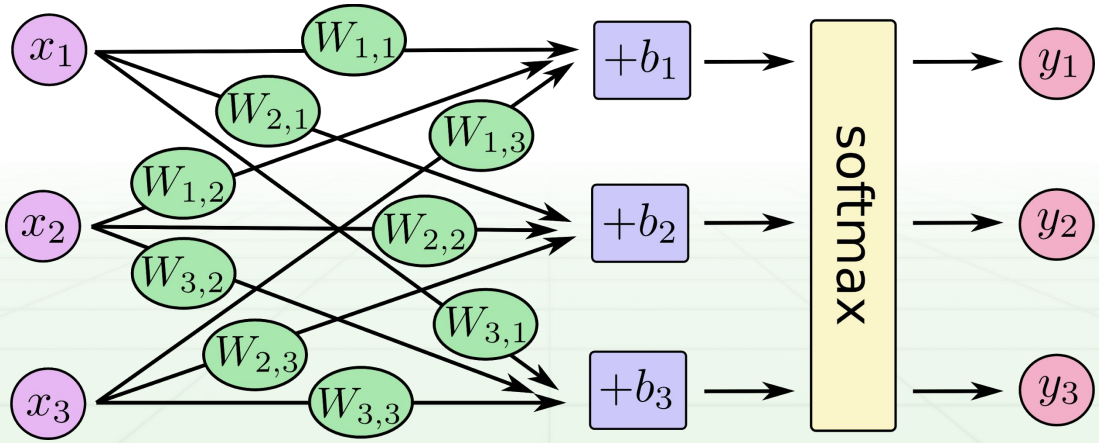
Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.



https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners

Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners

MNIST: Handwriting recognition

50,000 images of handwriting

28 x 28 x 1 (grayscale)

Numbers 0-9

10 class softmax regression

Input is 784 pixel values

Train with SGD

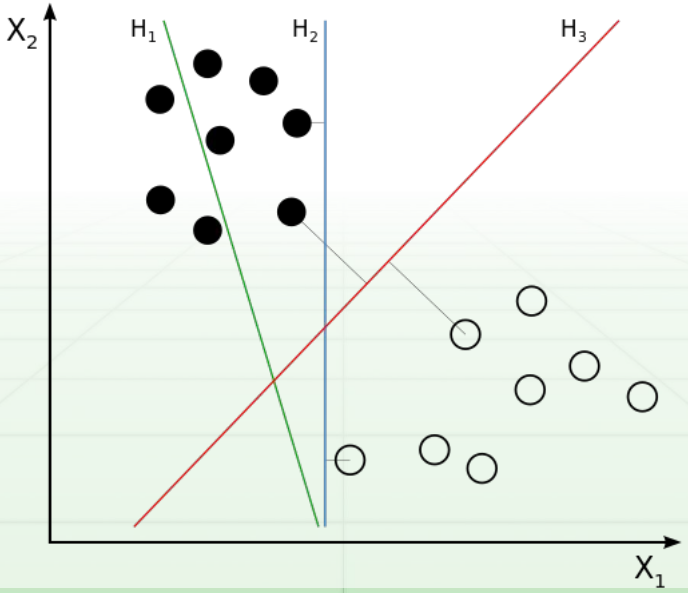
> 95% accuracy

Support Vector Machine (SVM)

Workhorse of old-school vision algorithms

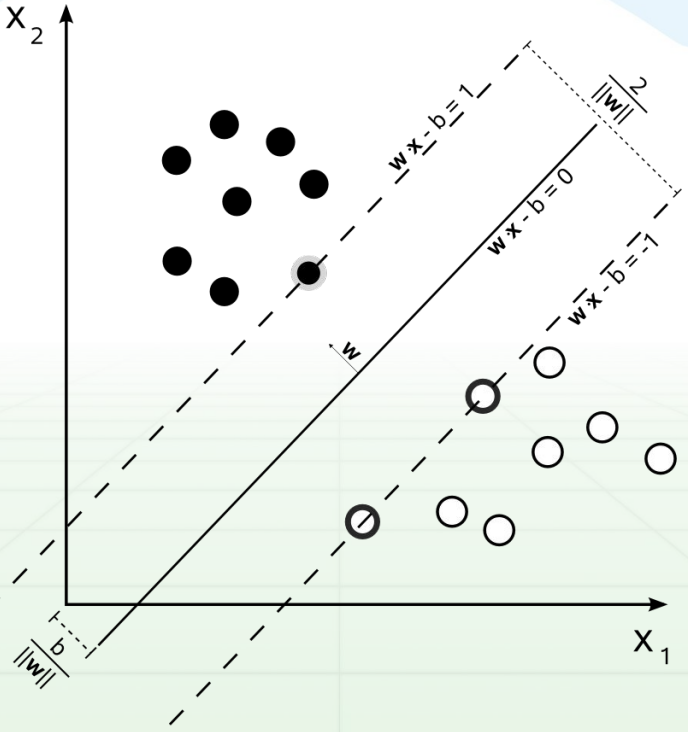
For some data, multiple linear classifiers

Maybe we want “best” one, maximum margin



Support Vector Machine (SVM)

Find max-margin classifier. Examples on the margin are supporting data points, support vectors.



Support Vector Machine (SVM)

If points are linearly separable, $y_n \in [-1, 1]$

$$\min ||\mathbf{w}||_2$$
$$\text{s.t. } y_n(\mathbf{w} \cdot \mathbf{x}_n - b) \geq 1, \; n = 1, 2 \ldots$$

Or: minimize weights such that margin for each point is at least 1

Sometimes not linearly separable. Introduce a slack variable so that misclassified points incur some penalty. Still a minimization problem.

Support Vector Machine (SVM)

If points are linearly separable, $y_n \in [-1, 1]$

$$\min ||\mathbf{w}||_2$$
$$\text{s.t. } y_n(\mathbf{w} \cdot \mathbf{x}_n - b) \geq 1, \; n = 1, 2 \ldots$$

Or: minimize weights such that margin for each point is at least 1

Sometimes not linearly separable. Introduce a slack variable so that misclassified points incur some penalty. Still a minimization problem.

Solve using “quadratic programming”, whatever that means...

Case study: Person detection

Dalal and Triggs ‘05:

Train SVM on HOG features of image

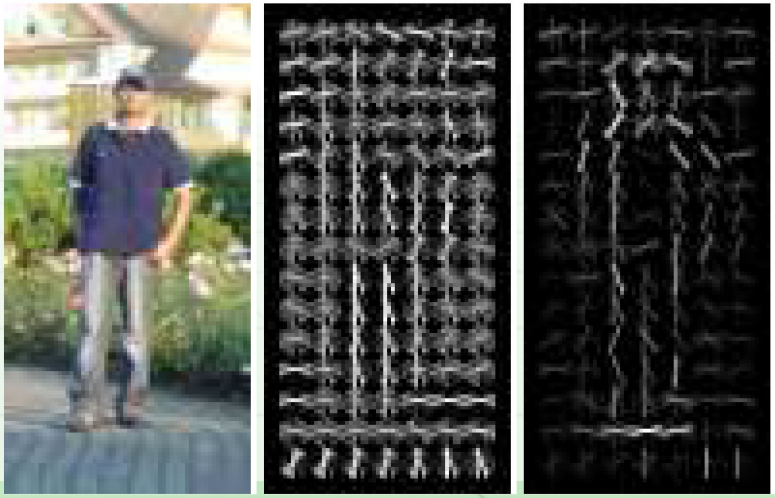
2 classes, person/not person

At test time:

Extract HOG features at many scales

Run SVM classifier at every location

High responses = person?

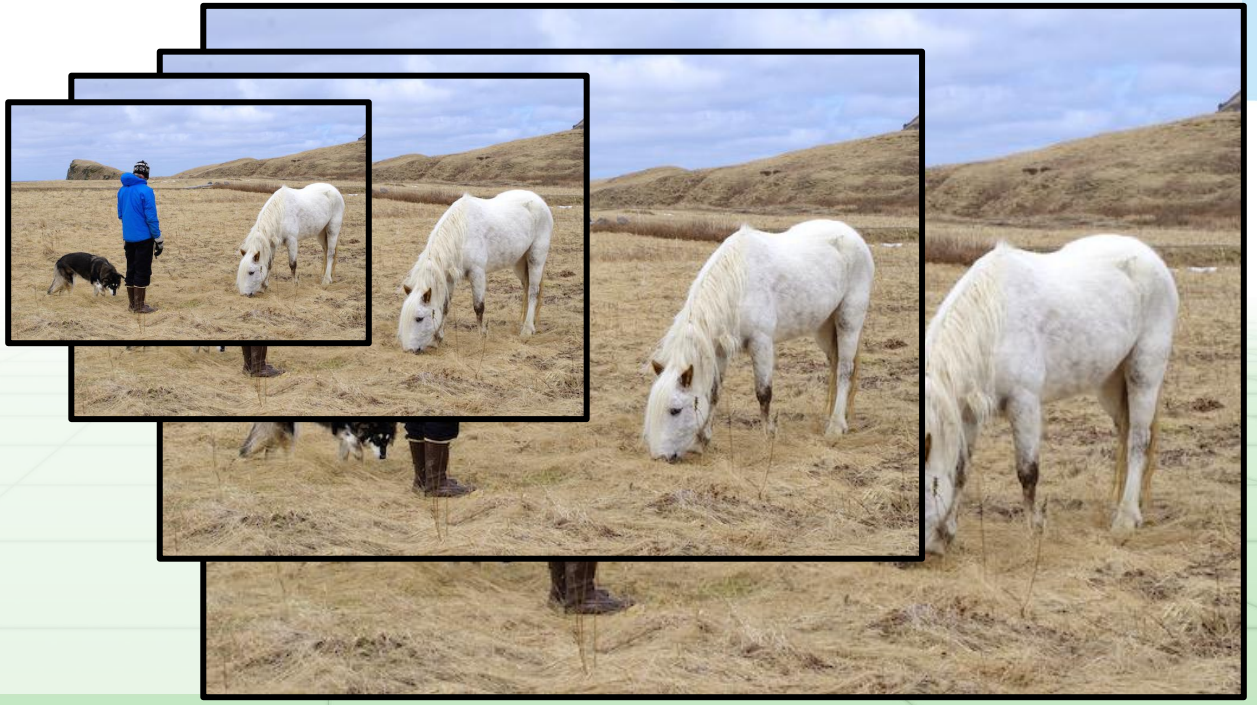


Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales

Every location



Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales

Every location

Person? No

Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales

Every location

Person? No

Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales

Every location

Person? No

Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales

Every location

Person? No

Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales

Every location

Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales

Every location

... many trials later

Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales
Every location

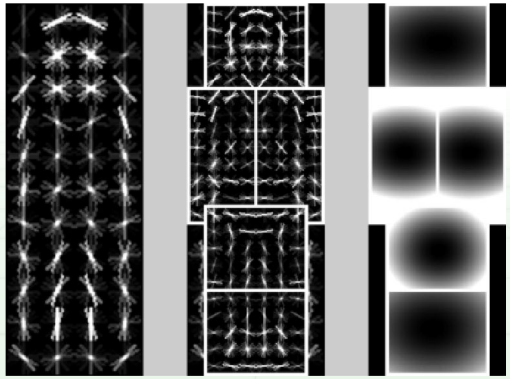
10k+ classifier evaluations per image.

Case study: Deformable parts models

Key idea: objects are made of parts

Sometimes those parts move!

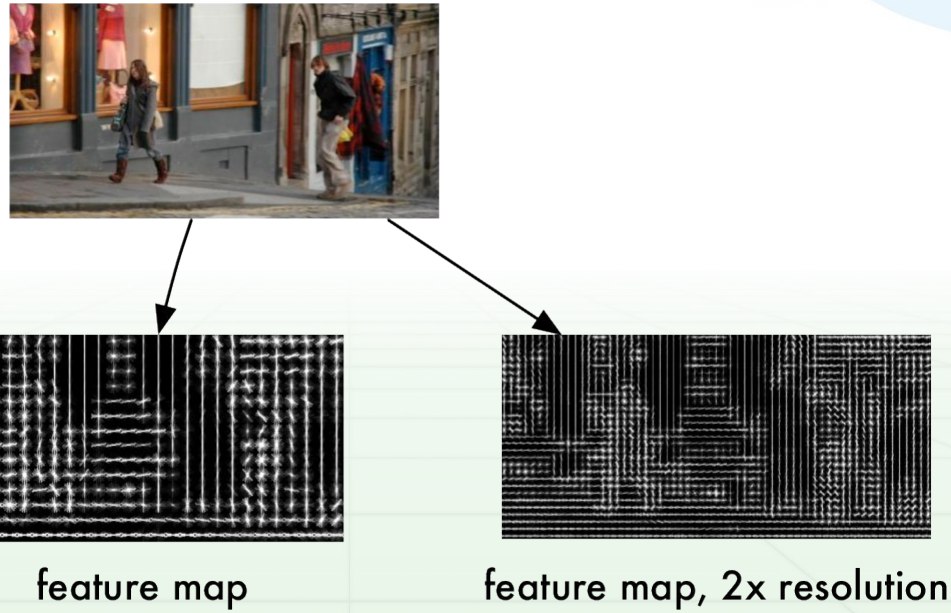
Train a model for base object but also for the parts and where they (generally) are on object



model

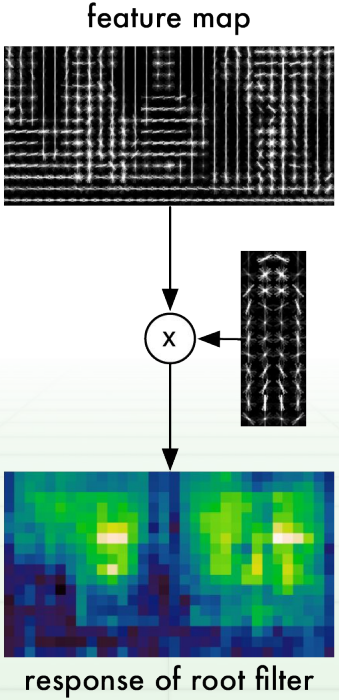
Case study: Deformable parts models

At test time, extract HOG features at variety of scales, normal and 2x resolution



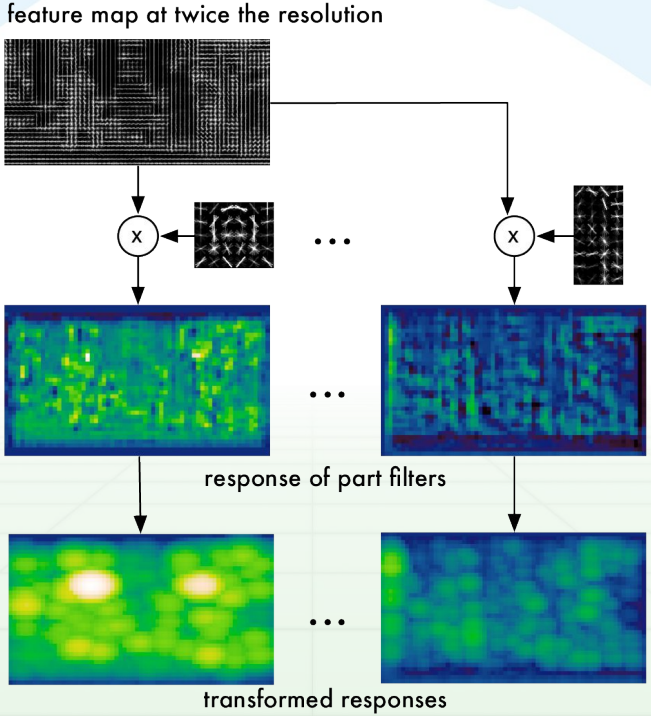
Case study: Deformable parts models

Run base model on feature map



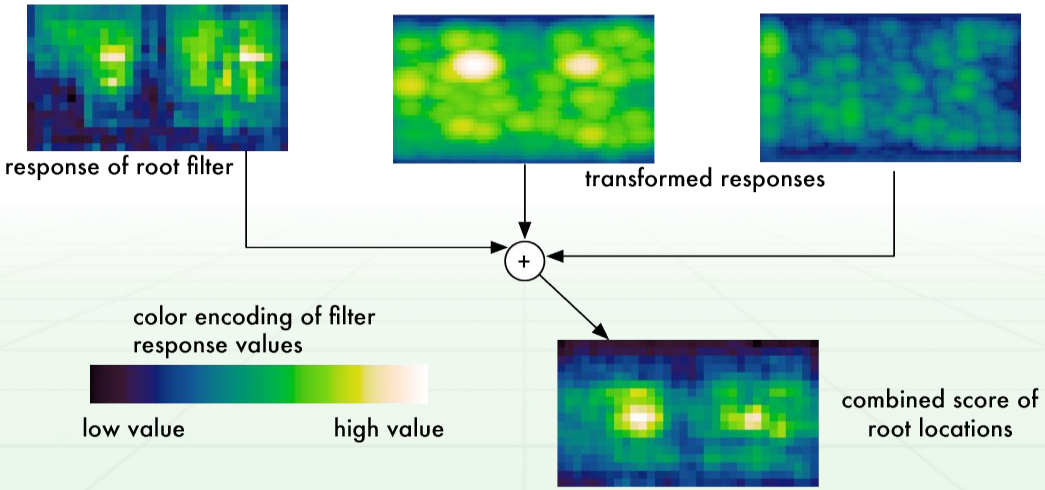
Case study: Deformable parts models

Run part models on 2x feature map



Case study: Deformable parts models

Combine output, weighted based on location of parts relative to base model



Case study: Deformable parts models

But part locations aren’t known!

Key idea: use Latent SVM to learn part locations and appearances

SVM simultaneously learns where parts are relative to object and how to recognize them!

Case study: Deformable parts models

But during training, MANY more negative than positive examples! Class imbalance

Key idea: Hard negative mining

Go over image, find areas that are misclassified as person, use those as “hard” negatives. Don’t train on easy background negatives.

Case study: Image classification

Given an image, what's in it?

Old state-of-the-art:

Extract features from image

    SIFT and Fisher Vectors

Train Linear SVM

On 1000 different classes, 54% accurate

https://lear.inrialpes.fr/~verbeek/mlcr.slides.11.12/sanchez11cvpr.pdf

What’s wrong with this?

Machine learning needs features!!

What are the right features?

HOG?

SIFT?

FV?

What’s wrong with this?

Machine learning needs features!!

What are the right features?

HOG?

SIFT?

FV?

Why not let the algorithm decide

What’s wrong with this?

Machine learning needs features!!

What are the right features?

HOG?

SIFT?

FV?

Why not let the algorithm decide

Neural networks: Feature extraction + linear model

Success of Neural Networks

Image classification:

54% -> 80% accuracy on 1000 classes

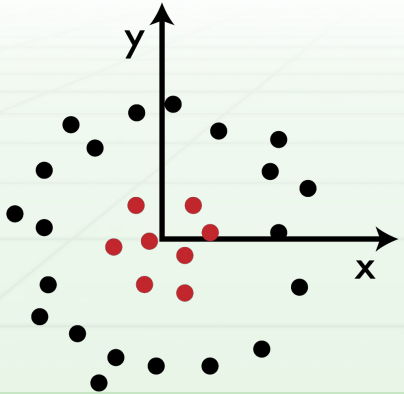
Object detection:

33% mAP (DPM) -> 88% mAP on 20 classes

What is feature engineering?

Arguably the **core** problem of machine learning (especially in practice)

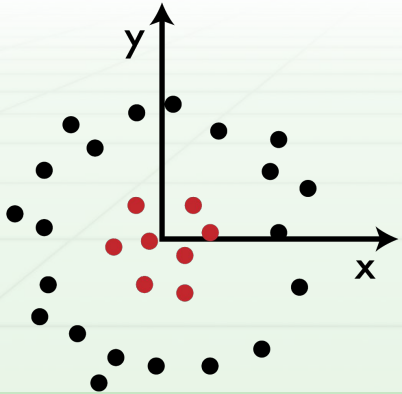
ML models work well if there is a clear relationship between the inputs and outputs of the function you are trying to model

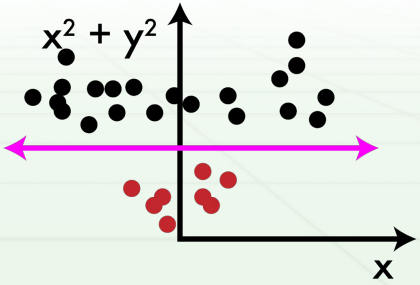


What is feature engineering?

Arguably the **core** problem of machine learning (especially in practice)

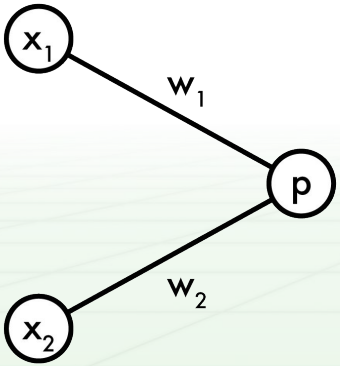
ML models work well if there is a clear relationship between the inputs and outputs of the function you are trying to model



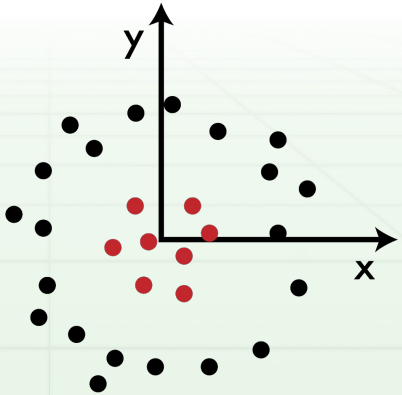


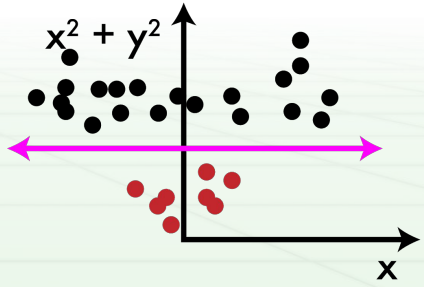
Linear model can’t do this

Cannot learn transformations of features, only use existing features. Human must create good features



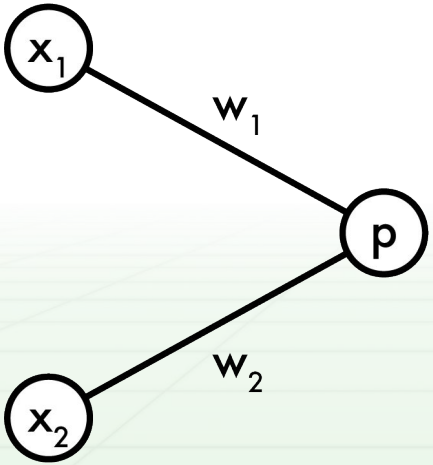
$$p = f(w_1x_1 + w_2x_2)$$





What if we added more processing?

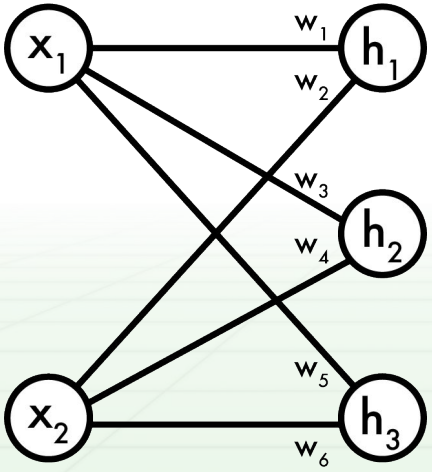
Generally, feature engineering is just coming up with combinations of the features we already have



$$p = f(w_1x_1 + w_2x_2)$$

What if we added more processing?

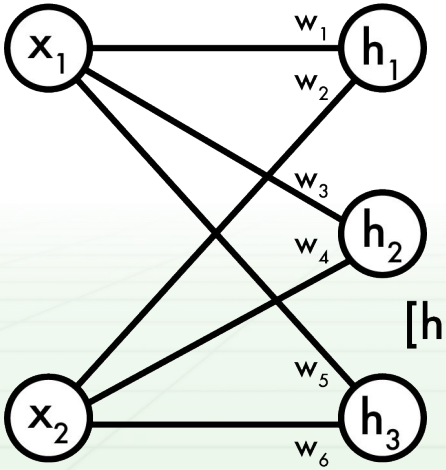
Create “new” features using old ones. We’ll call **H** our *hidden layer*



$h_1 = \varphi(w_1x_1 + w_2x_2)$
$h_2 = \varphi(w_3x_1 + w_4x_2)$
$h_3 = \varphi(w_5x_1 + w_6x_2)$

What if we added more processing?

As with linear model, **H** can be expressed in matrix operations



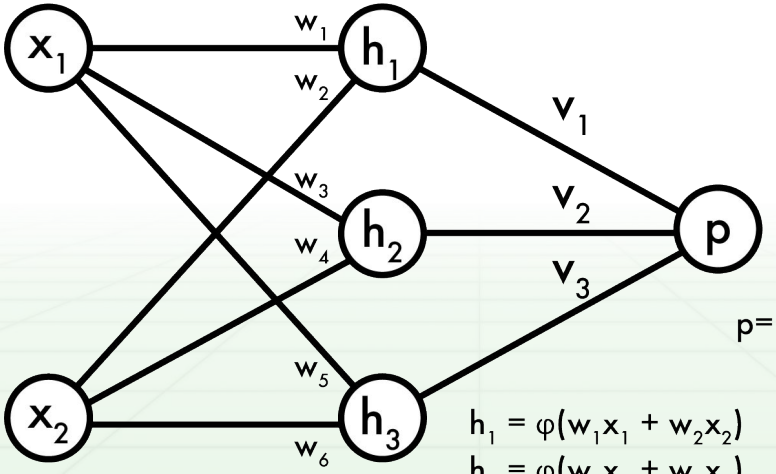
$$h_1 = \varphi(w_1 x_1 + w_2 x_2)$$
$$h_2 = \varphi(w_3 x_1 + w_4 x_2)$$
$$h_3 = \varphi(w_5 x_1 + w_6 x_2)$$

$$[h_1\ h_2\ h_3] = \varphi([x_1\ x_2]\begin{bmatrix} w_1 & w_3 & w_6 \\ w_2 & w_4 & w_5 \end{bmatrix})$$

$$\mathbf{H} = \varphi(\mathbf{X}\mathbf{w})$$

What if we added more processing?

Now our prediction p is a function of our hidden layer



$p = \varphi(v_1 h_1 + v_2 h_2 + v_3 h_3)$

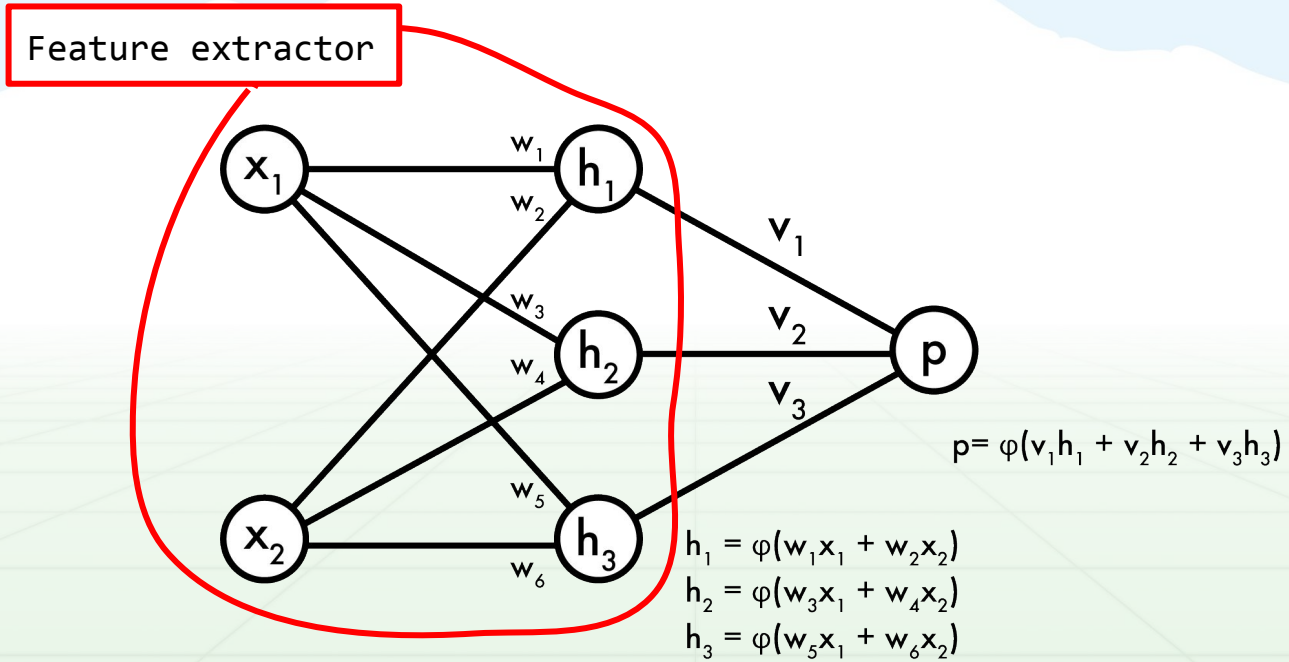
$h_1 = \varphi(w_1 x_1 + w_2 x_2)$

$h_2 = \varphi(w_3 x_1 + w_4 x_2)$

$h_3 = \varphi(w_5 x_1 + w_6 x_2)$

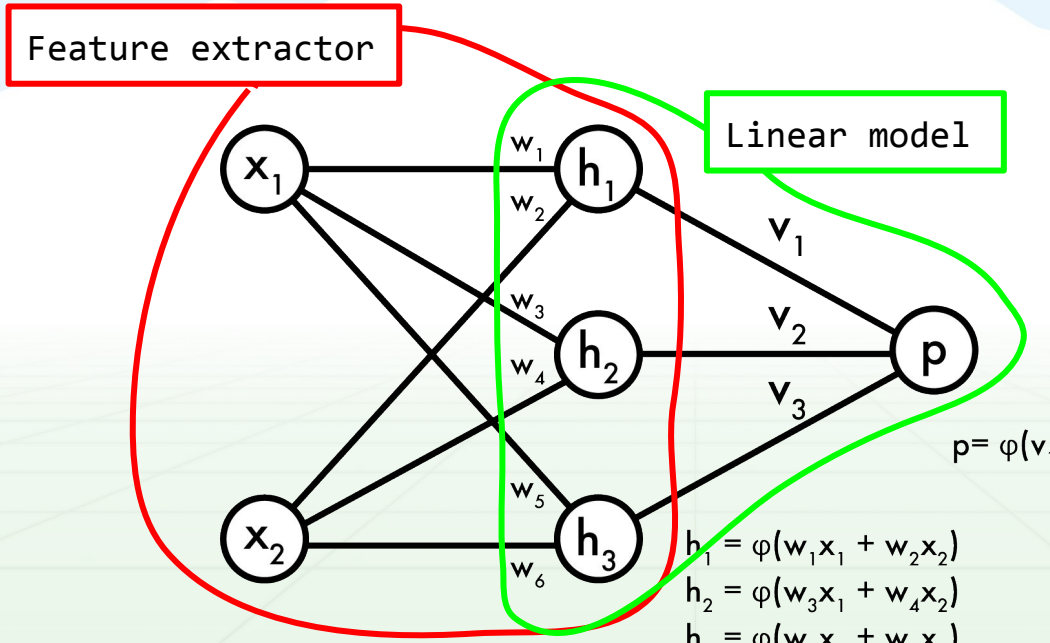
What if we added more processing?

Now our prediction p is a function of our hidden layer



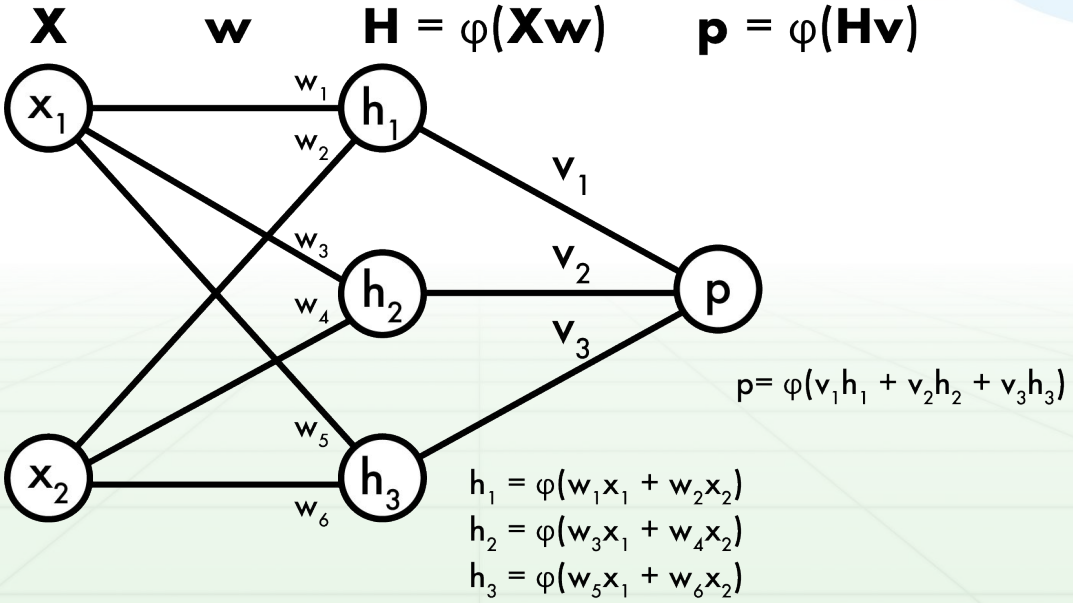
What if we added more processing?

Now our prediction p is a function of our hidden layer



What if we added more processing?

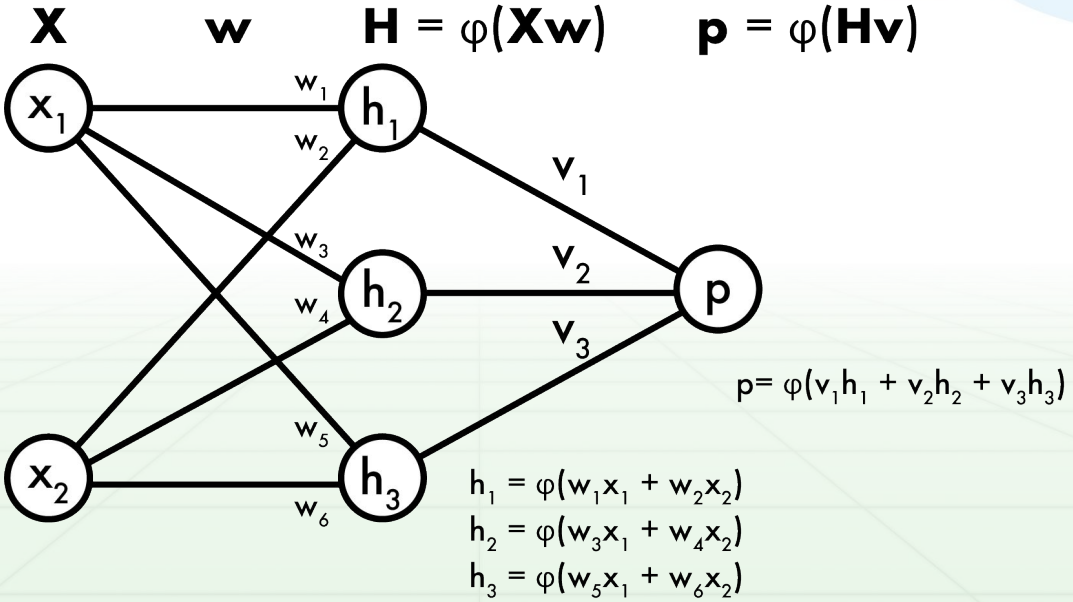
Can still express the whole process in matrix notation! Nice because matrix ops are fast



This is a neural network!

This one has 1 hidden layer, but can have **way** more

Each layer is just some function φ applied to linear combination of the previous layer

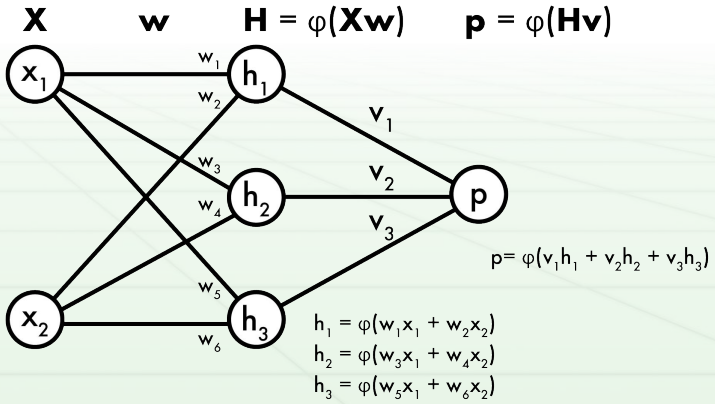


φ is our *activation function*

Want to apply some extra processing at each layer.

Why?

Imagine φ(x) = x, linear activation



φ is our *activation function*

Want to apply some extra processing at each layer.

Why?

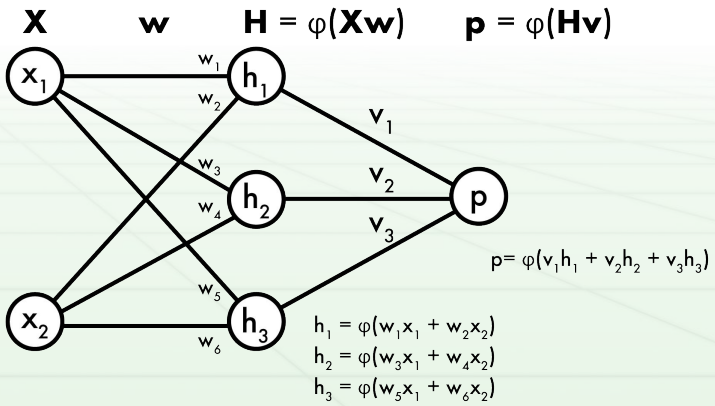
Imagine $\varphi(x) = x$, linear activation

$$p = v_1h_1 + v_2h_2 + v_3h_3$$

But $h_1 = x_1w_1 + x_2w_2$, $h_2 = \ldots$ etc

So

$$\begin{aligned} p &= v_1w_1x_1 + v_1w_2x_2 + v_2w_3x_1 + v_2w_4x_2 + v_3w_5x_1 + v_3w_6x_2 \\ &= (v_1w_1+v_2w_3+v_3w_5)x_1 + (v_1w_2+v_2w_4+v_3w_6)x_2 \\ &= u_1x_1 + u_2x_2 \end{aligned}$$



Universal approximation theorem

What if φ not linear?

Universal approximation theorem (Cybenko 89, Hornik 91)

φ: any nonconstant, bounded, monotonically increasing function

I_m: m-dimensional unit hypercube (interval [0-1] in m-d)

Then 1-layer neural network with φ as activation can model any continuous function f: I_m -> R

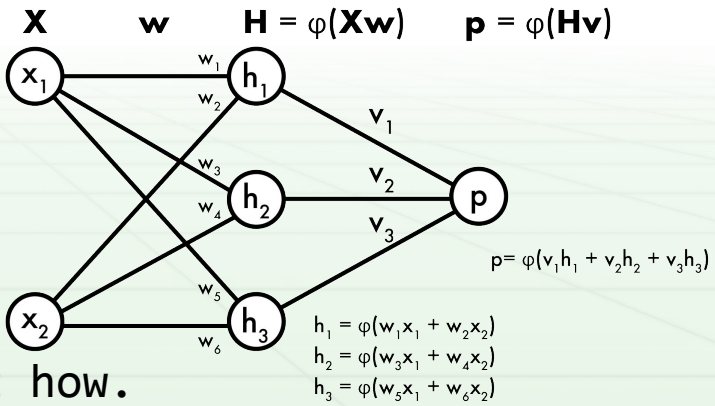
(no bound on size of hidden layer)

By extension, works on f: bounded R^m -> R

What can we learn? What can't we?

UAT just says it's possible to model, not how.

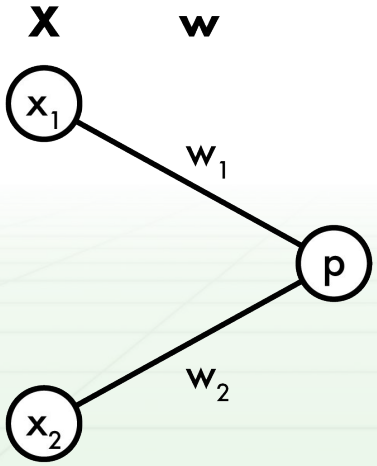
https://en.wikipedia.org/wiki/Universal_approximation_theorem



How do we learn it?

Neural networks are non-convex with no closed form solution (can't take derivative and set = 0)

Gradient descent! Recall for linear model:



$p = f(w_1x_1 + w_2x_2)$

$p = f(\mathbf{w} \cdot \mathbf{X})$

$\nabla \log L(\mathbf{w}) = \mathbf{X}[\mathbf{Y} - f(\mathbf{w} \cdot \mathbf{X})]$

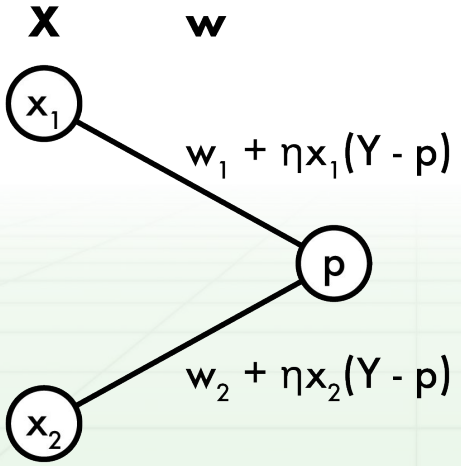
Weight update rule:

$\mathbf{w} = \mathbf{w} + \eta\, \mathbf{X}[\mathbf{Y} - f(\mathbf{w} \cdot \mathbf{X})]$

How do we learn it?

With gradient descent we calculate the partial derivatives of the loss (or likelihood) function for every weight: $\partial/\partial w_i \log L(\mathbf{w})$

Then do gradient descent (or ascent) by adding gradient to weight



$p = f(w_1x_1 + w_2x_2)$

$p = f(\mathbf{w}\cdot\mathbf{X})$

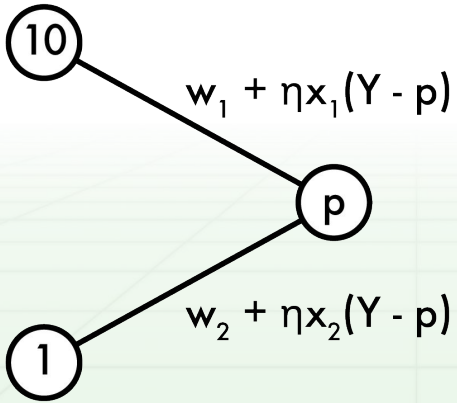
$\nabla \log L(\mathbf{w}) = \mathbf{X}[\mathbf{Y} - f(\mathbf{w}\cdot\mathbf{X})]$

Weight update rule:

$\mathbf{w} = \mathbf{w} + \eta\, \mathbf{X}[\mathbf{Y} - f(\mathbf{w}\cdot\mathbf{X})]$

How do we learn it?

Simple example, say we have a data point [10, 1] and we predict some p. We also know the “correct” label Y. Maybe our prediction p is too small and we want to make it larger. How do we adjust **w**?



Want p to be larger...

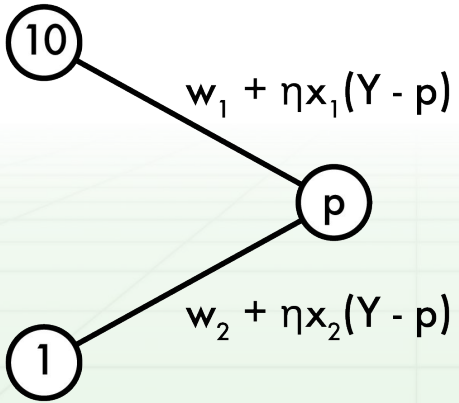
(Y - p) is positive

How do we change w_1 and w_2?

How do we learn it?

Simple example, say we have a data point [10, 1] and we predict some p. We also know the "correct" label Y. Maybe our prediction p is too small and we want to make it larger. How do we adjust **w**?

We adjust w_1 much more than w_2, why?



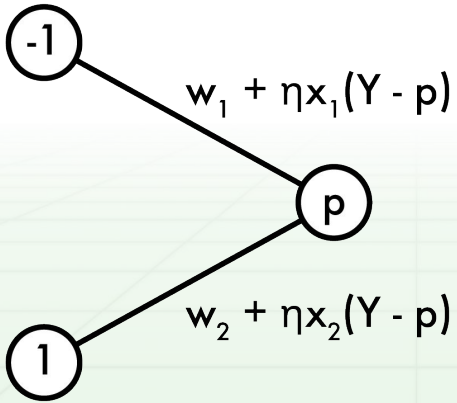
Want p to be larger...

(Y - p) is positive

How do we change w_1 and w_2?

How do we learn it?

Simple example, say we have a data point [-1, 1] and we predict some p. We also know the "correct" label Y. Maybe our prediction p is too small and we want to make it larger. How do we adjust **w**?



Want p to be larger...

(Y - p) is positive

How do we change w_1 and w_2?