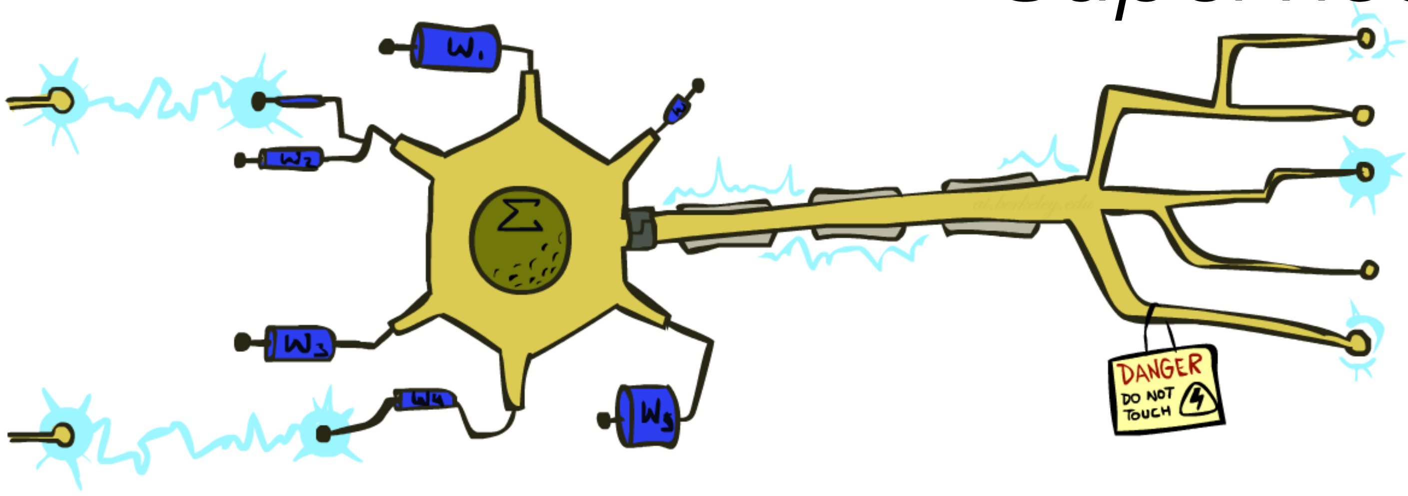
CS 4100 // artificial intelligence

instructor: byron wallace

Supervised learning 2



Attribution: many of these slides are modified versions of those distributed with the *UC Berkeley CS188* materials
Thanks to John DeNero and Dan Klein

Also, some of the Neural Network slides here are derived from Ray Mooney's.

Last time: Naïve Bayes

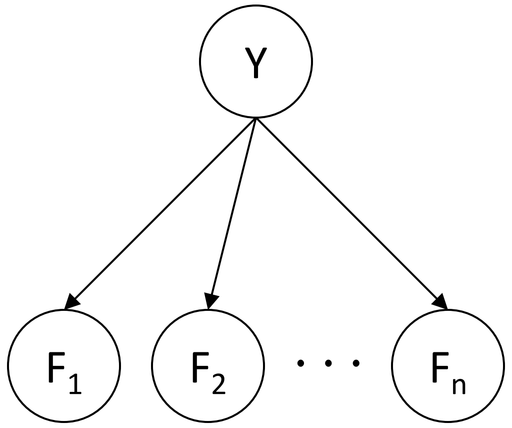
A general Naive Bayes model:

|Y| parameters

$$P(\mathsf{Y}, \mathsf{F}_1 \ldots \mathsf{F}_n) = \quad P(\mathsf{Y}) \prod_i P(\mathsf{F}_i | \mathsf{Y})$$

|Y| x $|F|^n$ values

n x |F| x |Y| parameters



- We only have to specify how each feature depends on the class
- Total number of parameters is linear in n
- Model is very simplistic, but often works anyway

Last time: Naïve Bayes

Naïve Bayes is a **generative** model

Estimates $P(X,y)$

Today we'll introduce a **discriminative** approach

Estimates $P(y|X)$

Last time: Spam v ham

Last time: Spam v ham

✗

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

✓

Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

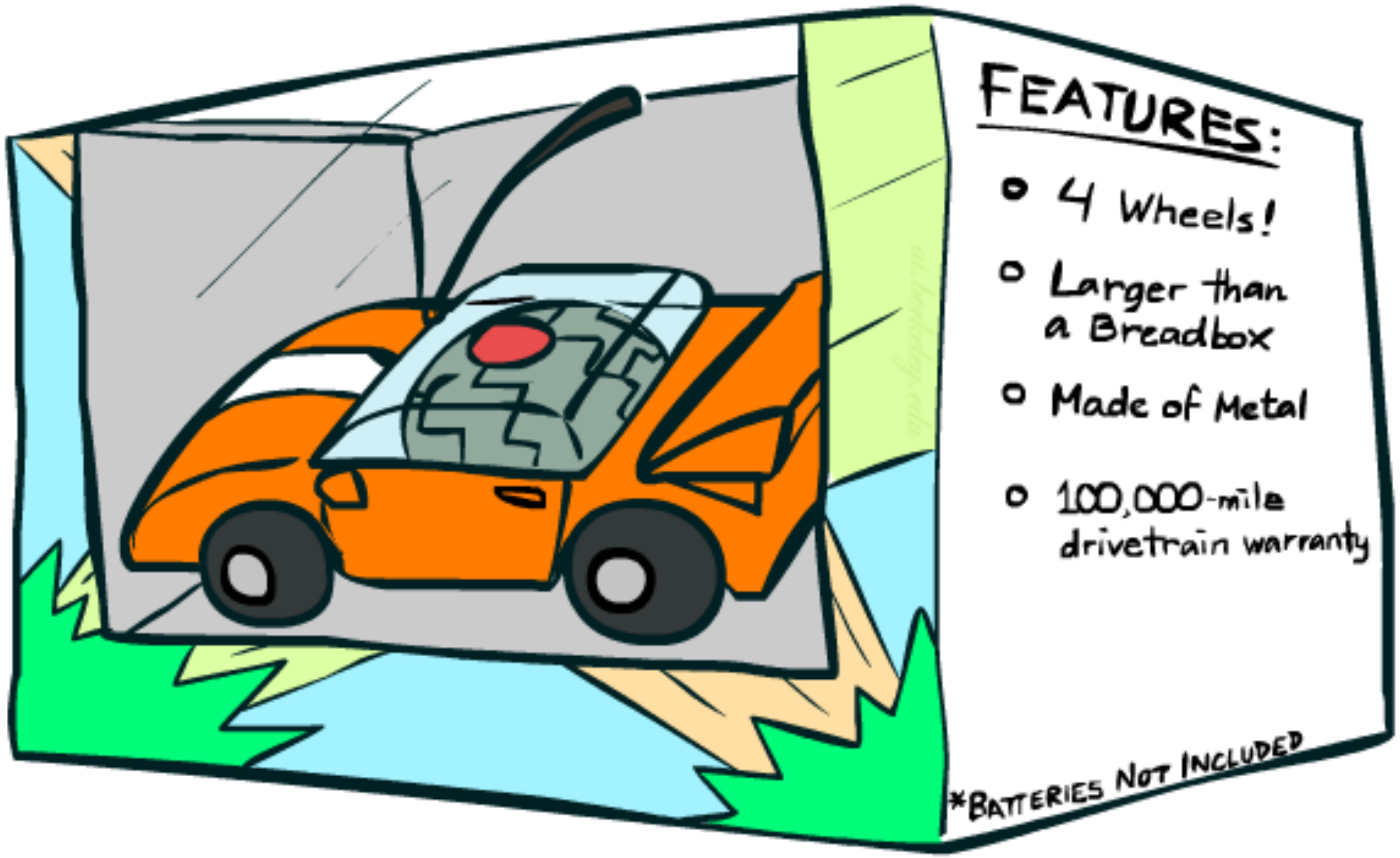
Errors

Examples of errors (words are not always enough!)

```
Dear GlobalSCAPE Customer,

GlobalSCAPE has partnered with ScanSoft to offer you the
latest version of OmniPage Pro, for just $99.99* - the regular
list price is $499! The most common question we've received
about this offer is - Is this genuine? We would like to assure
you that this offer is authorized by ScanSoft, is genuine and
valid. You can get the . . .
```

```
. . . To receive your $30 Amazon.com promotional certificate,
click through to

  http://www.amazon.com/apparel

and see the prominent link for the $30 offer. All details are
there. We hope you enjoyed receiving this message. However, if
you'd rather not receive future e-mails announcing new store
launches, please click . . .
```

Features


FEATURES:
4 Wheels!
Larger than a Breadbox
Made of Metal
100,000-mile drivetrain warranty
*BATTERIES NOT INCLUDED

What to do about errors?

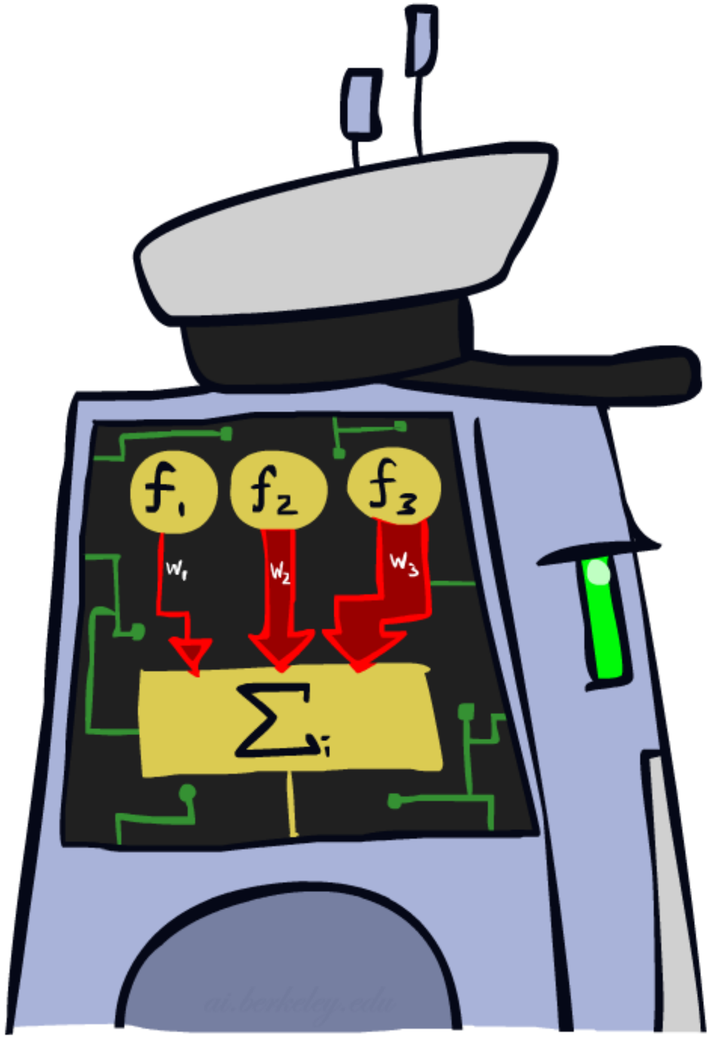
Need more *features*– words aren't enough!

- Have you emailed the sender before?
- Have 1K other people just gotten the same email?
- Is the sending information consistent?
- Is the email in ALL CAPS?
- Do inline URLs point where they say they point?
- Does the email address you by (your) name?

Can add these information sources as new variables in the NB model; but this isn't always natural or easy in "generative" models like NB

Today we will discuss models that make it easy to add arbitrary features

Linear classifiers

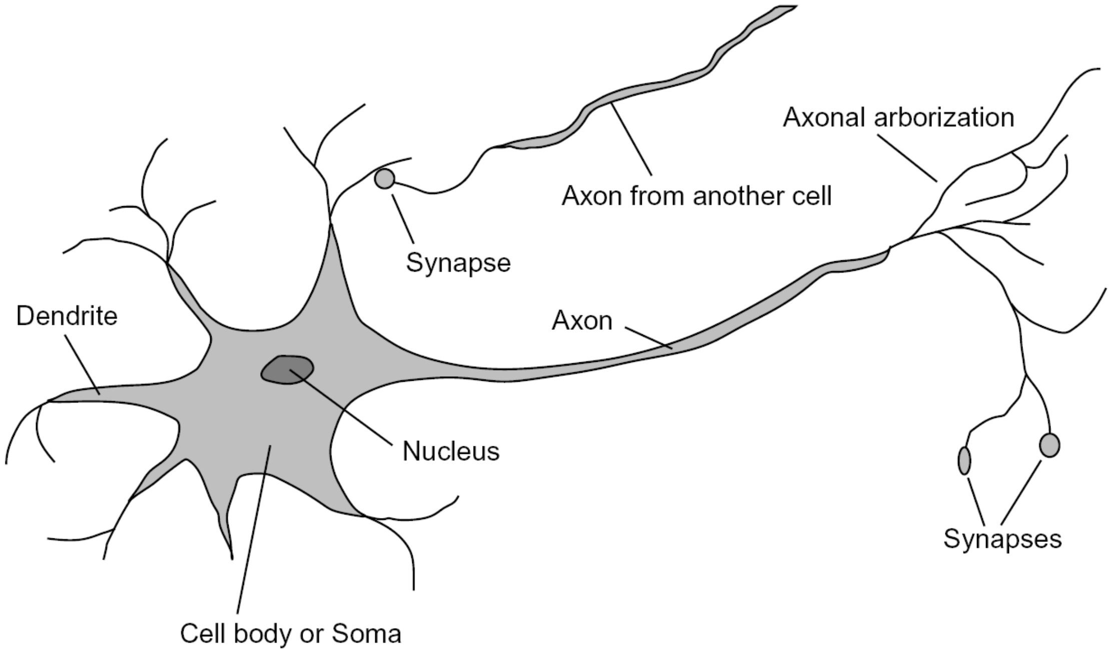


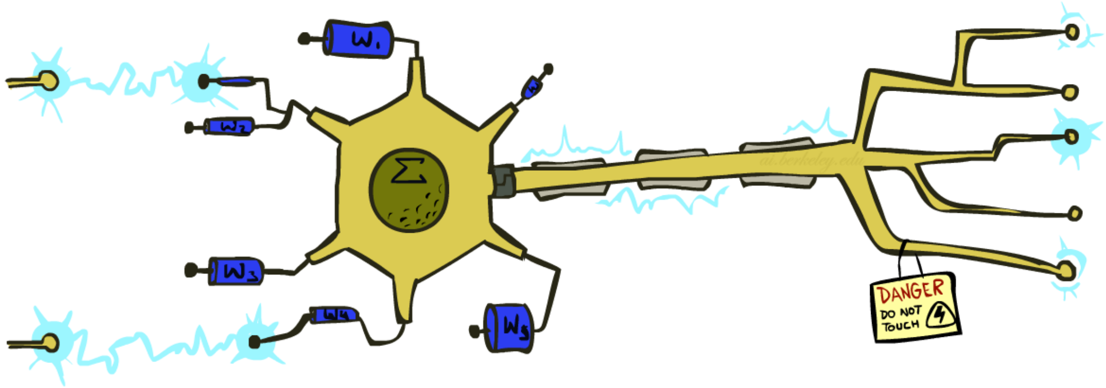
Feature vectors

x	$f(x)$	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"

“Neural” models: where the name comes from

- Very loose inspiration: human neurons



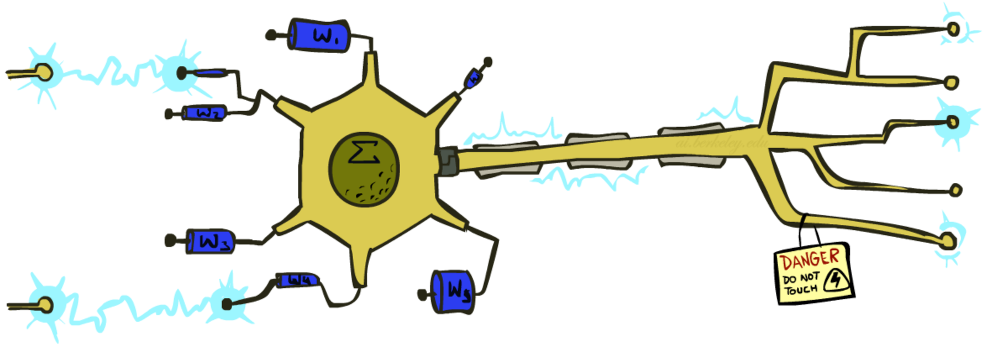


Linear classifiers

Inputs are **feature values**

Each feature has a **weight**

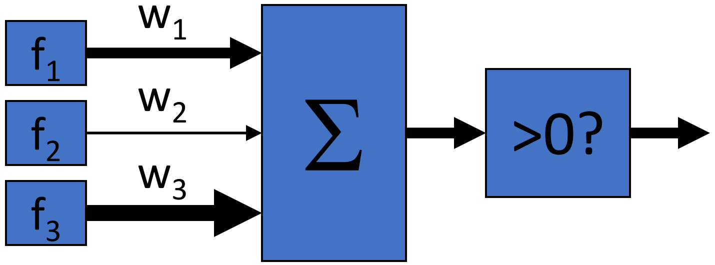
Sum is the **activation**



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

If the activation is:

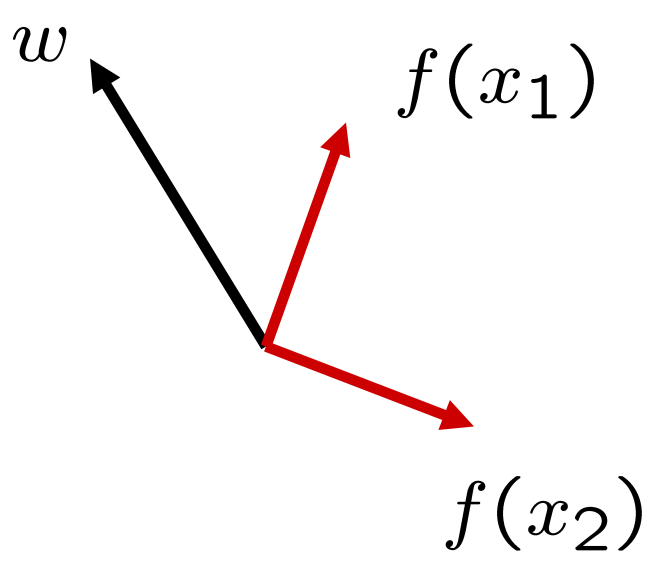
- Positive, output +1
- Negative, output -1



Weights

- Binary case: compare features to a weight vector
- *Learning*: figure out the weight vector from examples

```
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```

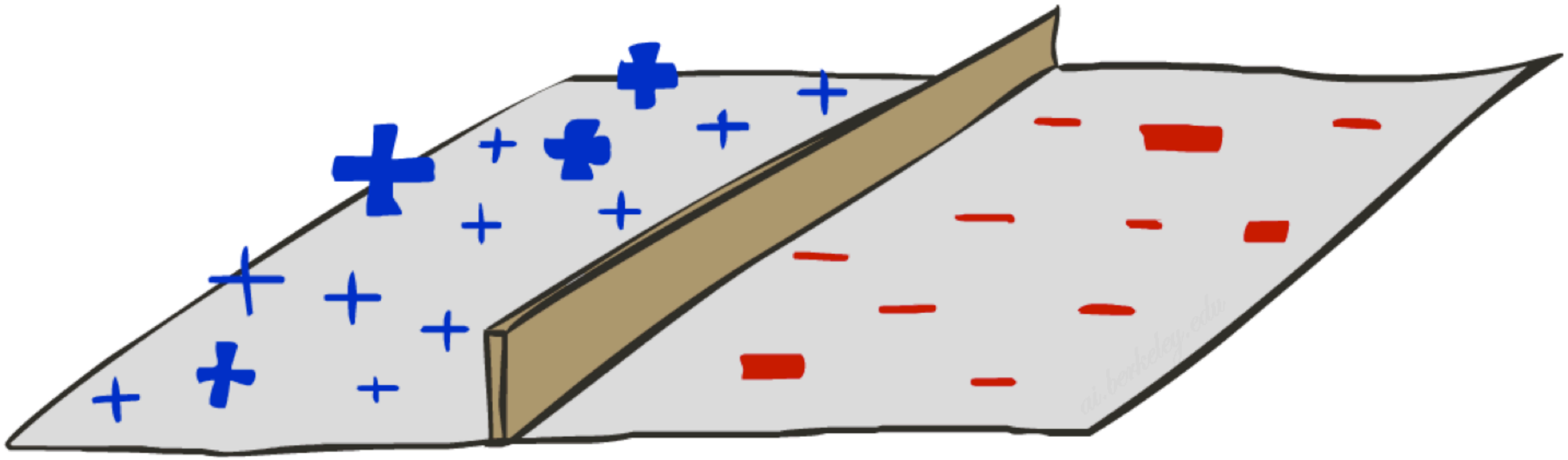



```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

```
# free       : 0
YOUR_NAME    : 1
MISSPELLED   : 1
FROM_FRIEND  : 1
...
```

Dot product $w \cdot f$ positive $\longrightarrow$ the positive class

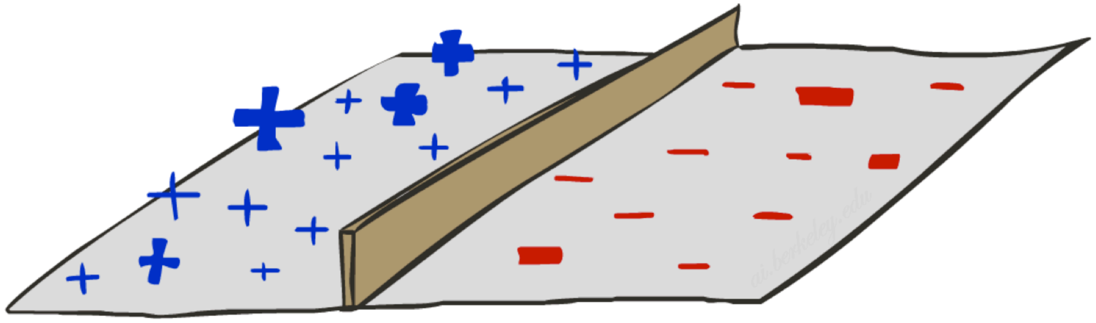
Decision rules



Binary decision rule

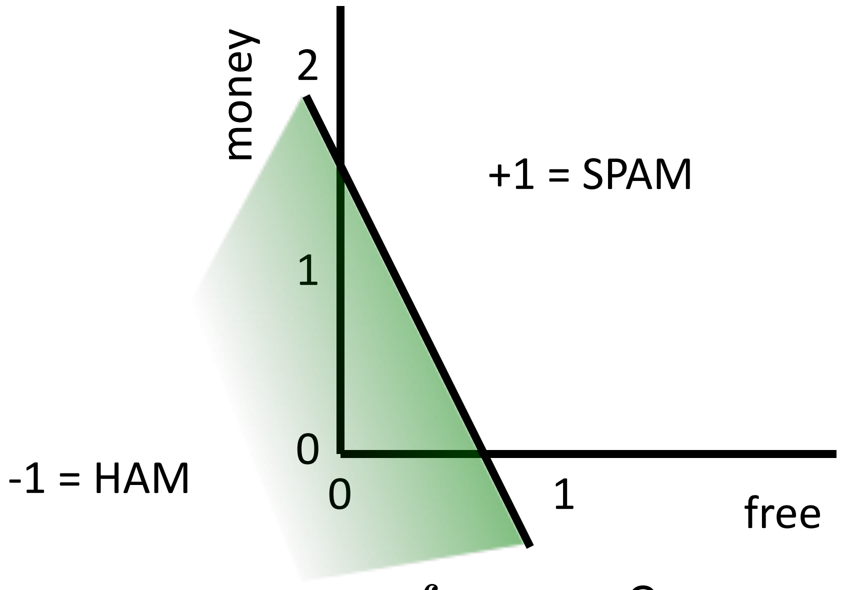
In the space of feature vectors

- Examples are points
- Any weight vector is a hyperplane
- One side corresponds to Y=+1
- Other corresponds to Y=-1



$$w$$

```
BIAS  : -3
free  :  4
money :  2
...
```



$$f \cdot w = 0$$

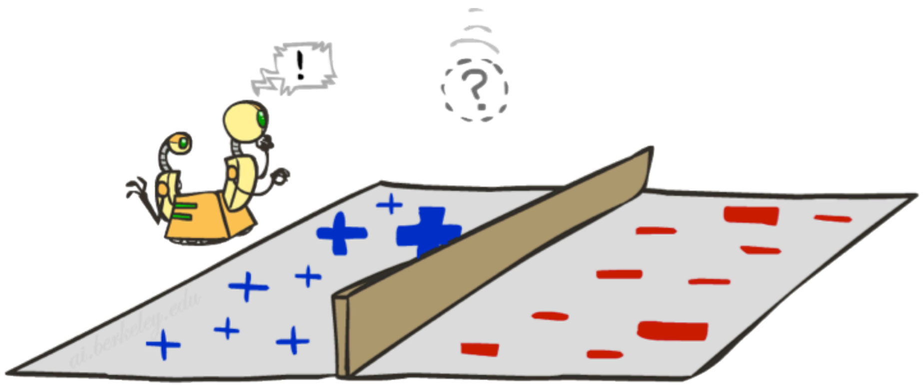
Weight updates

Learning: Binary perceptron

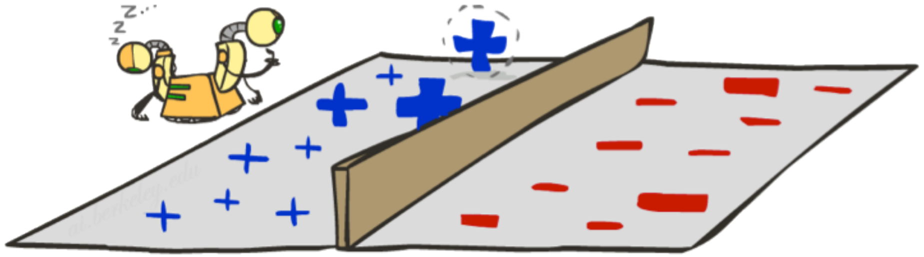
Start with weights = 0

For each training instance:

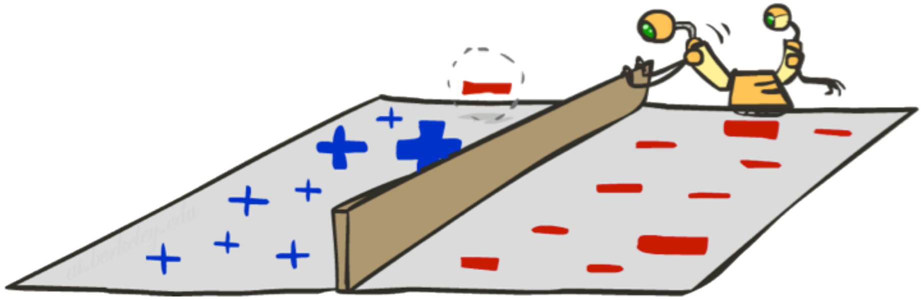
- Classify with current weights



- If correct (i.e., y=y*), no change!



- If wrong: adjust the weight vector



Learning: Binary perceptron

Start with weights = 0

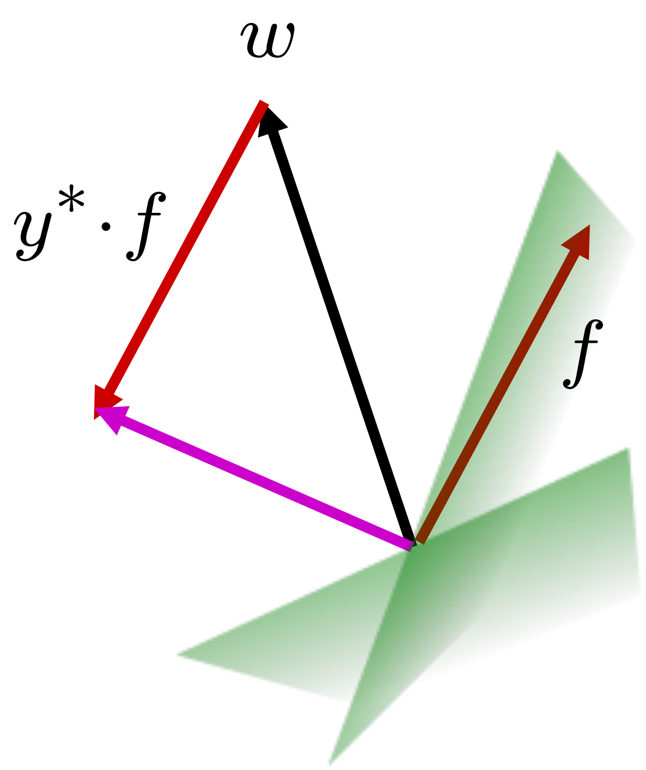
For each training instance:

- Classify with current weights

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

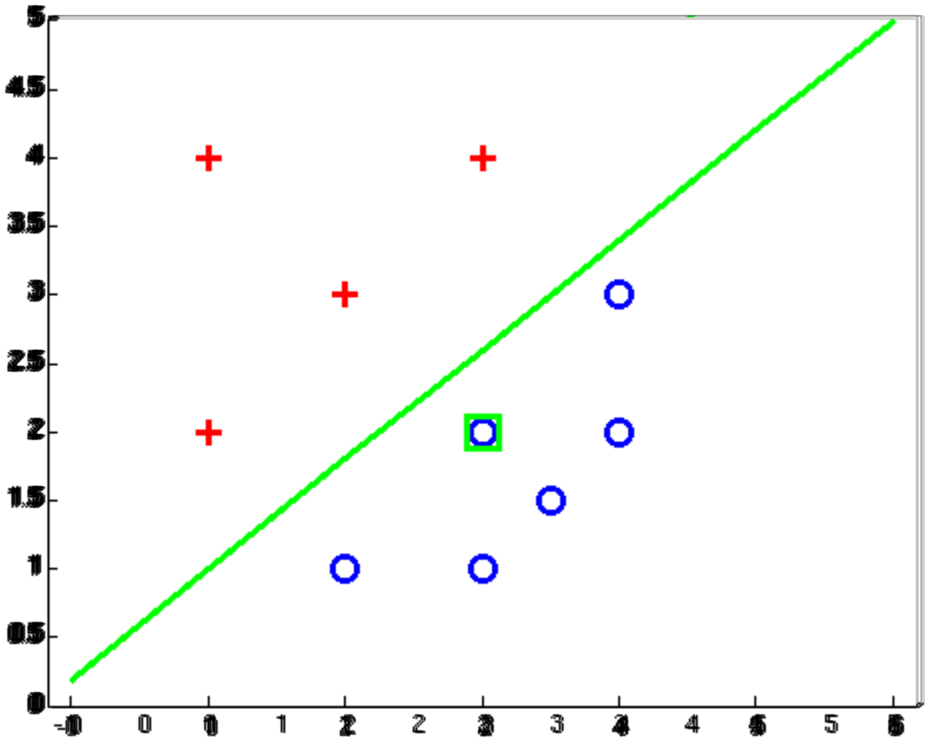
- If correct (i.e., y=y*), no change!
- If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

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



Examples: Perceptron

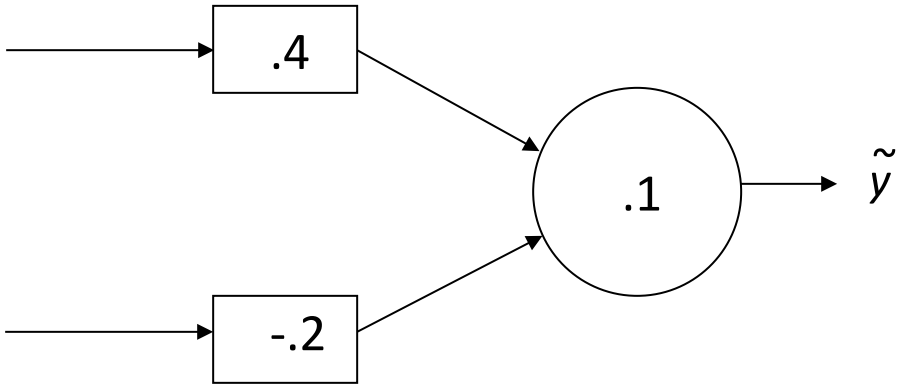
Separable Case



Perceptron

			y
$\boldsymbol{x}_1$	.8	.3	1
$\boldsymbol{x}_2$	.4	.1	-1

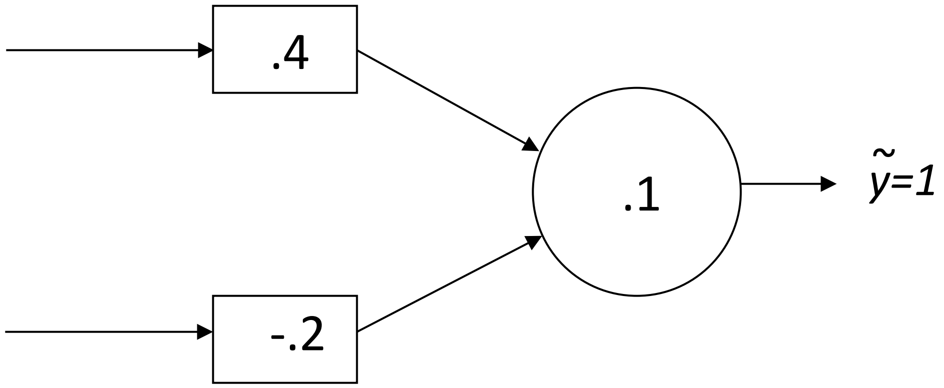
Training data



Perceptron

			y
$\boldsymbol{x}_1$	.8	.3	1
$\boldsymbol{x}_2$	.4	.1	-1

Training data

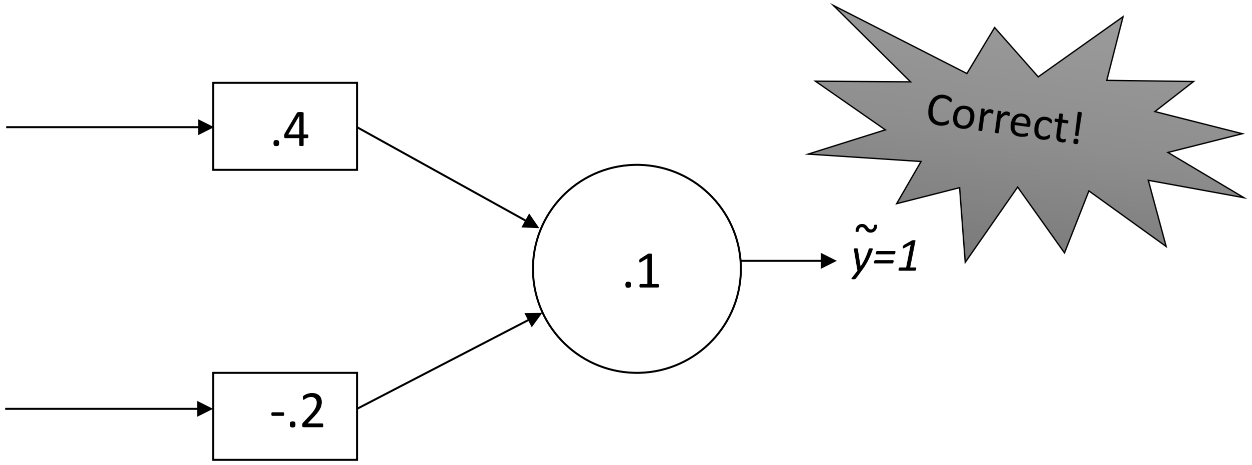


net = .8*.4 + .3*-.2 = .26

Perceptron

			y
$\boldsymbol{x}_1$	.8	.3	1
$\boldsymbol{x}_2$	.4	.1	-1

Training data

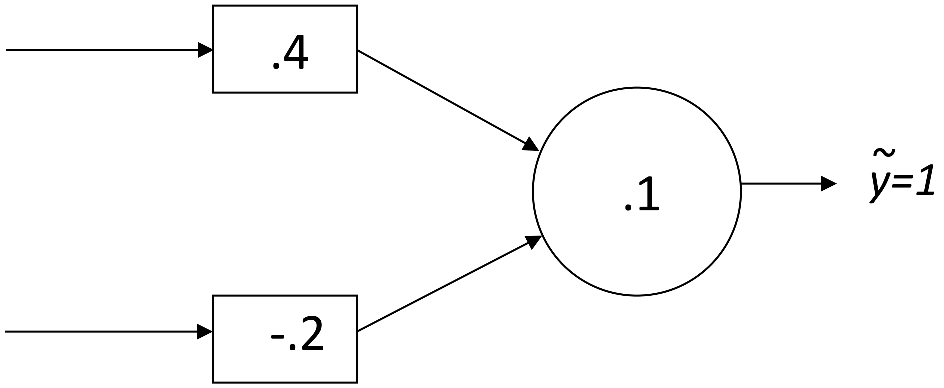


net = .8*.4 + .3*-.2 = .26

Perceptron

			y
$\boldsymbol{x}_1$	.8	.3	1
$\boldsymbol{x}_2$	.4	.1	-1

Training data

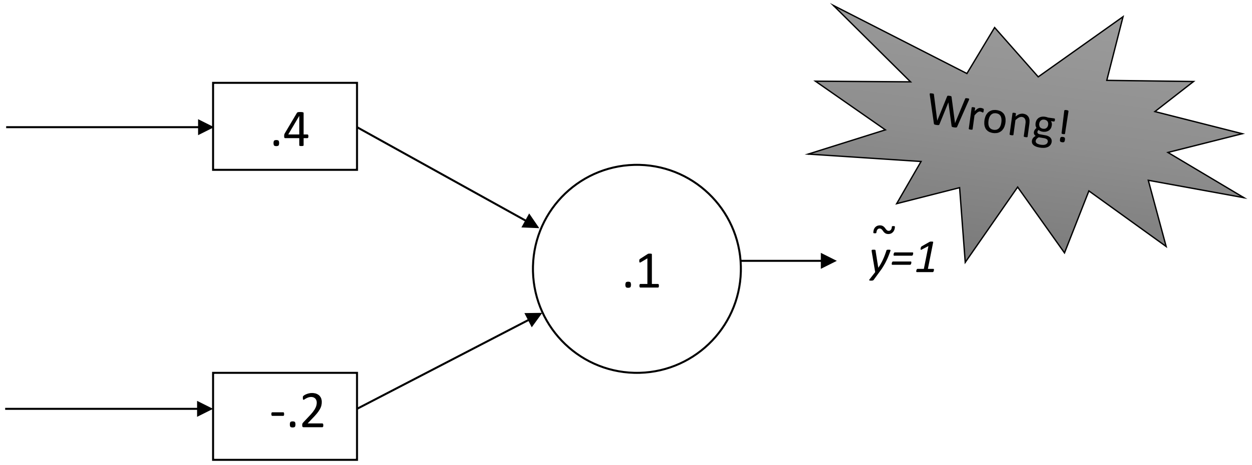


net = .4*.4 + .1*-.2 = .14

Perceptron

			y
$\boldsymbol{x}_1$	.8	.3	1
$\boldsymbol{x}_2$	.4	.1	-1

Training data



$$net = .4*.4 + .1*-.2 = .14$$

Perceptron: updating

prediction

true label

$$\Delta w_j = \alpha\, (y - \tilde{y}) * x_{ij}$$

‘learning weight’

Perceptron: updating

prediction

true label

$$\Delta w_j = \alpha (y - \tilde{y}) * x_{ij}$$

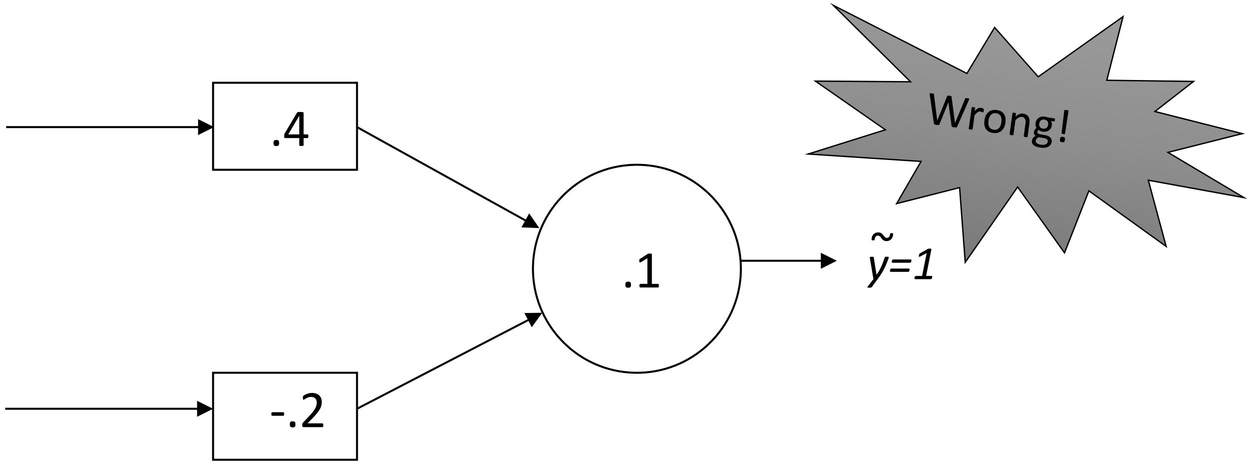
‘learning weight’

$$\boldsymbol{w} = \boldsymbol{w} + \Delta \boldsymbol{w}$$

Perceptron: updating

			y
$\boldsymbol{x}_1$	.8	.3	1
$\boldsymbol{x}_2$	.4	.1	-1

Training data



net = .4*.4 + .1*-.2 = .14

$\tilde{y}=1$; $y=-1$. assume $\alpha = .5$

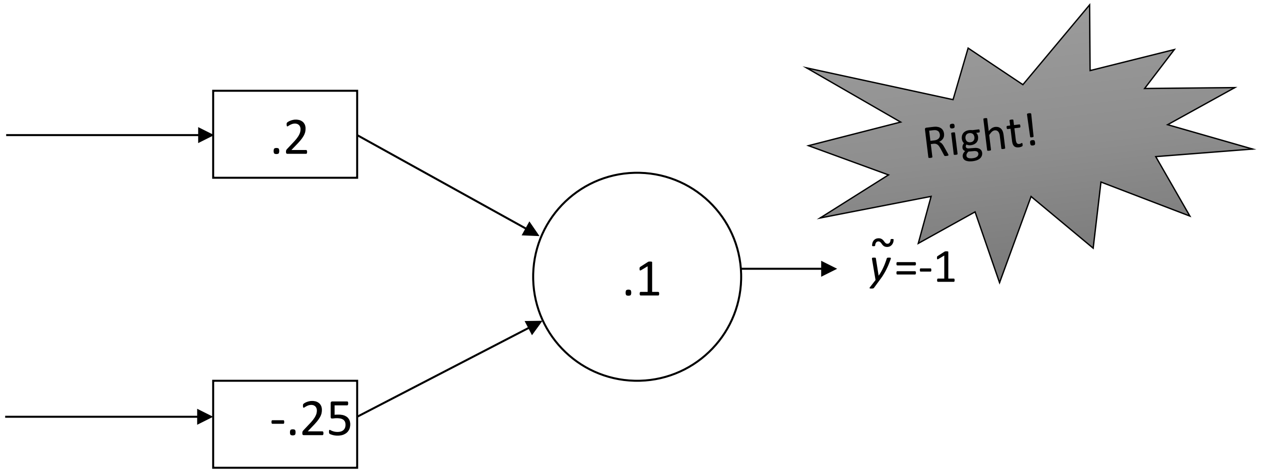
$\mathbf{w}^t = [.4, -.2]$

$$\mathbf{w}^{t+1} = [.4, -.2] + .5\,(0 - 1)\,\mathbf{x}_2$$
$$= [.4, -.2] - .5 * [.4, .1]$$
$$= [.2, -.25]$$

Perceptron

			y
$\boldsymbol{x}_1$	.8	.3	1
$\boldsymbol{x}_2$	.4	.1	0

Training data



net = .4*.2 + .1*-.25 = .055

$\mathbf{w}^{t+1}$ = [.2, -.25]

Multiclass decision rule

If we have multiple classes

- A weight vector for each class:

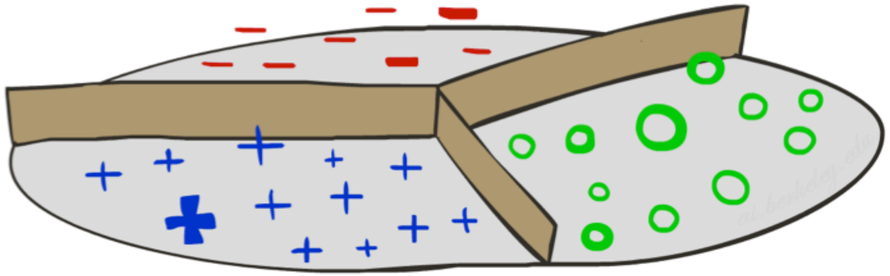
$$w_y$$

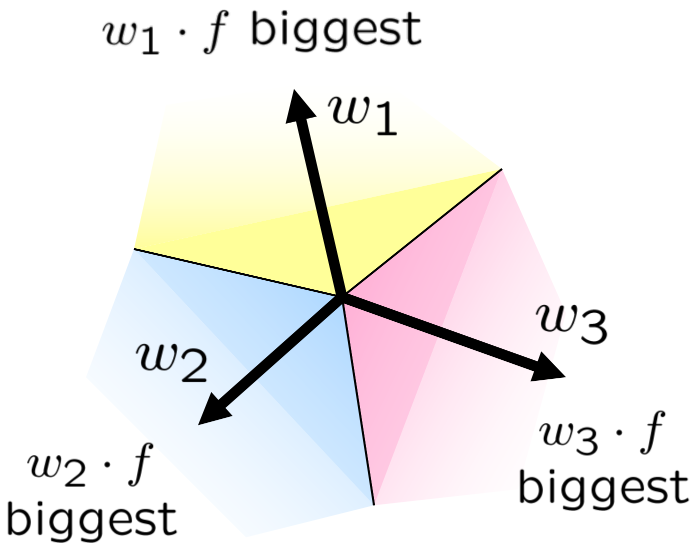
- Score (activation) of a class y:

$$w_y \cdot f(x)$$

- Prediction highest score wins

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





Learning: Multiclass Perceptron

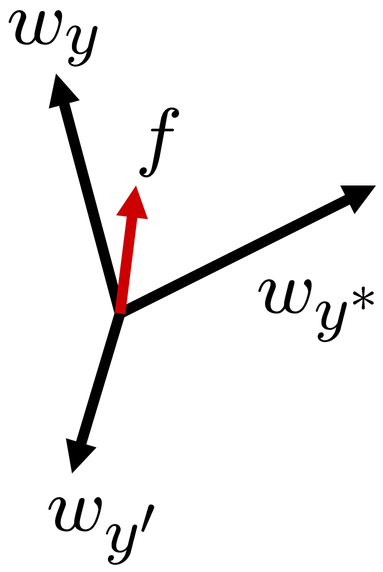
- Start with all weights = 0
- Consider training examples one by one
- Predict with current weights

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

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



Let's train this multiclass Perceptron by hand.

"win the vote"

"win the election"

"win the game"

w_{SPORTS}

```
BIAS   : 1
win    : 0
game   : 0
vote   : 0
the    : 0
...
```

$w_{POLITICS}$

```
BIAS   : 0
win    : 0
game   : 0
vote   : 0
the    : 0
...
```

w_{TECH}

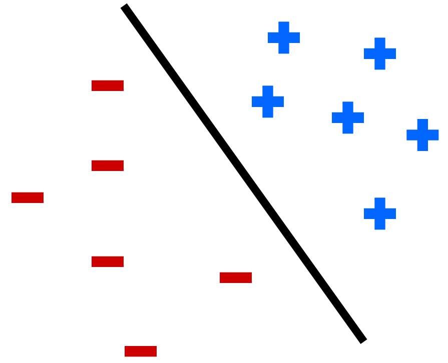
```
BIAS   : 0
win    : 0
game   : 0
vote   : 0
the    : 0
...
```

Properties of Perceptrons

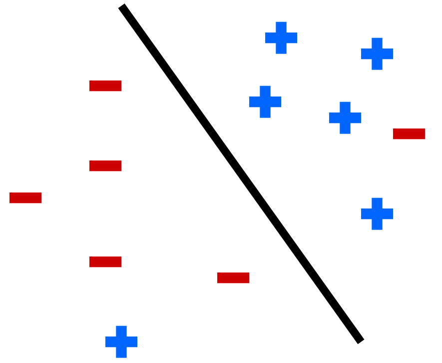
- Separability: true if some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable

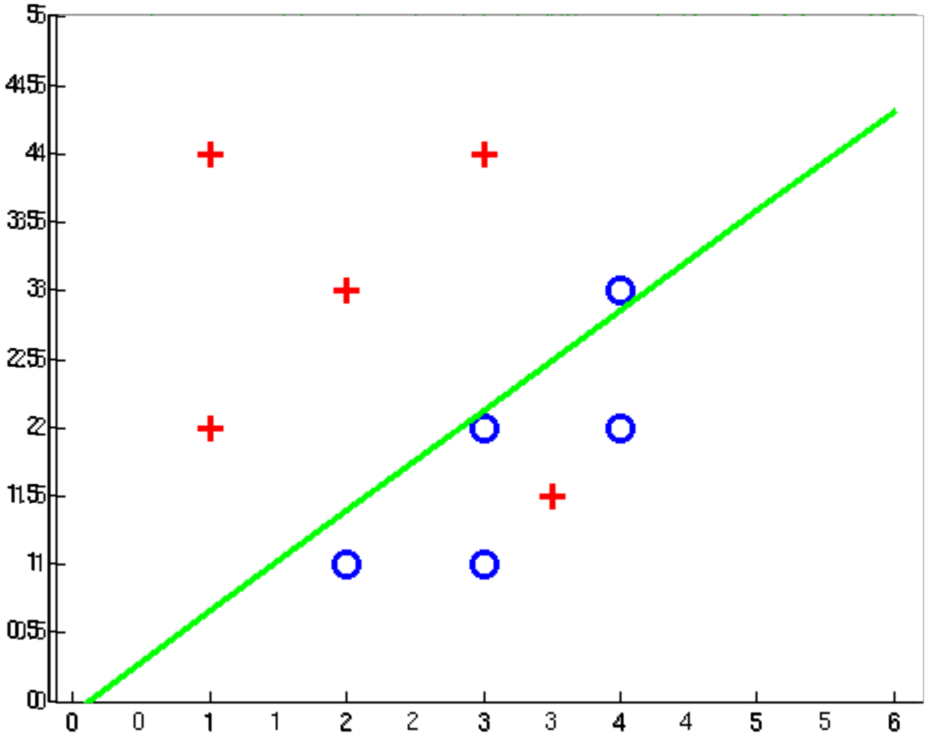


Non-Separable

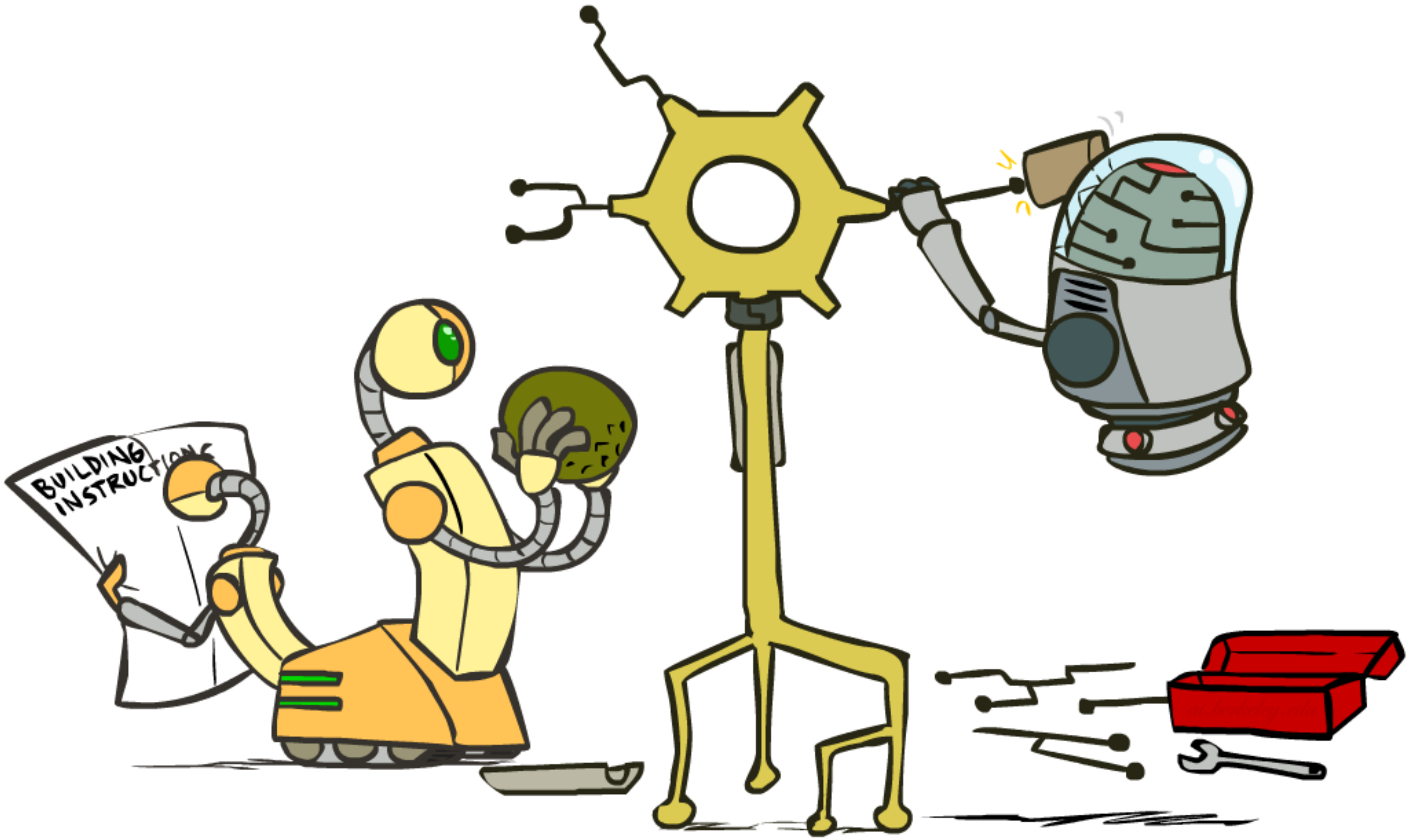


Examples: Perceptron

- Non-Separable Case



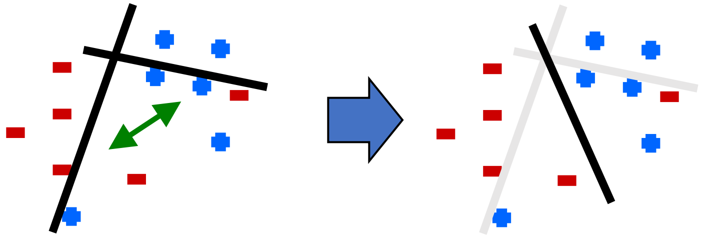
Improving the Perceptron

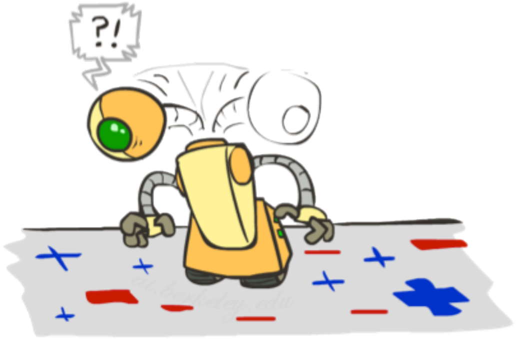


Problems with the Perceptron

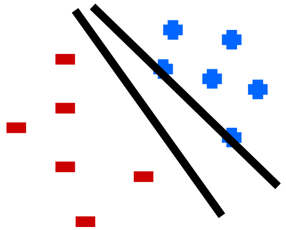
Noise: if the data isn't separable, weights might thrash

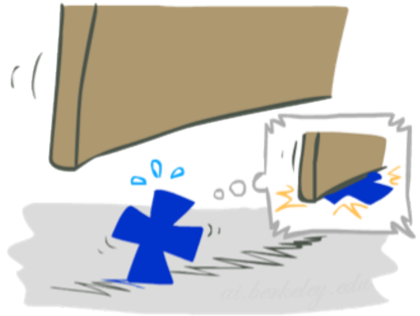
- Averaging weight vectors over time can help (averaged perceptron)





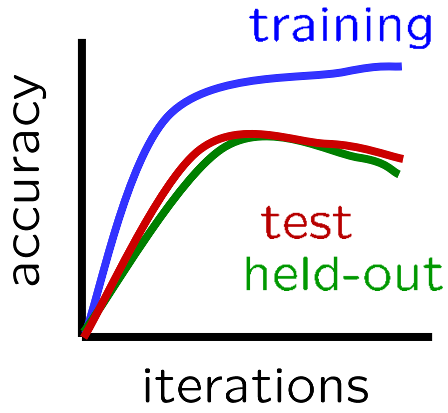
Mediocre generalization: finds a "barely" separating solution

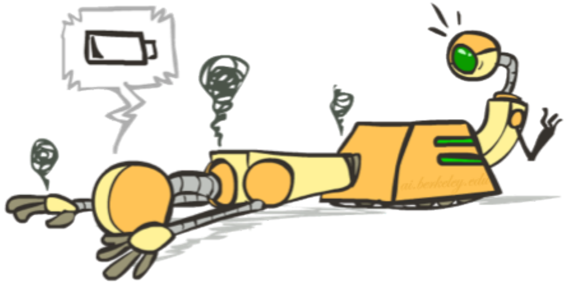




Overtraining: test / held-out accuracy usually rises, then falls

- Overtraining is a kind of overfitting





Fixing the Perceptron

Idea: adjust the weight update to mitigate these effects

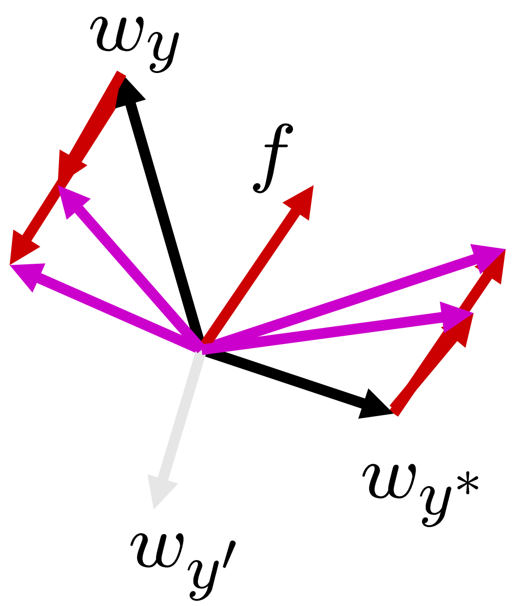
MIRA*: choose an update size that fixes the current mistake…

… but, minimizes the change to w

$$\min_w \frac{1}{2}\sum_y ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

The +1 helps to generalize



Guessed y instead of y^* on example x with features $f(x)$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$

* *Margin Infused Relaxed Algorithm*

Minimum correcting update

$$\min_{w} \frac{1}{2}\sum_{y} ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

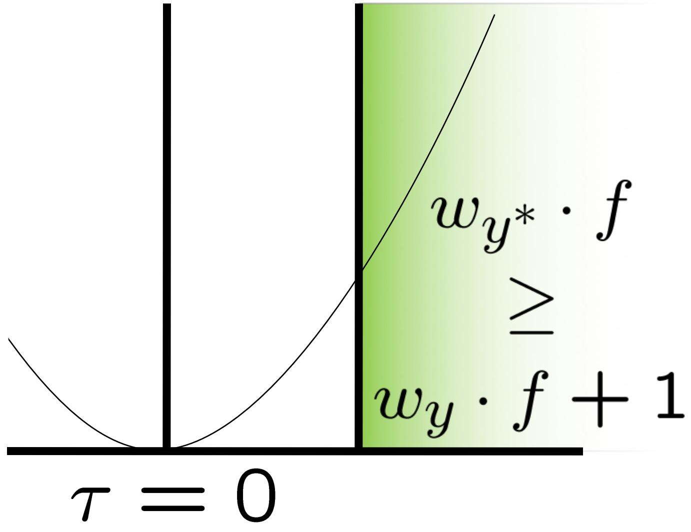
$$\min_{\tau} ||\tau f||^2$$

$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

$$(w'_{y^*} + \tau f) \cdot f = (w'_y - \tau f) \cdot f + 1$$

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2 f \cdot f}$$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$



min not τ=0, or would not have made an error, so min will be where equality holds

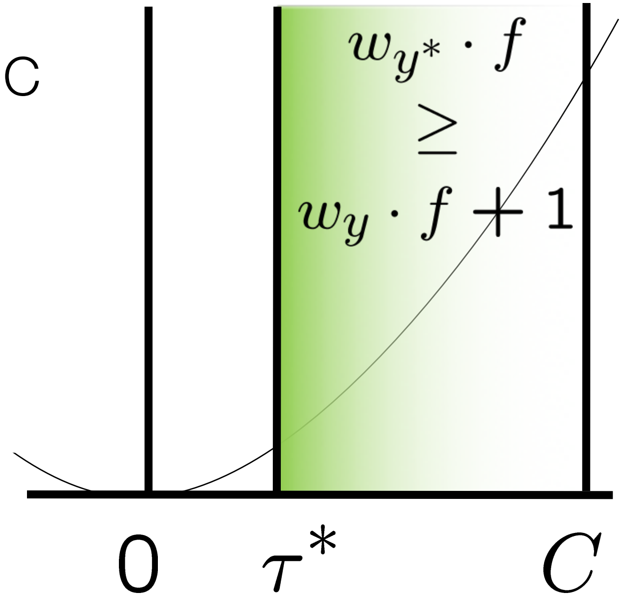
Maximum step size

In practice, it's also bad to make updates that are too large

- Example may be labeled incorrectly
- You may not have enough features
- Solution: cap the maximum possible value of τ with some constant C

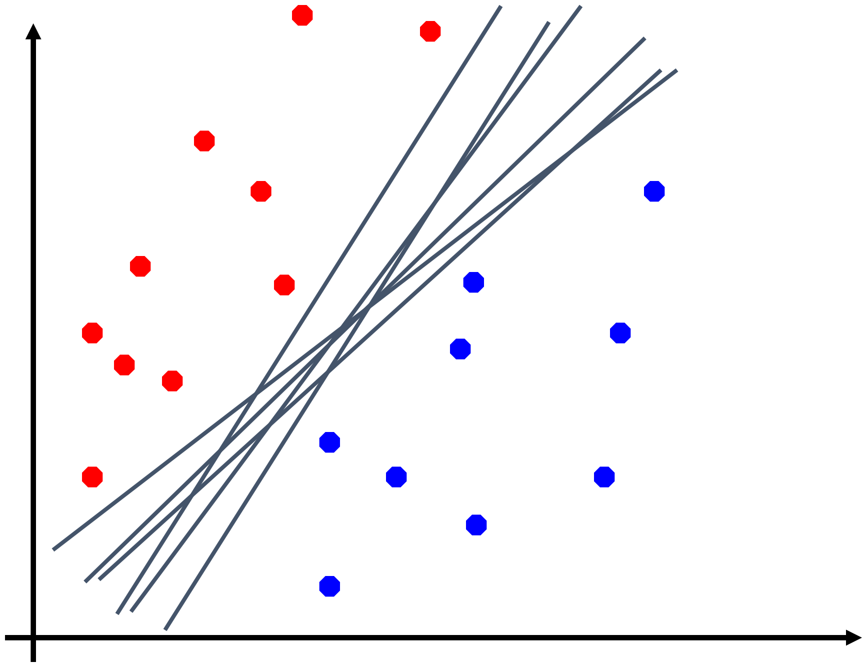
$$\tau^* = \min\left(\frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C\right)$$

- Corresponds to an optimization that assumes non-separable data
- Usually converges faster than perceptron
- Usually better, especially on noisy data



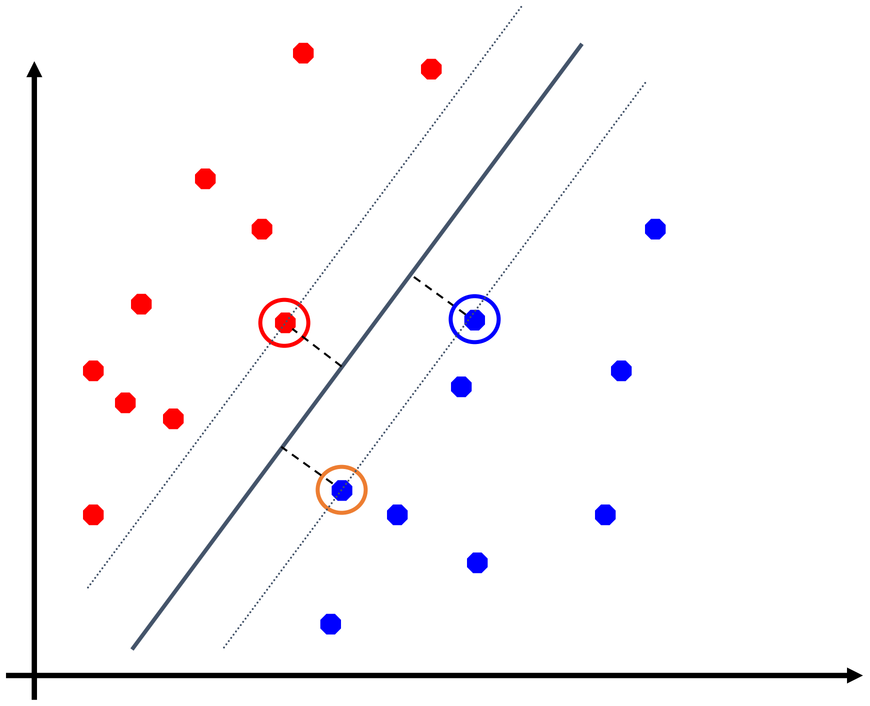
Linear separators

Which of these linear separators is optimal?



Support Vector Machines (SVMs)

- Maximizing the margin: good according to intuition, theory, practice
- Only support vectors matter; other training examples are ignorable
- Support vector machines (SVMs) find the separator with max margin
- Basically, SVMs are MIRA where you optimize over all examples at once



MIRA

$$\min_{w} \frac{1}{2}||w - w'||^2$$

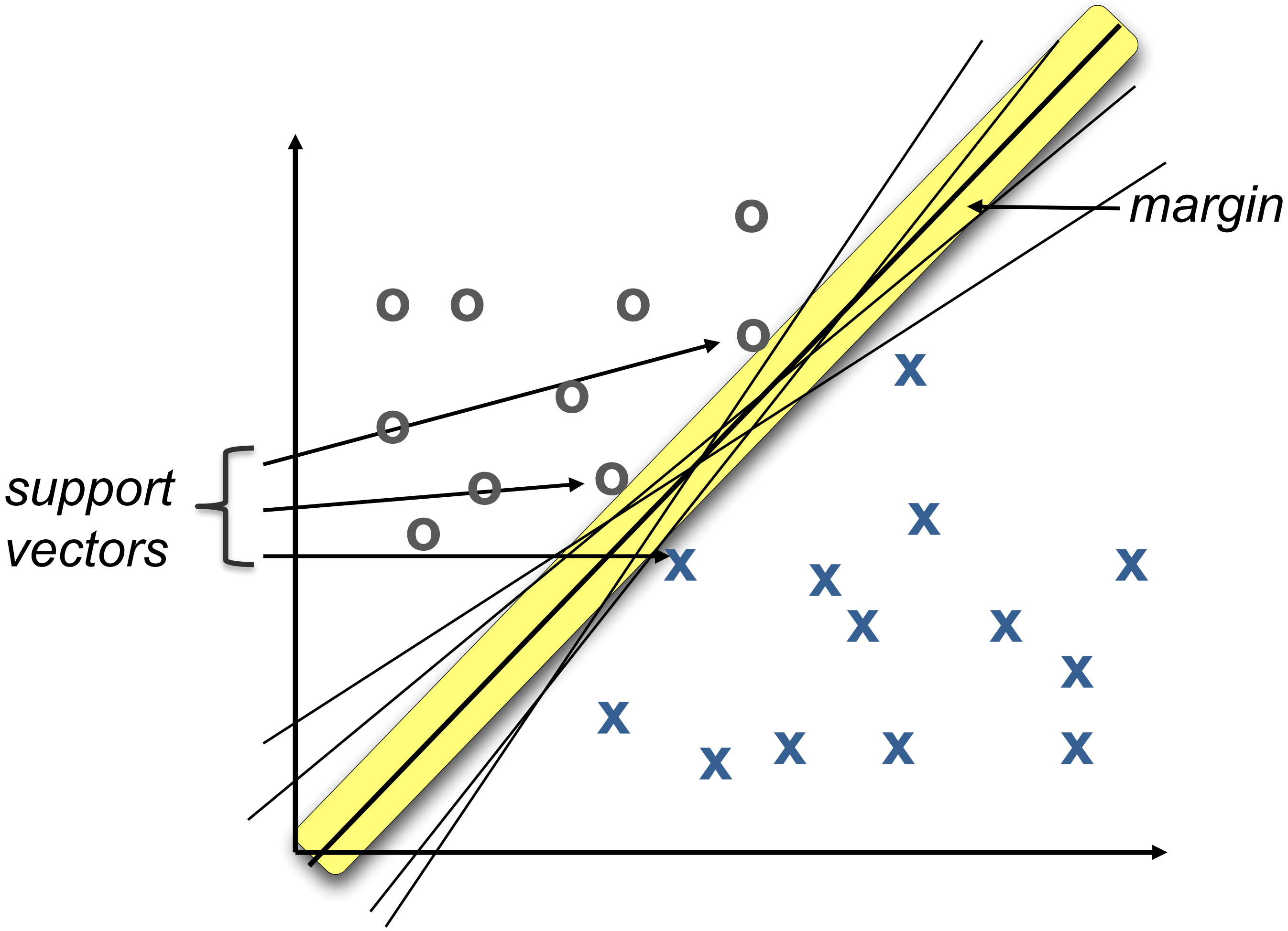
$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

SVM

$$\min_{w} \frac{1}{2}||w||^2$$

$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

Support Vector Machines (SVMs)



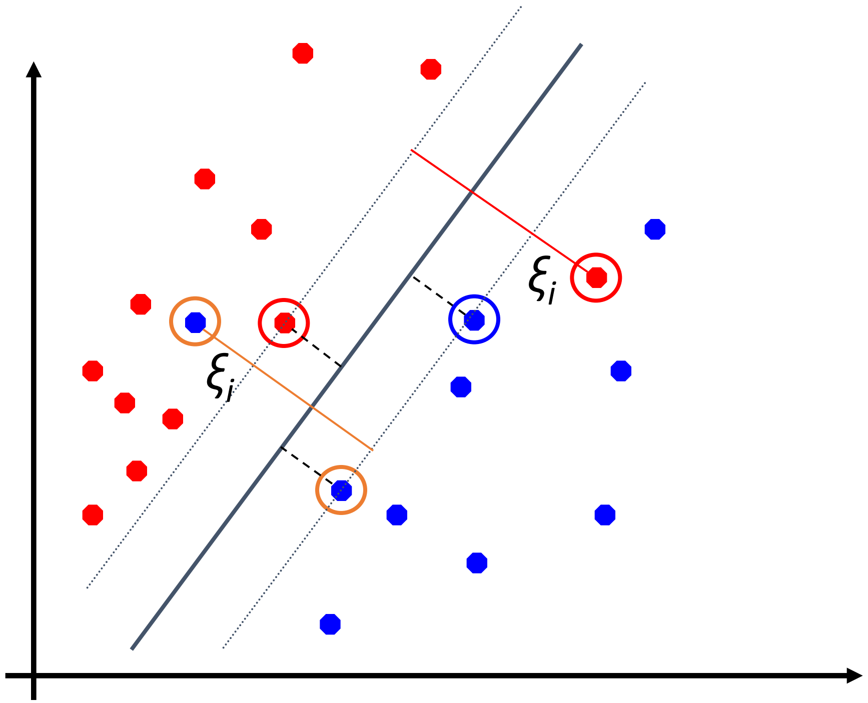
Solving the optimization problem

Find **w** and b such that
$\Phi(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Need to optimize a quadratic function subject to linear constraints.
- Fortunately: Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- Can also approximate via SGD!

“Soft margin” classification

- What if the training set is not linearly separable?
- Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called soft.



"Soft margin" classification

- The old formulation:

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0$

- Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.

Classification: comparison

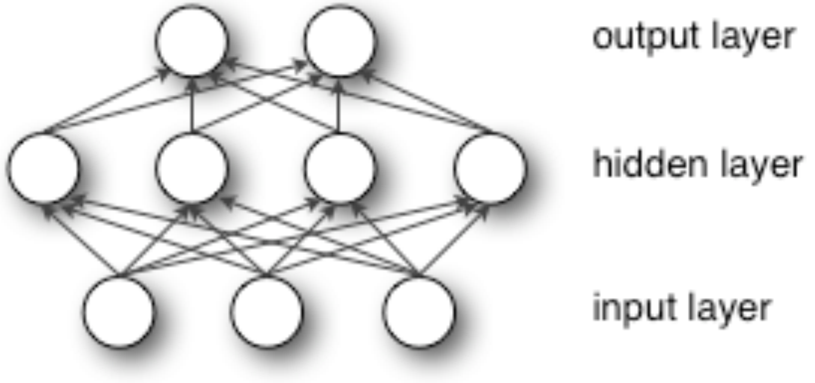
Naïve Bayes (generative model)

- Builds a model training data
- Gives prediction probabilities
- Strong assumptions about feature independence
- One pass through data (counting)

Perceptrons / MIRA / SVM (discriminative models)

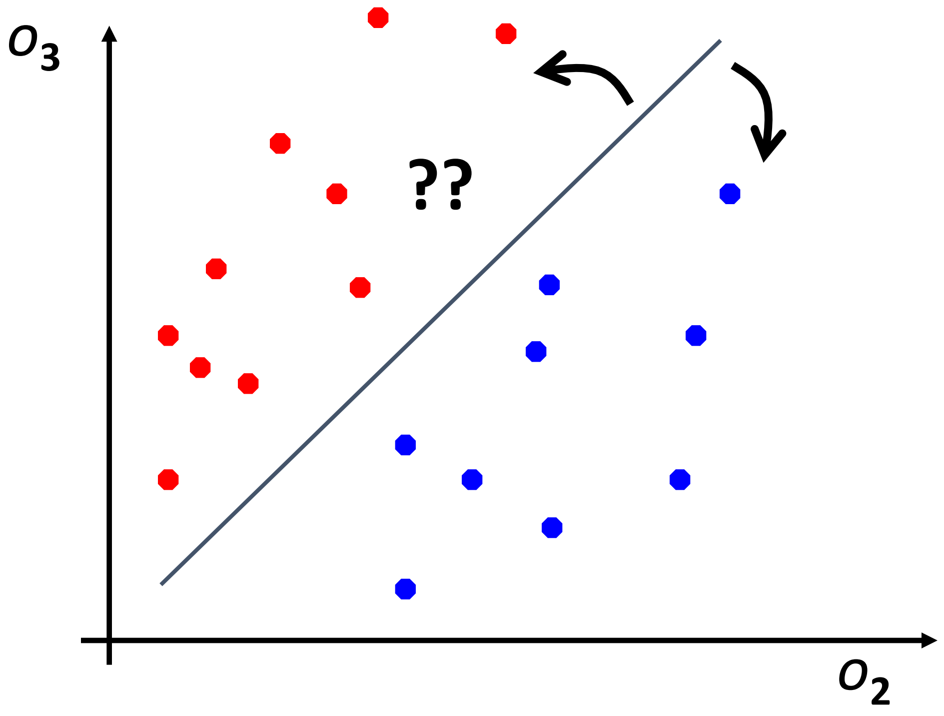
- Makes less assumptions about data
- Mistake-driven learning
- Multiple passes through data (prediction)
- Often more accurate

Multi-Layer perceptrons



Perceptron as a linear separator

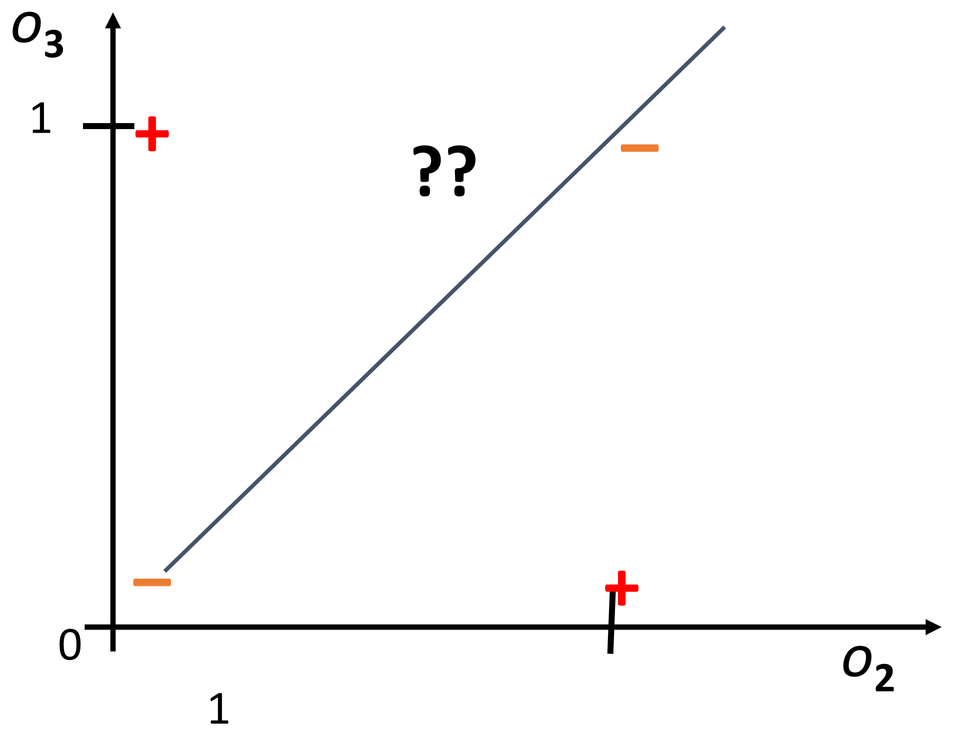
Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.



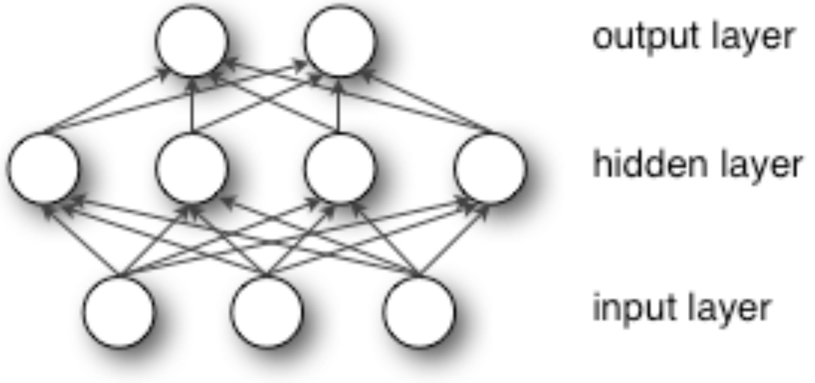
Or *hyperplane* in
n-dimensional space

Where Perceptron fails

Cannot learn *exclusive*-or!



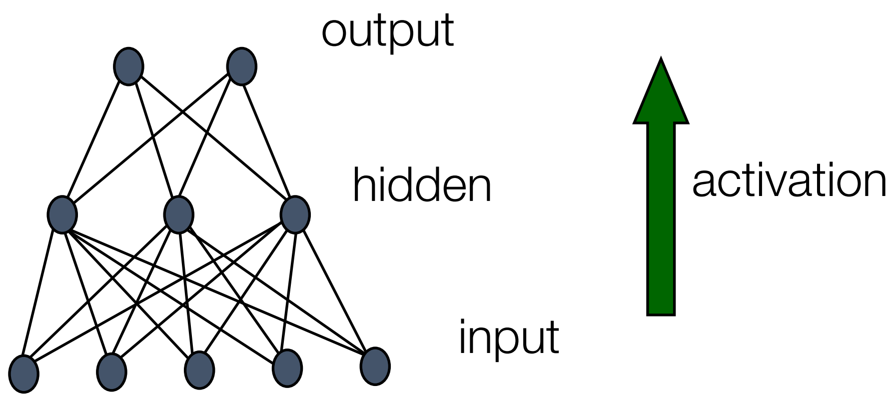
Multi-Layer Perceptrons to the rescue!



Multi-Layer networks ("*deep learning*")

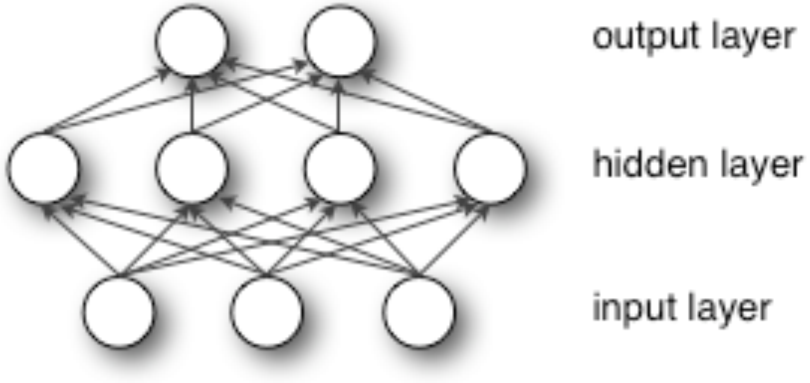
- Can represent arbitrary functions

- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.



- The weights determine the function computed. Given an arbitrary number of hidden units, any boolean function can be computed with a single hidden layer.

OK, great but… how do we fit this thing?



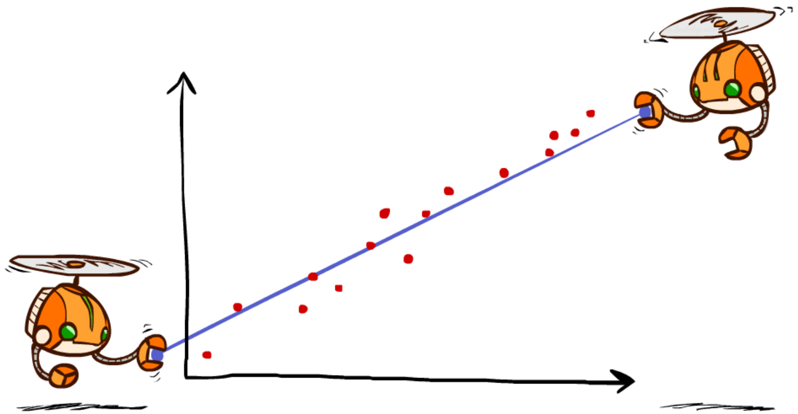
Flashback to Approx Q-learning: *Gradient Descent*

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\, \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

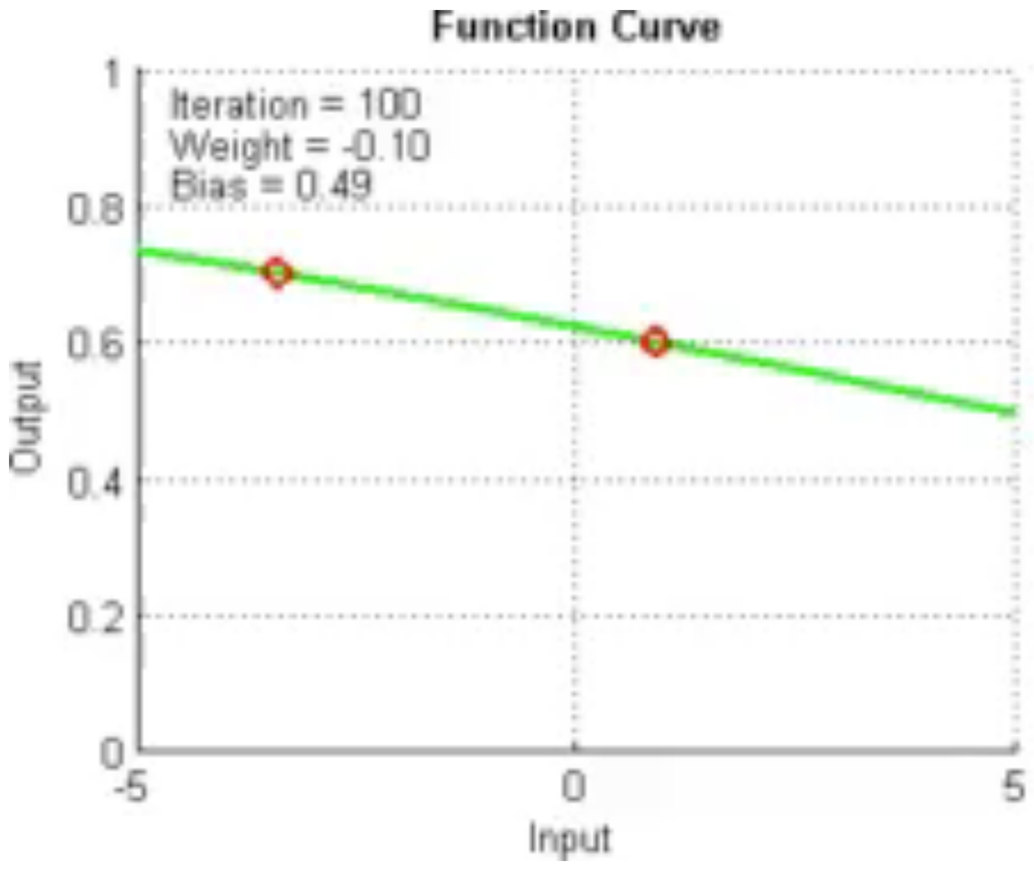
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

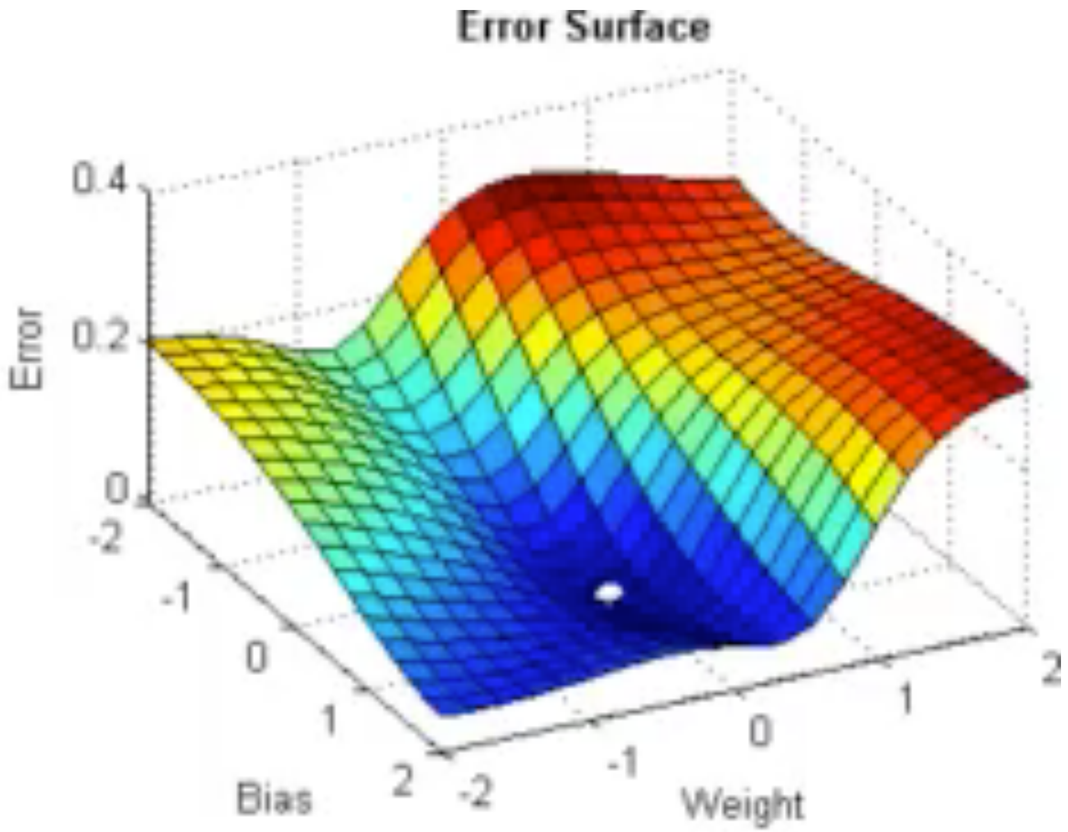


Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a) \right] f_m(s, a)$$

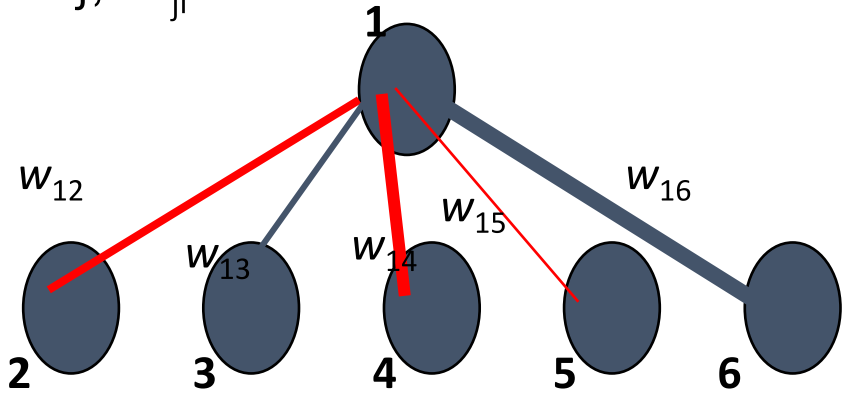
"target" "prediction"


Function Curve
Iteration = 100
Weight = -0.10
Bias = 0.49
Output
Input
1
0.8
0.6
0.4
0.2
0
-5
0
5


Error Surface
Error
0.4
0.2
0
Bias
-2
-1
0
1
2
Weight
-2
-1
0
1
2

Neural network model: more notation

- Model network as a graph with cells as nodes and synaptic connections as weighted edges from node i to node j, w_{ji}



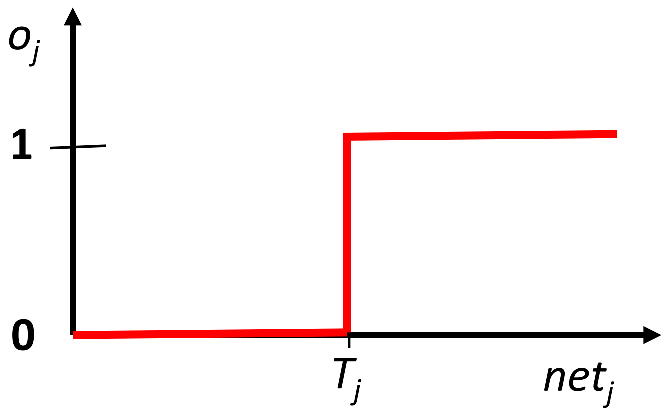
- Model net input to cell as

$$net_j = \sum_i w_{ji} o_i$$

- Cell output is:

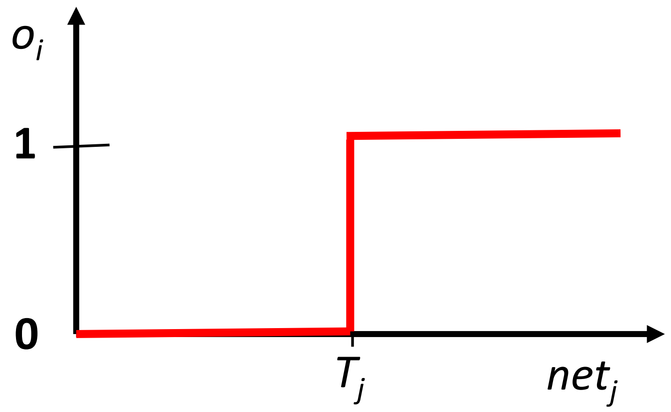
$$o_j = \begin{cases} 0 \text{ if } net_j < T_j \\ 1 \text{ if } net_i \geq T_j \end{cases}$$

(T_j is threshold for unit j)



Learning in multi-layer networks

- To do gradient descent, we need the output of a unit to be a differentiable function of its input and weights.
- Standard linear threshold function is not differentiable at the threshold.



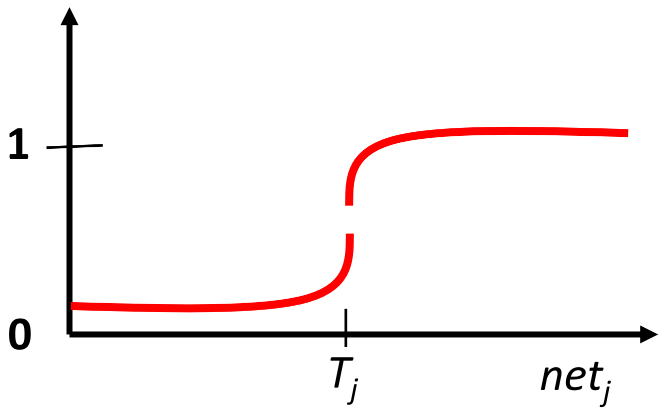
Differentiable output function

Need non-linear output function to move beyond linear functions.

- A multi-layer linear network is still linear

Standard solution is to use the non-linear, differentiable sigmoidal "logistic" function:

$$o_j = \frac{1}{1 + e^{-(net_j - T_j)}}$$



Gradient descent

Define objective to minimize error:

$$E(W) = \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2$$

where D is the set of training examples, K is the set of output units, t_{kd} and o_{kd} are, respectively, the label and current output for unit k for example d.

The derivative of a sigmoid unit with respect to net input is:

$$\frac{\partial o_j}{\partial net_j} = o_j(1 - o_j)$$

Learning rule to change weights to minimize error is:

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

Backpropagation learning rule

Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1 - o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

where η is a constant called the learning rate

t_j is the correct teacher output for unit j

δ_j is the error measure for unit j

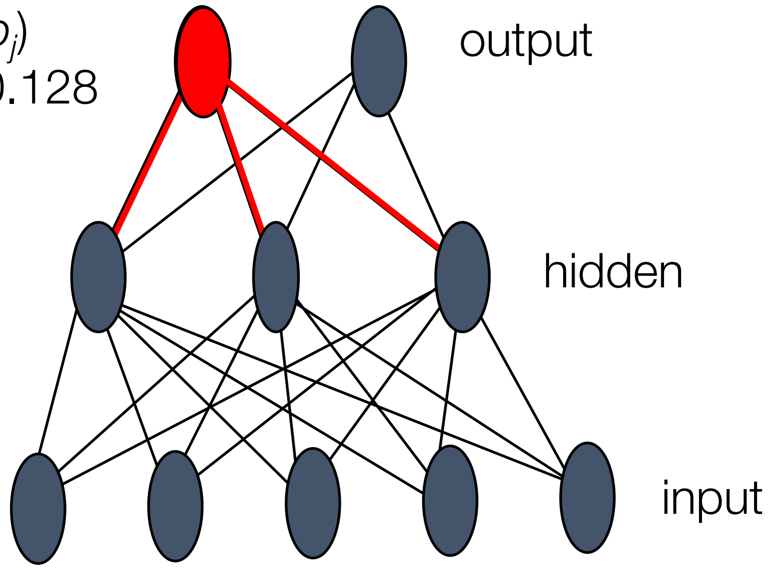
Backpropagation in action

First calculate error of output units and use this to change the top layer of weights.

Current output: o_j=0.2
Correct output: t_j=1.0
Error $\delta_j = o_j(1-o_j)(t_j-o_j)$
0.2(1–0.2)(1–0.2)=0.128

Update weights into j

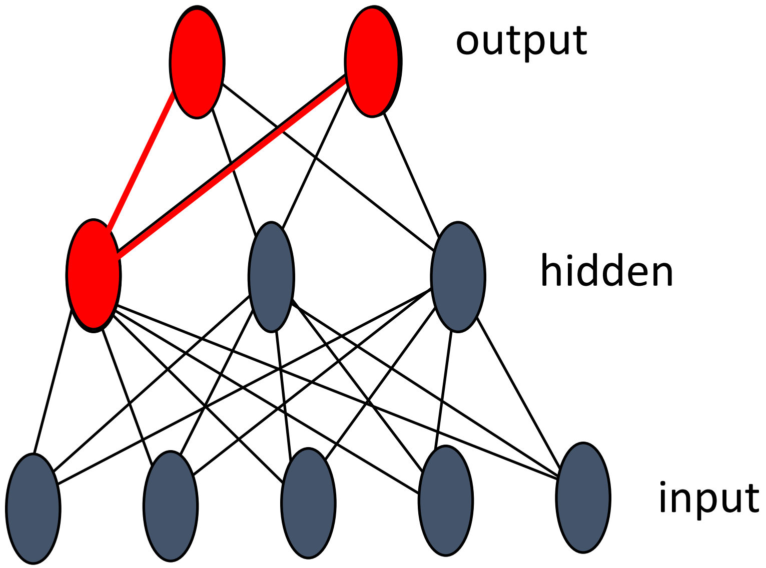
$$\Delta w_{ji} = \eta \delta_j o_i$$



Backpropagation in action

Next calculate error for hidden units based on errors on the output units it feeds into.

$$\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{kj}$$



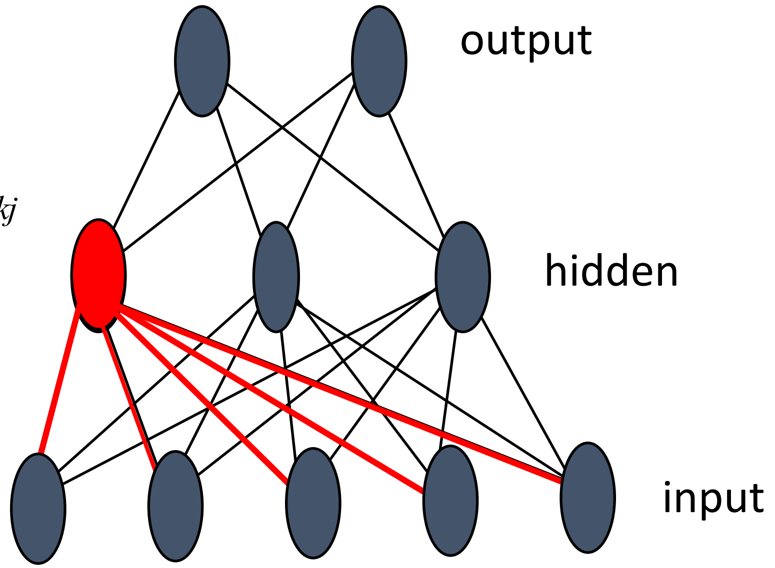
Backpropagation in action

Finally update bottom layer of weights based on errors calculated for hidden units.

$$\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{kj}$$

Update weights into *j*

$$\Delta w_{ji} = \eta \delta_j o_i$$



Hidden unit representations

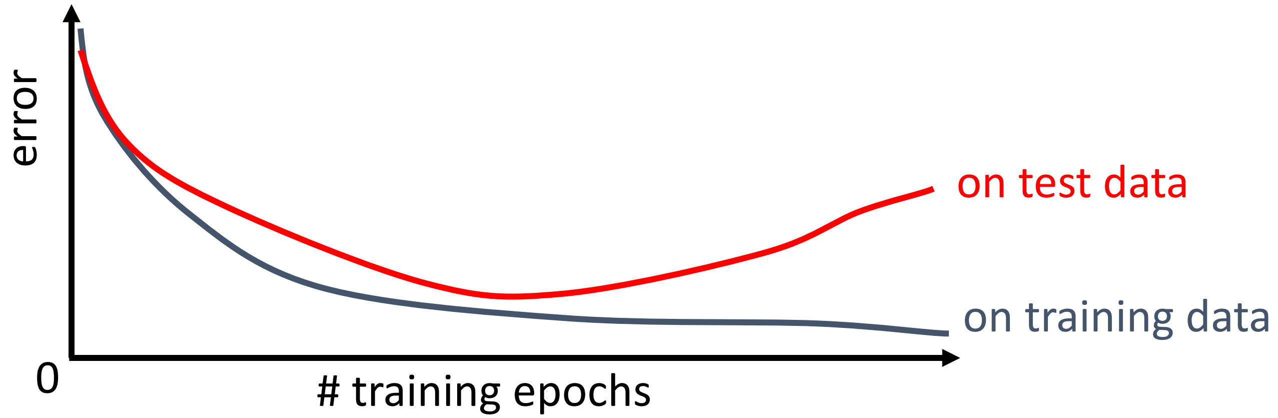
- Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space

- Hidden units can often be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc

- The hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature

Over-training prevention

Running too many epochs can result in over-fitting.



Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.

To avoid losing training data for validation:

- Use internal 10-fold CV on the training set to compute the average number of epochs that maximizes generalization accuracy.
- Train final network on complete training set for this many epochs.

That's it for today!

- Next week: more machine learning!
- Reminders:
  - **Homework 4** due **Friday!**
  - **Project proposals due next Tuesday (4/4)!!!**