CS 360: Machine Learning

Sara Mathieson, Sorelle Friedler

Spring 2024

HAVERFORD COLLEGE

Admin

- **Lab 4** and **Lab 5** graded
  - Any regrade requests (including midterm) must be brought within 1 week of receiving your grade
- **Lab 6** was due last night (see Piazza for runtime issues if you’re taking a late day)
- **Lab 7** posted, due Thurs April 4
  - Last lab with required partners
  - We can form partners during lab today
- **Project proposal** due April 8 (short)

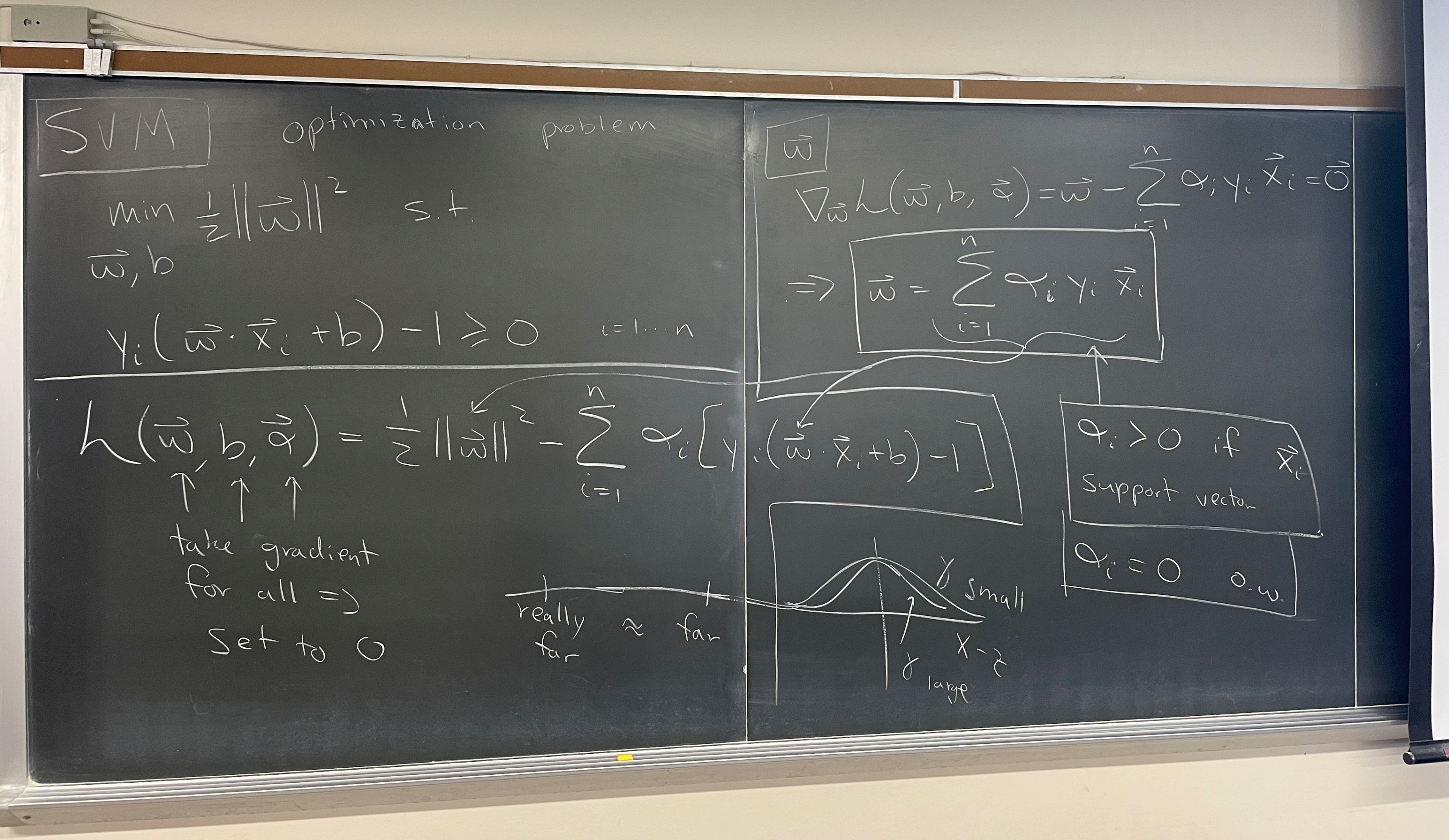
Outline for March 26

- SVM extensions
- Introduction to neural networks
- Fully connected (FC) neural networks
- Image data format and intro to Lab 7

Outline for March 26

- SVM extensions
- Introduction to neural networks
- Fully connected (FC) neural networks
- Image data format and intro to Lab 7

SVM dual optimization problem



$y_i = \{-1, +1\}$

$\boxed{b}$ $$\frac{\partial h(\vec{w}, b, \vec{\alpha})}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\Rightarrow \boxed{\sum_{i:y_i=-1} \alpha_i = \sum_{i:y_i=+1} \alpha_i}$$

$\boxed{\text{dual}}$

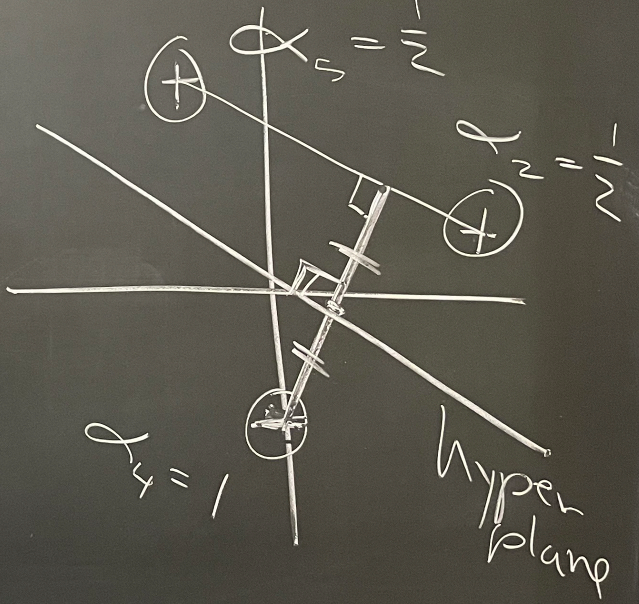
$$\max_{\vec{\alpha}} W(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \boxed{\vec{x}_i \cdot \vec{x}_j}$$

dot product

flexibility!

$$\text{s.t. } \boxed{\alpha_i \geq 0}$$

$$\& \sum_{i=1}^{n} \alpha_i y_i = 0$$



Kernel Idea

- By solving the dual form of the problem, we have seen how all computations can be done in terms of inner products between examples

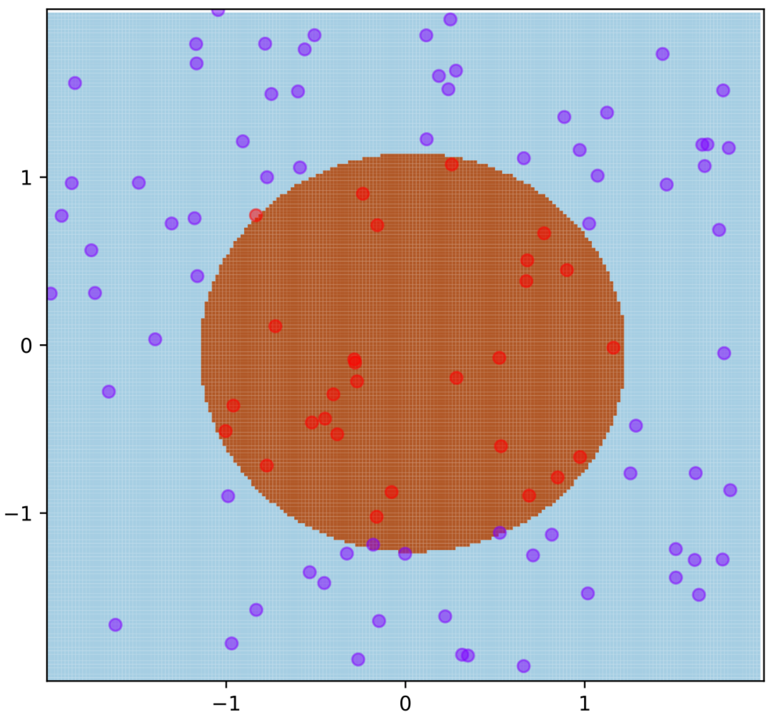
- One example of an inner product is the dot product, which is the linear version of SVMs

- But there are many others!

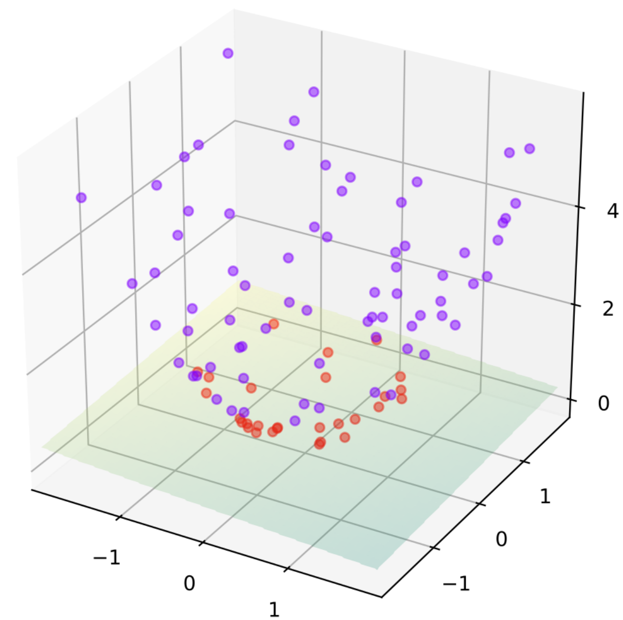
- Intuition: if points are close together, their kernel function will have a large value (measure of similarity)

Kernel Trick example

Feature mapping: $\varphi(\boldsymbol{x}) = (x_1,\ x_2,\ x_1^2 + x_2^2)$



Original feature space



Mapping after applying kernel (can now find a hyperplane)

Kernel function: $K(\boldsymbol{x},\ \boldsymbol{z}) = \boldsymbol{x} \bullet \boldsymbol{z} + ||\boldsymbol{x}||^2\, ||\boldsymbol{z}||^2$

Image: Shiyu Ji (wikipedia)

Gaussian Kernel

- Gaussian kernel is near 0 when points are far apart and near 1 when they are similar
- Also called Radial Basis Function (RBF) kernel

$$K(\vec{x}, \vec{z}) = \exp\left(-\frac{\|\vec{x} - \vec{z}\|^2}{2\sigma^2}\right)$$

Gaussian Kernel

- Gaussian kernel is near 0 when points are far apart and near 1 when they are similar
- Also called Radial Basis Function (RBF) kernel

$$K(\vec{x}, \vec{z}) = \exp\left(-\frac{\|\vec{x} - \vec{z}\|^2}{2\sigma^2}\right)$$

Often re-parametrized by gamma

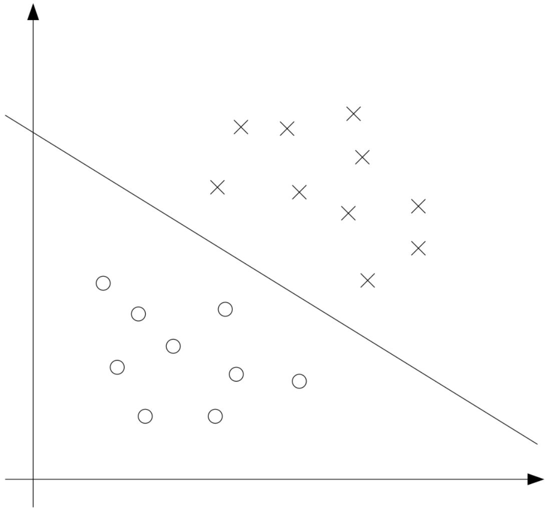
$$\boxed{\gamma = \frac{1}{2\sigma^2}}$$

$$K(\vec{x}, \vec{z}) = \exp\left(-\gamma\|\vec{x} - \vec{z}\|^2\right)$$

Soft-margin SVMs (non-separable case)

- Idea: we will use regularization to add a cost for each point being incorrectly classified by the hyperplane

- Hopefully many costs will be 0, but we can accommodate a few outliers



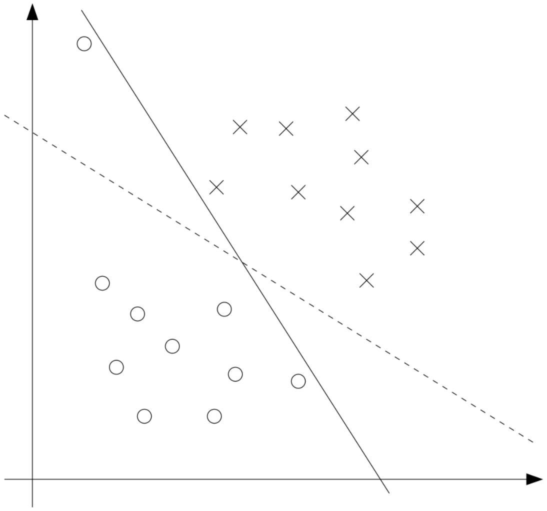


Figure: Andrew Ng

Soft-margin SVMs (non-separable case)

- New optimization problem with regularization

$$\min_{\xi,\vec{w},b} \quad \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

"flexible margin"

$$\text{s.t.} \quad y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \cdots, n$$

$$\text{and} \quad \xi_i \geq 0, \quad i = 1, \cdots, n$$

Meta-optimization process

- Incremental SVM optimization algorithm

Meta-optimization process

- Incremental SVM optimization algorithm

- Choose a subset S of examples and run optimization to get alpha values

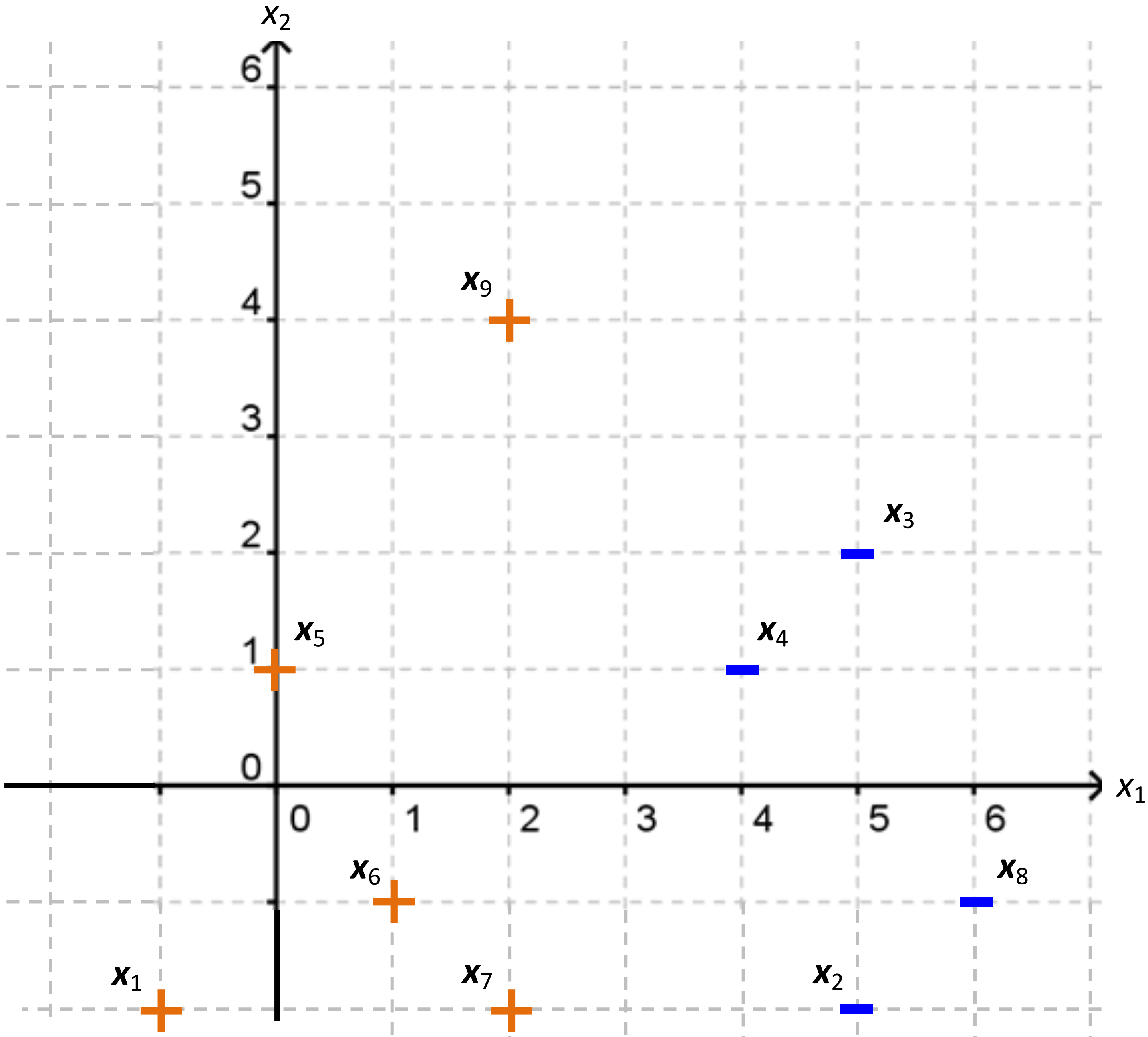
Meta-optimization process

- Incremental SVM optimization algorithm
- Choose a subset S of examples and run optimization to get alpha values
- Identify which alpha values are 0 => these cannot be support vectors in final solution!

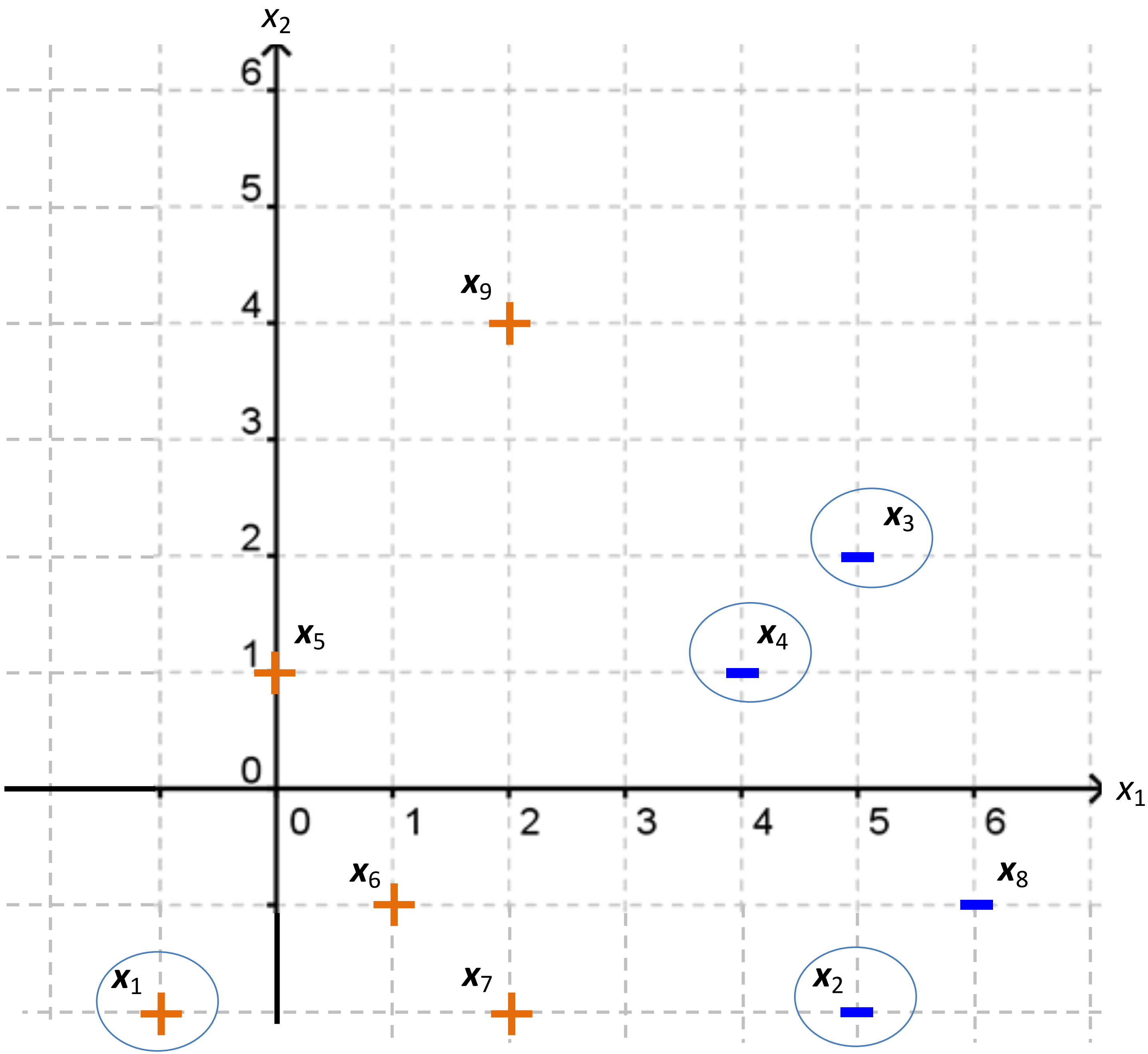
Meta-optimization process

- Incremental SVM optimization algorithm
- Choose a subset S of examples and run optimization to get alpha values
- Identify which alpha values are 0 => these cannot be support vectors in final solution!
- Discard these points and add new ones; repeat

Meta-optimization: example



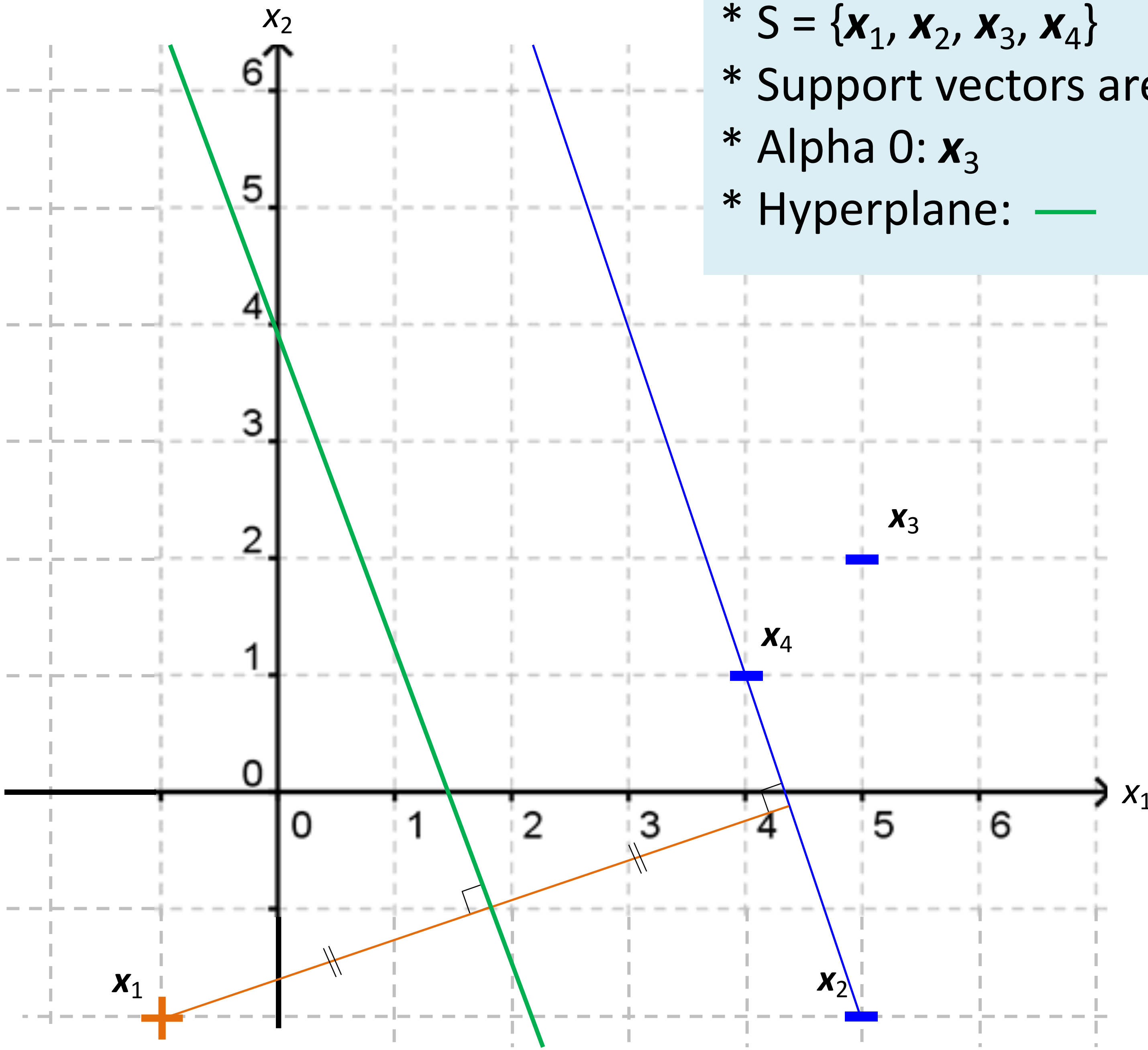
Meta-optimization: example



$K = 4$

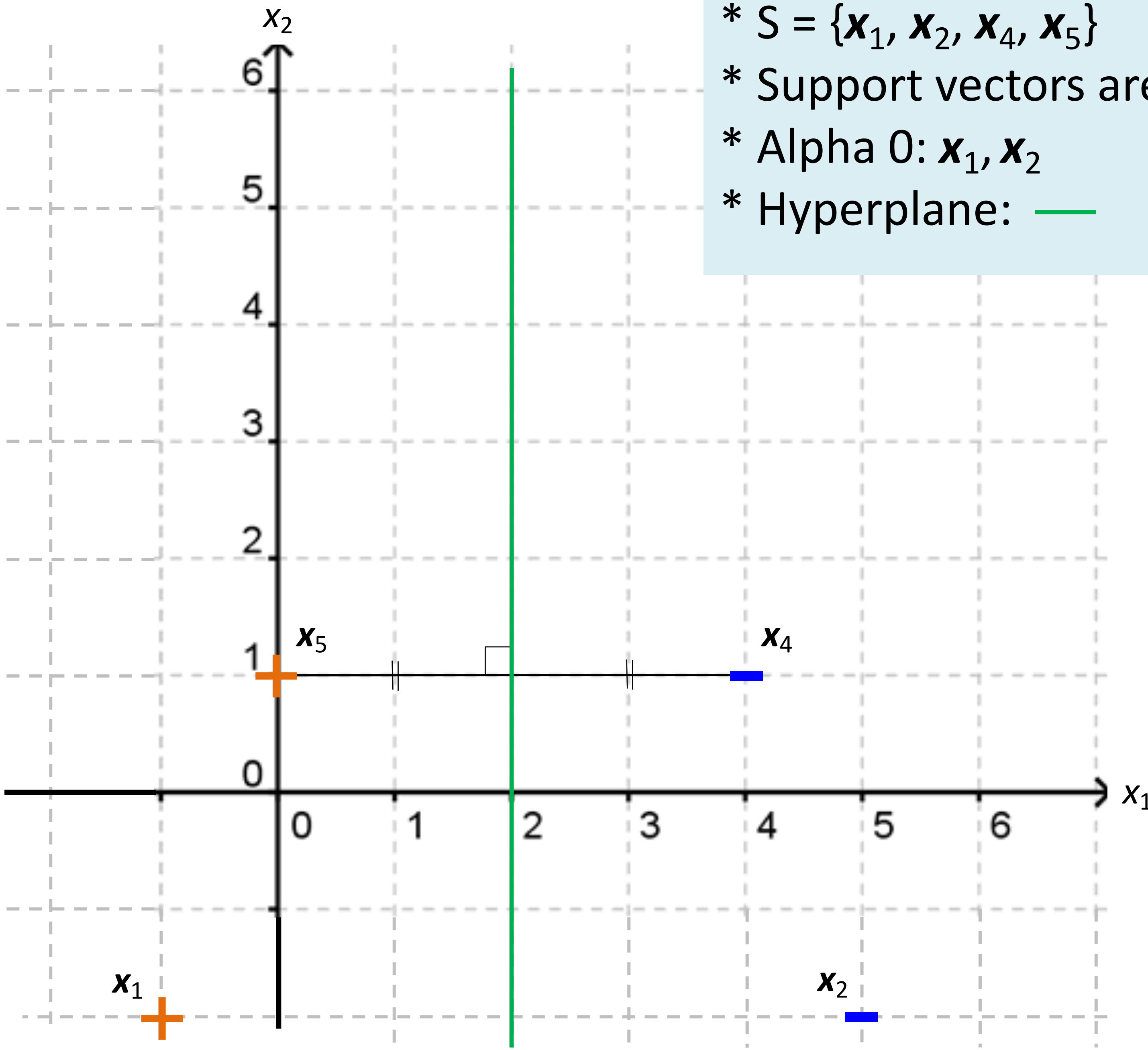
Round 1:

* S = {$\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_3$, $\boldsymbol{x}_4$}
* Support vectors are: $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_4$
* Alpha 0: $\boldsymbol{x}_3$
* Hyperplane: —



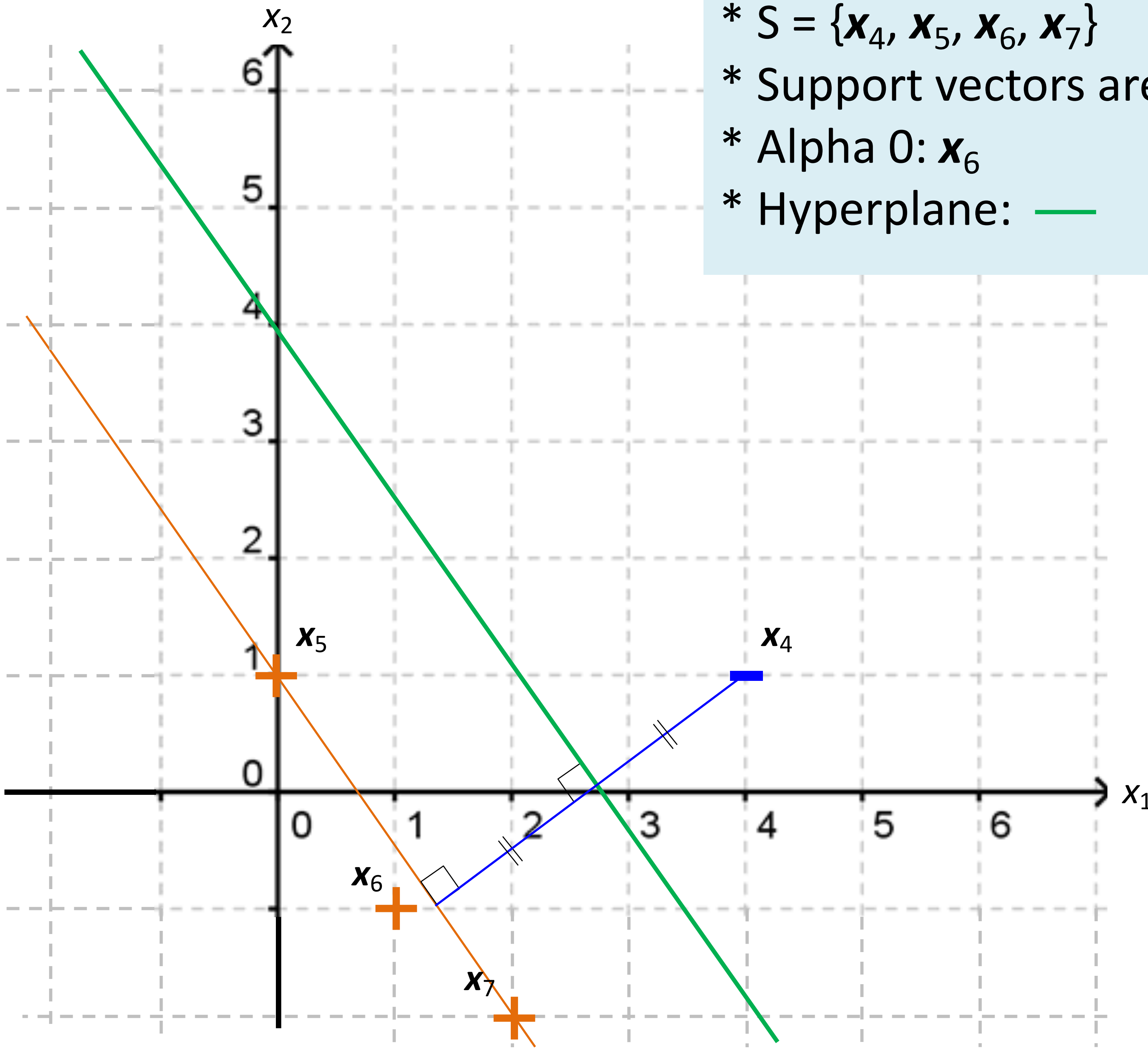
Round 1:

* S = {$\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{x}_4$, $\boldsymbol{x}_5$}
* Support vectors are: $\boldsymbol{x}_4$, $\boldsymbol{x}_5$
* Alpha 0: $\boldsymbol{x}_1$, $\boldsymbol{x}_2$
* Hyperplane:



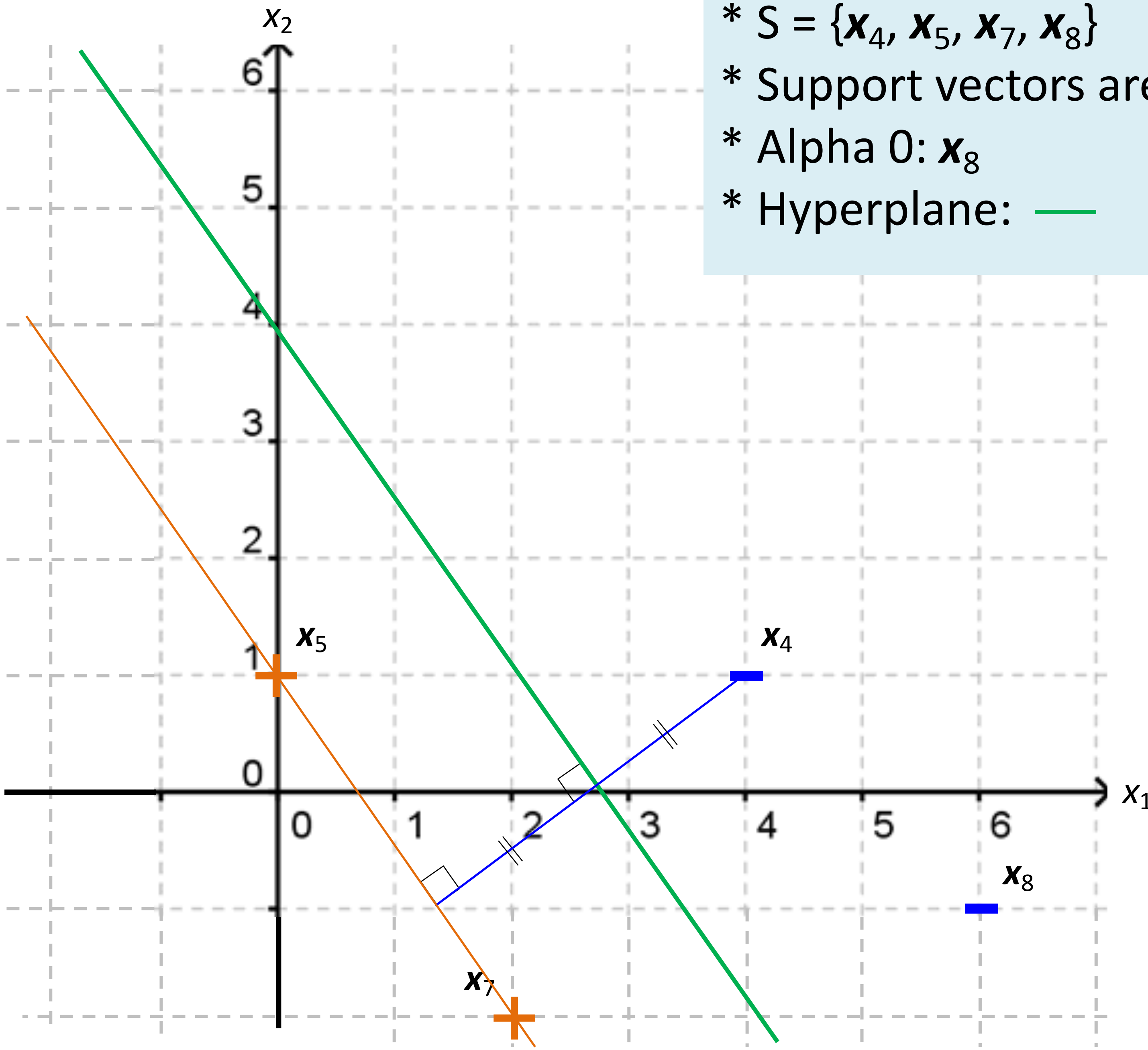
Round 3:

* S = {$\boldsymbol{x}_4$, $\boldsymbol{x}_5$, $\boldsymbol{x}_6$, $\boldsymbol{x}_7$}
* Support vectors are: $\boldsymbol{x}_4$, $\boldsymbol{x}_5$, $\boldsymbol{x}_7$
* Alpha 0: $\boldsymbol{x}_6$
* Hyperplane: —



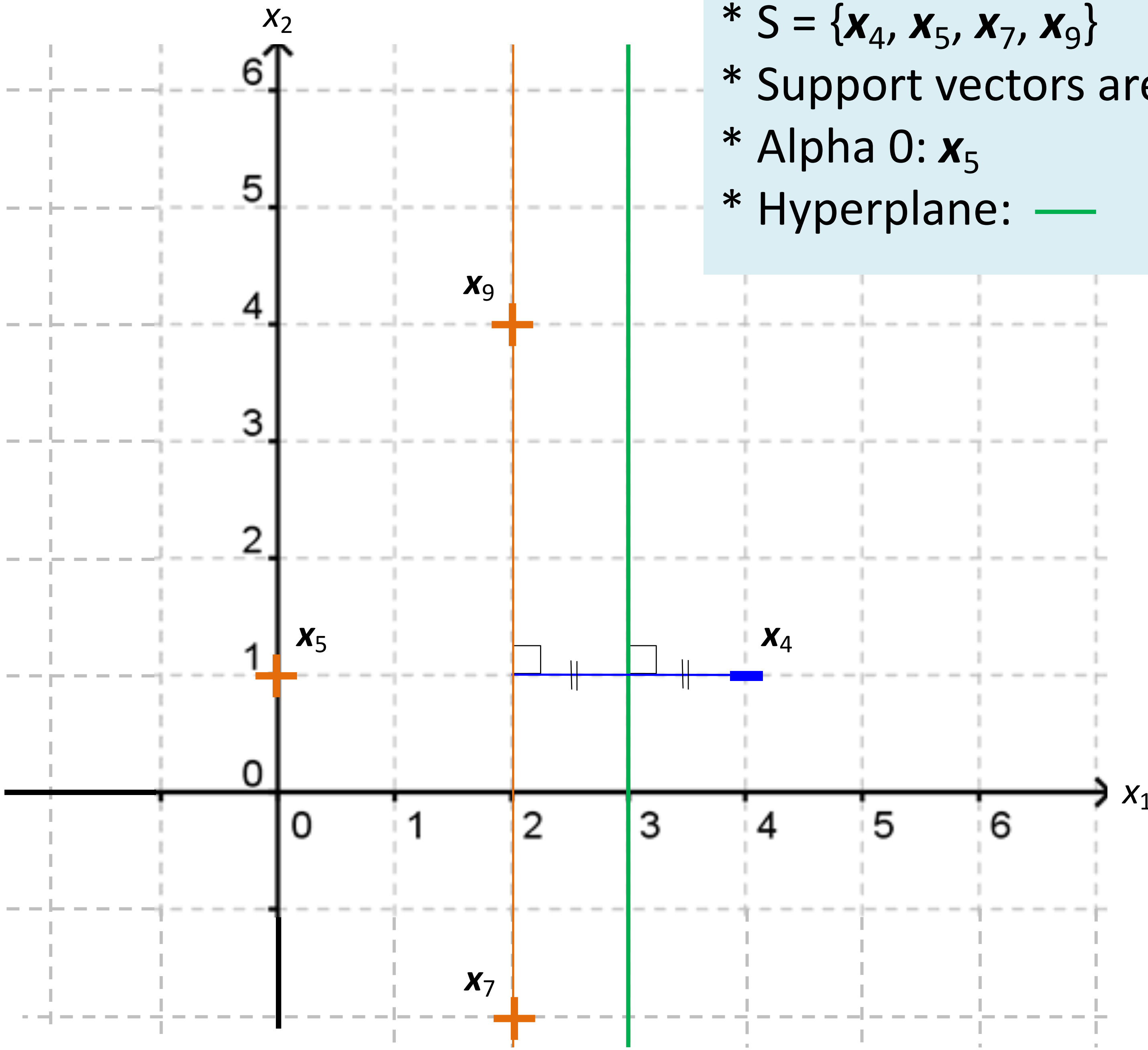
Round 4:

* S = {$\boldsymbol{x}_4$, $\boldsymbol{x}_5$, $\boldsymbol{x}_7$, $\boldsymbol{x}_8$}
* Support vectors are: $\boldsymbol{x}_4$, $\boldsymbol{x}_5$, $\boldsymbol{x}_7$
* Alpha 0: $\boldsymbol{x}_8$
* Hyperplane: —

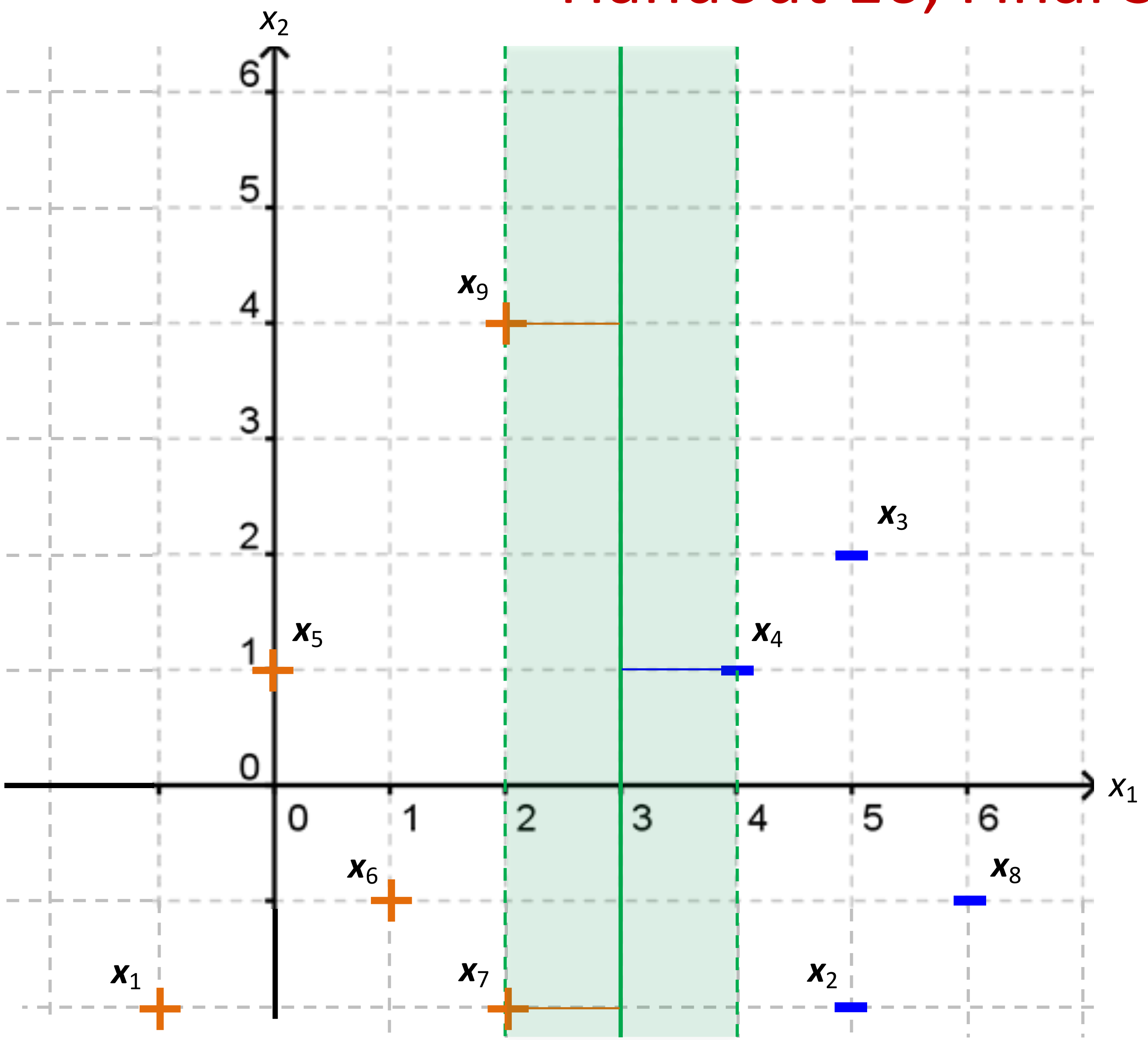


Round 5:

* S = {$\boldsymbol{x}_4$, $\boldsymbol{x}_5$, $\boldsymbol{x}_7$, $\boldsymbol{x}_9$}
* Support vectors are: $\boldsymbol{x}_4$, $\boldsymbol{x}_7$, $\boldsymbol{x}_9$
* Alpha 0: $\boldsymbol{x}_5$
* Hyperplane:



Handout 16, Final Solution



Discuss with a partner

1. If $\vec{x}_i$ is a support vector, what can we say about it? Circle all that apply:

   (a) its Lagrange multiplier $\alpha_i > 0$

   (b) its Lagrange multiplier $\alpha_i = 0$

   (c) $y_i(\vec{w} \cdot \vec{x}_i + b) = 0$

   (d) $y_i(\vec{w} \cdot \vec{x}_i + b) = 1$

   (e) $\vec{x}_i$ lies on the margin

Discuss with a partner

1. If $\vec{x}_i$ is a support vector, what can we say about it? Circle all that apply:

   (a) its Lagrange multiplier $\alpha_i > 0$

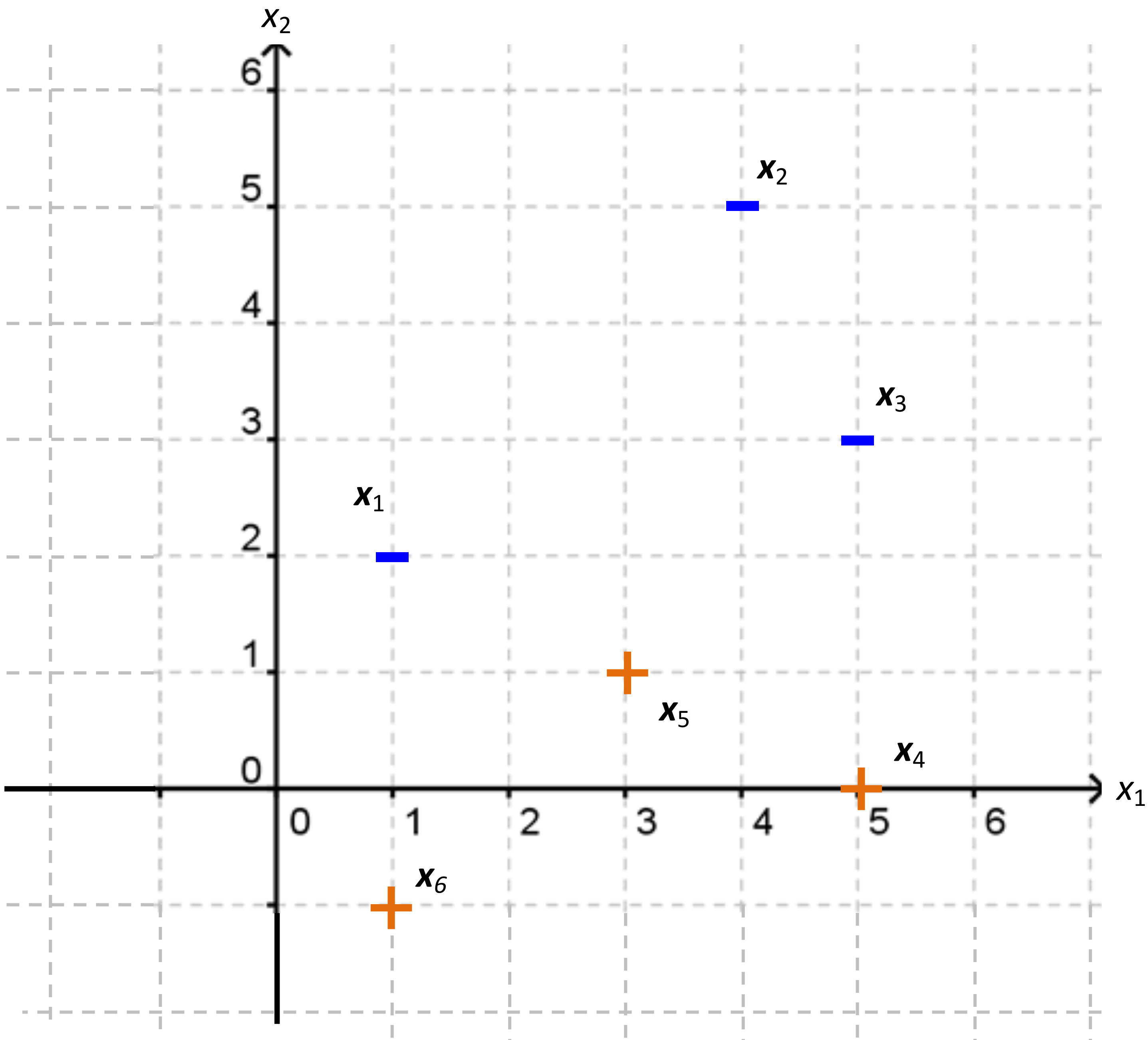
   (b) its Lagrange multiplier $\alpha_i = 0$

   (c) $y_i(\vec{w} \cdot \vec{x}_i + b) = 0$

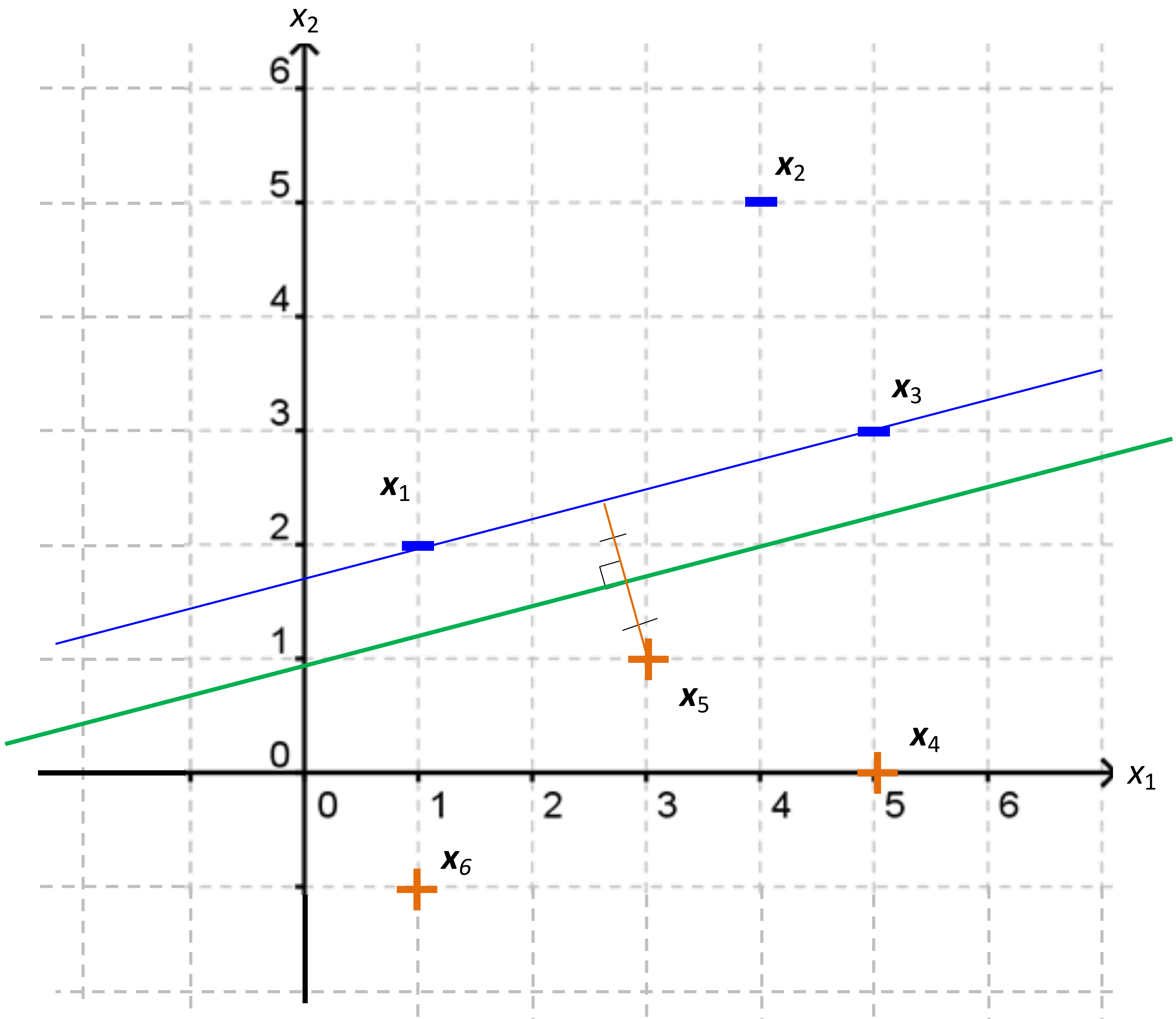
   (d) $y_i(\vec{w} \cdot \vec{x}_i + b) = 1$

   (e) $\vec{x}_i$ lies on the margin

Discuss with a partner: what are the support vectors?



Discuss with a partner: what are the support vectors?



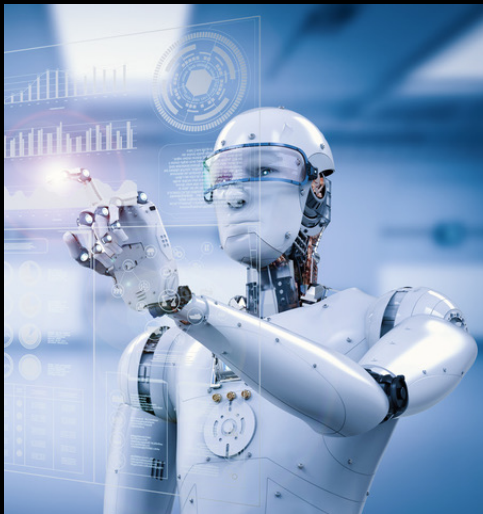
Disadvantages of SVMs

- Difficult to choose a kernel function
- Does not naturally take into account the correlations between features
- Hard to understand and interpret what the model has learned

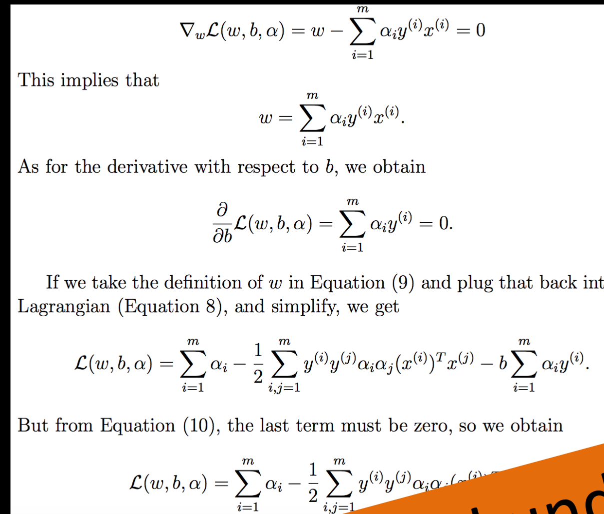
Outline for March 26

- SVM extensions
- Introduction to neural networks
- Fully connected (FC) neural networks
- Image data format and intro to Lab 7

MACHINE LEARNING



What society thinks I do



$$\nabla_w \mathcal{L}(w,b,\alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$$

This implies that

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}.$$

As for the derivative with respect to b, we obtain

$$\frac{\partial}{\partial b}\mathcal{L}(w,b,\alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0.$$

If we take the definition of w in Equation (9) and plug that back into Lagrangian (Equation 8), and simplify, we get

$$\mathcal{L}(w,b,\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b\sum_{i=1}^{m} \alpha_i y^{(i)}.$$

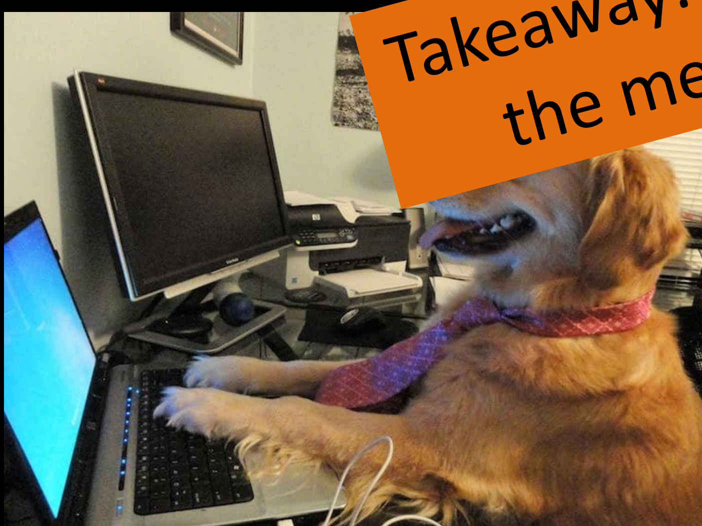
But from Equation (10), the last term must be zero, so we obtain

$$\mathcal{L}(w,b,\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \ldots$$

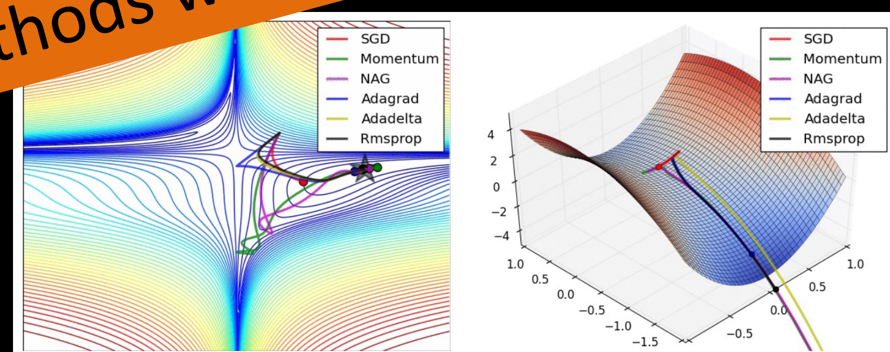
Wh...

...t other computer scientists think I do

Takeaway: we should understand the methods we are using!



What mathematicians think I do

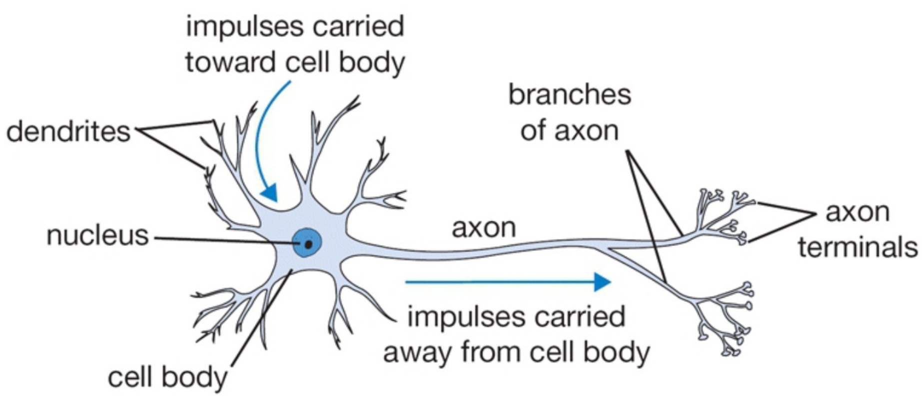


What I think I do

```
>>> from sklearn import svm
>>> import tensorflow as tf
```

What I really do

Biological Inspiration



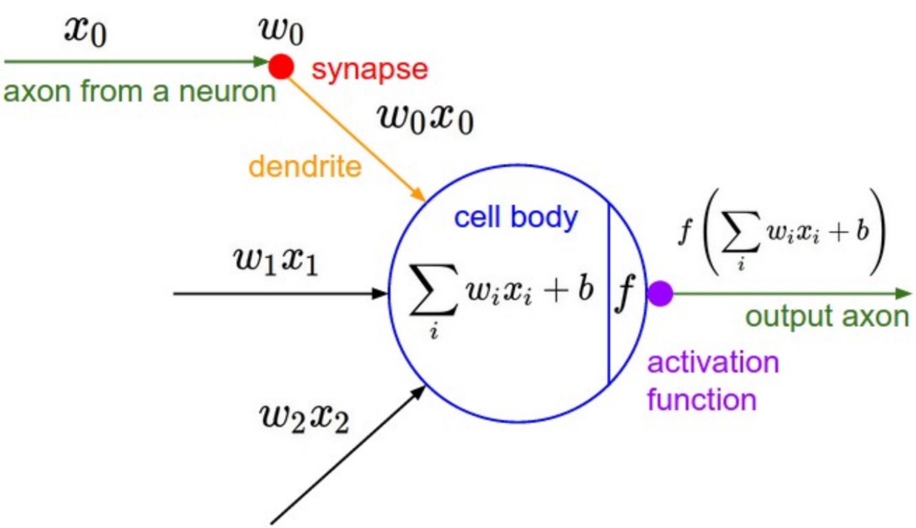


Figure: Stanford CS231n http://cs231n.github.io/neural-networks-1/

Goal: learn from complicated inputs

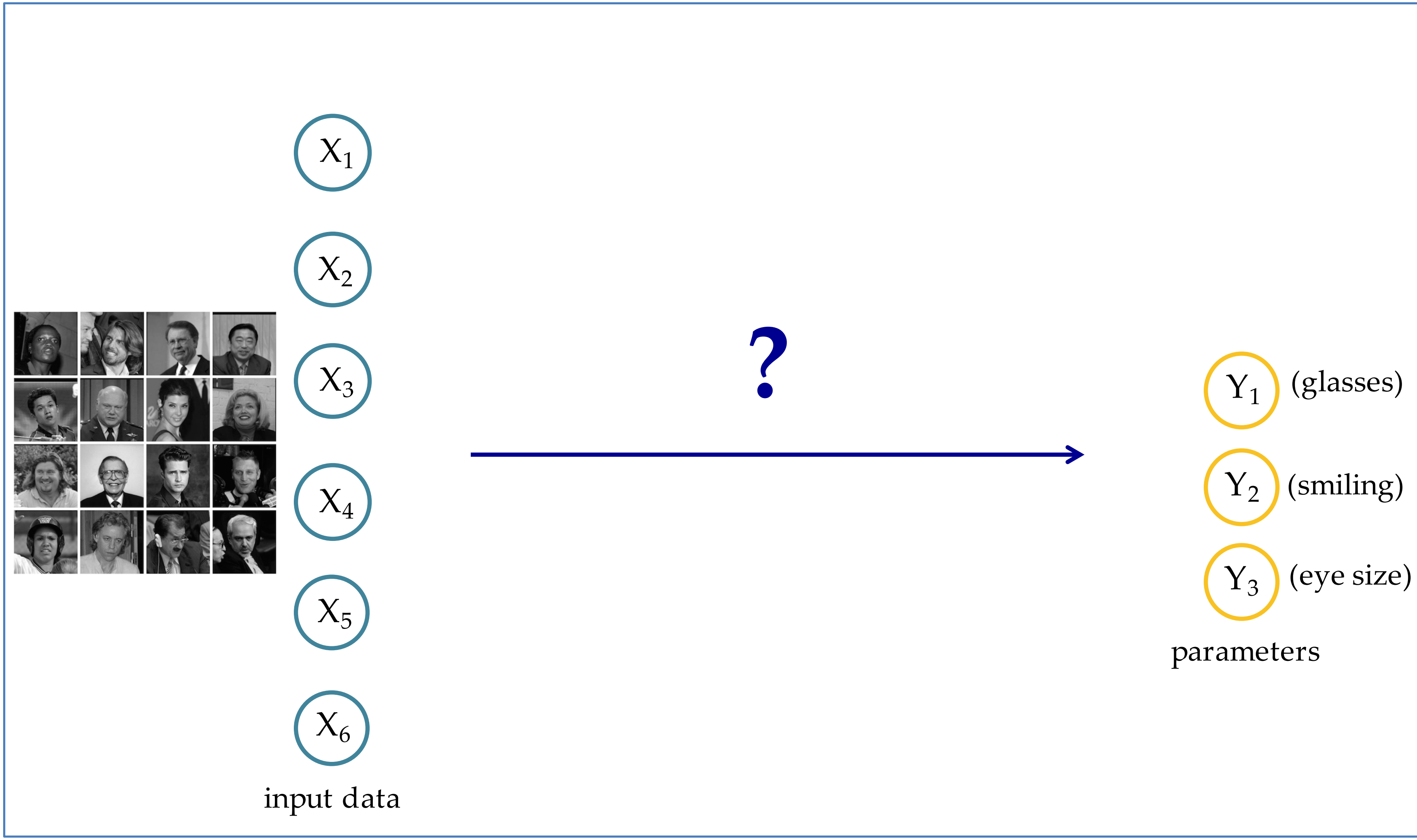


Image: Labeled Faces in the Wild (UMass)

Idea: transform data into lower dimension

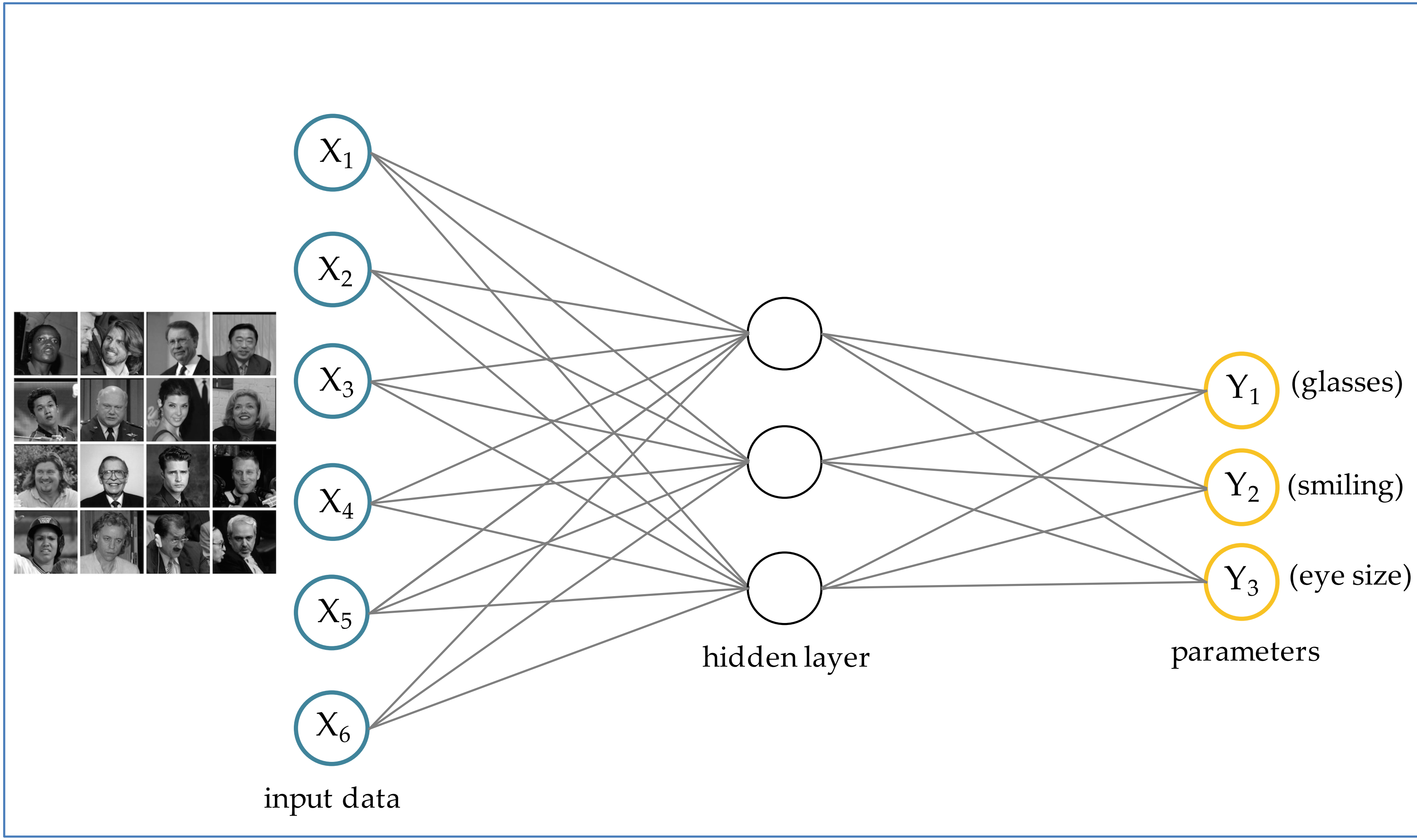


Image: Labeled Faces in the Wild (UMass)

Multi-layer networks = “deep learning”

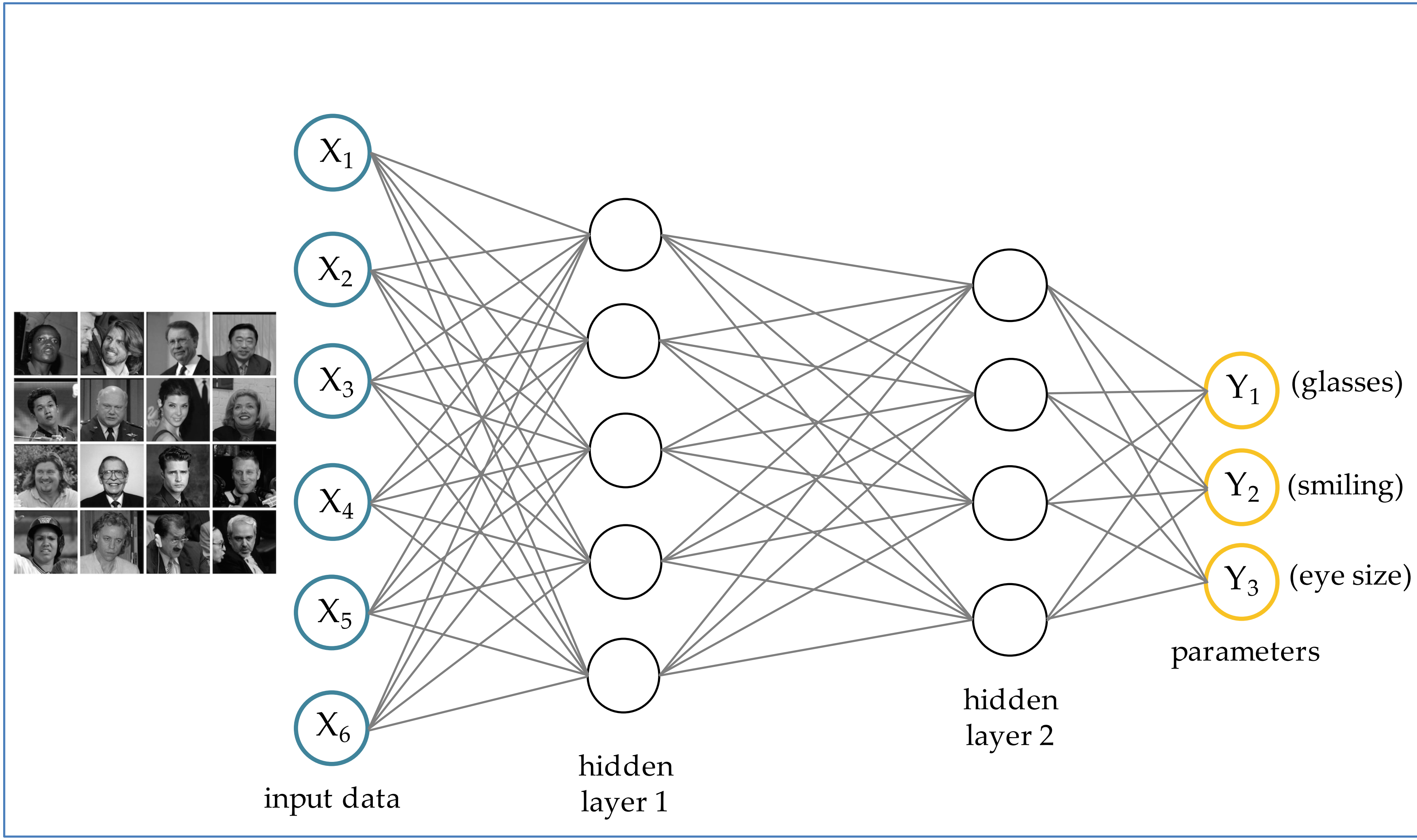
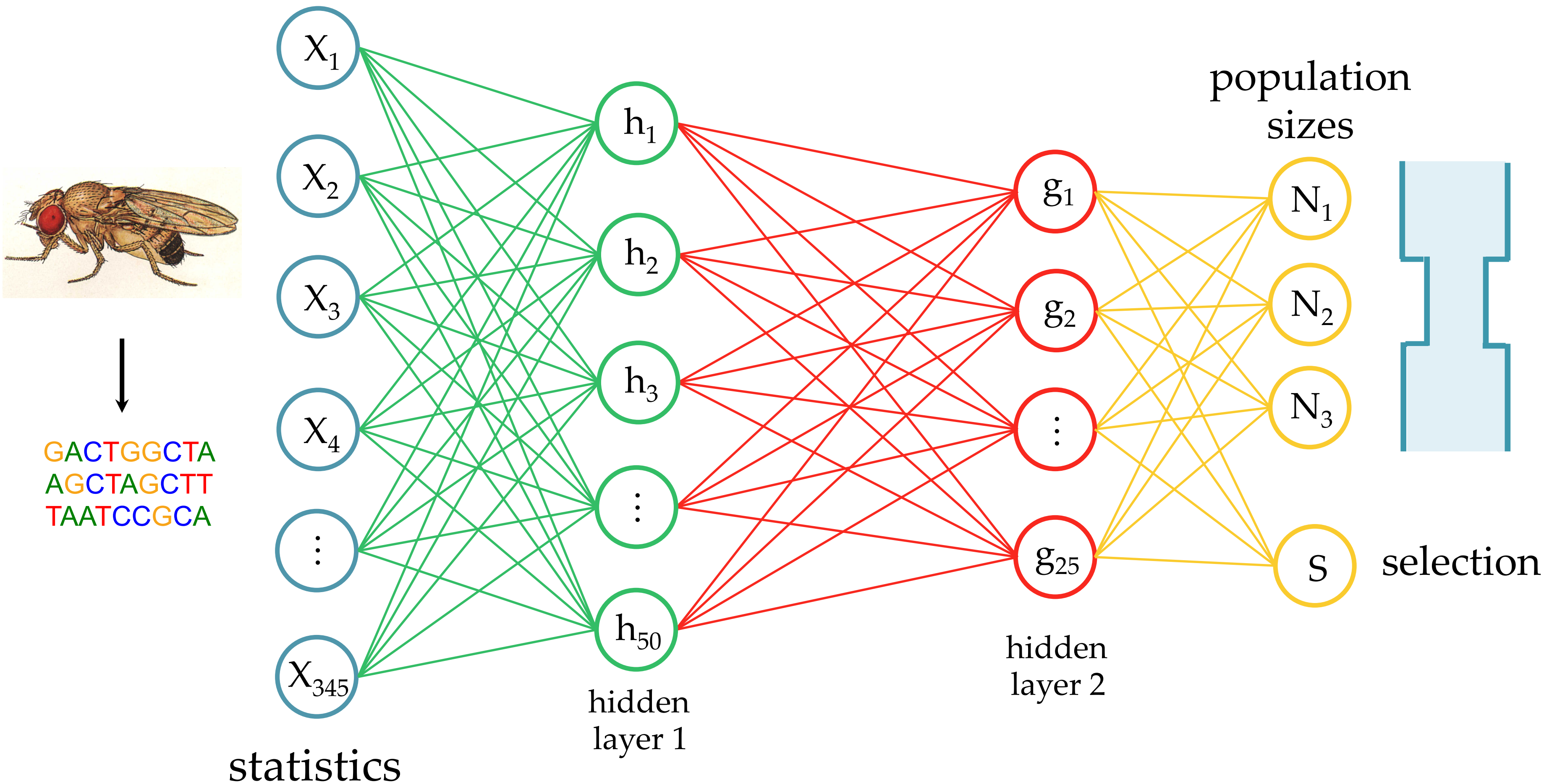


Image: Labeled Faces in the Wild (UMass)

Example from my research:
learning about evolution from genetic data

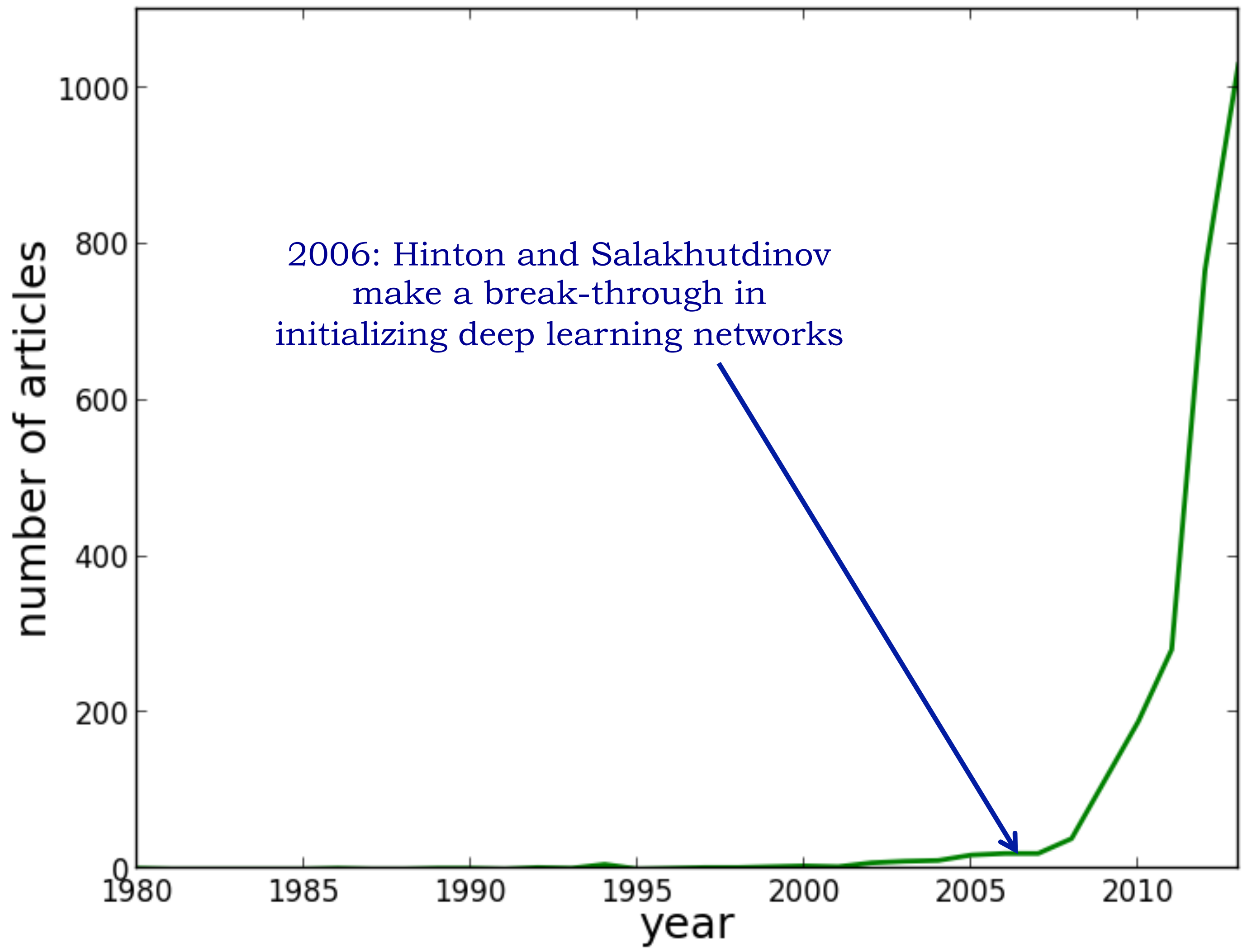


"Deep learning for population genetic inference", *PLOS Comp Bio*, 2016

History of Neural Networks

- Perceptron can be interpreted as a simple neural network
- Misconceptions about the weaknesses of perceptrons contributed to declining funding for NN research
- Difficulty of training multi-layer NNs contributed to second setback
- Mid 2000’s: breakthroughs in NN training contribute to rise of “deep learning”

Number of papers that mention "deep learning" over time



Big picture for today

- Neural networks can approximate any function!

Big picture for today

- Neural networks can approximate any function!

- For our purposes in ML, we want to use them to approximate a function from our inputs to our outputs

Big picture for today

- Neural networks can approximate any function!
- For our purposes in ML, we want to use them to approximate a function from our inputs to our outputs
- We will train our network by asking it to minimize the loss between its output and the true output

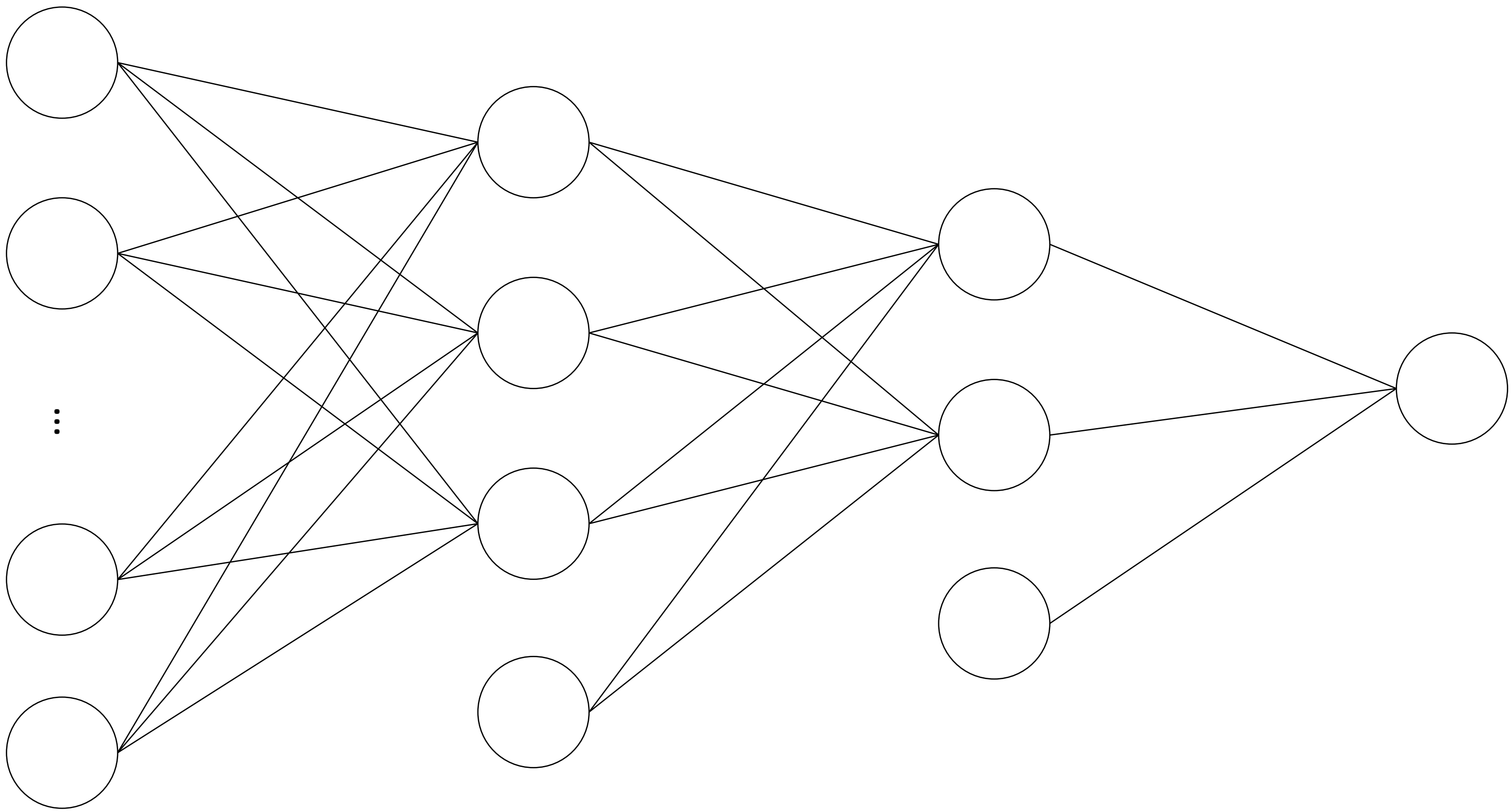
Big picture for today

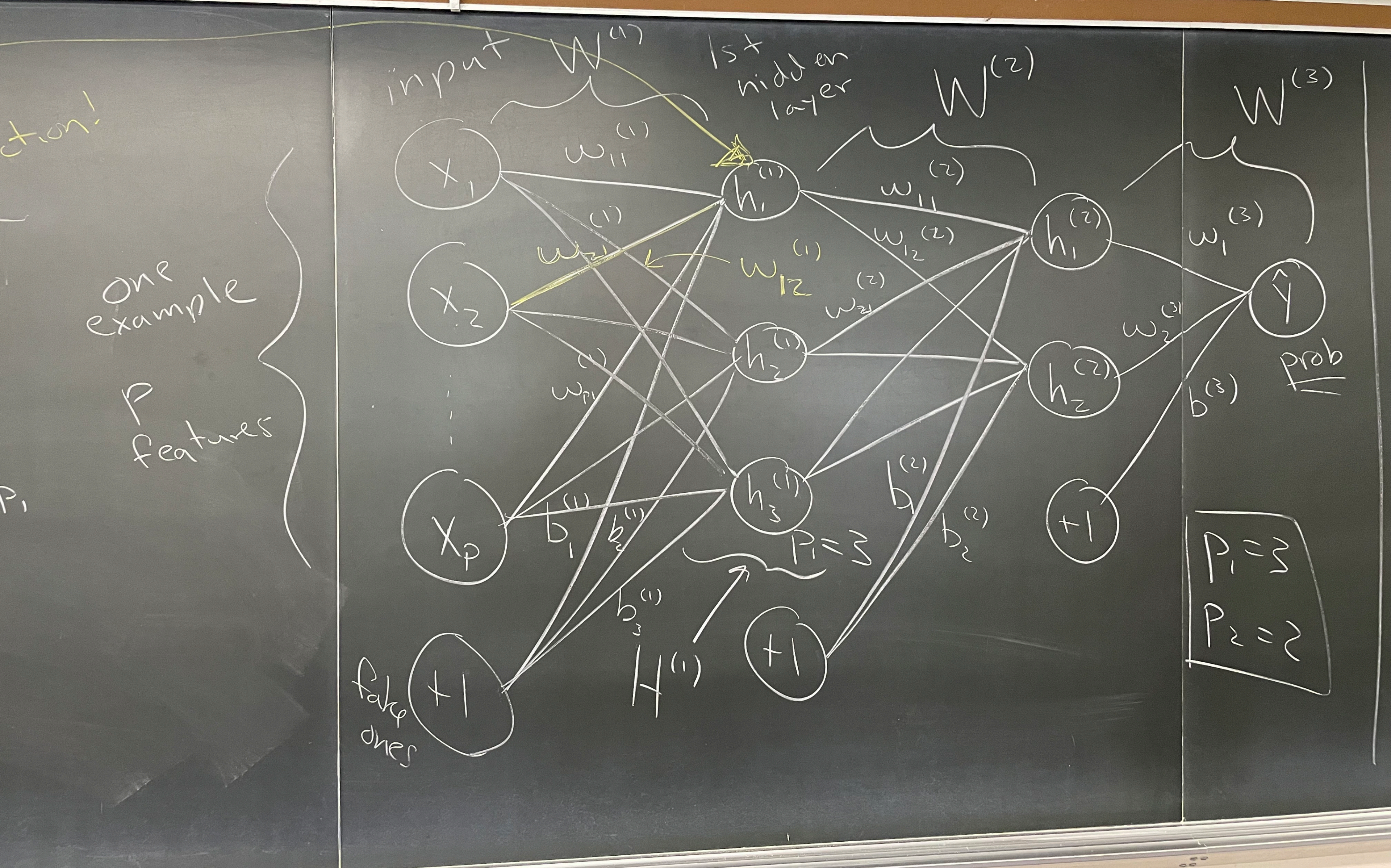
- Neural networks can approximate any function!
- For our purposes in ML, we want to use them to approximate a function from our inputs to our outputs
- We will train our network by asking it to minimize the loss between its output and the true output
- We will use SGD-like approaches to minimize loss

Outline for March 26

- SVM extensions
- Introduction to neural networks
- Fully connected (FC) neural networks
- Image data format and intro to Lab 7

Fully Connected Neural Network Architecture





ction!
one example
p features
input
$W^{(1)}$
1st hidden layer
$W^{(2)}$
$W^{(3)}$
x_1
x_2
x_p
+1
fake ones
$w_{11}^{(1)}$
$w_{21}^{(1)}$
$w_{12}^{(1)}$
$w_{p1}^{(1)}$
$b_1^{(1)}$
$b_2^{(1)}$
$b_3^{(1)}$
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$p_1 = 3$
$H^{(1)}$
$w_{11}^{(2)}$
$w_{12}^{(2)}$
$w_{21}^{(2)}$
$b_1^{(2)}$
$b_2^{(2)}$
$h_1^{(2)}$
$h_2^{(2)}$
$w_1^{(3)}$
$w_2^{(3)}$
$b^{(3)}$
$\hat{y}$
prob
$p_1 = 3$
$p_2 = 2$

$$X = \begin{bmatrix} \text{---}\ \vec{x}_i\ \text{---} \\ \\ \\ \end{bmatrix}_{n\times p}$$

$$b^{(1)} = \begin{bmatrix} -7 \\ 2 \\ 0.1 \end{bmatrix}$$

first hidden layer

$$\boxed{h_1^{(1)} = a\left(\underbrace{\vec{w}_1^{(1)} \cdot \vec{x} + b_1^{(1)}}_{\text{linear function!}}\right)}$$

activation function

$$H^{(1)} = a\left(\underbrace{X_{n\times p} W^{(1)}_{p\times p_1}}_{n\times p_1} + \underbrace{b^{(1)}_{p_1\times 1}}_{\text{broadcast to } n\times p_1}\right)$$

applied element-wise

one example

p features

$$H^{(2)} = a\left(H^{(1)}_{n\times p_1} W^{(2)}_{p_1\times p_2} + b^{(2)}_{p_2\times 1}\right)$$

$$\hat{y} = a\left(H^{(2)}_{n\times p_2} W^{(3)}_{p_2\times 1} + b^{(3)}_{1\times 1}\right)$$

$\boxed{n\times 1}$ prediction for each example

fake ones

K classes

5 classes

$$\text{pred } \hat{y} = \left[\frac{1}{10}, \frac{1}{10}, \frac{3}{10}, \frac{4}{10}, \frac{1}{10}\right]$$

$$y = [0, 0, 0, 1, 0]$$

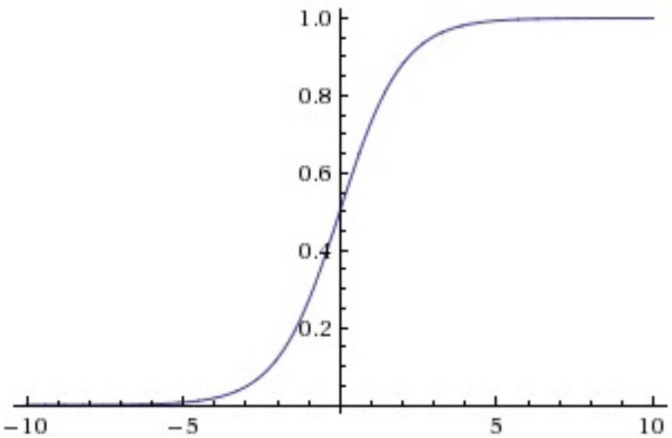
$$H(y, \hat{y}) = -1 \cdot \log_2\left(\frac{4}{10}\right)$$

$$H(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log_2 \hat{y}_k$$

Option 1: sigmoid function

- Input: all real numbers, output: [0, 1]

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



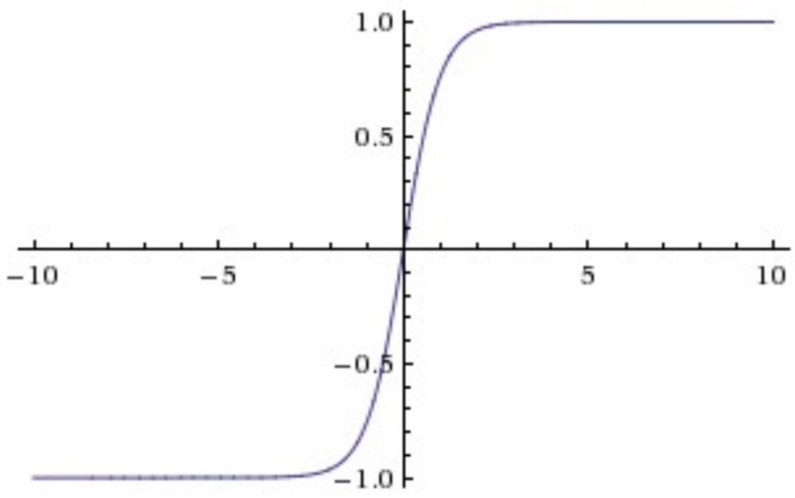
- Derivative is convenient

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Option 2: hyperbolic tangent

- Input: all real numbers, output: [-1, 1]

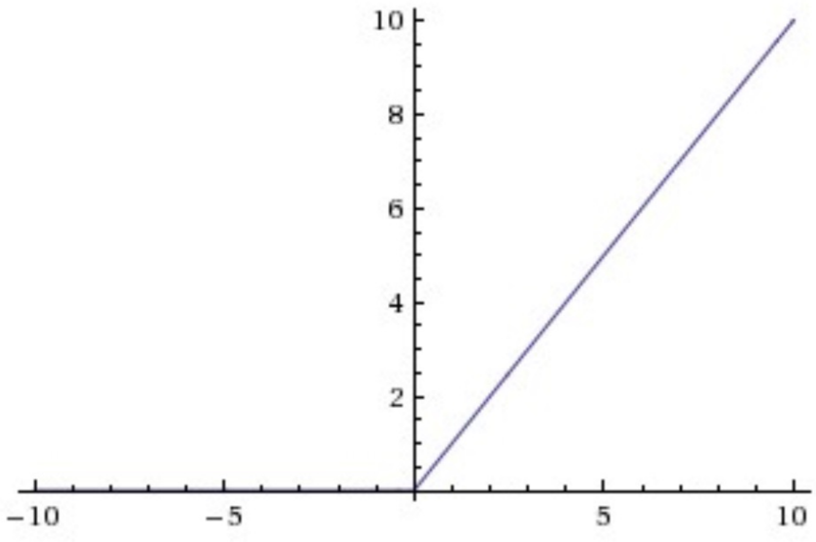
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Option 3: Rectified Linear Unit (ReLU)

- Return *x* if *x* is positive (i.e. threshold at 0)

$$\boxed{f(x) = \max(0, x)}$$



Pros and Cons of Activation Functions

1) Sigmoid

- (-) When input becomes very positive or very negative, gradient approaches 0 (saturates and stops gradient descent)
- (-) Not zero-centered, so gradient on weights can end up all positive or all negative (zig-zag in gradient descent)
- (+) Derivative is easy to compute given function value!

More info:
http://cs231n.github.io/neural-networks-1/

Pros and Cons of Activation Functions

1) Sigmoid

- (-) When input becomes very positive or very negative, gradient approaches 0 (saturates and stops gradient descent)
- (-) Not zero-centered, so gradient on weights can end up all positive or all negative (zig-zag in gradient descent)
- (+) Derivative is easy to compute given function value!

2) Tanh

- (-) Still has a tendency to prematurely kill the gradient
- (+) Zero-centered so we get a range of gradients
- (+) Rescaling of sigmoid function so derivative is also not too difficult

More info:
http://cs231n.github.io/neural-networks-1/

Pros and Cons of Activation Functions

1) Sigmoid

- (-) When input becomes very positive or very negative, gradient approaches 0 (saturates and stops gradient descent)
- (-) Not zero-centered, so gradient on weights can end up all positive or all negative (zig-zag in gradient descent)
- (+) Derivative is easy to compute given function value!

2) Tanh

- (-) Still has a tendency to prematurely kill the gradient
- (+) Zero-centered so we get a range of gradients
- (+) Rescaling of sigmoid function so derivative is also not too difficult

3) ReLU

- (+) Works well in practice (accelerates convergence)
- (+) Function value very easy to compute! (no exponentials)
- (-) Units can "die" (no signal) if input becomes too negative throughout gradient descent

More info:
http://cs231n.github.io/neural-networks-1/

Mini-batches

- So far in this class, we have considered *stochastic gradient descent*, where one data point is used to compute the gradient and update the weights
- On the flipside is *batch gradient descent*, where we compute the gradient with respect to all the data, and then update the weights
- A middle ground uses *mini-batches* of examples before updating the weights. This is the approach we will use in Lab 7.

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)

- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

- Then we apply *cross-entropy loss* to these probabilities

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)

- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

Think about outside of class:
- Why do we use exp?
- Why don't we just take the max score?

- Then we apply *cross-entropy loss* to these probabilities

1. $\sigma(0) = \frac{1}{2}$

   $\tanh(0) = 0$

   $\mathrm{ReLU}(0) = 0$

2. $$(\underset{\uparrow\, p}{3}+1)\,\underset{\uparrow\, p_1}{4} + (\underset{\uparrow\, p_1}{4}+1)\cdot \underset{\uparrow\, p_2}{4} + (\underset{\uparrow\, p_2}{4}+1)\,\underset{\uparrow\, \text{out}}{1}$$

   $$= \boxed{41}$$

3. $H(y, \hat{y}) = 1$

   $H(y, \hat{y}) = 3$

One example

$$\underbrace{(32, 32, 3)}$$

$$32 \cdot 32 \cdot 3 = \underset{p}{\boxed{3072}}$$

Handout 17

Outline for March 26

- SVM extensions
- Introduction to neural networks
- Fully connected (FC) neural networks
- Image data format and intro to Lab 7

Lab 7 data pre-processing

- It is helpful to have our data be zero-centered, so we will subtract off the mean

- It is also helpful to have the features be on the same scale, so we will divide by the standard deviation

- We will compute the mean and std with respect to the *training data*, then apply the same transformation to all datasets

Lab 7 data pre-processing

- Input is now itself a multi-dimensional array
  - Also known as a **tensor**!

- For images, often the shape of each image will be (width, height, 3) for RGB channels

- Need to "**flatten**" or "unravel" for fully connected networks