CS 360: Machine Learning

Prof. Sara Mathieson

Fall 2019

HAVERFORD COLLEGE

Adapted from: “Know Your Meme”

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

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

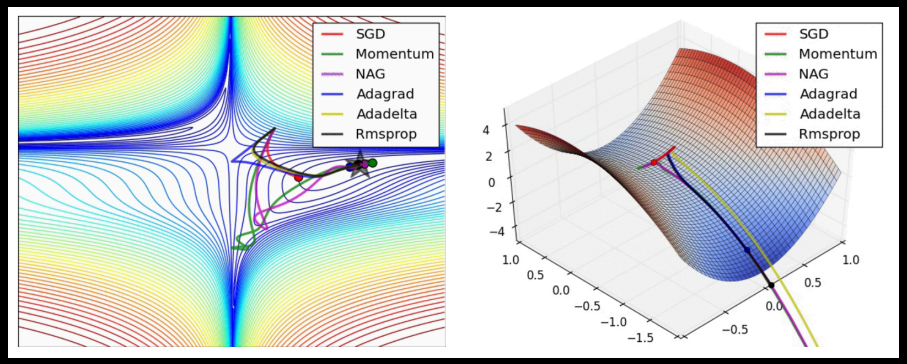
But from Equation (10), the last term must be zero, so we obtain

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

What my boss thinks I do

What other computer scientists think I do

What mathematicians think I do



What I think I do

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

What I really do

Adapted from: “Know Your Meme”

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

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

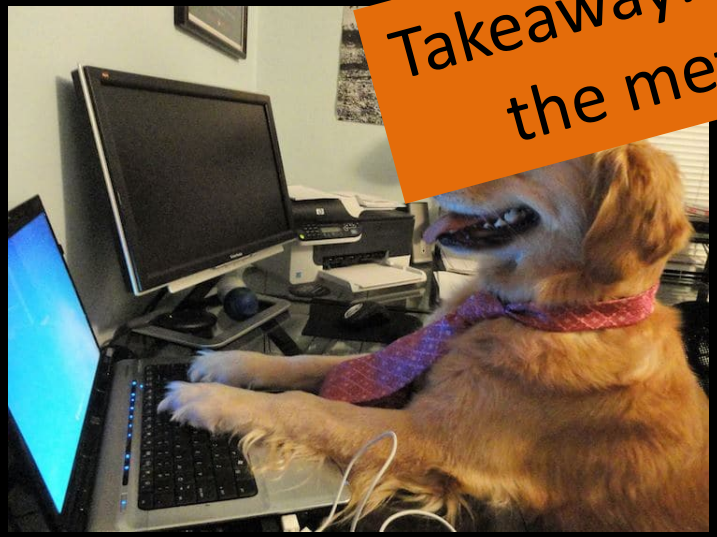
But from Equation (10), the last term must be zero, so we obtain

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

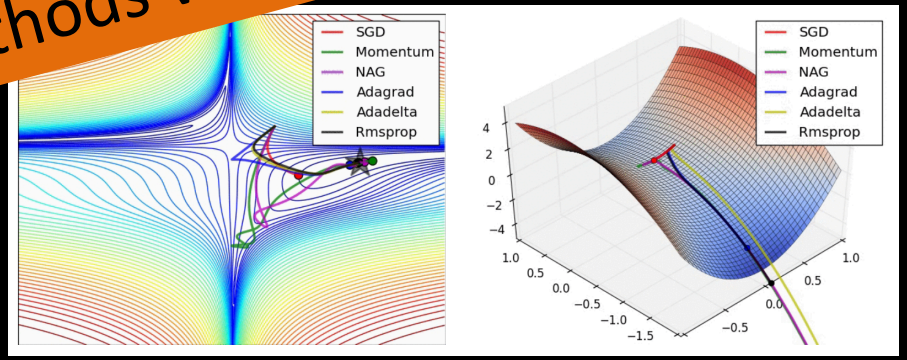
Wha

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

Admin

- Office hours **Friday 4-5pm** (in lab)
- Lab 7 due Sunday Nov 10
- No class Tuesday Nov 12 (away at conference)
  - time for project proposal!
- Project proposal due Wednesday Nov 13
- Lab 8 due Sunday Nov 17
- Midterm 2: Nov 21 (in-class) + take home due Tues Nov 26
- Nov 28-29: Thanksgiving break!

Outline for November 7

- Reading Quiz
- Revisit cross-validation
- Handout 17 solution/recap
- Introduction to neural networks
  - Fully connected architectures
  - Convolutional neural networks

- Lab check in TODAY! (start of either the problem set or the coding part)

Outline for November 7

- Reading Quiz
- Revisit cross-validation
- Handout 17 solution/recap
- Introduction to neural networks
  - Fully connected architectures
  - Convolutional neural networks

Reading Quiz #8

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

Reading Quiz #8

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

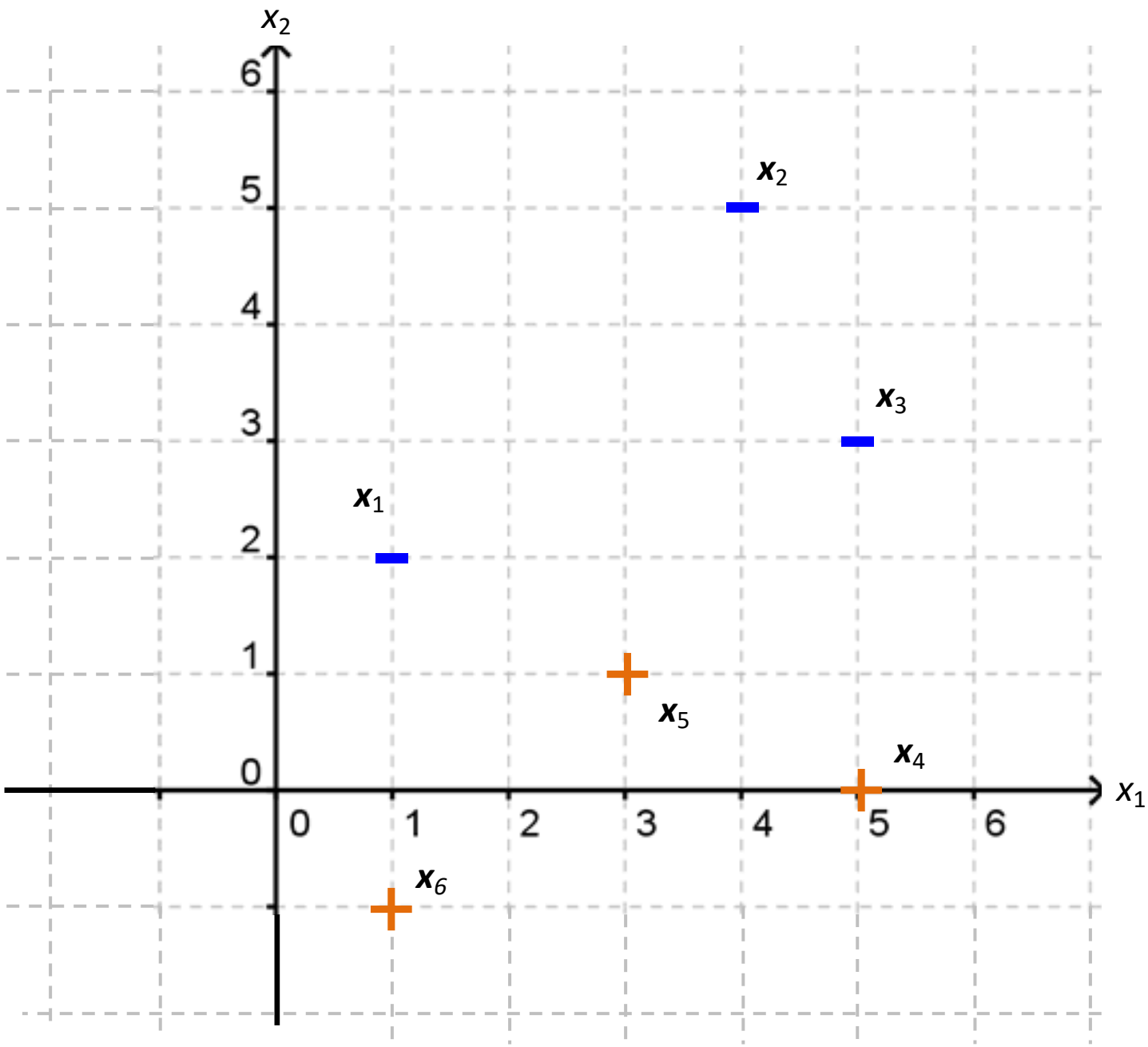
   (a) its Lagrange multiplier $\alpha_i > 0$

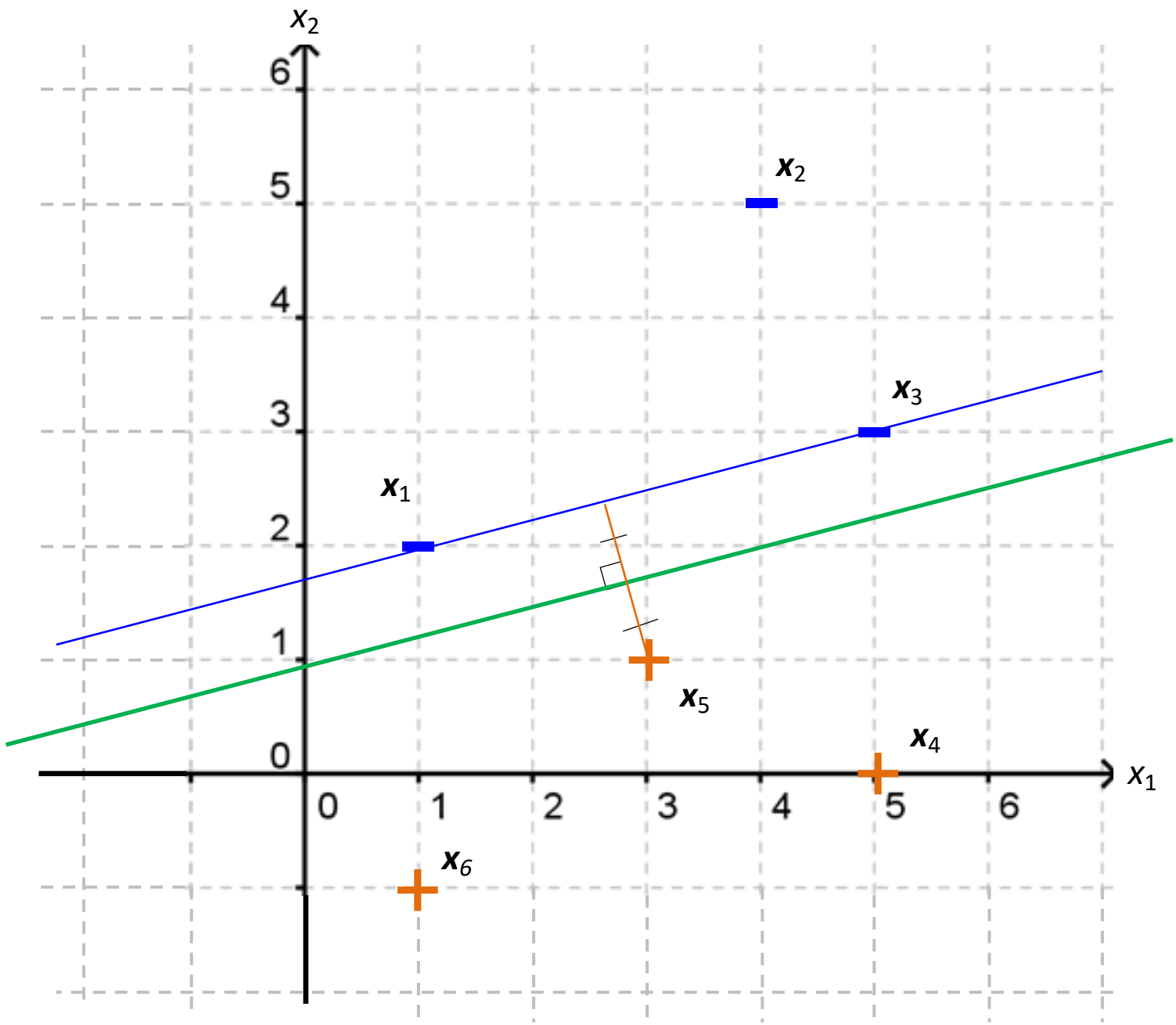
   (b) its Lagrange multiplier $\alpha_i = 0$

   (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


x_2
6
5
4
3
2
1
0
x_1
0
1
2
3
4
5
6
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_4$
$\boldsymbol{x}_5$
$\boldsymbol{x}_6$


x_2
6
5
4
3
2
1
0
x_1
0
1
2
3
4
5
6
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_4$
$\boldsymbol{x}_5$
$\boldsymbol{x}_6$

Reading Quiz #8

3. After training an SVM and obtaining the α values for each training example, I can use this formula to find the optimal weight vector:

$$\vec{w}^* = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i$$

Then when I predict a label for a test example $\vec{x}$, I can use:

$$\hat{y} = h(\vec{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i (\vec{x}_i \cdot \vec{x}) + b\right)$$

Explain why it does not take $O(n)$ work to predict a label for each test point.

Reading Quiz #8

3. After training an SVM and obtaining the α values for each training example, I can use this formula to find the optimal weight vector:

$$\vec{w}^* = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i$$

Then when I predict a label for a test example $\vec{x}$, I can use:

$$\hat{y} = h(\vec{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i (\vec{x}_i \cdot \vec{x}) + b\right)$$

Explain why it does not take $O(n)$ work to predict a label for each test point.

Most of the alpha values are 0, so we only need to consider the support vectors!

Outline for November 7

- Reading Quiz
- Revisit cross-validation
- Handout 17 solution/recap
- Introduction to neural networks
  - Fully connected architectures
  - Convolutional neural networks

Cross-Validation revisited

- Separate the data into K-folds
- For each fold we have train/test
  - Train: perform another cross-validation to choose the best hyper-parameters
  - Test: use these best hyper-parameters

Cross-Validation revisited

- Separate the data into K-folds
- For each fold we have train/test
  - Train: perform another cross-validation to choose the best hyper-parameters
  - Test: use these best hyper-parameters
- After K-fold cross-validation, choose the hyper-parameters that occurred most frequently
  - Use these params on held-aside test data to estimate generalization accuracy
  - OR: use these params on test data from each fold

Cross-Validation revisited

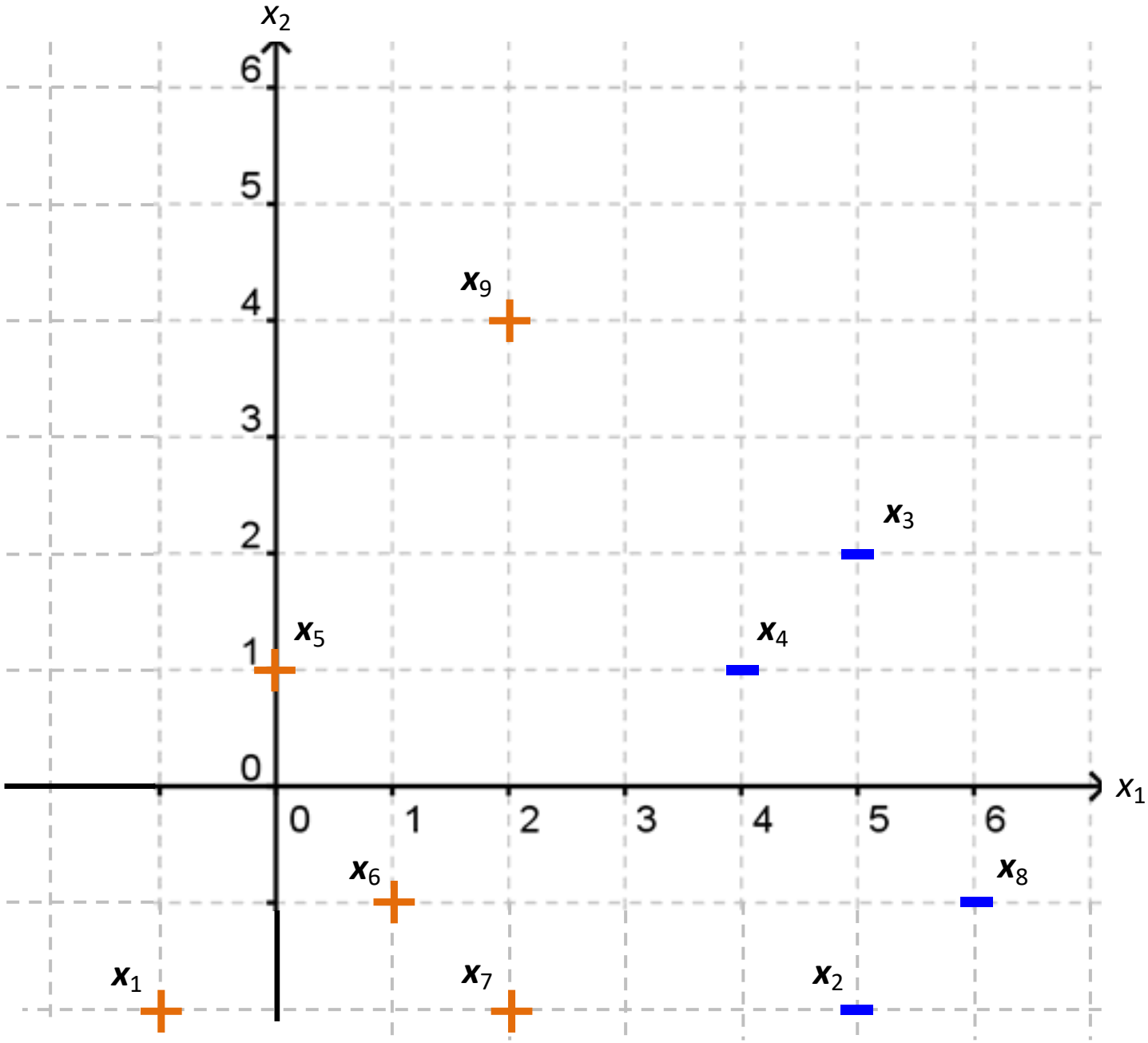
- Often we are trying to get an estimate of the generalization accuracy of the *type of method* (i.e. SVM vs. Random Forest)

- In this case we may have different hyper-parameters for each fold, but that's okay

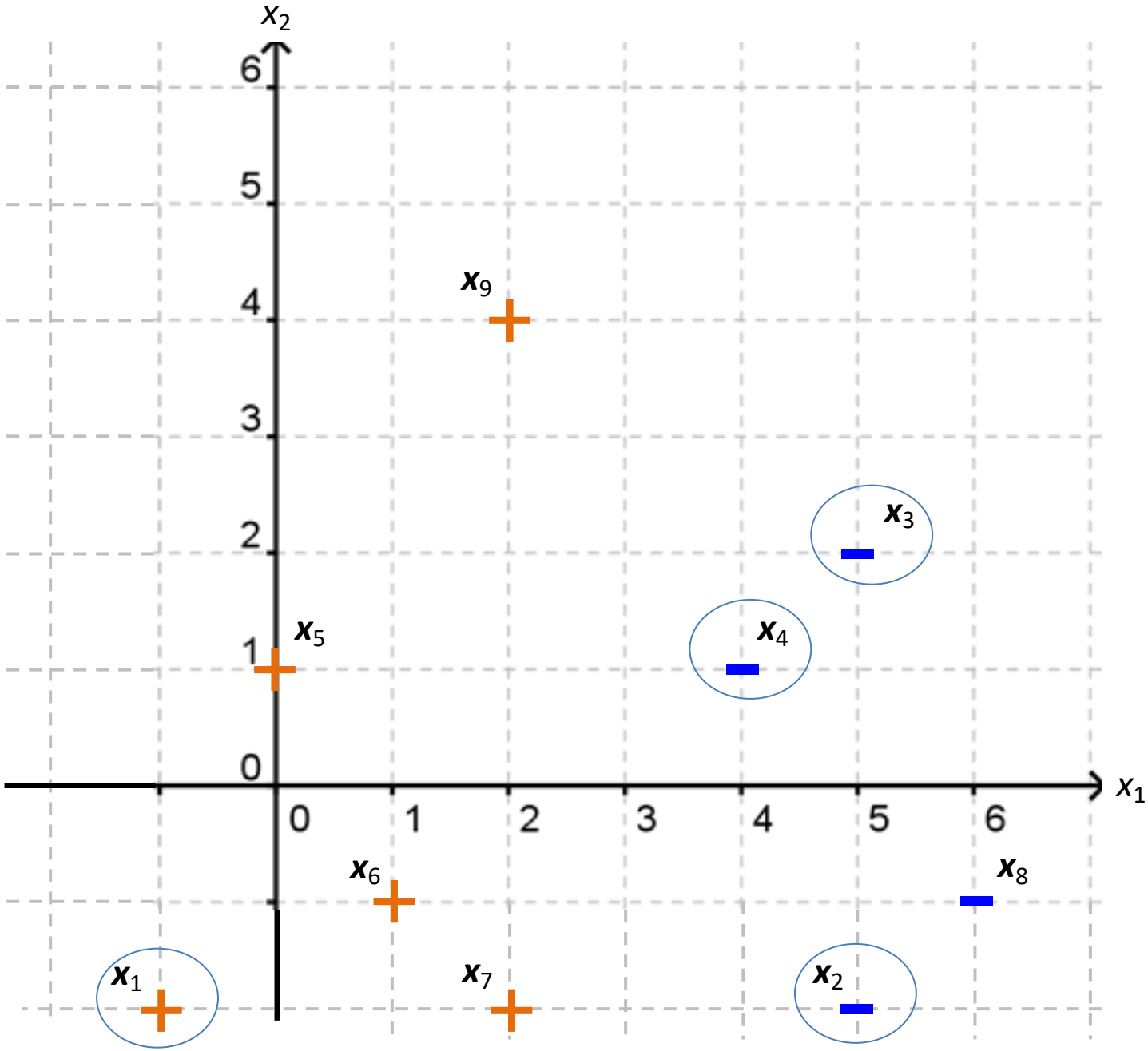
Outline for November 7

- Reading Quiz
- Revisit cross-validation
- Handout 17 solution/recap
- Introduction to neural networks
  - Fully connected architectures
  - Convolutional neural networks

Meta-optimization: example



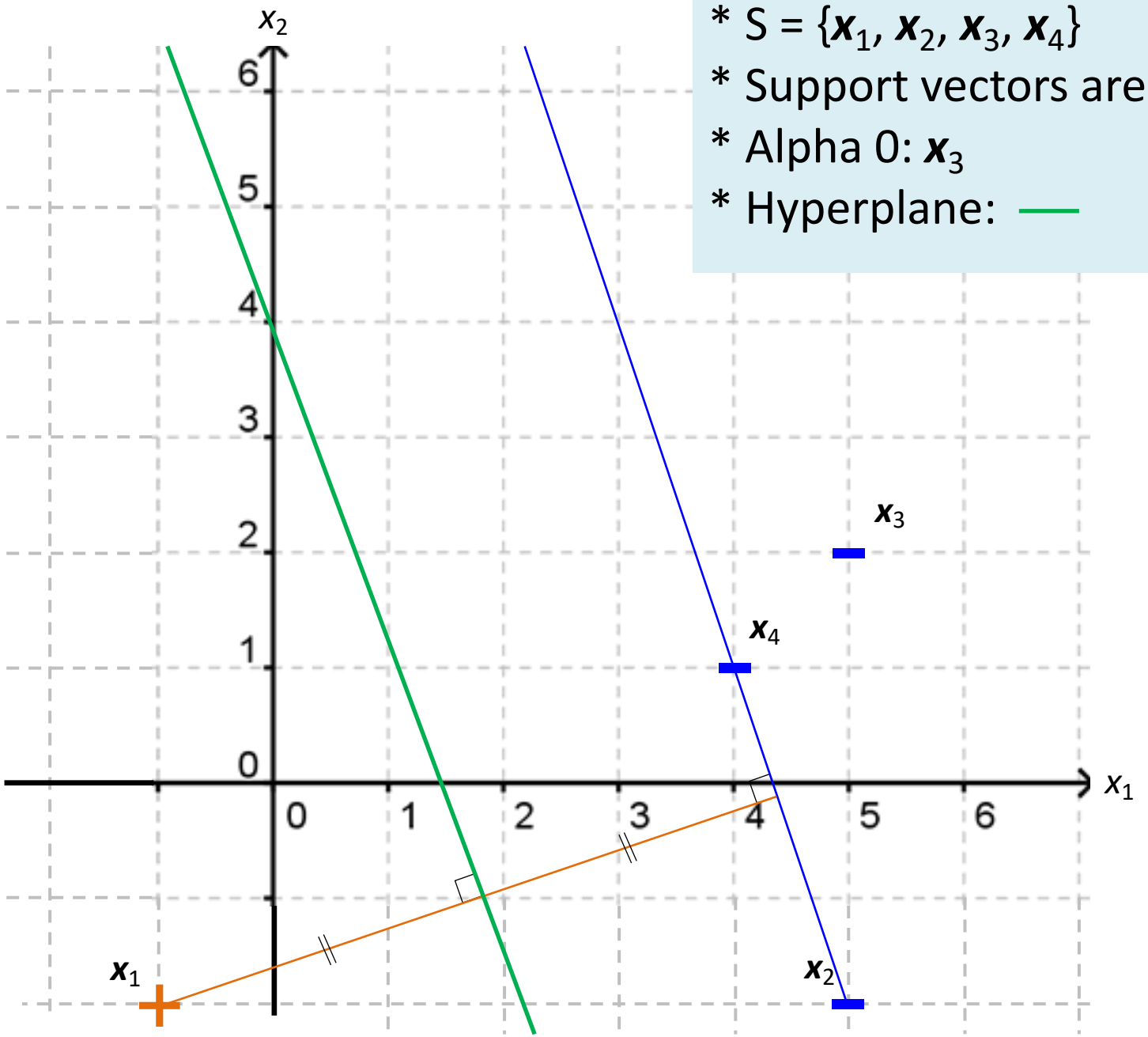
Meta-optimization: example



$K = 4$

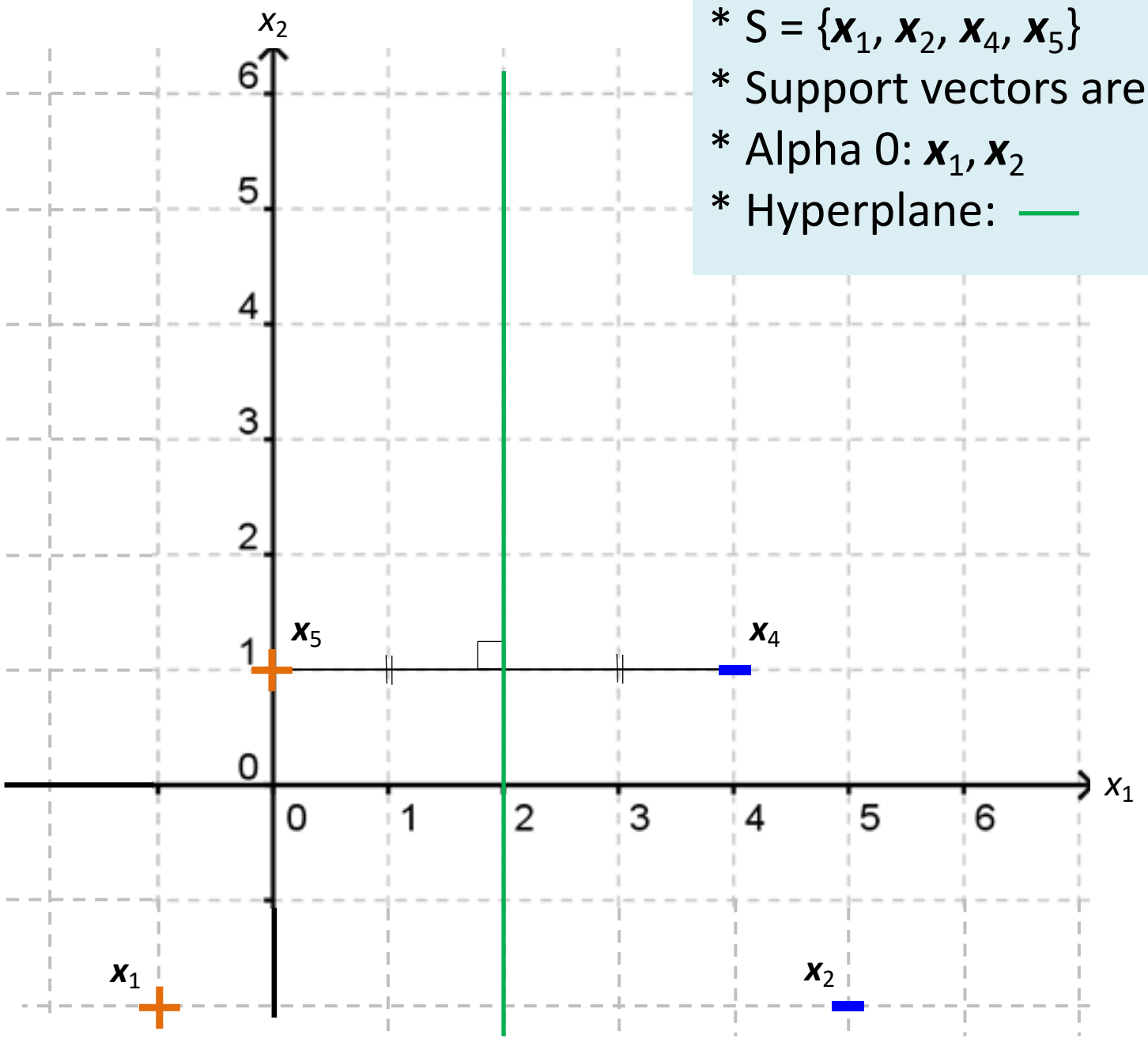
Round 1:

* 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:



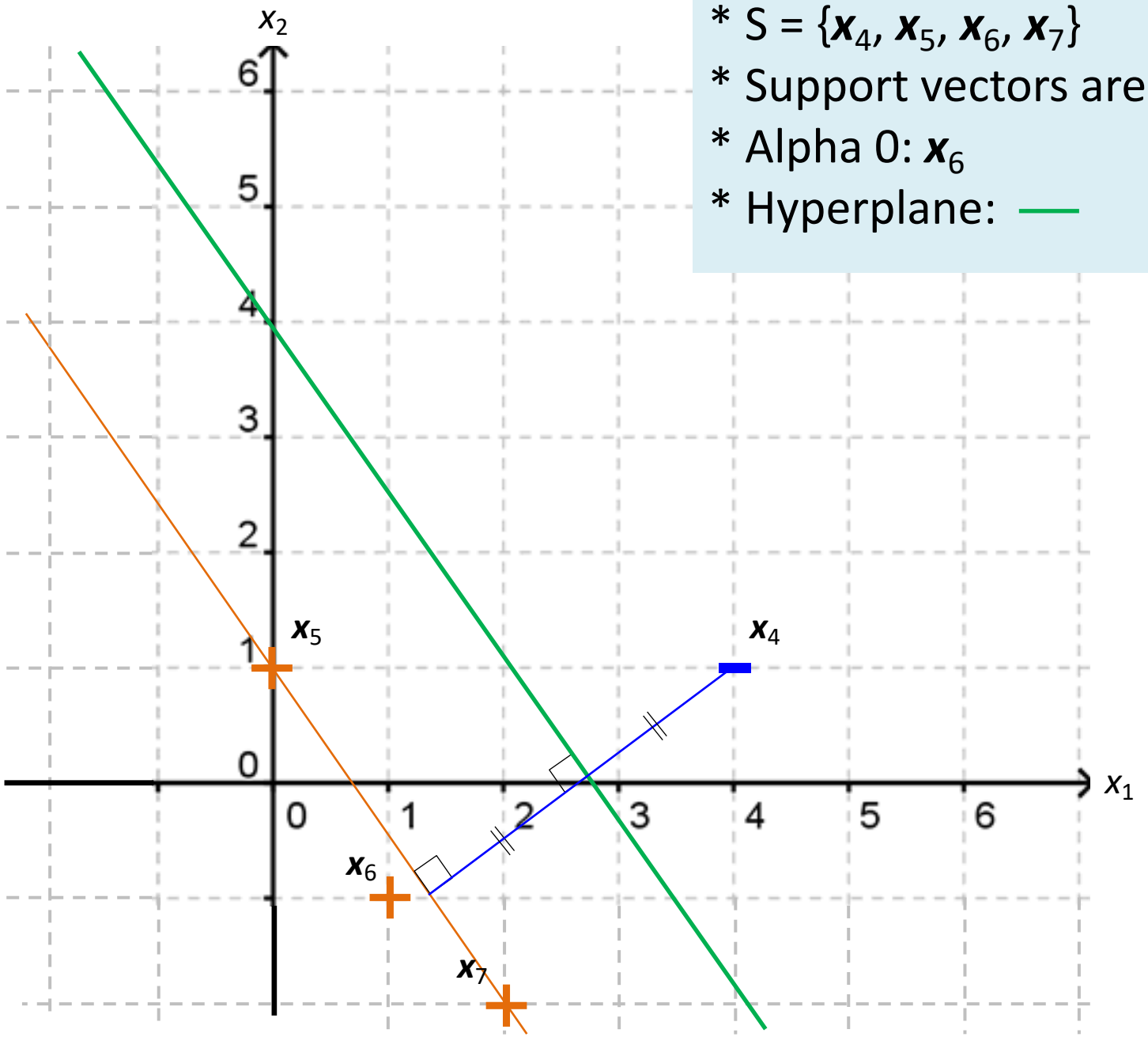
Round 1:

* 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:



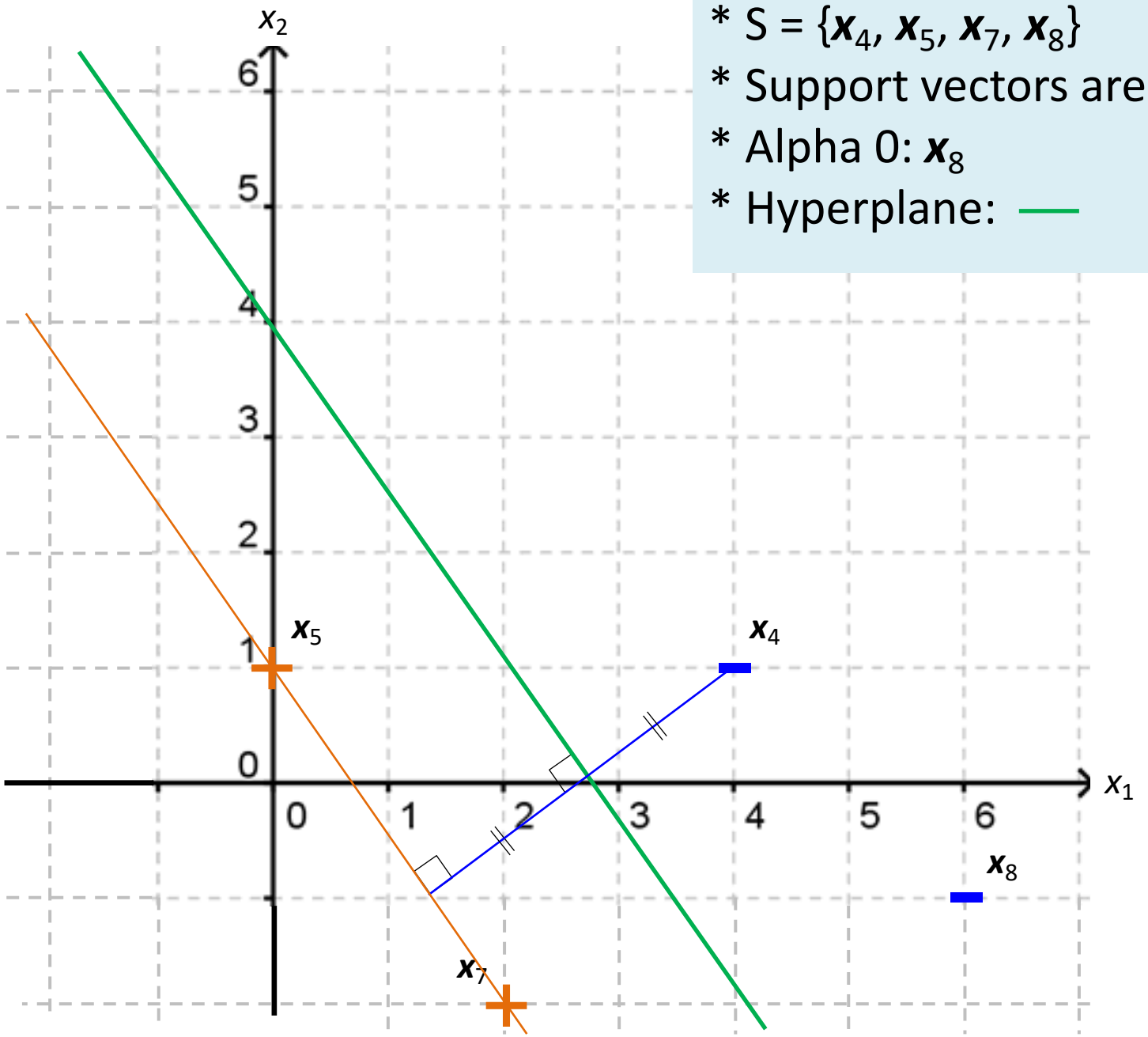
Round 3:

* 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:



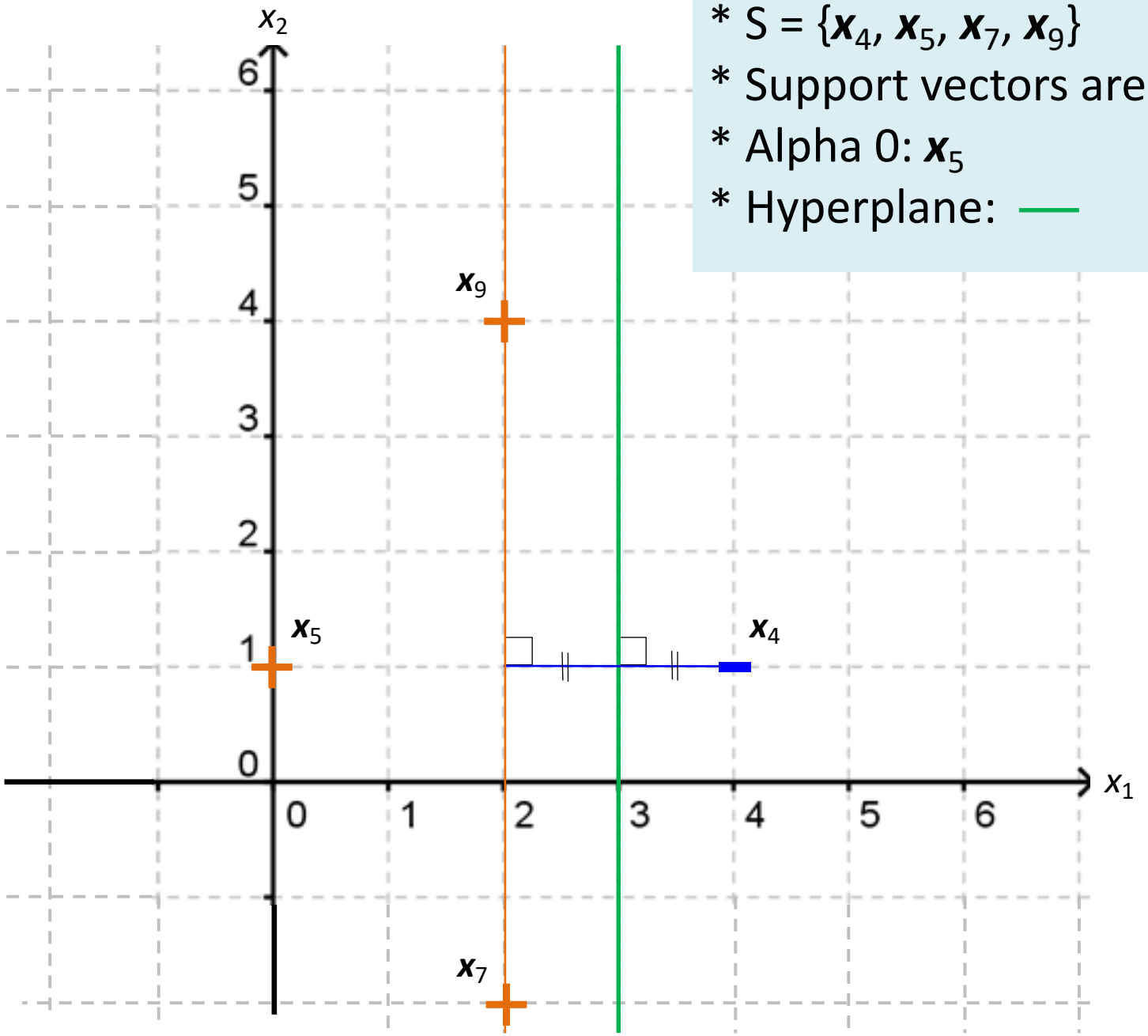
Round 4:

* 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:

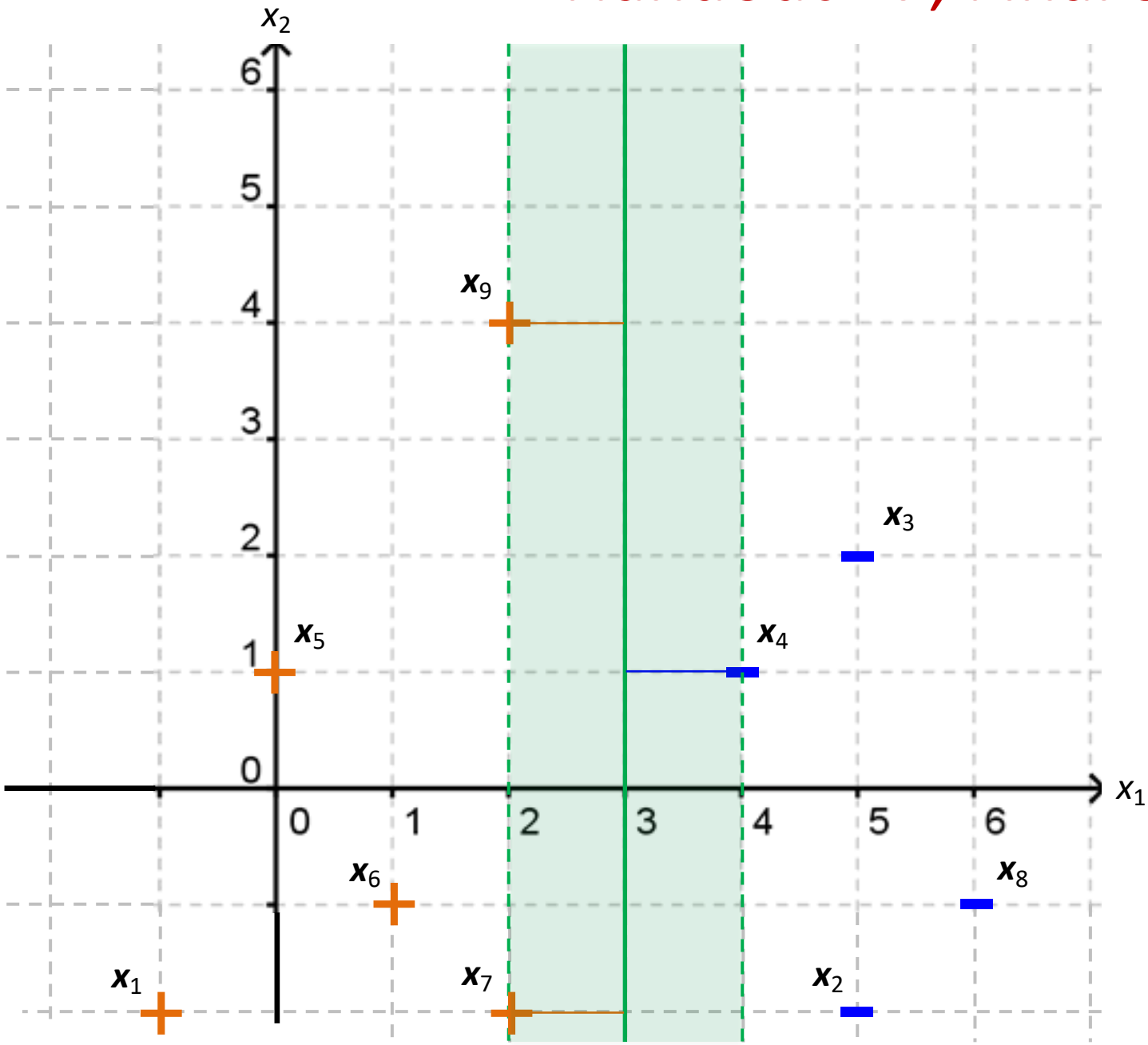


Round 5:

* 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 17, Final Solution



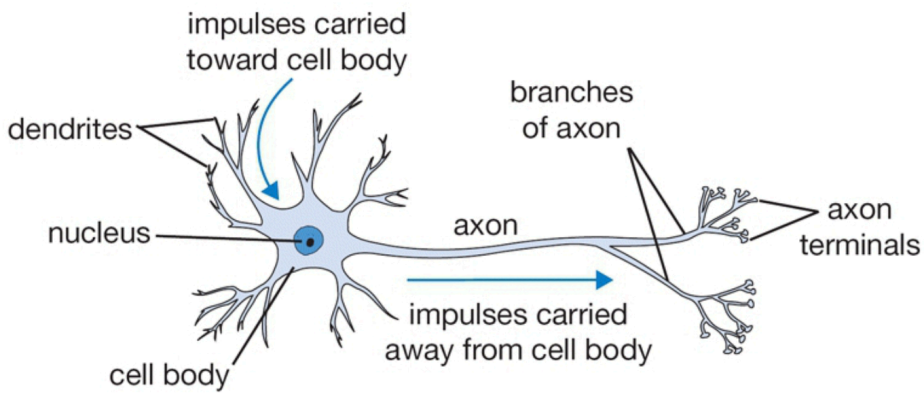
Outline for November 7

- Reading Quiz
- Revisit cross-validation
- Handout 17 solution/recap
- Introduction to neural networks
  - Fully connected architectures
  - Convolutional neural networks

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

Biological Inspiration



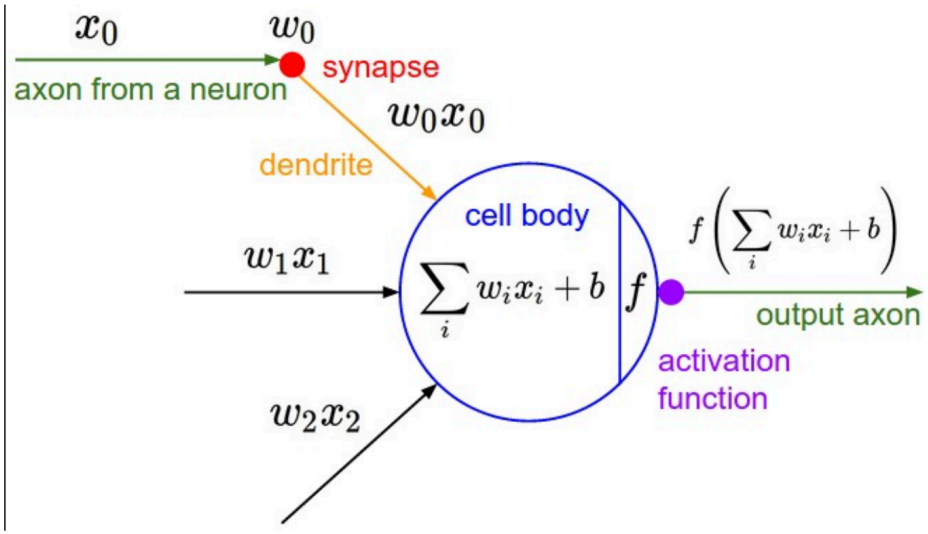


Figure: Stanford CS231n http://cs231n.github.io/neural-networks-1/

Goal: learn from complicated inputs

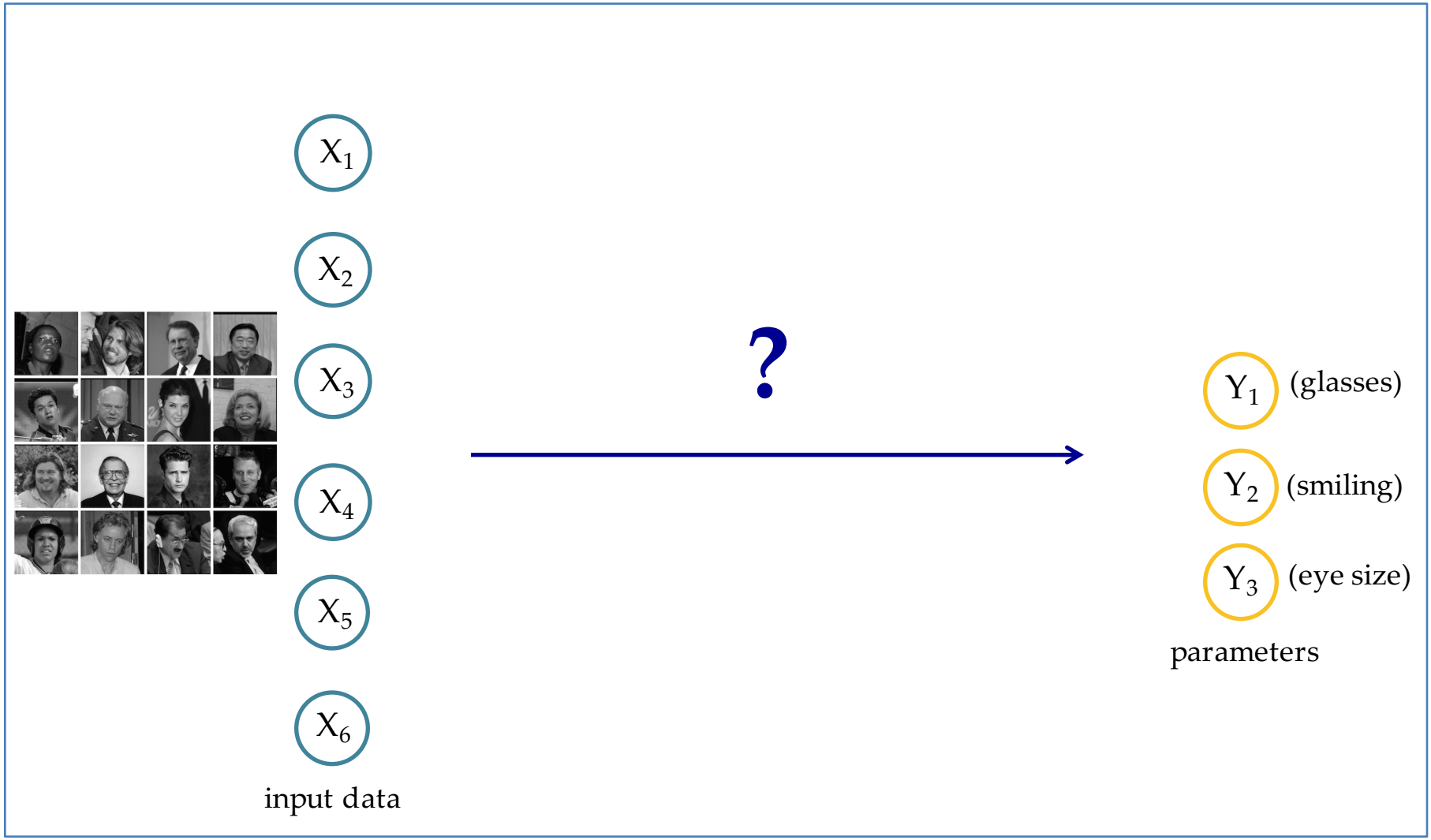


Image: Labeled Faces in the Wild (UMass)

Idea: transform data into lower dimension

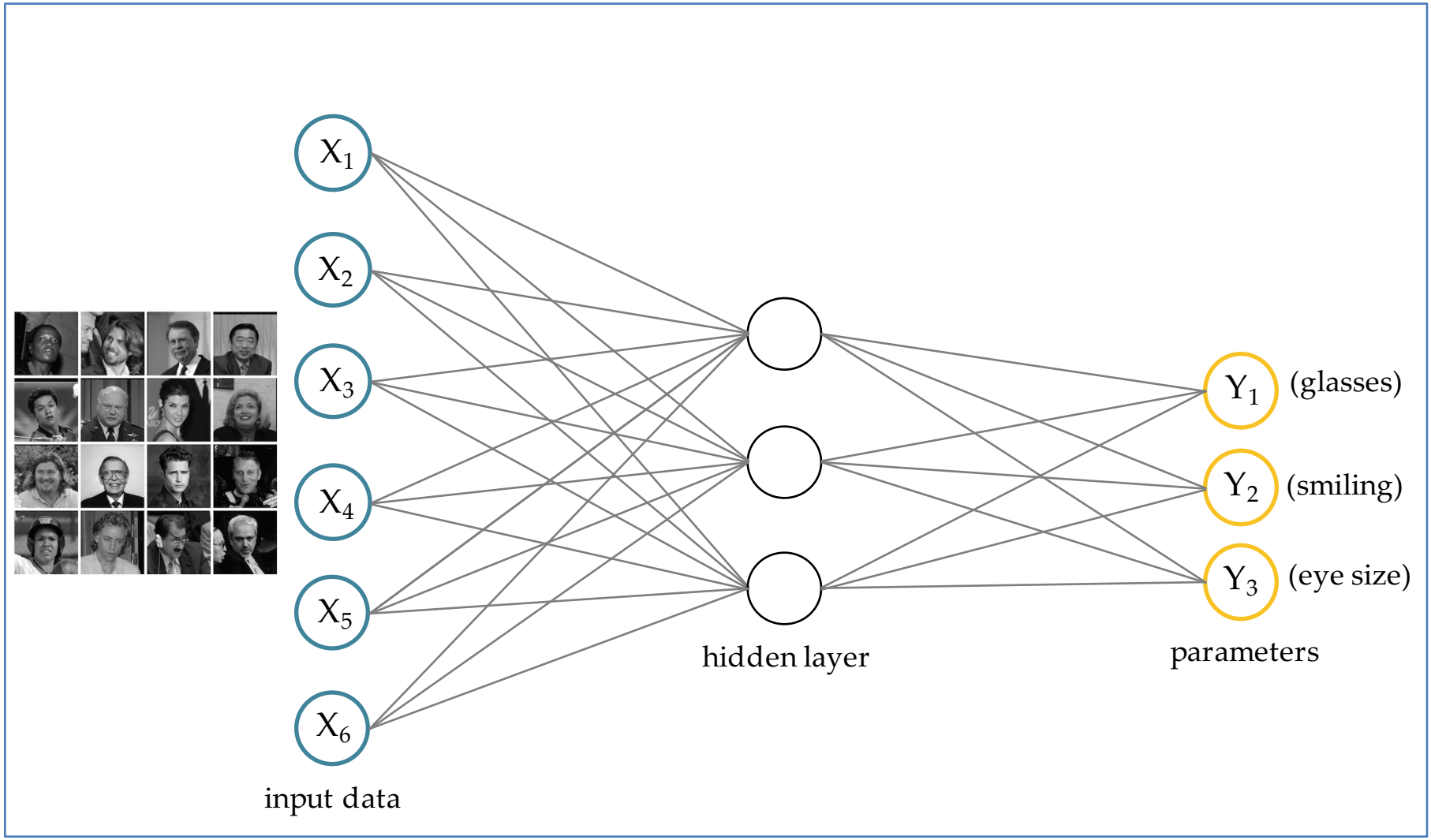


Image: Labeled Faces in the Wild (UMass)

Multi-layer networks = “deep learning”

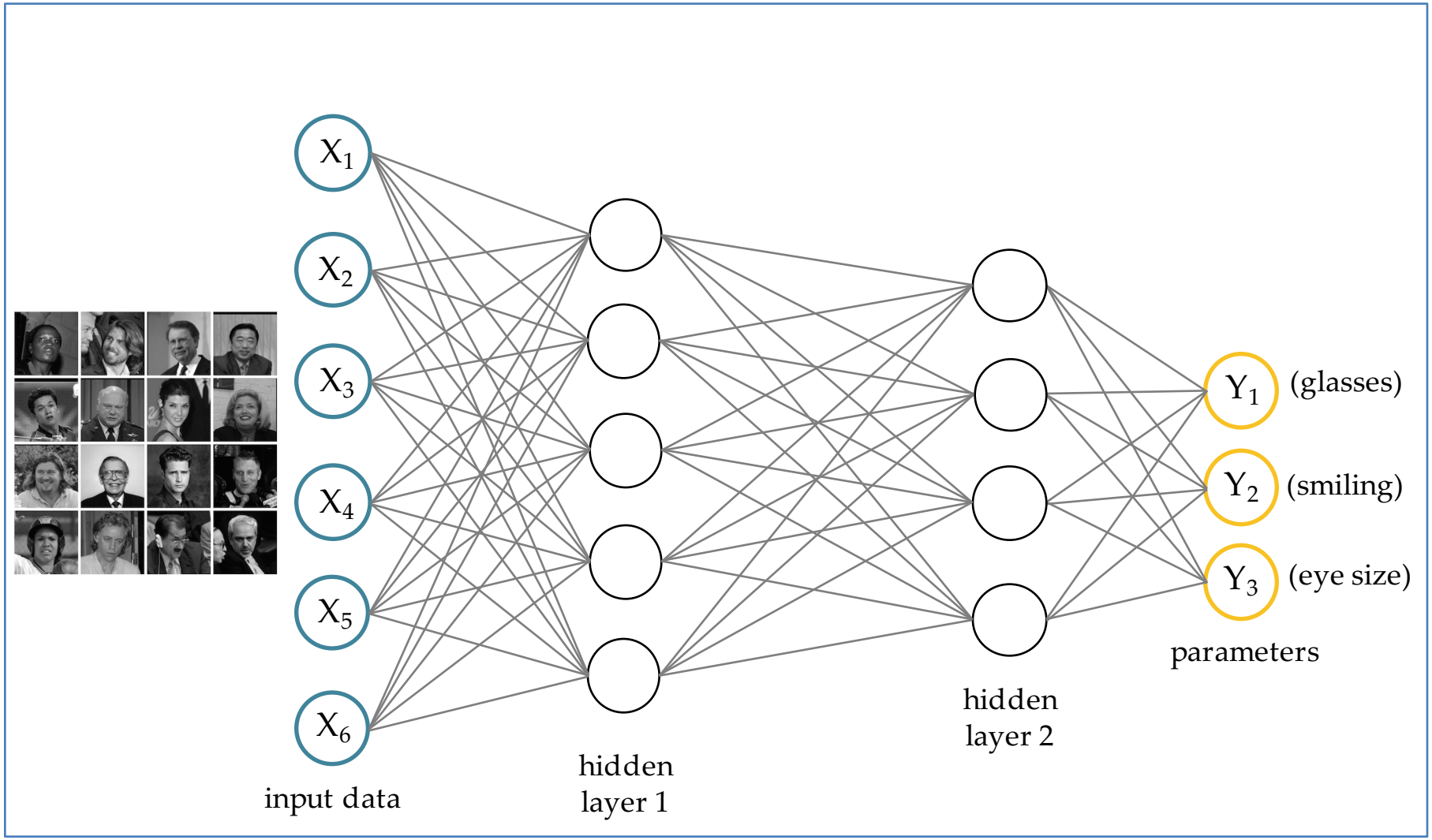
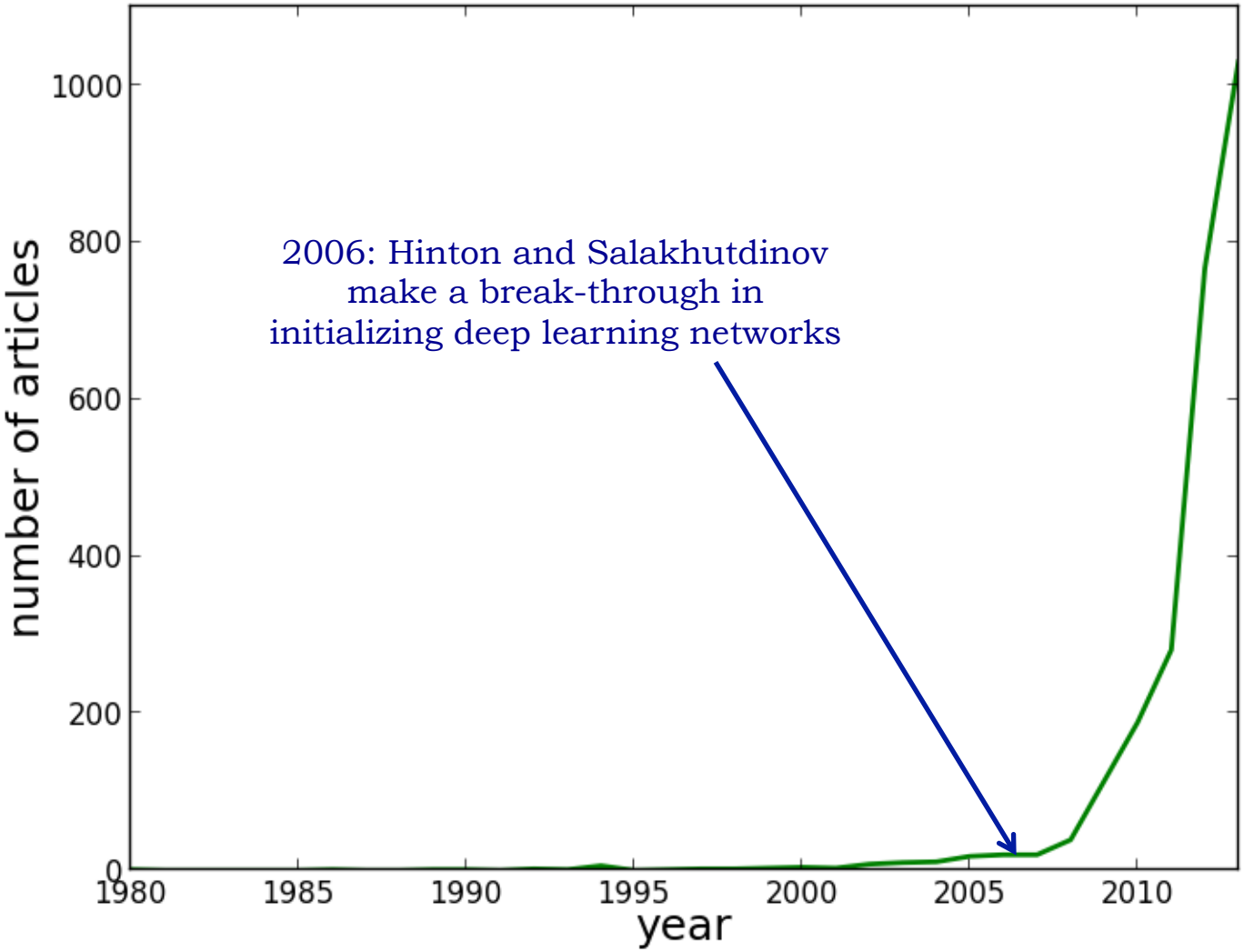


Image: Labeled Faces in the Wild (UMass)

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

$$X = \begin{bmatrix} | & | & & | \\ \vec{x}_1 & \vec{x}_2 & \cdots & \vec{x}_n \\ | & | & & | \end{bmatrix}$$

$p \times n$

$$b = \begin{bmatrix} | & | & & | \\ \vec{b}^{(1)} & \vec{b}^{(1)} & \cdots & \vec{b}^{(1)} \\ | & | & & | \end{bmatrix}$$

$p_1 \times n$

$$\underset{p_1 \times n}{H^{(1)}} = a\left(\underset{p_1 \times p}{W^{(1)}} \underset{p \times n}{X} + \underset{p_1 \times 1}{\vec{b}^{(1)}} \right)$$

"broadcast" into $p_1 \times n$

"activation function" / "non-linearity" } applied element-wise.

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

$\Rightarrow p_2 \times n$

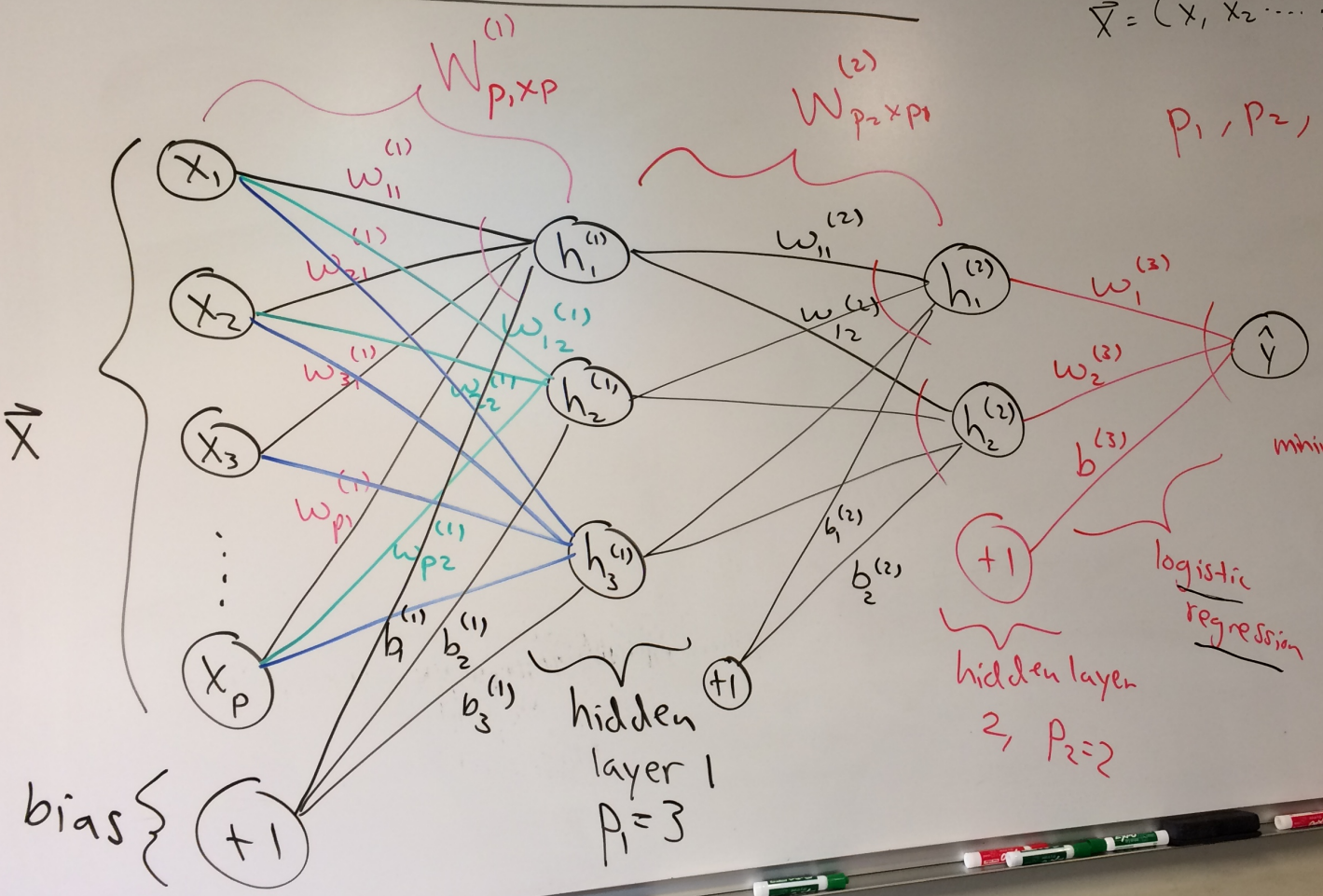
$$H^{(3)} = \underset{1 \times n}{\hat{\vec{y}}} = a\left(\underset{1 \times p_2}{W^{(3)}} \underset{p_2 \times n}{H^{(2)}} + \underset{1 \times 1 \Rightarrow 1 \times n}{\vec{b}^{(3)}} \right)$$

Fully Connected Neural Network

$\vec{X} = (x_1, x_2 \cdots x_p)^T$ $\quad$ Y is some output

$p_1, p_2, L \Rightarrow$ hyperparameters

L: # of layers



Loss

minimize $\text{loss}(\hat{Y}, Y)$

Cross - Entropy (discrete)

probability distributions p & q

$$H(p,q) = -\sum_{x \in X} p(x) \log q(x)$$

($K=2$ # classes) $H(y,\hat{y}) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$

$K>2 \Rightarrow$ $$\boxed{H(\underbrace{\vec{y},\hat{\vec{y}}}_{\text{one for every class}}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)}$$

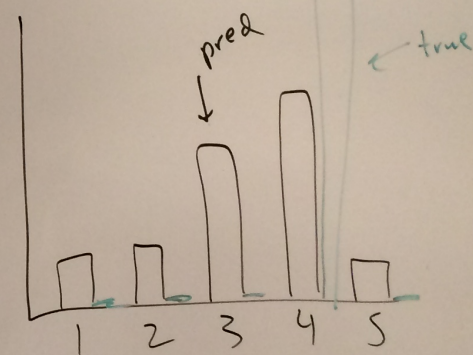
$K=5$

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

$\vec{y} = [0, 0, 0, 1, 0]$ true

true label was 4

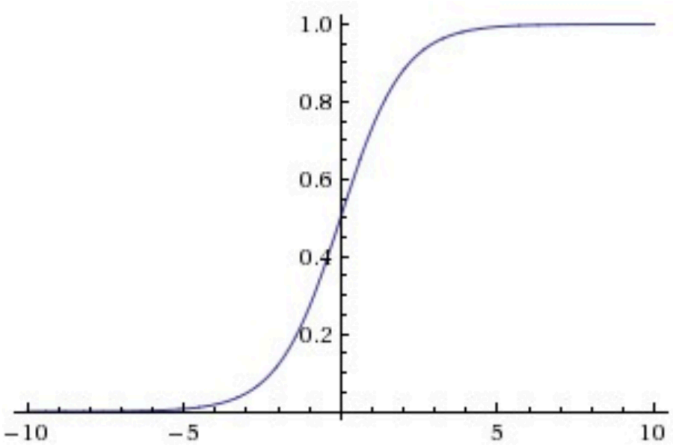
$$\boxed{H(\vec{y},\hat{\vec{y}}) = -1 \cdot \log_2\left(\frac{4}{10}\right)}$$



Option 1: sigmoid function

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

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



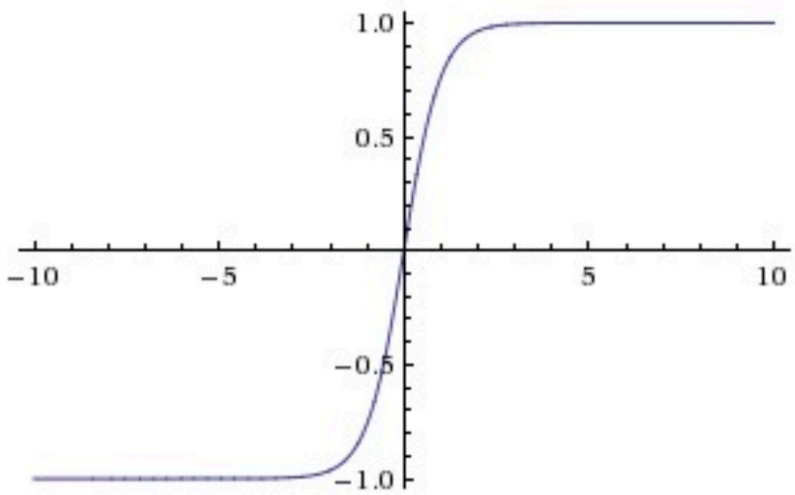
- Derivative is convenient

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

Option 2: hyperbolic tangent

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

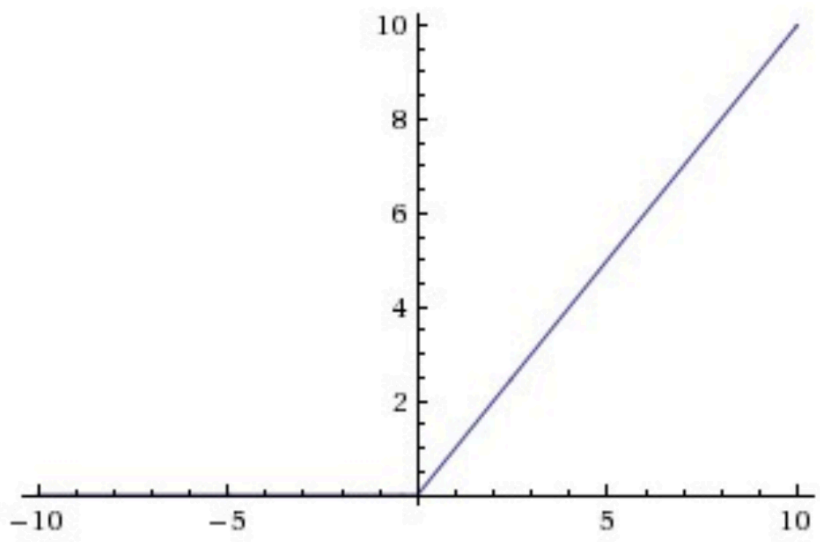
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Option 3: Rectified Linear Unit (ReLU)

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

$$f(x) = \max(0, x)$$



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/

Weight initialization

- All 0's initialization is bad! Causes nodes to compute the same outputs, so then the weights go through the same updates during gradient descent

- Need asymmetry! => usually use small random values

1

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

$$\tanh(0) = 0$$

$$f(0) = 0$$

2

$$3 \cdot 4 + 4$$

$$(3+1)\cdot 4 + (4+1)\cdot 4 + (4+1)\cdot 1$$

$$= \boxed{41}$$

3

$$\hat{y} = \left[\tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4}\right]$$

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

$$\Rightarrow \boxed{H(y, \hat{y}) = 1}$$

$$\hat{y} = \left[\tfrac{3}{8}, \tfrac{1}{8}, \tfrac{1}{2}\right]$$

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

Handout 18, first page