CS 360: Machine Learning

Sara Mathieson, Sorelle Friedler

Spring 2024

HAVERFORD
COLLEGE

Admin

- **Project meetings today** in lab with all groups
  - Try to come to the same lab as your group

- **Midterm April 25** in class (this Thursday)
  - Can still do handout videos for extra credit! Up to 24 hours before exam
  - Create your own “study sheet” (front and back) to use during the exam
  - No other notes or resources

Faculty Interview TODAY!

- The candidate joining us is Dakotah Lambert. Dakotah received their PHD from Stony Brook University in the Department of Linguistics and Institute for Advanced Computational Science, with a research focus on understanding communication structures between humans and computers using algebraic and logical analysis.
- Date: **23rd April 2024** (Today)
- Research talk: **4:30 pm** onwards **in H109**. The research talk will be preceded by a tea at around **4:15 pm**
- Title**: Finite automata: interpretation and normal forms**
- Abstract: In this lesson we will explore finite-state automata, a standard representation of abstract machines that implement regular expressions. These are commonly used, perhaps with extensions, in compiler design, language modeling, and machine learning. We will see how to read standard depictions of these abstract machines, to understand which languages they represent, and to determine whether two machines implement the same functionality.

Outline for April 23

- Review temperature, cross-entropy, loss functions, and ML pipeline
- Convolutional filters, backpropagation, and CNNs
- Gaussian mixture model handout
- Perceptron and SVM

Outline for April 23

- Review temperature, cross-entropy, loss functions, and ML pipeline
- Convolutional filters, backpropagation, and CNNs
- Gaussian mixture model handout
- Perceptron and SVM

Review temperature in language generation

Handout 21, Q1

- Temperature = 1
  - Scores (logits) `[-0.6931472, -0.9162907, -2.3025851]`
  - Probabilities `[0.5, 0.4, 0.1]`
- Temperature = ~~0.01~~ **100 (corrected!)**
  - Scores (logits) `[-0.00693147, -0.00916291, -0.02302585]`
  - Probabilities `[0.33536728108, 0.3346197634, 0.3300129554]`
- Temperature = ~~100~~ **0.01 (corrected!)**
  - Scores (logits) `[ -69.31472 ,  -91.629074, -230.25851 ]`
  - Probabilities `[0.9999999, 2.0370382e-10, 1.26765211e-70]`

Temperature: don’t always pick the letter with maximum probability

```
>>> print(extend_text("To be or not to be", temperature=0.01))
To be or not to be the duke
as it is a proper strange death,
and the
```

```
>>> print(extend_text("To be or not to be", temperature=1))
To be or not to behold?

second push:
gremio, lord all, a sistermen,
```

```
>>> print(extend_text("To be or not to be", temperature=100))
To be or not to bef ,mt'&o3fpadm!$
wh!nse?bws3est--vgerdjw?c-y-ewznq
```

Q: is tf.argmax different from temp=1?

Geron, Chap 16

Core Machine Learning Pipeline

```
def train_step(model, images, labels, loss_func, optimizer):
    with tf.GradientTape() as tape:
1       predictions = model(images)
2       loss = loss_func(labels, predictions)
3   gradients = tape.gradient(loss, model.trainable_variables)
4   optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    return loss, predictions
```

linear regression

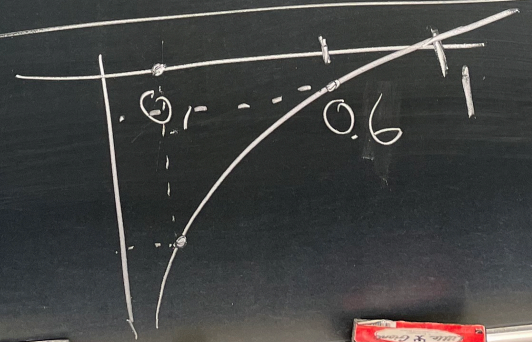
① $\hat{y} = \vec{w} \cdot \vec{x}$ features

squared error (MSE)

② $J(\vec{w}) = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(\vec{w} \cdot \vec{x} - y)^2$

③ $\nabla J(\vec{w}) = (\vec{w} \cdot \vec{x} - y)\vec{x}$

④ $\vec{w} \leftarrow \vec{w} - \alpha \nabla J(\vec{w})$



y=1 dog

y=2 cat

y=3 bird

cat $\Rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

Stochastic Gradient Descent (high-level)

set **w = 0** vector

while **cost J(w)** still changing (or max iter reached):

    shuffle data points

    for i = 1…n:

        w <- **w – alpha(derivative of J(w) wrt x_i)**

    store J(w)

Gradient descent variations

- **Batch gradient descent**
  - Go over *all* training data before making a weight update

- **Stochastic gradient descent**
  - Shuffle data and make a weight update after *each training example*

- **Mini-batch gradient descent**
  - Update after a "*mini-batch*" of training examples (i.e. 50)
  - Vocab word: **epoch** (one pass through the data)

3 important pieces to SGD

(with logistic regression concrete example)

- Hypothesis function (prediction)

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = p(y = 1|\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$$

- Cost/loss function (want to minimize)

$$J(\boldsymbol{w}) = -\sum_{i=1}^{n} y_i \log h_{\boldsymbol{w}}(\boldsymbol{x_i}) + (1 - y_i) \log(1 - h_{\boldsymbol{w}}(\boldsymbol{x_i}))$$

- Gradient of cost wrt single data point x_i

$$\nabla J_{\boldsymbol{x}_i}(\boldsymbol{w}) = (h_{\boldsymbol{w}}(\boldsymbol{x_i}) - y_i)\boldsymbol{x_i}$$

binary cross entropy (2 classes)

$H(y, \hat{y}) = -y \log \hat{y} - (1-y) \log_2 (1-\hat{y})$

$y \in \{0, 1\}$

$\hat{y} \in [0, 1]$ "prob pos"

---

categorical cross entropy (K classes) ci-far $K = 10$

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

$-1 \cdot \log_2(0.6)$

$\vec{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ "hot"

$\hat{\vec{y}} = \begin{bmatrix} 0.2 \\ 0.6 \\ 0.2 \end{bmatrix}$

$\hat{\vec{y}} = \begin{bmatrix} 0.8 \\ 0.1 \\ 0.1 \end{bmatrix}$

Sparse categorical cross entropy

$y = 2$ example

$y \in \{1, 2, \cdots K\}$ general

$-\log \hat{y}_k$ if $y = k$

Outline for April 23

- Review temperature, cross-entropy, loss functions, and ML pipeline
- Convolutional filters, backpropagation, and CNNs
- Gaussian mixture model handout
- Perceptron and SVM

Handout 25, page 1 and 2!

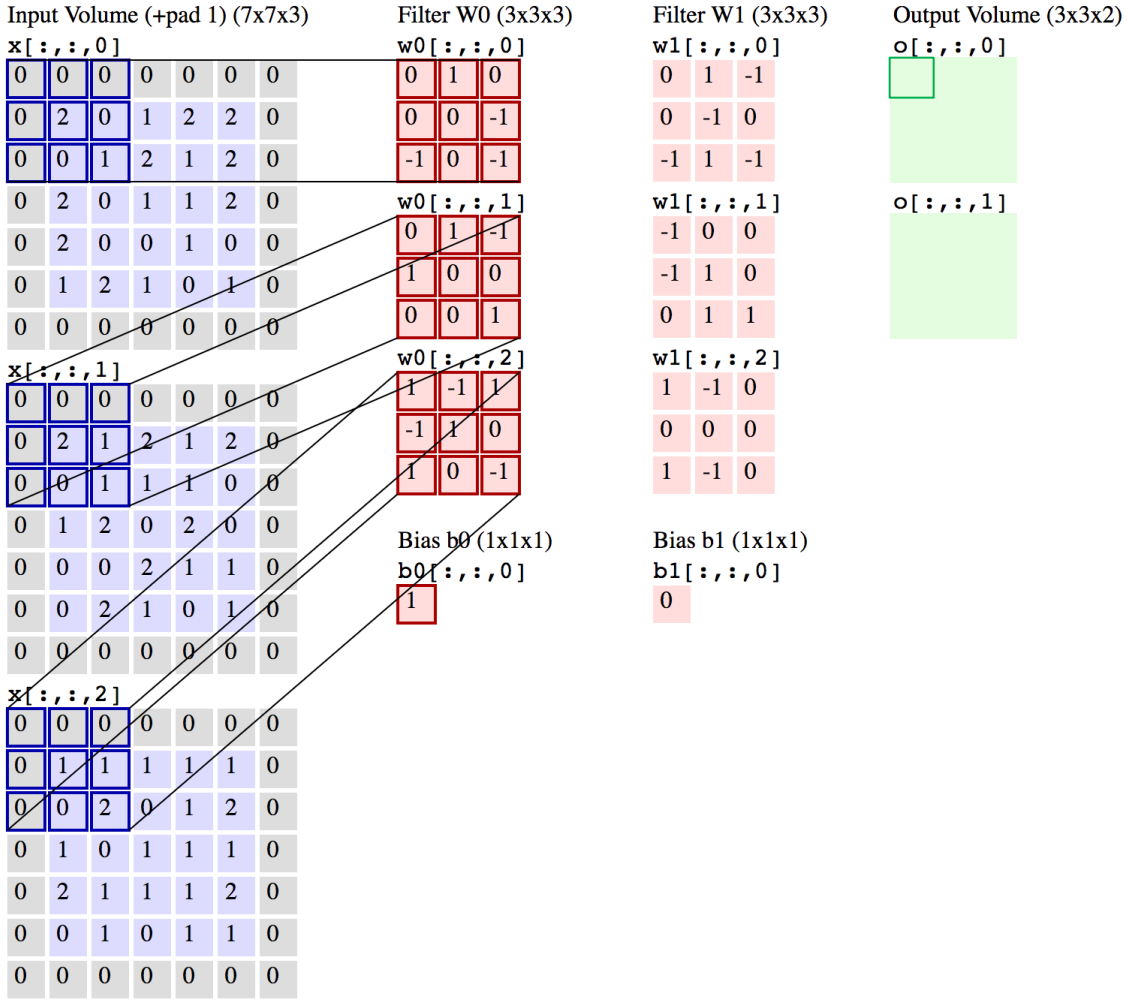


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

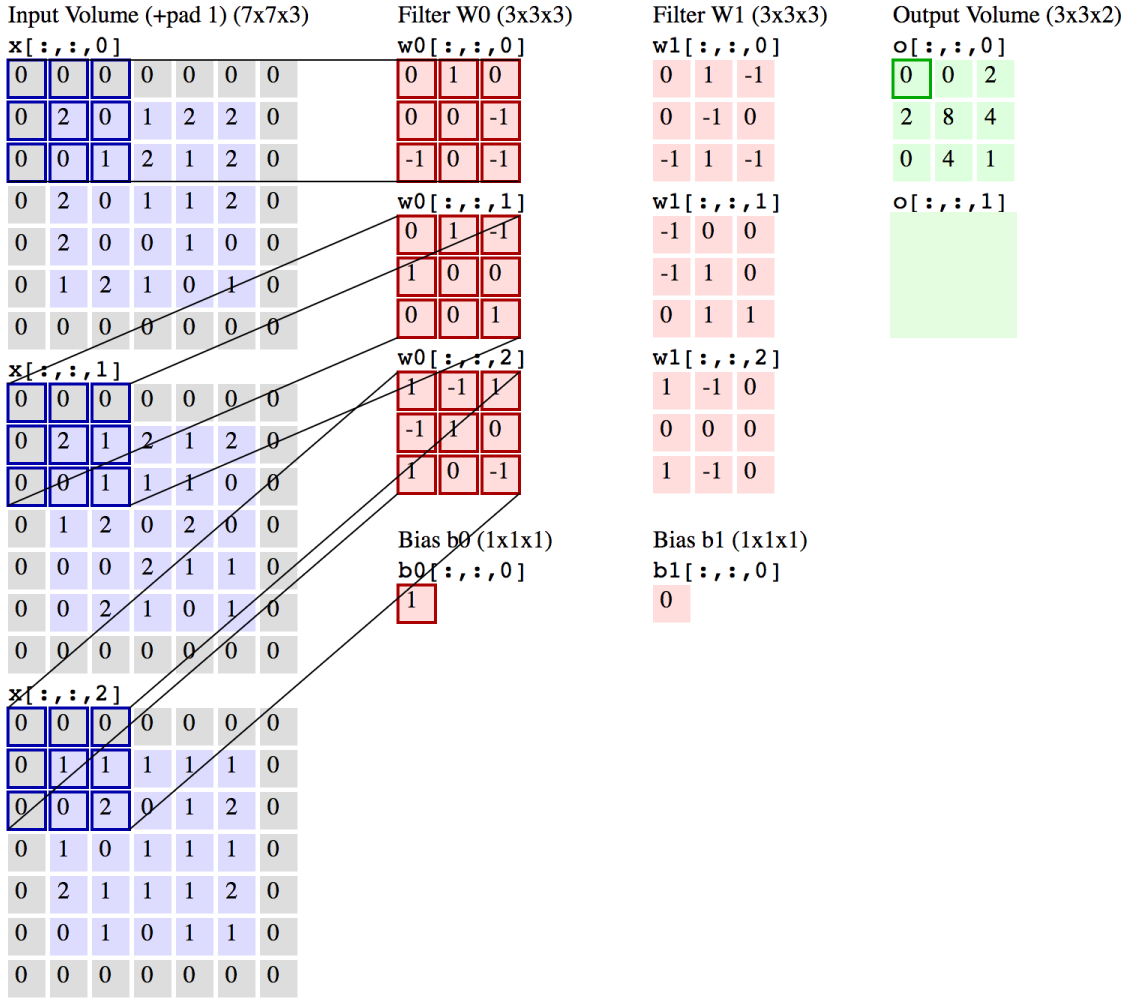


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

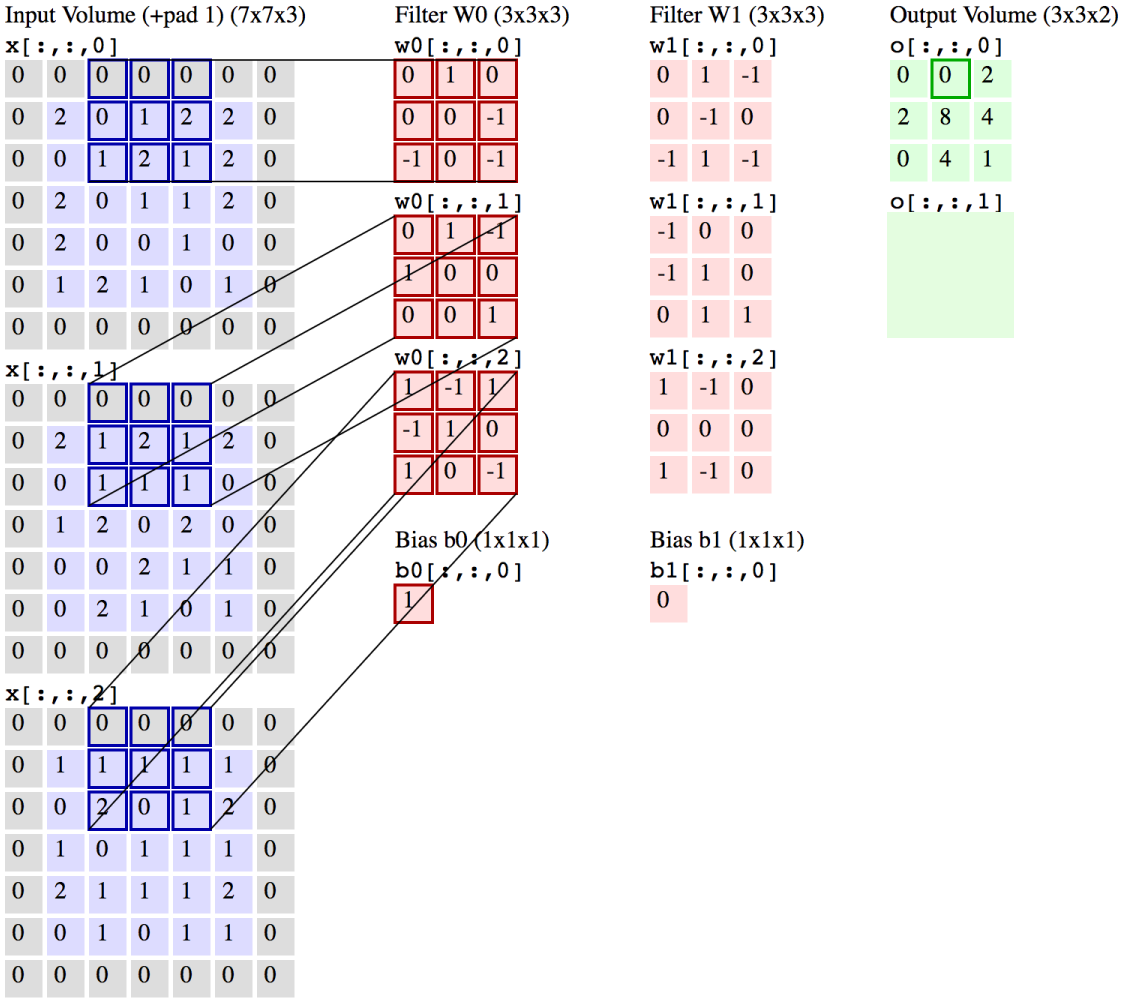


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

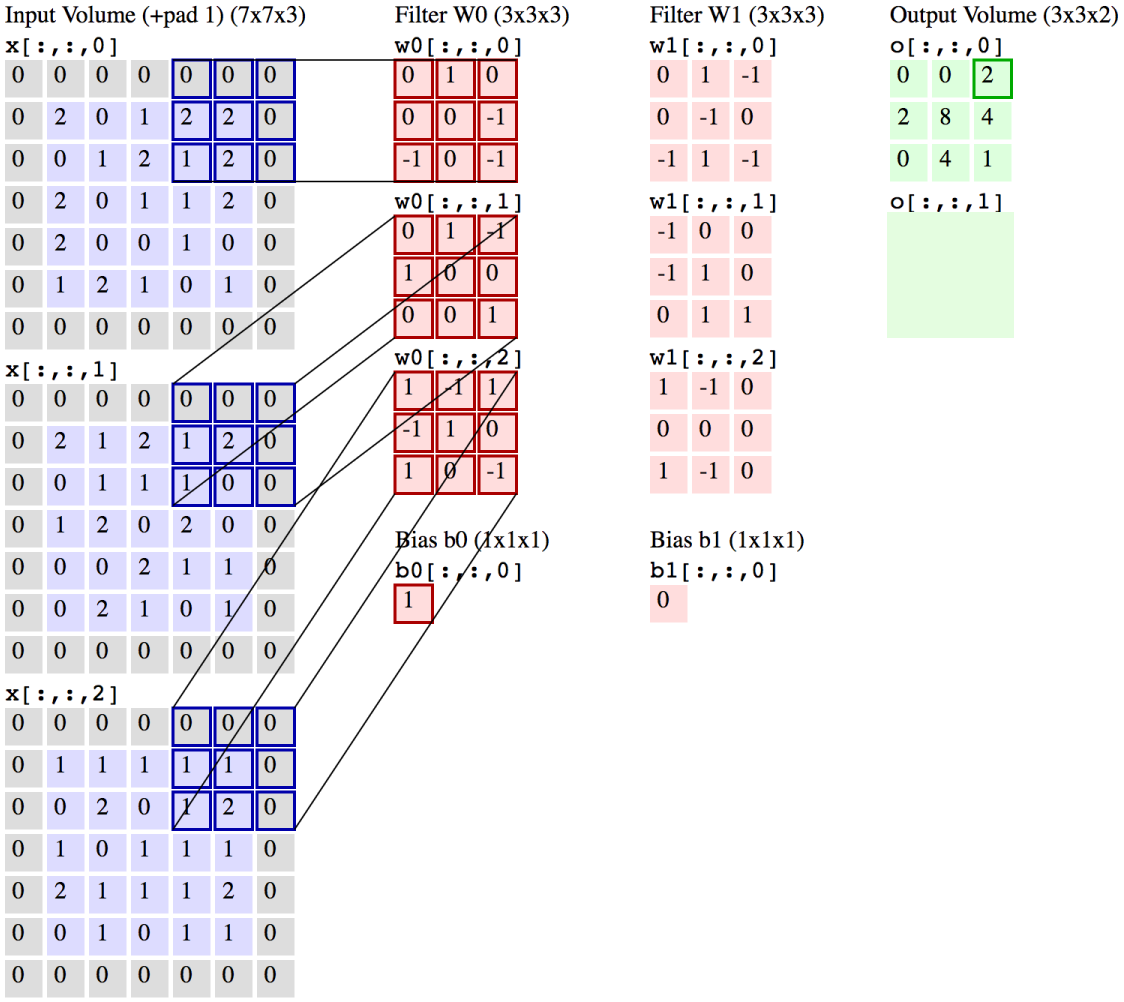


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

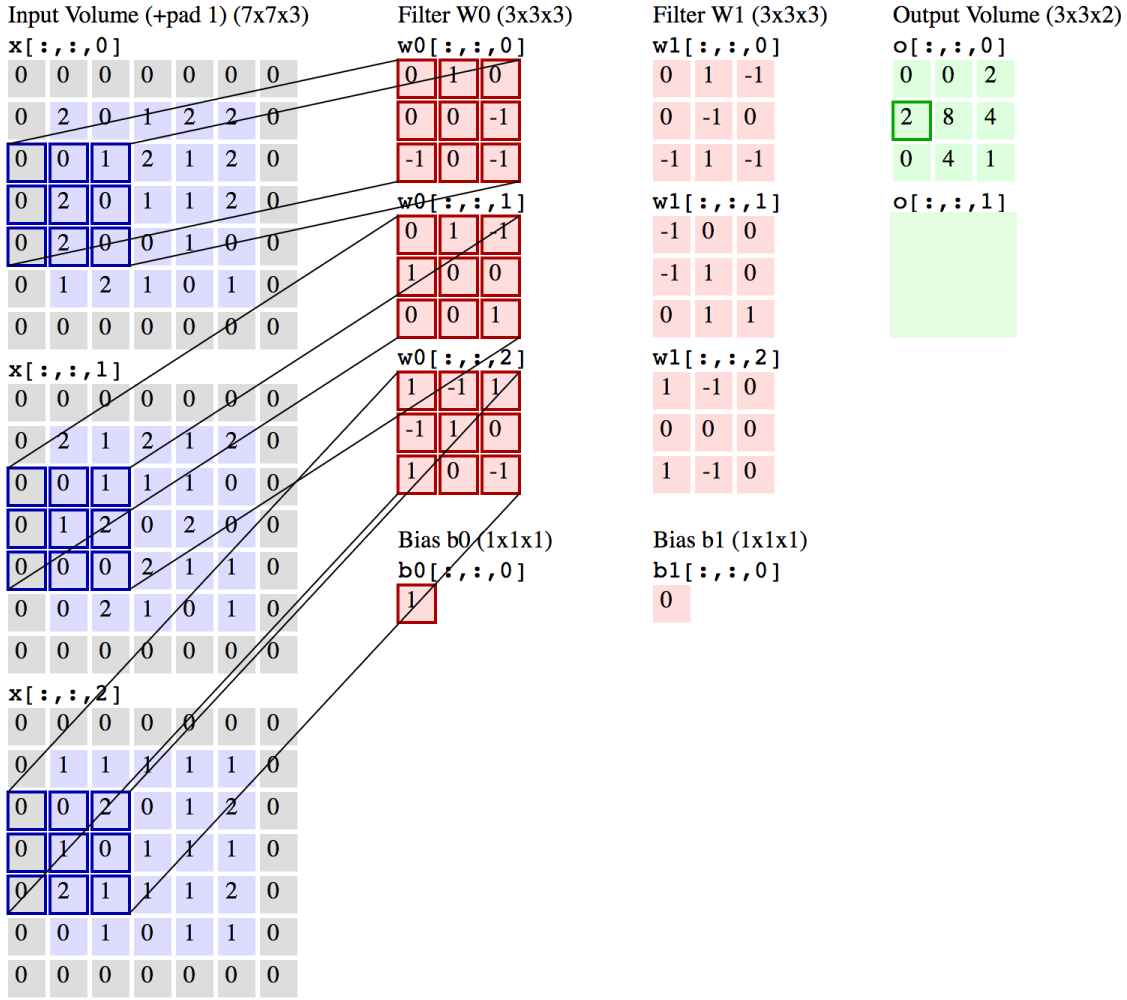


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

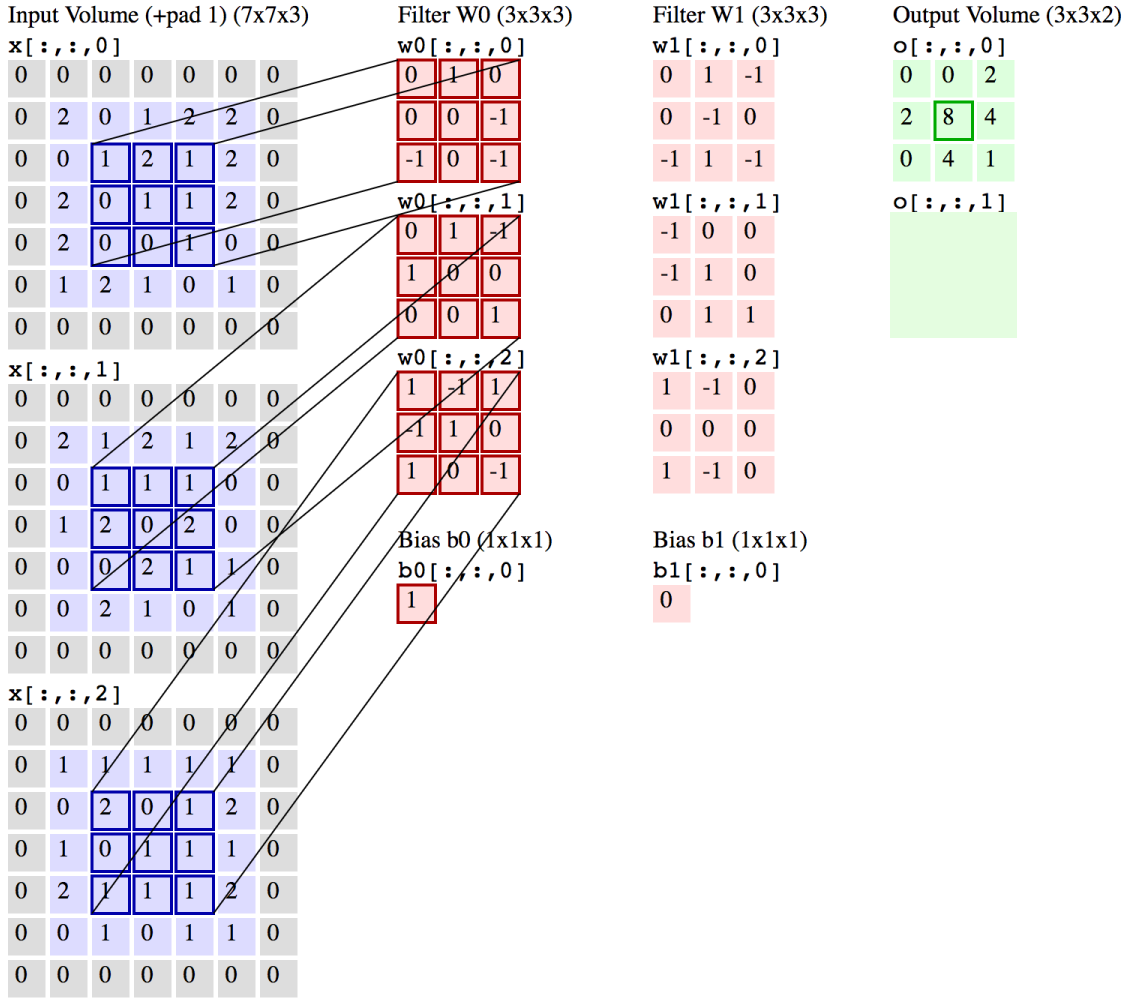


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

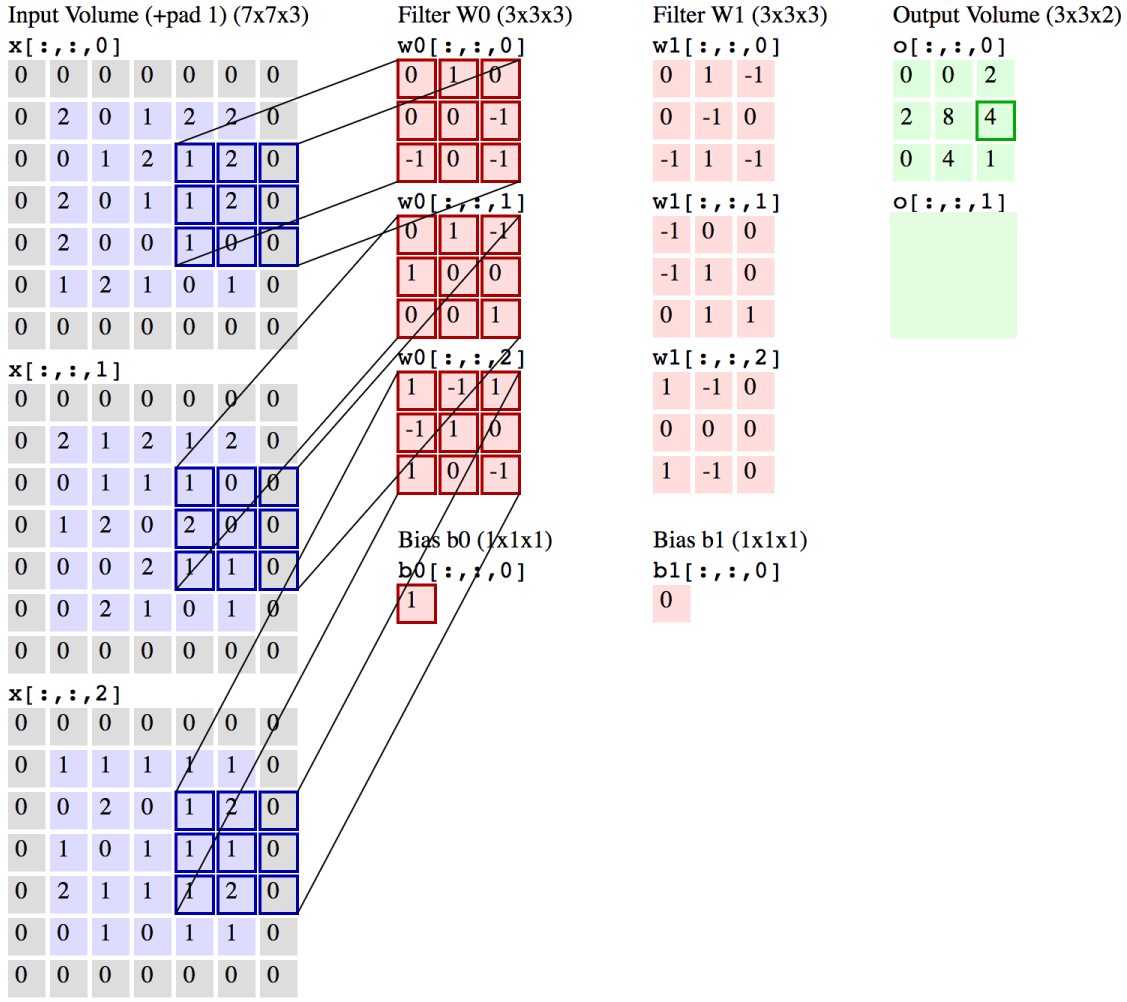


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

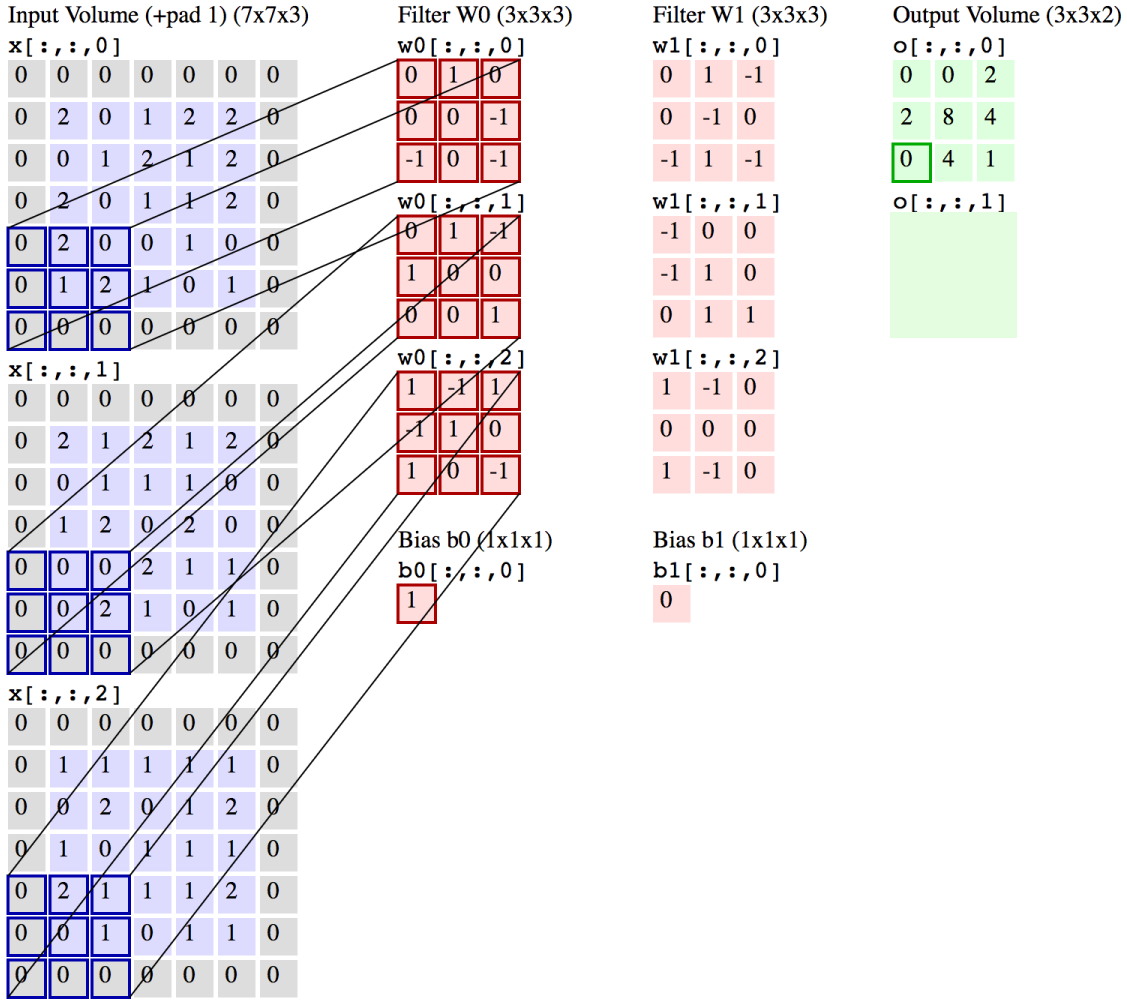


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

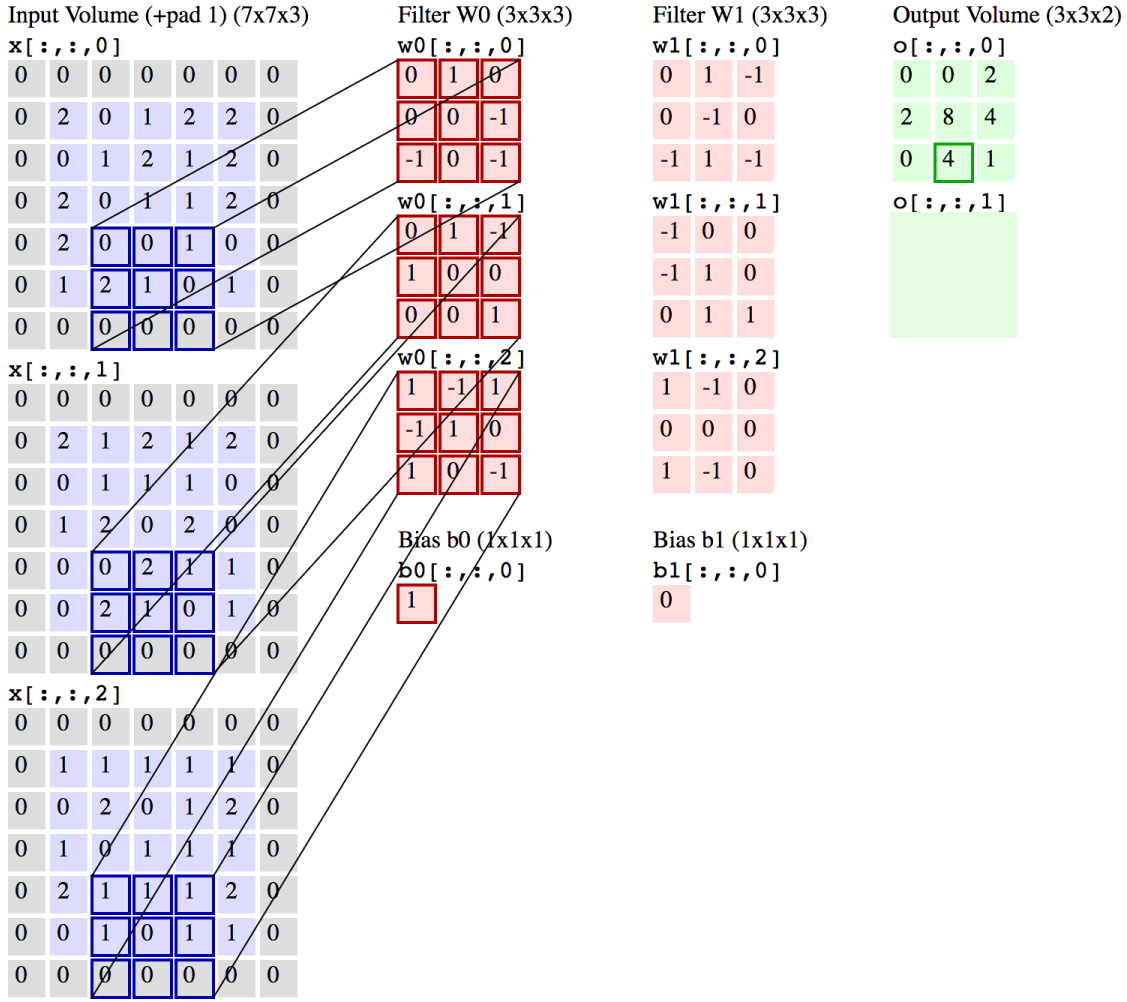


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

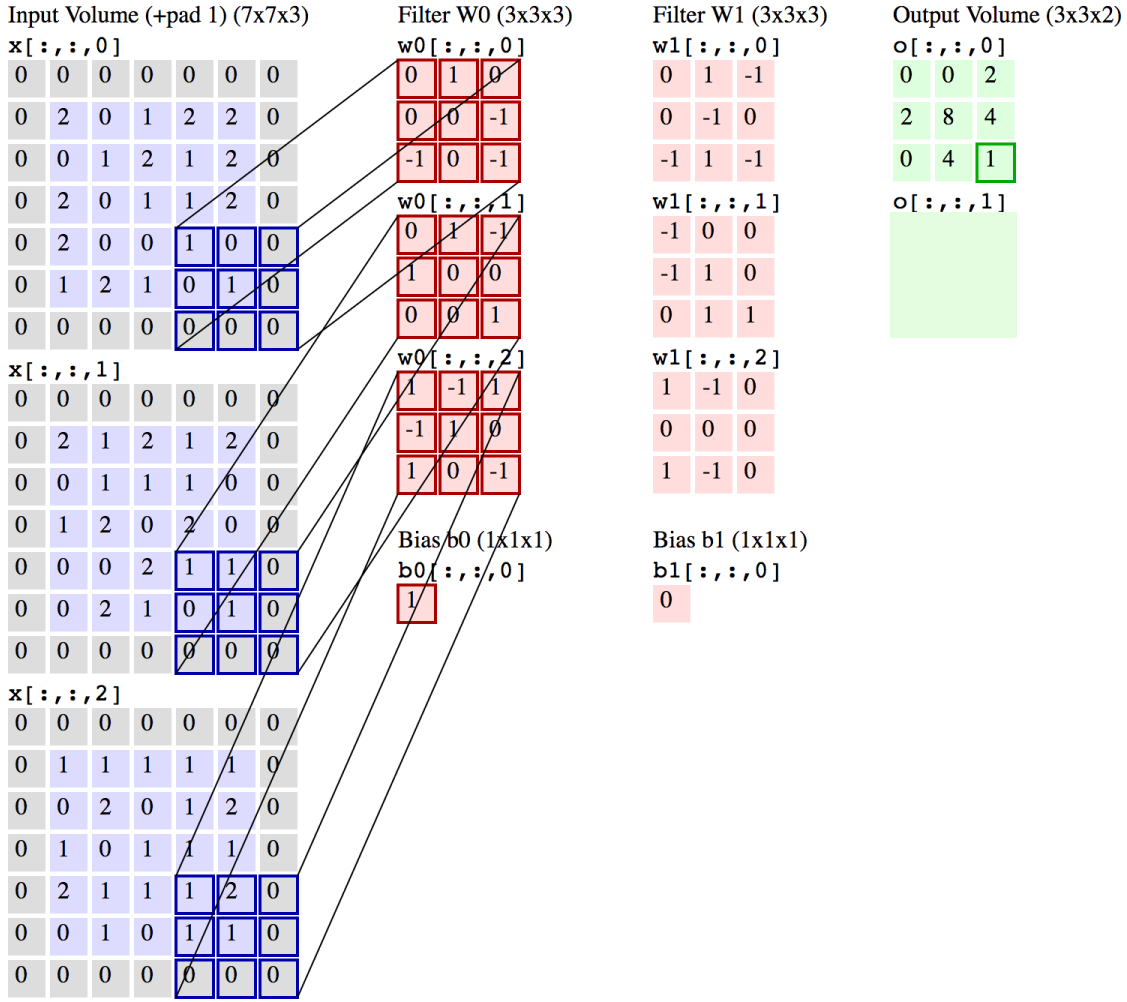


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

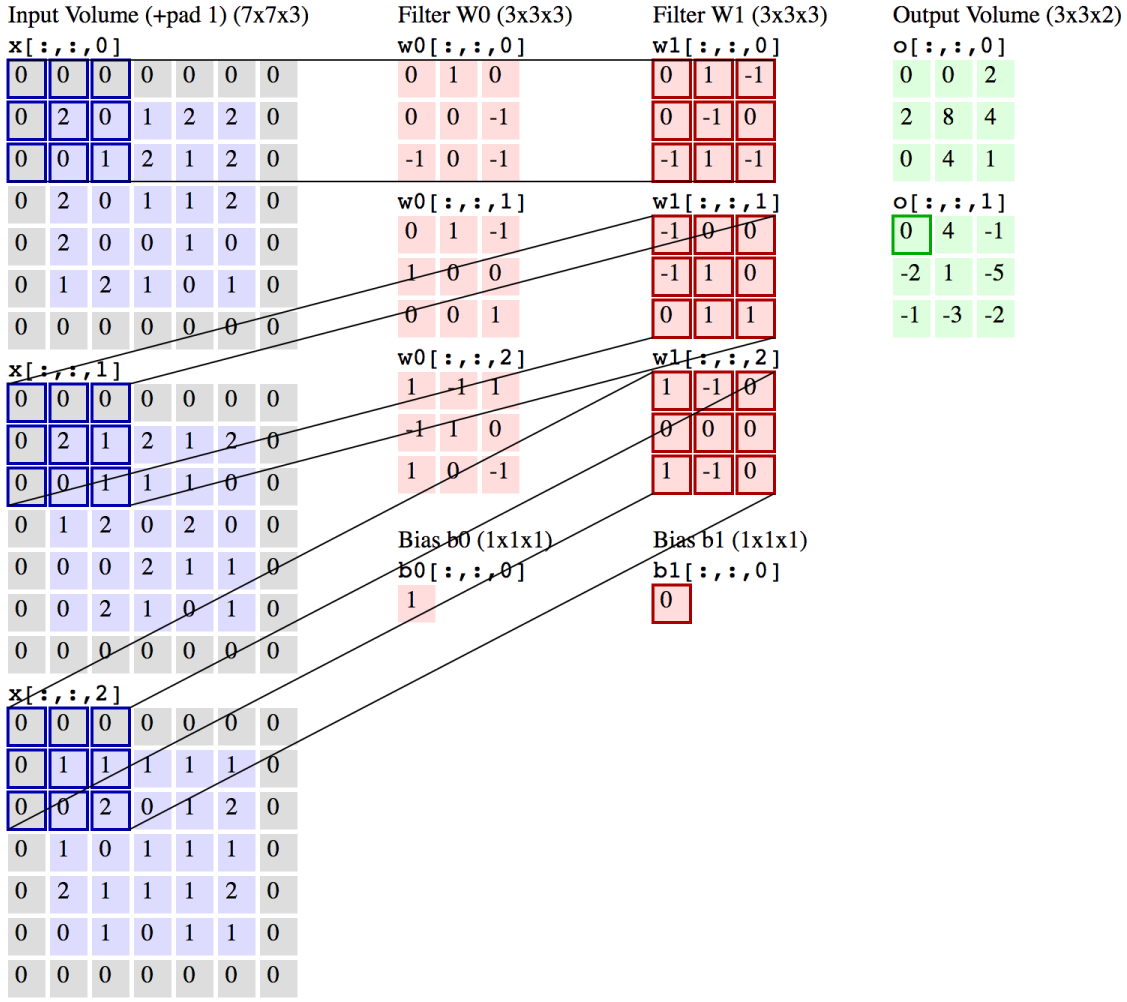


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

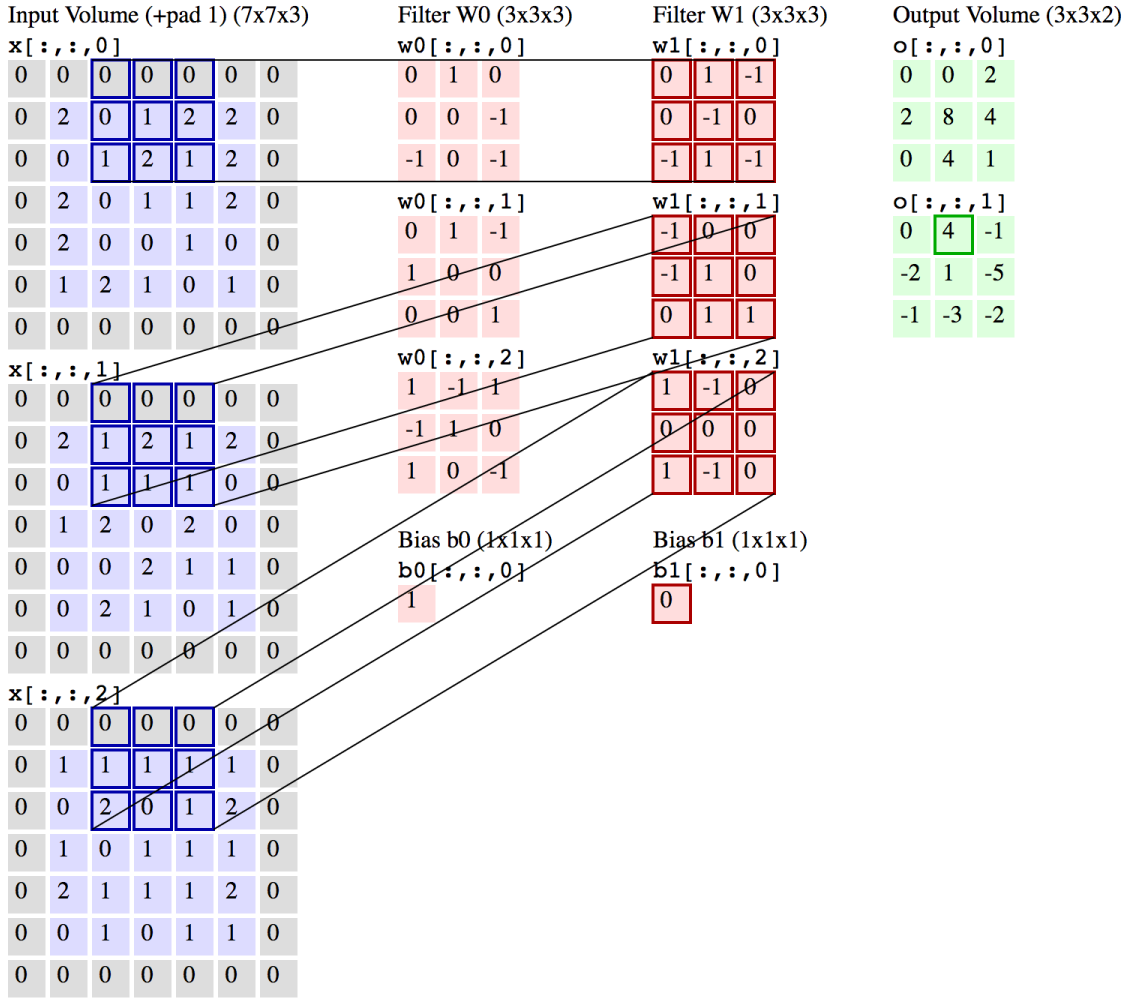


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

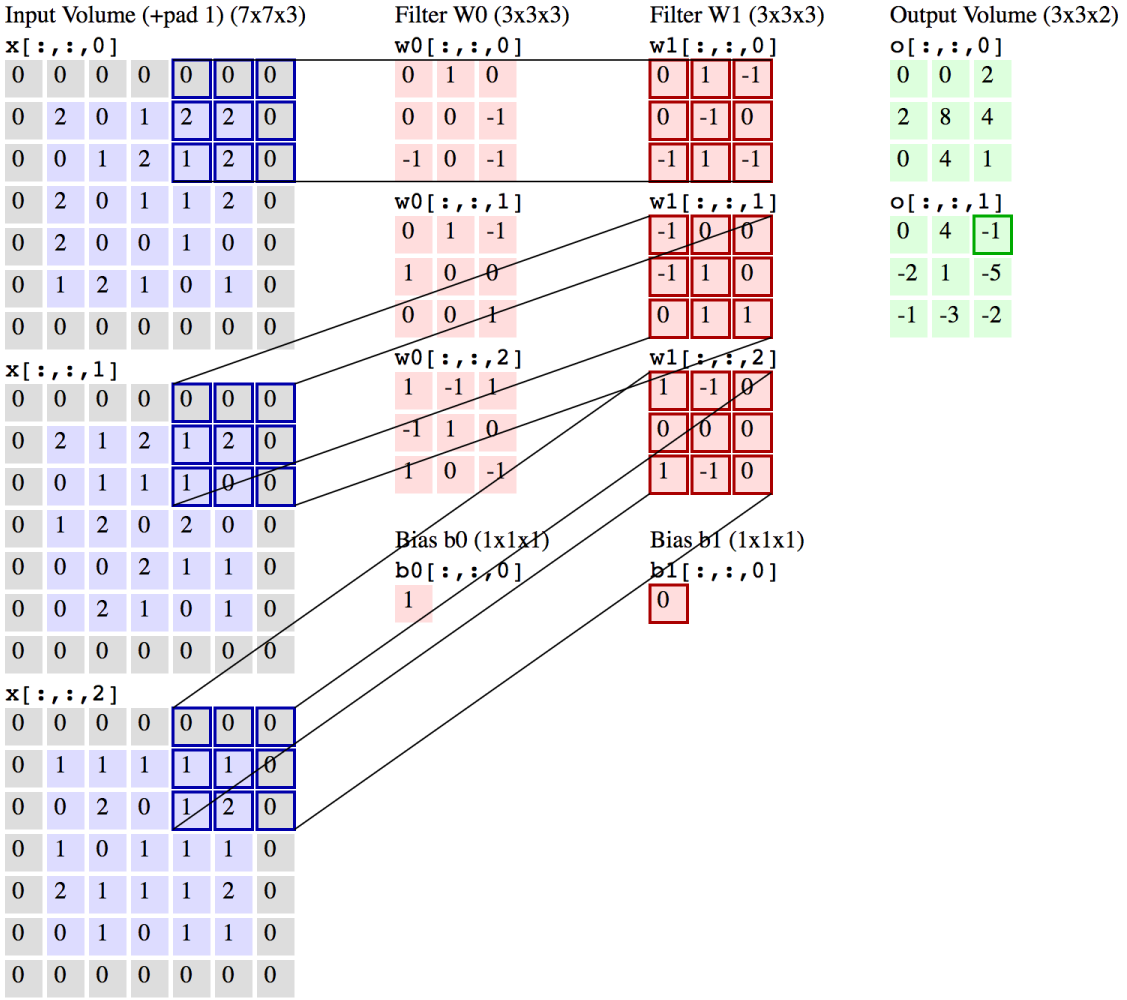


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

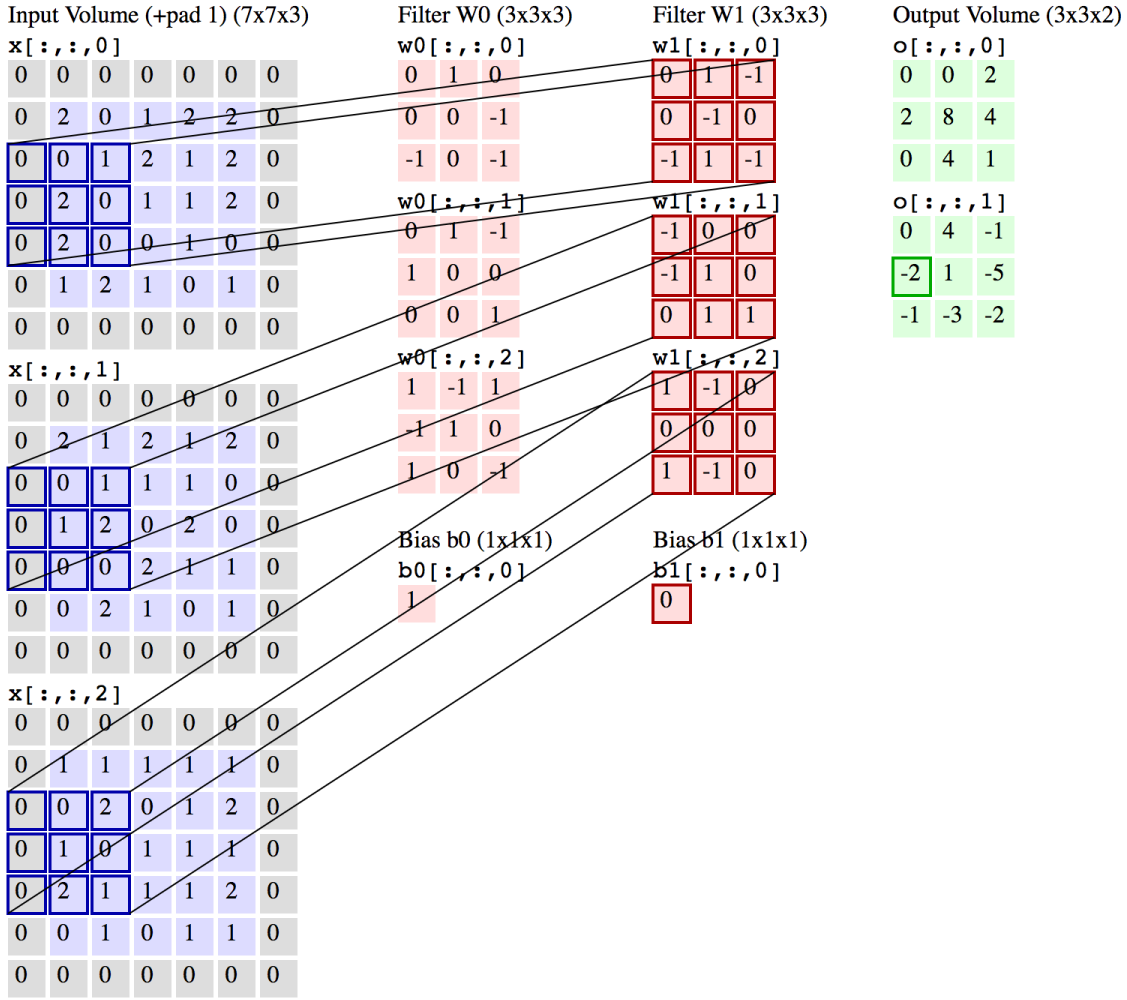


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

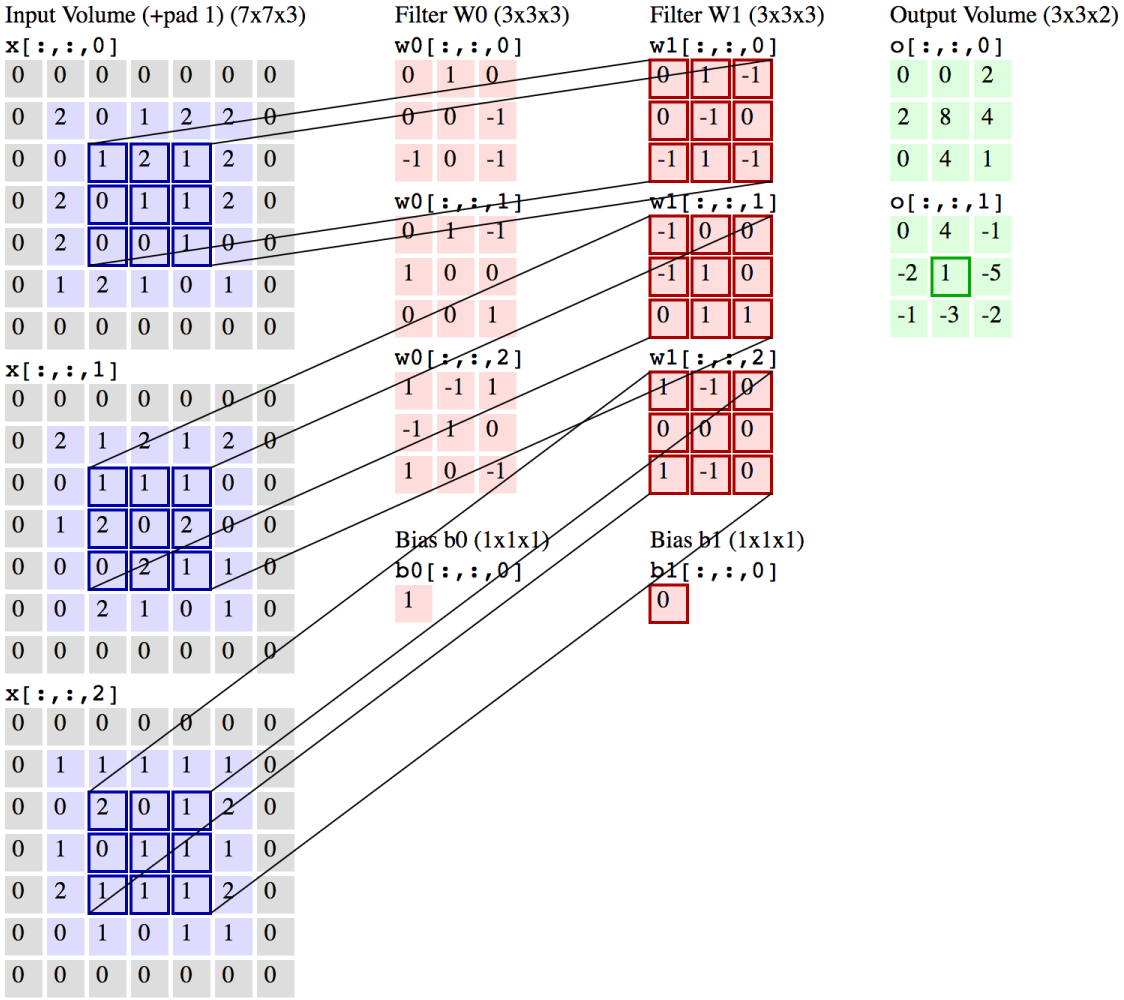


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

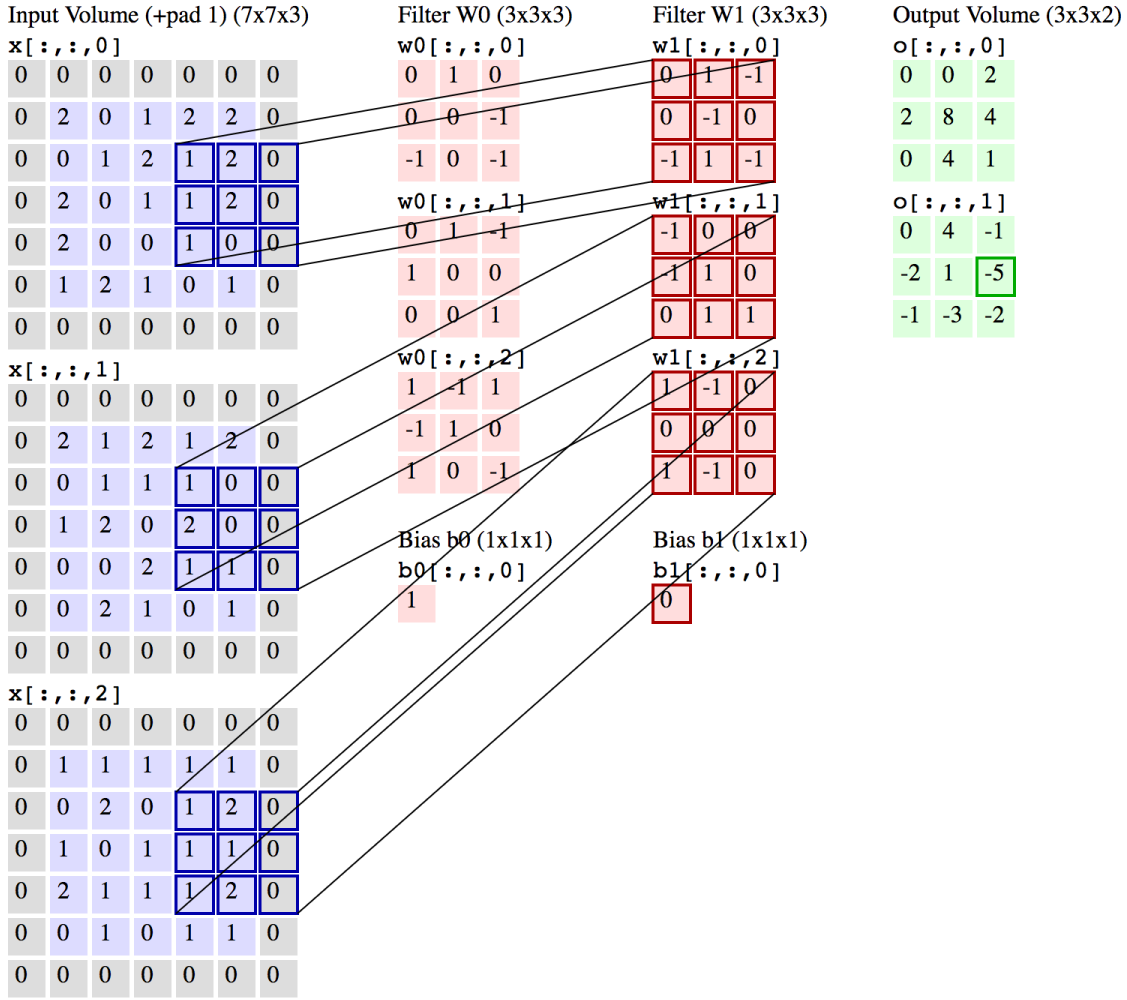


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

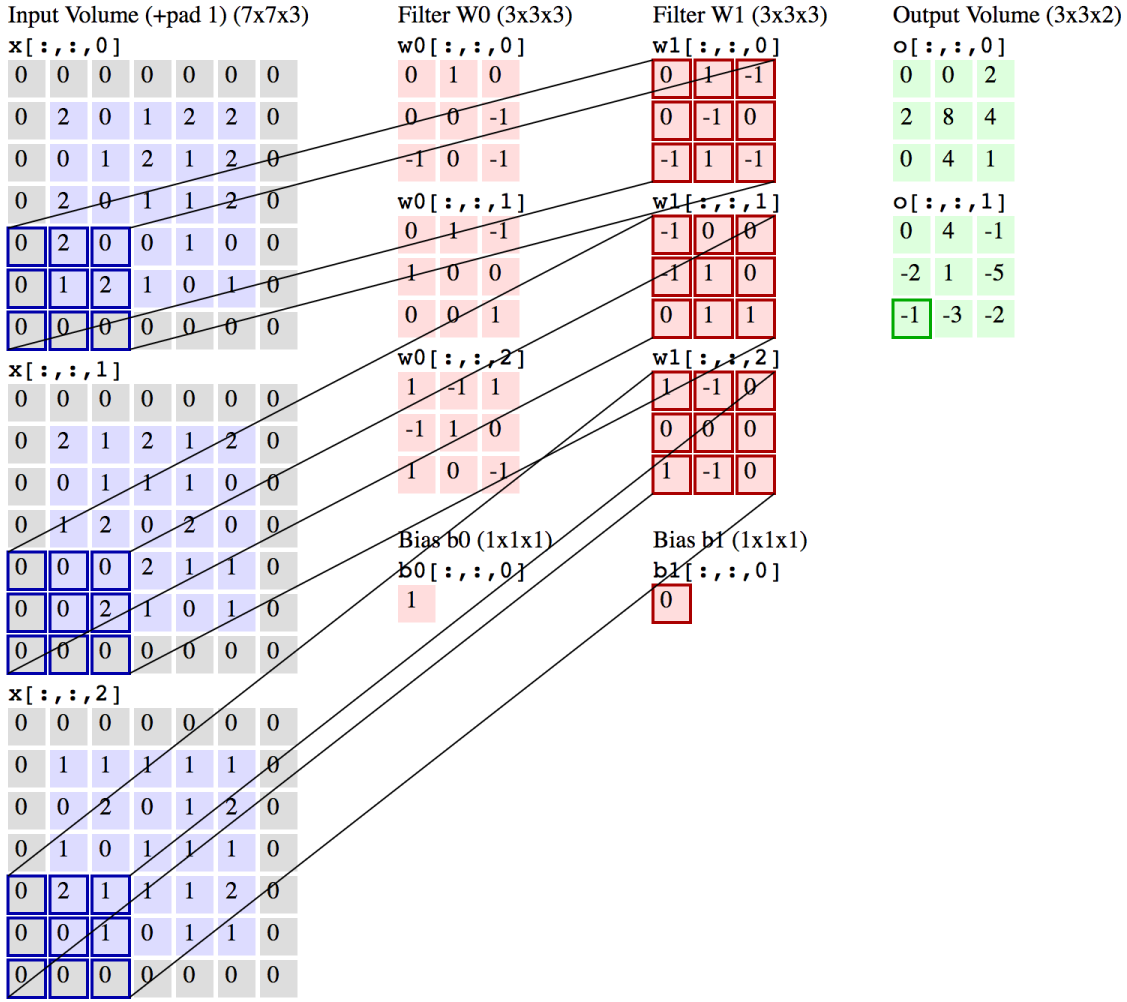


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

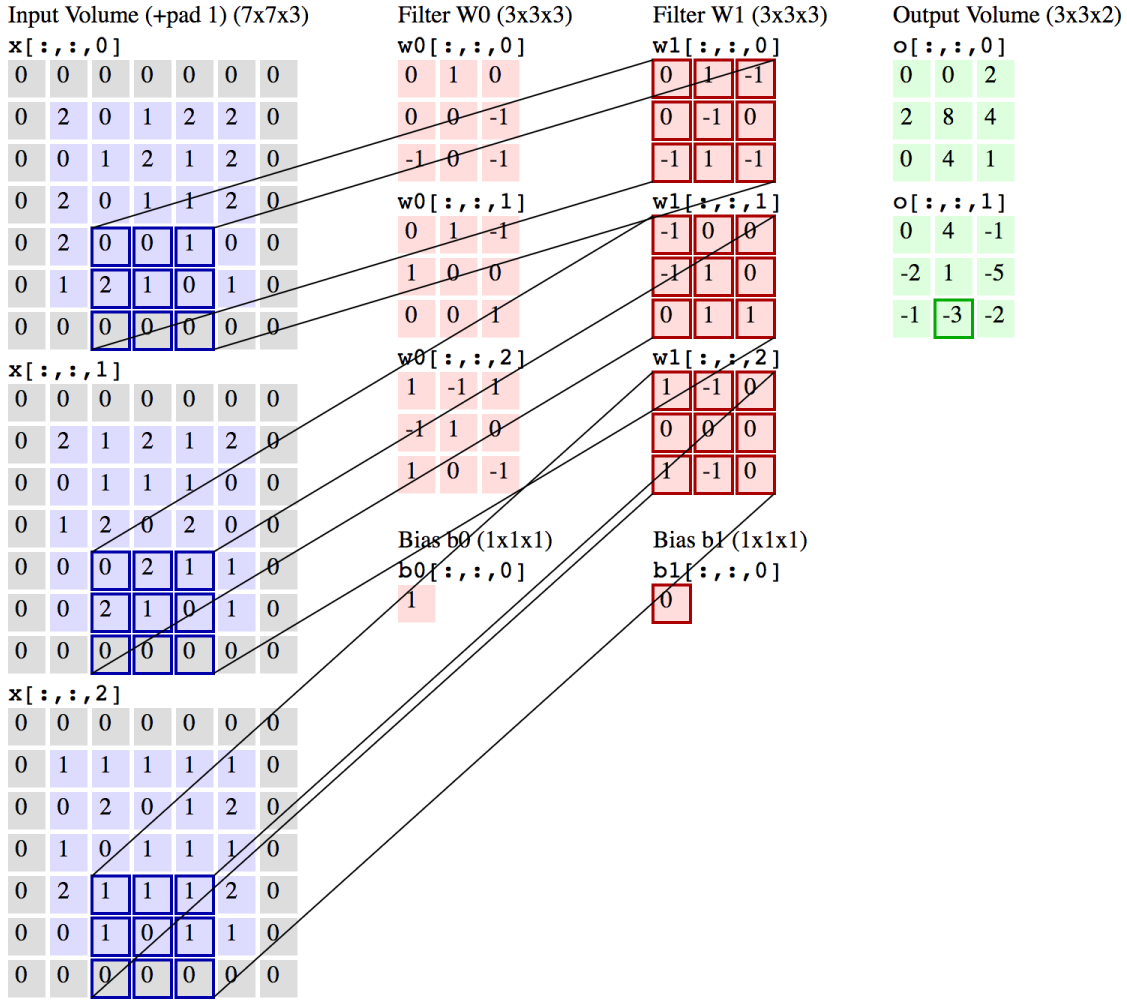


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

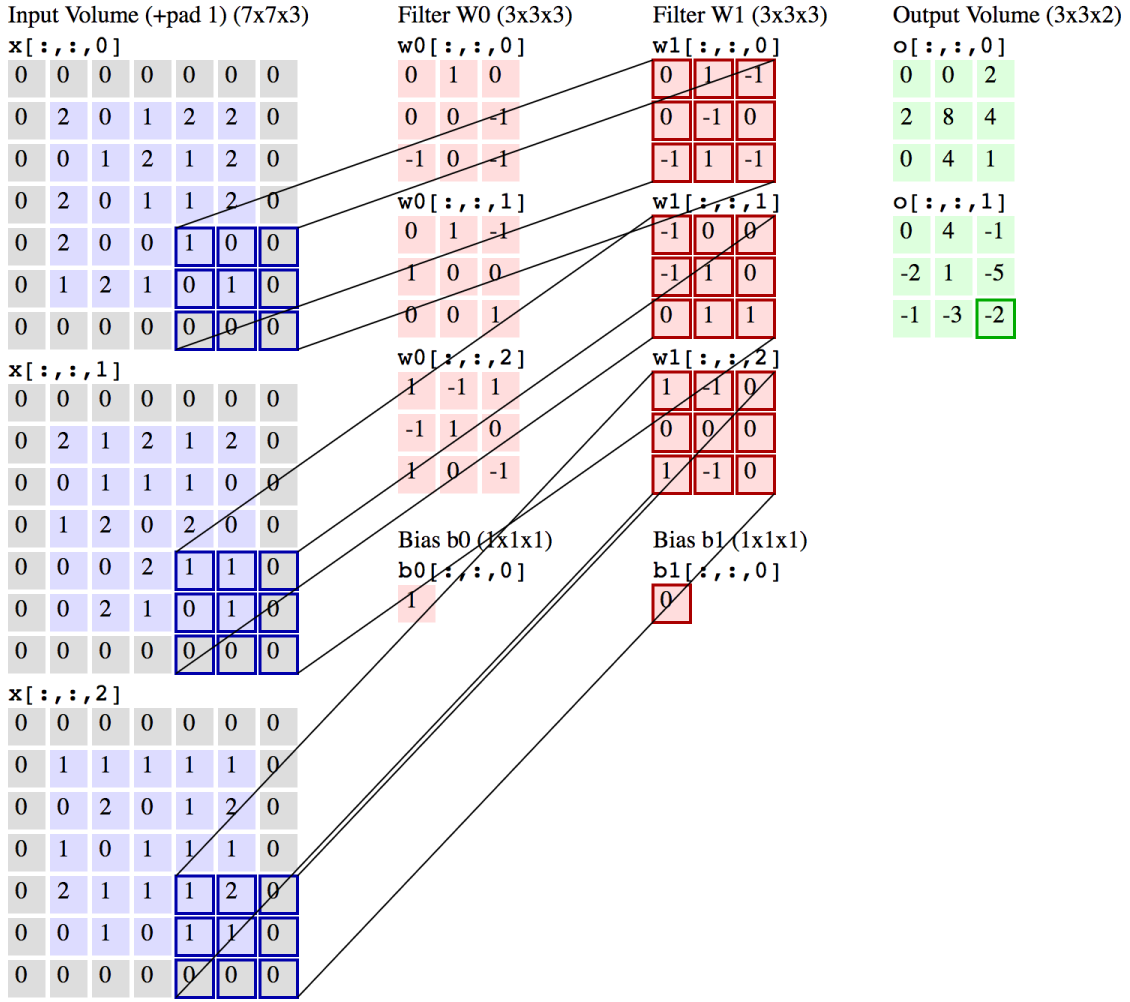
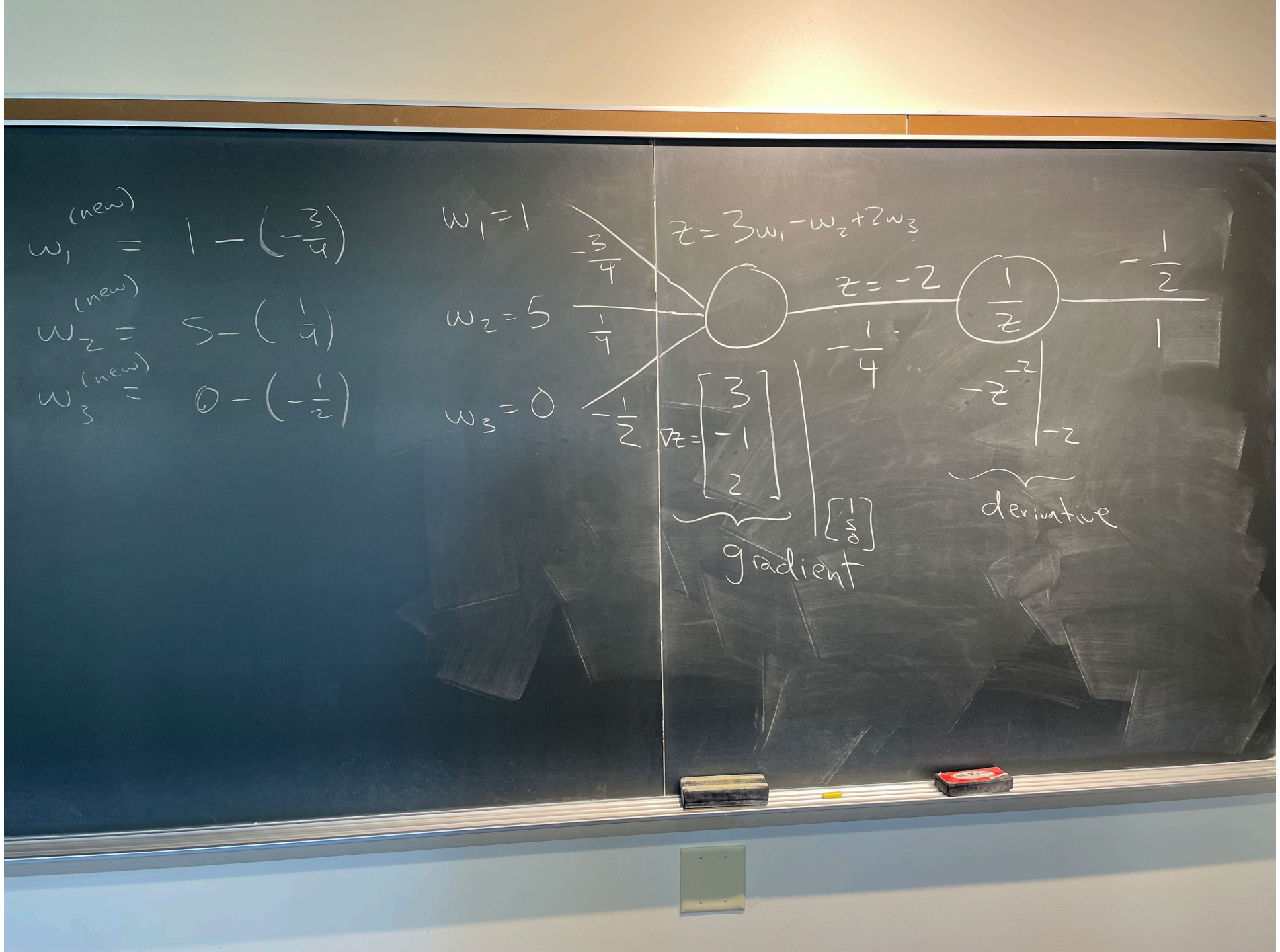
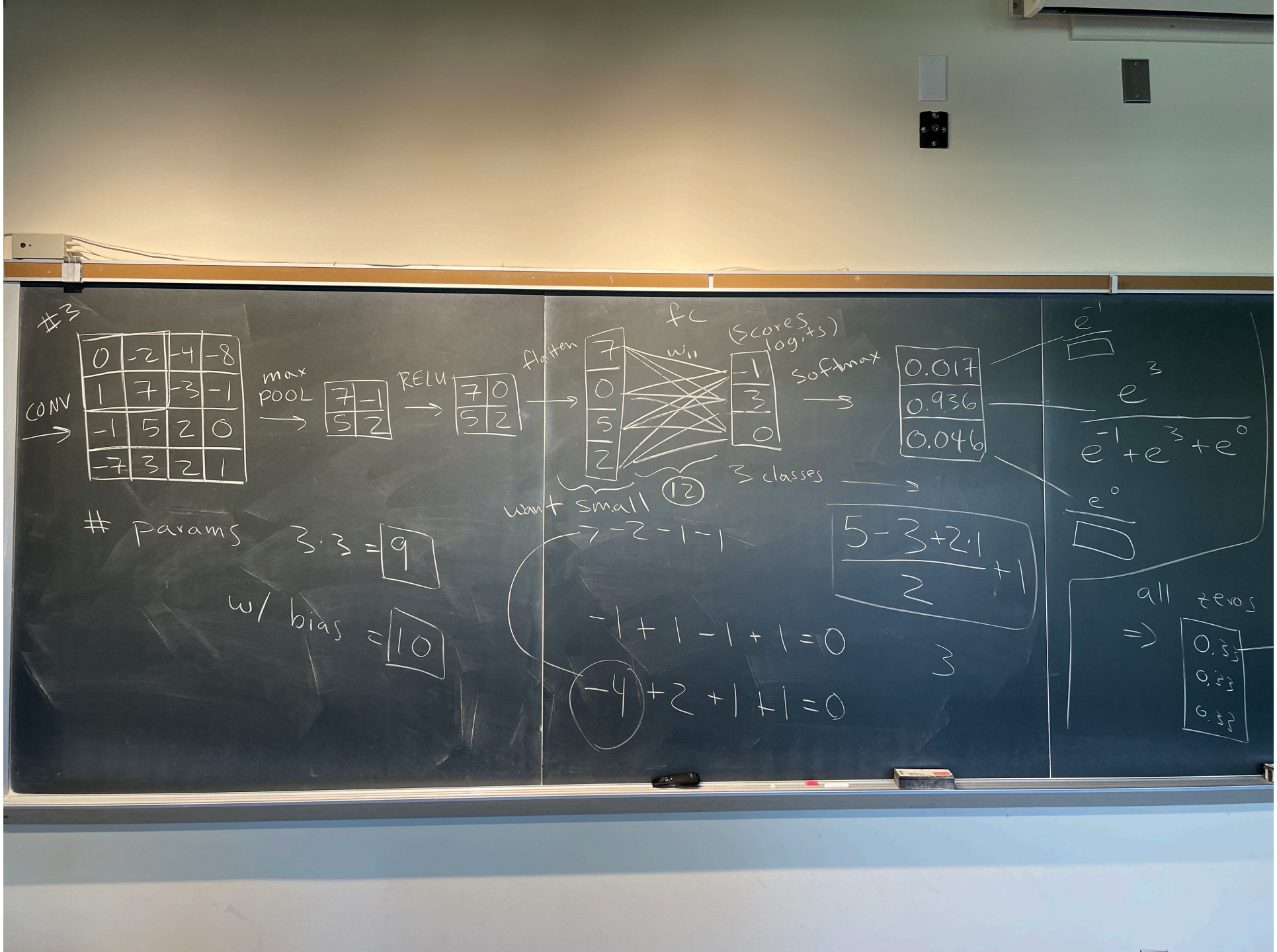


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/


$w_1^{(new)} = 1 - \left(-\frac{3}{4}\right)$
$w_2^{(new)} = 5 - \left(\frac{1}{4}\right)$
$w_3^{(new)} = 0 - \left(-\frac{1}{2}\right)$
$w_1 = 1$
$w_2 = 5$
$w_3 = 0$
$-\frac{3}{4}$
$\frac{1}{4}$
$-\frac{1}{2}$
$z = 3w_1 - w_2 + 2w_3$
$z = -2$
$\frac{1}{z}$
$-\frac{1}{2}$
1
$-\frac{1}{4}$
$-z^{-2}$
$-z$
$\nabla z = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}$
$\begin{bmatrix} \frac{1}{5} \\ 0 \end{bmatrix}$
gradient
derivative


#3
CONV
max POOL
RELU
flatten
fc
w11
(scores, logits)
softmax
0.017
0.936
0.046
3 classes
want small
12
params 3·3 = 9
w/ bias = 10
-2-1-1
-1+1-1+1=0
-4+2+1+1=0
3
5-3+2·1/2+1
all zeros

Output size formula in one dimension

- W = input width (same for height though, but not for depth!)
- F = filter size
- P = padding (on one side)
- S = strides

formula for output size:

$$\frac{W - F + 2P}{S} + 1$$

Handout 18

(a) Which steps require parameter learning? (out of CONV, RELU, POOL, FLATTEN, FC)

CONV, FC

(b) First layer params 5*5*3*20 + 20 = 1520

(c) Second layer params 3*3*20*10 + 10 = 1810

(d) Third layer params 8*8*10*10 + 10 = 6410

(e) Total # params 9740

If we had a FC with p_1=100 and p_2=50, we would have 312,860 params to learn (check this after class). CNN is much better!

Outline for April 23

- Review temperature, cross-entropy, loss functions, and ML pipeline
- Convolutional filters, backpropagation, and CNNs
- Gaussian mixture model handout

Handout 24

- Perceptron and SVM

$$W = \begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.7 & 0.1 & 0.2 \\ 0.1 & 0.4 & 0.5 \\ 0.8 & 0.1 & 0.1 \end{bmatrix} \quad \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix}$$

Cluster 1 | Cluster 2 | Cluster 3

$n = 4$
$K = 3$

$$M_k = \sum_{i=1}^{n} w_{ik}$$

$$M_1 = 1.8$$

$$\Rightarrow \quad \pi_1 = \frac{1.8}{4}$$

$$\pi_2 = \frac{1.1}{4}$$

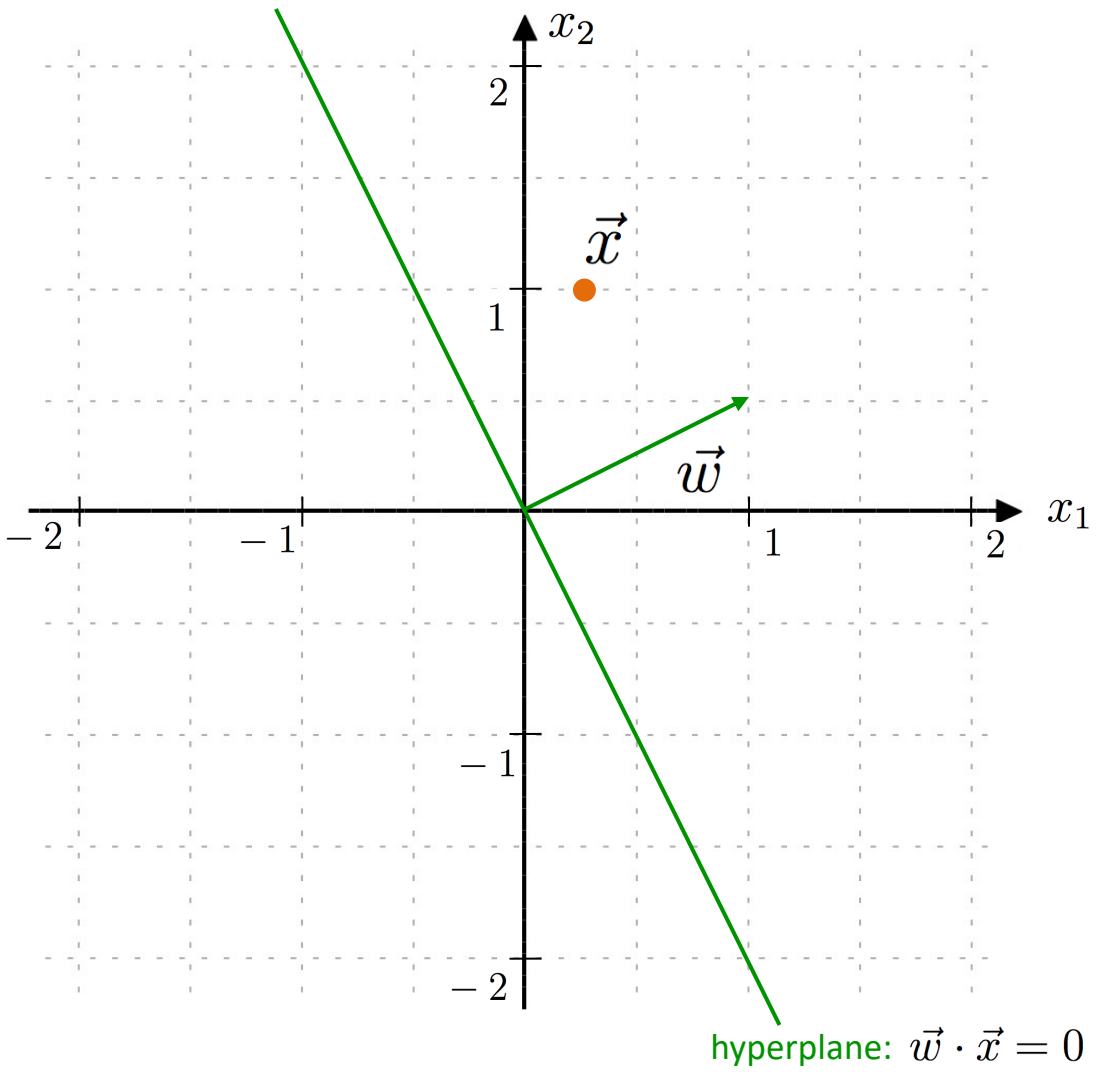
$$\pi_3 = \frac{1.1}{4}$$

Outline for April 23

- Review temperature, cross-entropy, loss functions, and ML pipeline
- Convolutional filters, backpropagation, and CNNs
- Gaussian mixture model handout
- Perceptron and SVM

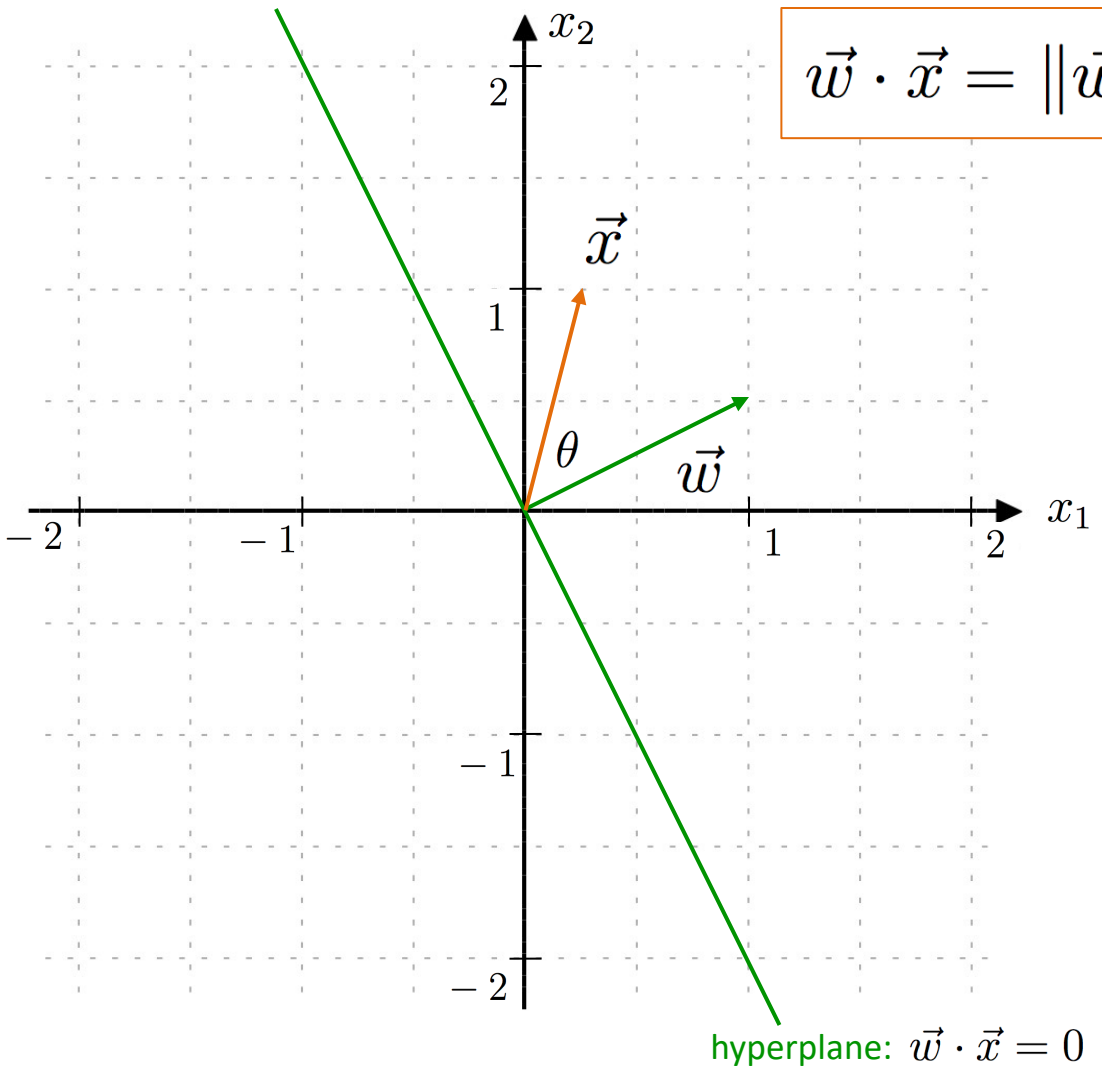
Handout 25, page 3!

Intuition behind the dot product

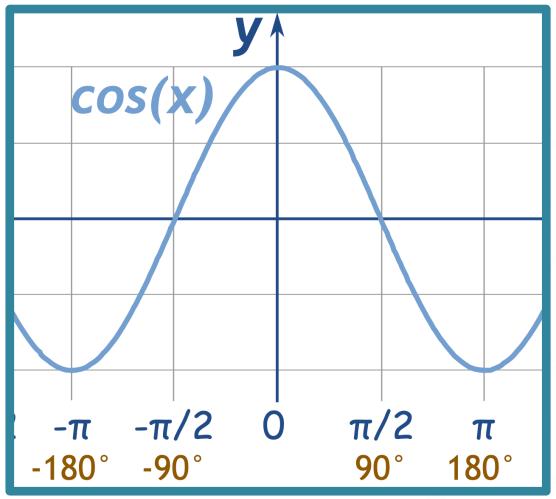


Intuition behind the dot product

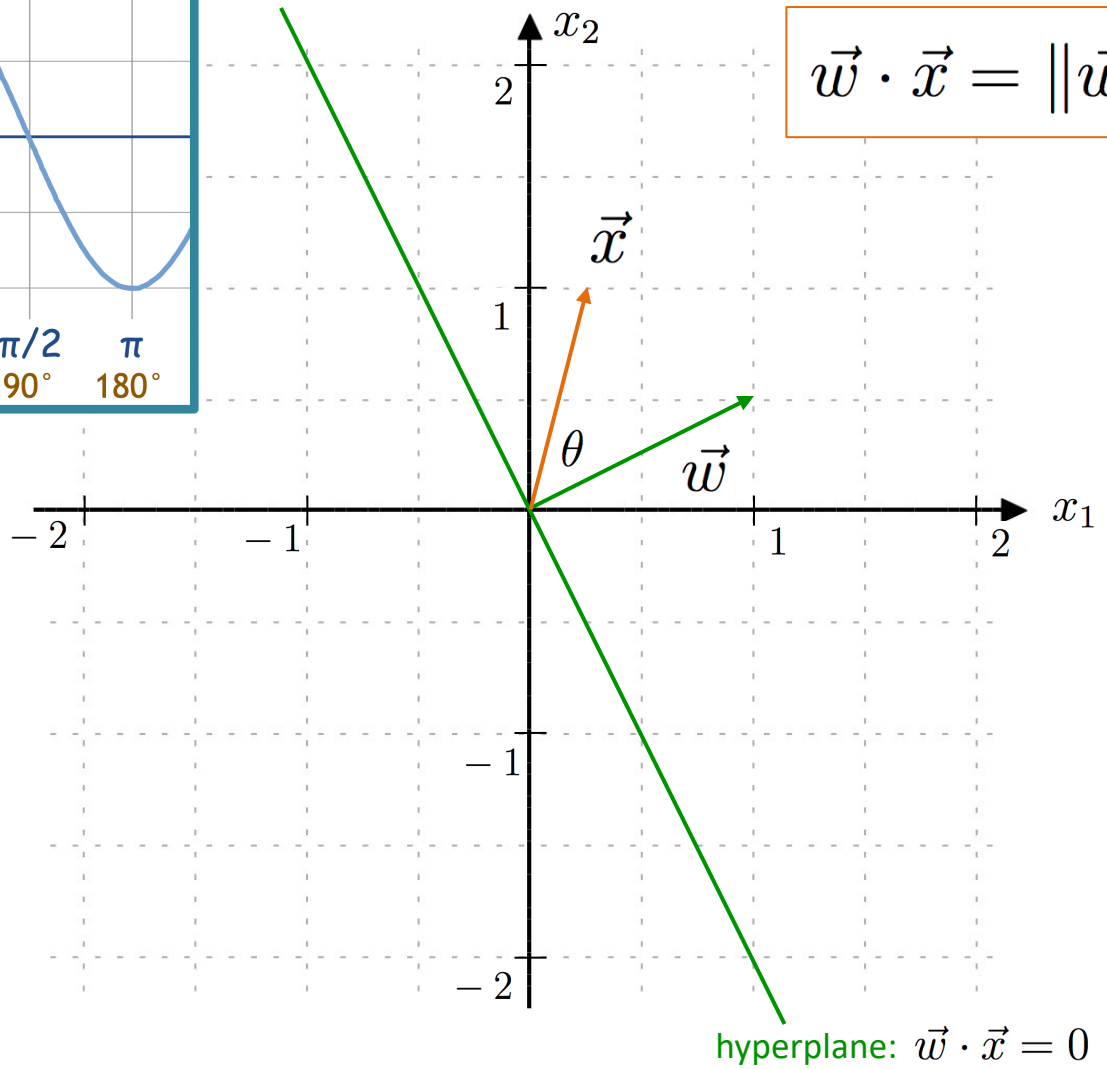
$$\vec{w} \cdot \vec{x} = \|\vec{w}\| \|\vec{x}\| \cos\theta$$



Intuition behind the dot product



$$\vec{w} \cdot \vec{x} = \|\vec{w}\| \|\vec{x}\| \cos\theta$$



Intuition behind the dot product

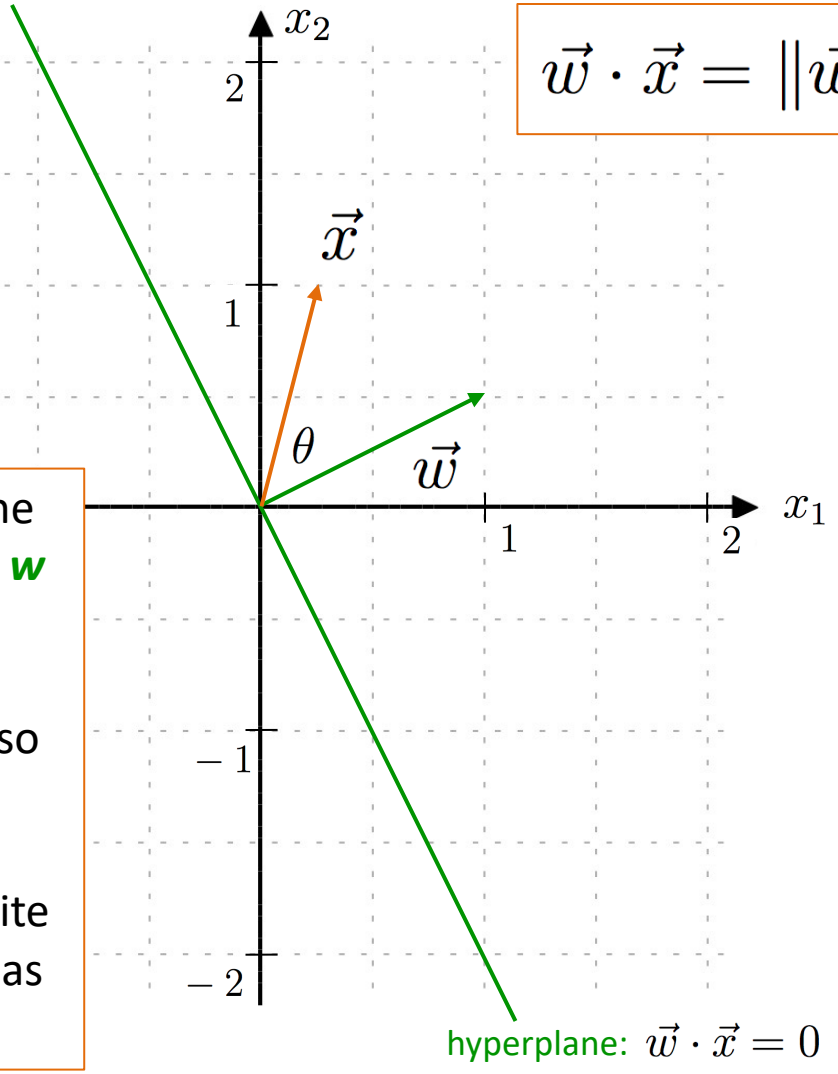
Takeaway: we only care about the sign of the angle between ***x*** and ***w***

- If cos θ > 0, ***x*** is on the same side of the hyperplane as ***w***, so we classify it as positive
- If cos θ < 0, ***x*** is on the opposite side from ***w***, so we classify it as negative

Takeaway: we only care about the sign of the angle between ***x*** and ***w***

- If cos θ > 0, ***x*** is on the same side of the hyperplane as ***w***, so we classify it as positive
- If cos θ < 0, ***x*** is on the opposite side from ***w***, so we classify it as negative

$$\vec{w} \cdot \vec{x} = \|\vec{w}\| \|\vec{x}\| \cos\theta$$



Perceptron algorithm and intuition

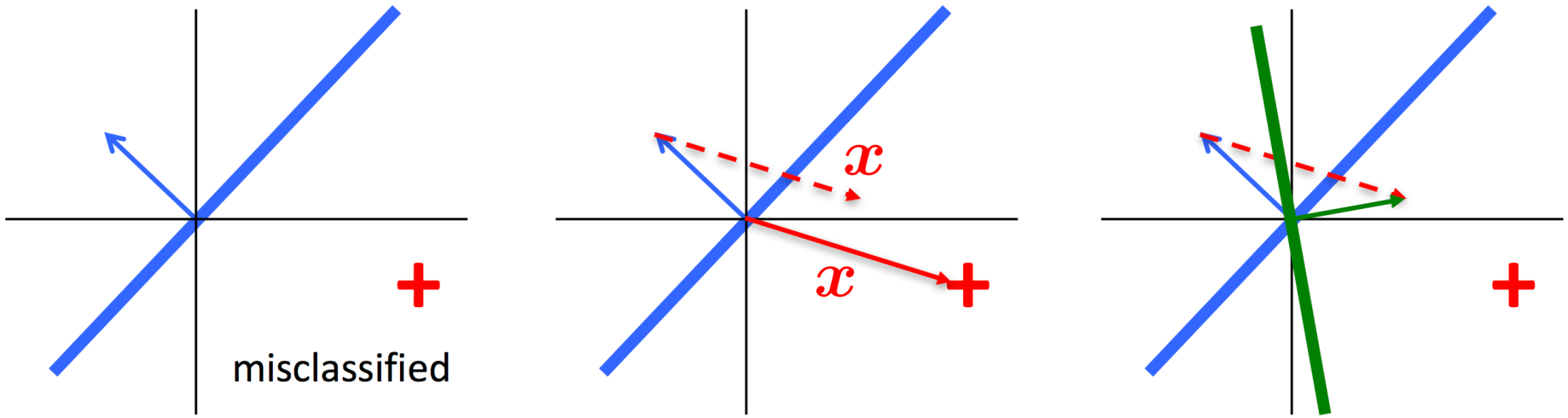
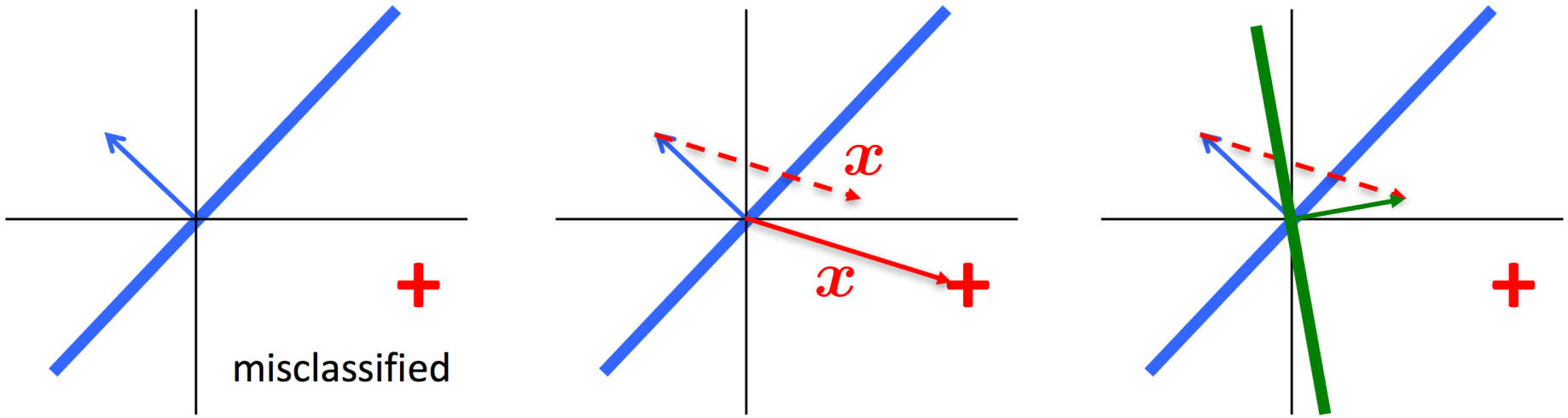


Image and Algorithm: modified from Eric Eaton

Perceptron algorithm and intuition



Let $\vec{w} = [0, 0, \cdots, 0]^T$

Repeat until convergence:

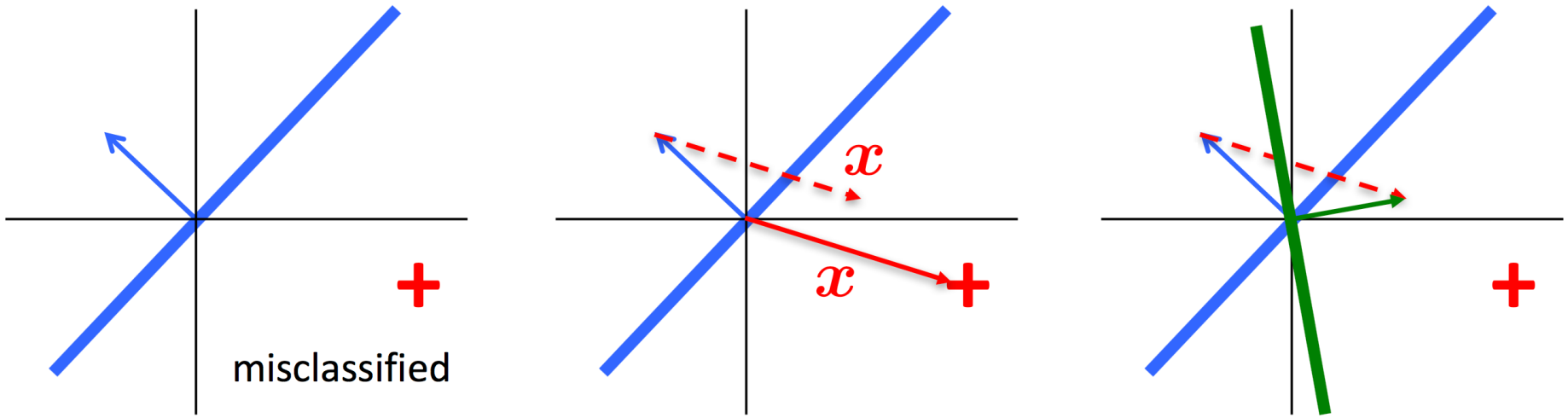
Receive training example $(\vec{x}_i, y_i)$

If $y_i(\vec{w}^T \vec{x}_i) \leq 0$ (incorrectly classified)

$$\vec{w} \leftarrow \vec{w} + \alpha y_i \vec{x}_i$$

Image and Algorithm: modified from Eric Eaton

Perceptron algorithm and intuition



Let $\vec{w} = [0, 0, \cdots, 0]^T$

Repeat until convergence:

Receive training example $(\vec{x}_i, y_i)$

If $y_i(\vec{w}^T \vec{x}_i) \leq 0$ (incorrectly classified)

$$\vec{w} \leftarrow \vec{w} + \alpha y_i \vec{x}_i$$

Convergence:

- All data points correctly classified
- Fixed number of iterations passed

Often: alpha = 1 (only changes magnitude of weight vector)

Image and Algorithm: modified from Eric Eaton

Functional and Geometric Margins

SVM classifier:
(same as perceptron)

$$h(\vec{x}) = \text{sign}\left(\vec{w} \cdot \vec{x} + b\right)$$

Functional and Geometric Margins

SVM classifier:
(same as perceptron)

$$h(\vec{x}) = \text{sign}\left(\vec{w} \cdot \vec{x} + b\right)$$

Functional Margin:

$$\hat{\gamma}_i = y_i\left(\vec{w} \cdot \vec{x}_i + b\right)$$

Functional and Geometric Margins

SVM classifier:
(same as perceptron)

$$h(\vec{x}) = \text{sign}\left(\vec{w} \cdot \vec{x} + b\right)$$

Functional Margin:

$$\hat{\gamma}_i = y_i\left(\vec{w} \cdot \vec{x}_i + b\right)$$

Geometric Margin:
(distance between
example and hyperplane)

$$\gamma_i = y_i\left(\frac{\vec{w}}{\|\vec{w}\|} \cdot \vec{x}_i + \frac{b}{\|\vec{w}\|}\right)$$

Functional and Geometric Margins

SVM classifier:
(same as perceptron)

$$h(\vec{x}) = \text{sign}\left(\vec{w} \cdot \vec{x} + b\right)$$

Functional Margin:

$$\hat{\gamma}_i = y_i\left(\vec{w} \cdot \vec{x}_i + b\right)$$

Geometric Margin:
(distance between
example and hyperplane)

$$\gamma_i = y_i\left(\frac{\vec{w}}{\|\vec{w}\|} \cdot \vec{x}_i + \frac{b}{\|\vec{w}\|}\right)$$

Note:

$$\boxed{\gamma_i = \frac{\hat{\gamma}_i}{\|\vec{w}\|}}$$

Optimization Problem: try 1

Goal: maximize the minimum distance between example and hyperplane

$$\gamma = \min_{i=1,\cdots,n} \gamma_i$$

Optimization Problem: try 1

Goal: maximize the minimum distance between example and hyperplane

$$\gamma = \min_{i=1,\cdots,n} \gamma_i$$

Formulation: optimize a function with respect to a constraint

$$\begin{aligned} \max_{\gamma,\vec{w},b} \quad & \gamma \\ \text{s.t.} \quad & y_i(\vec{w} \cdot \vec{x}_i + b) \geq \gamma, \quad i = 1, \cdots, n \\ \text{and} \quad & \|\vec{w}\| = 1 \end{aligned}$$

(force functional and geometric margin to be equal)

Optimization Problem: try 2

Idea: substitute functional margin
divided by magnitude of weight vector

$$\max_{\hat{\gamma},\vec{w},b} \quad \frac{\hat{\gamma}}{\|\vec{w}\|}$$
$$\text{s.t.} \quad y_i(\vec{w}\cdot\vec{x}_i + b) \geq \hat{\gamma}, \quad i = 1, \cdots, n$$

(gets rid of non-convex constraint)

Optimization Problem: try 3

Idea: put arbitrary constraint on functional margin

$$\hat{\gamma} = 1$$

$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2}\|\vec{w}\|^2 \\ \text{s.t.} \quad & y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1, \quad i = 1, \cdots, n \end{aligned}$$

Optimization Problem: try 3

Idea: put arbitrary constraint on functional margin

$$\hat{\gamma} = 1$$

$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2}\|\vec{w}\|^2 \\ \text{s.t.} \quad & y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1, \quad i = 1, \cdots, n \end{aligned}$$

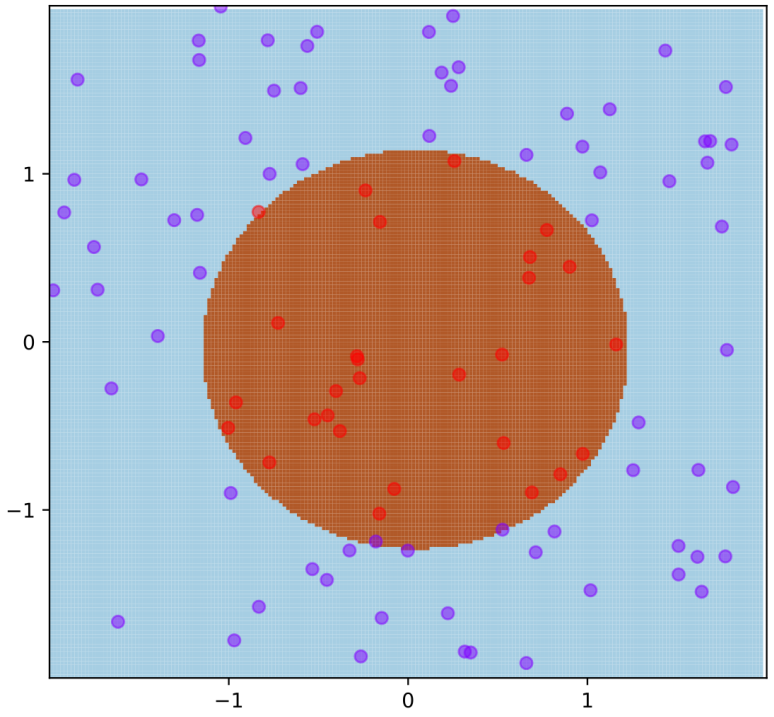
$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2}\|\vec{w}\|^2 \\ \text{s.t.} \quad & -y_i(\vec{w}\cdot\vec{x}_i + b) + 1 \leq 0, \quad i = 1, \cdots, n \end{aligned}$$

Kernel Idea

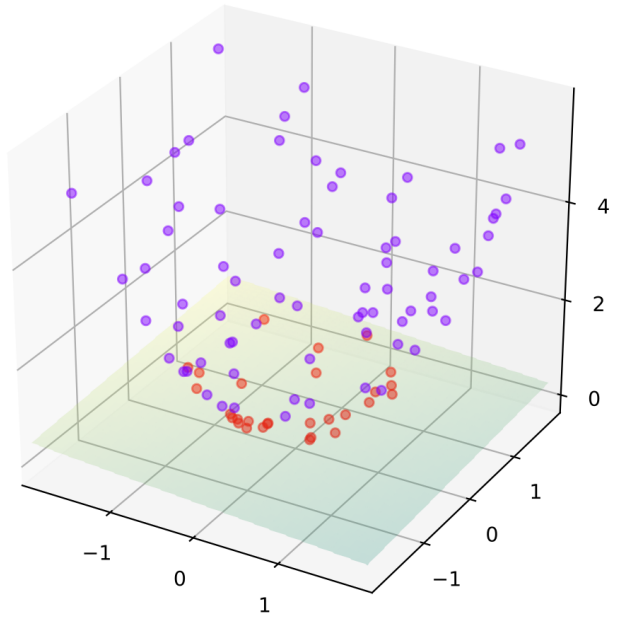
- By solving the dual form of the problem, we have seen how all computations can be done in terms of inner products between examples
- One example of an inner product is the dot product, which is the linear version of SVMs
- But there are many others!
- Intuition: if points are close together, their 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: $K(\boldsymbol{x},\ \boldsymbol{z}) = \boldsymbol{x} \cdot \boldsymbol{z} + ||\boldsymbol{x}||^2\ ||\boldsymbol{z}||^2$

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

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

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

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

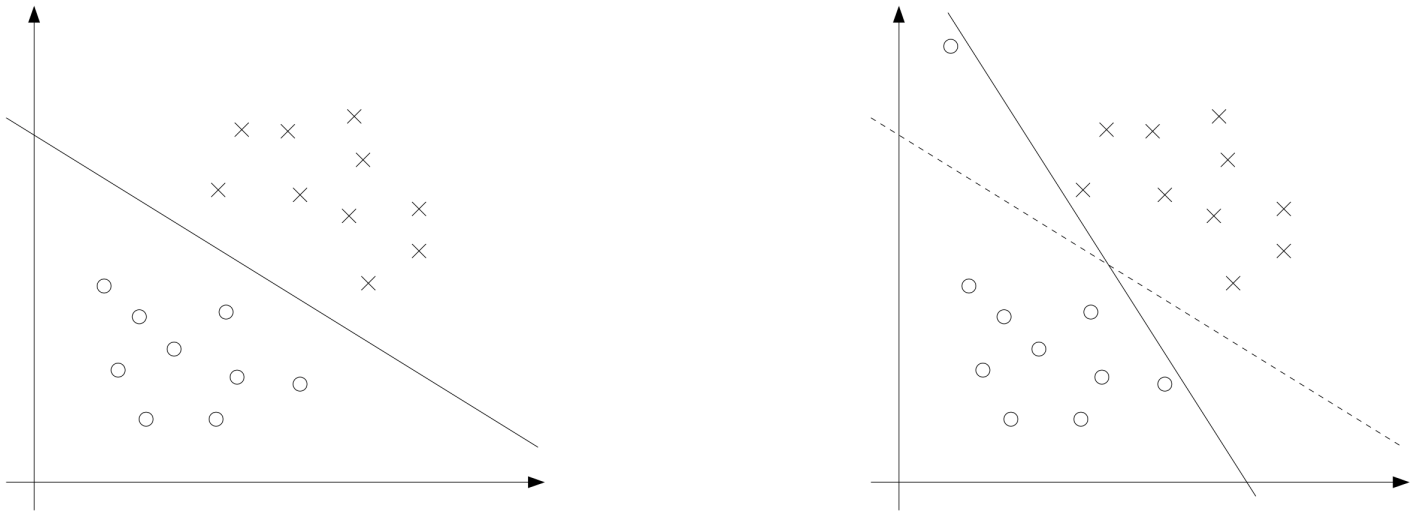


Figure: Andrew Ng

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

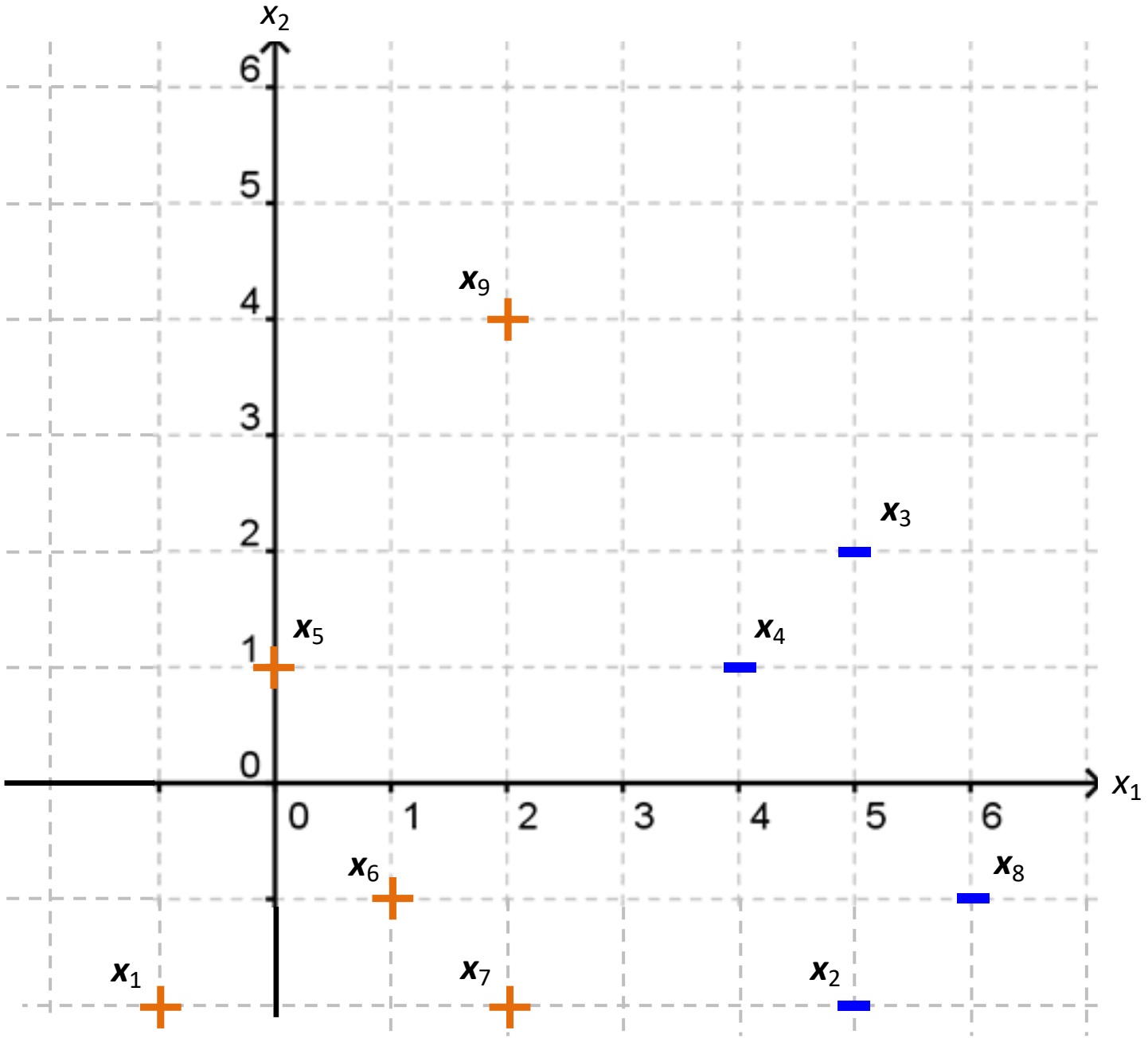
Meta-optimization process

- 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!

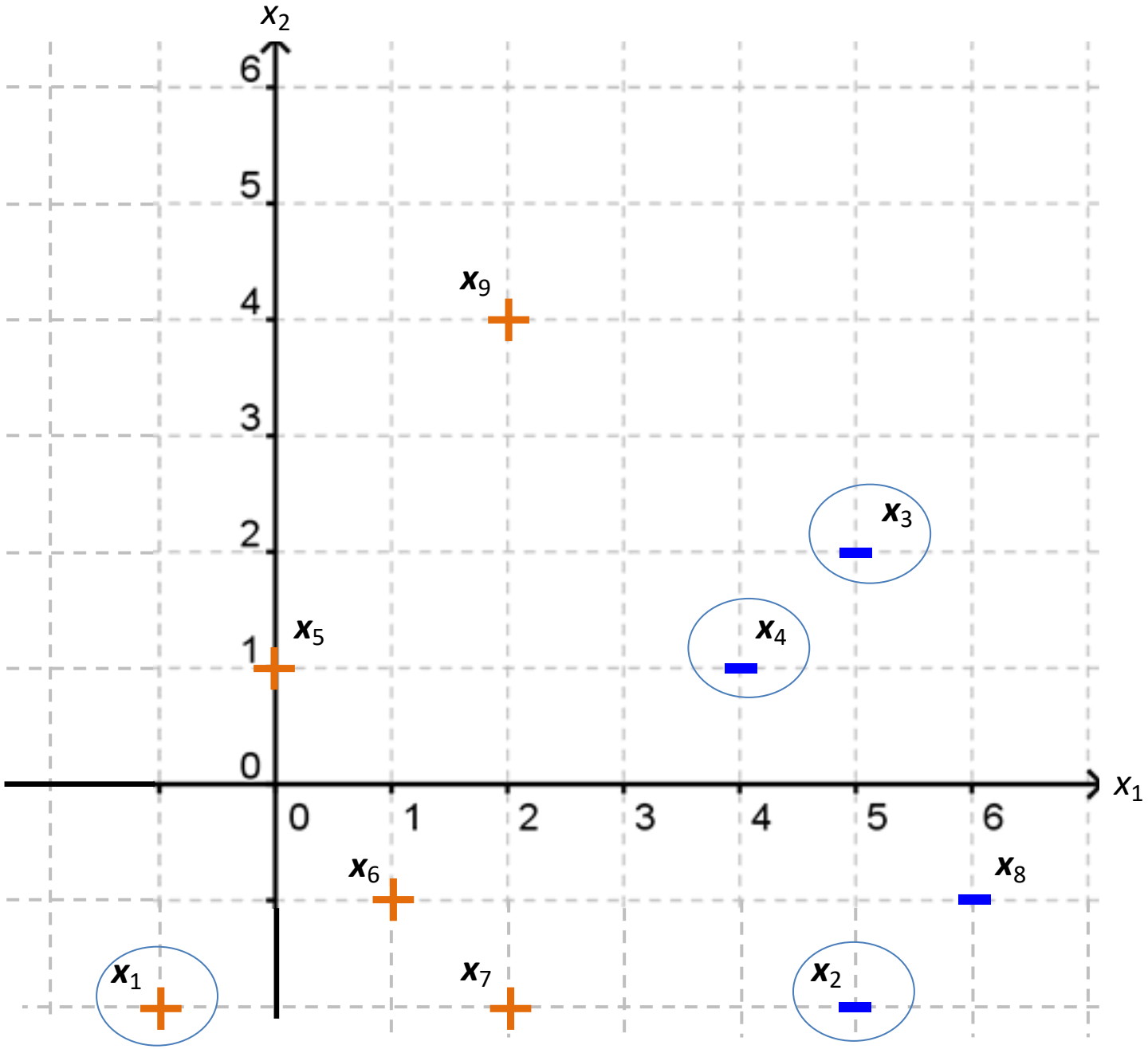
Meta-optimization process

- 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



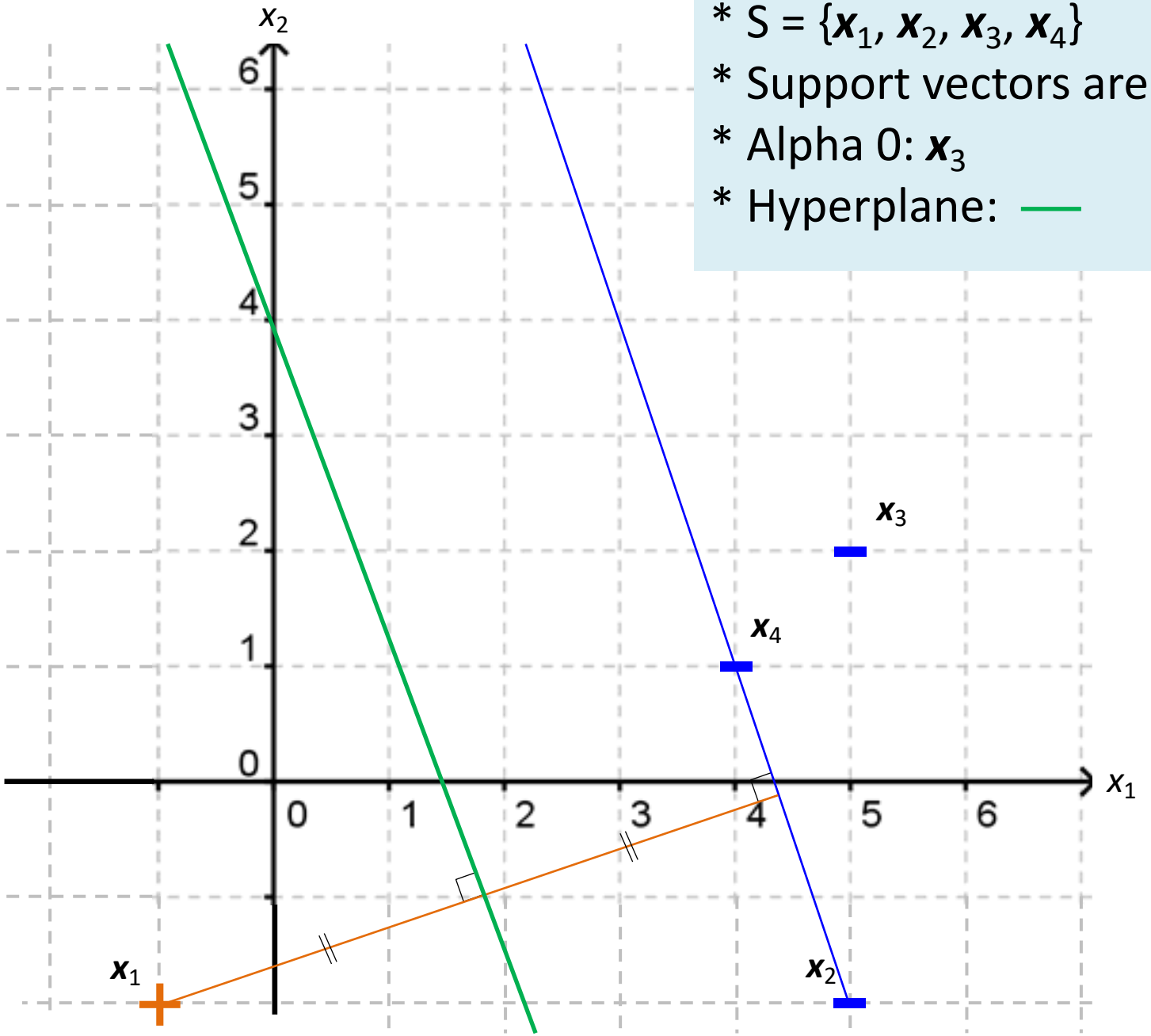
Meta-optimization: example



$K = 4$

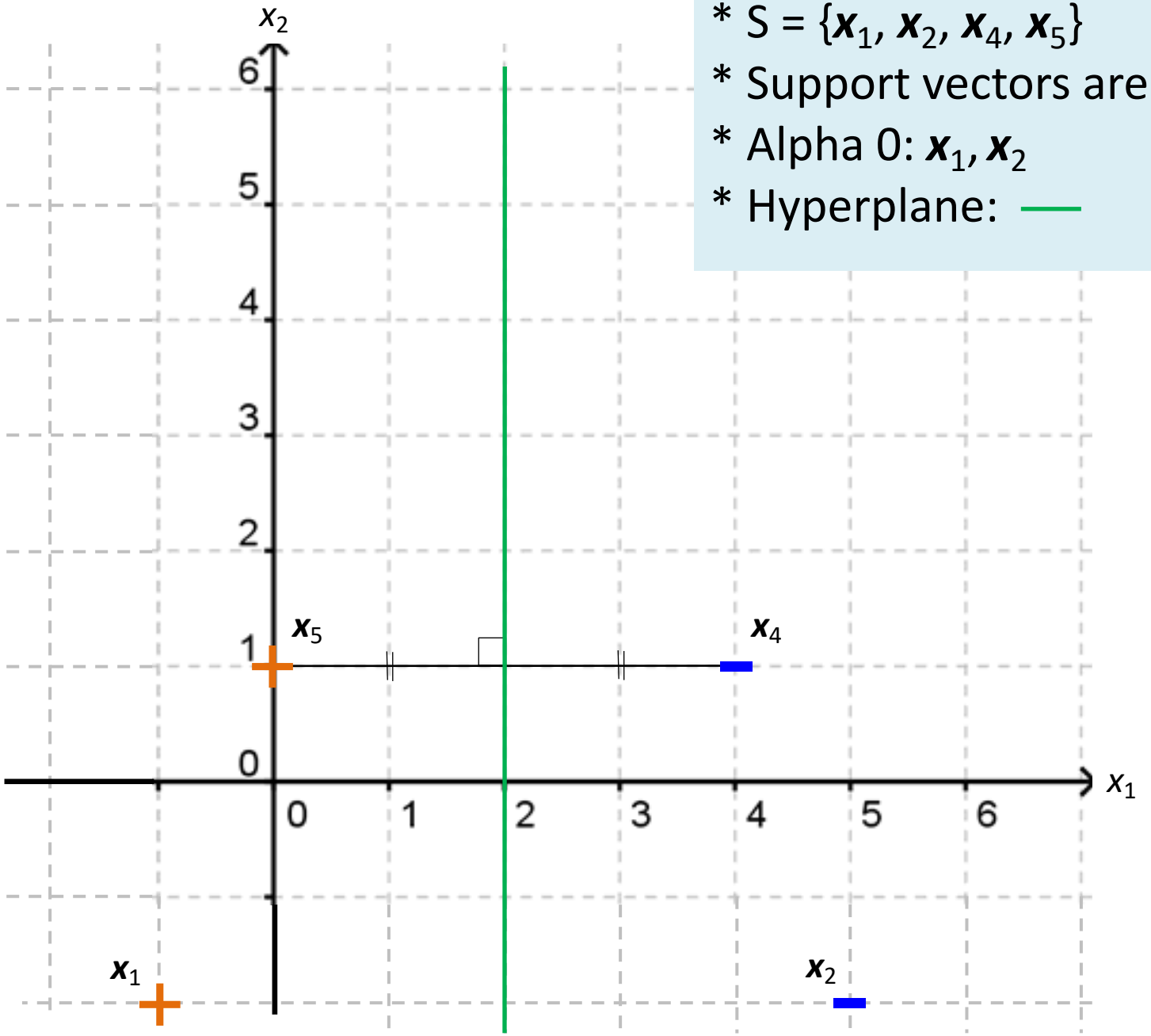
Round 1:

* 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:



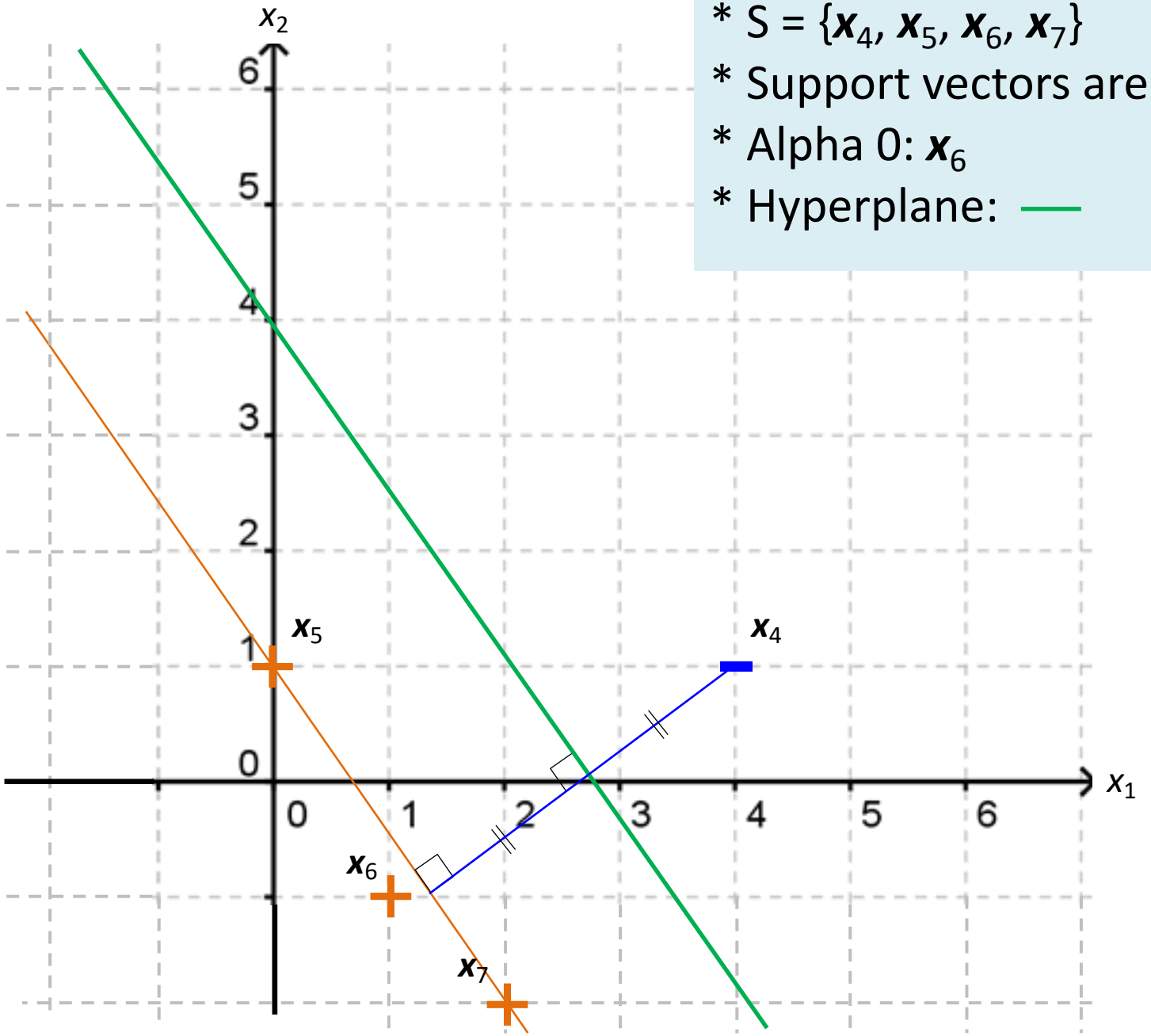
Round 1:

* 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: —



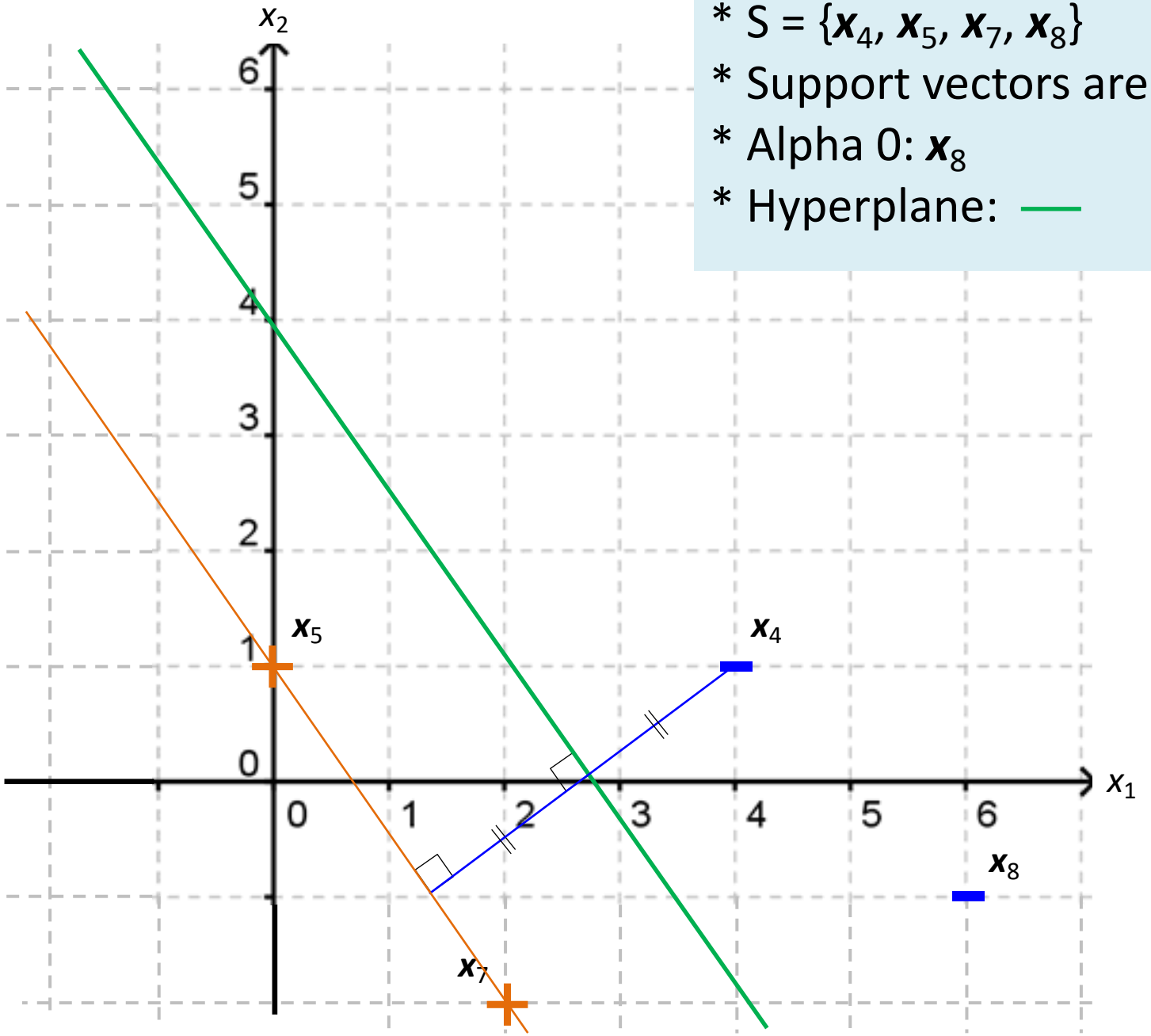
Round 3:

* 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: —



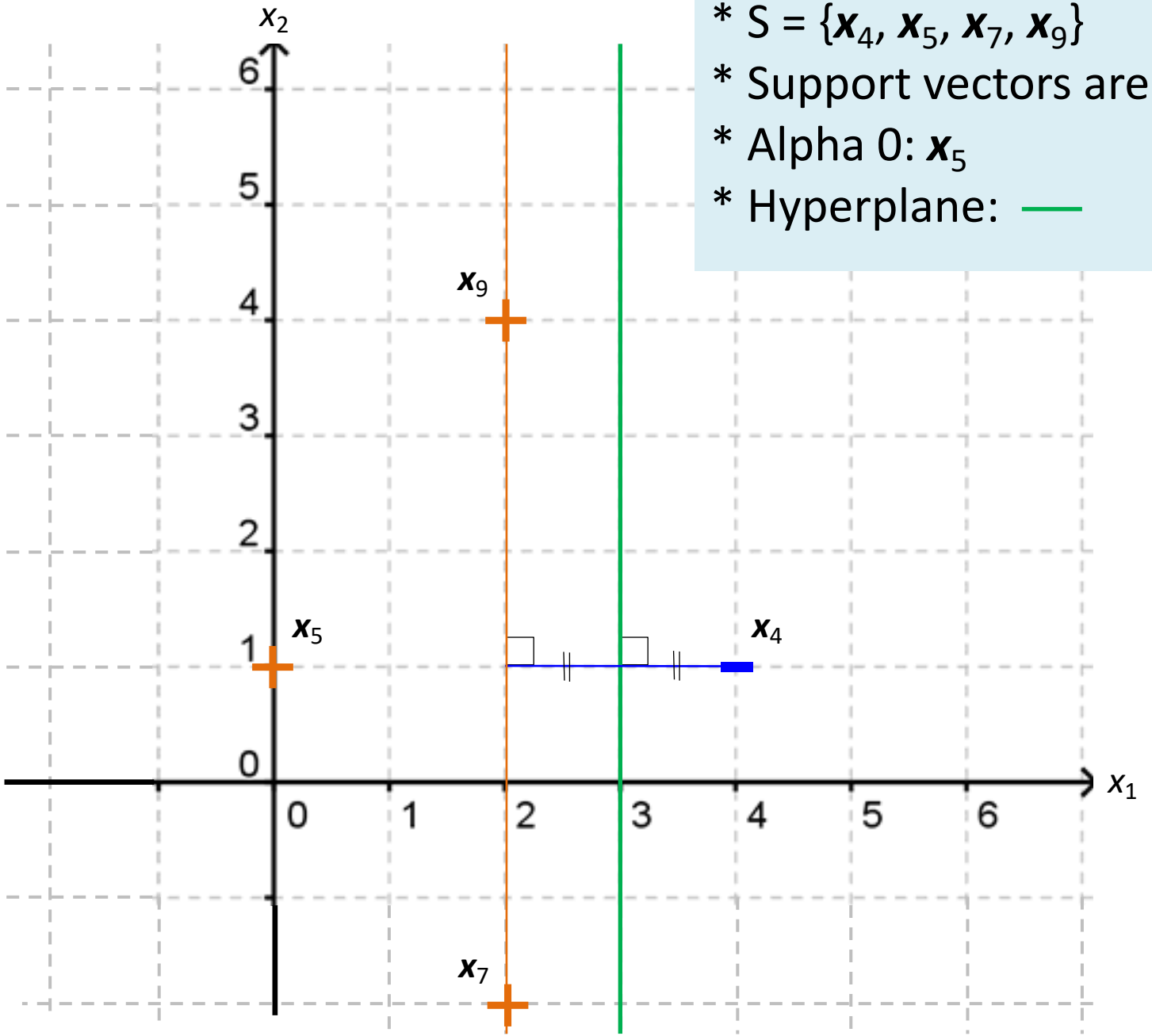
Round 4:

* 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: —

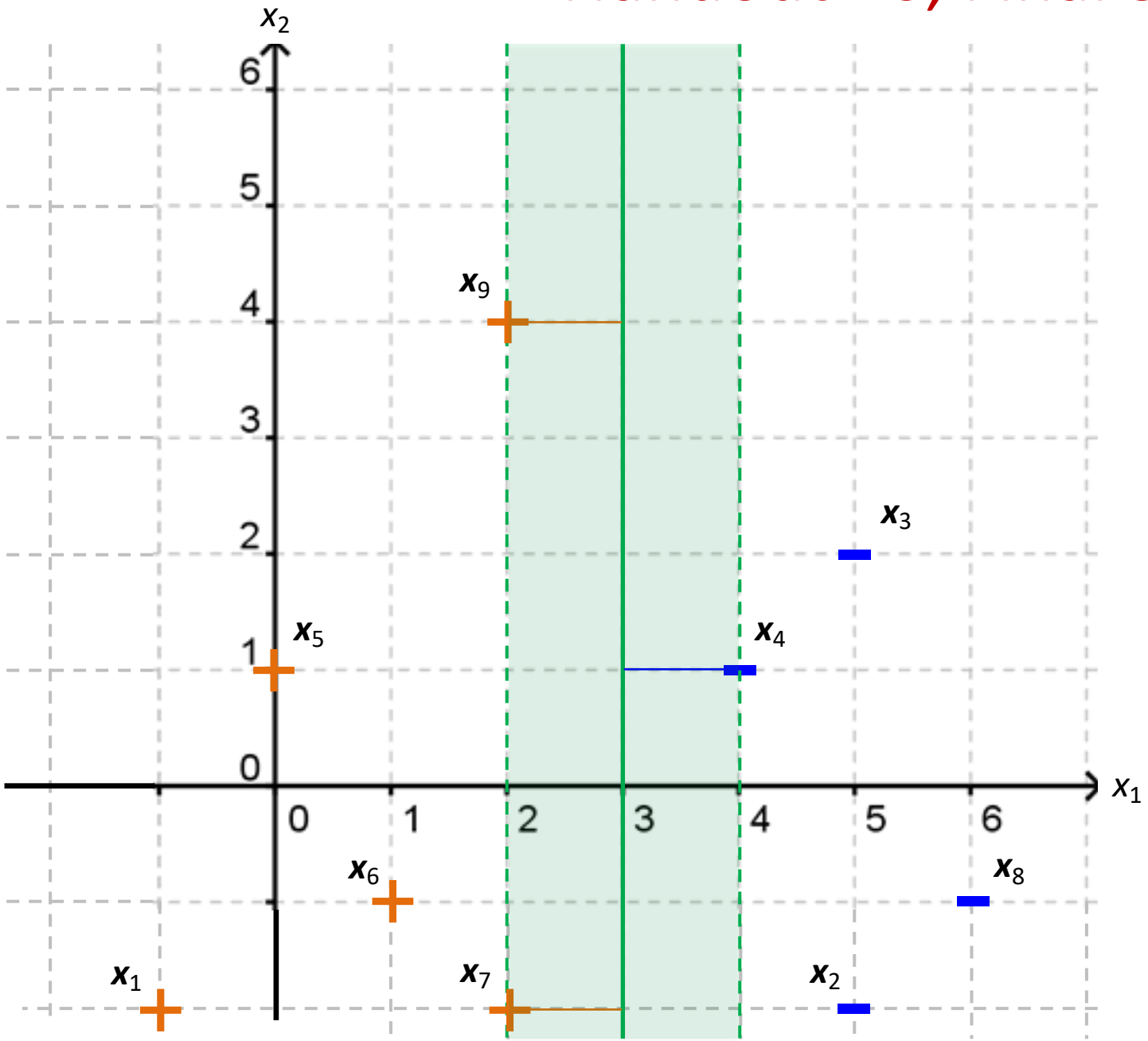


Round 5:

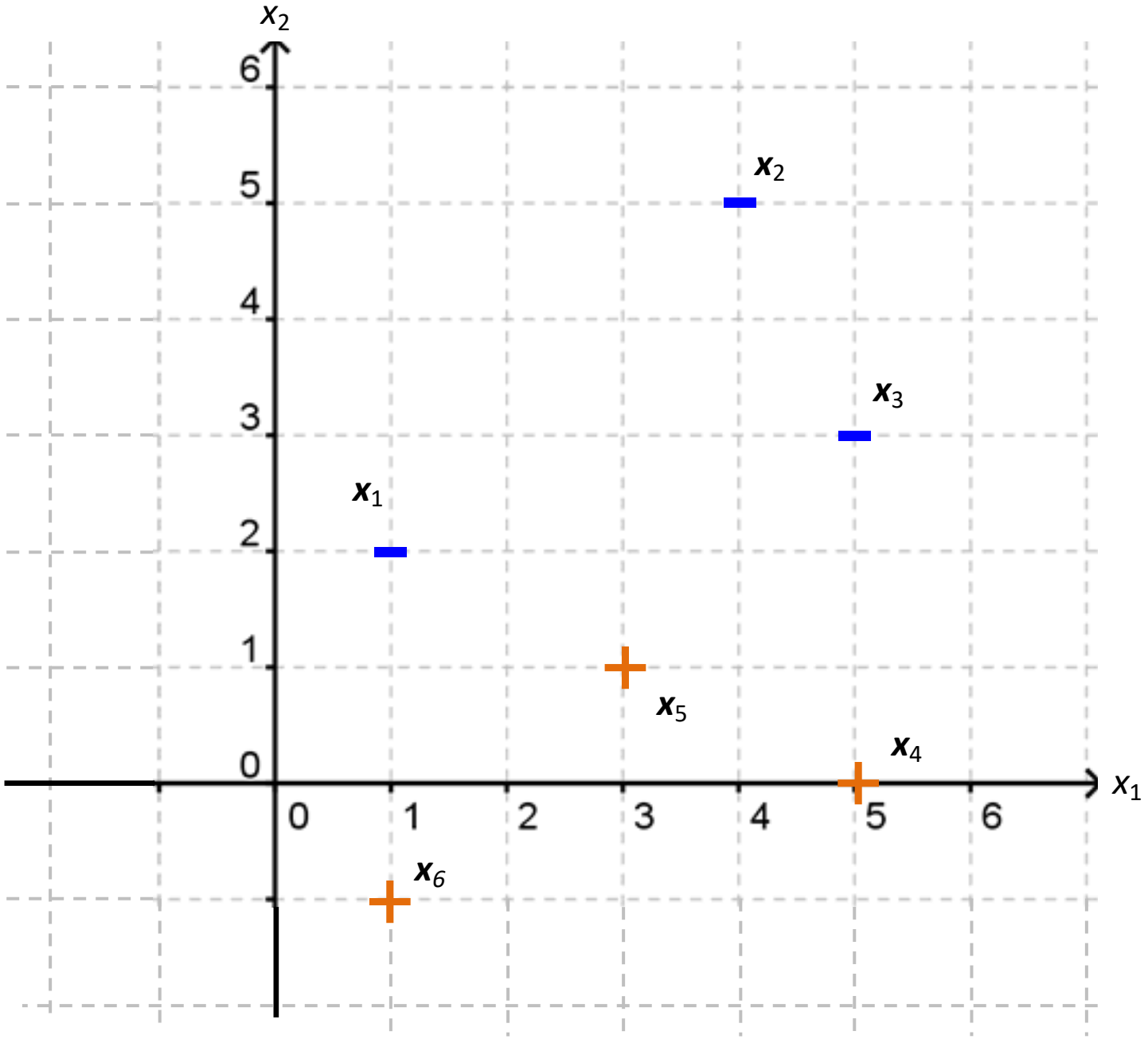
* S = {x_4, x_5, x_7, x_9}
* Support vectors are: x_4, x_7, x_9
* Alpha 0: x_5
* Hyperplane:



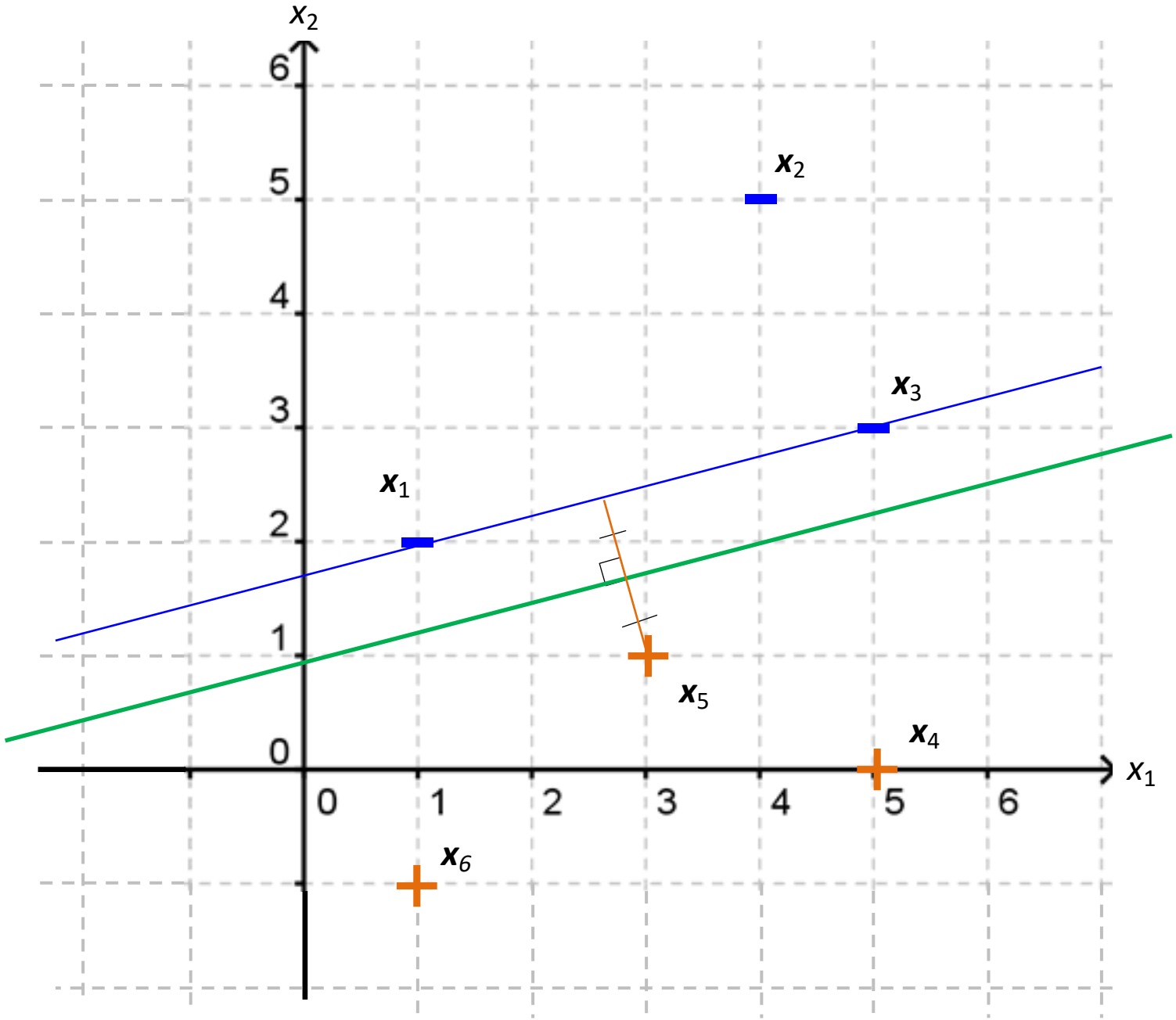
Handout 16, Final Solution

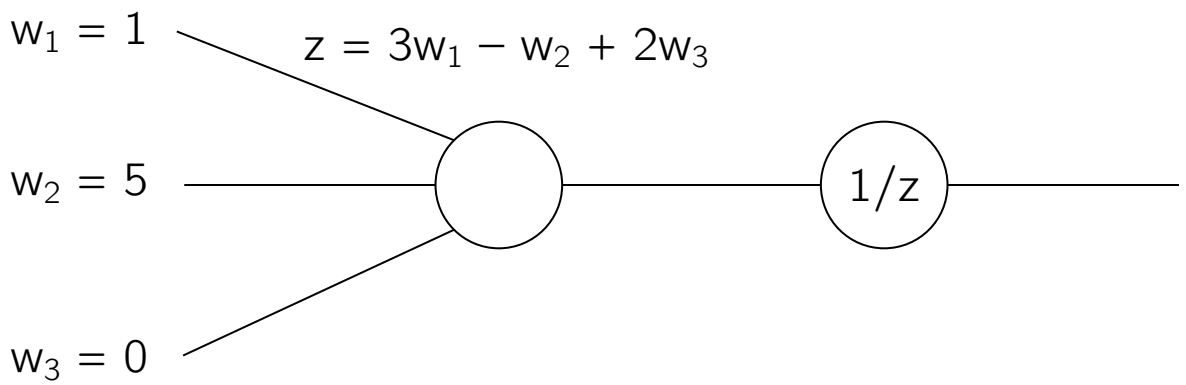


Discuss with a partner: what are the support vectors?



Discuss with a partner: what are the support vectors?




$w_1 = 1$
$z = 3w_1 - w_2 + 2w_3$
$w_2 = 5$
1/z
$w_3 = 0$