PHYS 139/239: Machine Learning in Physics

Lecture 2: Perceptron Learning Algorithm & (Stochastic) Gradient Descent

Javier Duarte — January 12, 2023

Recap: Bias-variance tradeoff

- If L is the squared loss, we can decompose the expected test error:

$$\mathbb{E}\left[L_P(f(x\,|\,w_S)\right] = \mathbb{E}_S\mathbb{E}_{(x,y)\sim P(x,y)}\left[L(y, f(x\,|\,w_S))\right]$$

$$= \mathbb{E}_{(x,y)\sim P(x,y)}\left[\underbrace{\mathbb{E}_S\left[(f(x\,|\,w_S) - F(x))^2\right]}_{\text{Variance}} + \underbrace{(F(x) - y)^2}_{\text{(Squared) bias}}\right]$$

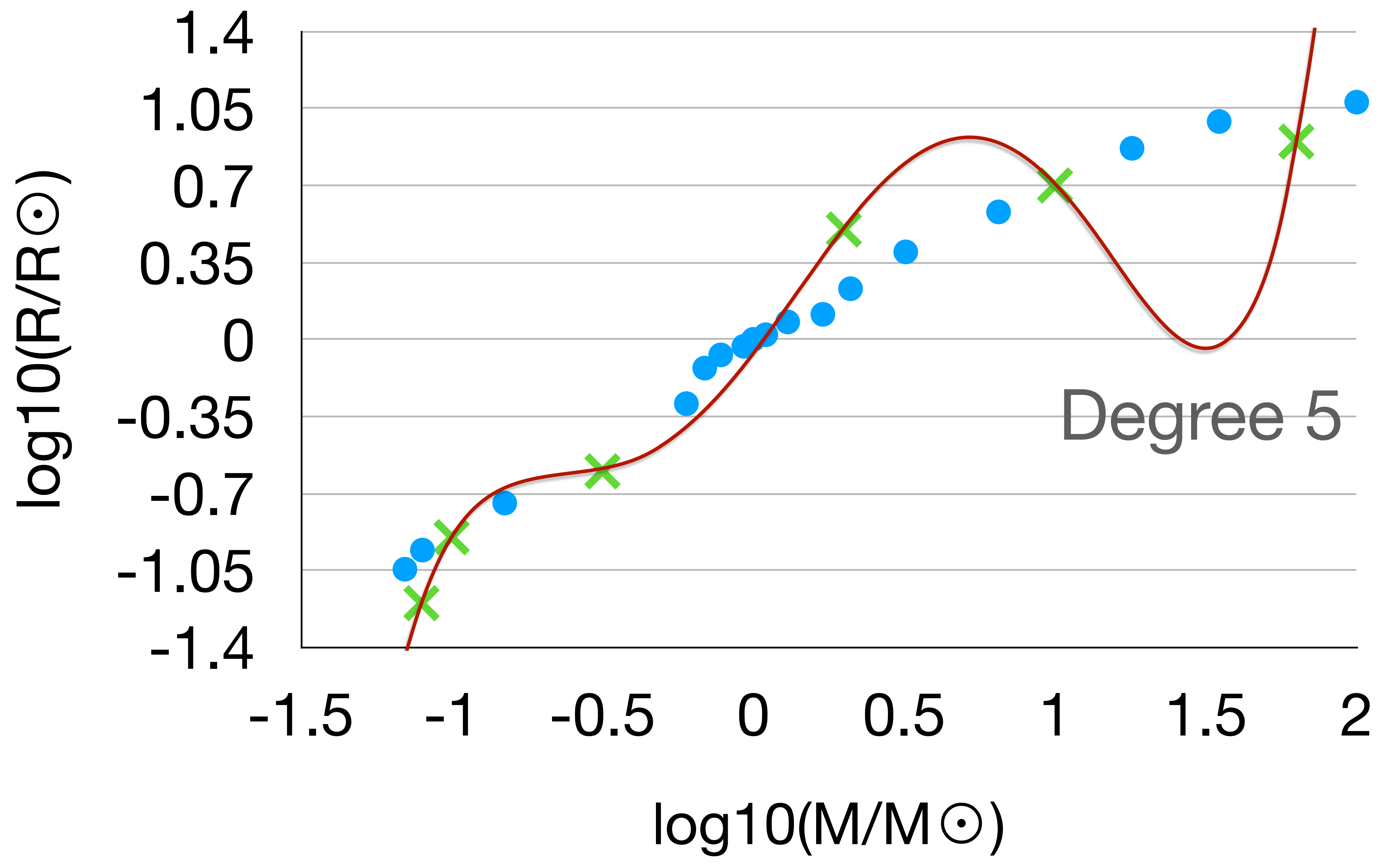
- where $F(x) = \mathbb{E}_S\left[f(x\,|\,w_S)\right]$ is the average prediction of our model over different possible training datasets

- Variance: difference in predictions when training on different datasets

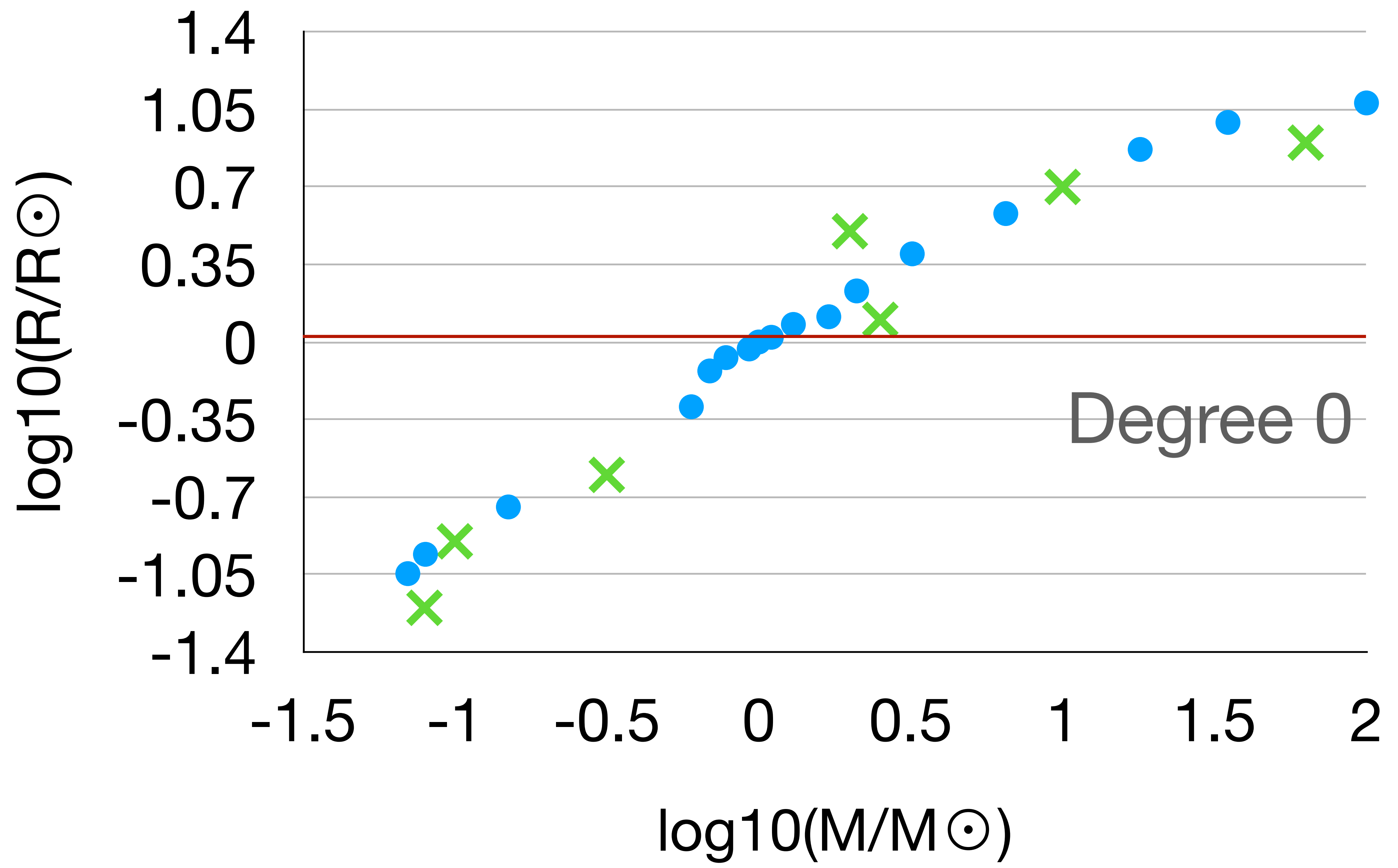
- Bias: difference from ground truth

Overfitting vs. underfitting

- Overfitting implies high variance (unstable model class)
    - Variance increases with model complexity
    - Variance decreases with more training data



- Underfitting implies high bias
    - Even with no variance, model class has high error
    - Underfitting happens whenever model complexity is too low



Model selection

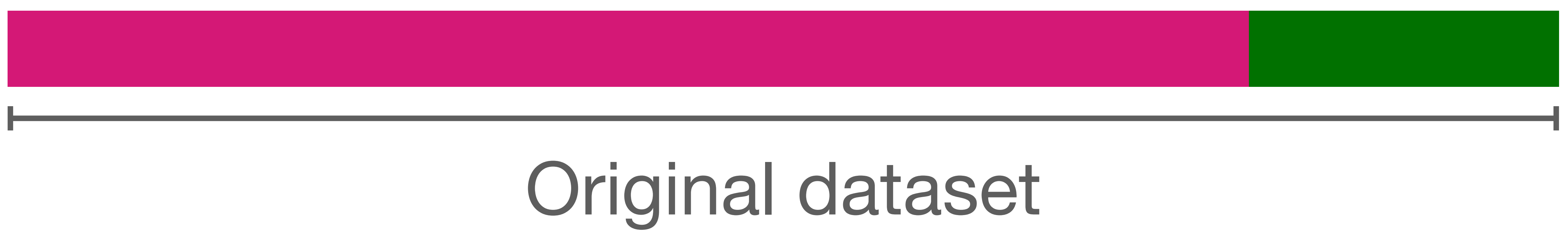
- We only have a finite training dataset
- We cannot measure the true test error
- Simple model classes underfit
- Complex model classes overfit

Bias-variance tradeoff

(but not so straightforward for deep neural networks!)

- **Goal**: Select the model class with the lowest test error

Validation set



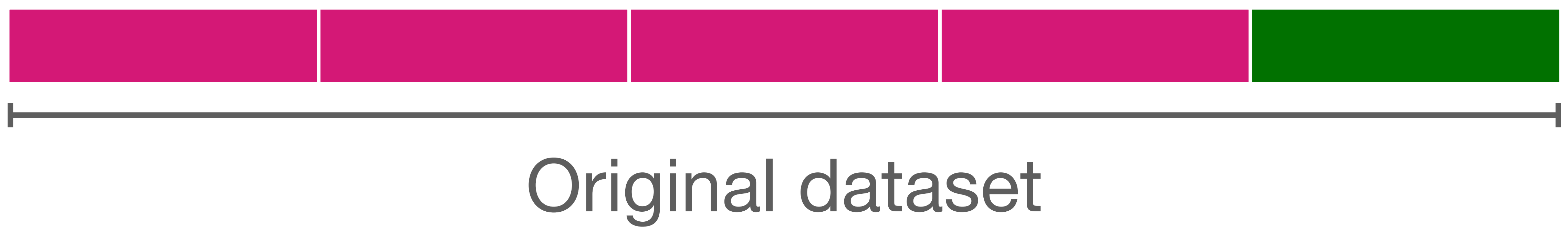
- Split the original dataset into a training and validation set
- Train model on the training set
- Evaluate on the validation set to estimate the test error
- Select the model class that gives the lowest estimated error
- Optionally, re-train the selected model class on the whole dataset (training + validation)

- **Issue**: we would like both training and validation sets to be as large as possible (so that the estimate is better), but they must not overlap!

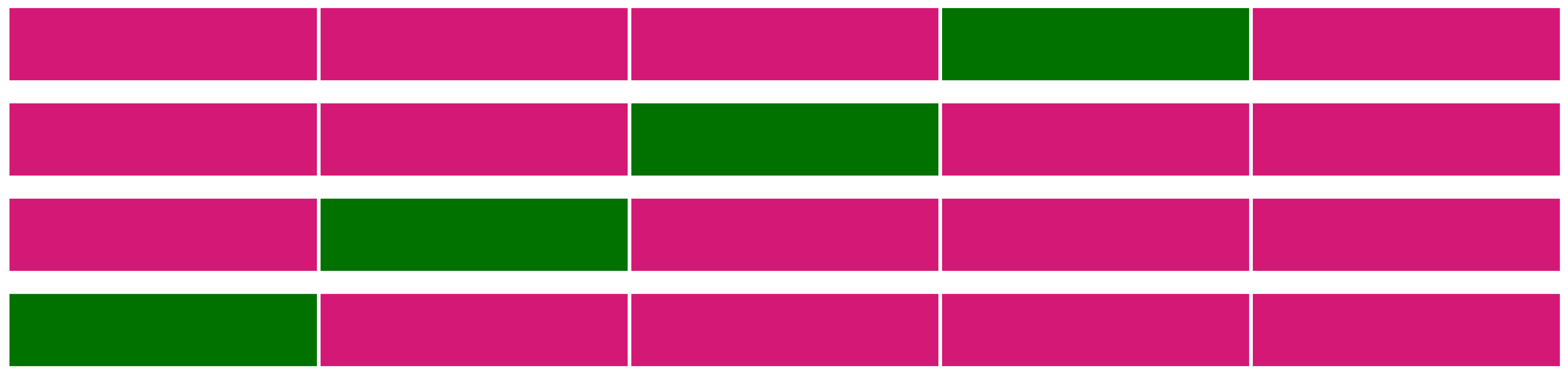
k-fold cross-validation

- Split the original dataset into k equal parts (e.g, $k = 5$)

- Train on the $k - 1$ parts and validate on the remaining one



- Repeat for every choice of the $k - 1$ parts and average the validation errors



- **Advantage**: use all data as validation to improve the estimate of the test error, at the cost of more computation (k trainings)

Recap: Supervised learning pipeline

- Training dataset: $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x \in \mathbb{R}^D$ and $y \in \mathbb{R}$
- Model / hypothesis class: $f(x \mid w) = w^\intercal x$ (linear models)

  or $\phi(x)$ instead of x
- Loss function: $L(y, y') = (y - y')^2$ (squared loss)
- Optimization algorithm to minimize the learning objective:

$$\arg\min_w \sum_{i=1}^{N} L(y_i, f(x_i \mid w))$$

- Cross validation and model selection:

- Testing and deployment

Important: if a testing set is available, never use it to make decisions on the model!

Today: Learning algorithms

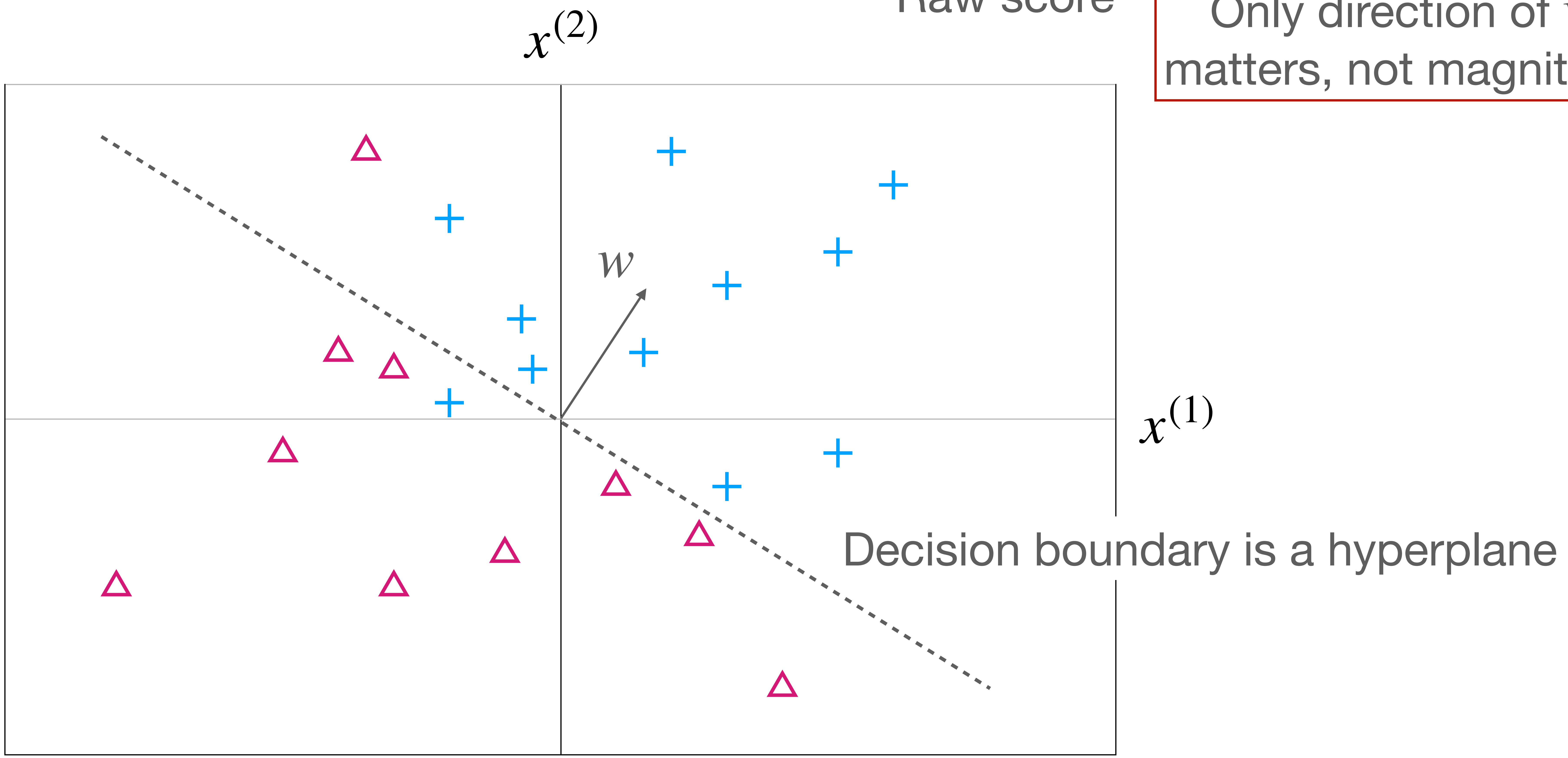
- Perceptron learning algorithm
- (Stochastic) gradient descent
    - Solving the actual optimization problem in general
- How to view the perceptron learning algorithm as an example of stochastic gradient descent

Linear models for binary classification

- Linear model for regression: $f(x \mid w) = w^\intercal x$

- Linear model for binary classification: $f(x \mid w) = \text{sign}(\underbrace{w^\intercal x}_{\text{Raw score}}) \in \{\overset{+}{+1}, \overset{\triangle}{-1}\}$

Only direction of w matters, not magnitude



Linearly separable datasets

- Linear model for binary classification: $f(x \mid w) = \text{sign}(w^\intercal x)$

Not linearly separable:
no hyperplane separates the classes perfectly

Linearly separable:
there is a hyperplane that separates the classes

Margin of the classifier = distance between the decision boundary and the closest data point

Linearly separable datasets

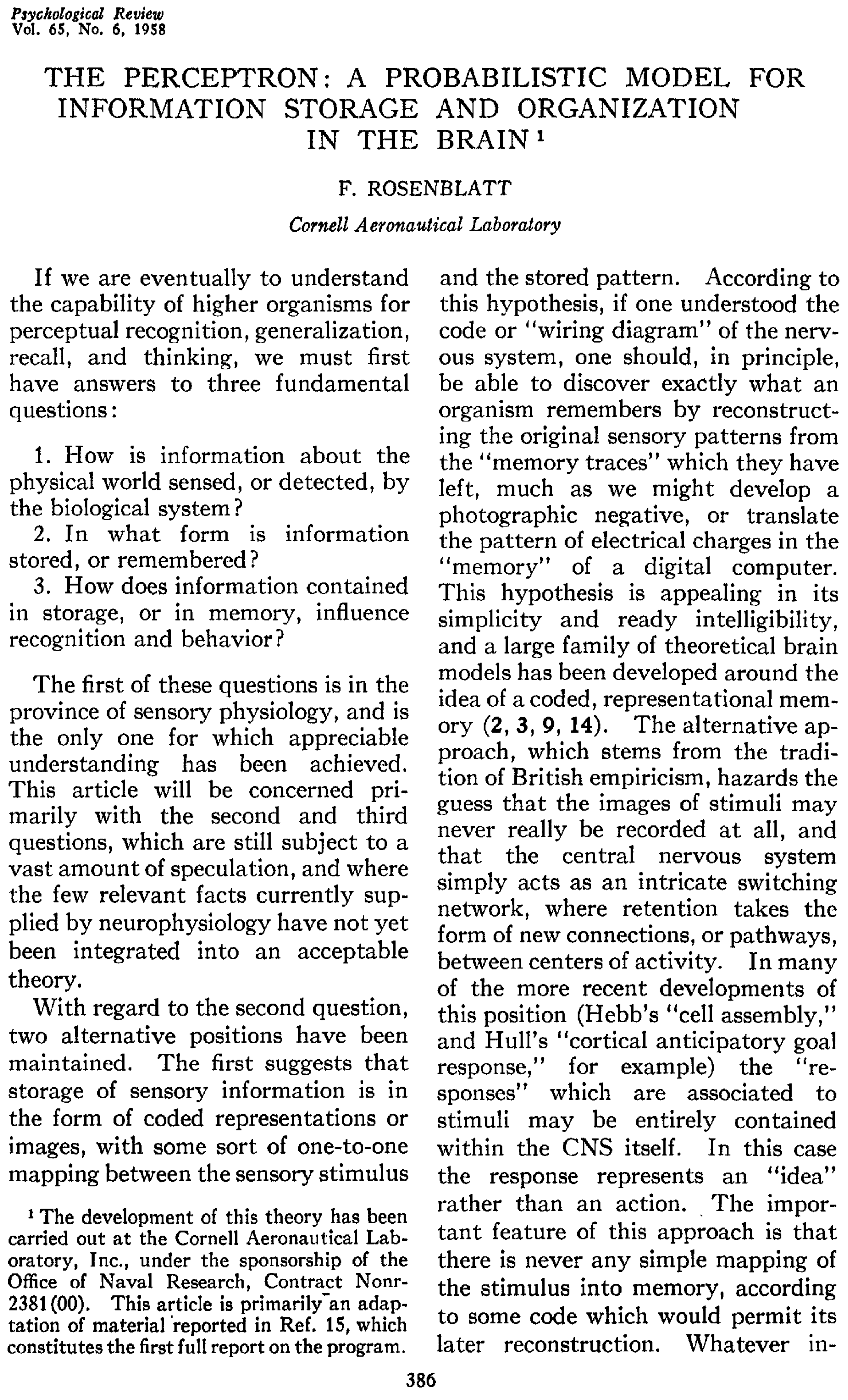
- By adding a bias term, the decision boundary does not have to pass through the origin

Not linearly separable without bias

Adding the bias (through dummy feature $x^{(0)} = 1$) makes it linearly separable

Perceptron

- One of the earliest learning algorithms
  - 1957 by Frank Rosenblatt [10.1037/h0042519]
- Still a great algorithm
  - Fast
  - Clean analysis
  - Precursor to neural networks



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN [1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

The first of these questions is in the province of sensory physiology, and is the only one for which appreciable understanding has been achieved. This article will be concerned primarily with the second and third questions, which are still subject to a vast amount of speculation, and where the few relevant facts currently supplied by neurophysiology have not yet been integrated into an acceptable theory.

With regard to the second question, two alternative positions have been maintained. The first suggests that storage of sensory information is in the form of coded representations or images, with some sort of one-to-one mapping between the sensory stimulus and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain models has been developed around the idea of a coded, representational memory (**2**, **3**, **9**, **14**). The alternative approach, which stems from the tradition of British empiricism, hazards the guess that the images of stimuli may never really be recorded at all, and that the central nervous system simply acts as an intricate switching network, where retention takes the form of new connections, or pathways, between centers of activity. In many of the more recent developments of this position (Hebb's "cell assembly," and Hull's "cortical anticipatory goal response," for example) the "responses" which are associated to stimuli may be entirely contained within the CNS itself. In this case the response represents an "idea" rather than an action. The important feature of this approach is that there is never any simple mapping of the stimulus into memory, according to some code which would permit its later reconstruction. Whatever in-

[1] The development of this theory has been carried out at the Cornell Aeronautical Laboratory, Inc., under the sponsorship of the Office of Naval Research, Contract Nonr-2381(00). This article is primarily an adaptation of material reported in Ref. 15, which constitutes the first full report on the program.

Perceptron learning algorithm

- Set $w(t=0) = 0$
- At iteration t,
  - Receive example (x, y)
  - If $f(x \mid w(t)) = y$ (example is correctly classified)
    - $w(t+1) = w(t)$ (no update)
  - Else
    - $w(t+1) = w(t) + yx$ (update!)

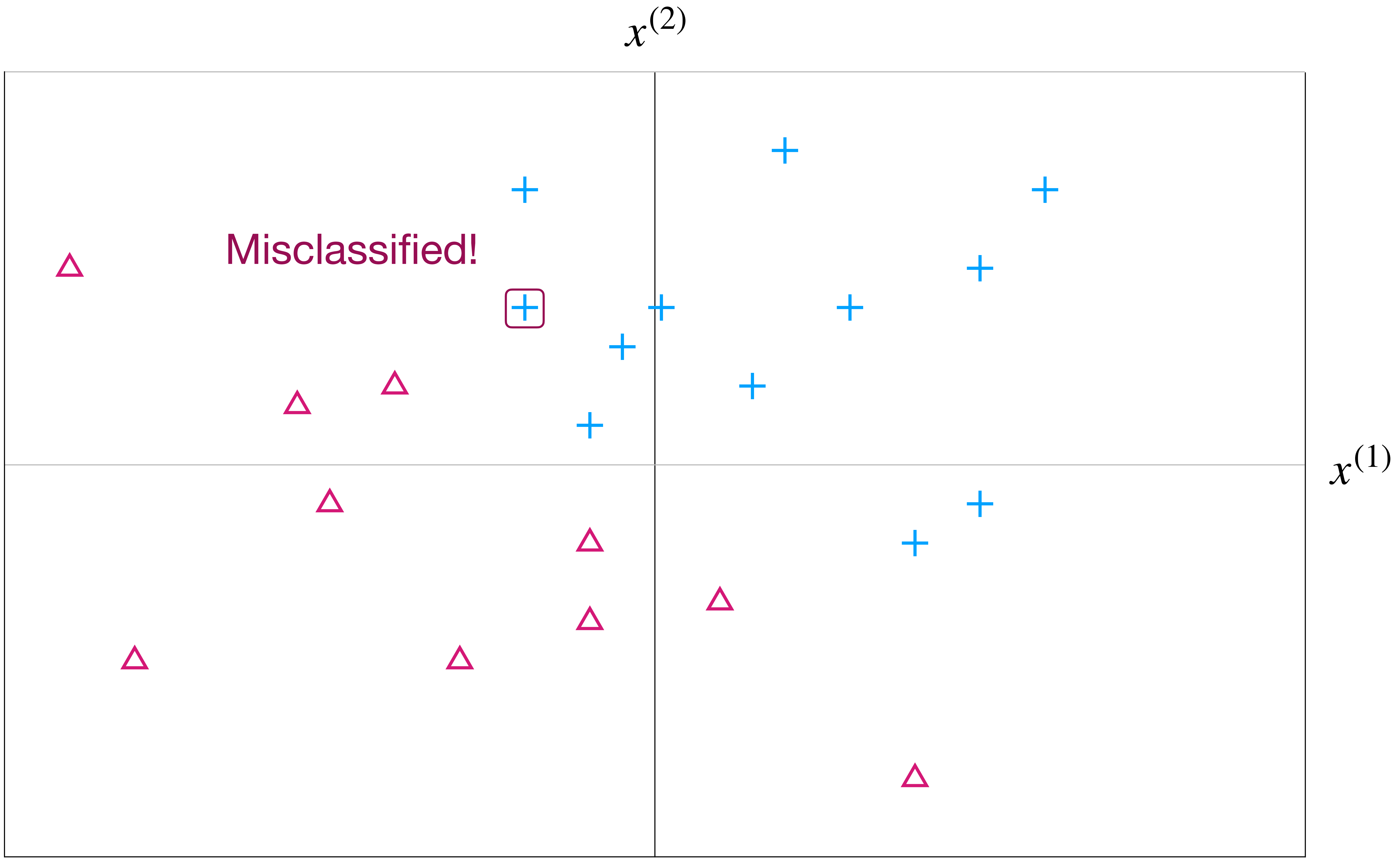
Go through training set in an arbitrary order (e.g., randomly)

Model:
$$f(x \mid w) = \mathrm{sign}(w^\intercal x)$$

Training set:
$$S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$$
$$y \in \{+1, -1\}$$

Perceptron learning algorithm

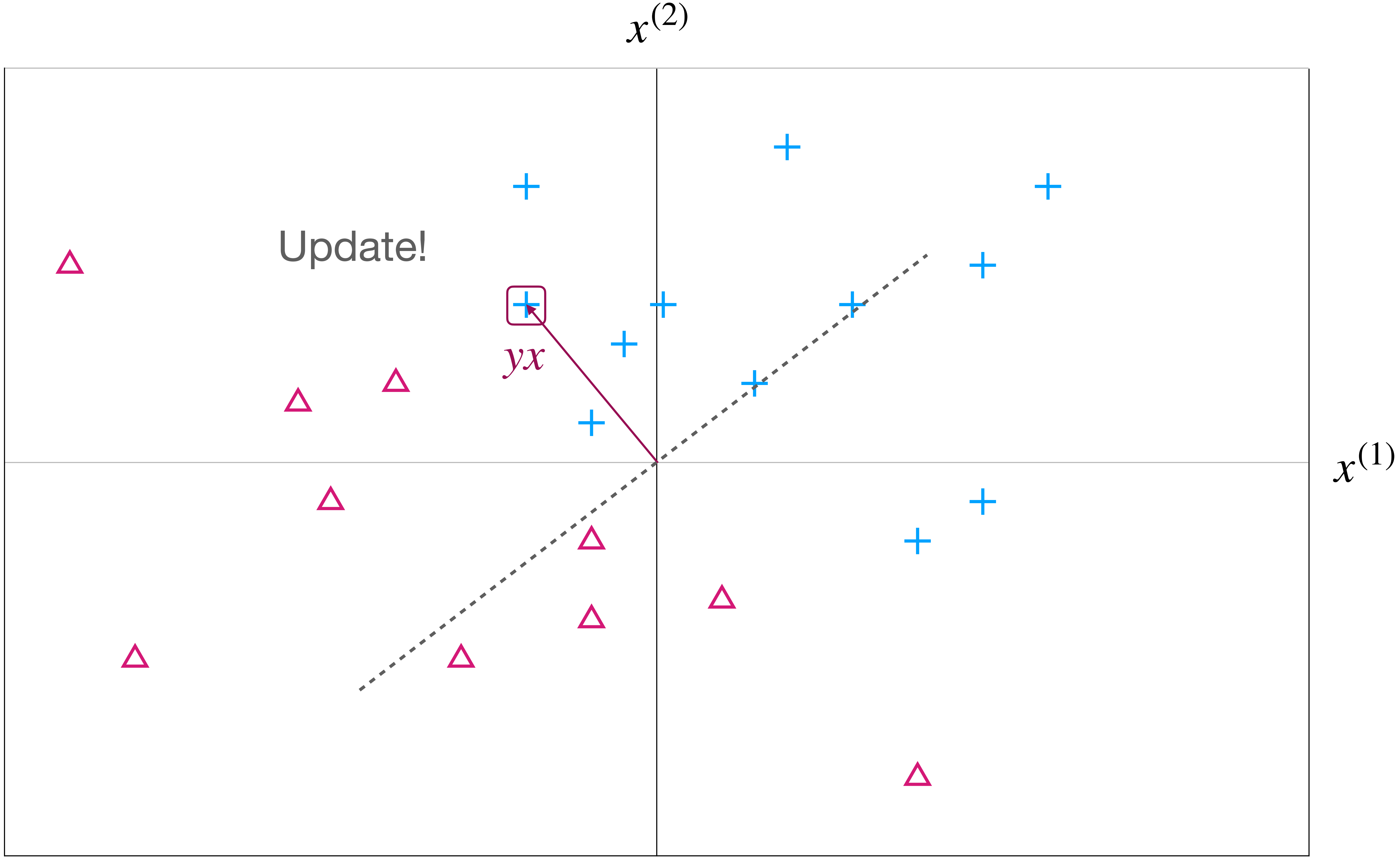
$t = 0$



Note: assume $\mathrm{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{else} \end{cases}$

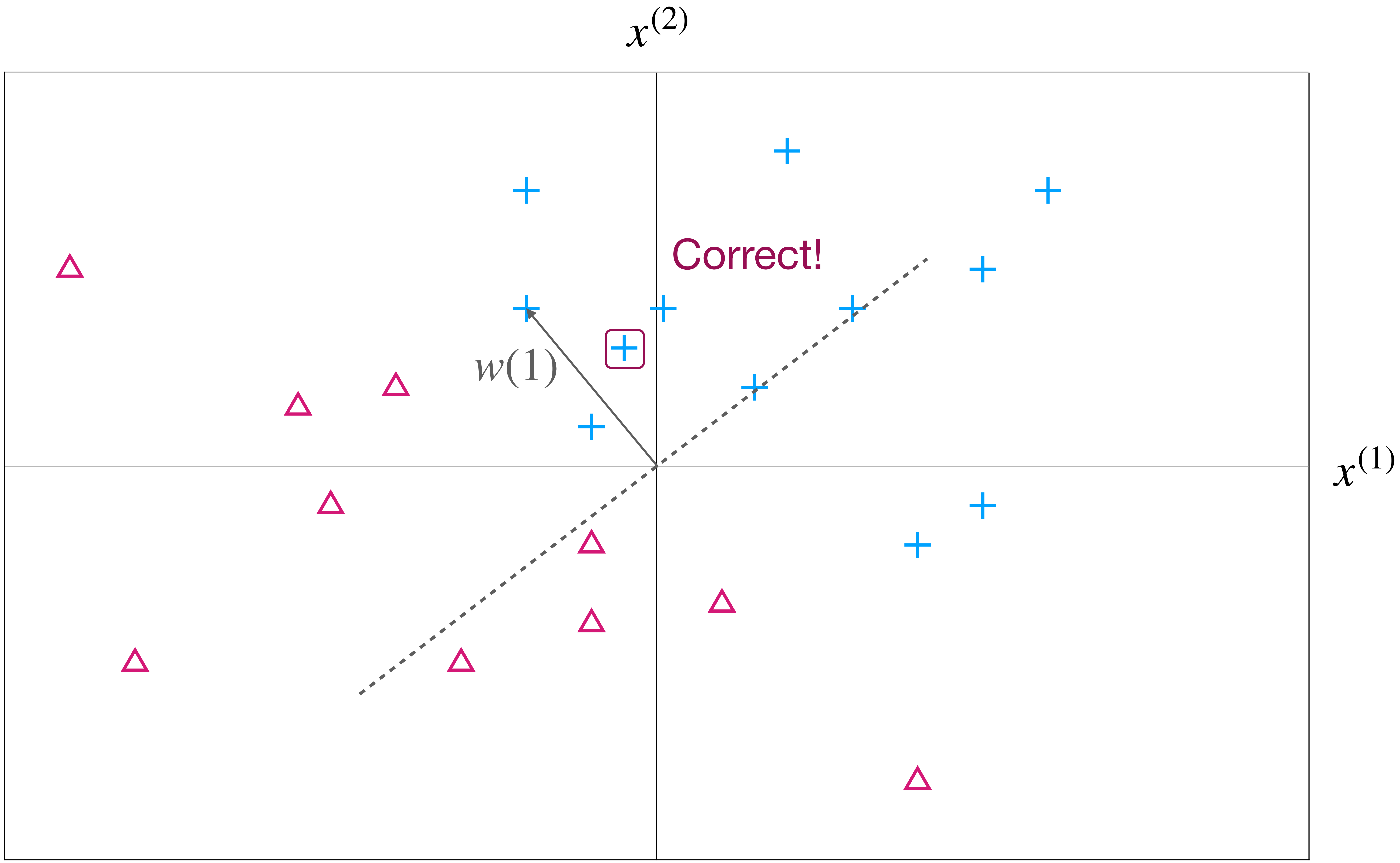
Perceptron learning algorithm

$t = 0$



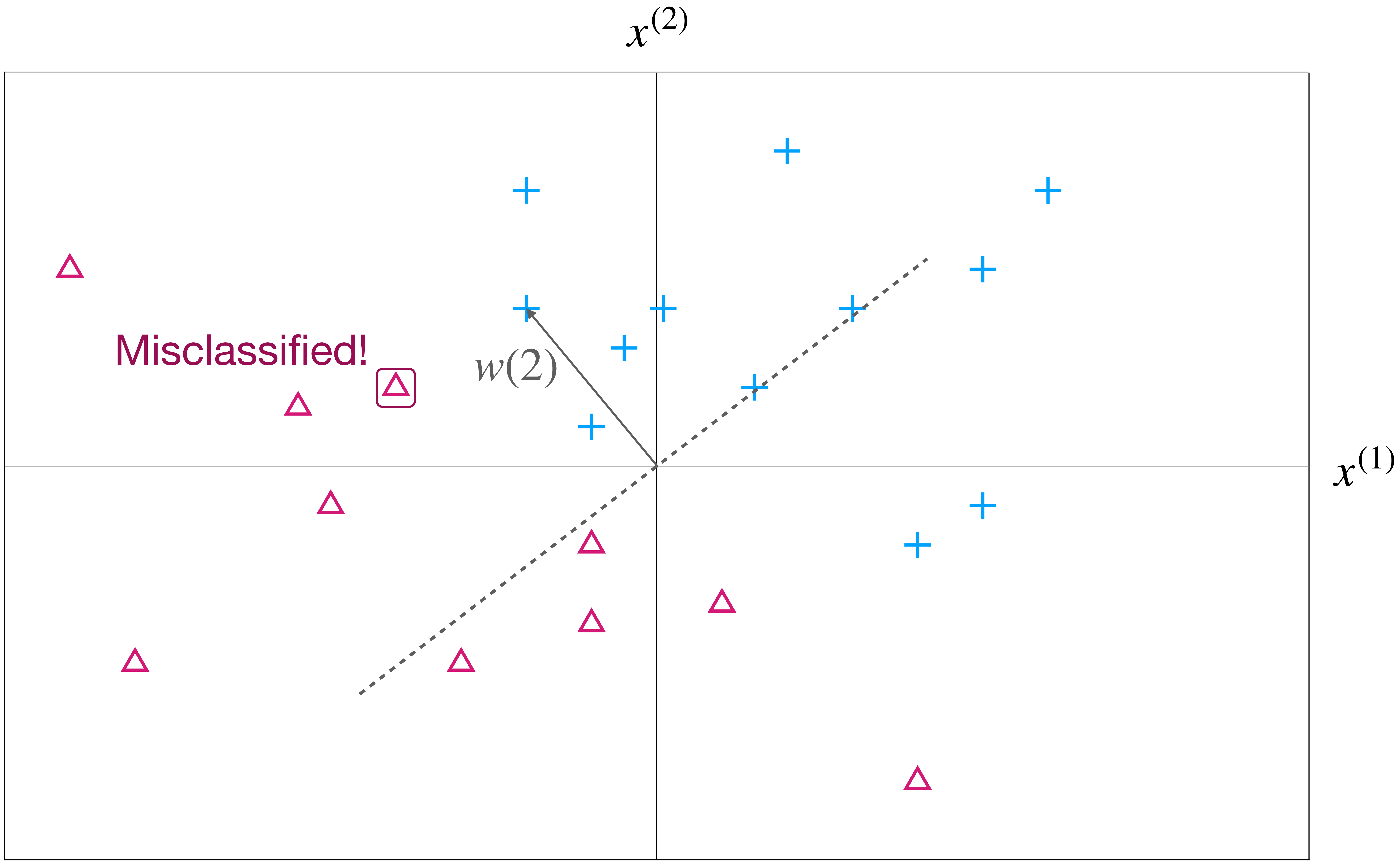
Perceptron learning algorithm

$t = 1$



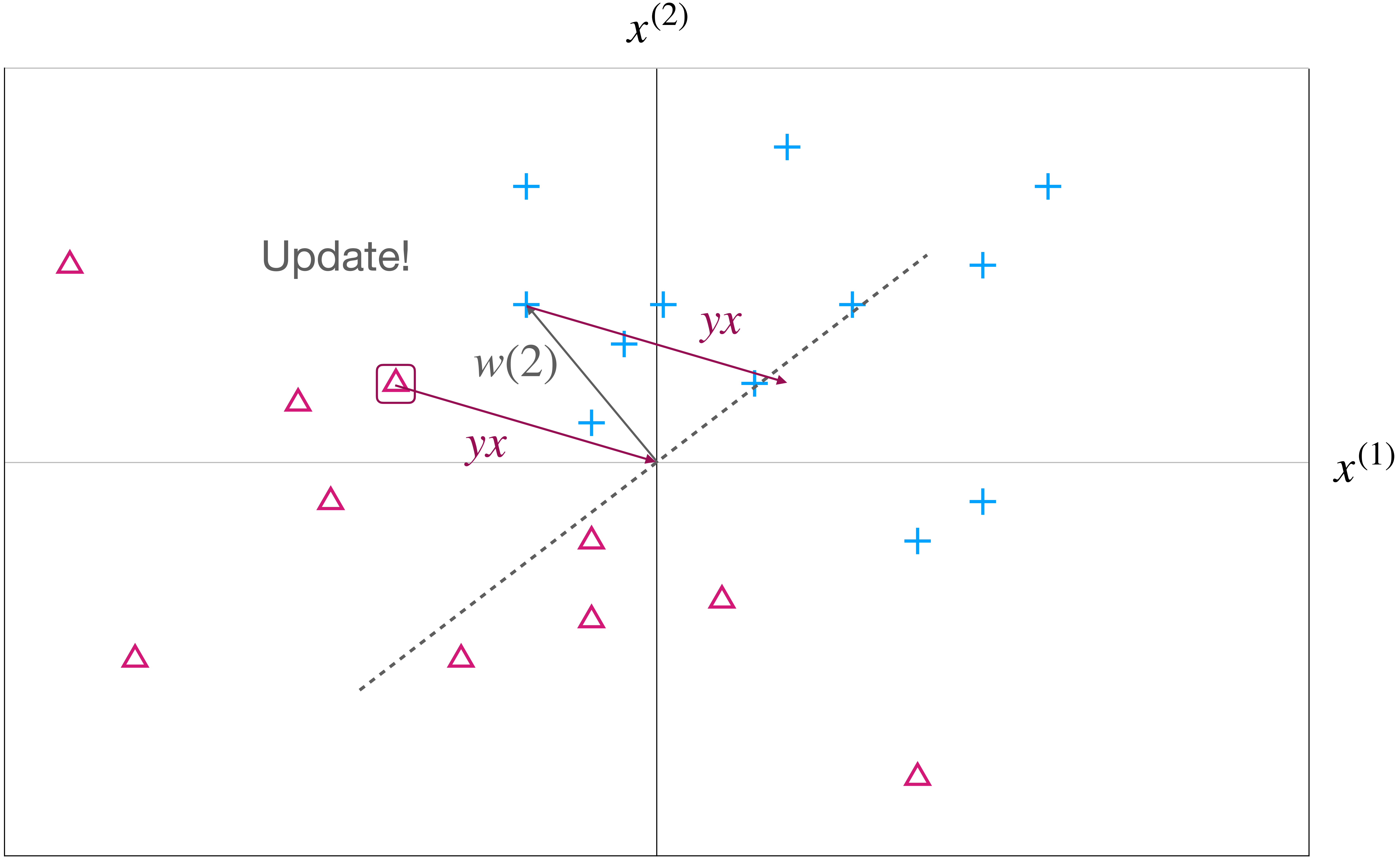
Perceptron learning algorithm

$t = 2$



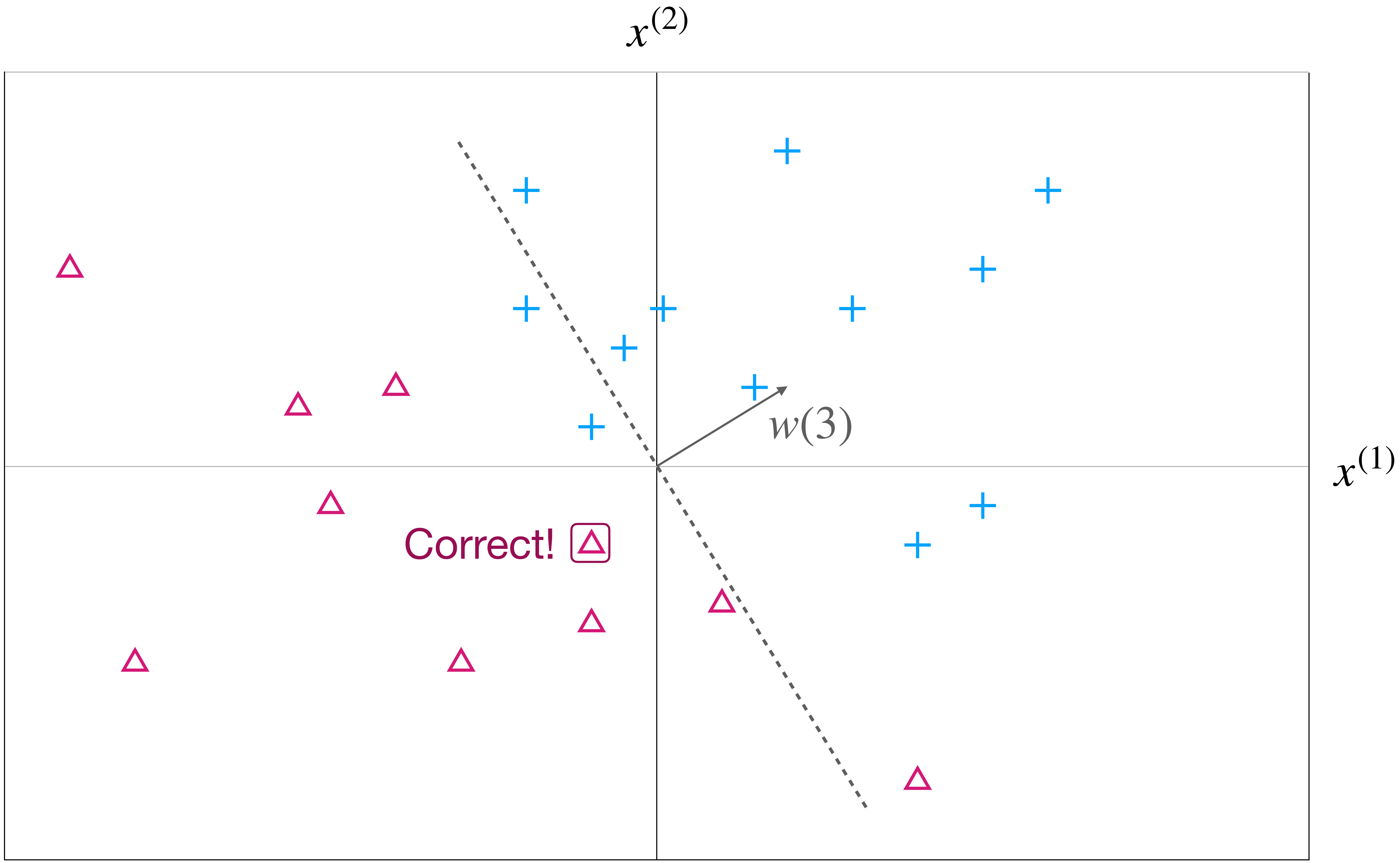
Perceptron learning algorithm

$t = 2$



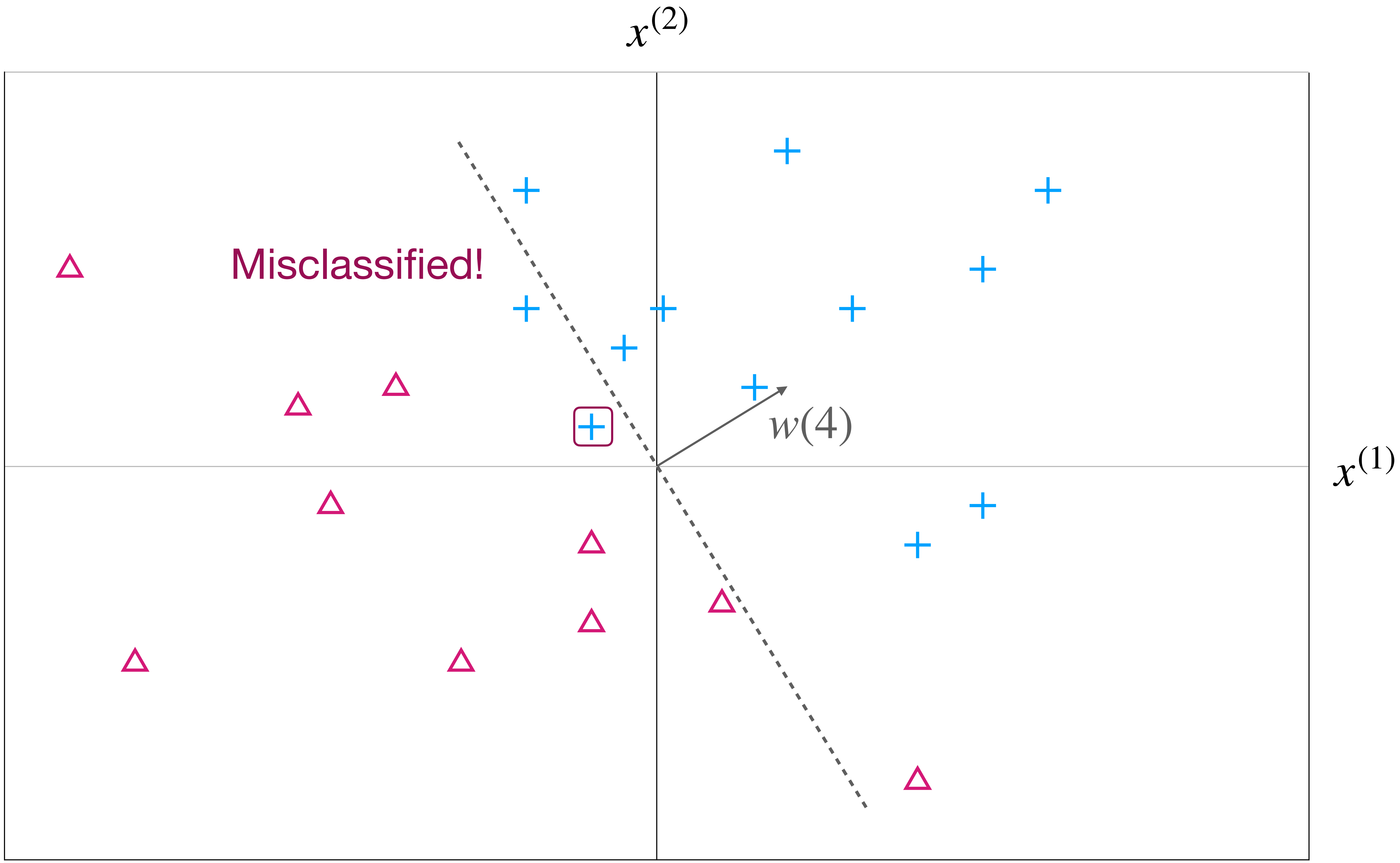
Perceptron learning algorithm

$t = 3$



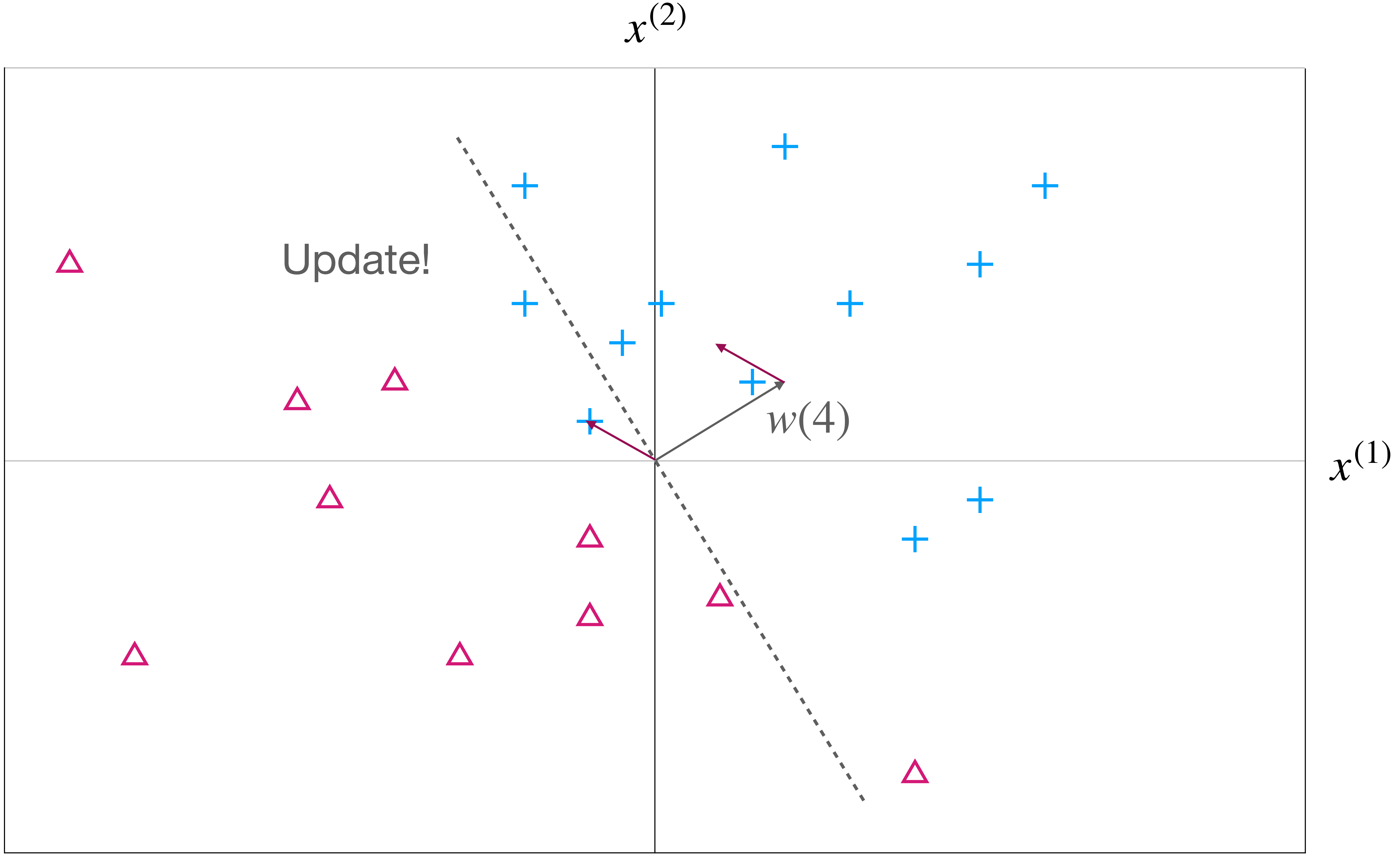
Perceptron learning algorithm

$t = 4$



Perceptron learning algorithm

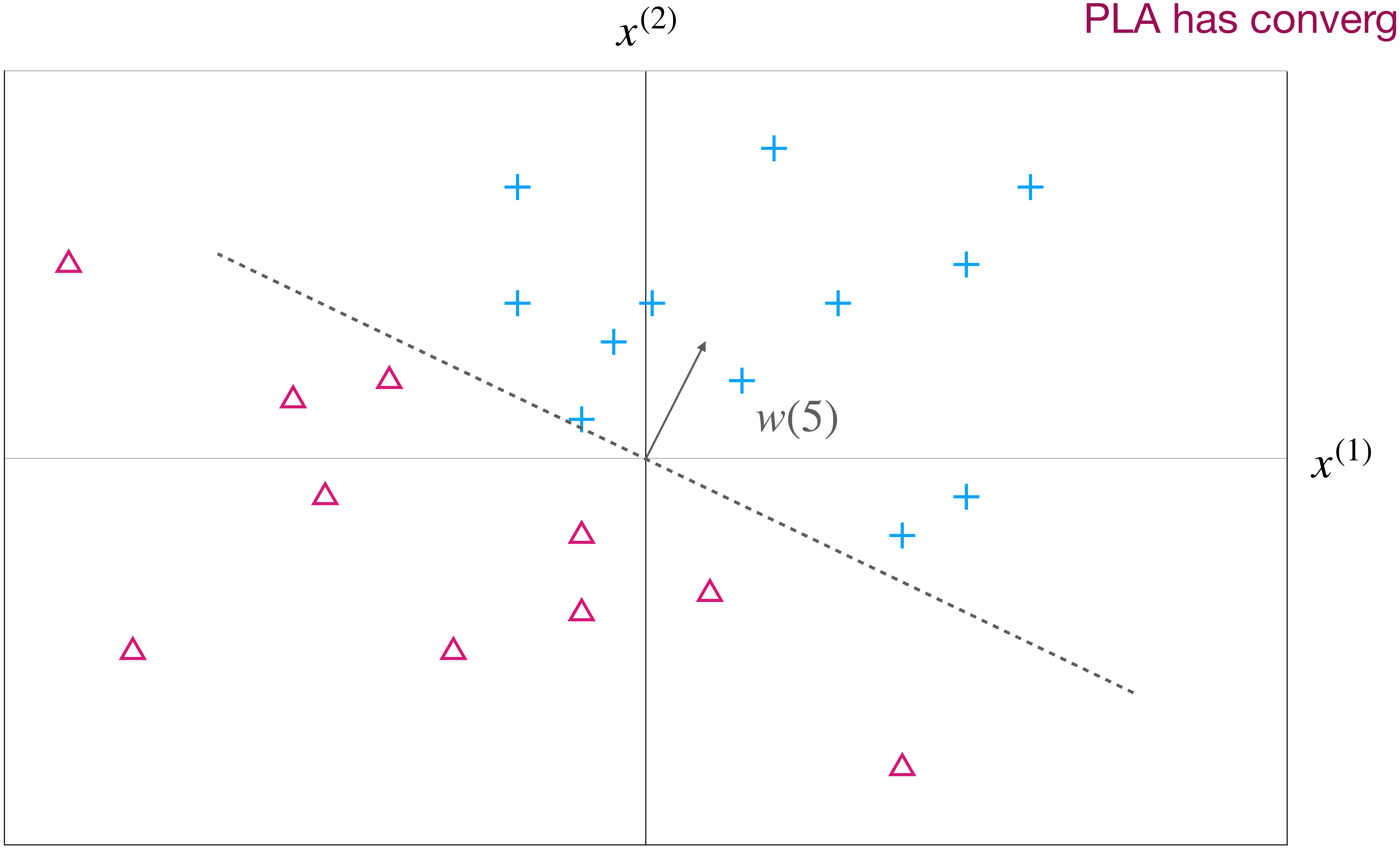
$t = 4$



Perceptron learning algorithm

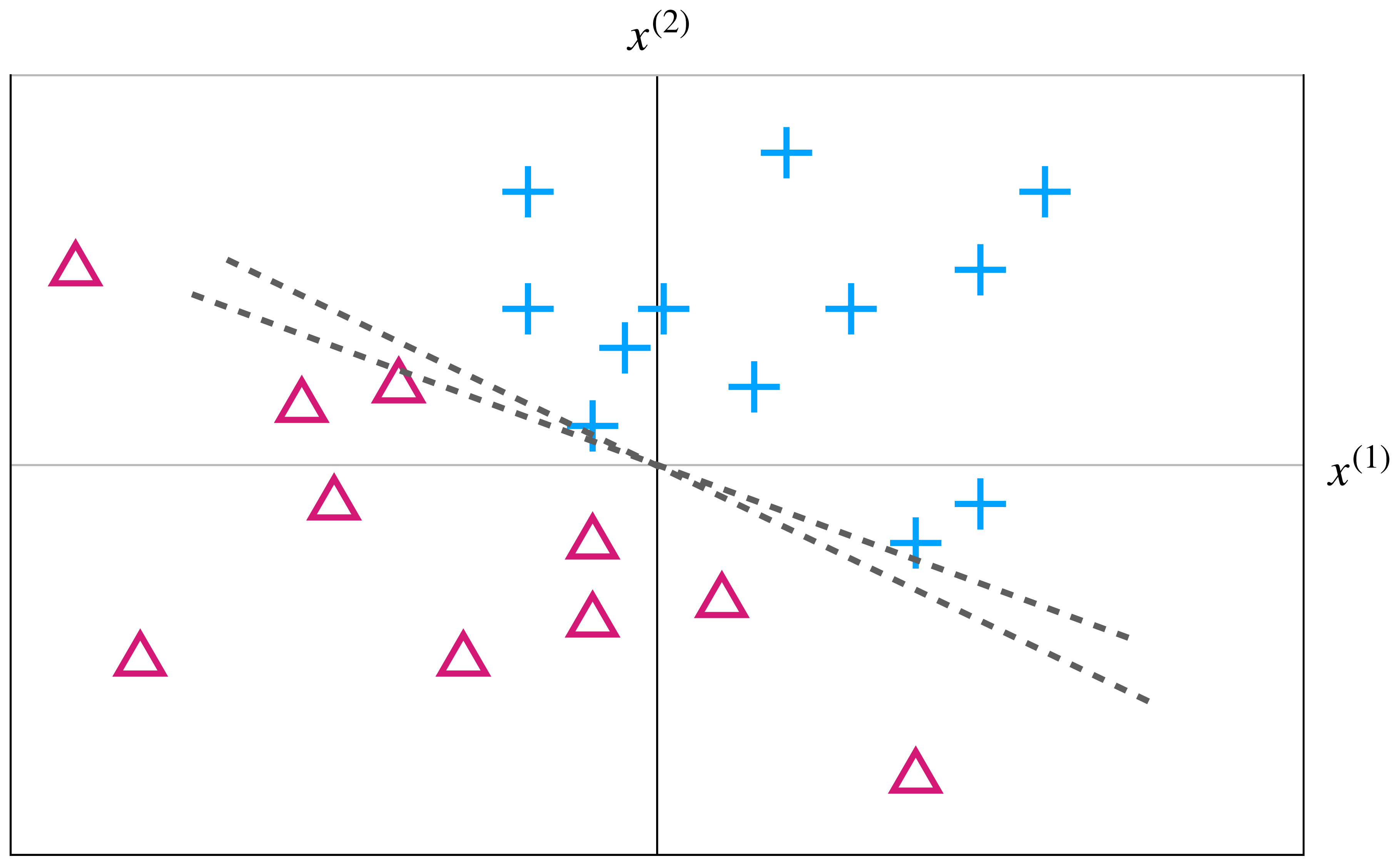
All examples correctly classified!
PLA has converged

$t = 5$



Perceptron learning algorithm

- Converges in a finite number of steps if the training dataset is linearly separable (but may take a long time)
- Does not converge if the dataset is not linearly separable, so *early stopping* is needed
- The final model depends on the initialization and the order you traverse the data points
  - And it may not be the "best" (maximum margin) model
- How can we recast PLA as an example of our supervised learning pipeline?



Two valid solutions: better one has a bigger margin, but PLA stops as soon as it finds any valid solution

Optimizing the learning objective

- In supervised learning, we want to optimize the objective

$$l(w) = \sum_{i=1}^{N} L(y_i, f(x_i \,|\, w))$$

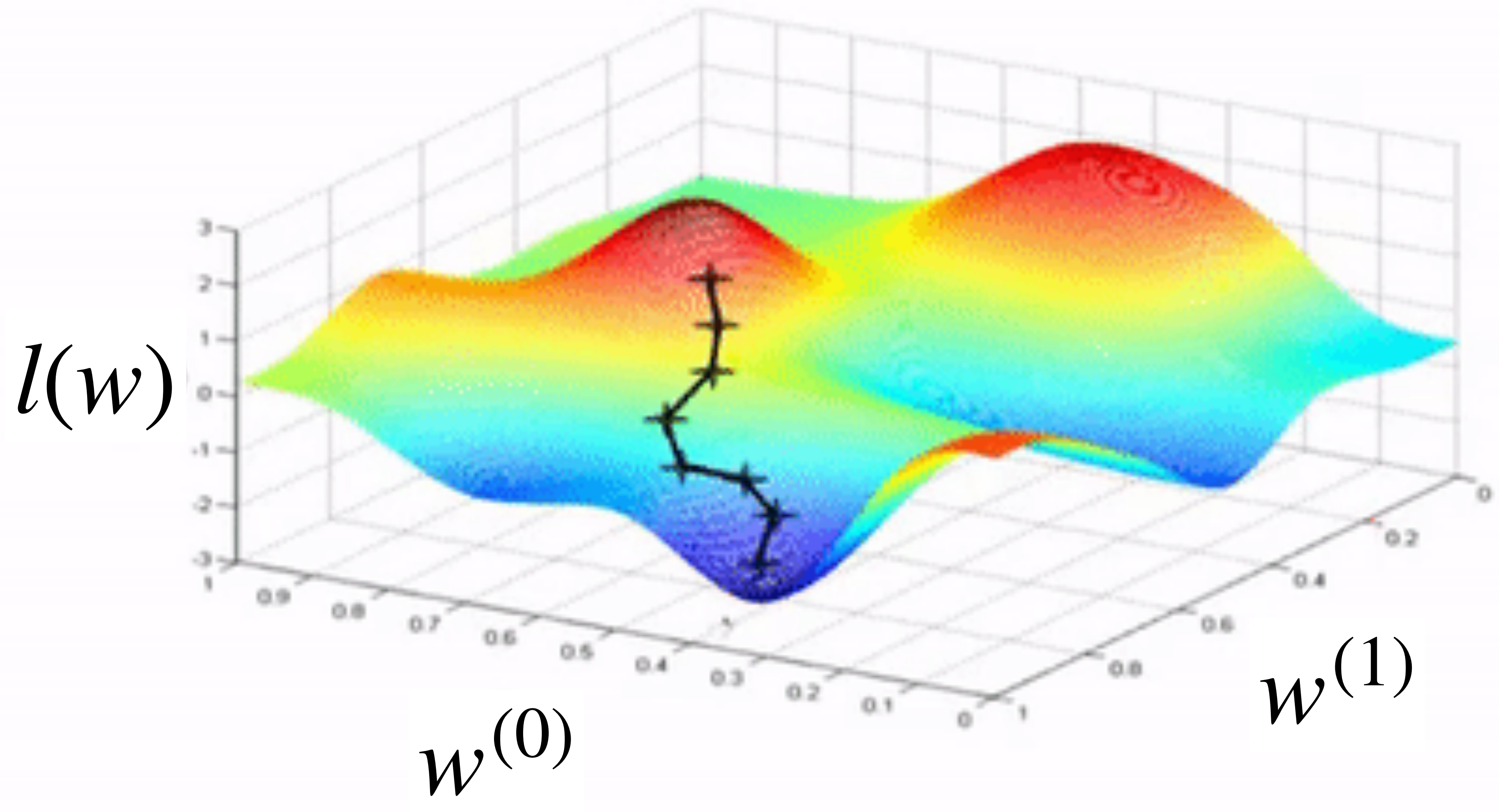
- For linear regression, we had a closed-form solution, but in general?
- We need an optimization algorithm to find the **optimal** (or just **“good”**) w

Gradient descent

- Set $w(t=0)$ to some values (e.g., $w(0)=0$ or some random value)
- At iteration t,
  - Compute the gradient $\nabla_w l(w(t))$: direction of steepest increase of $l(w)$ at $w(t)$
  - Take a small step in the opposite direction:

$$w(t+1) = w(t) - \eta \nabla_w l(w(t))$$

Step size / learning rate



Linear regression with gradient descent

- Learning objective:

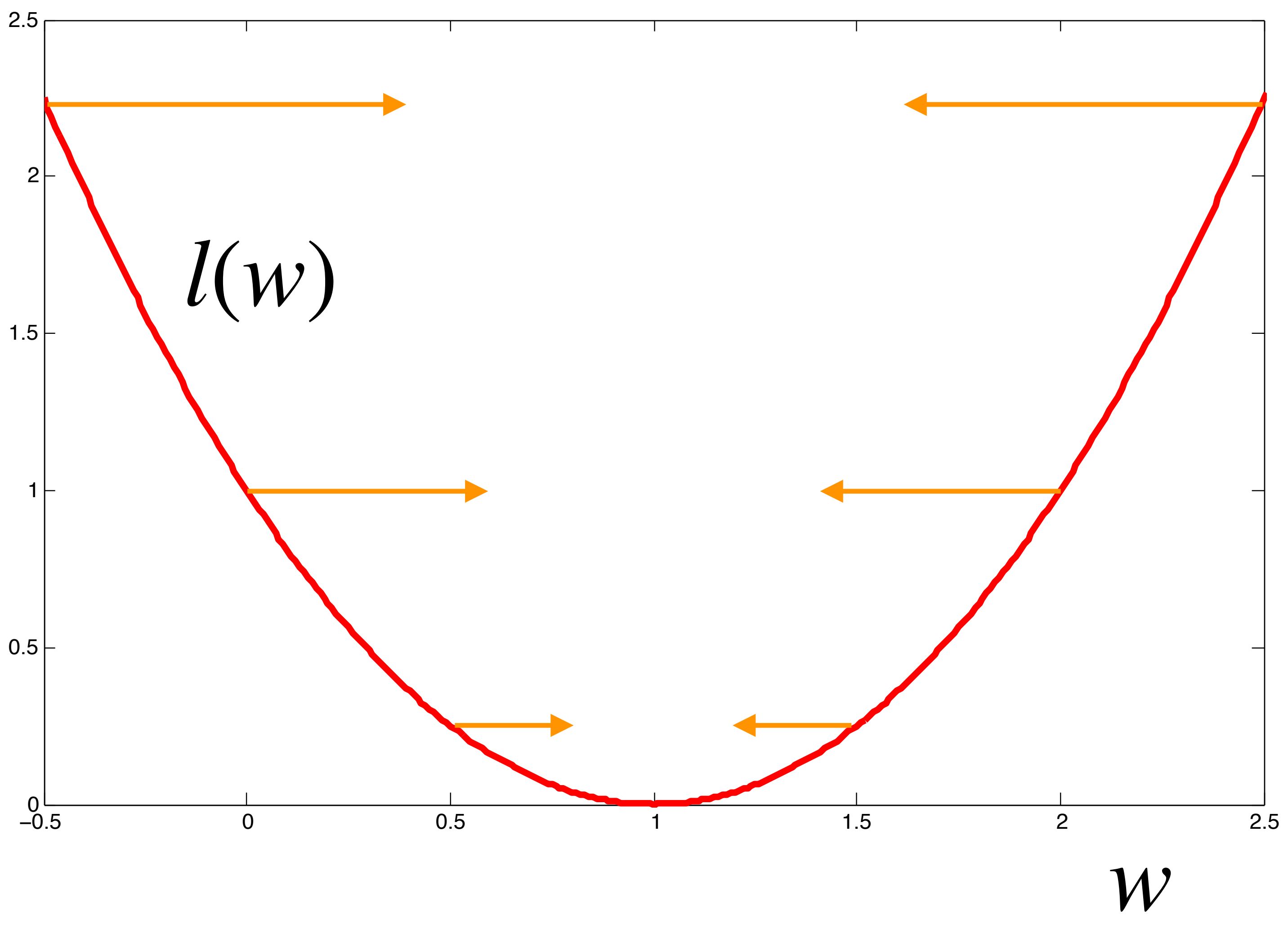
$$l(w) = \sum_{i=1}^{N} (y_i - w^\intercal x_i)^2$$

- Partial derivatives:

$$\frac{\partial}{\partial w^{(j)}} l(w) = \frac{\partial}{\partial w^{(j)}} \sum_{i=1}^{N} (y_i - w^\intercal x_i)^2 = -2 \sum_{i=1}^{N} (y_i - w^\intercal x_i) x_i^{(j)}$$

- Gradient descent update:

$$w(t+1) = w(t) + 2\eta \sum_{i=1}^{N} (y_i - w^\intercal x_i) x_i$$



Limitation of gradient descent

- Requires a full pass over the training dataset at each iteration:

$$w(t+1) = w(t) - \eta \nabla_w \underbrace{\sum_{i=1}^{N} L(y_i, f(x_i, | w))}_{l(w)}$$

- Prohibitively expensive if the training dataset is large!

Stochastic gradient descent

- The learning objective decomposes additively:

$$l(w) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, | w)) = \mathbb{E}_{(x,y)\in S}\left[L(y, f(x | w)\right]$$

add normalization

- The total gradient is the expected gradient of the single-example losses:

$$\nabla_w l(w) = \nabla_w \mathbb{E}_{(x,y)\in S}\left[L(y, f(x | w)\right] = \mathbb{E}_{(x,y)\in S}\left[\nabla_w L(y, f(x | w)\right]$$

- Gradient descent update: $w(t+1) = w(t) - \eta \nabla_w l(w(t))$
- SGD update: $w(t+1) = w(t) - \eta \underbrace{\nabla_w L(y, f(x | w(t)))}$

Unbiased estimate of $\nabla_w l(w(t))$

for a random $(x, y) \in S$

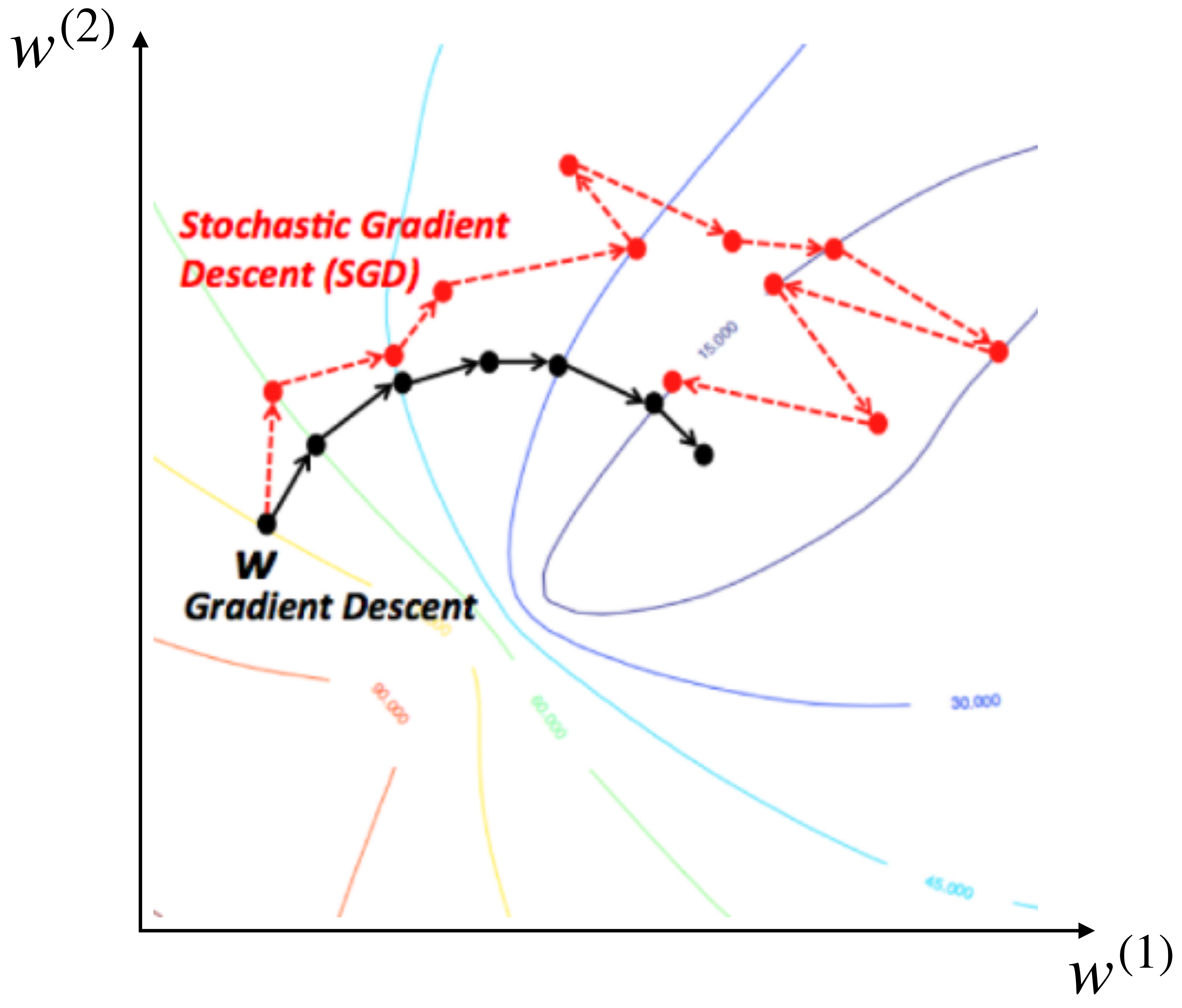
Stochastic gradient descent

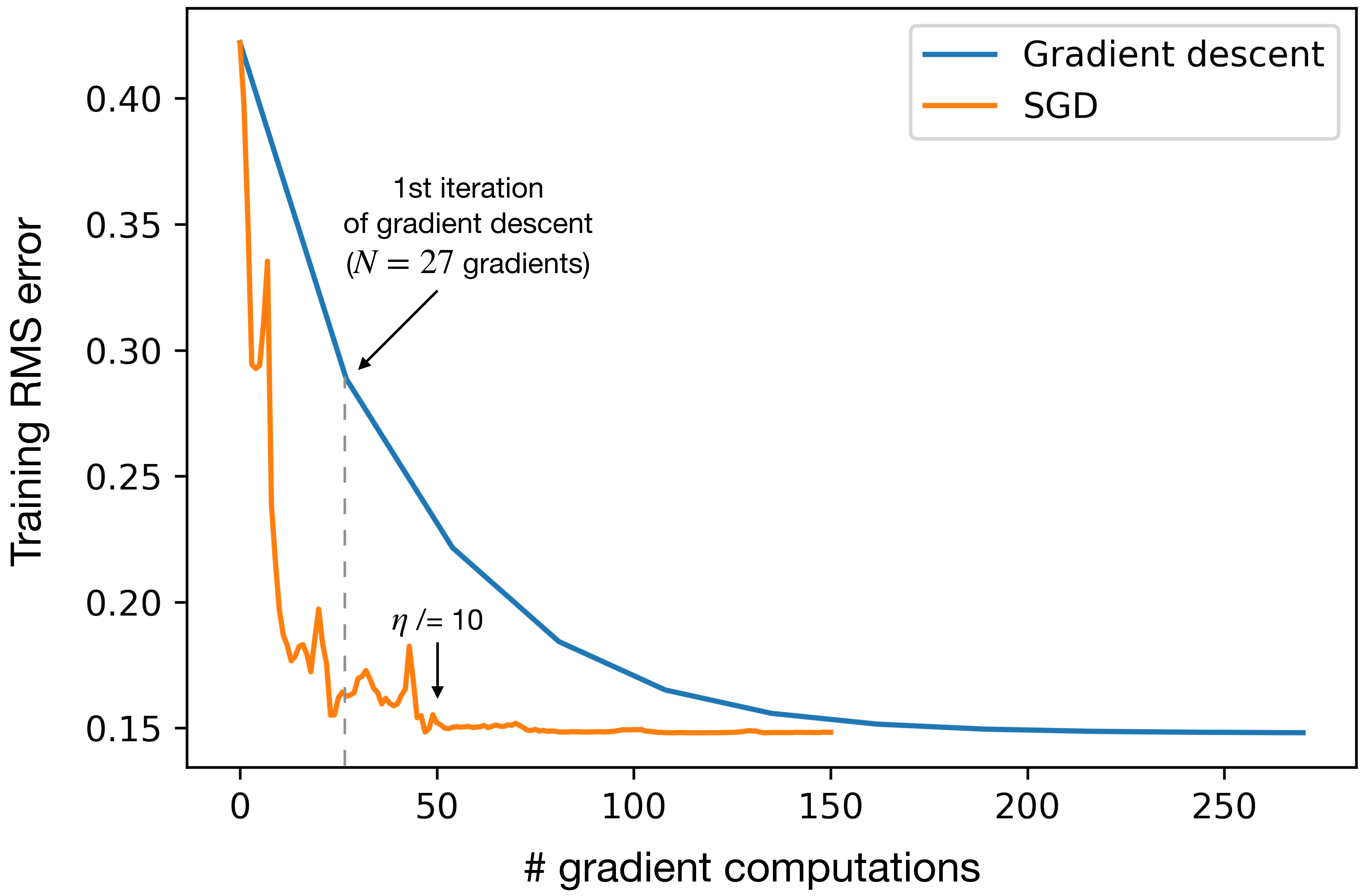
- SGD is an **online** optimization algorithm (only needs one example at a time)
- In practice, we use mini-batch SGD: compute gradient on a mini-batch of training example (e.g., 8, 16, or 32 examples)
    - Leverages fast vector operations (especially on GPU)
    - Decreases volatility of gradient updates (but there is still some **useful** noise)
    - Can be parallelized (e.g. different cores compute gradients on different mini-batches)
    - Useful also for least-squares regression on large datasets
- Note: no need to check validation error at every iteration

Gradient descent vs. SGD

- Gradient descent update: $w(t+1) = w(t) - \eta \nabla_w l(w(t))$

- SGD update: $w(t+1) = w(t) - \eta \nabla_w L(y, f(x \mid w(t)))$
  for a random $(x, y) \in S$





SGD requires less gradient computations

Updated: Supervised learning pipeline

- Training dataset: $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x \in \mathbb{R}^D$ and $y \in \mathbb{R}$

For regression

- Model / hypothesis class: $f(x \mid w) = w^\intercal x$ (linear models)

or $\phi(x)$ instead of x

- Loss function: $L(y, y') = (y - y')^2$ (squared loss)
- Optimization algorithm: SGD
- Cross validation and model selection:
- Testing and deployment

Important: if a testing set is available, never use it to make decisions on the model!

Optimizing the linear classification model

- The most straightforward loss for classification is the 0/1 loss

$$L(y, y') = \delta_{y \neq y'} \text{ where } \delta_{y \neq y'} = \begin{cases} 1 & y \neq y' \\ 0 & \text{otherwise} \end{cases}$$

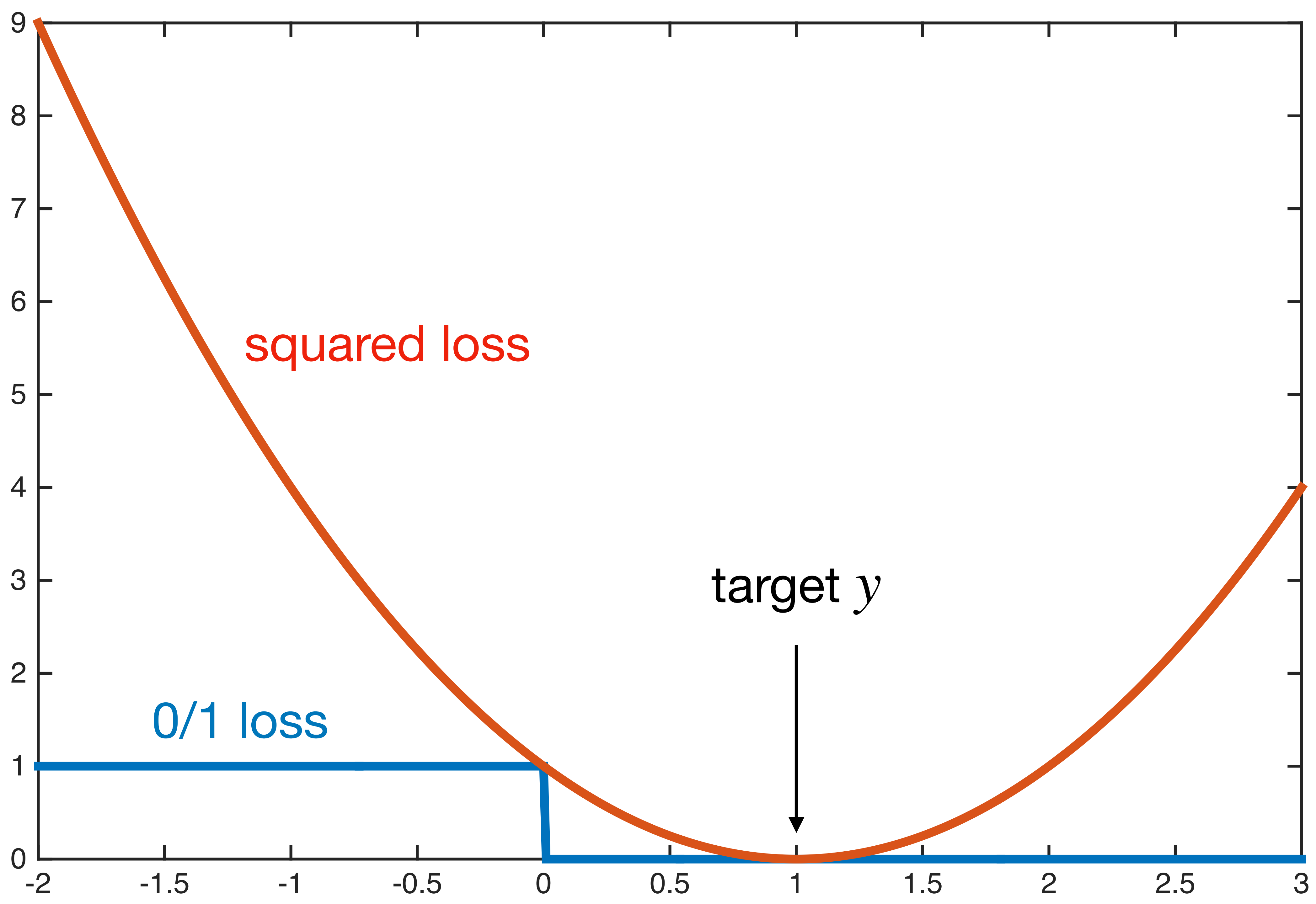
Loss is equal to the number of mistakes

- Good to evaluate the validation/test error, difficult to optimize (gradient is 0)

- We can optimize the raw score $w^\intercal x$ using another loss (e.g., squared loss)

$$\arg\min_w \sum_{i=1}^{N} (y_i - w^\intercal x_i)^2$$

- But the squared loss does not always work well even for a linearly separable dataset

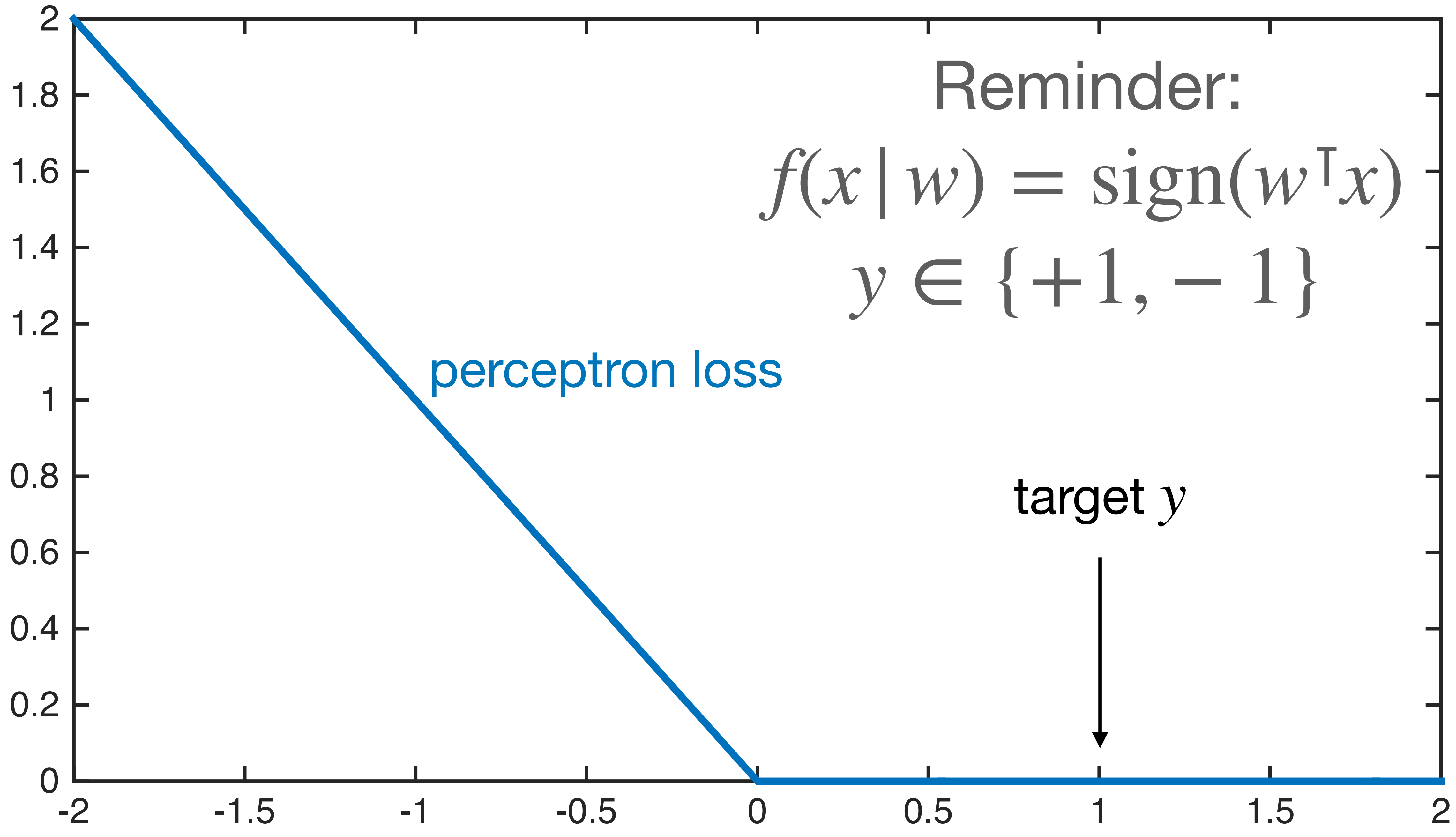


Perceptron algorithm revisited

- An alternative is the perceptron loss

$$L(y, y') = \begin{cases} 0 & \mathrm{sign}(y') = y \\ -yy' & \text{otherwise} \end{cases}$$

(0 if correct, $-y'$ if $y = +1$, $+y'$ if $y = -1$)



- Running SGD with this loss yields the perceptron algorithm

  - Initialize $w(t = 0) = 0$

  - For iteration t, pick a random example (x, y) and perform an update

$$w(t+1) = w(t) - \eta \nabla_w L(y, f(x \mid w(t))) = \begin{cases} w(t) & \text{correct} \\ w(t) + \eta y x & \text{otherwise} \end{cases}$$

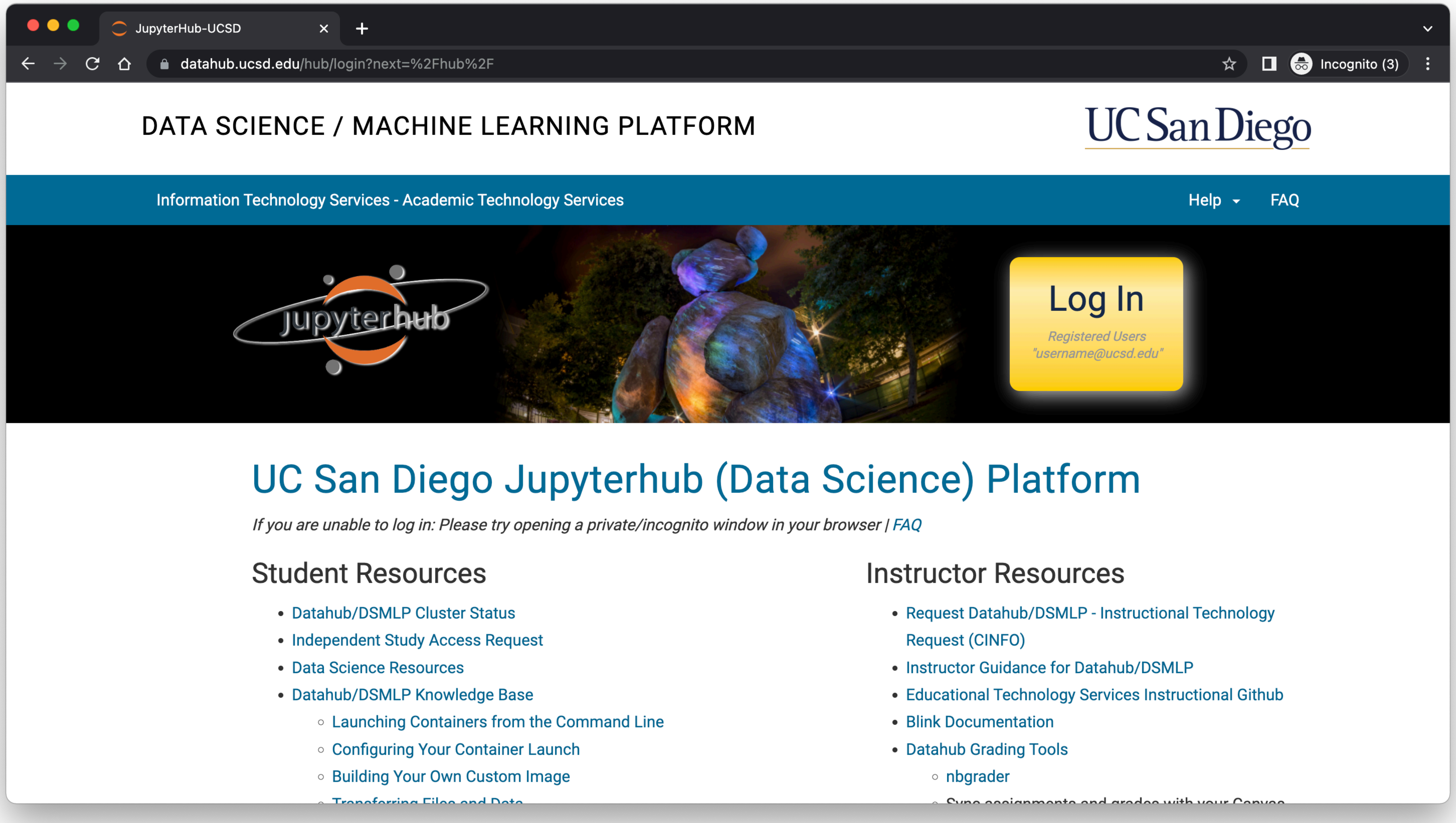
(we can set $\eta = 1$ because the magnitude of w does not matter)

Next time

- More on (stochastic) gradient descent
- Different loss functions
  - Hinge loss (support vector machine)
  - Log loss / cross entropy loss (logistic regression)

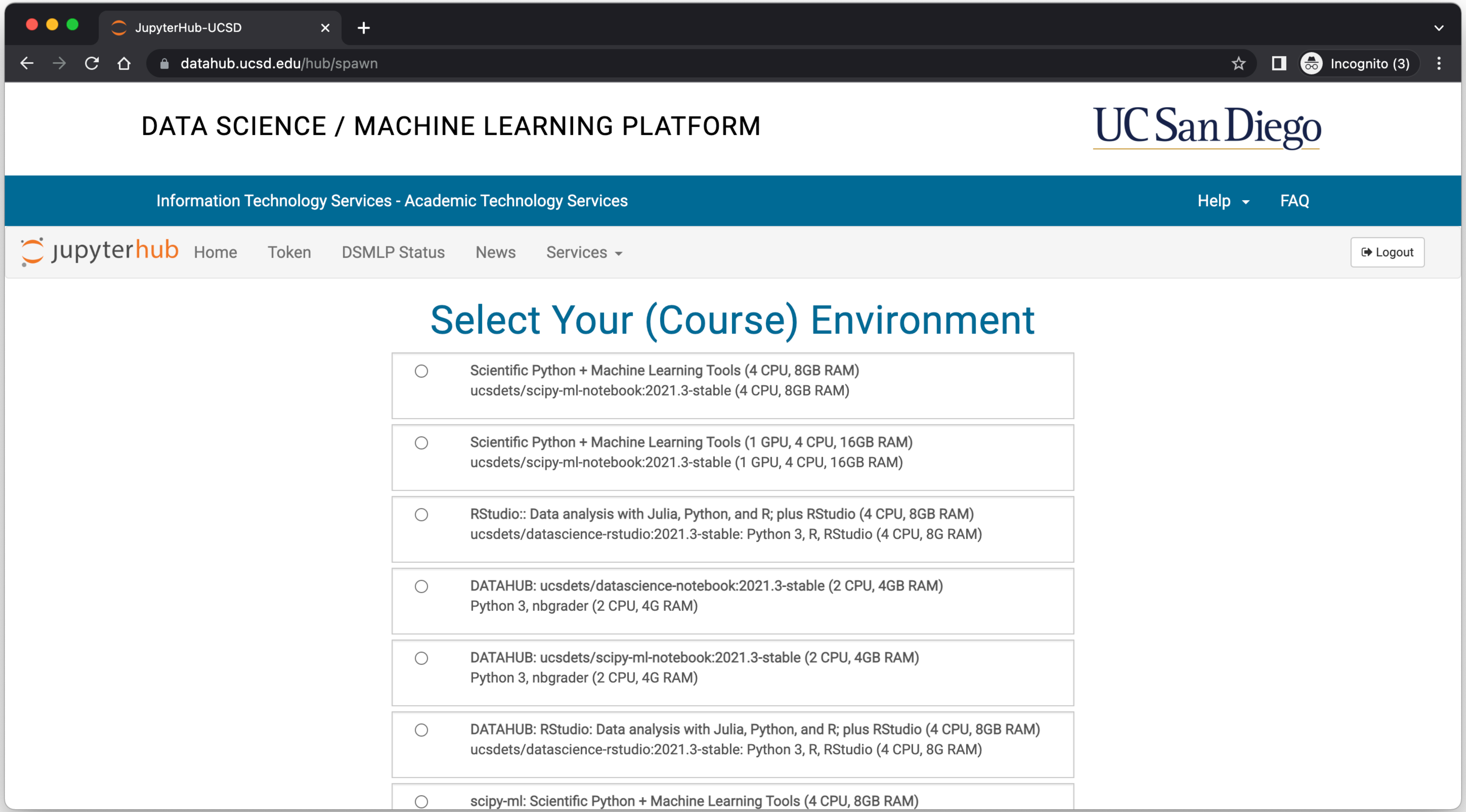
DataHub

- We will use DataHub for in-class hands-on portions
  - Recommend to use it for homework, final project, etc.
- Address: datahub.ucsd.edu
- Similar to public, free services Google Colab, but with access to better CPUs and GPUs and run by UCSD
- Provides a “Jupyter notebook” interface (Python-based but interactive coding like MATLAB/Mathematica)



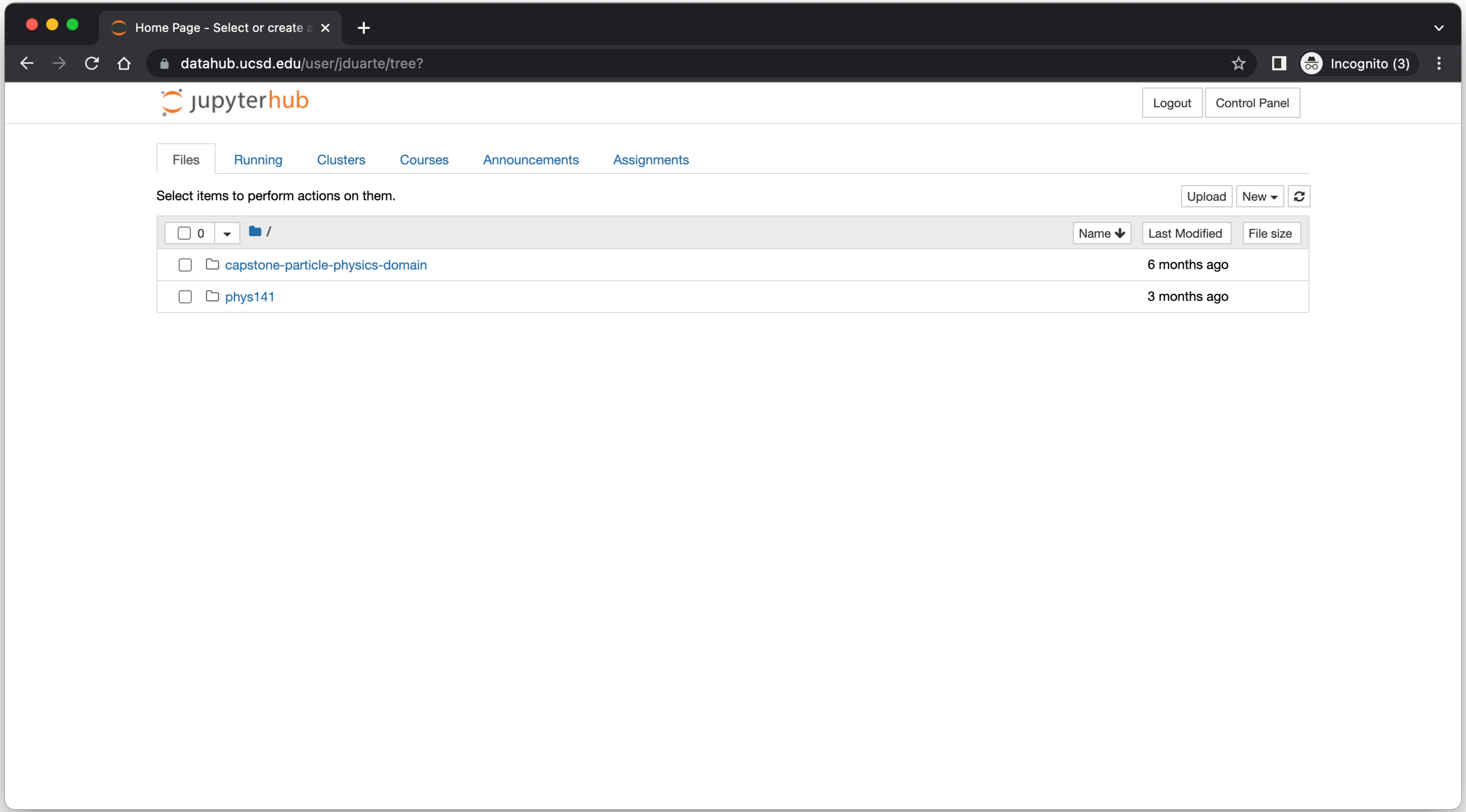
Logging in

- After logging in, choose a course environment…
  PHYS 139 PHYS 239 - Special Topics - Machine Learning - Duarte [WI23], ucsdets/scipy-ml-notebook:2022.3-stable, (2 CPU, 4G RAM)



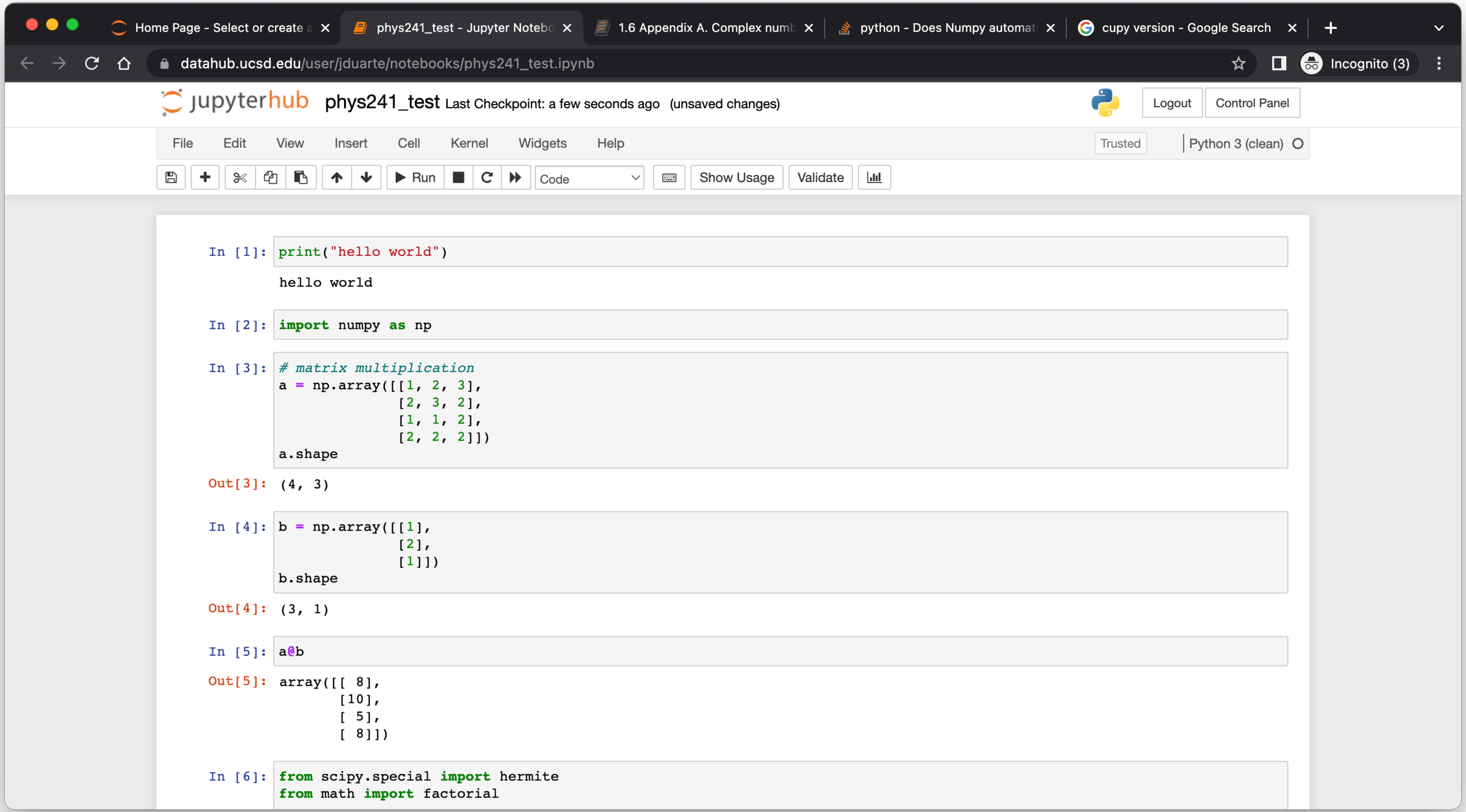
Jupyter interface

- Spawns a “JupyterHub” (let’s go step by step through all the buttons)



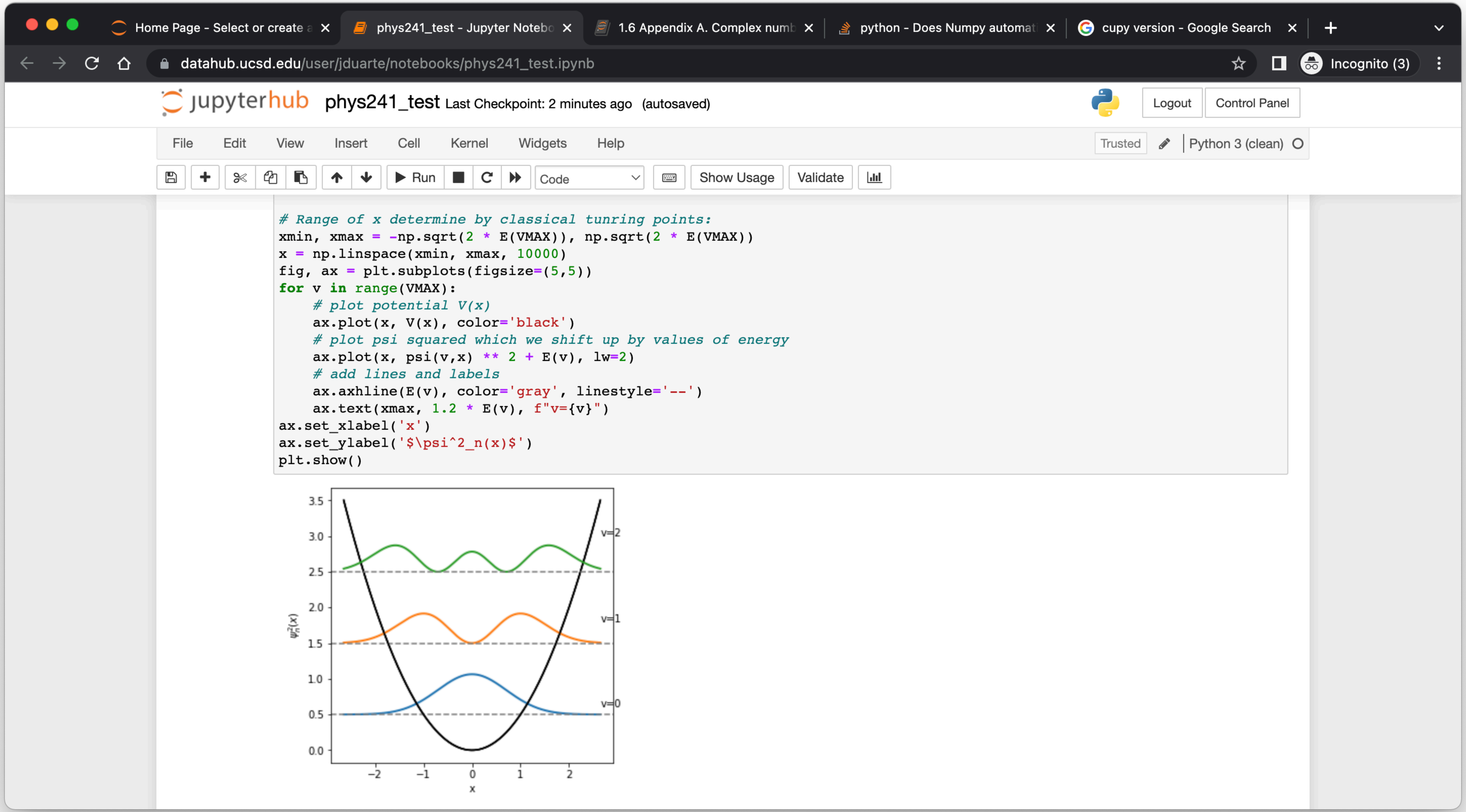
Start coding!

- Coding in Jupyter notebooks



Plotting

- Plotting can be done easily with Matplotlib



Installing a missing library

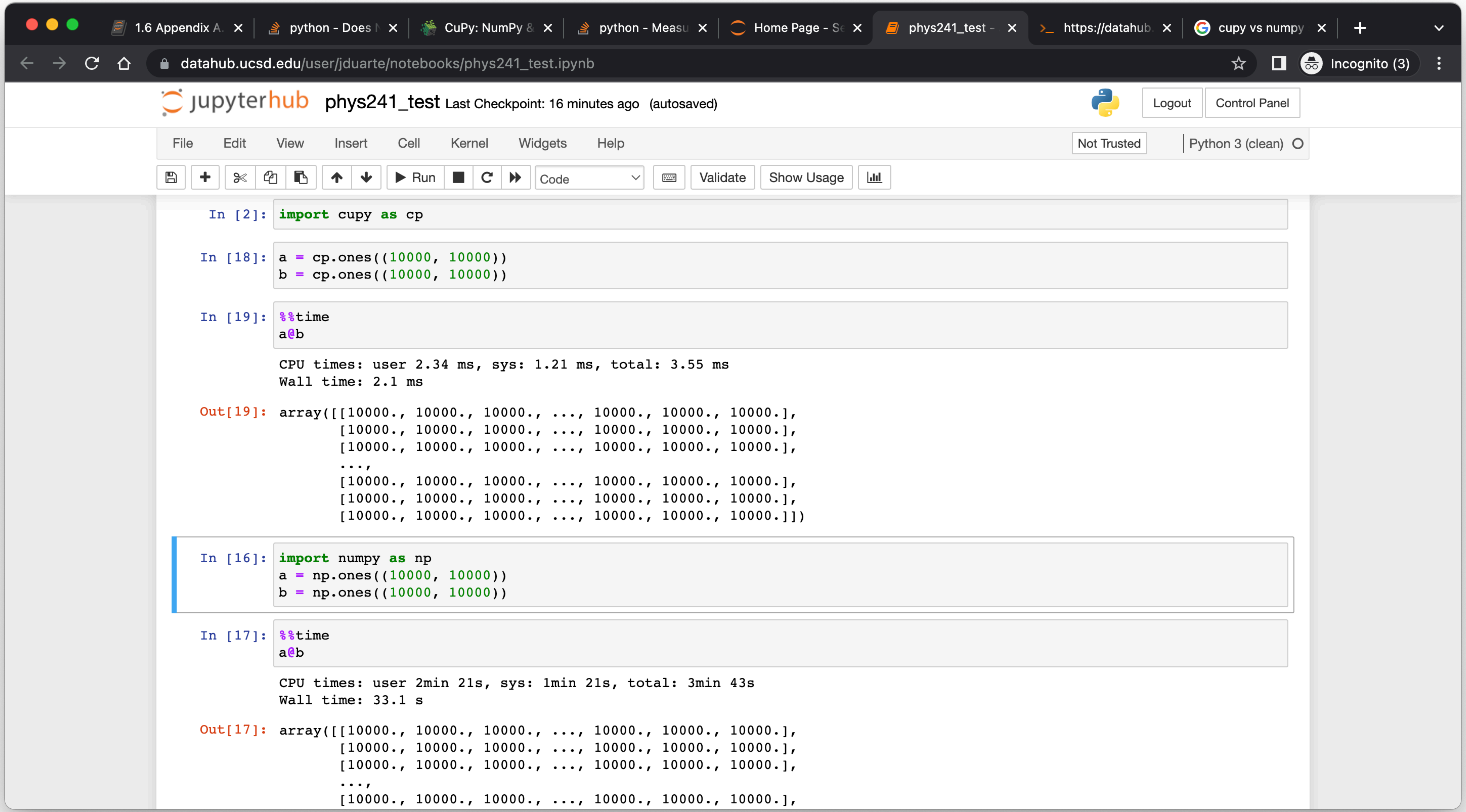
- Not all libraries are preloaded, but it’s easy to install a new one
- Note: restart your “kernel” after doing this

```
In [1]: !pip install cupy-cuda112 --user

Collecting cupy-cuda112
  Using cached cupy_cuda112-10.3.1-cp39-cp39-manylinux1_x86_64.whl (78.9 MB)
Collecting fastrlock>=0.5
  Using cached fastrlock-0.8-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86
_64.whl (49 kB)
Requirement already satisfied: numpy<1.25,>=1.18 in /opt/conda/lib/python3.9/site-packages (from cupy-cuda112) (1.19.
5)
Installing collected packages: fastrlock, cupy-cuda112
Successfully installed cupy-cuda112-10.3.1 fastrlock-0.8
```

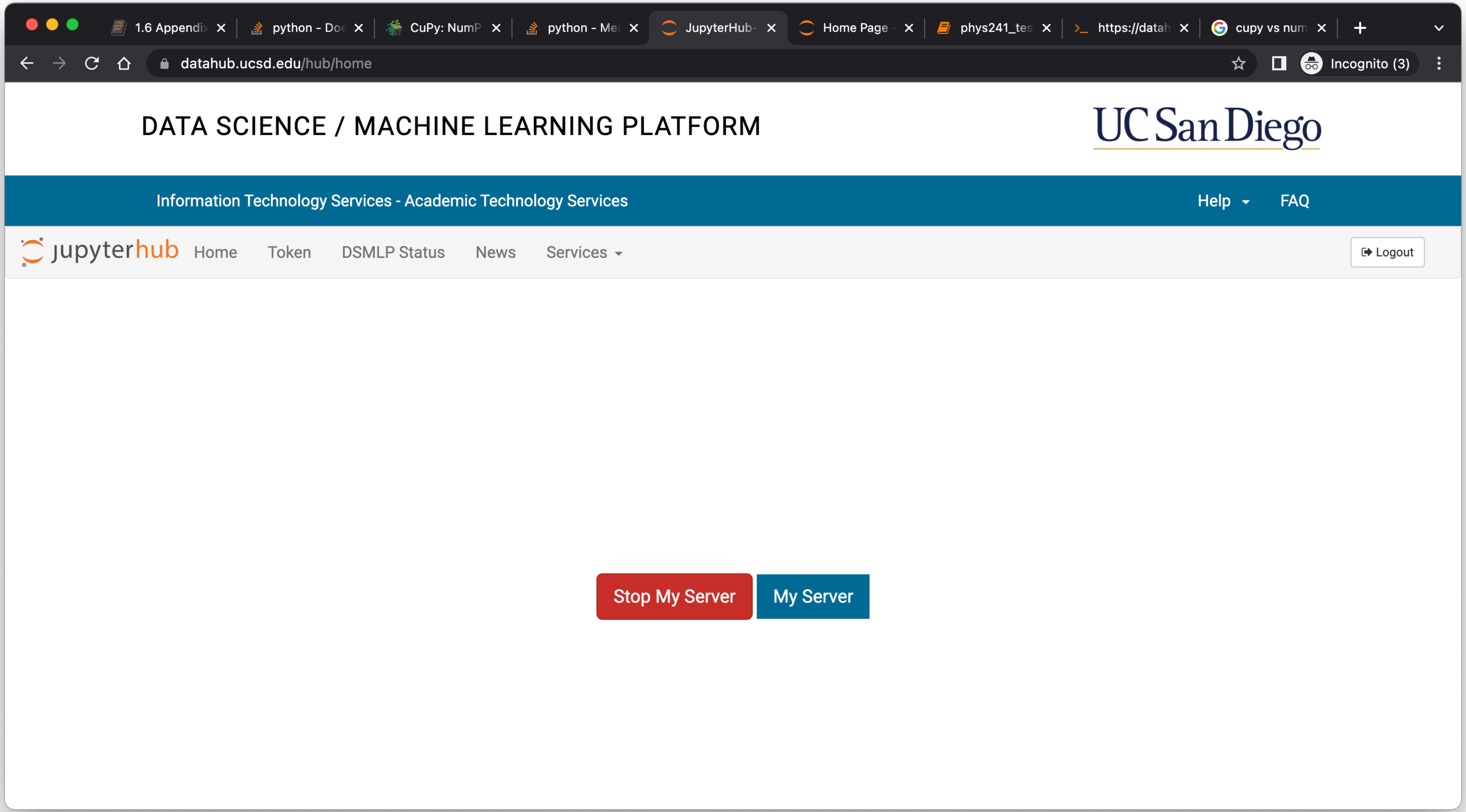
Speed up numerical calculations with the GPU

- Many libraries to do this: TensorFlow, CuPy, PyTorch, …



Exiting / shutting down server

- When you’re done; hit control panel and “stop your server”
  When you start it again, all your data will still be there



Data Science & Machine Learning Platform (DSMLP)

- UCSD Data Science & Machine Learning Platform (DSMLP)
  - Based on docker and **Kubernetes** (K8s): open-source system for automating deployment, scaling, and management of containerized applications
- Overview:
  - You log in to a remote "login” node
  - From that login node, you can launch a “pod” running a container
    - The pod automatically starts a “JupyterHub” web server (only accessible from campus so you must VPN if off campus)
    - You are also automatically logged into that pod so you can run interactive terminal commands, etc.

More Resources

- UCSD DSMLP Cluster
  - Using the DSMLP server
    - Customizing the DSMLP Containers
- Terminal and Command-Line Interface
  - Bash Scripting Reference
  - Git(Hub) Resources