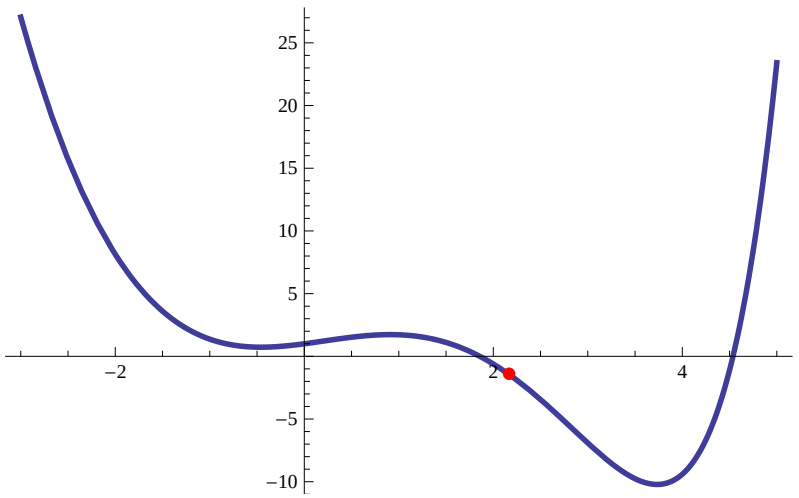
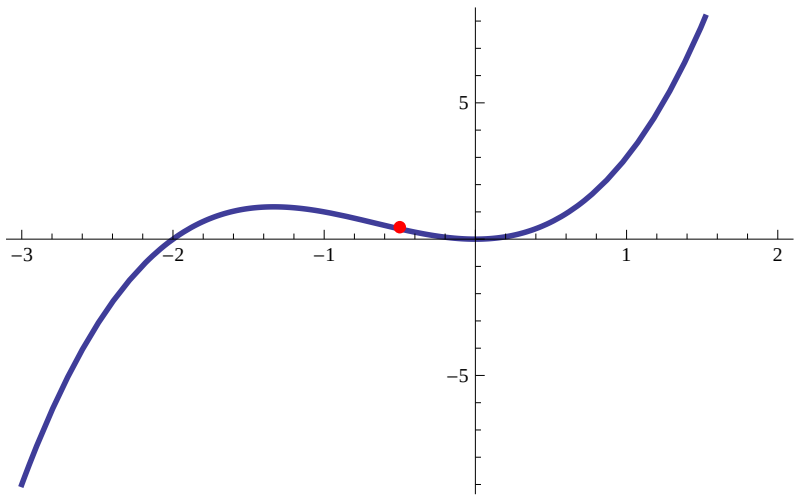
Gradient Descent and Local Minima





- Suppose for both functions above, gradient descent is started at the point marked red.
- It will "walk downhill" as far as possible, then terminate.
- For the function on the left, the minimum it finds is global. For the function on the right, it is only a local minimum.
- Since the derivative at both minima is 0, gradient descent cannot detect whether they are global or local.

For smooth functions, gradient descent finds *local* minima. If the function is complicated, there may be no way to tell whether the solution is also a global minimum.

Summary so far

- The derivative/gradient provides local information about how a function changes around a point x.
- Optimization algorithms: If we know the gradient at our current location x, we can use this information to make a step in "downhill" direction, and move closer to a (local) minimum.

What we do not know yet

That assumes that we can compute the gradient. There are two possibilities:

- For some functions, we are able to derive $\nabla f(x)$ as a function of x. Gradient descent can evaluate the gradient by evaluating that function.
- Otherwise, we have to estimate $\nabla f(x)$ by evaluating the function f at points close to x.

For now, we will assume that we can compute the gradient as a function.

Given

- Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.

The data analyst chooses

- A class $\mathcal{H}$ of possible solutions (e.g. $\mathcal{H} =$ all linear classifier).
- A loss function L (e.g. 0-1 loss) that measures how well the solution represents the data.

Learning from training data

- Define the empirical risk of a solution $f \in \mathcal{H}$ on the training data,

$$\hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} L(f(\tilde{\mathbf{x}}_i), \tilde{y}_i)$$

- Find an optimal solution f^* by minimizing the empirical risk using gradient descent (or another optimization algorithm).

THE PERCEPTRON ALGORITHM

Task: Train a Linear Classifier

We want to determine a linear classifier in $\mathbb{R}^d$ using 0-1 loss.

Recall

- A solution is determined by a normal $\mathbf{v}_{\mathrm{H}} \in \mathbb{R}^d$ and an offset $c > 0$.
- For training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$, the empirical risk is

$$\hat{R}_n(\mathbf{v}_{\mathrm{H}}, c) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{\mathrm{sgn}(\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x}_i \rangle - c) \neq y_i\}$$

- The empirical risk minimization approach would choose a classifer given by $(\mathbf{v}_{\mathrm{H}}^*, c^*)$ for which

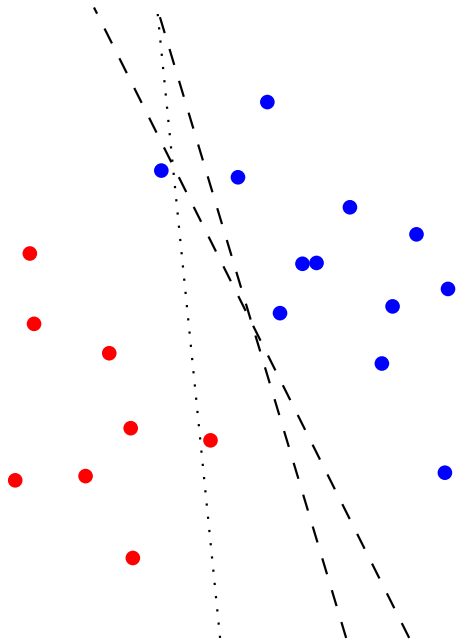
$$(\mathbf{v}_{\mathrm{H}}^*, c^*) = \arg\min_{\mathbf{v}_{\mathrm{H}}, c} \hat{R}_n(\mathbf{v}_{\mathrm{H}}, c)$$

Idea

Can we use gradient descent to find the minimizer of $\hat{R}_n$?

Example

- In the example on the right, the two dashed classifiers both get a single (blue) training point wrong.
- Both of them have different values of $(\mathbf{v}_H, c)$, but for both of these values, the empirical risk is identical.
- Suppose we shift one of the to dashed lines to obtain the dotted line. On the way, the line moves over a single red point. The moment it passes that point, the empirical risk jumps from $\frac{1}{n}$ to $\frac{2}{n}$.

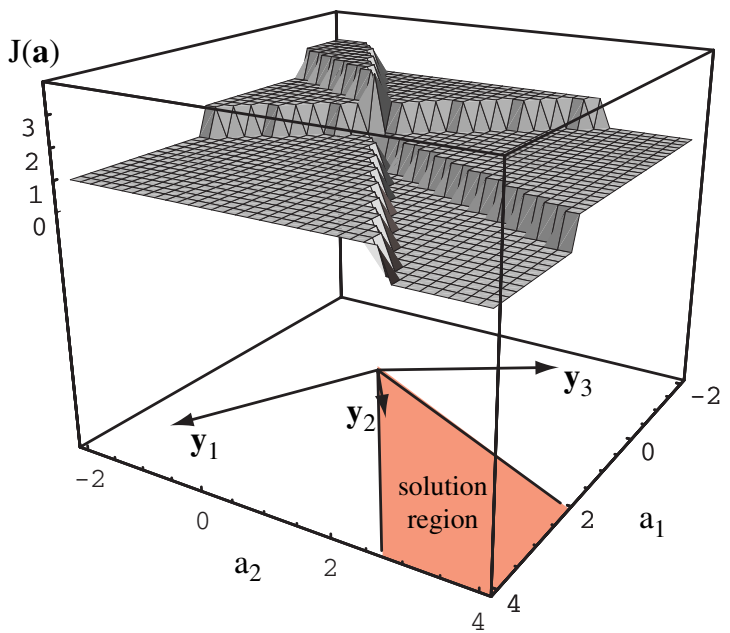


Conclusion

Consider the empirical risk function $\hat{R}_n(\mathbf{v}_H, c)$:

- If $(\mathbf{v}_H, c)$ defines an affine plain that contains one of the training points, $\hat{R}_n$ is discontinuous at $(\mathbf{v}_H, c)$ (it "jumps"). That means it is not differentiable.
- At all other points $(\mathbf{v}_H, c)$, the function is constant. It is differentiable, but the length of the gradient is 0.

The empirical risk of a linear classifer under 0-1 loss is piece-wise constant.



Formal problem

Even if we can avoid points where $\hat{R}_n$ jumps, the gradient is always 0. Gradient descent never moves anywhere.

Intuition

- Remember that we can only evaluate local information about $\hat{R}_n$ around a given point $(\mathbf{v}_\text{H}, c)$.
- In every direction around $(\mathbf{v}_\text{H}, c)$, the function looks identical.
- The algorithm cannot tell what a good direction to move in would be.
- Note that is also the case for every other optimization algorithm, since optimization algorithms depend on local information.

Solution idea

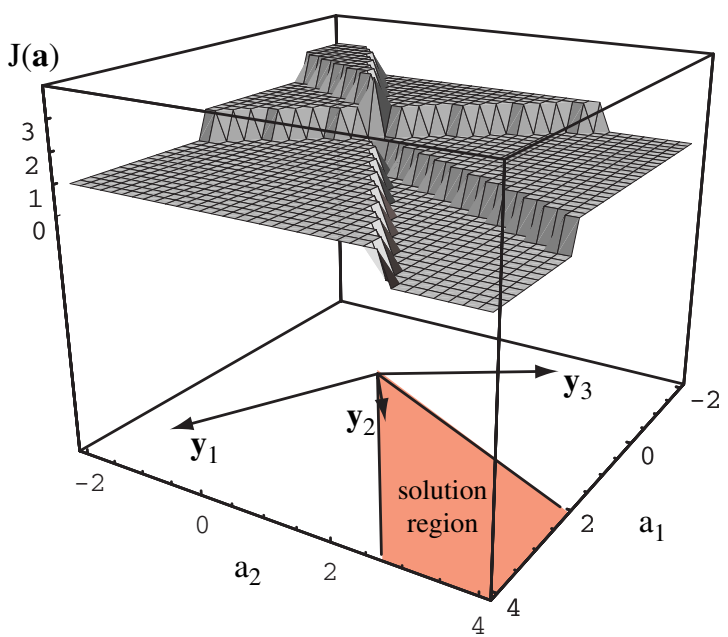
Find an approximation to $\hat{R}_n$ that is not piece-wise constant, and decreases in direction of an optimal solution. We try to keep the approximation as simple as possible.

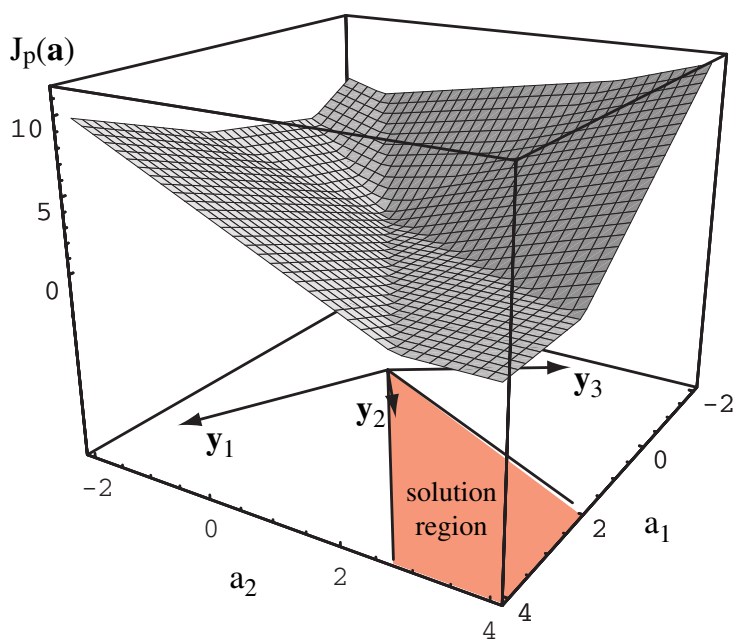
Perceptron Cost Function

We replace the empirical risk $\hat{R}_n(\mathbf{v}_H, c) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{\text{sgn}(\langle \mathbf{v}_H, \mathbf{x}_i\rangle) - c) \neq y_i\}$ by the piece-wise linear function

$$\hat{S}_n(\mathbf{v}_H, c) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathbb{I}\{\text{sgn}(\langle \mathbf{v}_H, \mathbf{x}_i\rangle) - c) \neq y_i\}}_{\text{“switches off” correctly classified points}} \cdot \overbrace{|\langle \mathbf{v}_H, \mathbf{x}\rangle - c|}^{\text{measures distance to plane}}$$

$\hat{S}_n$ is called the **perceptron cost function**.





Training

- **Given**: Training data $(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)$.
- **Training**: Fix a precision ε and a learning rate α, and run gradient descent on the perceptron cost function to find an affine plane $(\mathbf{v}_{\mathrm{H}}^*, c^*)$
- Define a classifier as $f(\mathbf{x}) := \operatorname{sgn}(\langle \mathbf{v}_{\mathrm{H}}, \mathbf{x} \rangle - c)$.

(If the gradient algorithm returns $c < 0$, flip signs: Use $(-\mathbf{v}_{\mathrm{H}}^*, -c^*)$.)

Prediction

- For a given data point $\mathbf{x} \in \mathbb{R}^d$, predict the class label $y := f(\mathbf{x})$.

This classifier is called the **perceptron**. It was first proposed by Rosenblatt in 1962.

The Perceptron Gradient

One can show that the gradient of the cost function is

$$\nabla \hat{S}_n(\mathbf{v}_\text{H}, c) = -\sum_{i=1}^{n} \mathbb{I}\{f(\tilde{\mathbf{x}}_i) \neq \tilde{y}_i\} \cdot \tilde{y}_i \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix} \ .$$

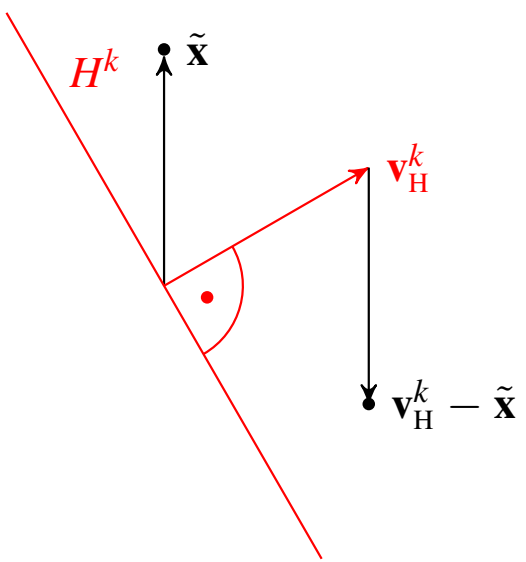
This is an example of gradient descent where we do not have to approximate the derivative numerically. Gradient descent steps then look like this:

$$\begin{pmatrix} \mathbf{v}_\text{H}^{(k+1)} \\ c^{(k+1)} \end{pmatrix} := \begin{pmatrix} \mathbf{v}_\text{H}^{(k)} \\ c^{(k)} \end{pmatrix} + \sum_{i|\tilde{\mathbf{x}}_i \text{ misclassified}} \tilde{y}_i \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix}$$

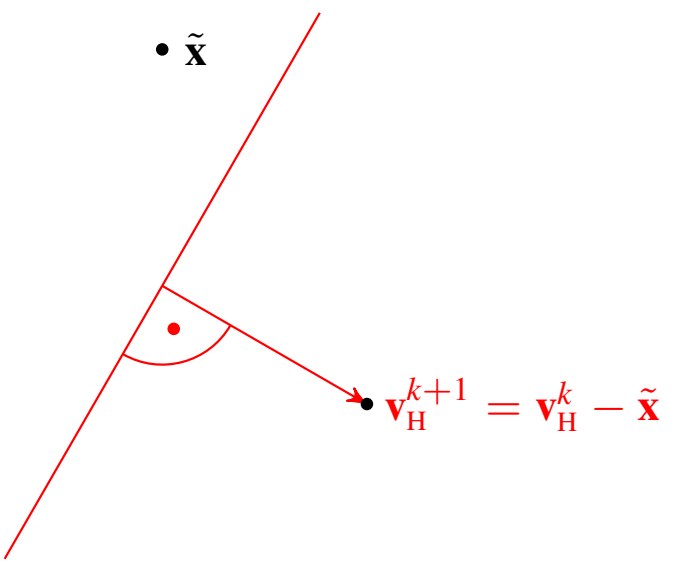
$$\begin{pmatrix} \mathbf{v}_{\mathrm{H}}^{(k+1)} \\ c^{(k+1)} \end{pmatrix} := \begin{pmatrix} \mathbf{v}_{\mathrm{H}}^{(k)} \\ c^{(k)} \end{pmatrix} + \sum_{i|\tilde{\mathbf{x}}_i \text{ misclassified}} \tilde{y}_i \begin{pmatrix} \tilde{\mathbf{x}}_i \\ 1 \end{pmatrix}$$

Effect for a single training point

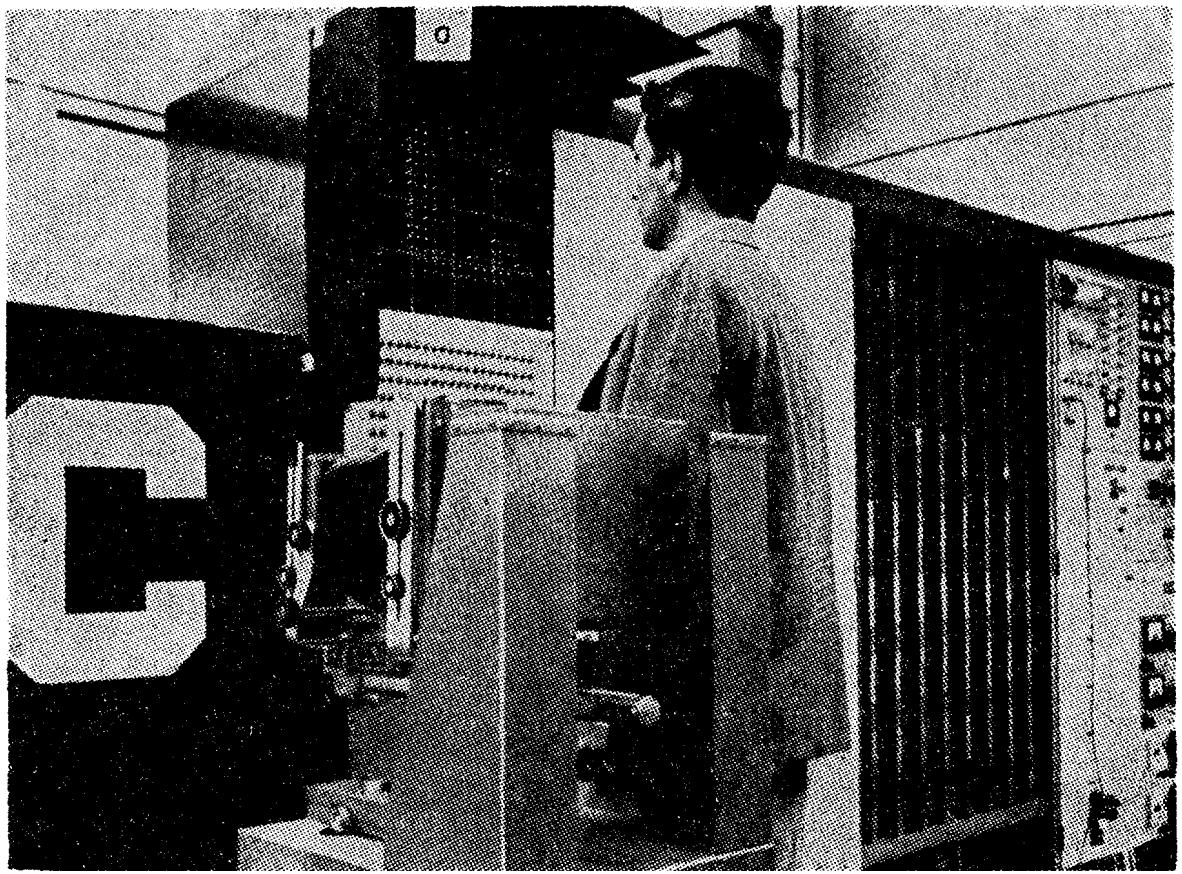
Step k: $\tilde{\mathbf{x}}$ (in class -1) classified incorrectly



Step $k+1$



Simplifying assumption: H contains origin, so $c = 0$.



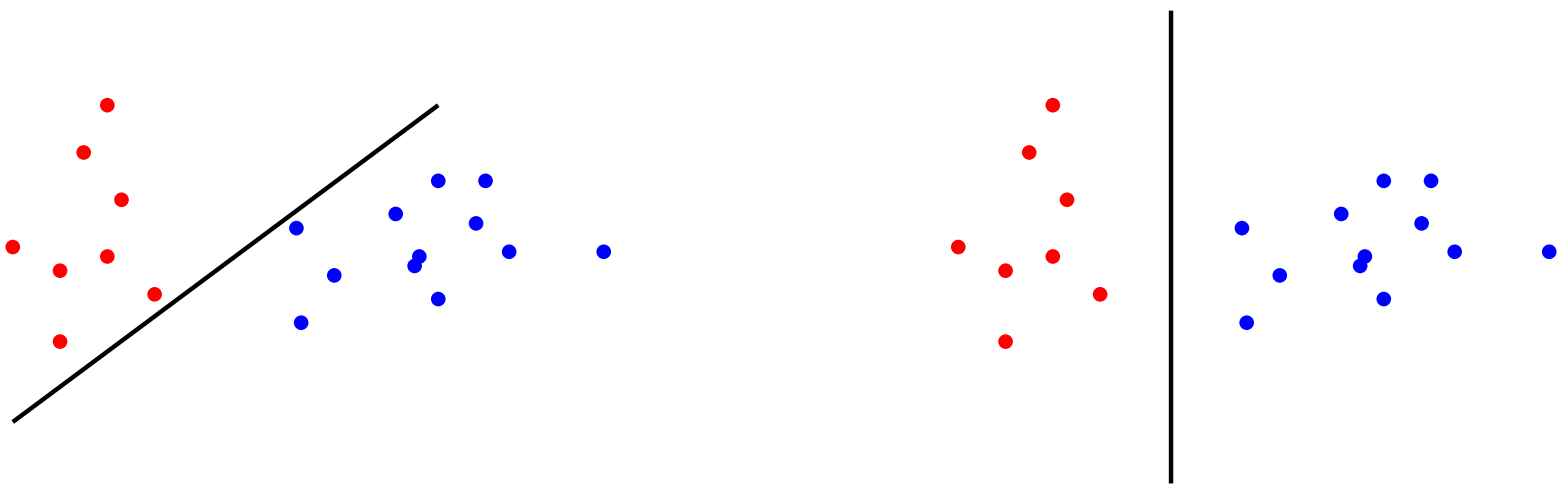
Does the Perceptron Work?

Theorem: Perceptron convergence

If the training data is linearly separable, the Perceptron learning algorithm (with fixed step size) terminates after a finite number of steps with a valid solution $(\mathbf{v}_H, c)$ (i.e. a solution which classifies all training data points correctly).

Issues

The perceptron selects *some* hyperplane between the two classes. The choice depends on initialization, step size etc.



The solution on the right will probably predict better than the one on the left, but the perceptron may return either.