Deep learning

3.1. The perceptron

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

The first mathematical model for a neuron was the Threshold Logic Unit, with Boolean inputs and outputs:

$$f(x) = \mathbf{1}_{\{w \sum_i x_i + b \geq 0\}}.$$

It can in particular implement

$$\begin{aligned} or(u, v) &= \mathbf{1}_{\{u+v-0.5 \geq 0\}} && (w = 1, b = -0.5) \\ and(u, v) &= \mathbf{1}_{\{u+v-1.5 \geq 0\}} && (w = 1, b = -1.5) \\ not(u) &= \mathbf{1}_{\{-u+0.5 \geq 0\}} && (w = -1, b = 0.5) \end{aligned}$$

Hence, **any Boolean function can be build with such units.**

(McCulloch and Pitts, 1943)

The perceptron is very similar

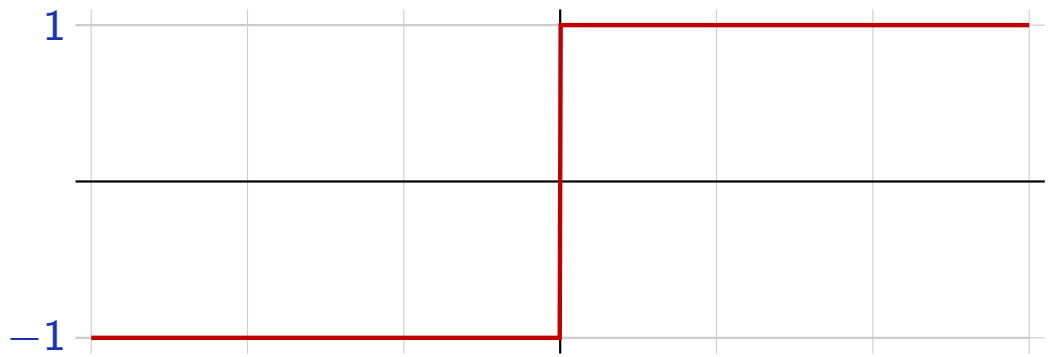
$$f(x) = \begin{cases} 1 & \text{if} \quad \sum_i w_i \, x_i + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

but the inputs are real valued and weights can be different (Rosenblatt, 1957).

It was originally motivated by biology, with w_i being the *synaptic weights*, and x_i and f firing rates. However, it is a (very) crude biological model.

To make things simpler we take responses ± 1. Let

$$\sigma(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

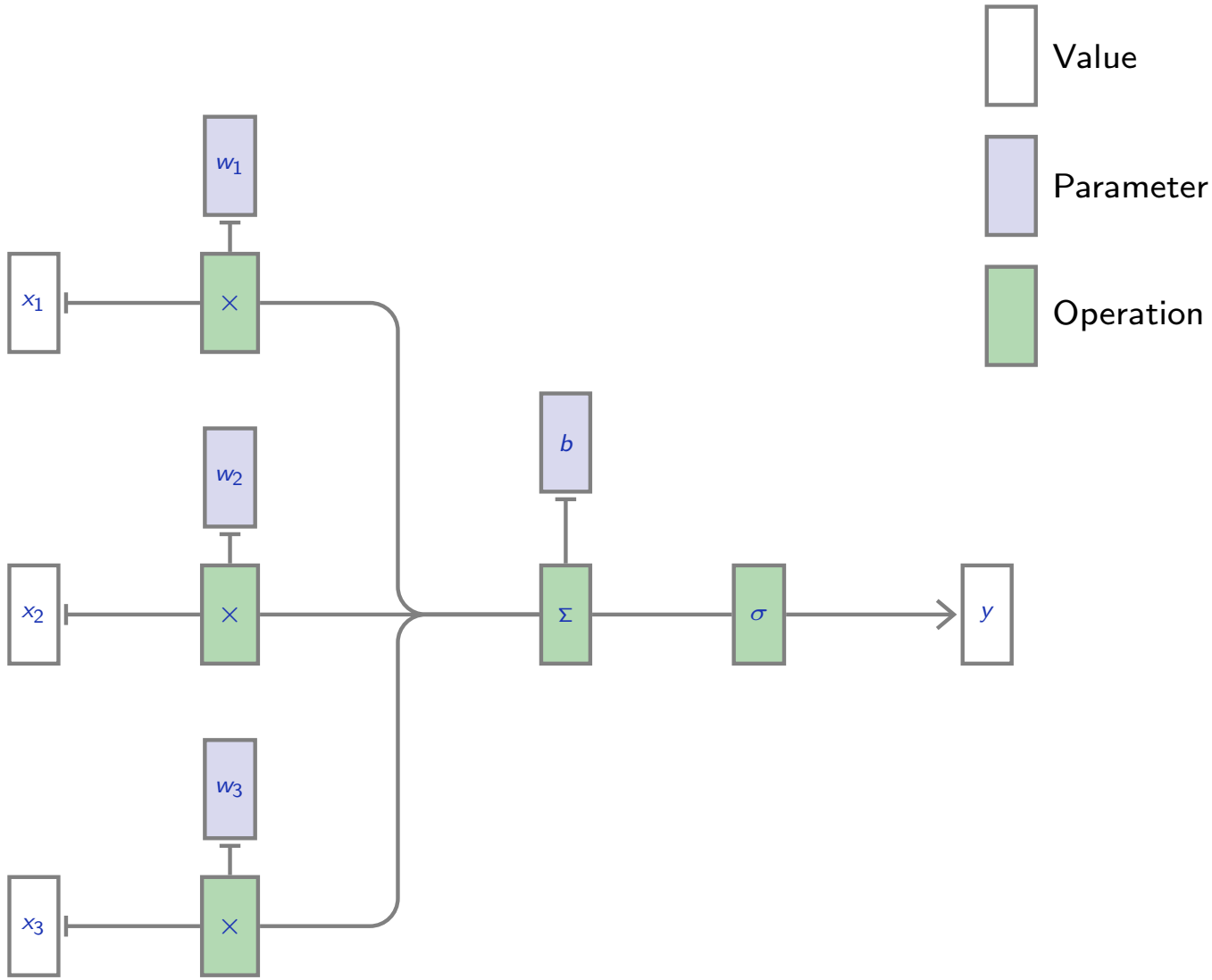


The perceptron classification rule boils down to

$$f(x) = \sigma(w \cdot x + b).$$

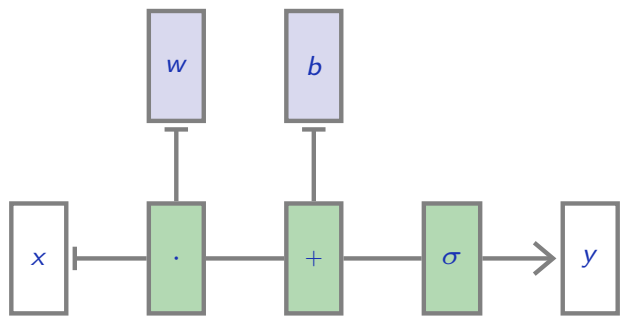
For neural networks, the function σ that follows a linear operator is called the **activation function**.

We can represent this "neuron" as follows:



We can also use tensor operations, as in

$$f(x) = \sigma(w \cdot x + b).$$



Given a training set

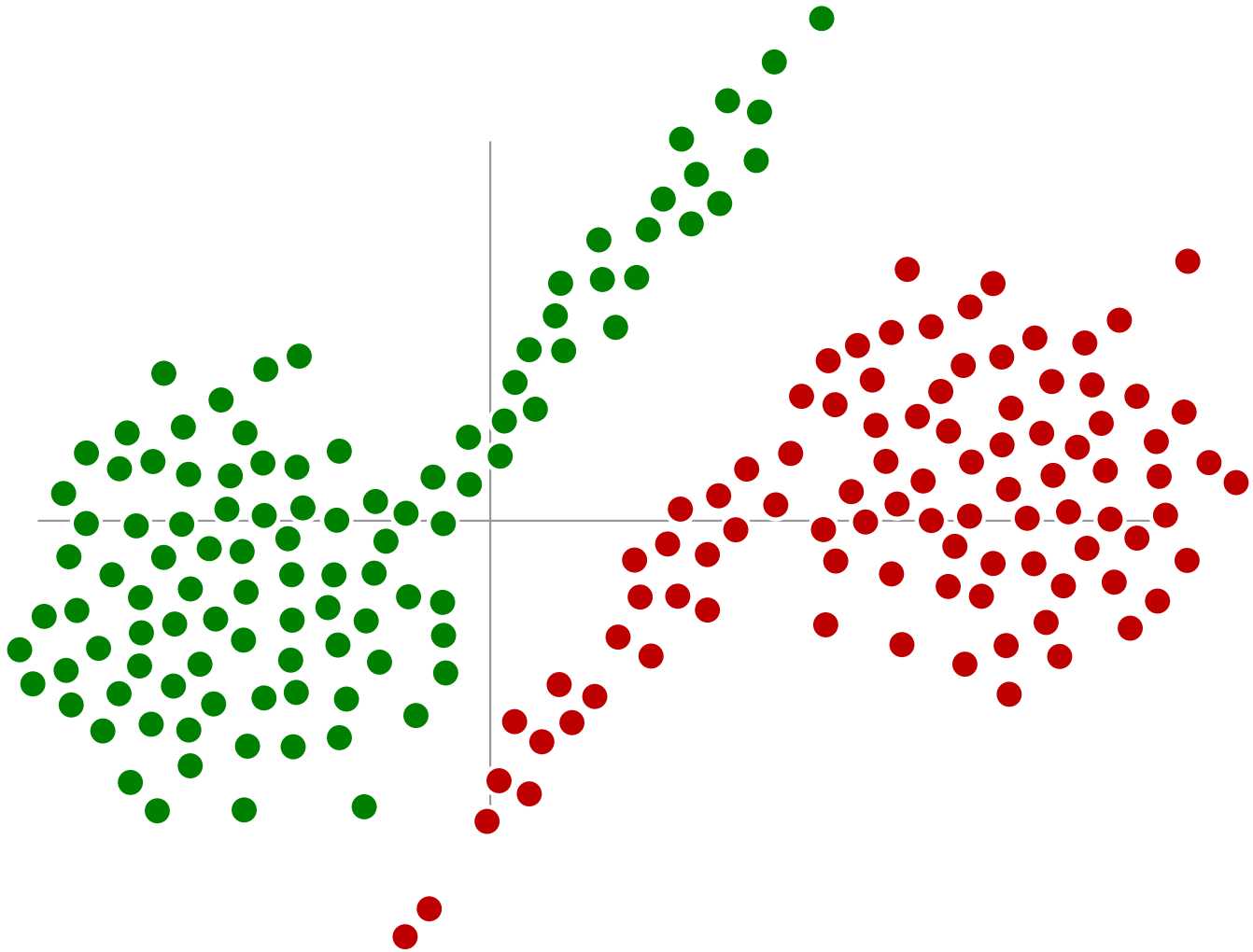
$$(x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}, \quad n = 1, \ldots, N,$$

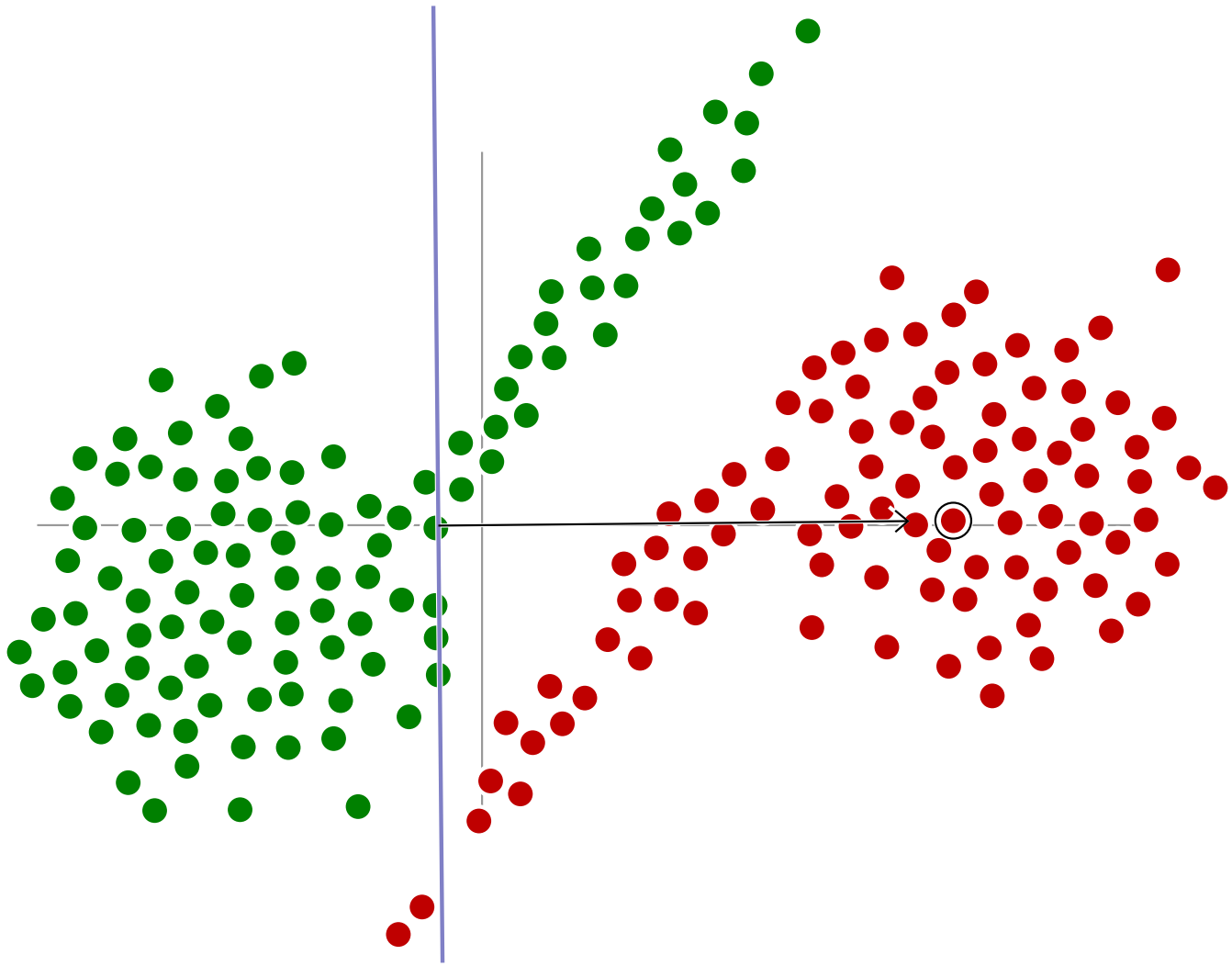
a very simple scheme to train such a linear operator for classification is the **perceptron algorithm:**

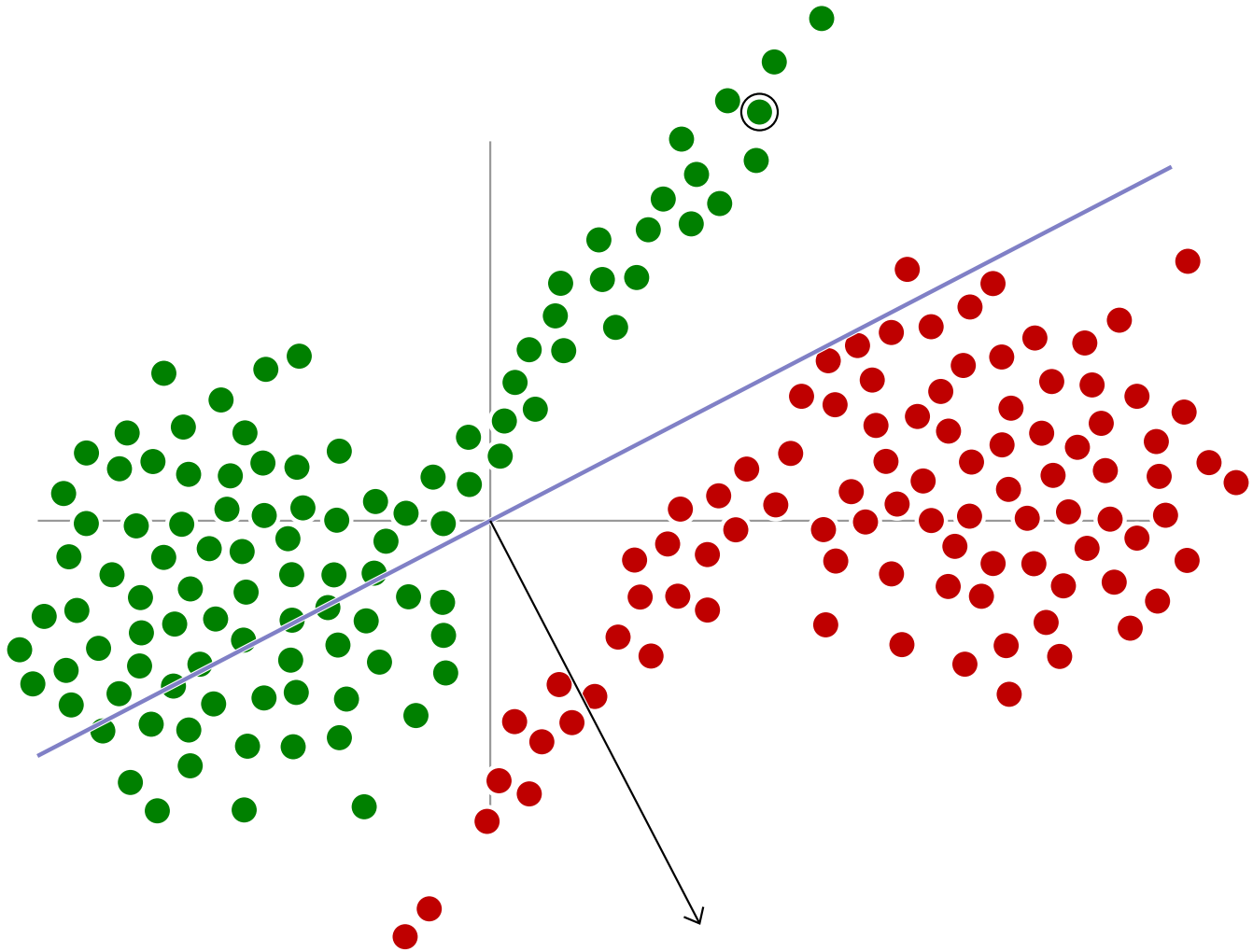
1. Start with $w^0 = 0$,
2. while $\exists n_k$ s.t. $y_{n_k} \left(w^k \cdot x_{n_k} \right) \leq 0$, update $w^{k+1} = w^k + y_{n_k} x_{n_k}$.

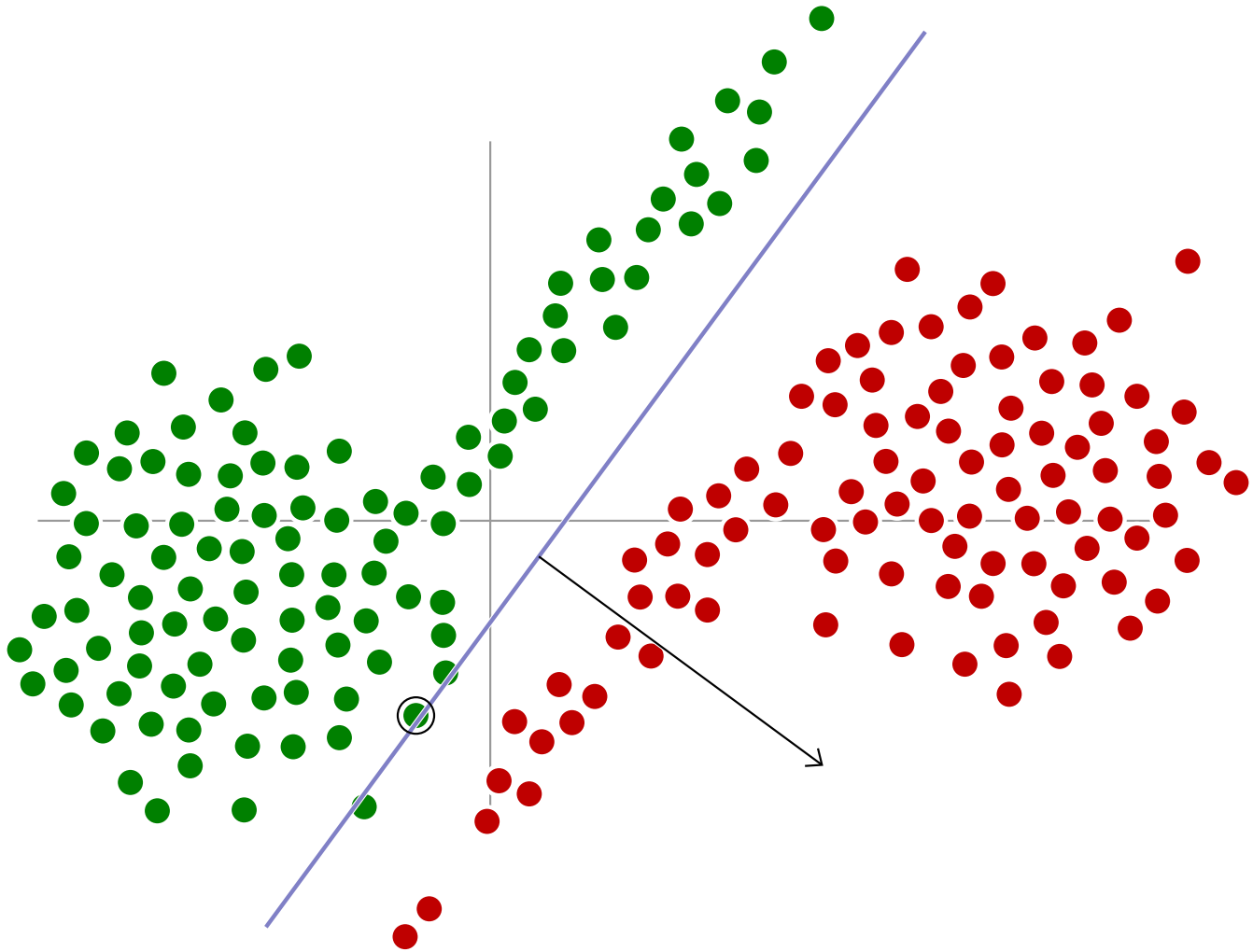
The bias b can be introduced as one of the ws by adding a constant component to x equal to 1.

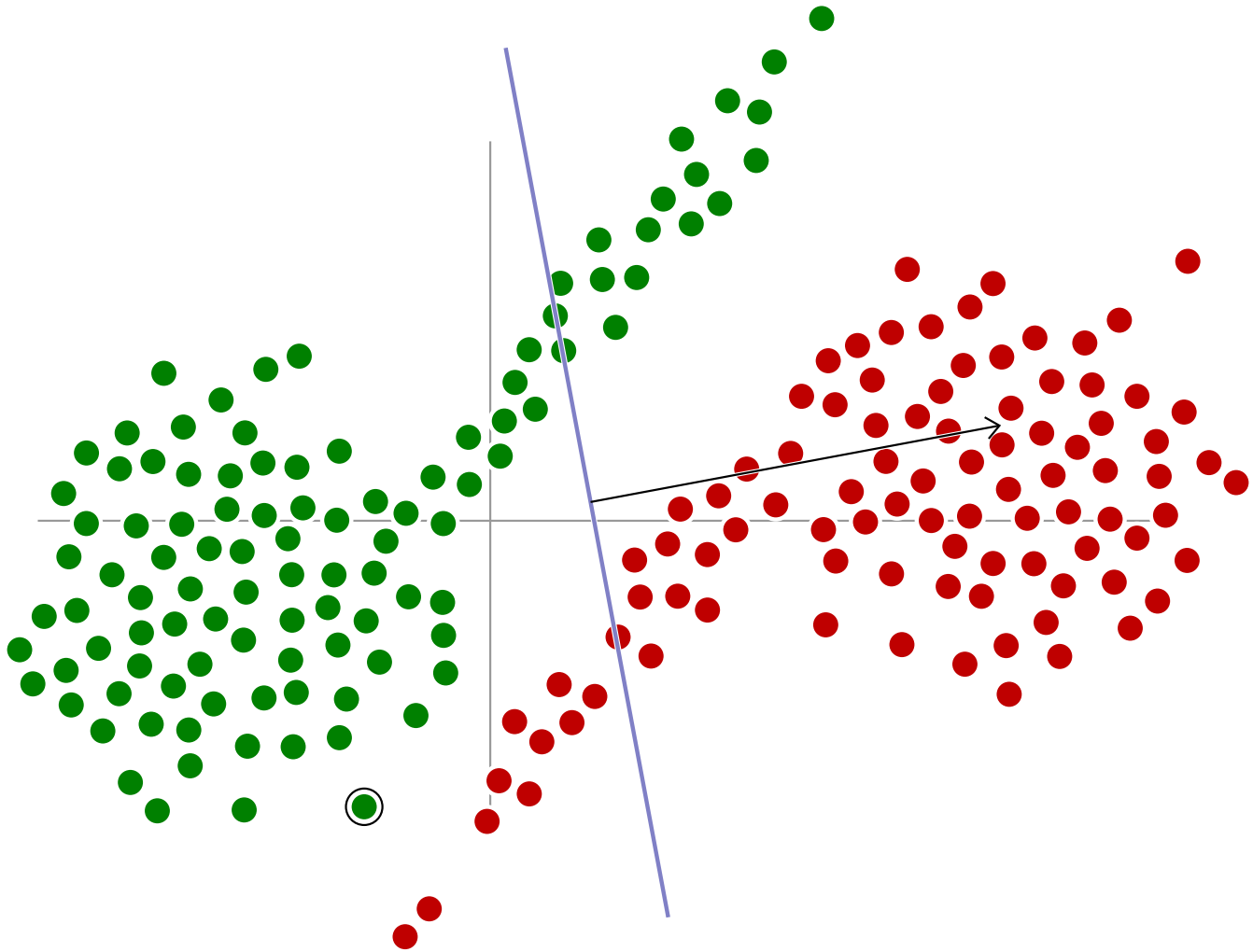
```
def train_perceptron(x, y, nb_epochs_max):
    w = torch.zeros(x.size(1))

    for e in range(nb_epochs_max):
        nb_changes = 0
        for i in range(x.size(0)):
            if x[i].dot(w) * y[i] <= 0:
                w = w + y[i] * x[i]
                nb_changes = nb_changes + 1
        if nb_changes == 0: break;

    return w
```

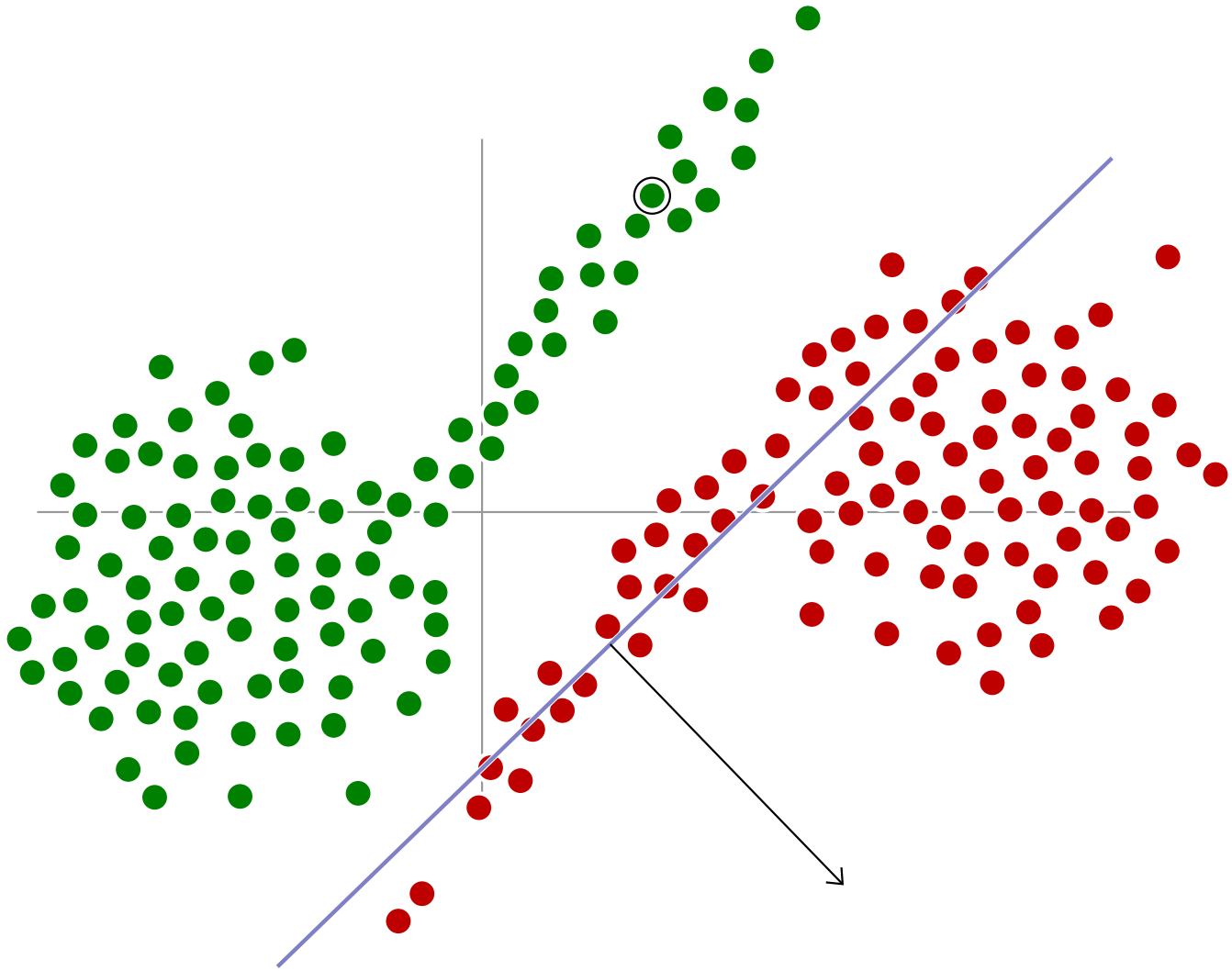


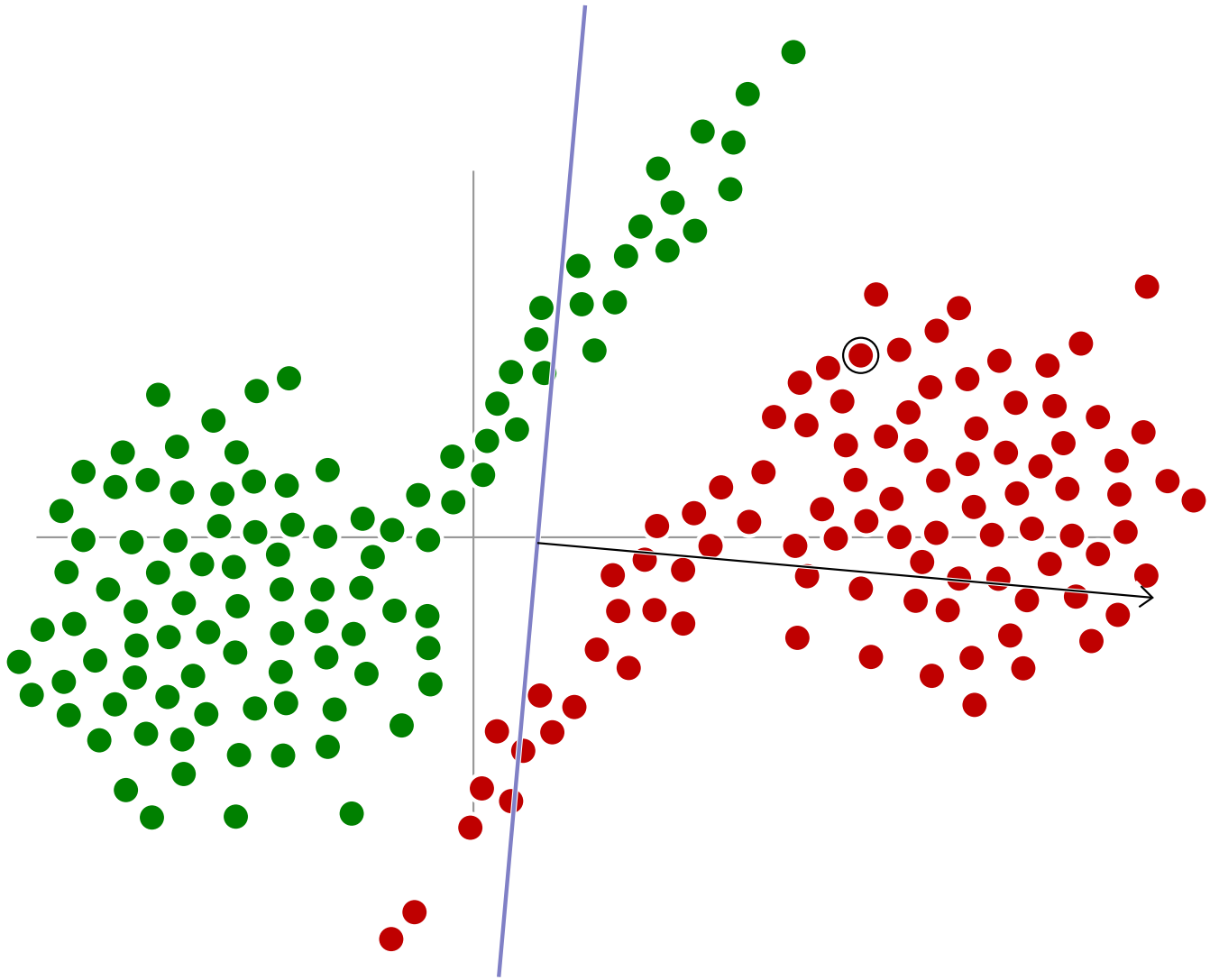


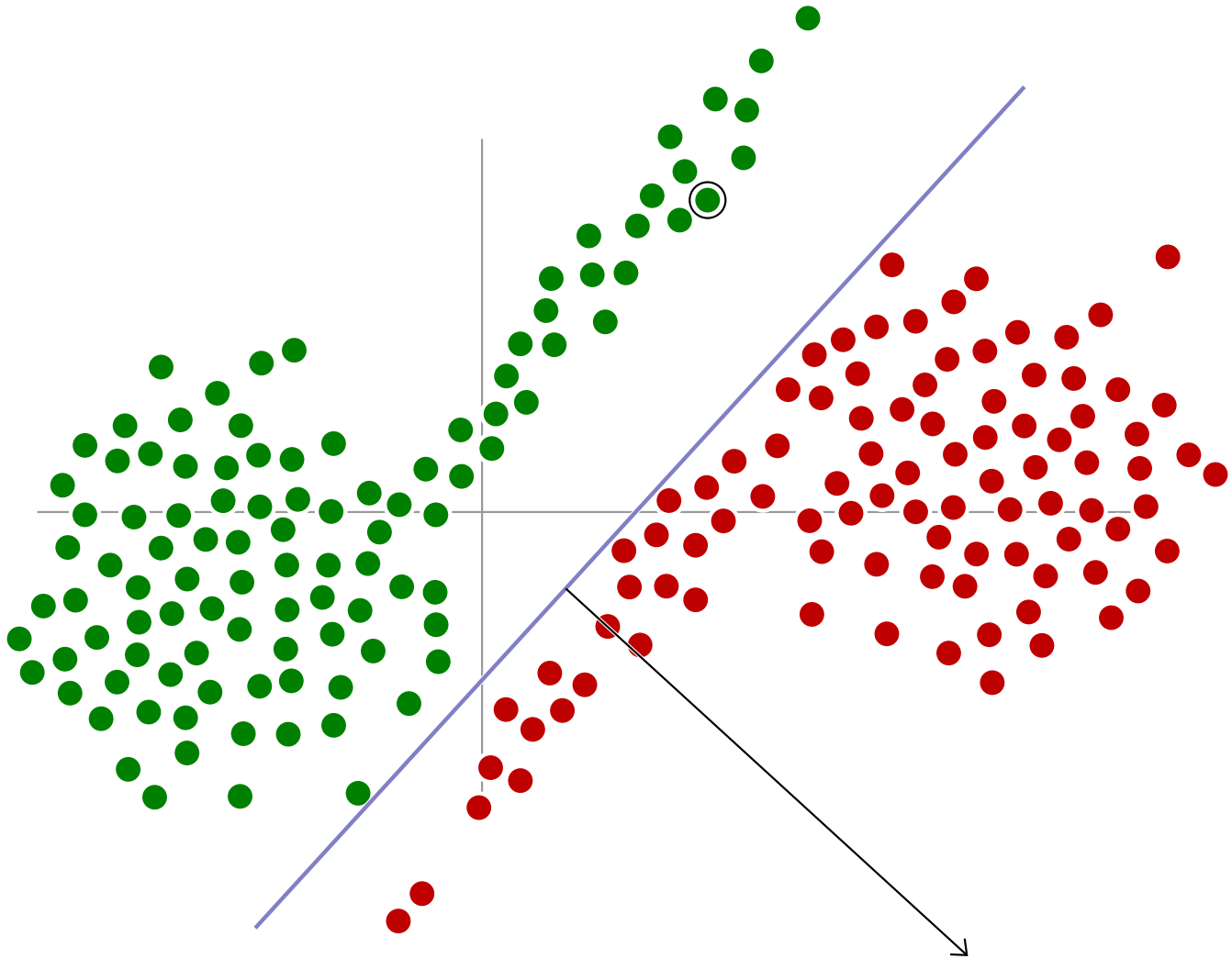




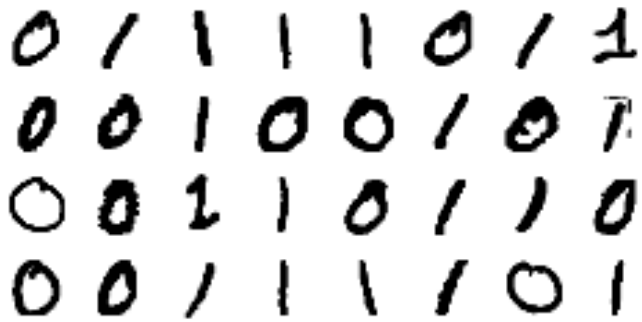




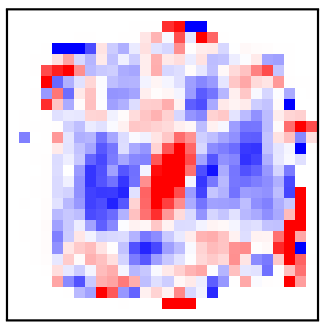




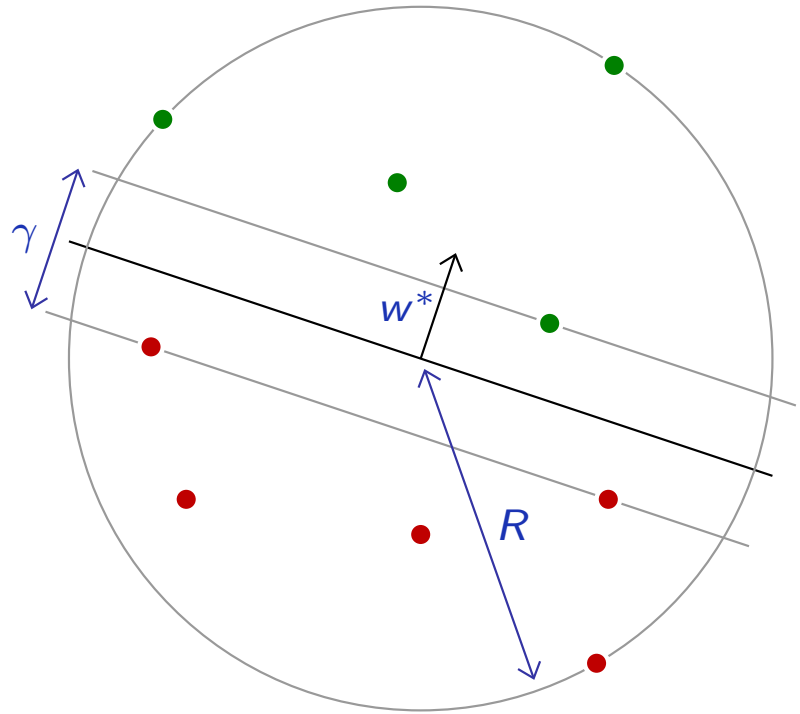
This crude algorithm works often surprisingly well. With MNIST's "0"s as negative class, and "1"s as positive one.



```
epoch 0 nb_changes 64 train_error 0.23% test_error 0.19%
epoch 1 nb_changes 24 train_error 0.07% test_error 0.00%
epoch 2 nb_changes 10 train_error 0.06% test_error 0.05%
epoch 3 nb_changes 6 train_error 0.03% test_error 0.14%
epoch 4 nb_changes 5 train_error 0.03% test_error 0.09%
epoch 5 nb_changes 4 train_error 0.02% test_error 0.14%
epoch 6 nb_changes 3 train_error 0.01% test_error 0.14%
epoch 7 nb_changes 2 train_error 0.00% test_error 0.14%
epoch 8 nb_changes 0 train_error 0.00% test_error 0.14%
```



We can get a convergence result under two assumptions:



1. The x_n are in a sphere of radius R:

$$\exists R > 0,\ \forall n,\ \|x_n\| \leq R.$$

2. The two populations can be separated with a margin γ:

$$\exists w^*,\ \|w^*\| = 1,\ \exists \gamma > 0,\ \forall n,\ y_n\left(x_n \cdot w^*\right) \geq \gamma/2.$$

To prove the convergence, let us make the assumption that there still is a misclassified sample at iteration k.

We have

$$\begin{aligned} w^{k+1} \cdot w^* &= \left(w^k + y_{n_k} x_{n_k}\right) \cdot w^* \\ &= w^k \cdot w^* + y_{n_k} \left(x_{n_k} \cdot w^*\right) \\ &\geq w^k \cdot w^* + \gamma/2 \\ &\geq (k+1)\,\gamma/2. \end{aligned}$$

Since

$$\|w^k\| \|w^*\| \geq w^k \cdot w^*,$$

we get

$$\begin{aligned} \|w^k\|^2 &\geq \left(w^k \cdot w^*\right)^2 / \|w^*\|^2 \\ &\geq k^2 \gamma^2 / 4. \end{aligned}$$

And

$$
\begin{aligned}
\|w^{k+1}\|^2 &= w^{k+1} \cdot w^{k+1} \\
&= \left(w^k + y_{n_k} x_{n_k}\right) \cdot \left(w^k + y_{n_k} x_{n_k}\right) \\
&= w^k \cdot w^k + 2 \underbrace{y_{n_k}\, w^k \cdot x_{n_k}}_{\leq 0} + \underbrace{\|x_{n_k}\|^2}_{\leq R^2} \\
&\leq \|w^k\|^2 + R^2 \\
&\leq (k+1)\, R^2.
\end{aligned}
$$

Putting these two results together, we get

$$k^2\gamma^2/4 \leq \|w^k\|^2 \leq \ k\,R^2$$

hence

$$k \leq 4R^2/\gamma^2,$$

hence no misclassified sample can remain after $\lfloor 4R^2/\gamma^2 \rfloor$ iterations.

This result makes sense:

- The bound does not change if the population is scaled, and
- the larger the margin, the more quickly the algorithm classifies all the samples correctly.

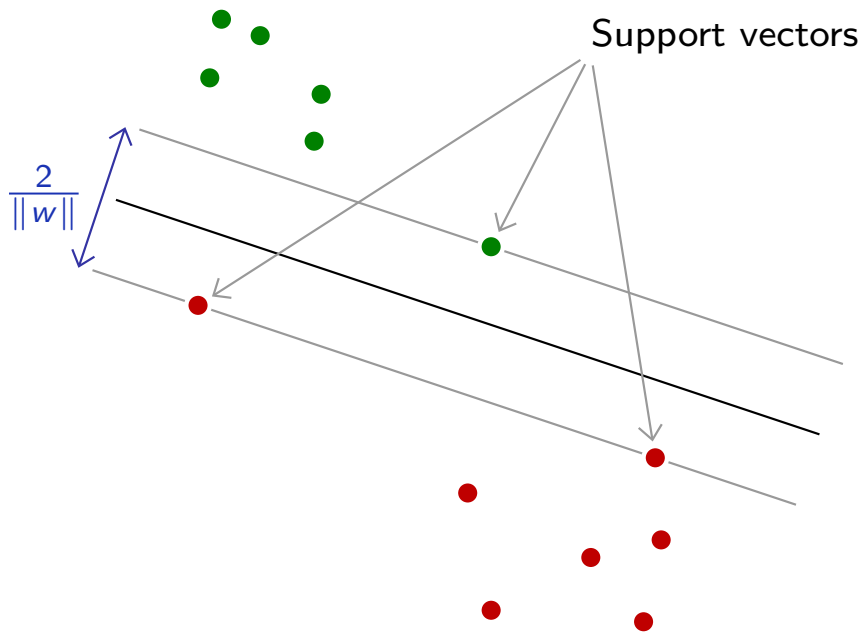
The perceptron stops as soon as it finds a separating boundary. Other algorithms maximize the distance of samples to the decision boundary, which improves robustness to noise.

Support Vector Machines (SVM) achieve this by minimizing

$$\mathscr{L}(w, b) = \lambda \|w\|^2 + \frac{1}{N} \sum_n \max(0, 1 - y_n(w \cdot x_n + b)),$$

which is convex and has a global optimum.

$$\mathscr{L}(w, b) = \lambda \|w\|^2 + \frac{1}{N} \sum_n \max(0, 1 - y_n(w \cdot x_n + b))$$



Minimizing $\max(0, 1 - y_n(w \cdot x_n + b))$ pushes the nth sample beyond the plane $w \cdot x + b = y_n$, and minimizing $\|w\|^2$ increases the distance between the $w \cdot x + b = \pm 1$.

At convergence, only a small number of samples matter, the "support vectors".

The term

$$\max(0, 1 - \alpha)$$

is the so called "hinge loss"

Deep learning

3.2. Probabilistic view of a linear classifier

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

The Linear Discriminant Analysis (LDA) algorithm provides a nice bridge between these linear classifiers and probabilistic modeling.

Consider the following class populations

$$\forall y \in \{0,1\}, x \in \mathbb{R}^D,$$

$$\mu_{X|Y=y}(x) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(x - m_y)\Sigma^{-1}(x - m_y)^T\right).$$

That is, they are Gaussian with **the same covariance matrix Σ.** This is the **homoscedasticity** assumption.

Intuitively we can map data linearly to make all the covariance matrices identity, there the Bayesian separation is a plan, so it is also in the original space.

We have

$$
\begin{aligned}
P(Y=1 \mid X=x) &= \frac{\mu_{X \mid Y=1}(x) P(Y=1)}{\mu_X(x)} \\
&= \frac{\mu_{X \mid Y=1}(x) P(Y=1)}{\mu_{X \mid Y=0}(x) P(Y=0) + \mu_{X \mid Y=1}(x) P(Y=1)} \\
&= \frac{1}{1 + \frac{\mu_{X \mid Y=0}(x)}{\mu_{X \mid Y=1}(x)} \frac{P(Y=0)}{P(Y=1)}} \\
&= \sigma\left(\log \frac{\mu_{X \mid Y=1}(x)}{\mu_{X \mid Y=0}(x)} + \log \frac{P(Y=1)}{P(Y=0)} \right),
\end{aligned}
$$

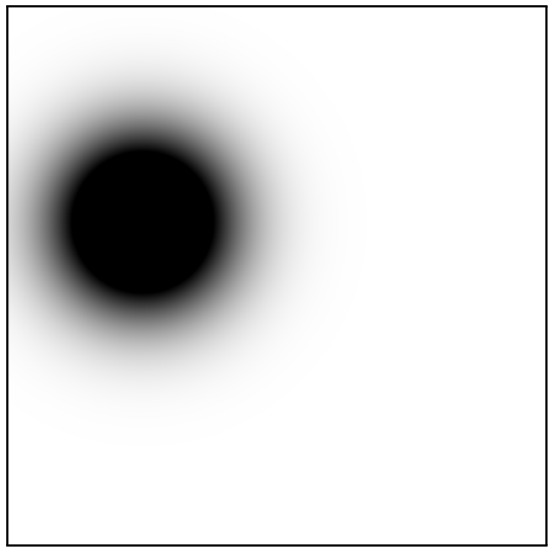
with

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

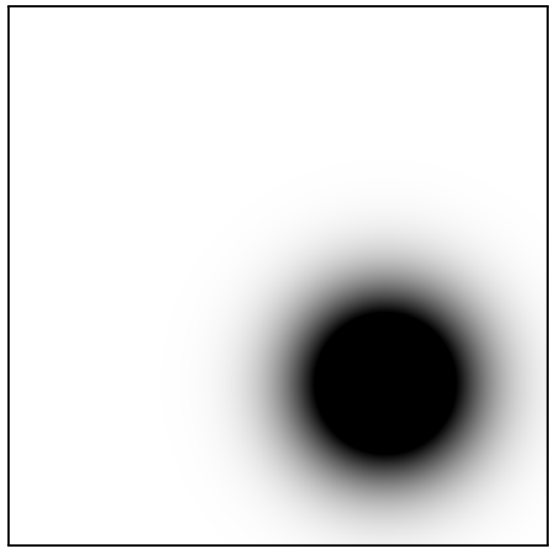
So with our Gaussians $\mu_{X|Y=y}$ of same Σ, we get

$$
\begin{aligned}
&P(Y=1 \mid X=x) \\
&\quad = \sigma\left(\log \frac{\mu_{X|Y=1}(x)}{\mu_{X|Y=0}(x)} + \underbrace{\log \frac{P(Y=1)}{P(Y=0)}}_{Z} \right) \\
&\quad = \sigma\left(\log \mu_{X|Y=1}(x) - \log \mu_{X|Y=0}(x) + Z \right) \\
&\quad = \sigma\left(-\frac{1}{2}(x-m_1)\Sigma^{-1}(x-m_1)^T + \frac{1}{2}(x-m_0)\Sigma^{-1}(x-m_0)^T + Z \right) \\
&\quad = \sigma\Big(-\frac{1}{2}x\Sigma^{-1}x^T + m_1\Sigma^{-1}x^T - \frac{1}{2}m_1\Sigma^{-1}m_1^T \\
&\qquad\quad + \frac{1}{2}x\Sigma^{-1}x^T - m_0\Sigma^{-1}x^T + \frac{1}{2}m_0\Sigma^{-1}m_0^T + Z \Big) \\
&\quad = \sigma\left(\underbrace{(m_1-m_0)\Sigma^{-1}}_{w} x^T + \underbrace{\frac{1}{2}\left(m_0\Sigma^{-1}m_0^T - m_1\Sigma^{-1}m_1^T \right) + Z}_{b} \right) \\
&\quad = \sigma(w \cdot x + b).
\end{aligned}
$$

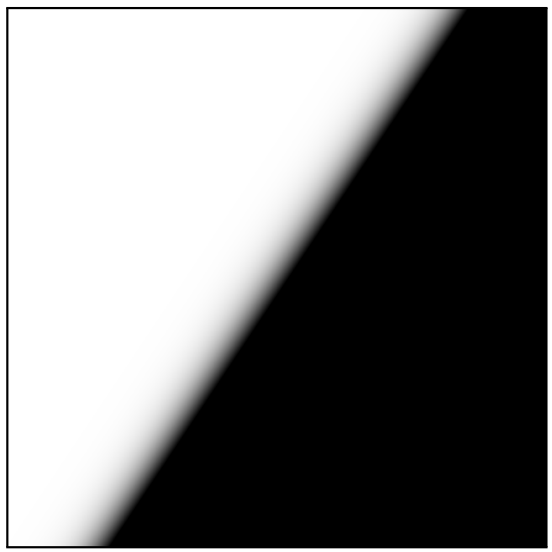
The homoscedasticity makes the second-order terms vanish.



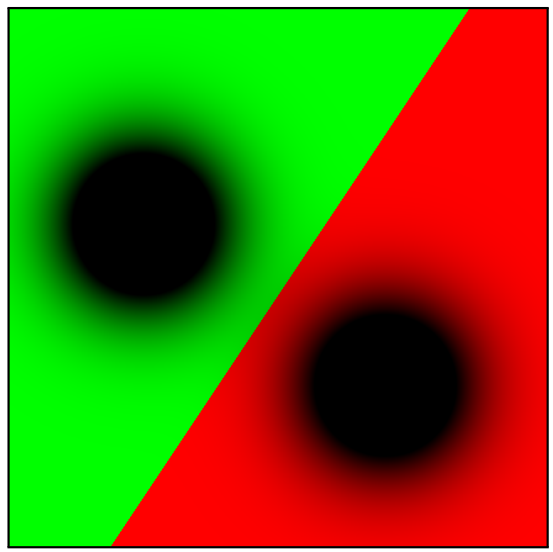
$\mu_{X|Y=0}$

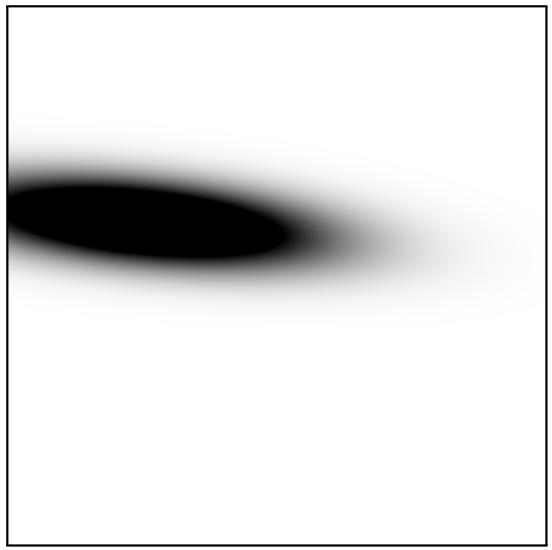


$\mu_{X|Y=1}$

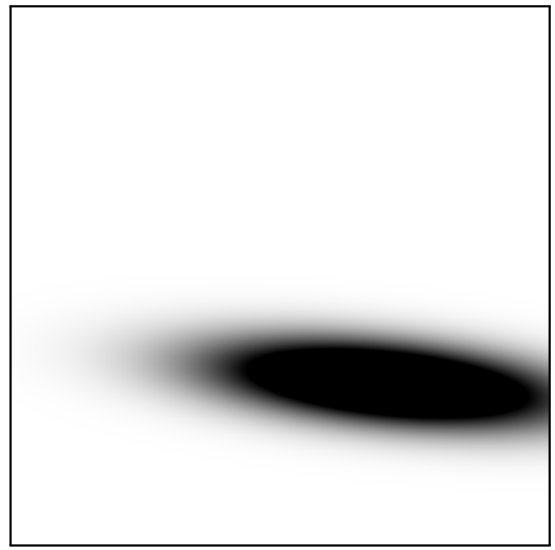


$P(Y = 1 \mid X = x)$

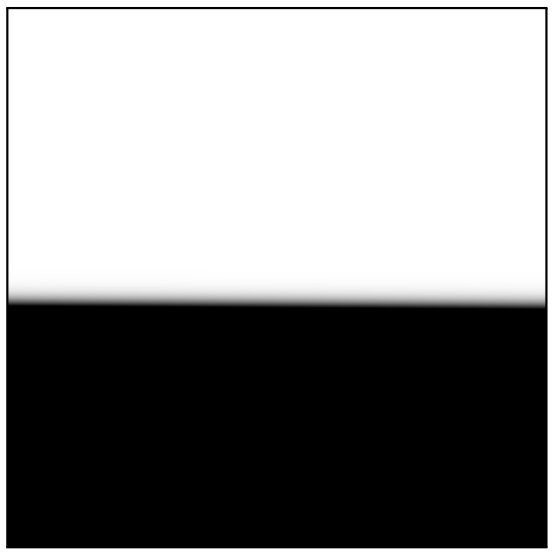




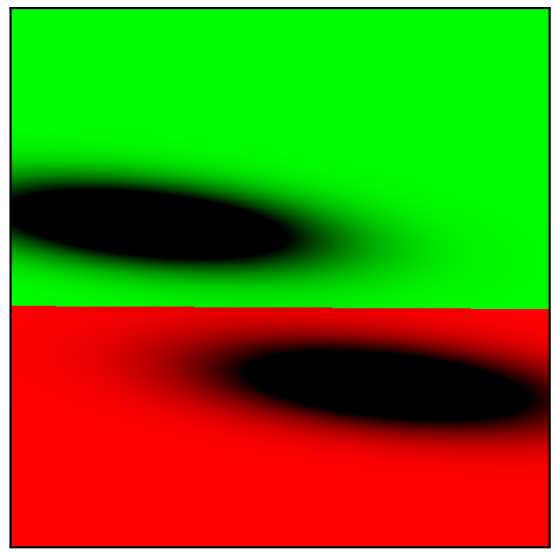
$\mu_{X|Y=0}$

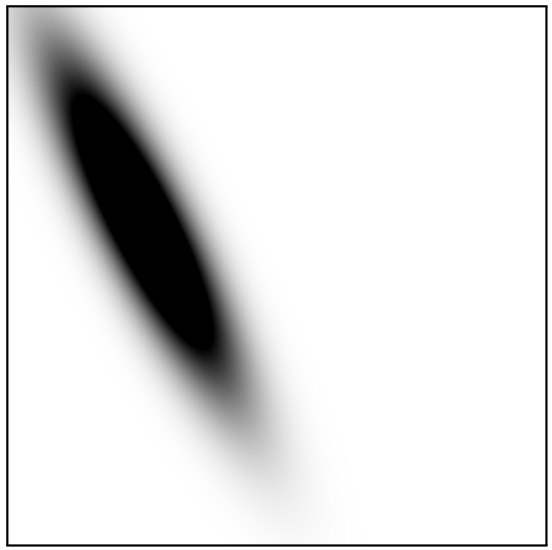


$\mu_{X|Y=1}$



$P(Y = 1 \mid X = x)$

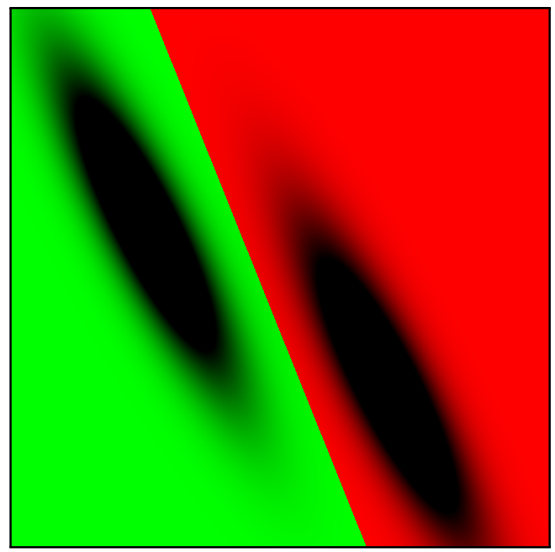




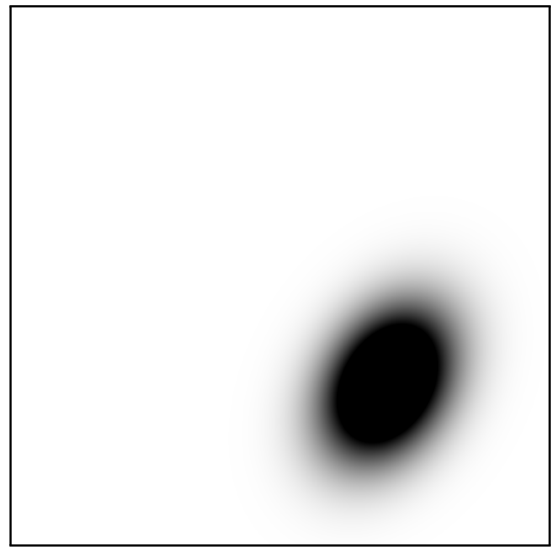
$\mu_{X|Y=0}$

$\mu_{X|Y=1}$

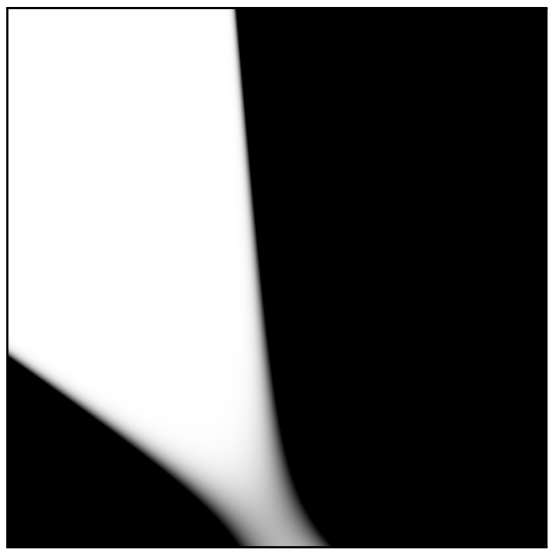
$P(Y = 1 \mid X = x)$



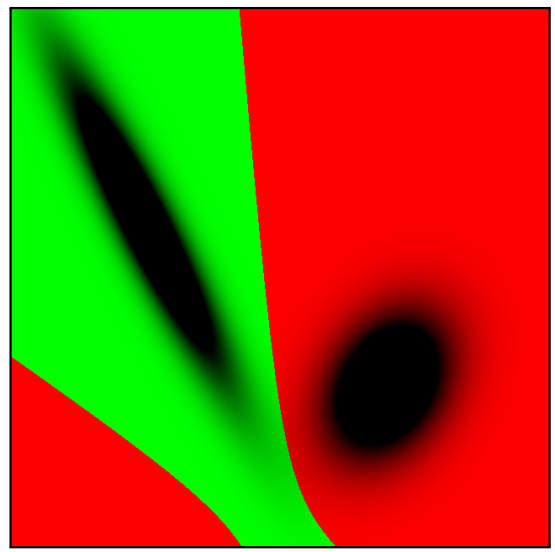
$\mu_{X|Y=0}$



$\mu_{X|Y=1}$



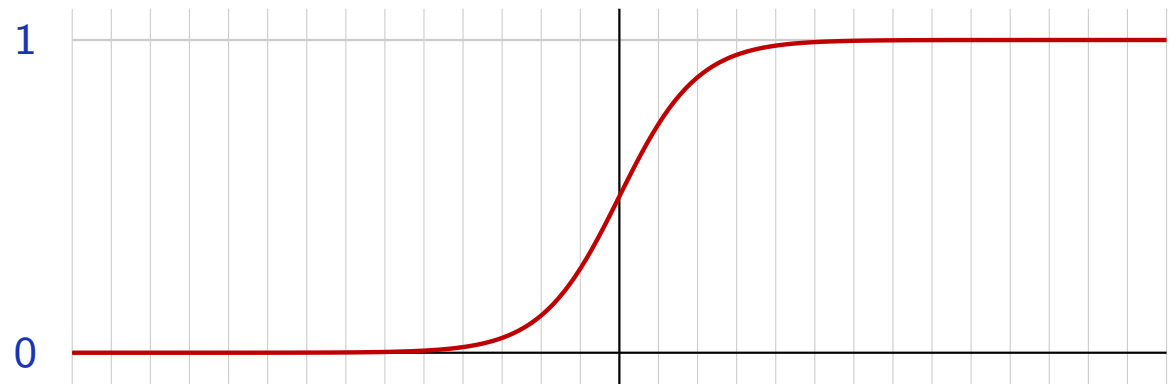
$P(Y = 1 \mid X = x)$



Note that the (logistic) sigmoid function

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

looks like a "soft heavyside"



So the overall model

$$f(x; w, b) = \sigma(w \cdot x + b)$$

looks very similar to the perceptron.

We can use the model from LDA

$$f(x; w, b) = \sigma(w \cdot x + b)$$

but instead of modeling the densities and derive the values of w and b, directly compute them by maximizing their probability given the training data.

First, to simplify the next slide, note that we have

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

hence if Y takes value in $\{-1, 1\}$ then

$$\forall y \in \{-1, 1\}, \quad P(Y = y \mid X = x) = \sigma(y(w \cdot x + b)).$$

We have

$$\begin{aligned}
&\log \mu_{W,B}(w, b \mid \mathscr{D} = \mathbf{d}) \\
&\qquad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid W = w, B = b)\, \mu_{W,B}(w, b)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\qquad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid W = w, B = b) + \log \mu_{W,B}(w, b) - \log Z \\
&\qquad = \sum_n \log \sigma(y_n(w \cdot x_n + b)) + \log \mu_{W,B}(w, b) - \log Z'
\end{aligned}$$

This is the **logistic regression**, whose loss aims at minimizing

$$-\log \sigma(y_n f(x_n)).$$

Although the probabilistic and Bayesian formulations may be helpful in certain contexts, the bulk of deep learning is disconnected from such modeling.

We will come back sometime to a probabilistic interpretation, but most of the methods will be envisioned from the signal-processing and optimization angles.

Deep learning

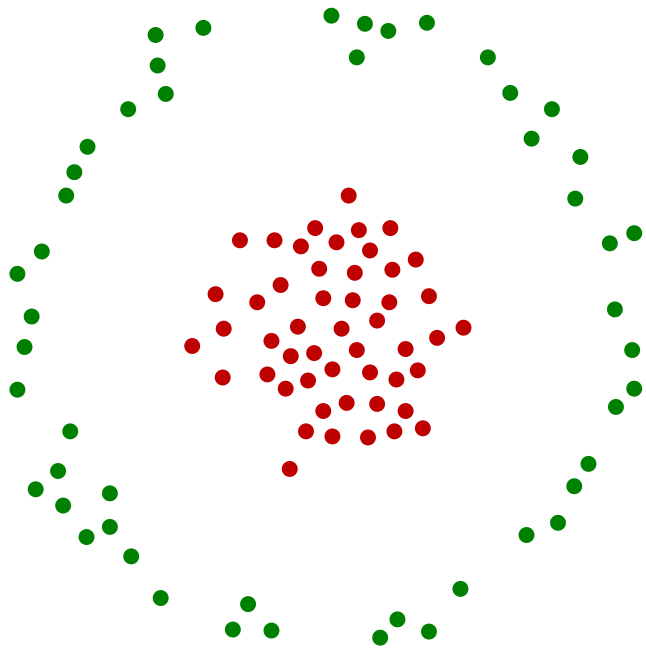
3.3. Linear separability and feature design

François Fleuret

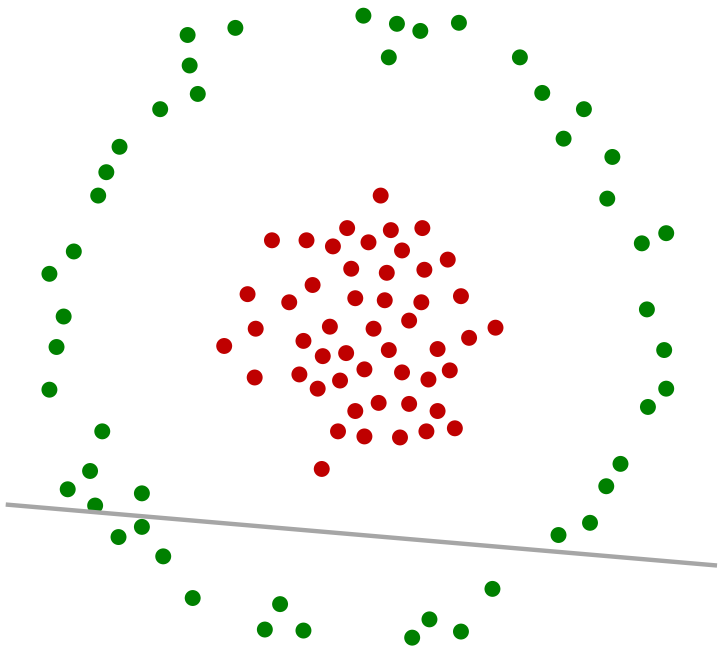
https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

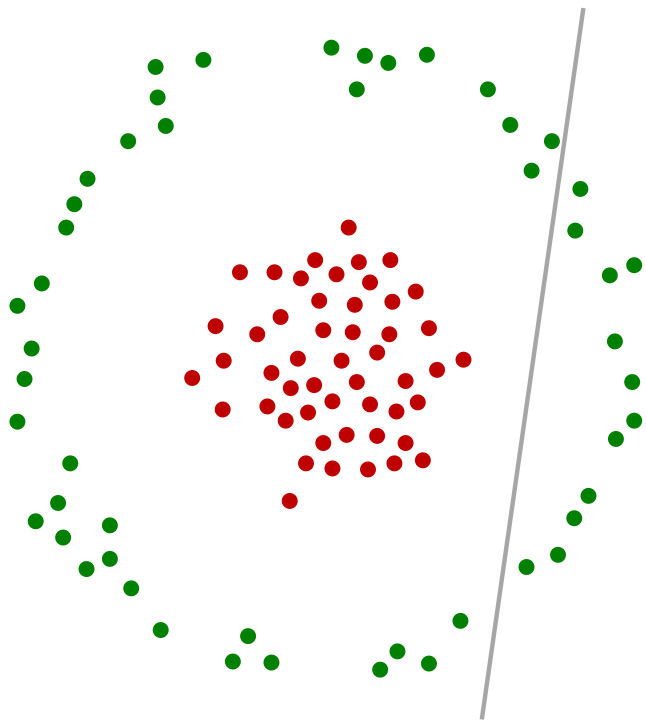
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.



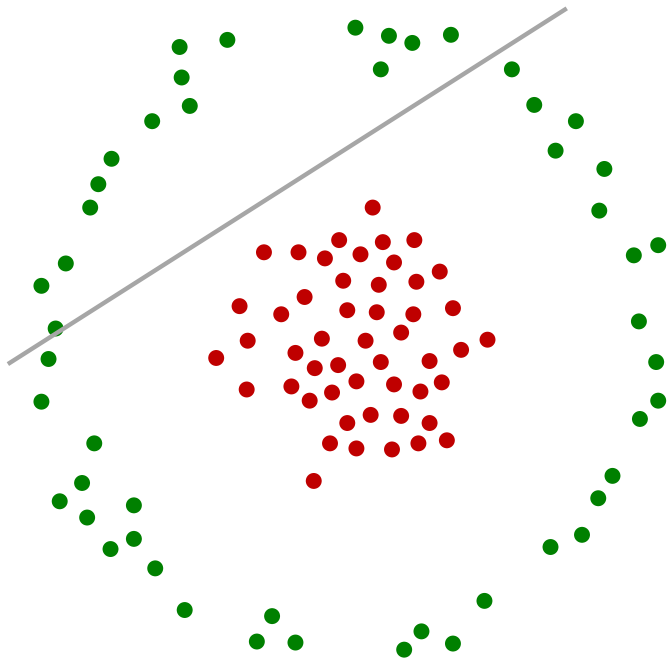
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.



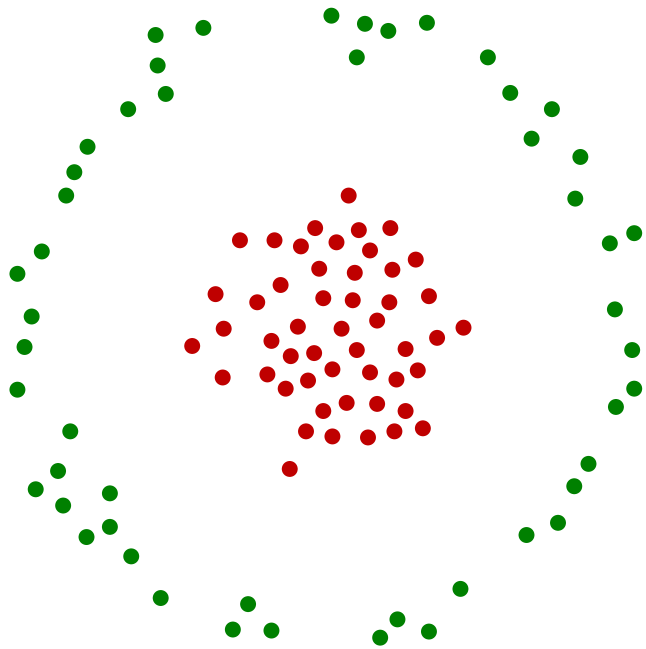
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.

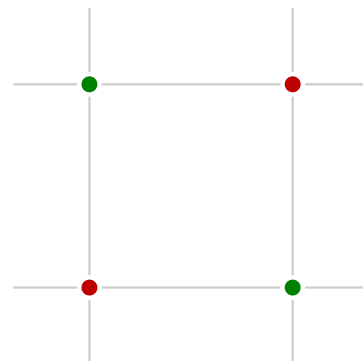


The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.



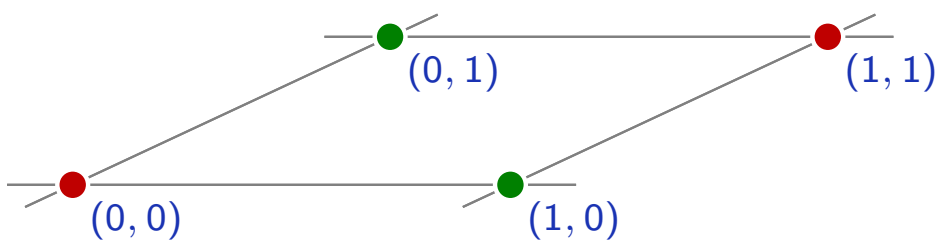
The main weakness of linear predictors is their lack of capacity. For classification, the populations have to be **linearly separable**.





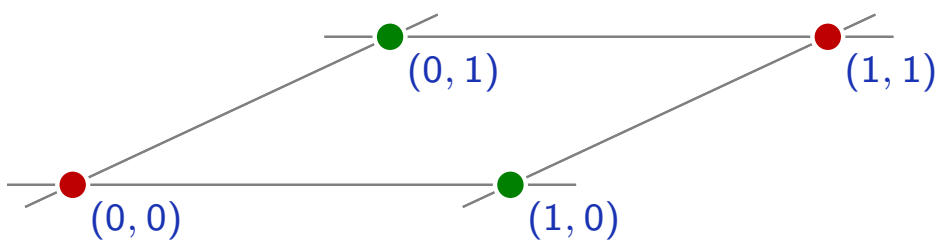
"xor"

The xor example can be solved by pre-processing the data to make the two populations linearly separable.



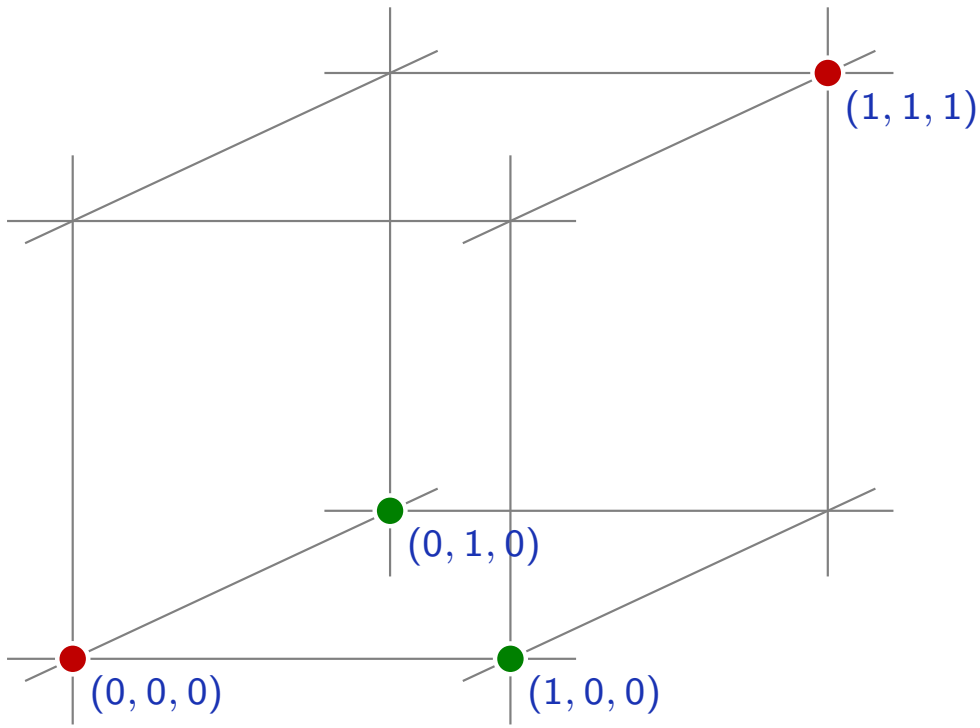
The xor example can be solved by pre-processing the data to make the two populations linearly separable.

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



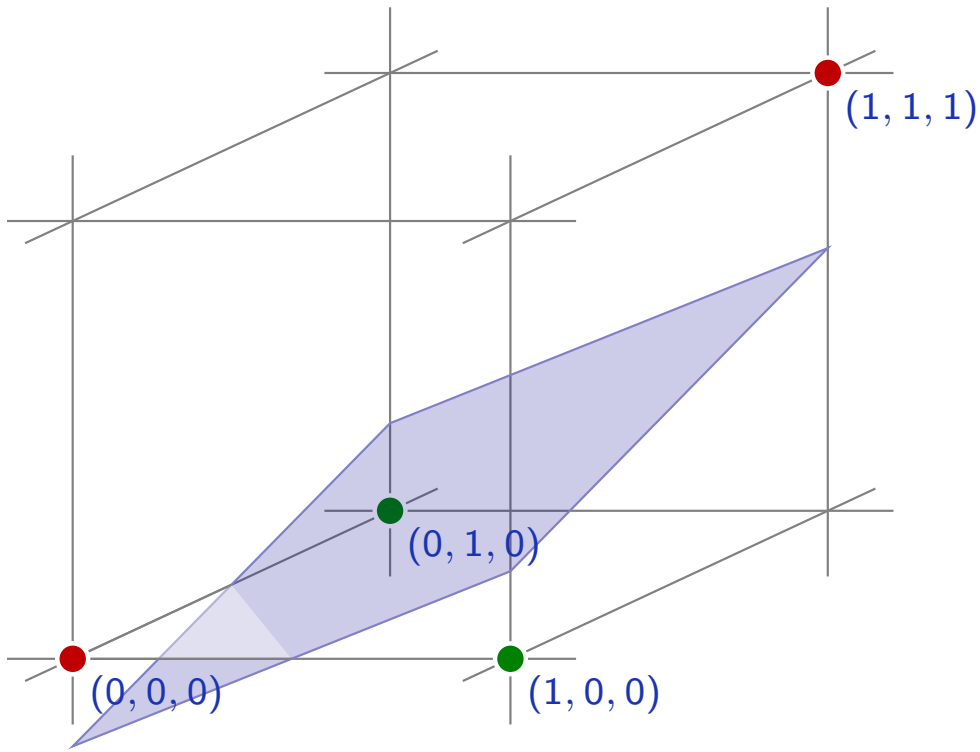
The xor example can be solved by pre-processing the data to make the two populations linearly separable.

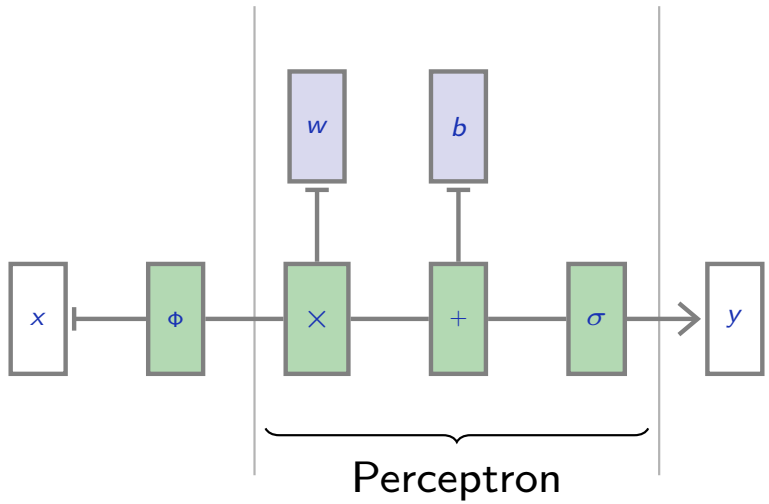
$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$



The xor example can be solved by pre-processing the data to make the two populations linearly separable.

$$\Phi : (x_u, x_v) \mapsto (x_u, x_v, x_u x_v).$$




w
b
x
Φ
×
+
σ
y
Perceptron

This is similar to the polynomial regression. If we have

$$\Phi : x \mapsto (1, x, x^2, \ldots, x^D)$$

and

$$\alpha = (\alpha_0, \ldots, \alpha_D)$$

then

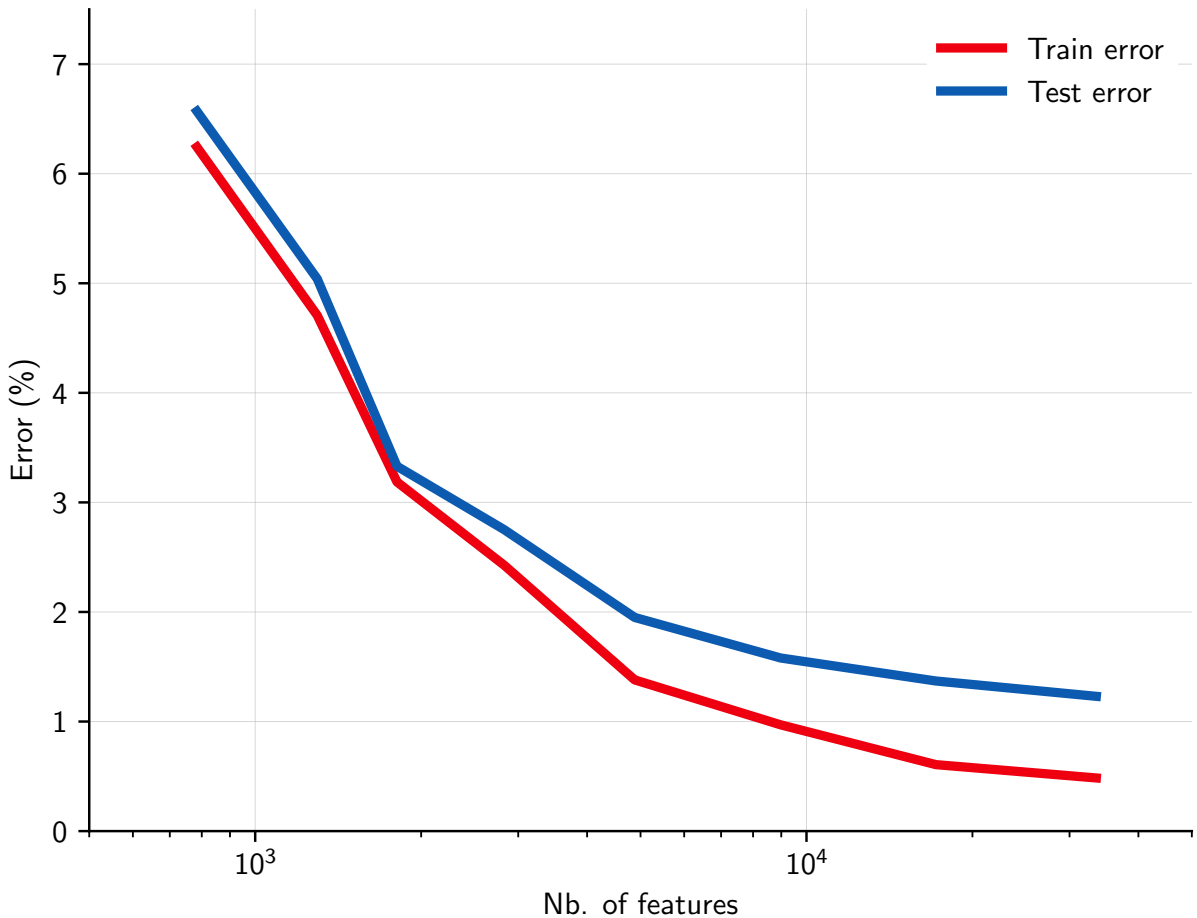
$$\sum_{d=0}^{D} \alpha_d x^d = \alpha \cdot \Phi(x).$$

By increasing D, we can approximate any continuous real function on a compact space (Stone-Weierstrass theorem).

It means that we can make the capacity as high as we want.

We can apply the same to a more realistic binary classification problem: MNIST's "8" vs. the other classes with a perceptron.

The original $28 \times 28 = 784$ features are supplemented with the products of pairs of features taken at random.



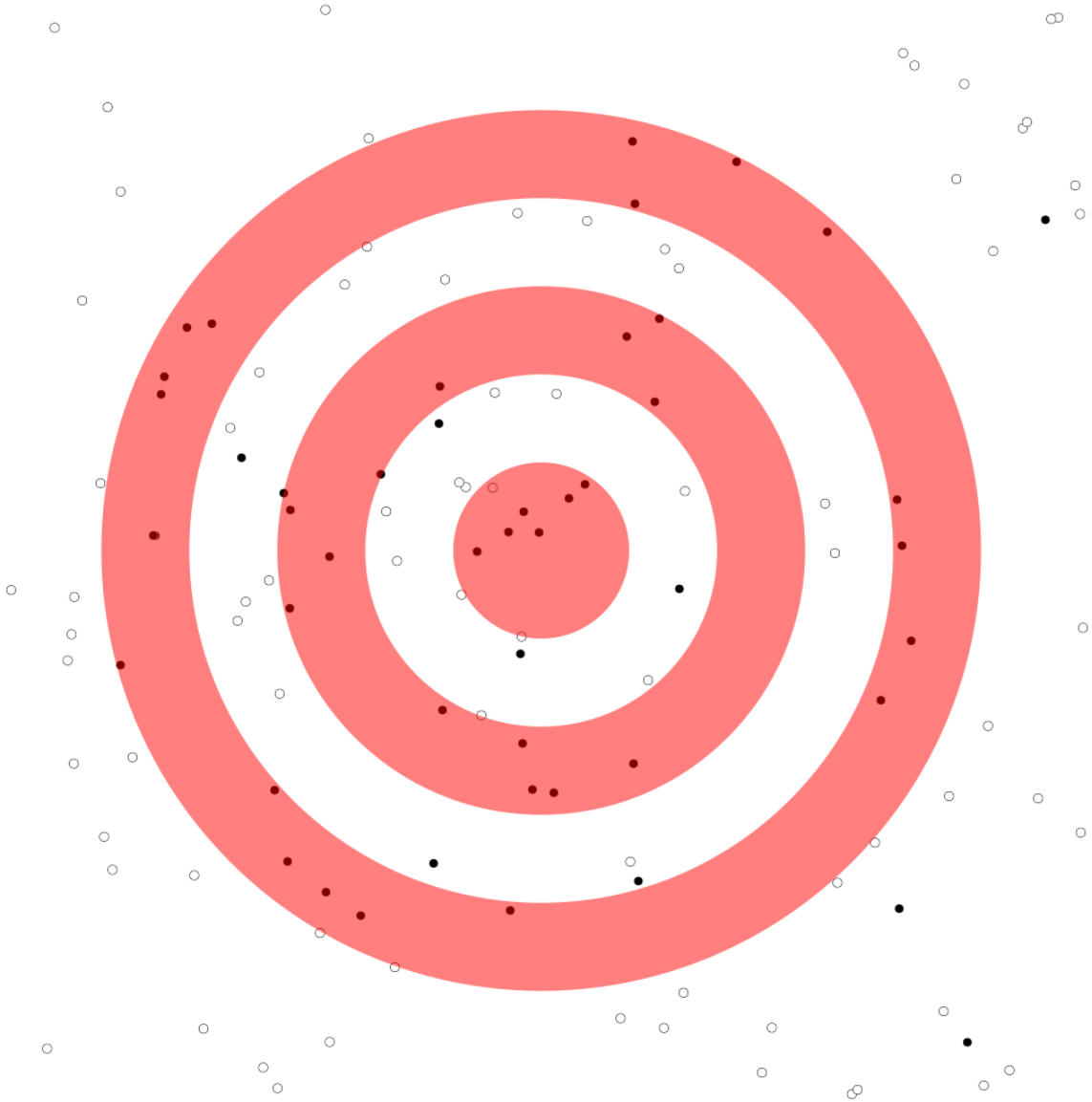
Remember the bias-variance tradeoff we saw in 2.3. "Bias-variance dilemma"

$$\mathbb{E}((Y - y)^2) = \underbrace{(\mathbb{E}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}(Y)}_{\text{Variance}}.$$

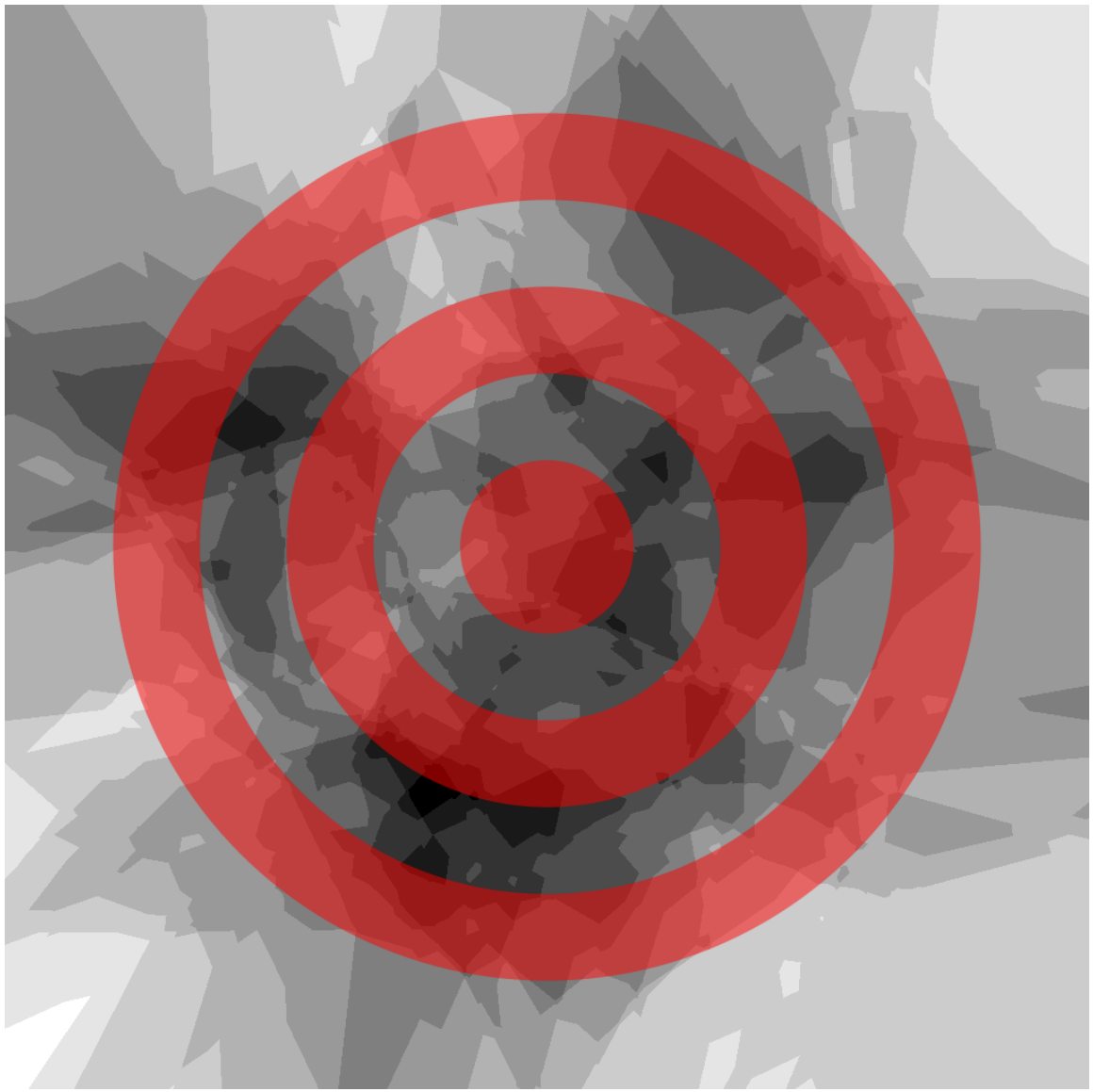
The right class of models reduces the bias more and increases the variance less.

Beside increasing capacity to reduce the bias, "feature design" may also be a way of reducing capacity without hurting the bias, or with improving it.

In particular, good features should be invariant to perturbations of the signal known to keep the value to predict unchanged.

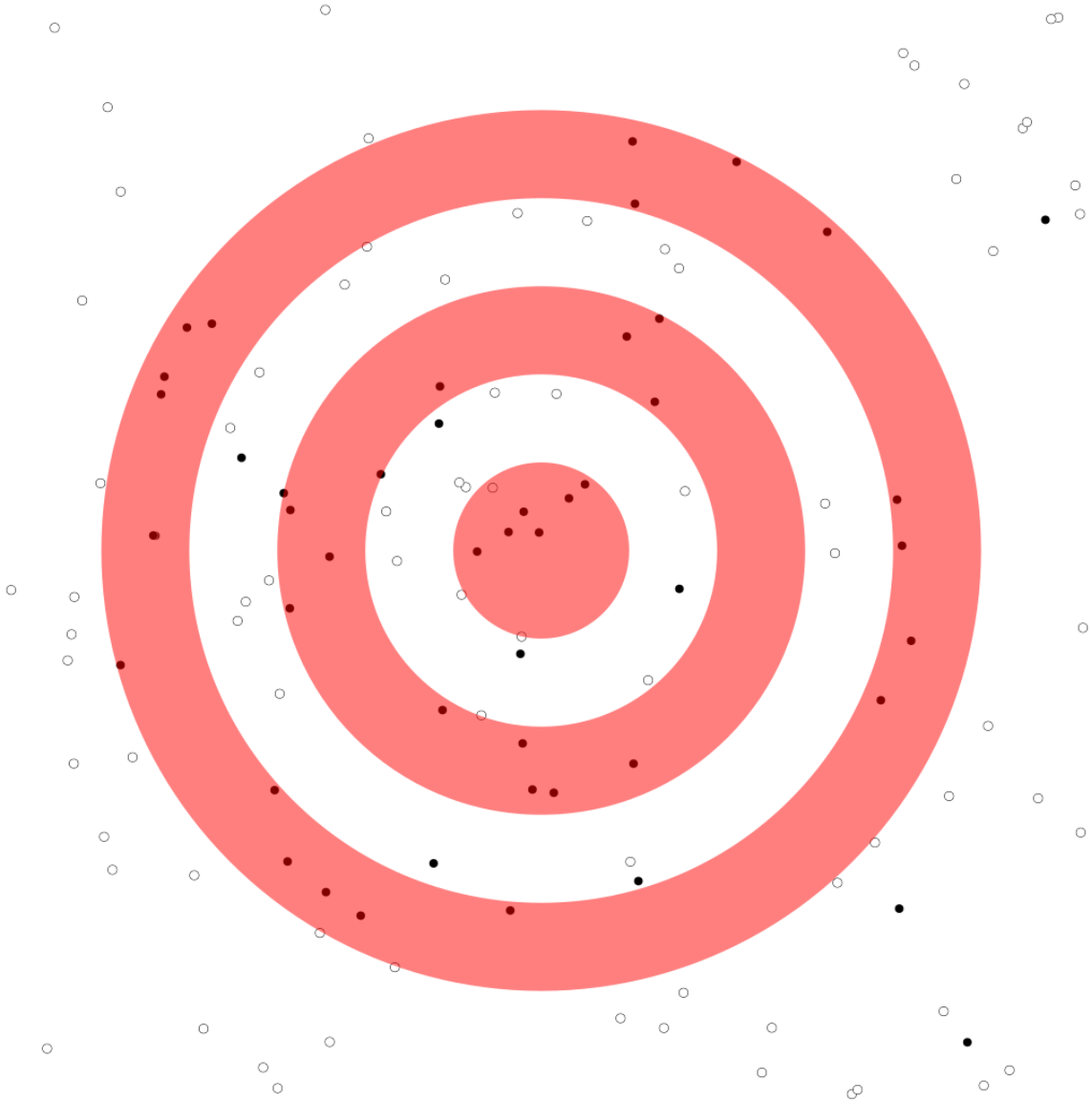


Training points



Votes (K=11)

Prediction (K=11)



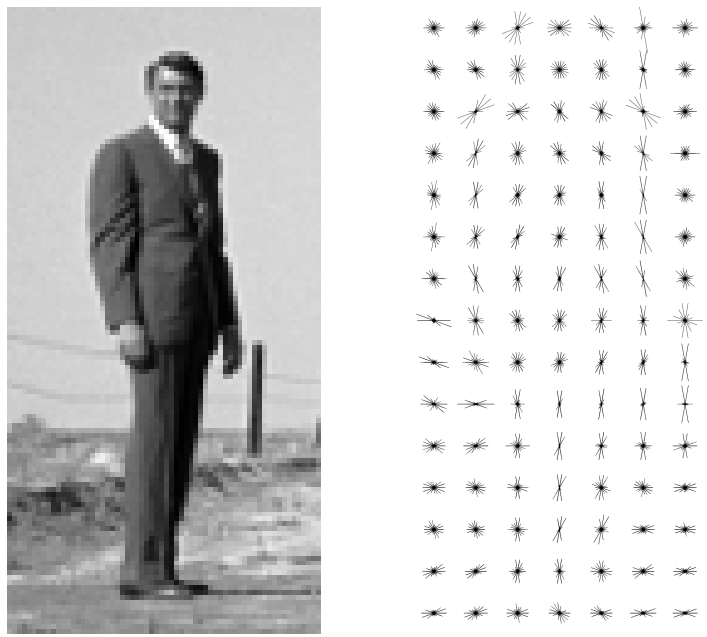
Training points

Votes, radial feature (K=11)

Prediction, radial feature (K=11)

A classical example is the "Histogram of Oriented Gradient" descriptors (HOG), initially designed for person detection.

Roughly: divide the image in 8×8 blocks, compute in each the distribution of edge orientations over 9 bins.



Dalal and Triggs (2005) combined them with a SVM, and Dollár et al. (2009) extended them with other modalities into the "channel features".

Many methods (perceptron, SVM, k-means, PCA, etc.) only require to compute $\kappa(x, x') = \Phi(x) \cdot \Phi(x')$ for any (x, x').

So one needs to specify κ alone, and may keep Φ undefined.

This is the **kernel trick**, which we will not talk about in this course.

Training a model composed of manually engineered features and a parametric model such as logistic regression is now referred to as **"shallow learning"**.

The signal goes through a single processing trained from data.

Deep learning

3.4. Multi-Layer Perceptrons

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

A linear classifier of the form

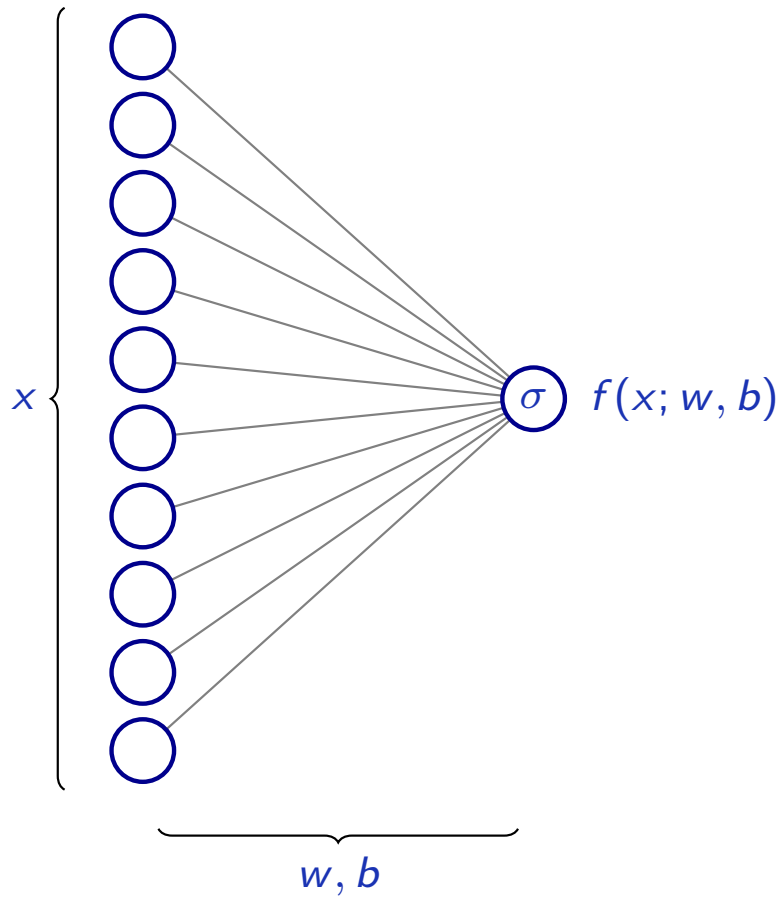
$$\begin{aligned} \mathbb{R}^D &\to \mathbb{R} \\ x &\mapsto \sigma(w \cdot x + b), \end{aligned}$$

with $w \in \mathbb{R}^D$, $b \in \mathbb{R}$, and $\sigma : \mathbb{R} \to \mathbb{R}$, can naturally be extended to a multi-dimension output by applying a similar transformation to every output

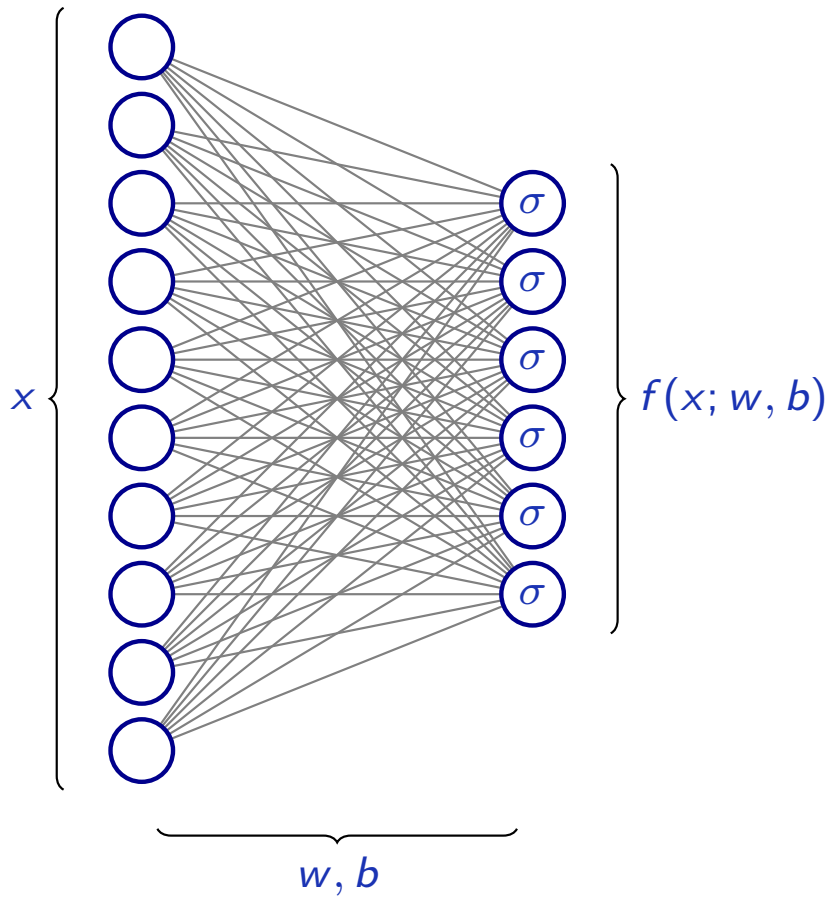
$$\begin{aligned} \mathbb{R}^D &\to \mathbb{R}^C \\ x &\mapsto \sigma(wx + b), \end{aligned}$$

with $w \in \mathbb{R}^{C \times D}$, $b \in \mathbb{R}^C$, and σ is applied component-wise.

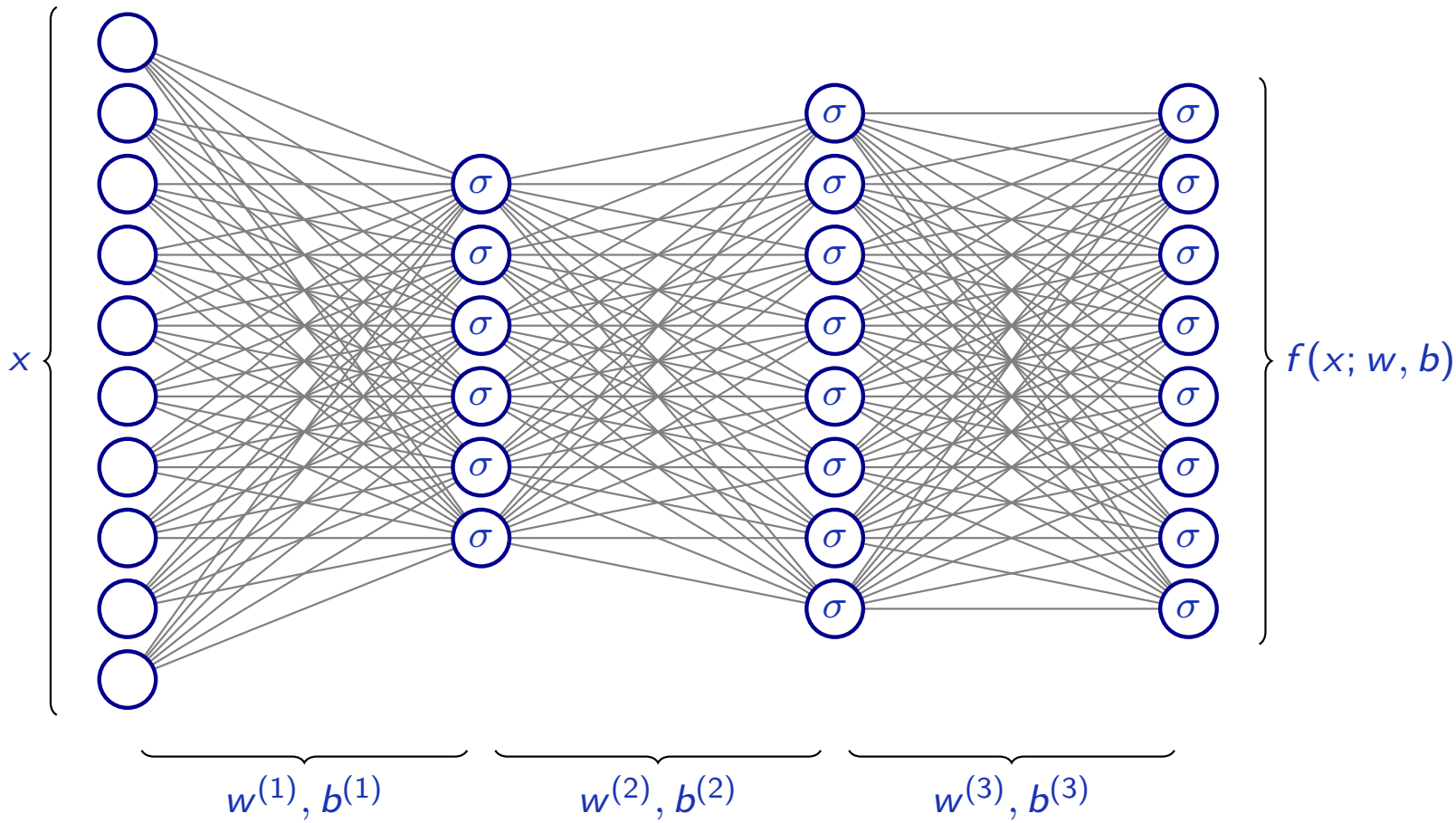
Even though it has no practical value implementation-wise, we can represent such a model as a combination of units. More importantly, we can extend it.



Even though it has no practical value implementation-wise, we can represent such a model as a combination of units. More importantly, we can extend it.



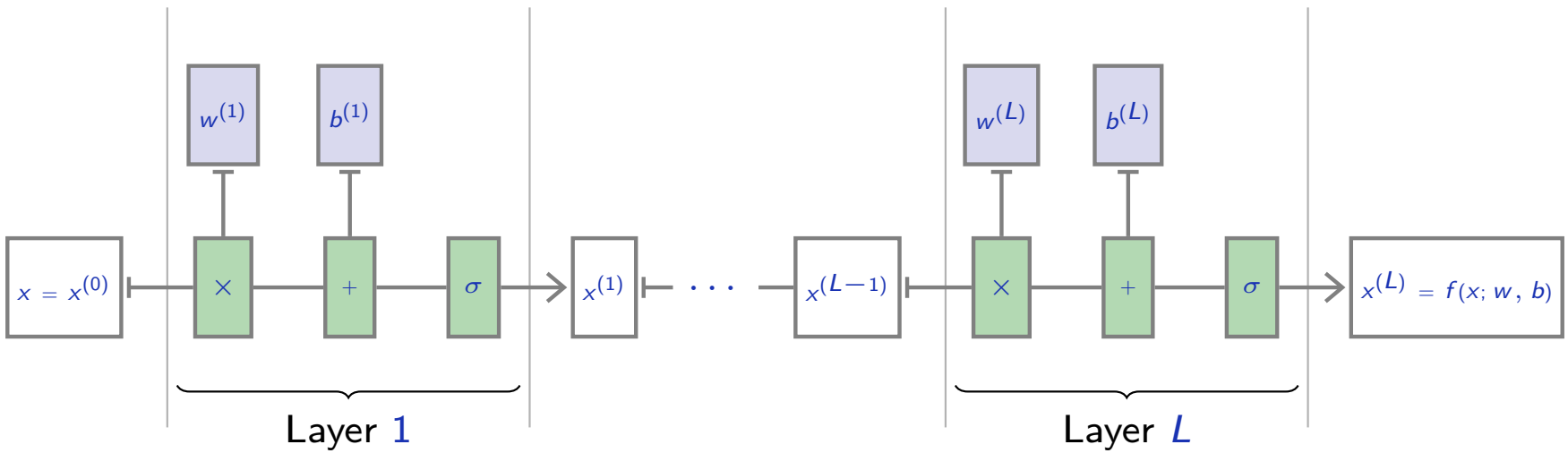
Even though it has no practical value implementation-wise, we can represent such a model as a combination of units. More importantly, we can extend it.



This latter structure can be formally defined, with $x^{(0)} = x$,

$$\forall l = 1, \ldots, L, \; x^{(l)} = \sigma\left(w^{(l)} x^{(l-1)} + b^{(l)}\right)$$

and $f(x; w, b) = x^{(L)}$.



Such a model is a **Multi-Layer Perceptron (MLP).**

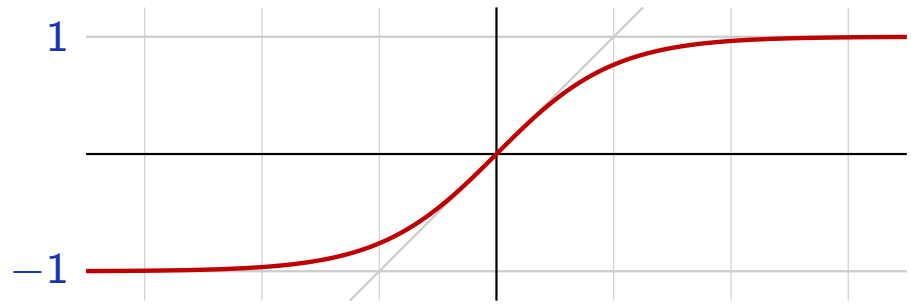
Note that if σ is an affine transformation, the full MLP is a composition of affine mappings, and itself an affine mapping.

Consequently:

The activation function σ should not be affine. Otherwise the resulting MLP would be an affine mapping with a peculiar parametrization.

The two classical activation functions are the hyperbolic tangent

$$x \mapsto \frac{2}{1 + e^{-2x}} - 1$$

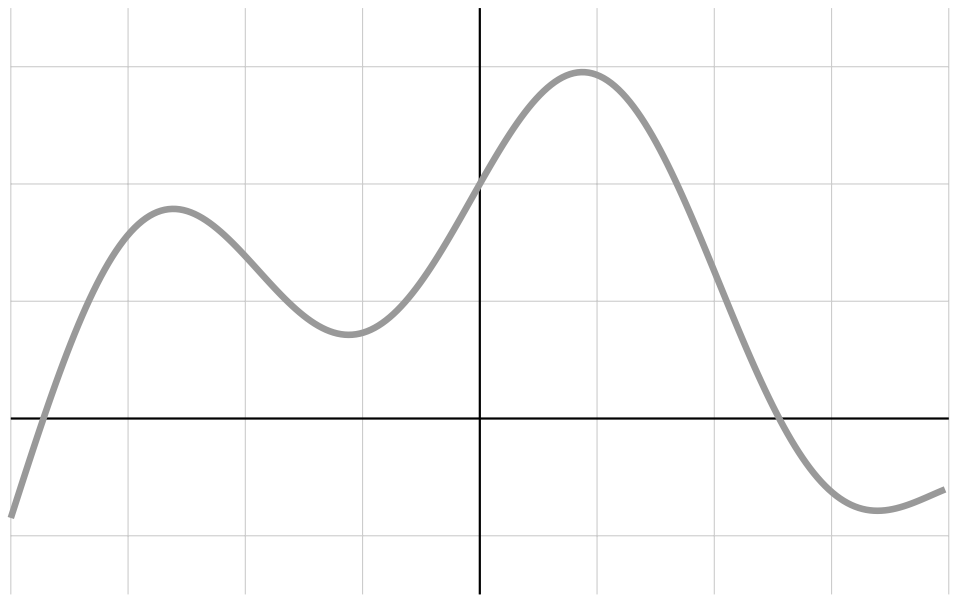


and the rectified linear unit (ReLU, Glorot et al., 2011)

$$x \mapsto \max(0, x)$$

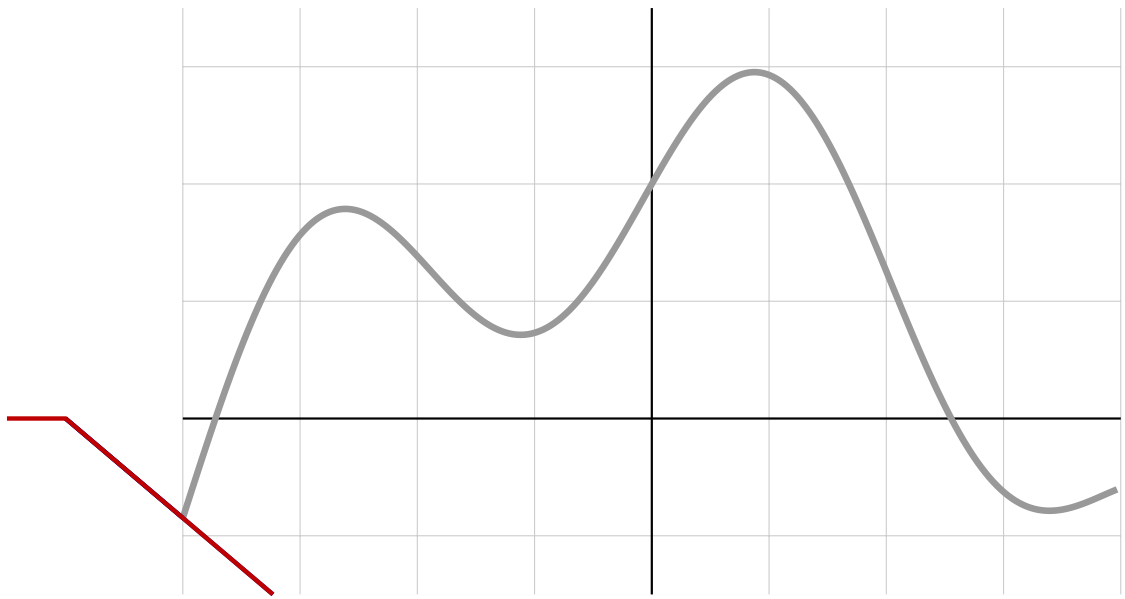
Universal approximation

We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.



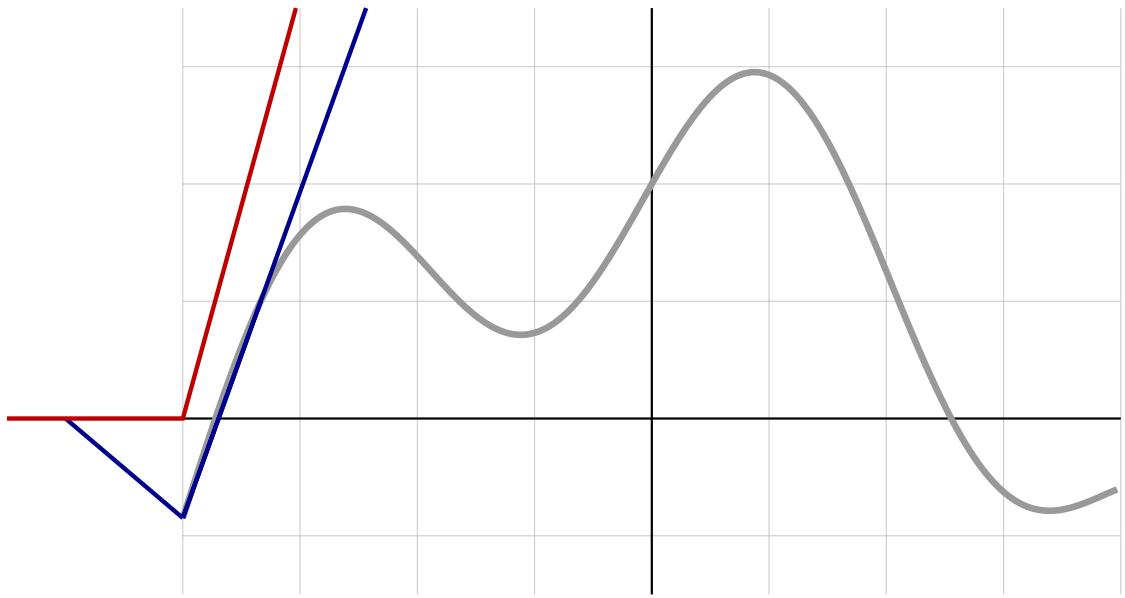
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1)$$



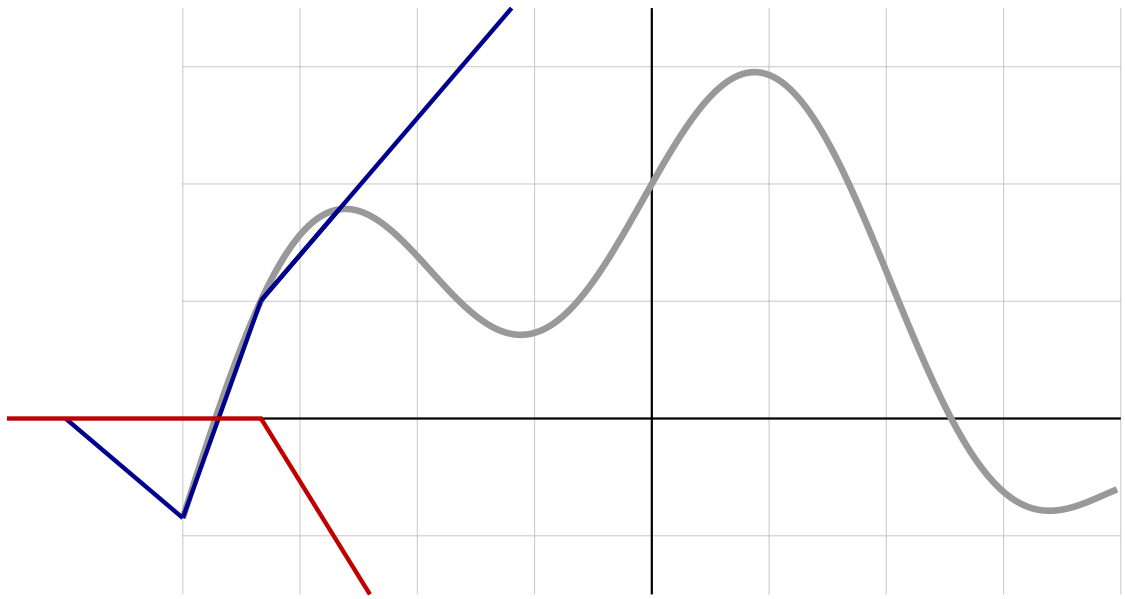
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2)$$



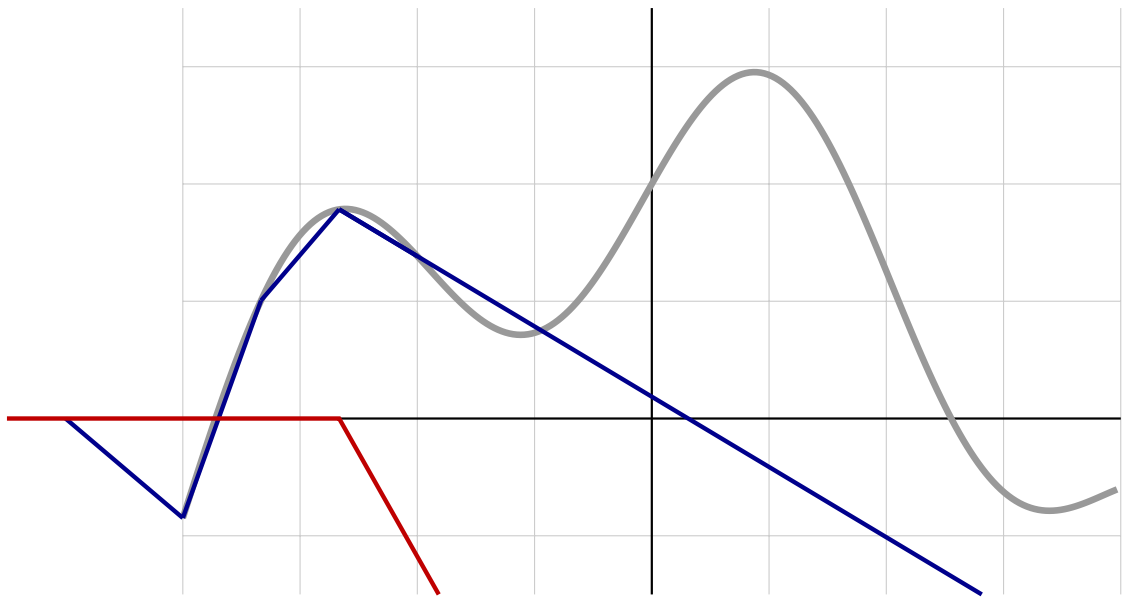
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3)$$



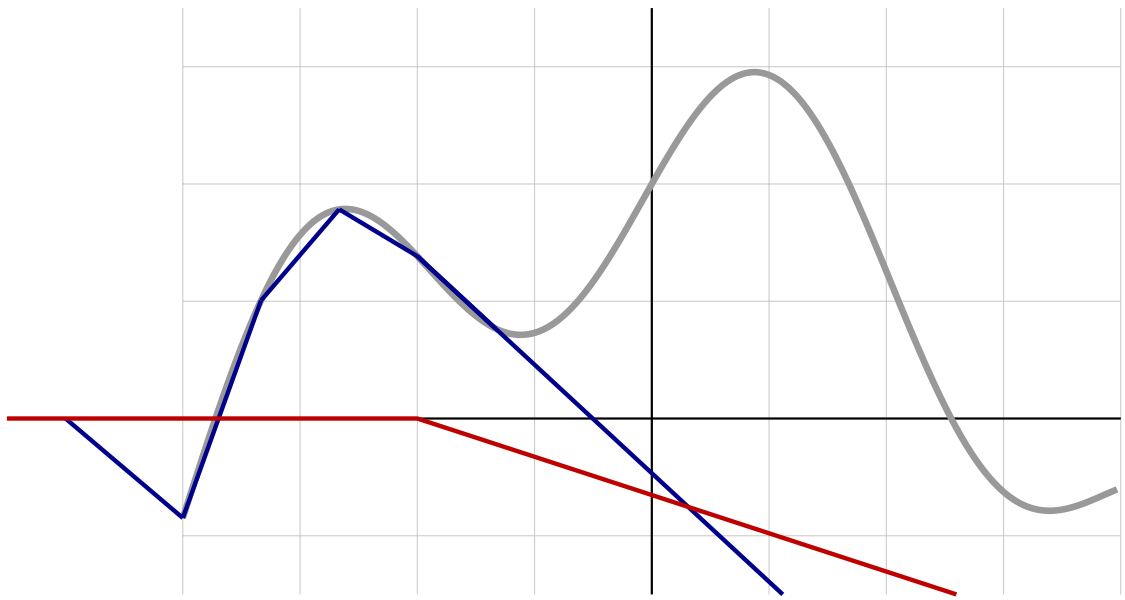
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



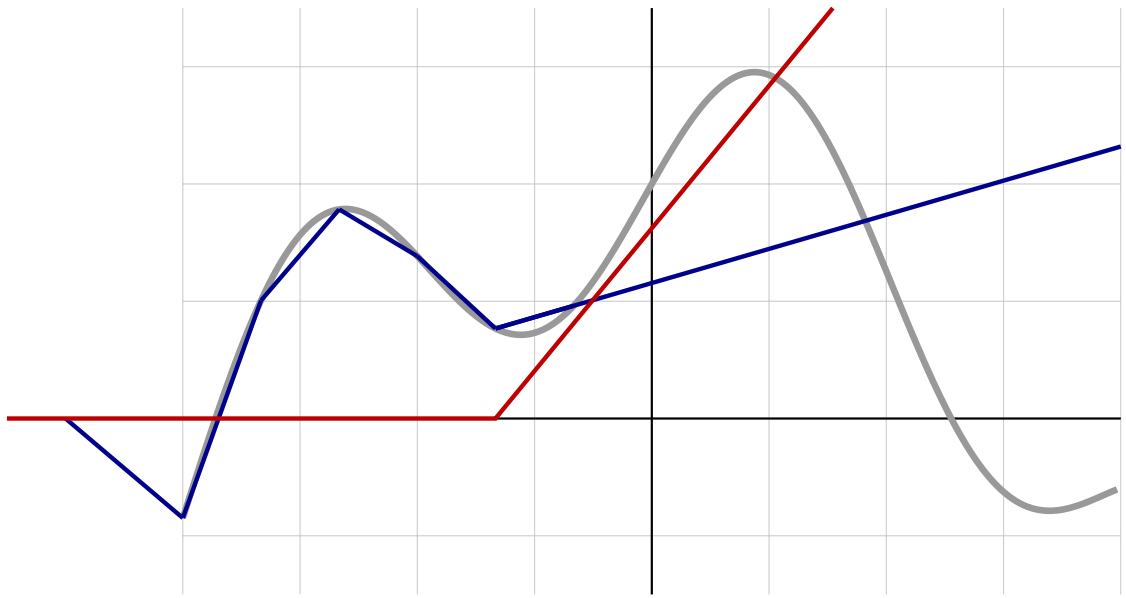
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



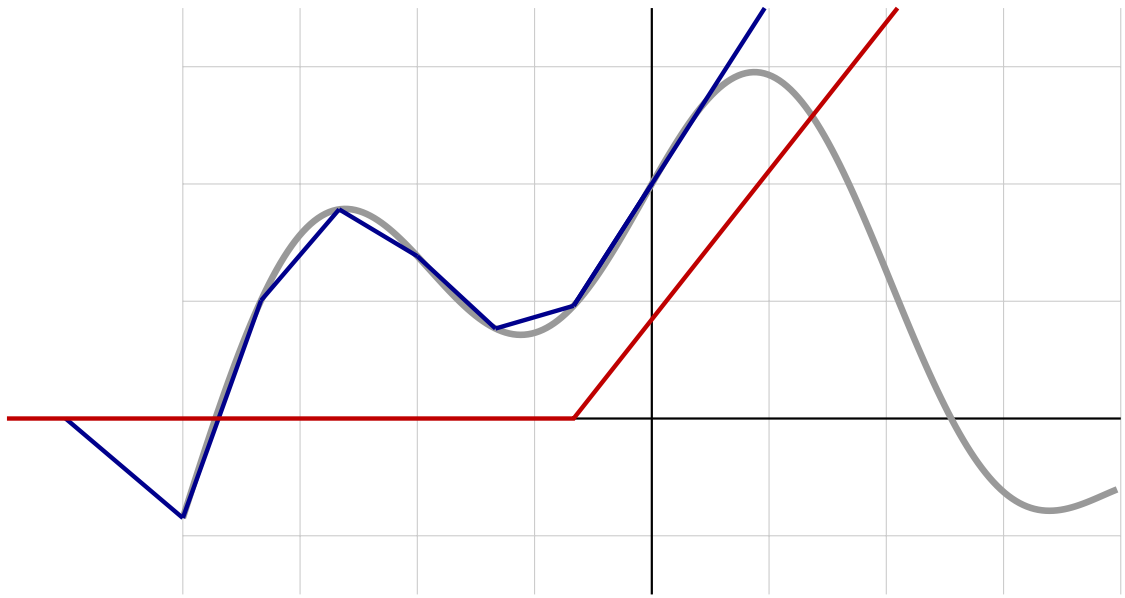
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



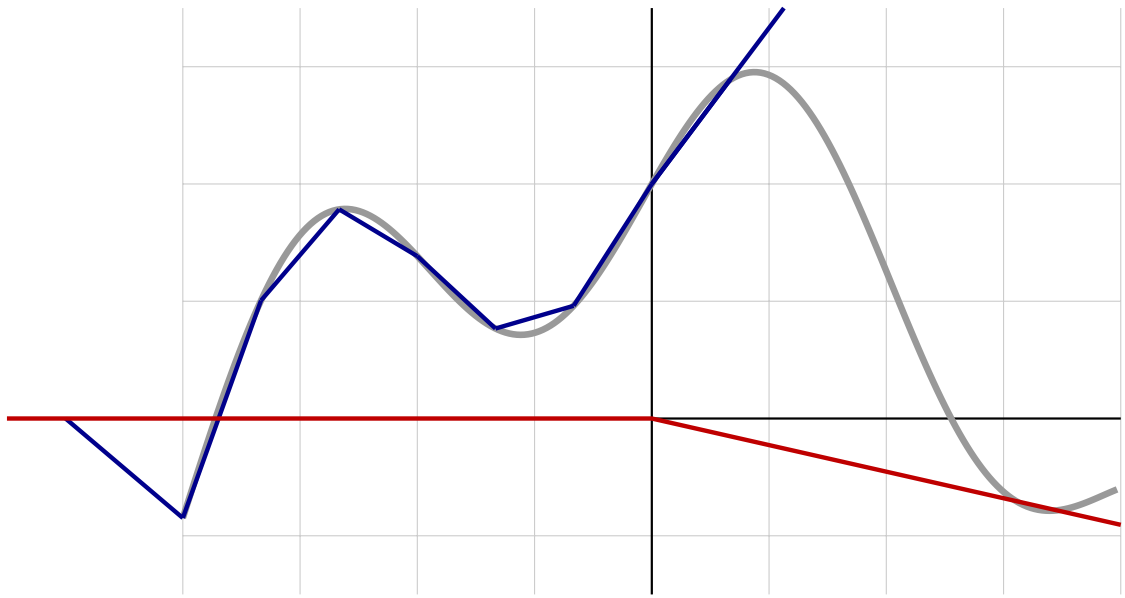
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



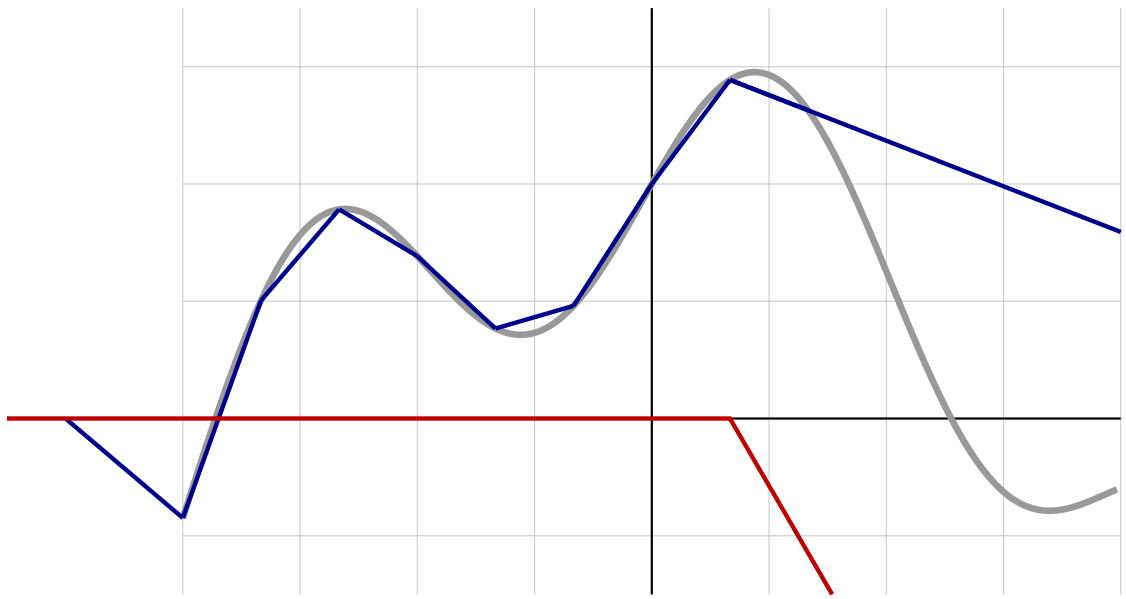
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



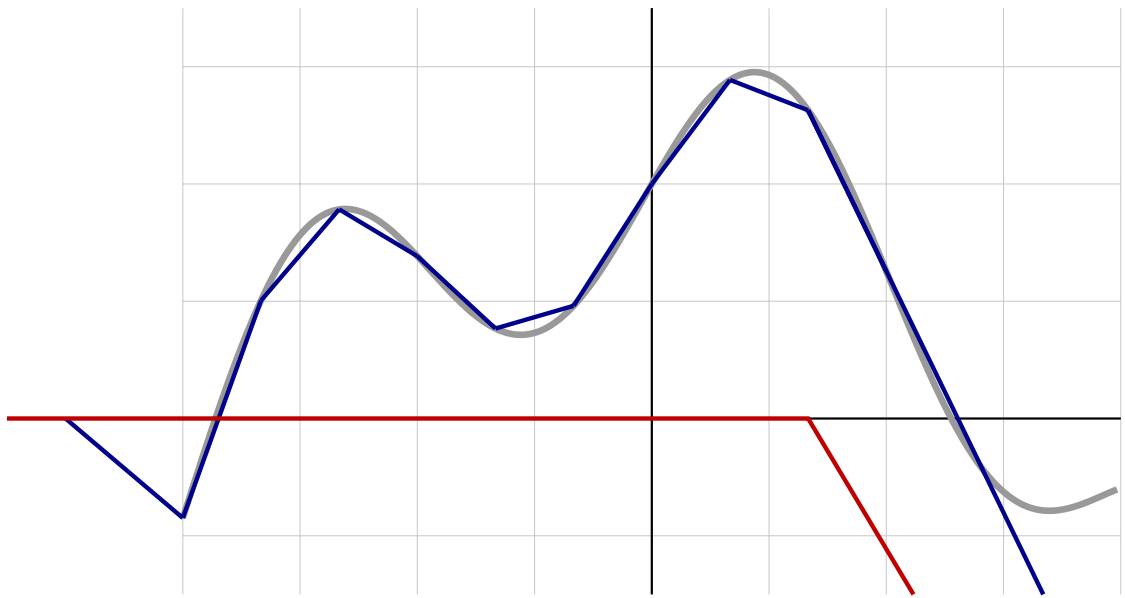
We can approximate any $\psi \in \mathscr{C}([a,b],\mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



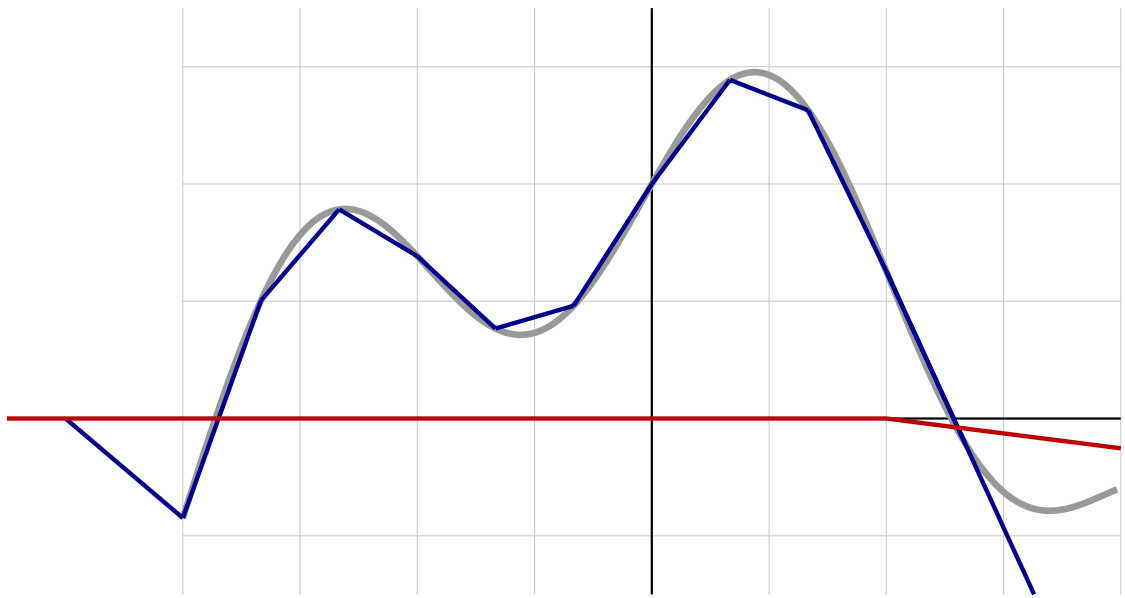
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$



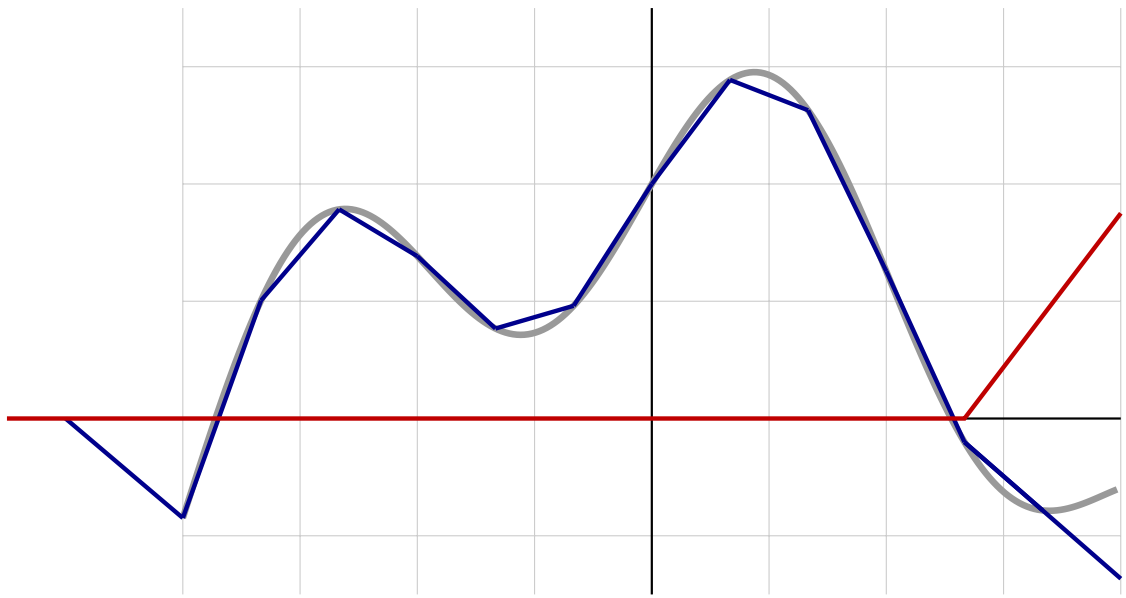
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



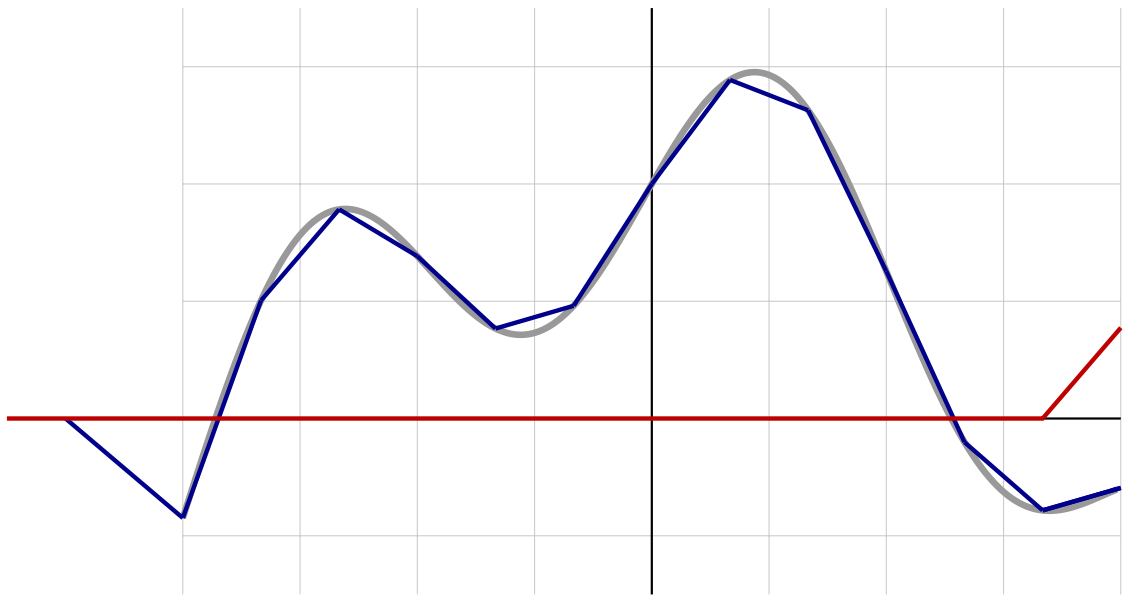
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



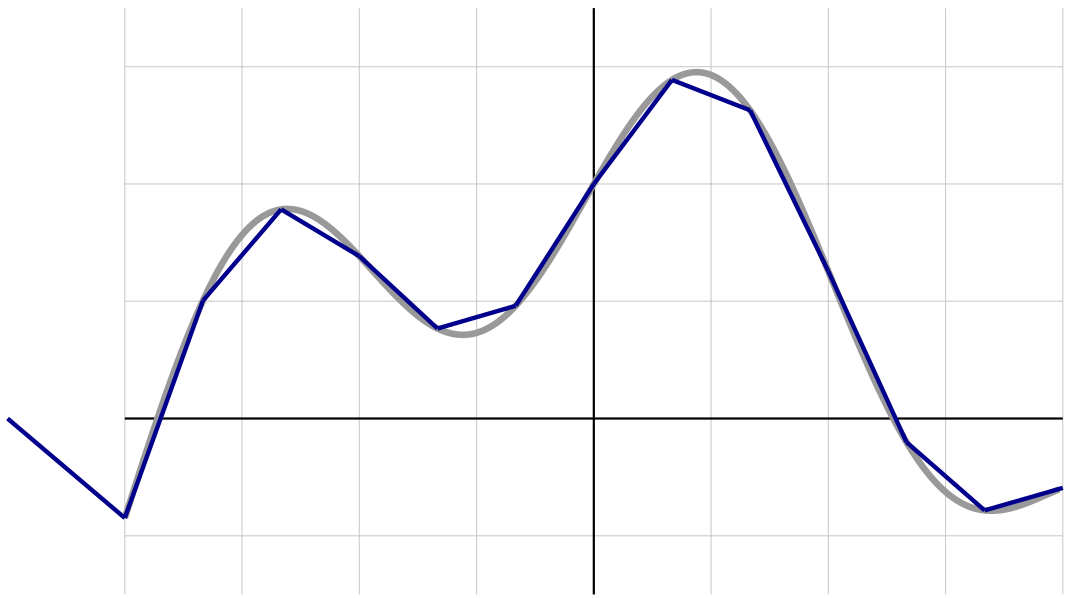
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



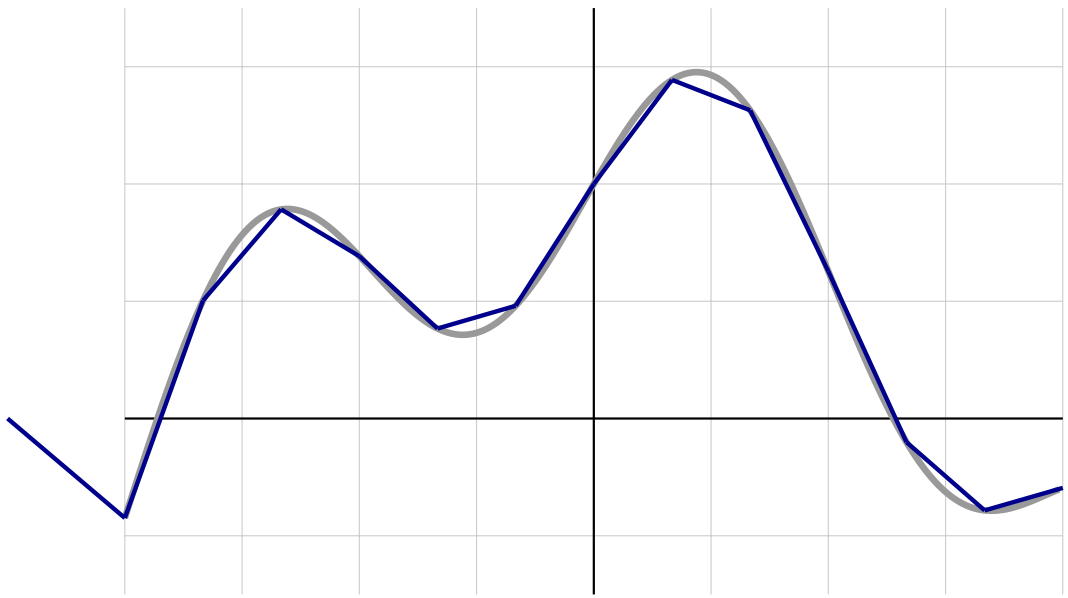
We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$



We can approximate any $\psi \in \mathscr{C}([a, b], \mathbb{R})$ with a linear combination of translated/scaled ReLU functions.

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



This is true for other activation functions under mild assumptions.

Extending this result to any $\psi \in \mathscr{C}([0,1]^D, \mathbb{R})$ requires a bit of work.

We can approximate the sin function with the previous scheme, and use the density of Fourier series to get the final result:

$$\forall \epsilon > 0, \exists K, w \in \mathbb{R}^{K \times D}, b \in \mathbb{R}^K, \omega \in \mathbb{R}^K, \text{ s.t.}$$

$$\max_{x \in [0,1]^D} |\psi(x) - \omega \cdot \sigma(w\,x + b)| \leq \epsilon.$$

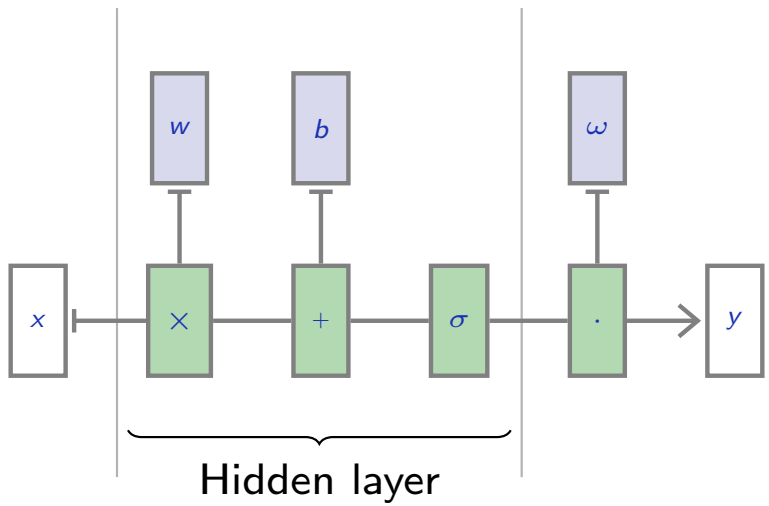
So we can approximate any continuous function

$$\psi : [0,1]^D \to \mathbb{R}$$

with a one hidden layer perceptron

$$x \mapsto \omega \cdot \sigma(w\,x + b),$$

where $b \in \mathbb{R}^K$, $w \in \mathbb{R}^{K \times D}$, and $\omega \in \mathbb{R}^K$.



This is the **universal approximation theorem.**

A better approximation requires a larger hidden layer (larger K), and this theorem says nothing about the relation between the two.

So this results states that we can make the **training error** as low as we want by using a larger hidden layer. It states nothing about the **test error**.

Deploying MLP in practice is often a balancing act between under-fitting and over-fitting.

Deep learning

3.5. Gradient descent

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

We saw that training consists of finding the model parameters minimizing an empirical risk or loss, for instance the mean-squared error (MSE)

$$\mathscr{L}(w, b) = \frac{1}{N} \sum_n \left(f(x_n; w, b) - y_n\right)^2 .$$

Other losses are more fitting for classification, certain regression problems, or density estimation. We will come back to this.

So far we minimized the loss either with an analytic solution for the MSE, or with *ad hoc* recipes for the empirical error rate (k-NN and perceptron).

There is generally no *ad hoc* method. The logistic regression for instance

$$P_w(Y = 1 \mid X = x) = \sigma(w \cdot x + b), \text{ with } \sigma(x) = \frac{1}{1 + e^{-x}}$$

leads to the loss

$$\mathscr{L}(w, b) = -\sum_n \log \sigma(y_n(w \cdot x_n + b))$$

which cannot be minimized analytically.

The general minimization method used in such a case is the **gradient descent**.

Given a functional

$$\begin{aligned} f : \mathbb{R}^D &\to \mathbb{R} \\ x &\mapsto f(x_1, \ldots, x_D), \end{aligned}$$

its gradient is the mapping

$$\begin{aligned} \nabla f : \mathbb{R}^D &\to \mathbb{R}^D \\ x &\mapsto \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_D}(x) \right). \end{aligned}$$

To minimize a functional

$$\mathscr{L} : \mathbb{R}^D \to \mathbb{R}$$

the gradient descent uses local linear information to iteratively move toward a (local) minimum.

For $w_0 \in \mathbb{R}^D$, consider an approximation of $\mathscr{L}$ around w_0

$$\tilde{\mathscr{L}}_{w_0}(w) = \mathscr{L}(w_0) + \nabla\mathscr{L}(w_0)^\top(w - w_0) + \frac{1}{2\eta}\|w - w_0\|^2.$$

Note that the chosen quadratic term does not depend on $\mathscr{L}$.

We have

$$\nabla\tilde{\mathscr{L}}_{w_0}(w) = \nabla\mathscr{L}(w_0) + \frac{1}{\eta}(w - w_0),$$

which leads to

$$\operatorname*{argmin}_w \tilde{\mathscr{L}}_{w_0}(w) = w_0 - \eta\nabla\mathscr{L}(w_0).$$

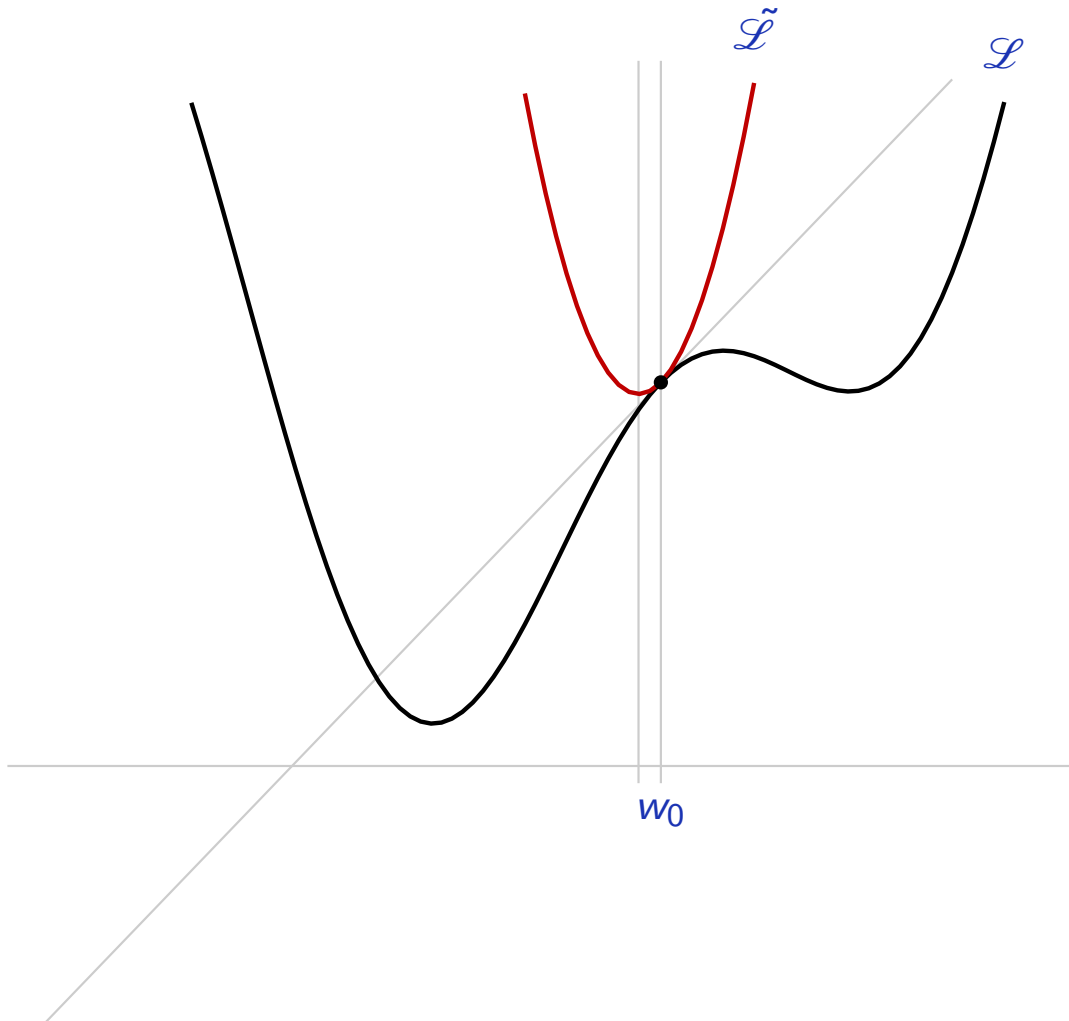
The resulting iterative rule, which goes to the minimum of the approximation at the current location, takes the form:

$$w_{t+1} = w_t - \eta \nabla \mathscr{L}(w_t),$$

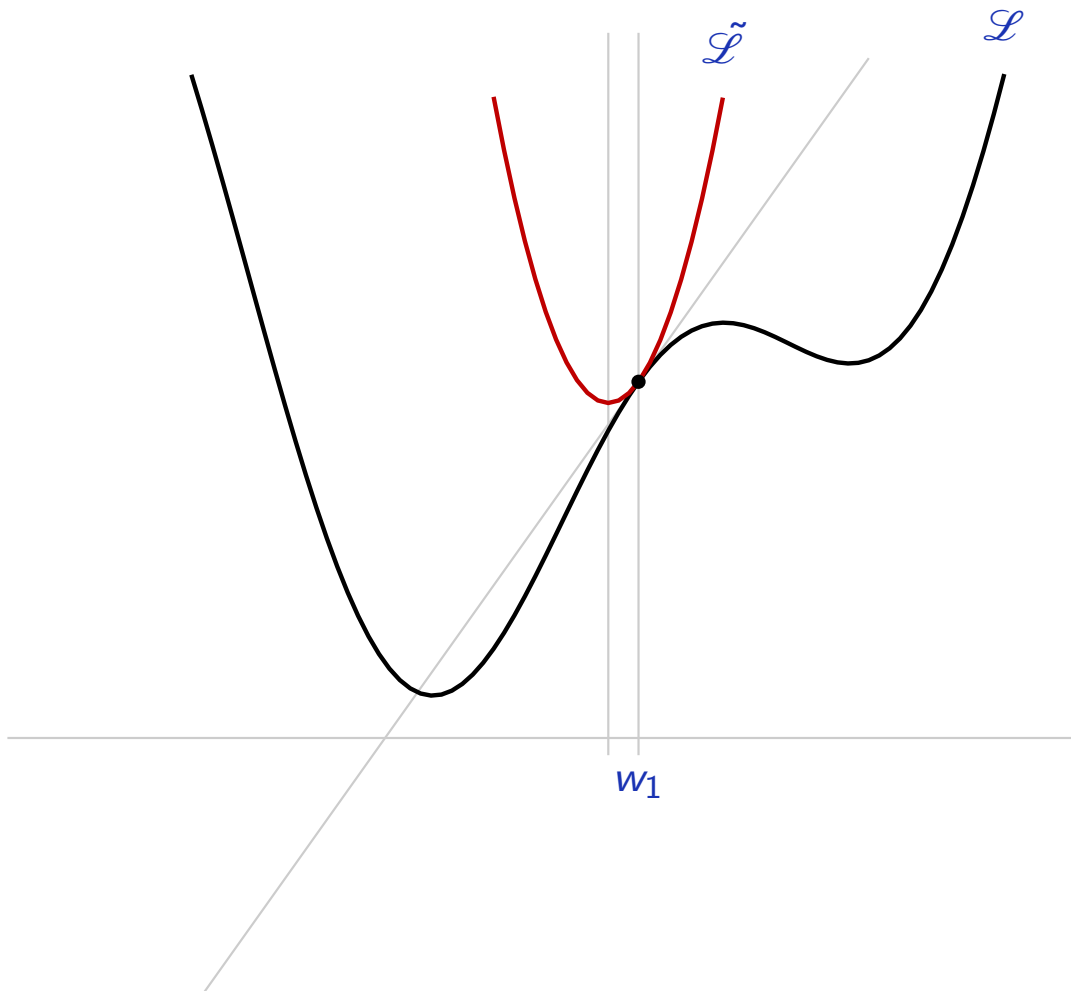
which corresponds intuitively to "following the steepest descent".

This [most of the time] eventually ends up in a **local** minimum, and the choices of w_0 and η are important.

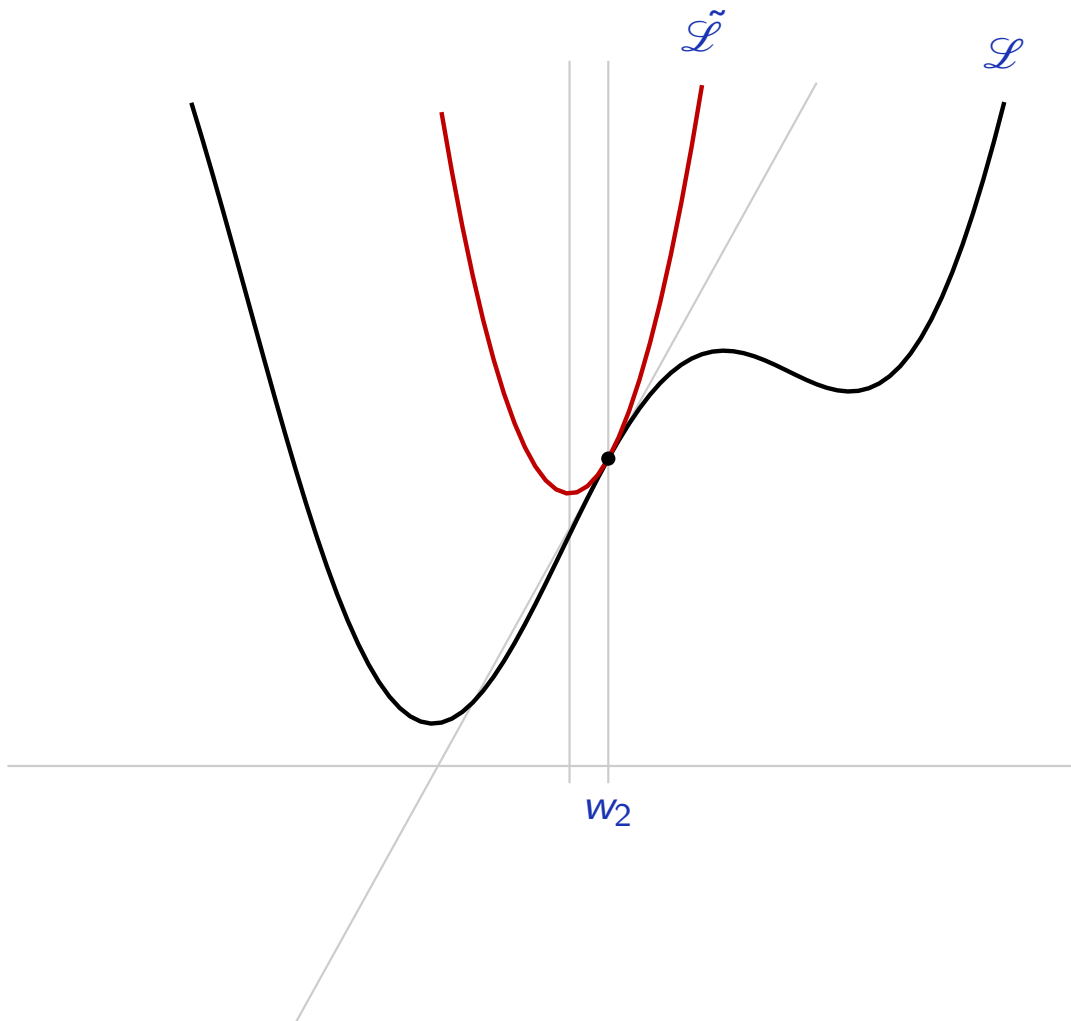
$\eta = 0.125$



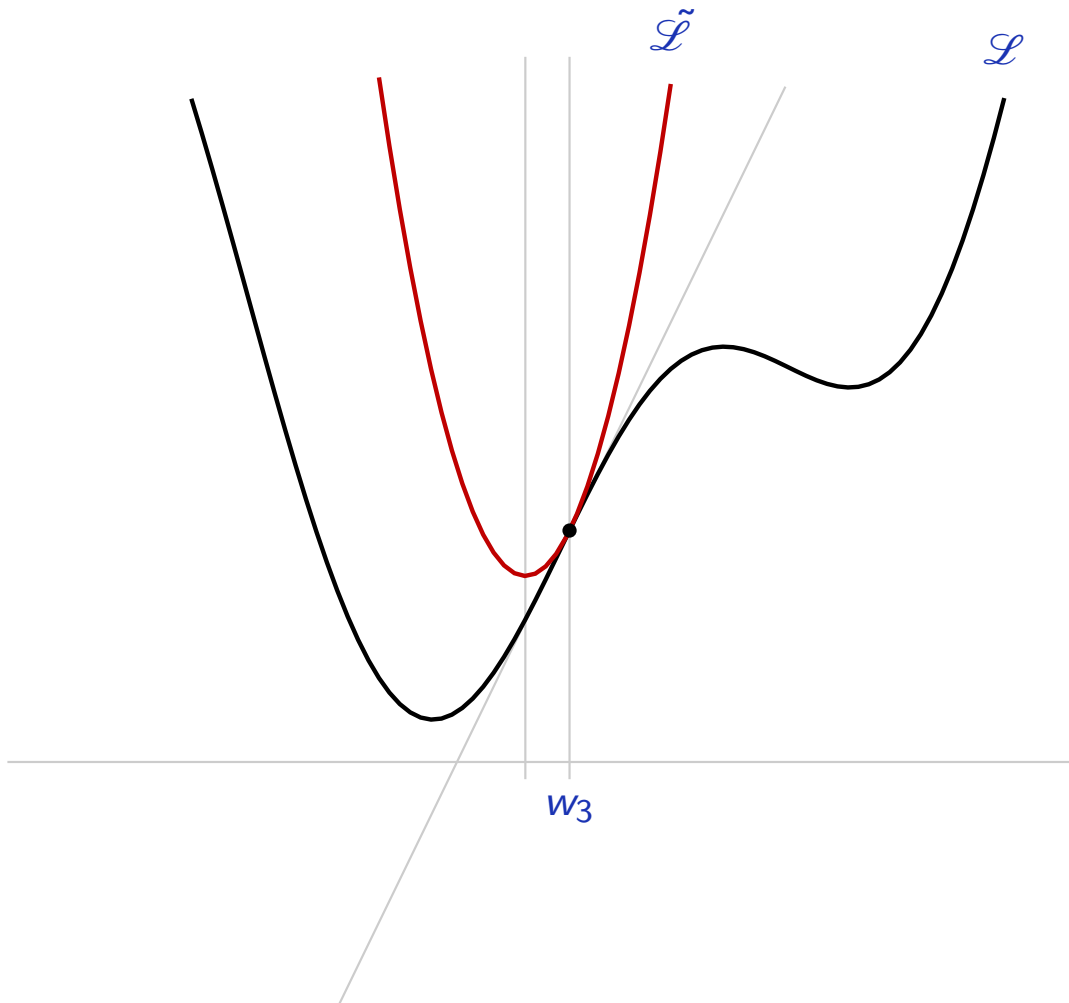
$\eta = 0.125$



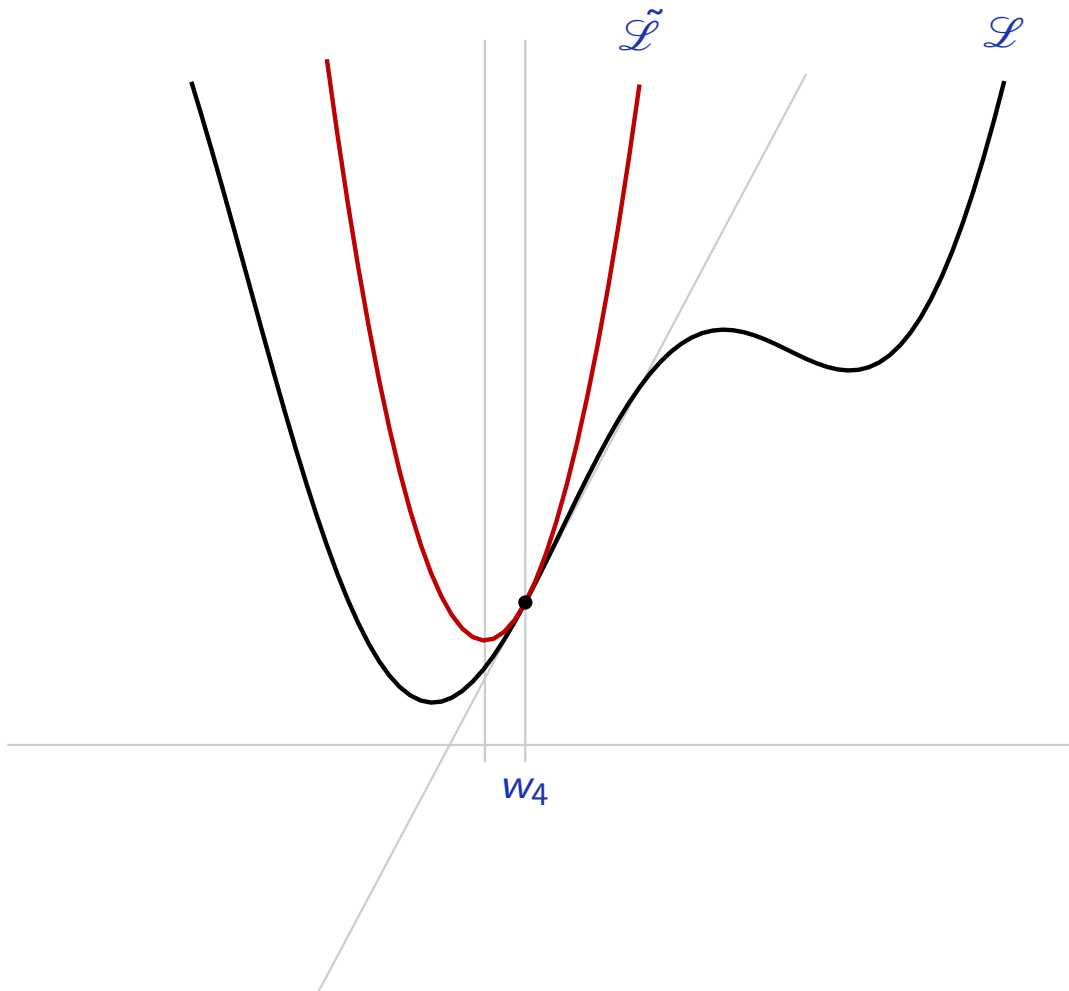
$\eta = 0.125$



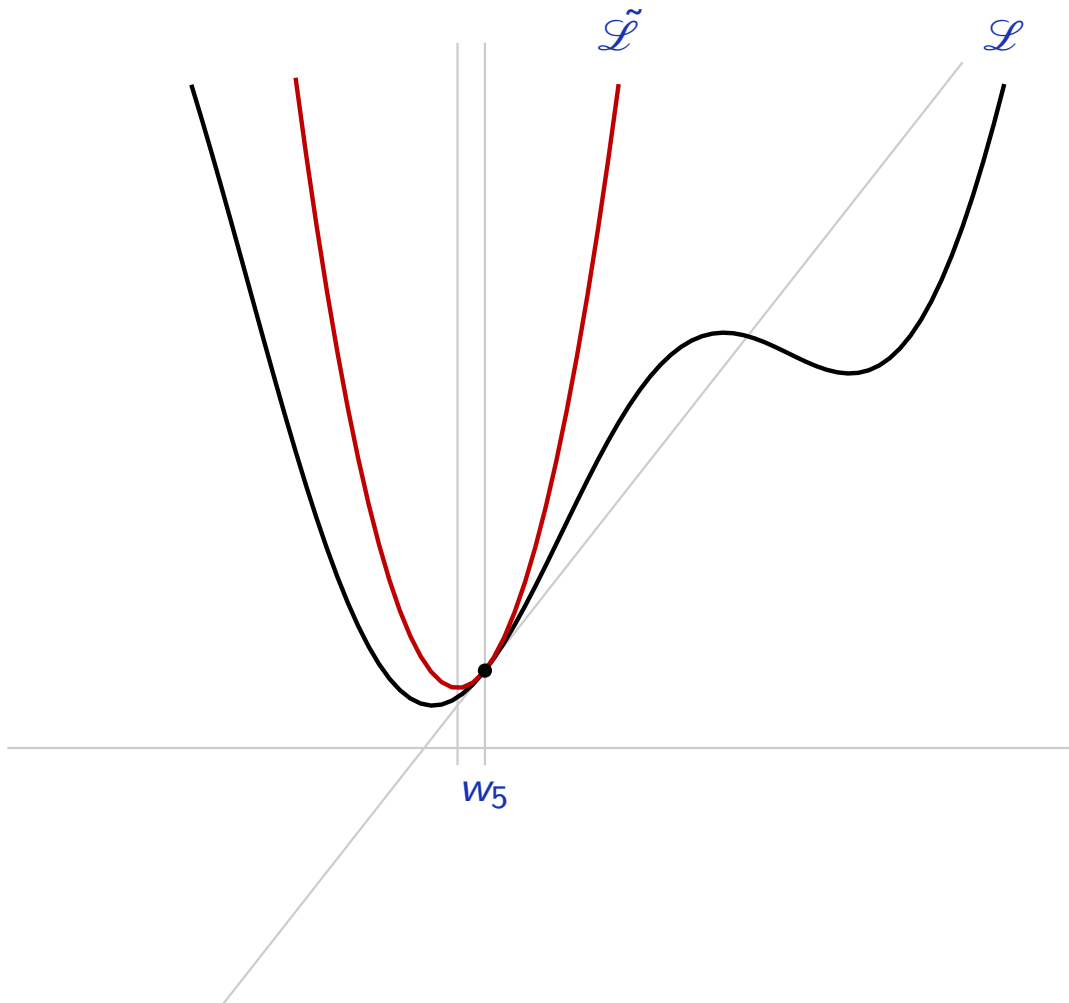
$\eta = 0.125$



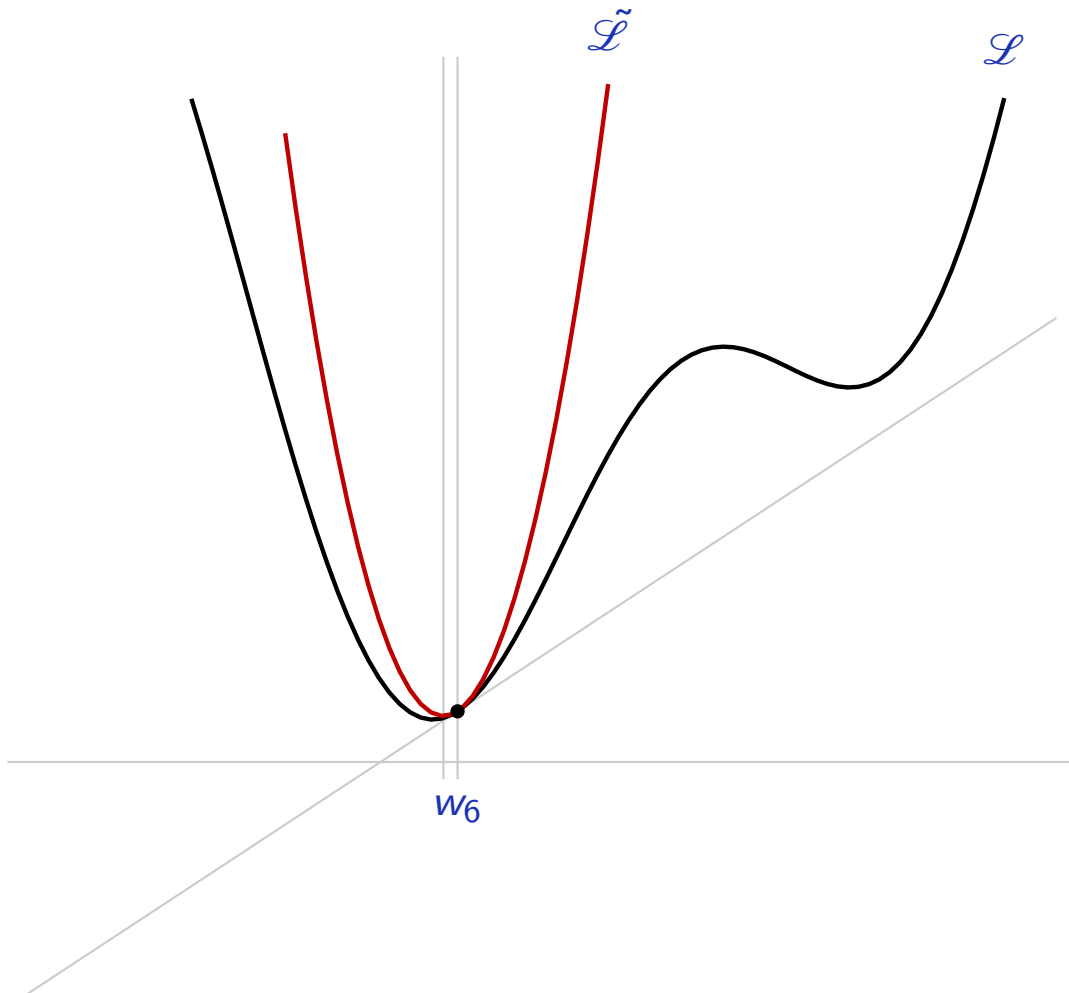
$\eta = 0.125$



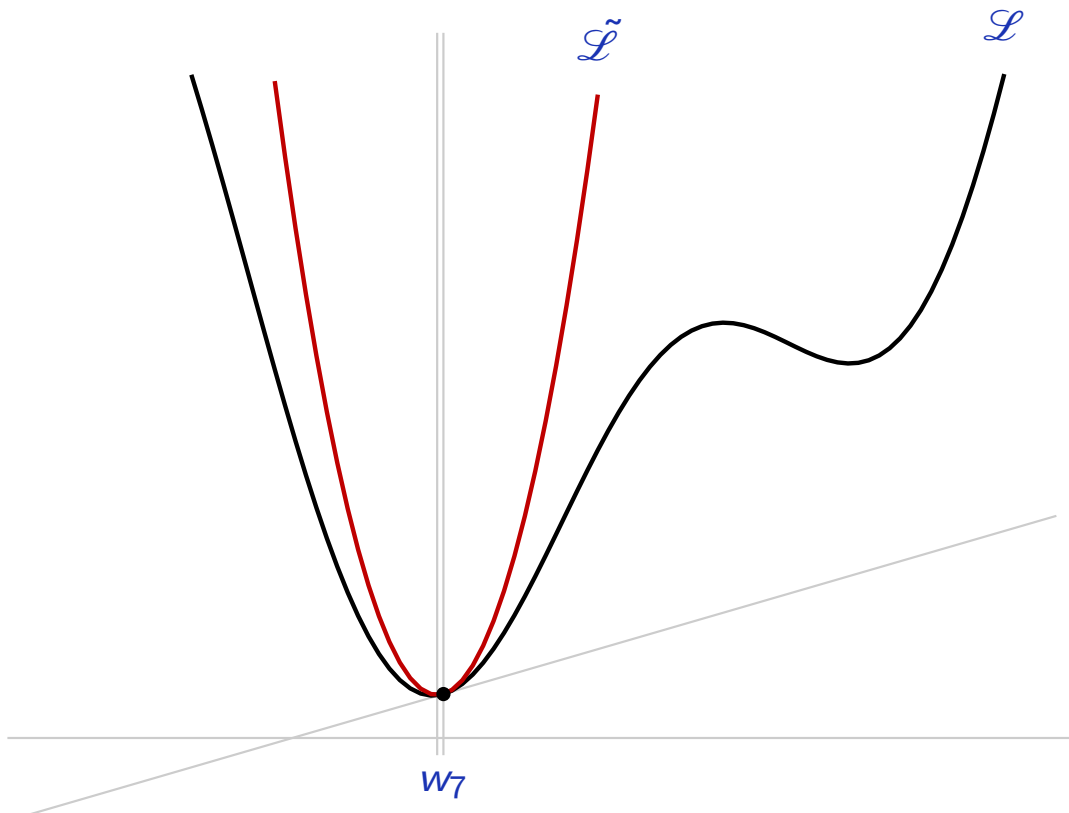
$\eta = 0.125$



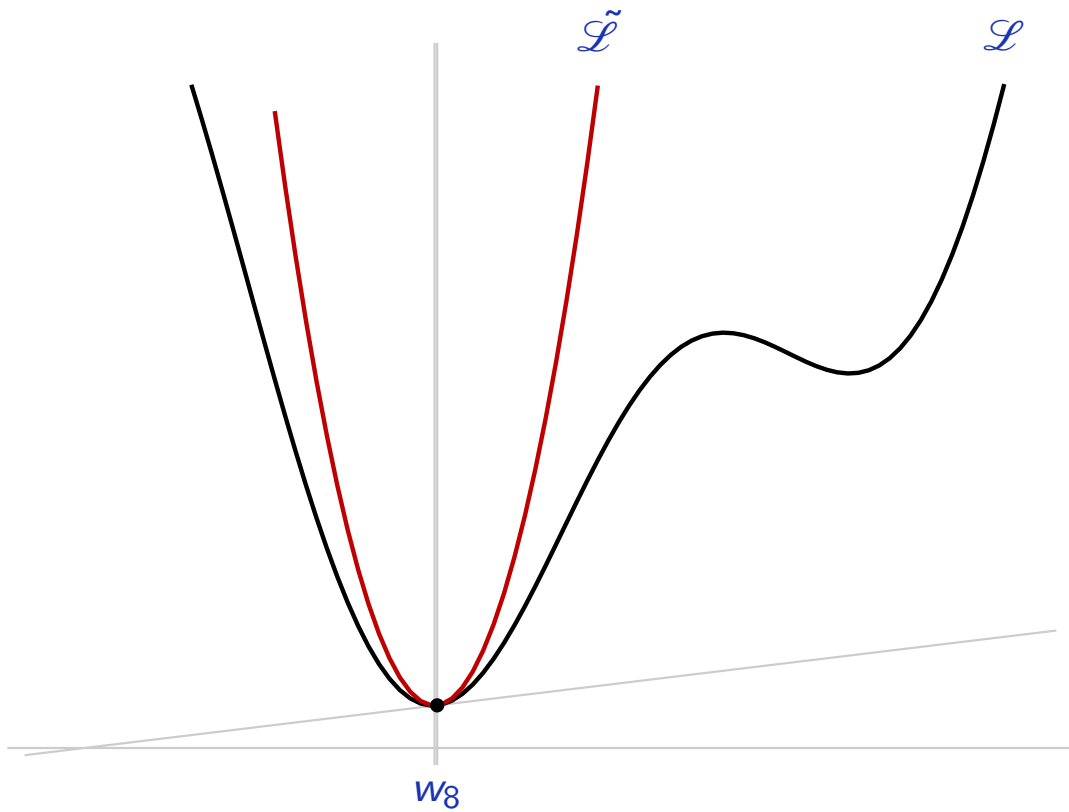
$\eta = 0.125$



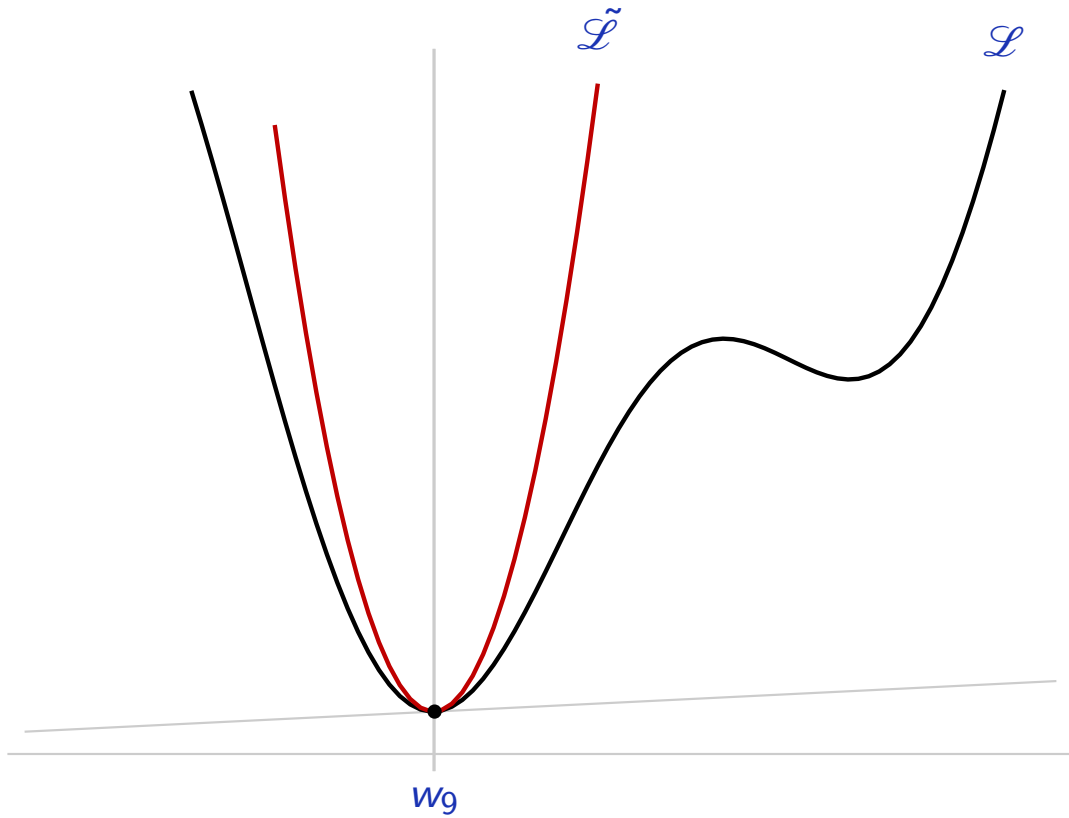
$\eta = 0.125$



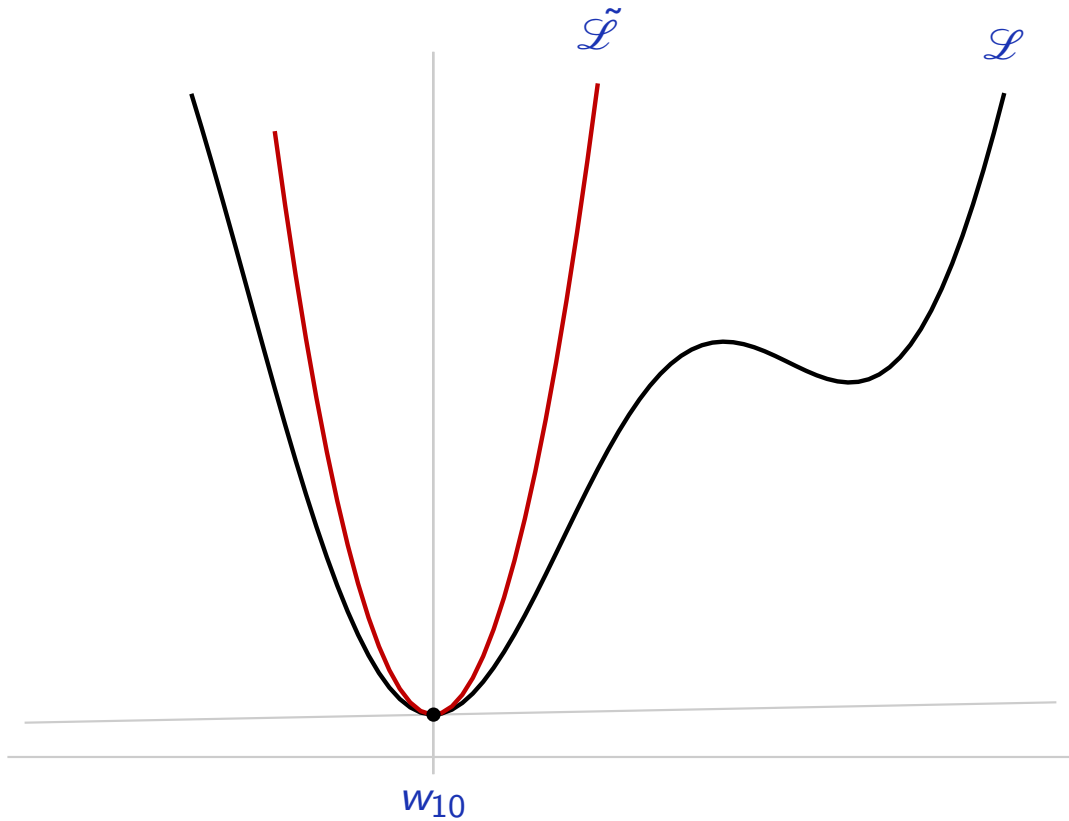
$\eta = 0.125$



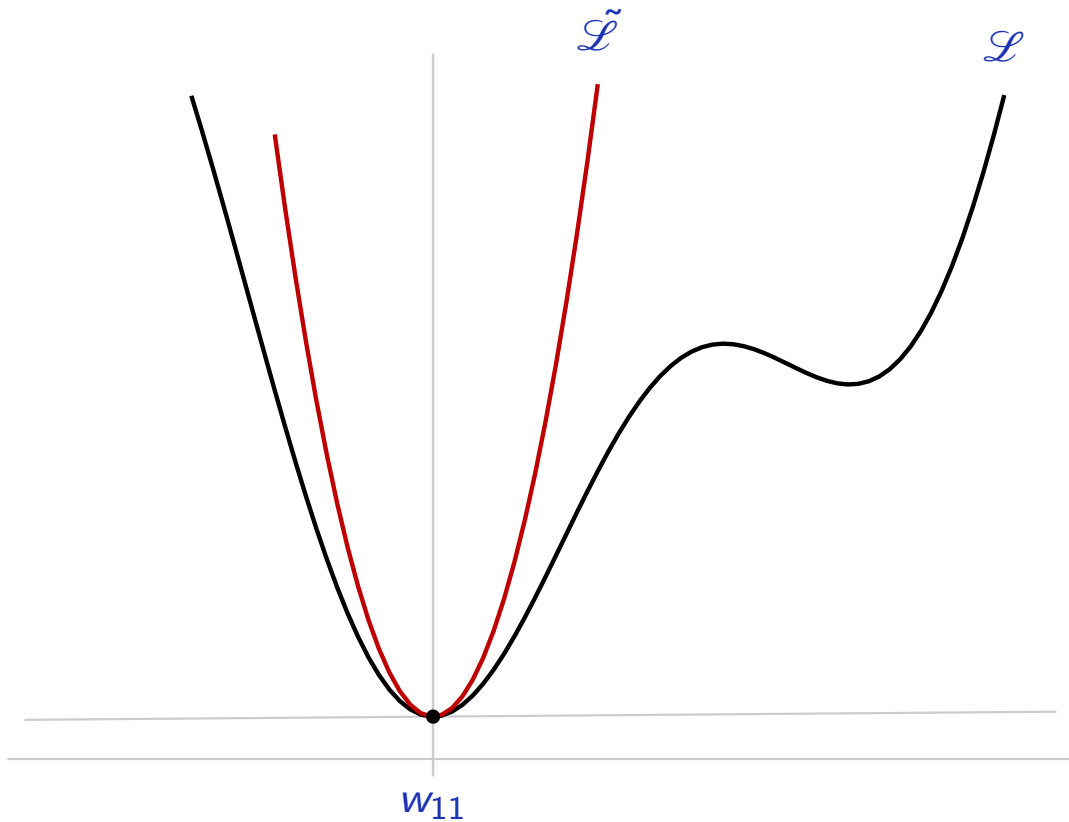
$\eta = 0.125$



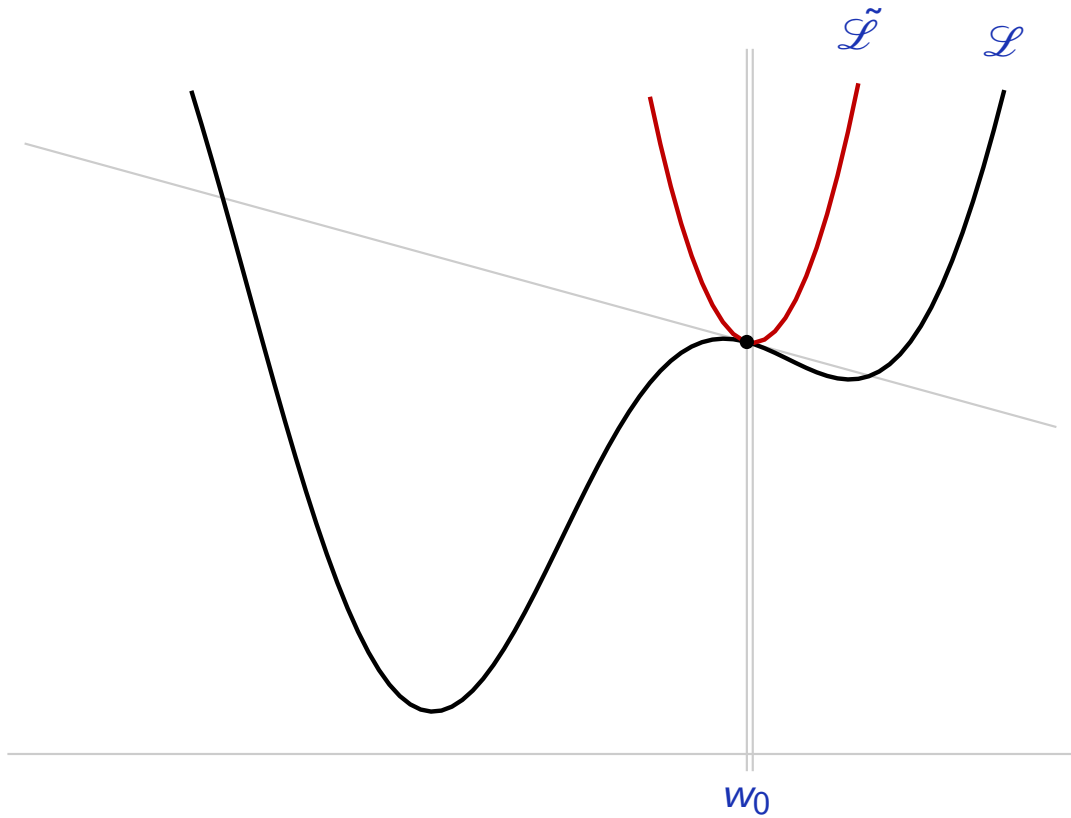
$\eta = 0.125$



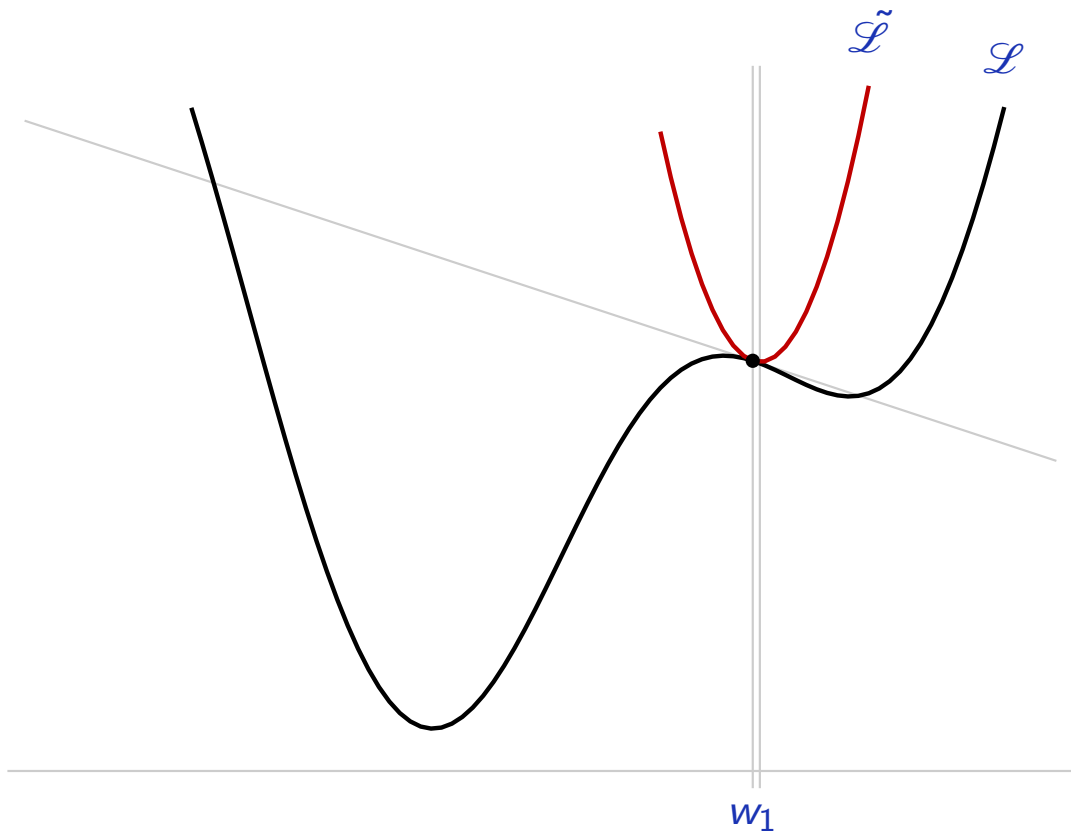
$\eta = 0.125$



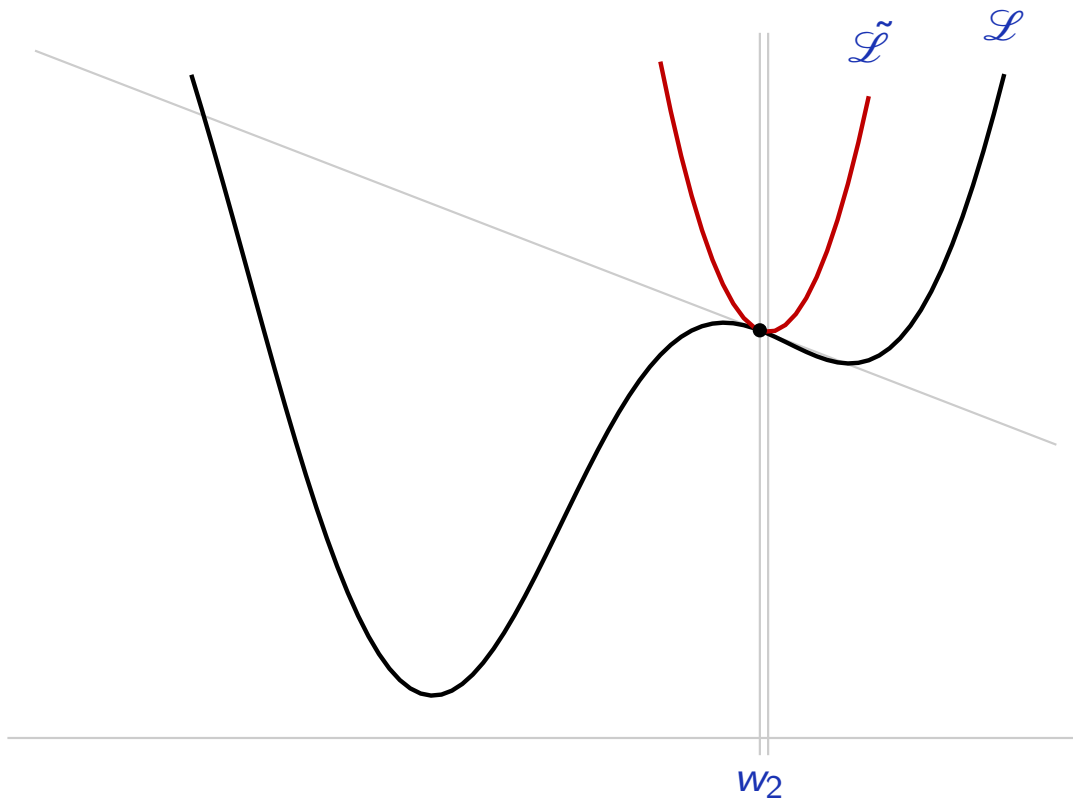
$\eta = 0.125$



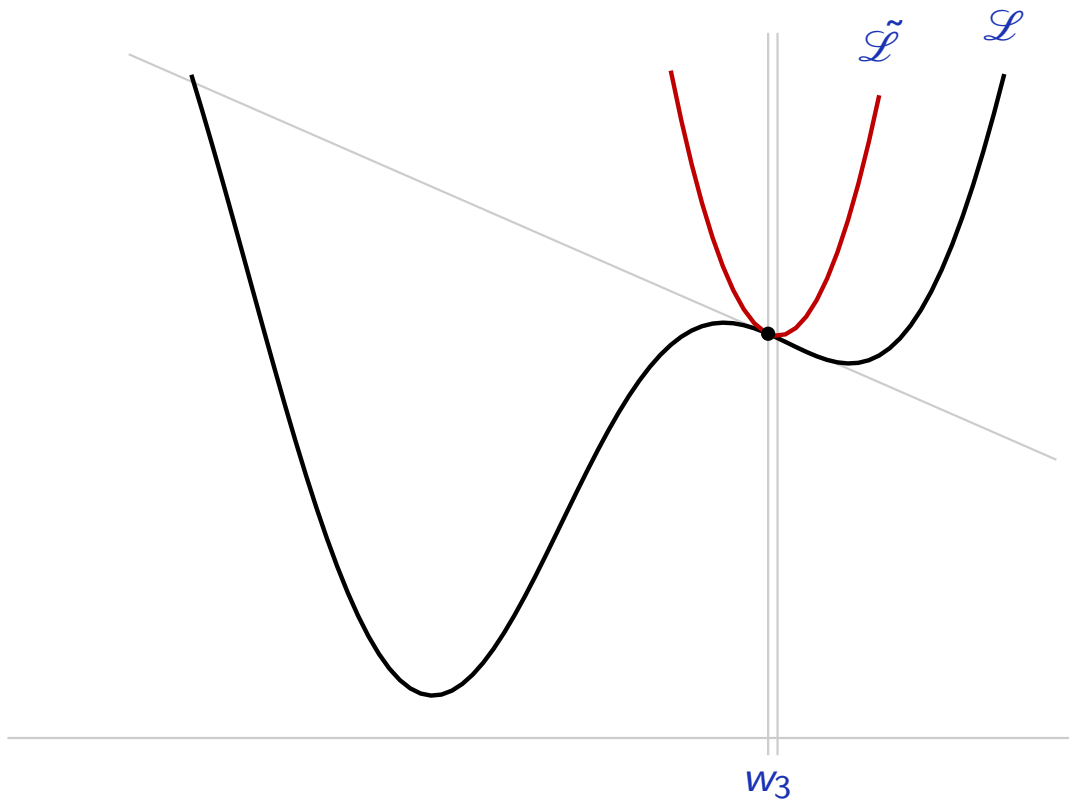
$\eta = 0.125$



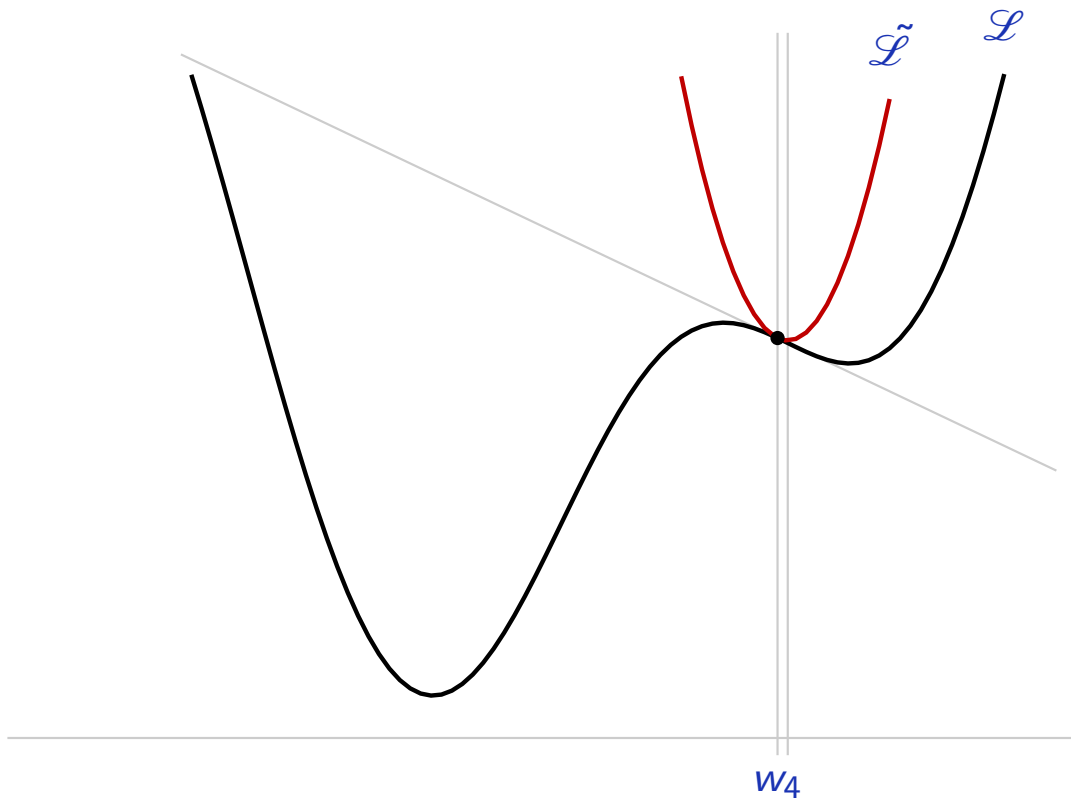
$\eta = 0.125$



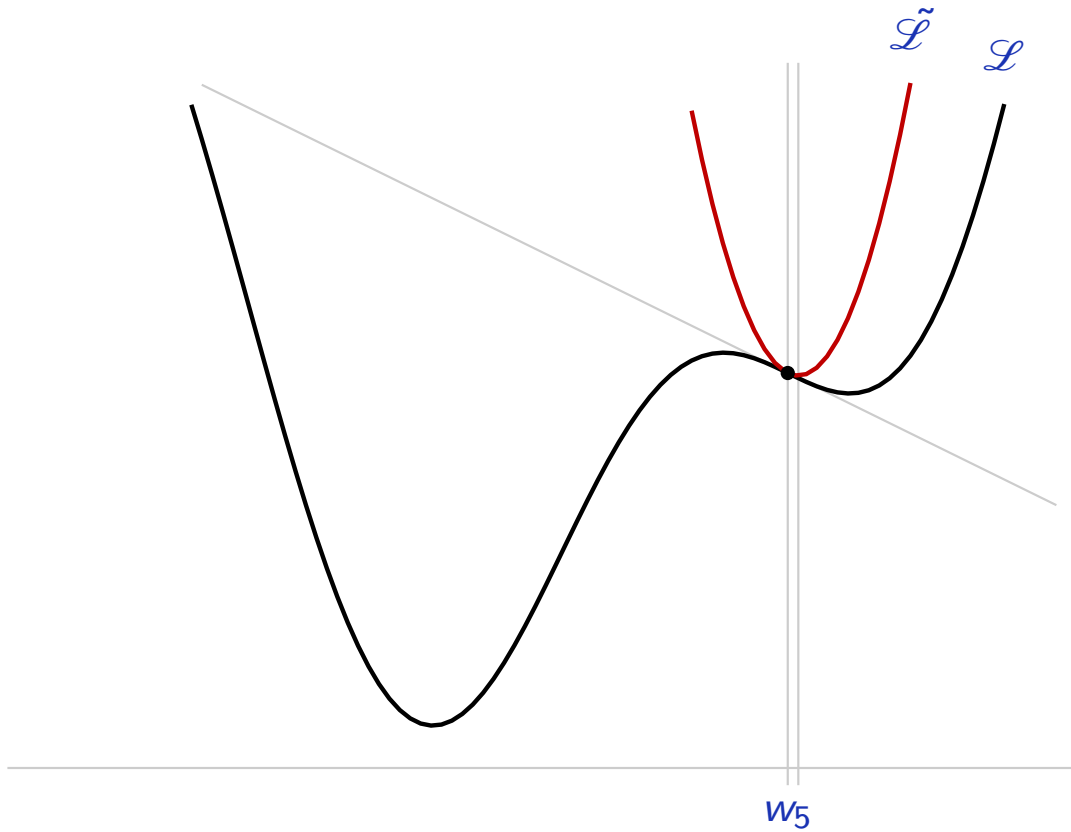
$\eta = 0.125$



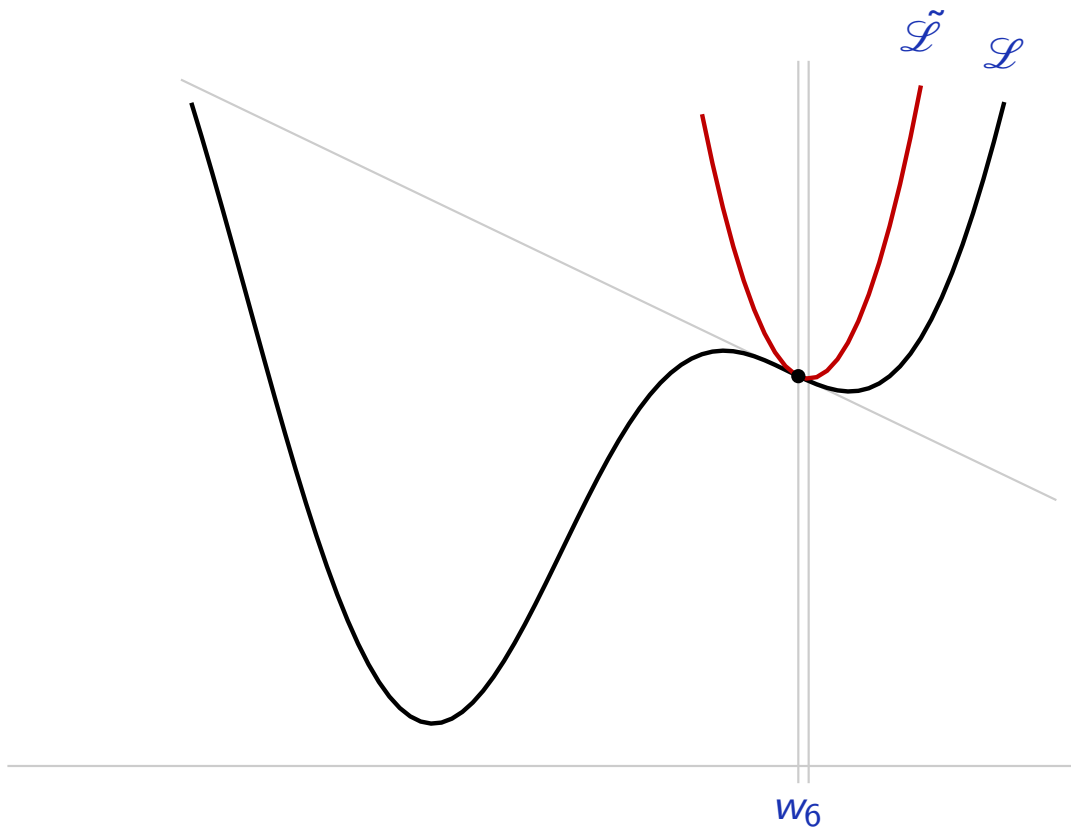
$\eta = 0.125$



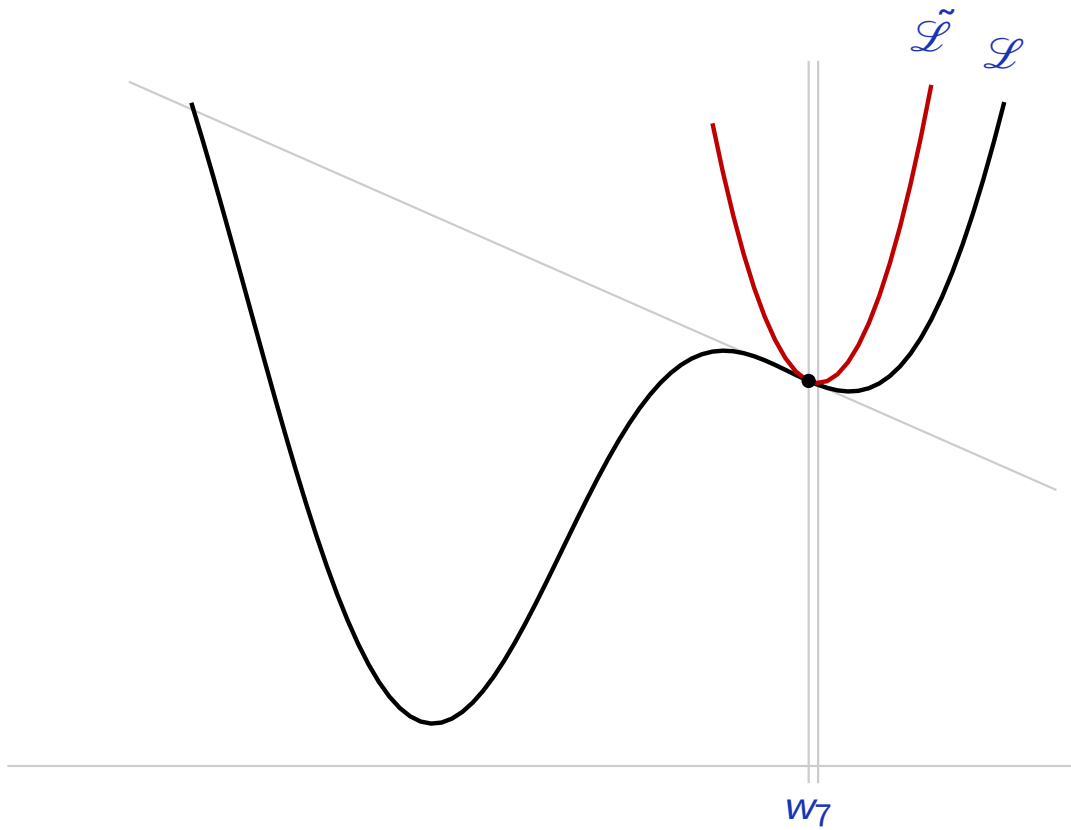
$\eta = 0.125$



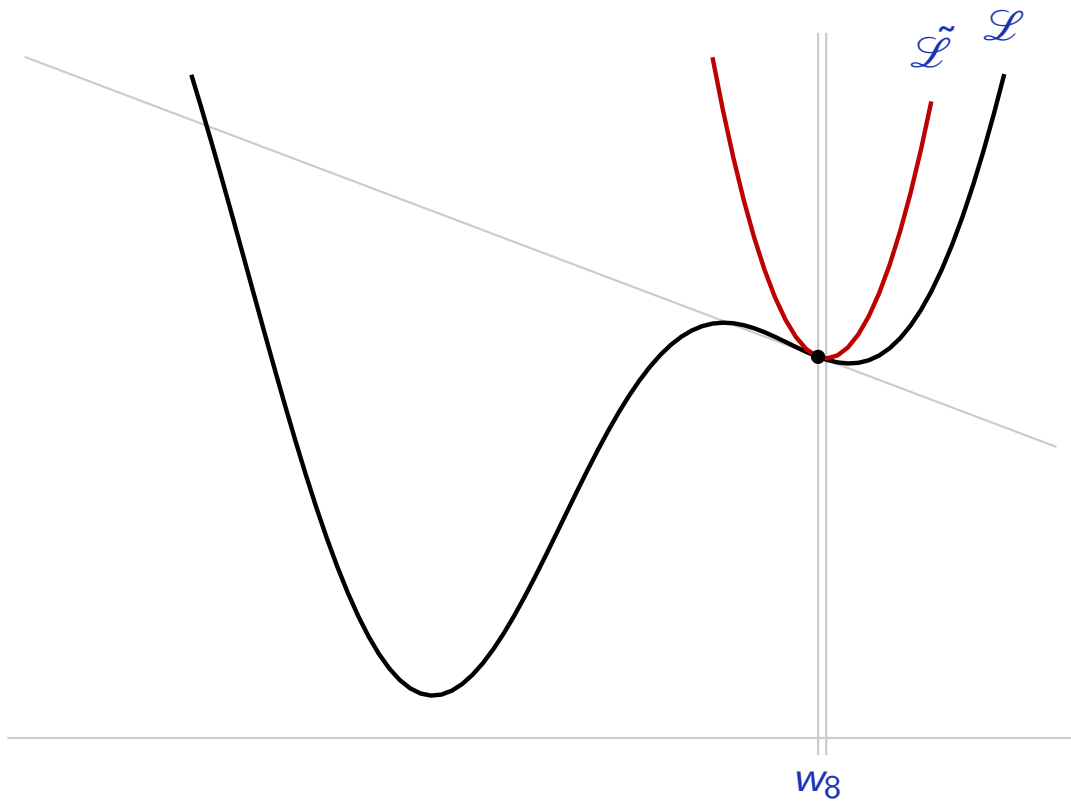
$\eta = 0.125$



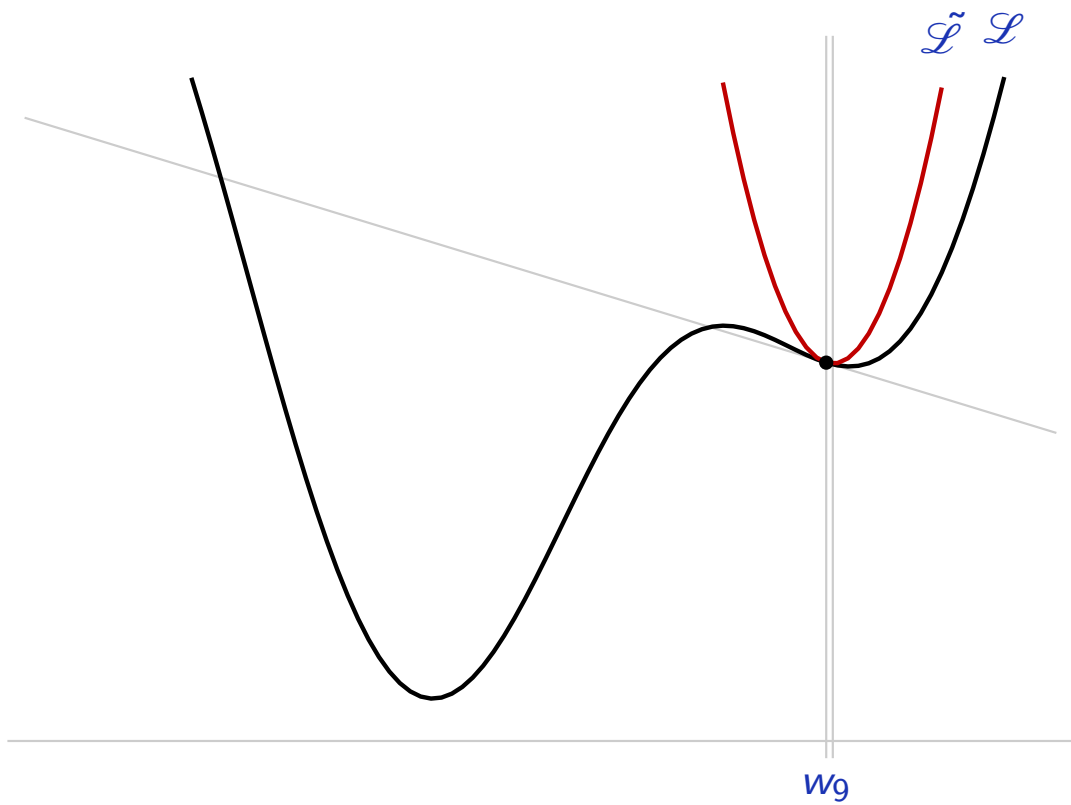
$\eta = 0.125$



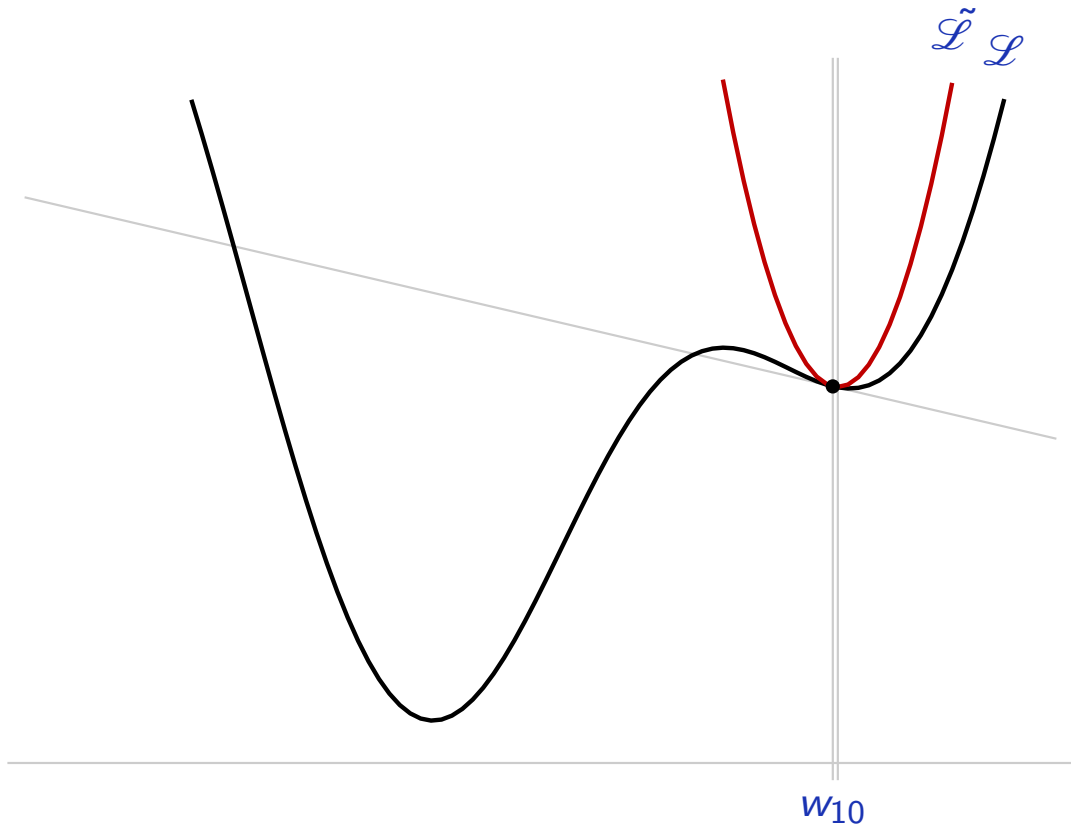
$\eta = 0.125$



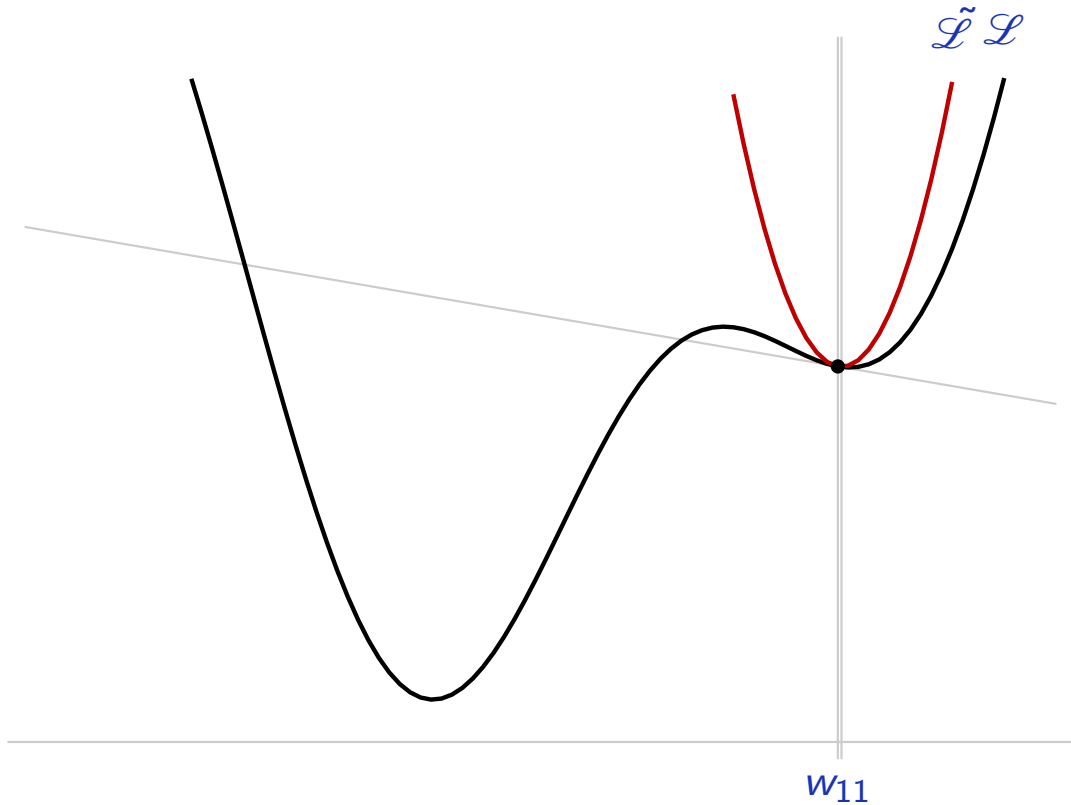
$\eta = 0.125$



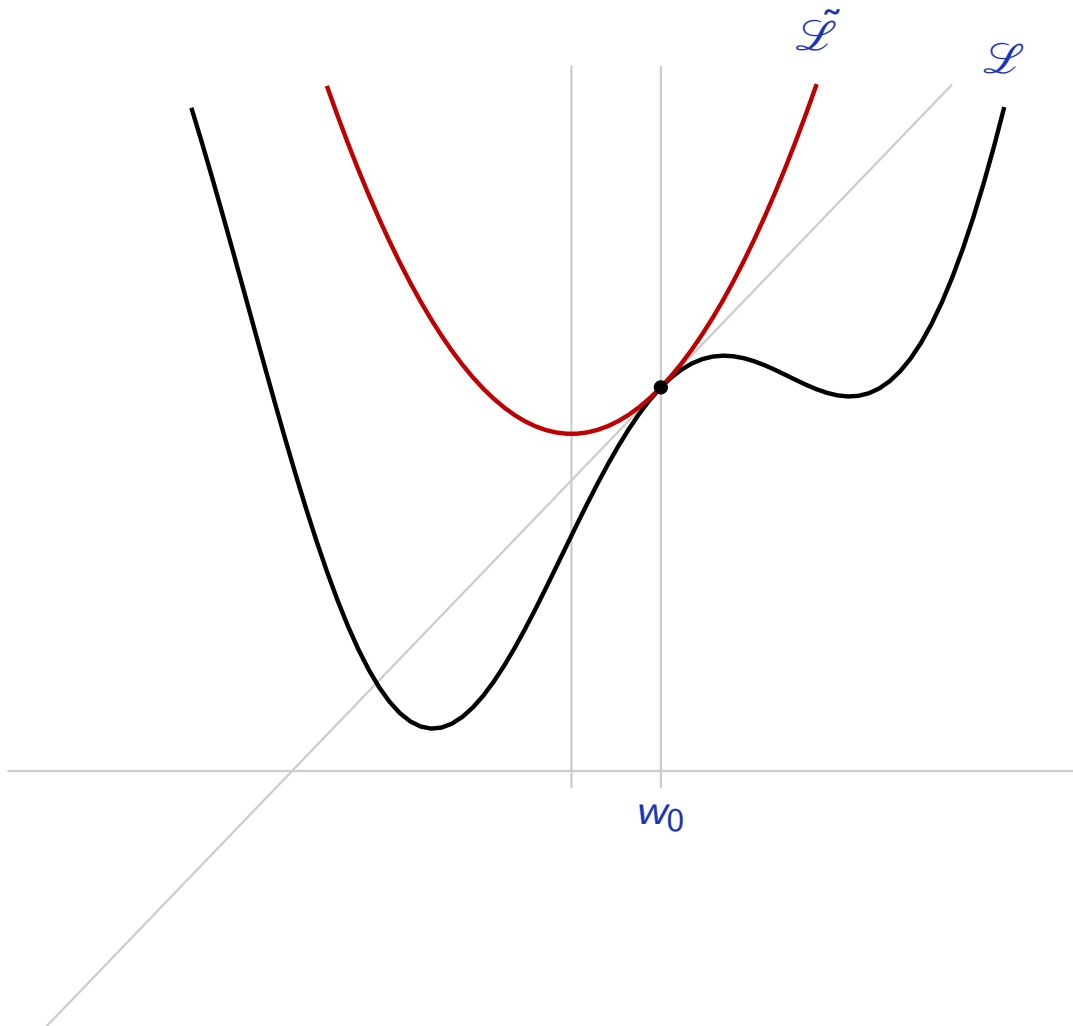
$\eta = 0.125$



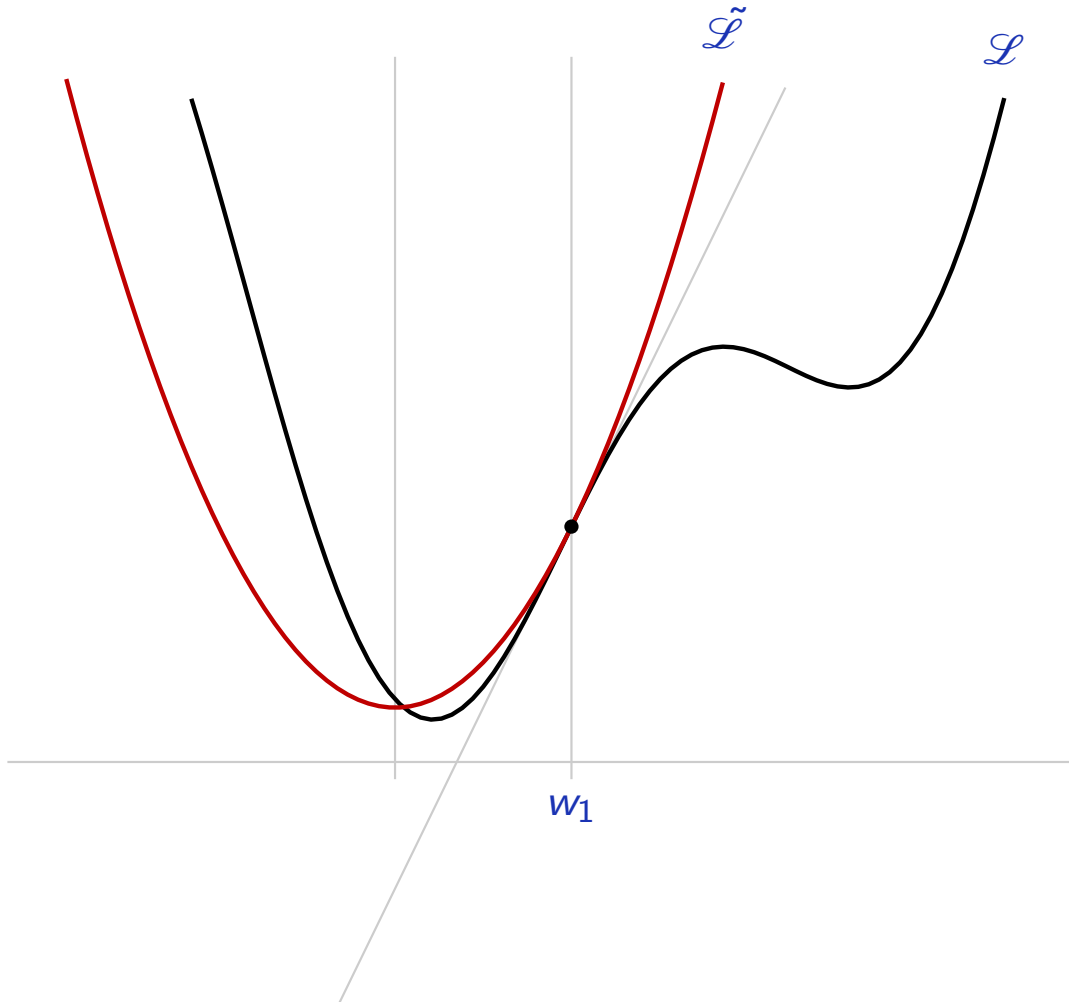
$\eta = 0.125$



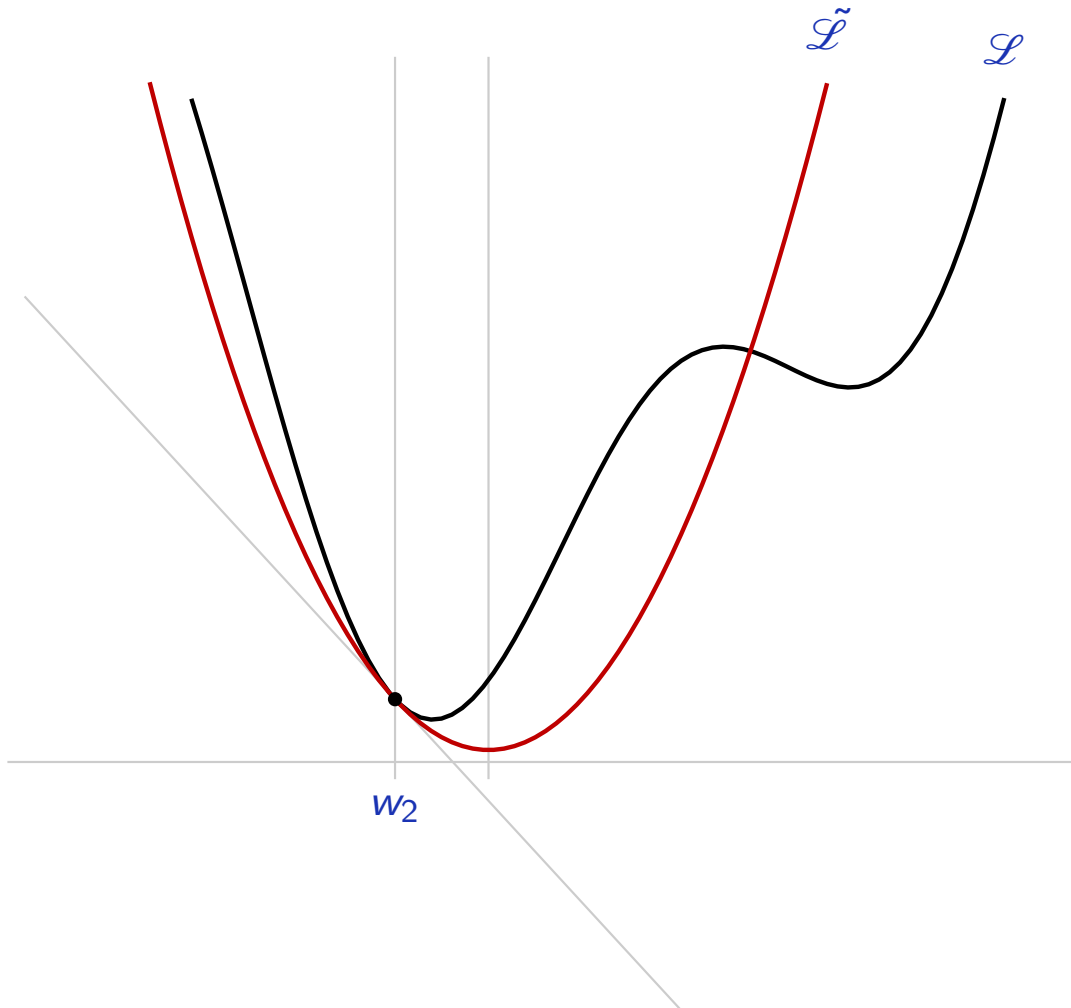
$\eta = 0.5$



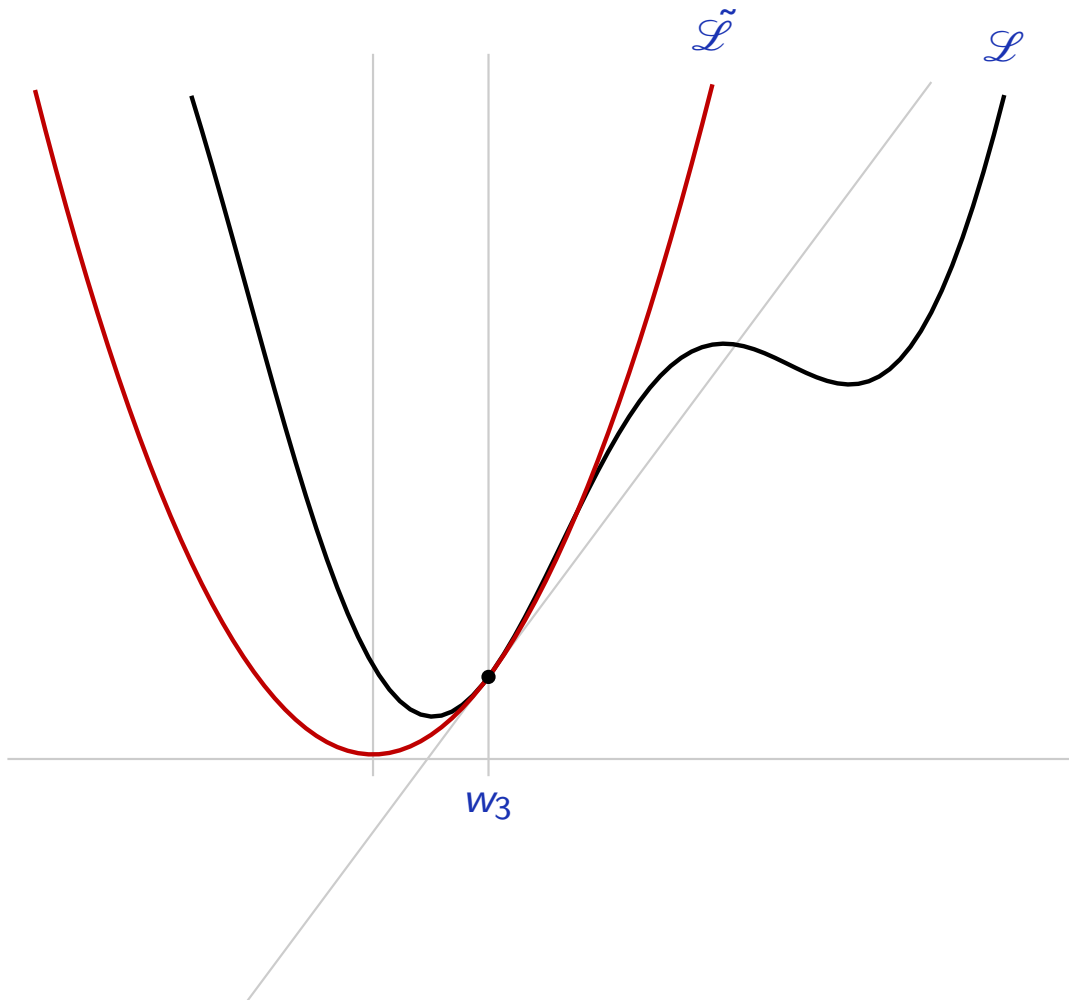
$\eta = 0.5$



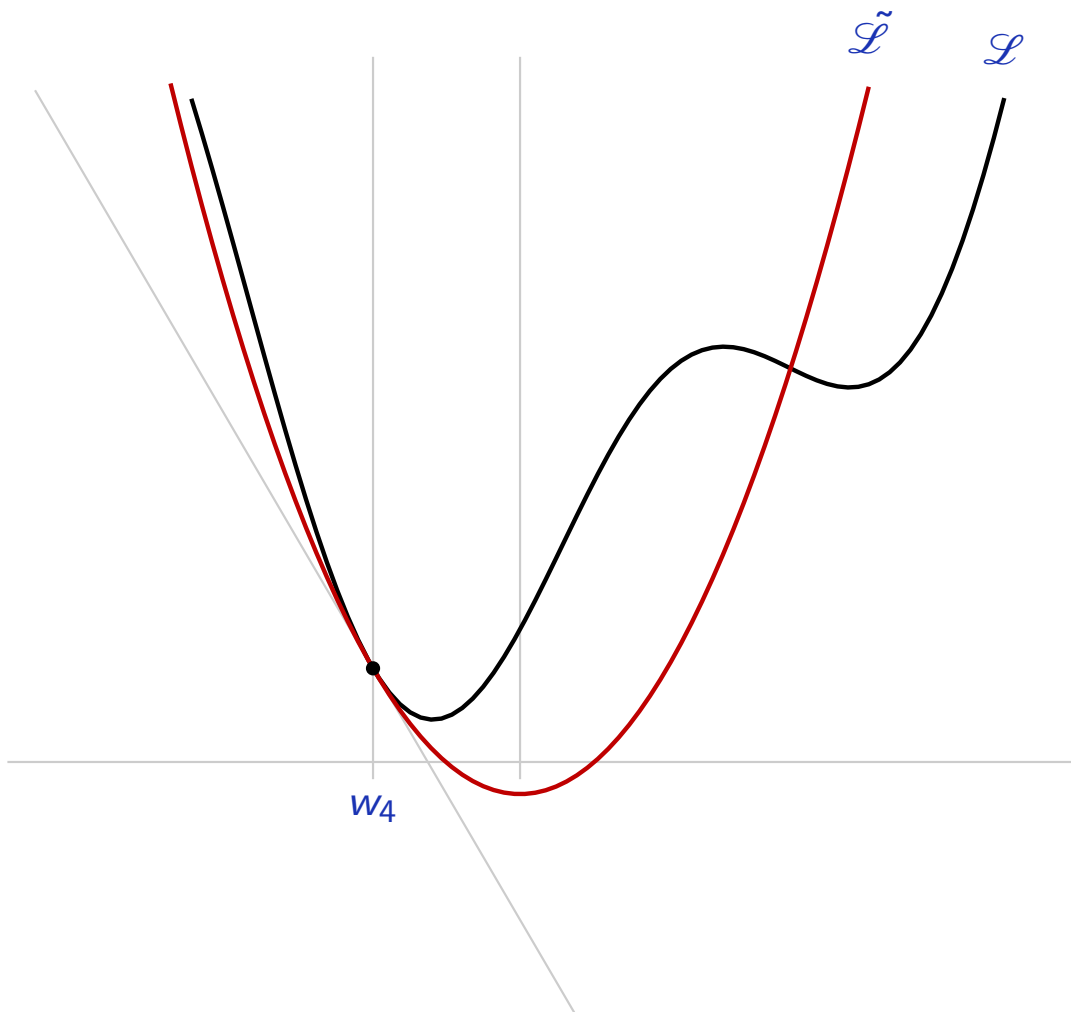
$\eta = 0.5$



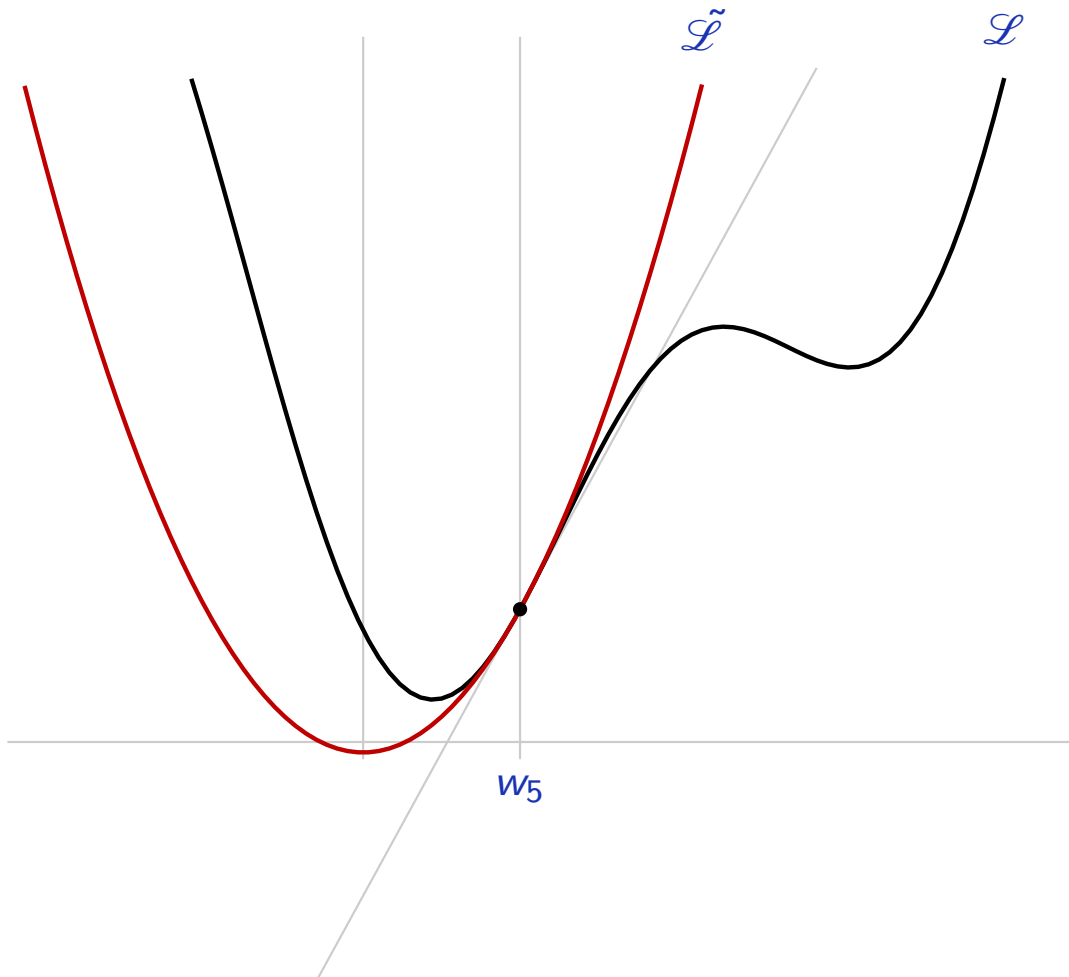
$\eta = 0.5$



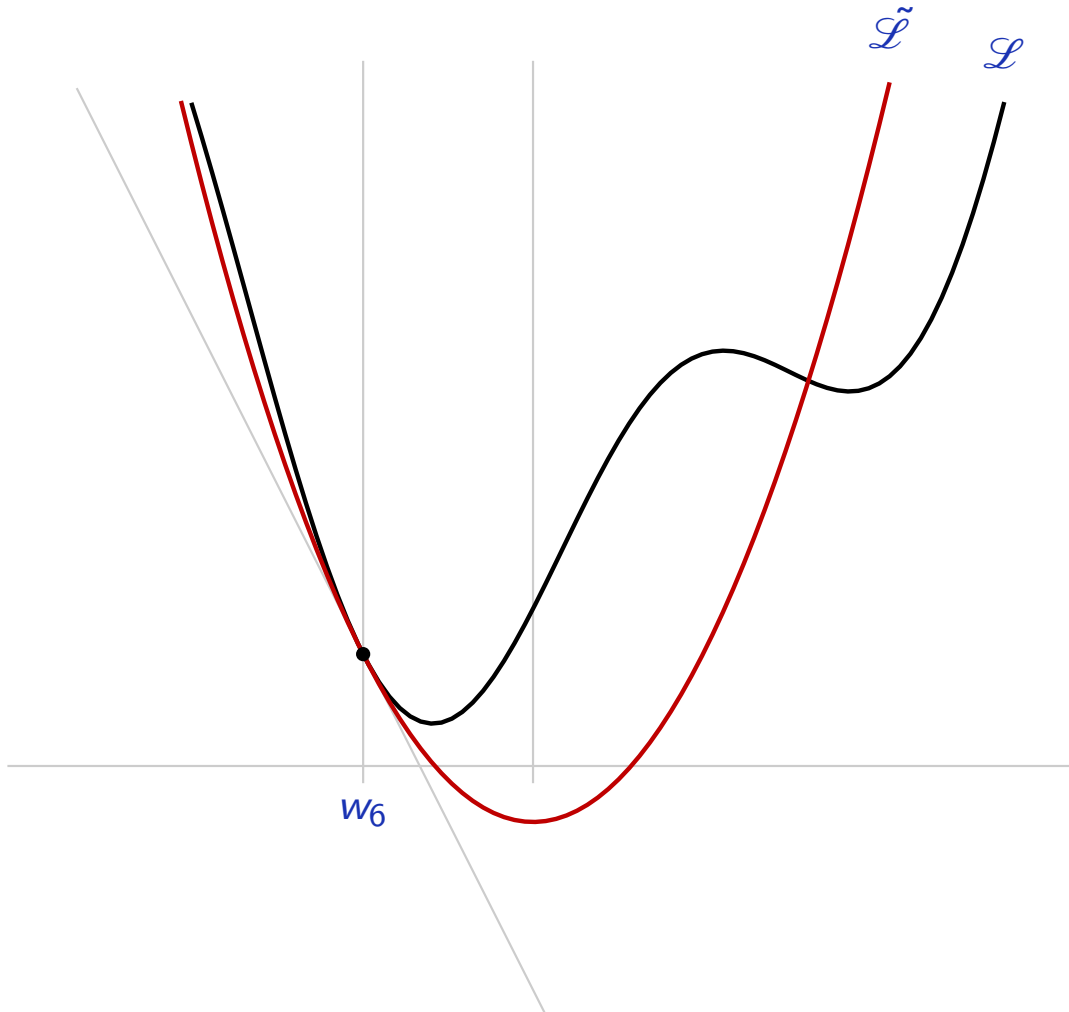
$\eta = 0.5$



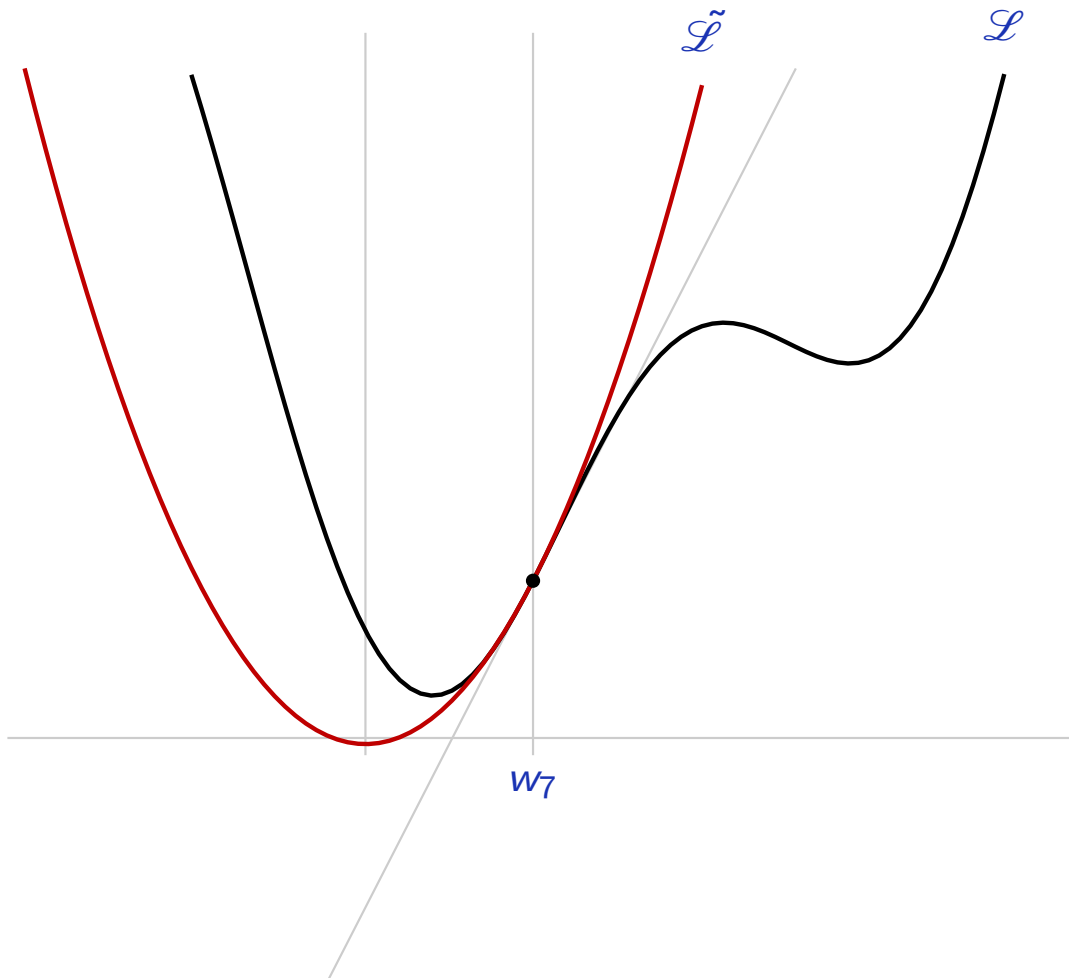
$\eta = 0.5$



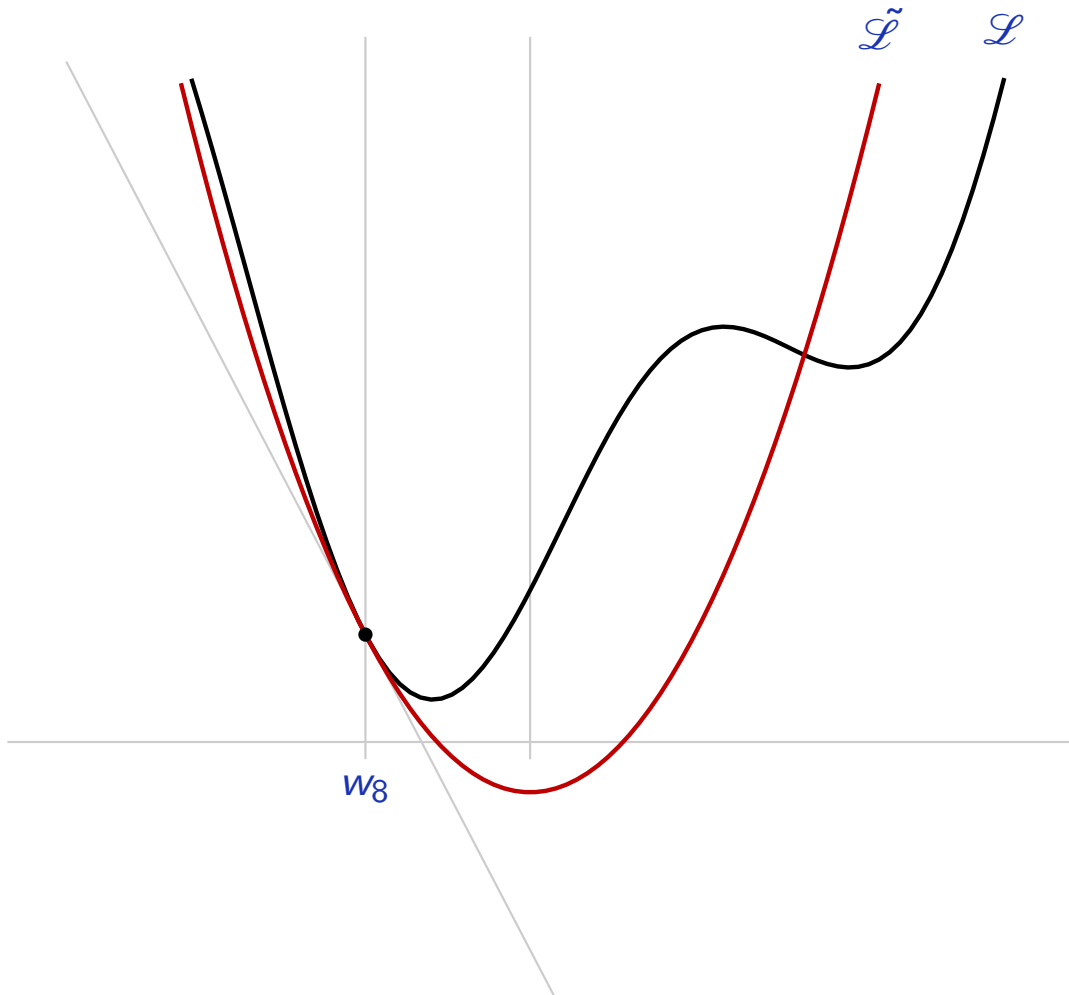
$\eta = 0.5$



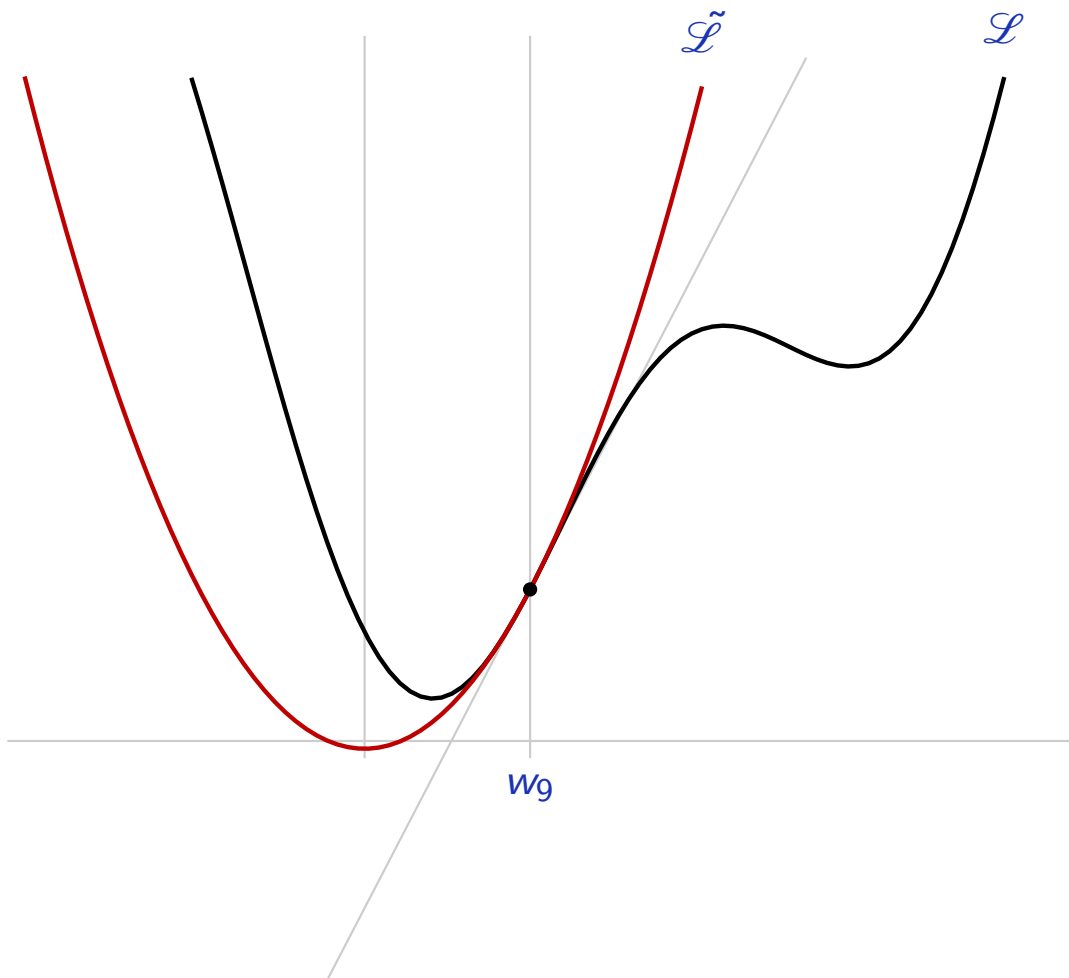
$\eta = 0.5$



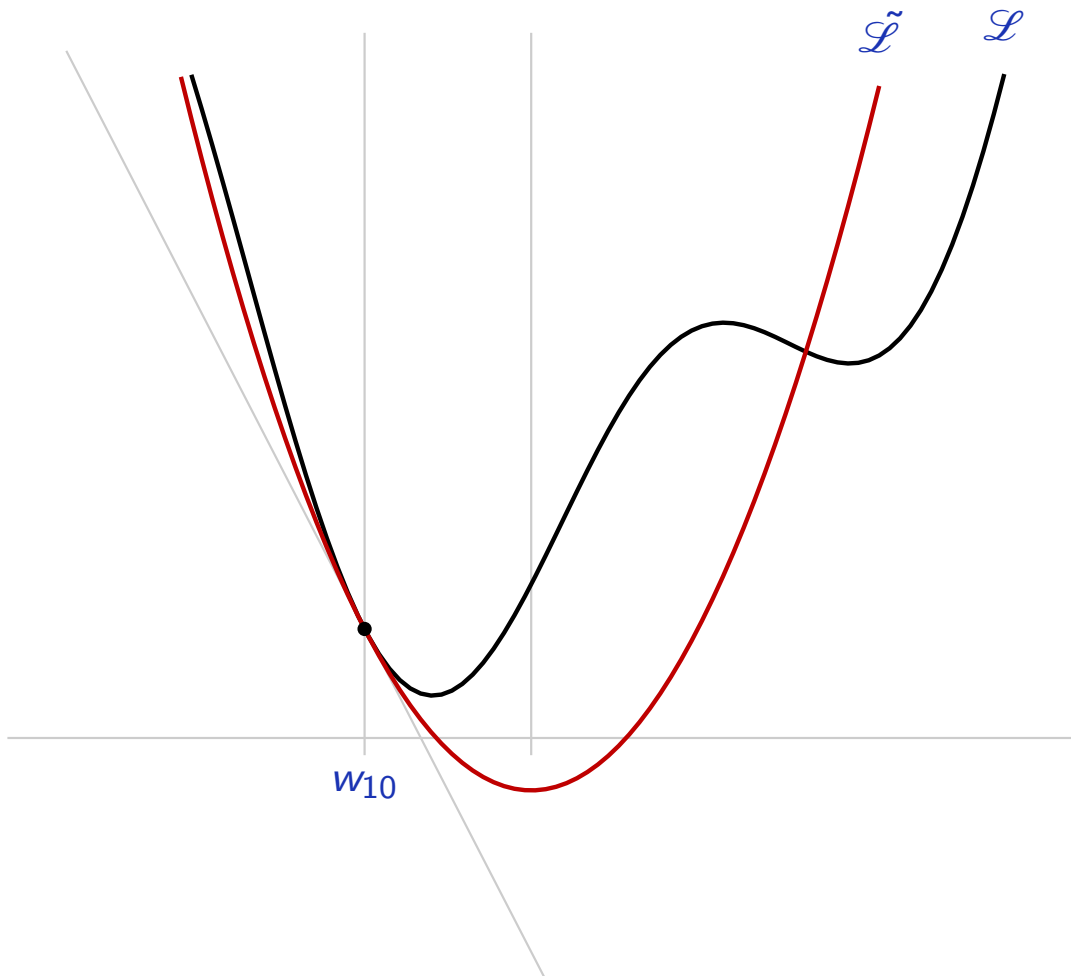
$\eta = 0.5$



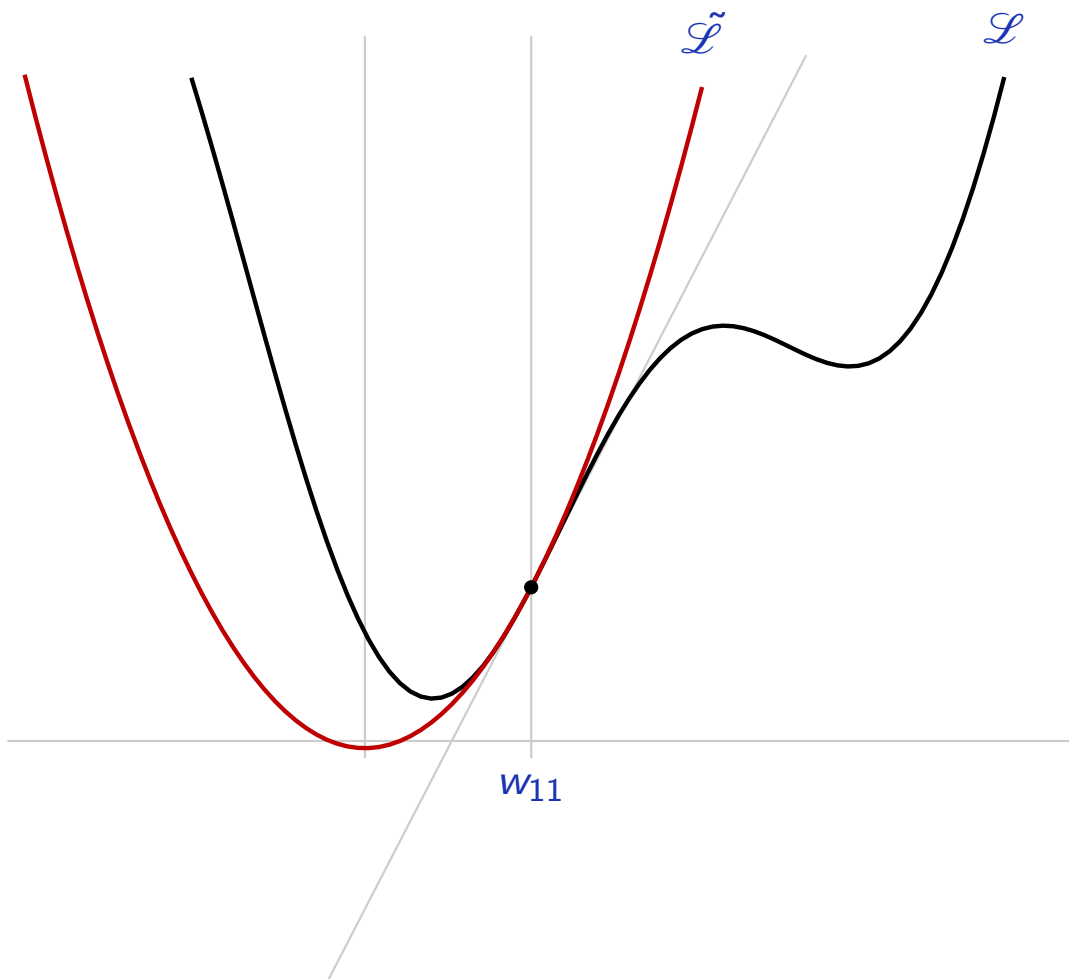
$\eta = 0.5$

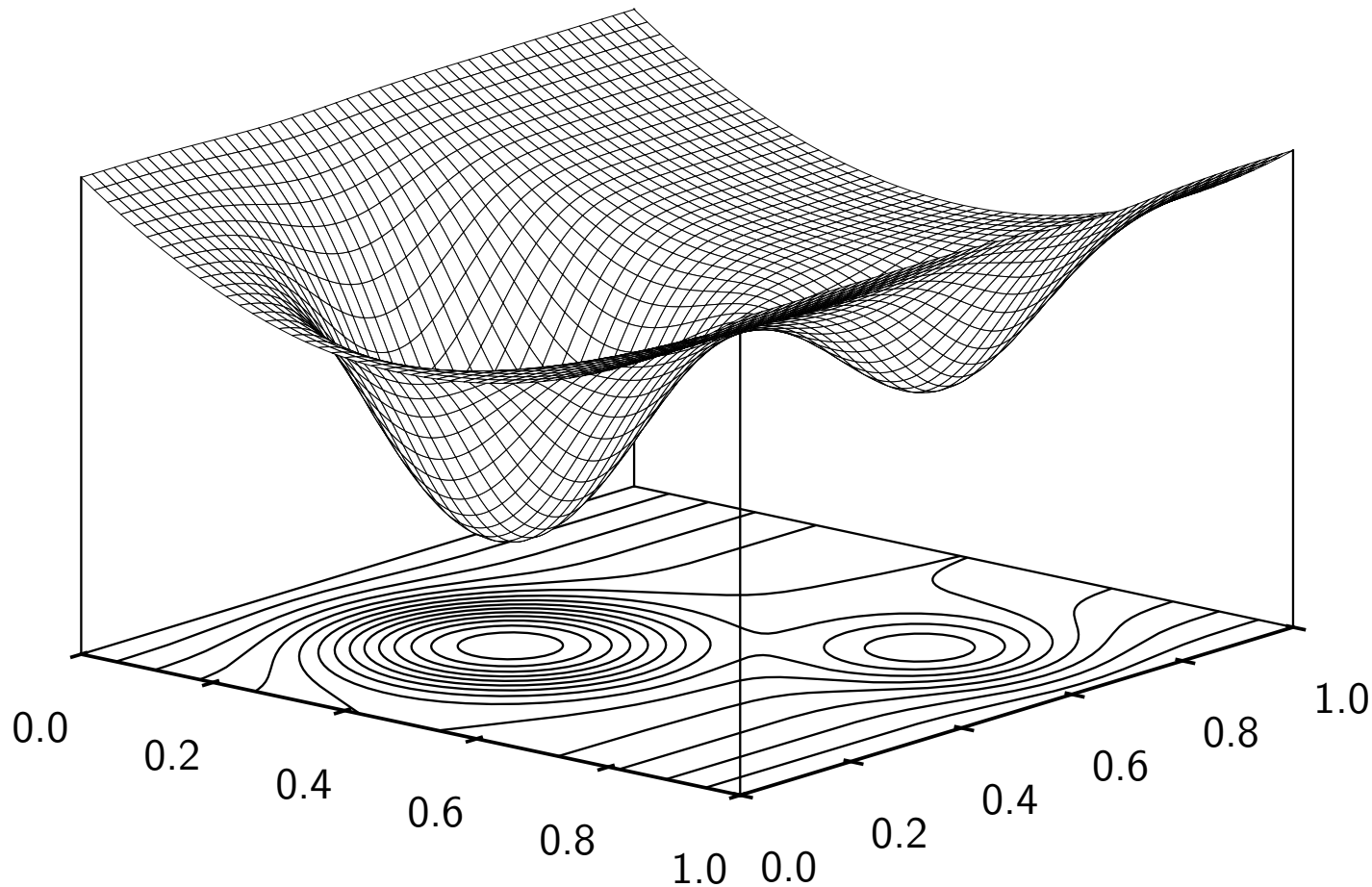


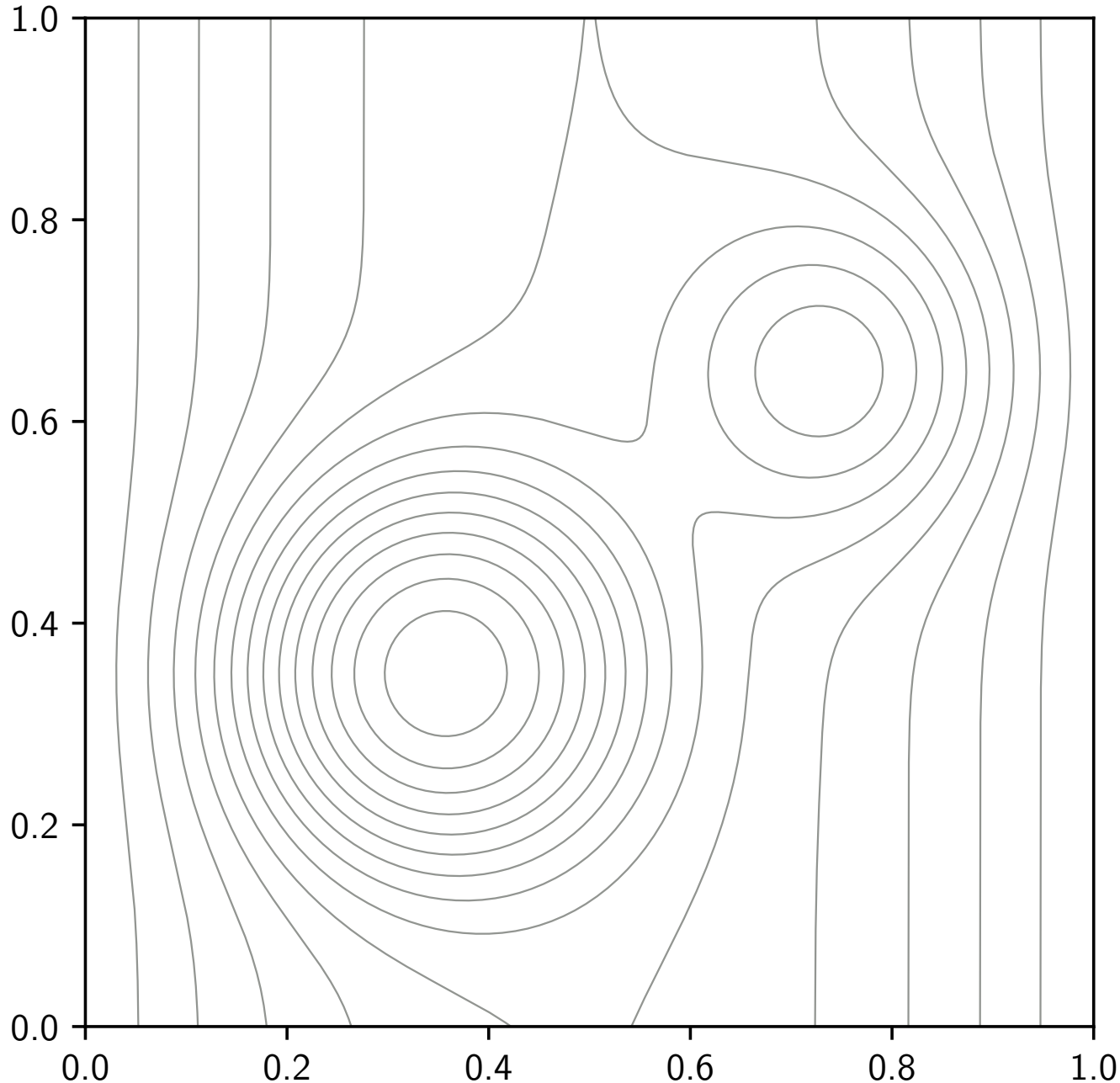
$\eta = 0.5$

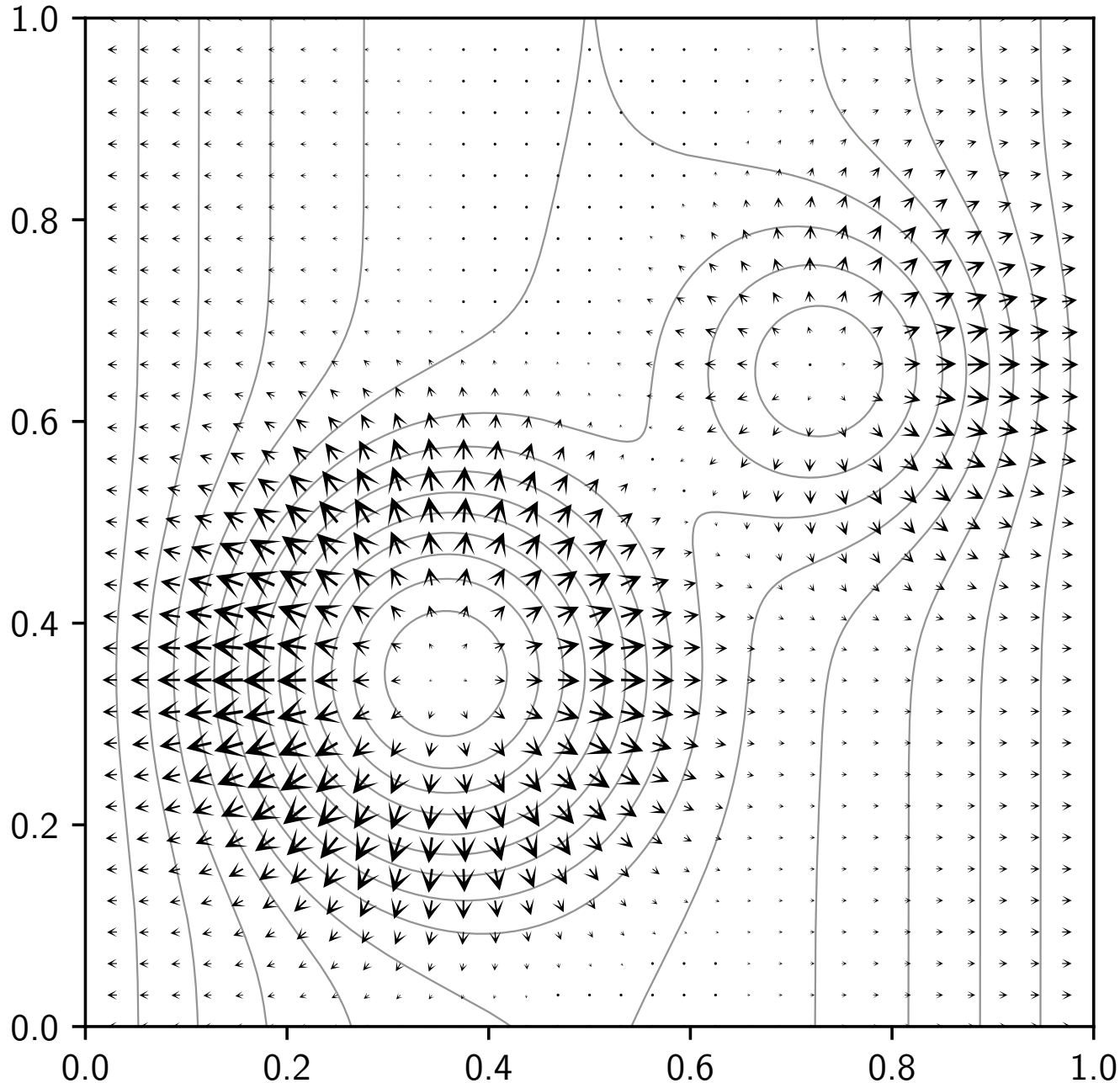


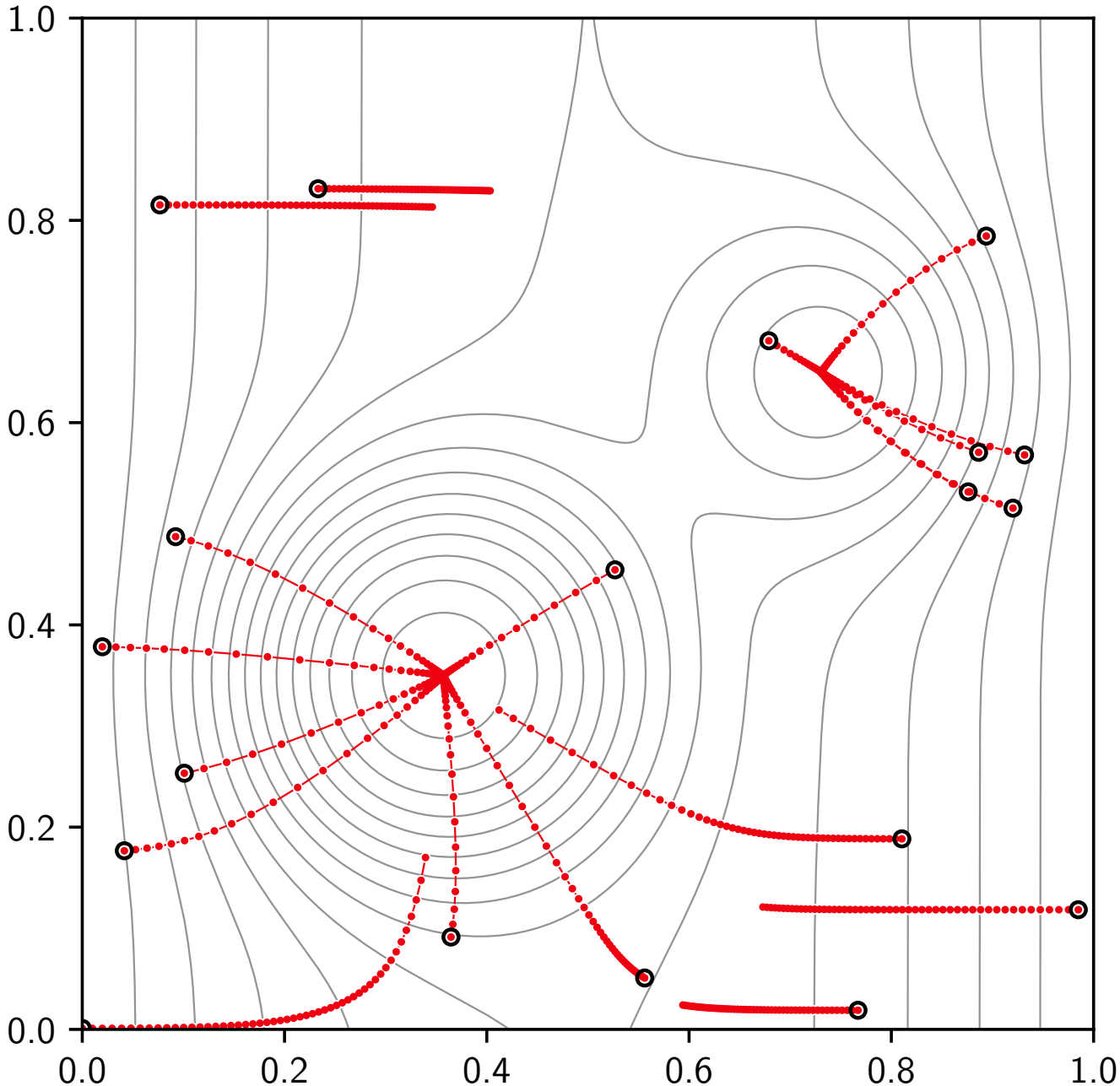
$\eta = 0.5$




0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

We saw that the minimum of the logistic regression loss

$$\mathscr{L}(w, b) = -\sum_n \log \sigma(y_n(w \cdot x_n + b))$$

does not have an analytic form.

We can derive

$$\frac{\partial \mathscr{L}}{\partial b} = -\sum_n \underbrace{y_n \, \sigma(-y_n(w \cdot x_n + b))}_{u_n},$$

$$\forall d, \ \frac{\partial \mathscr{L}}{\partial w_d} = -\sum_n \underbrace{x_{n,d} \, y_n \, \sigma(-y_n(w \cdot x_n + b))}_{v_{n,d}},$$

We can derive

$$\frac{\partial \mathscr{L}}{\partial b} = -\sum_n \underbrace{y_n \, \sigma(-y_n(w \cdot x_n + b))}_{u_n},$$

$$\forall d, \ \frac{\partial \mathscr{L}}{\partial w_d} = -\sum_n \underbrace{x_{n,d} \, y_n \, \sigma(-y_n(w \cdot x_n + b))}_{v_{n,d}},$$

which can be implemented as

```
def gradient(x, y, w, b):
    u = y * ( - y * (x @ w + b)).sigmoid()
    v = x * u.view(-1, 1) # Broadcasting
    return - v.sum(0), - u.sum(0)
```

We can derive

$$\frac{\partial \mathscr{L}}{\partial b} = -\sum_n \underbrace{y_n \, \sigma(-y_n(w \cdot x_n + b))}_{u_n},$$

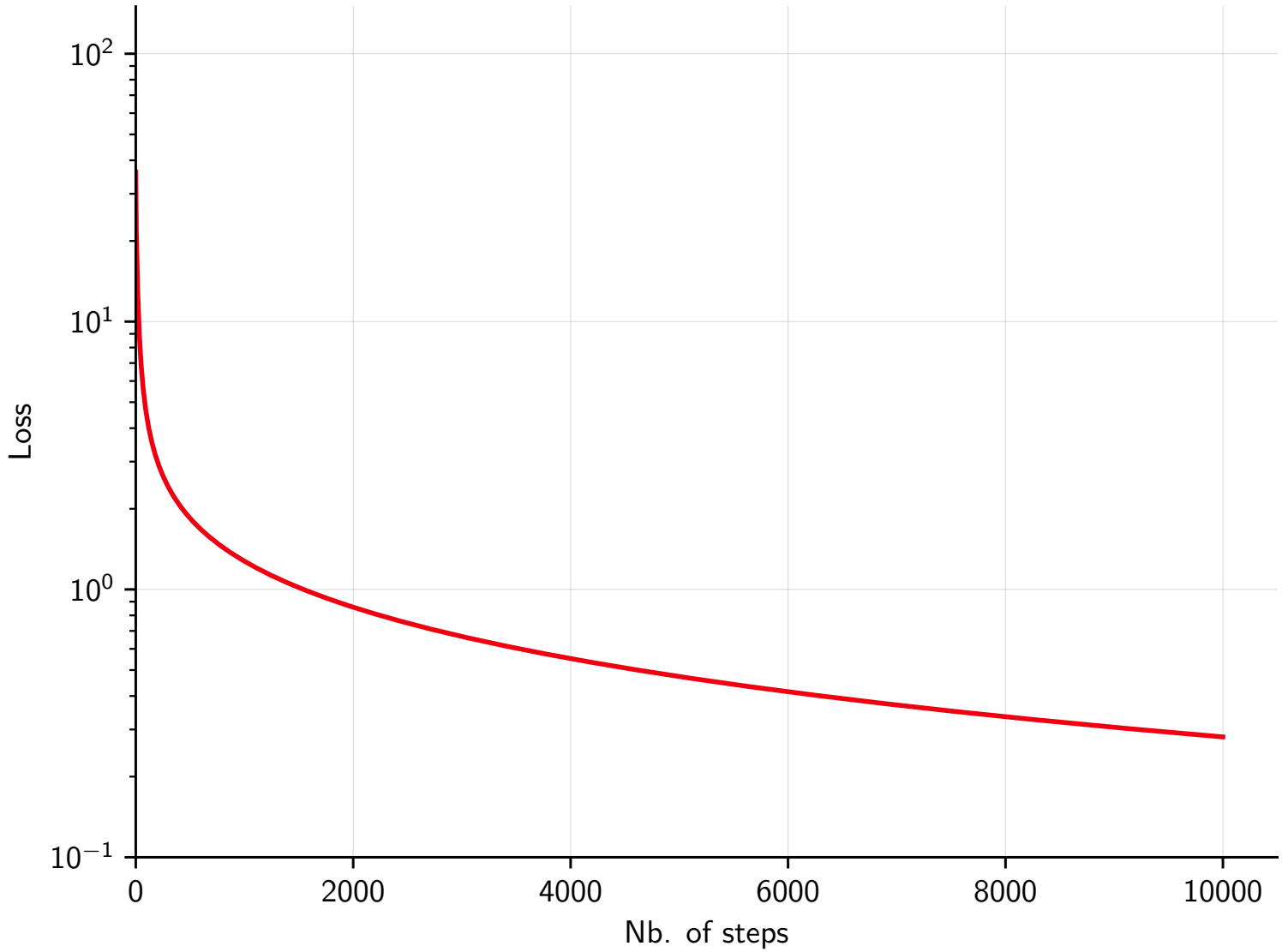
$$\forall d, \ \frac{\partial \mathscr{L}}{\partial w_d} = -\sum_n \underbrace{x_{n,d} \, y_n \, \sigma(-y_n(w \cdot x_n + b))}_{v_{n,d}},$$

which can be implemented as

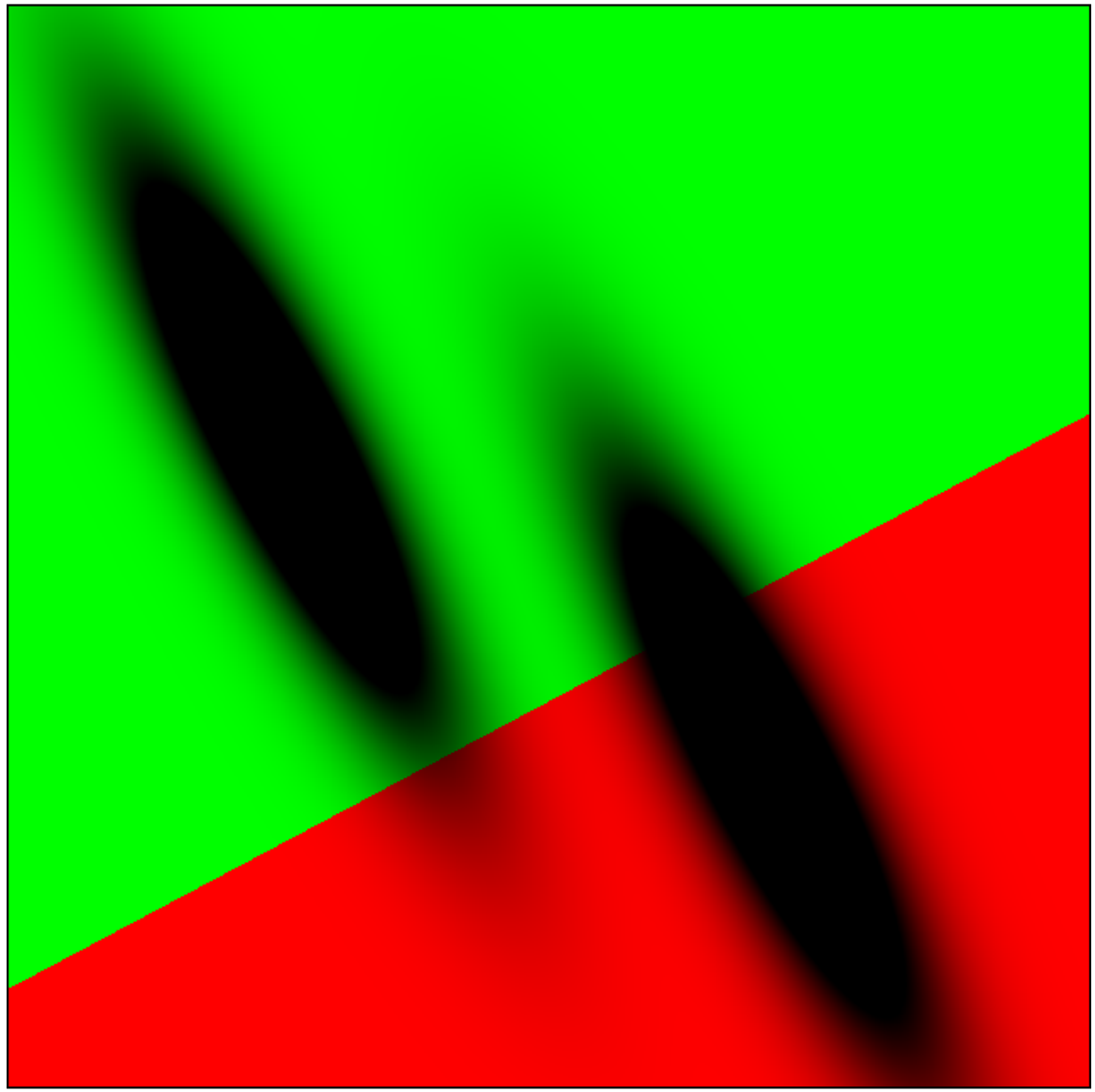
```
def gradient(x, y, w, b):
    u = y * ( - y * (x @ w + b)).sigmoid()
    v = x * u.view(-1, 1) # Broadcasting
    return - v.sum(0), - u.sum(0)
```

and the gradient descent as

```
w, b = torch.randn(x.size(1)), 0
eta = 1e-1

for k in range(nb_iterations):
    print(k, loss(x, y, w, b))
    dw, db = gradient(x, y, w, b)
    w -= eta * dw
    b -= eta * db
```

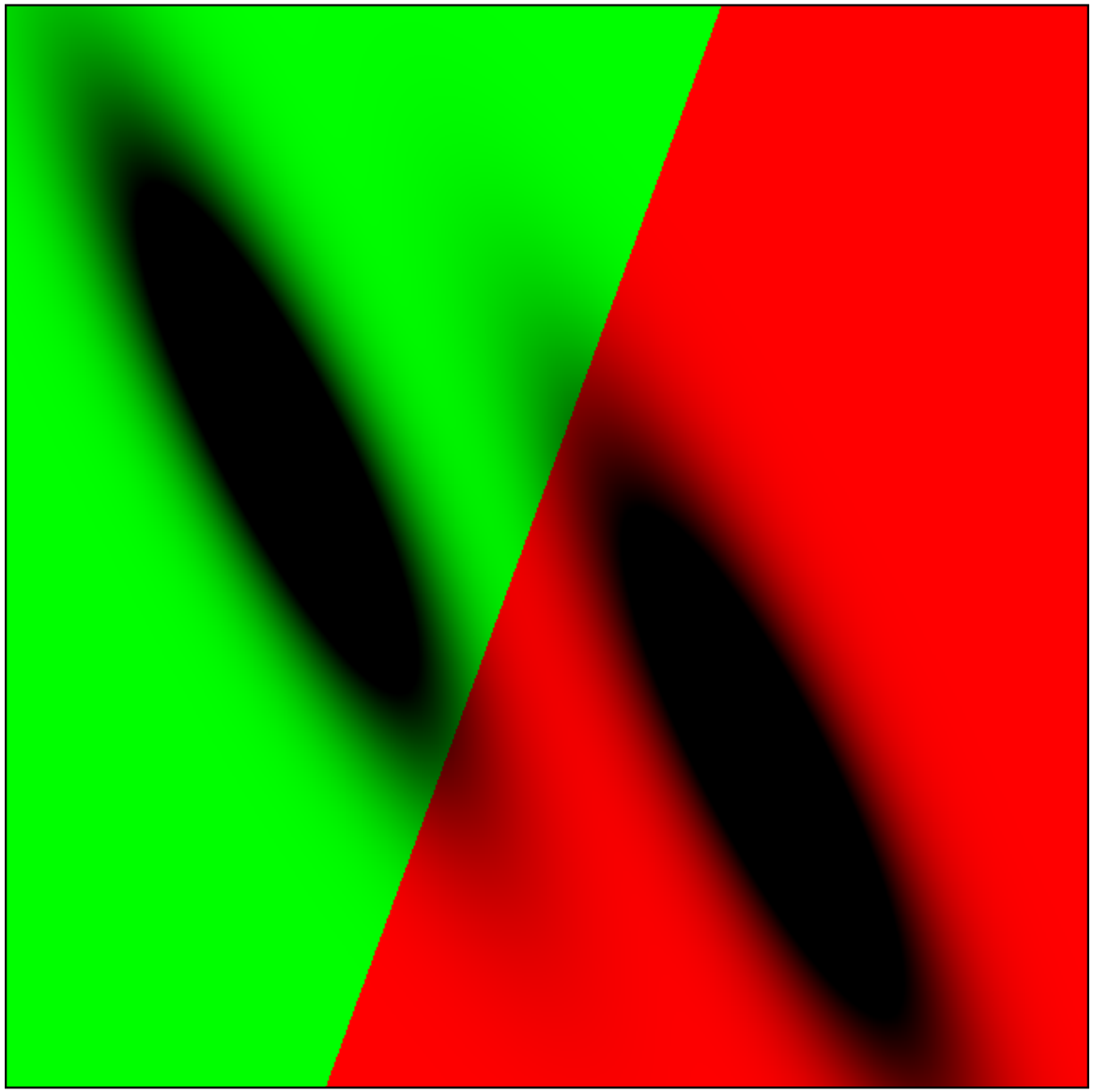

10^2
10^1
10^0
10^{-1}
Loss
0
2000
4000
6000
8000
10000
Nb. of steps

With 100 training points and $\eta = 10^{-1}$.



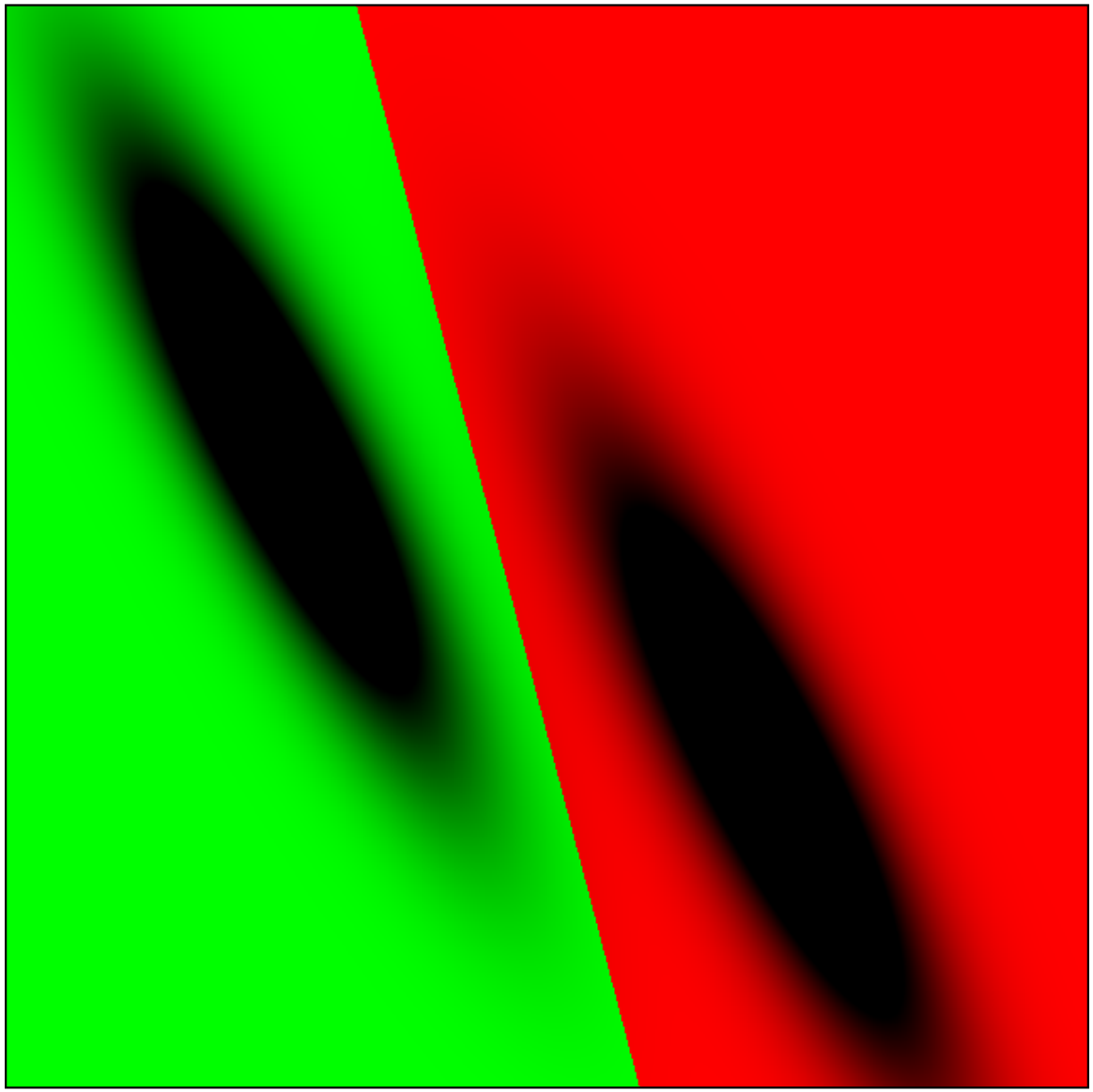
$n = 0$

With 100 training points and $\eta = 10^{-1}$.



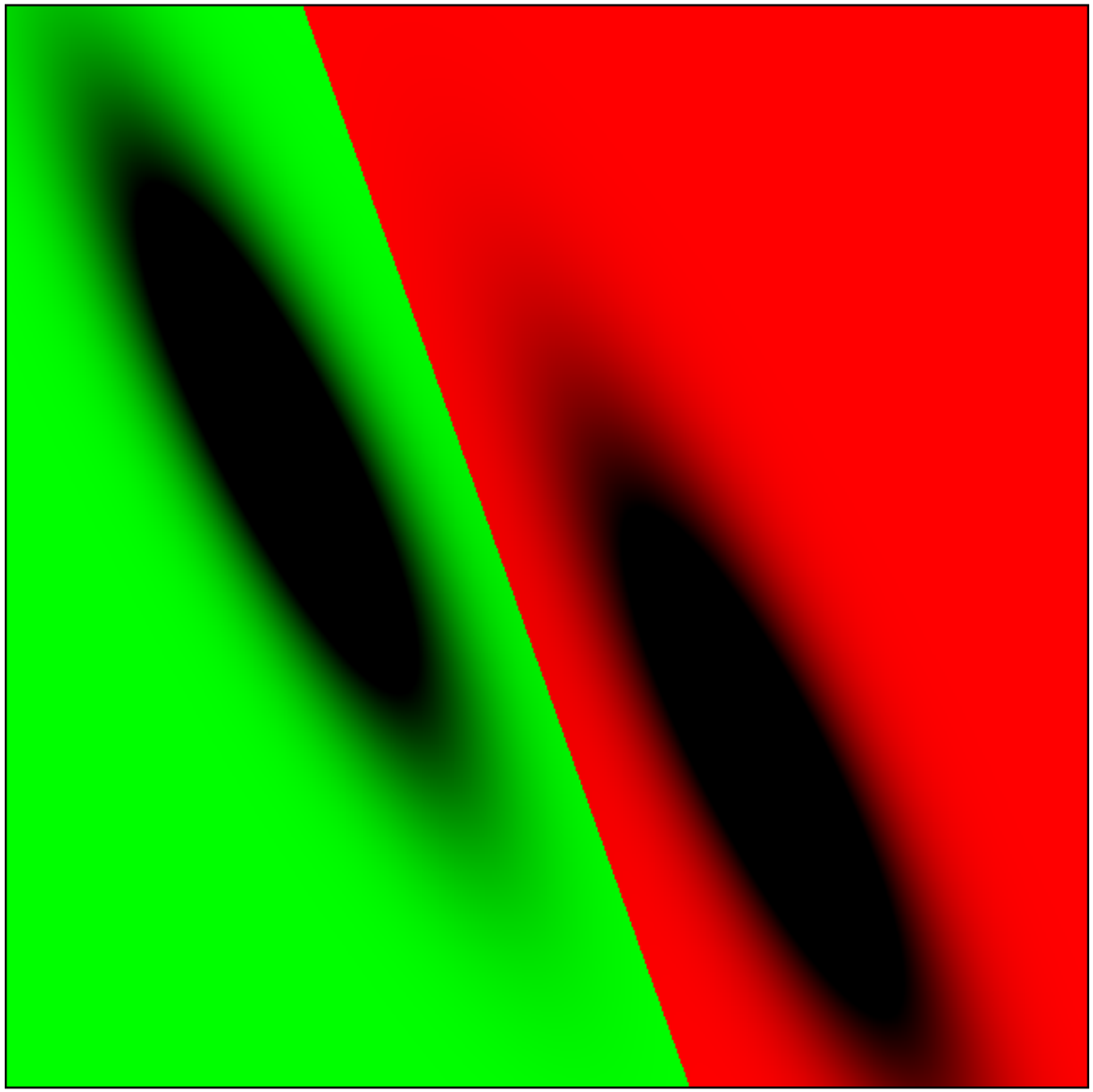
$n = 10$

With 100 training points and $\eta = 10^{-1}$.



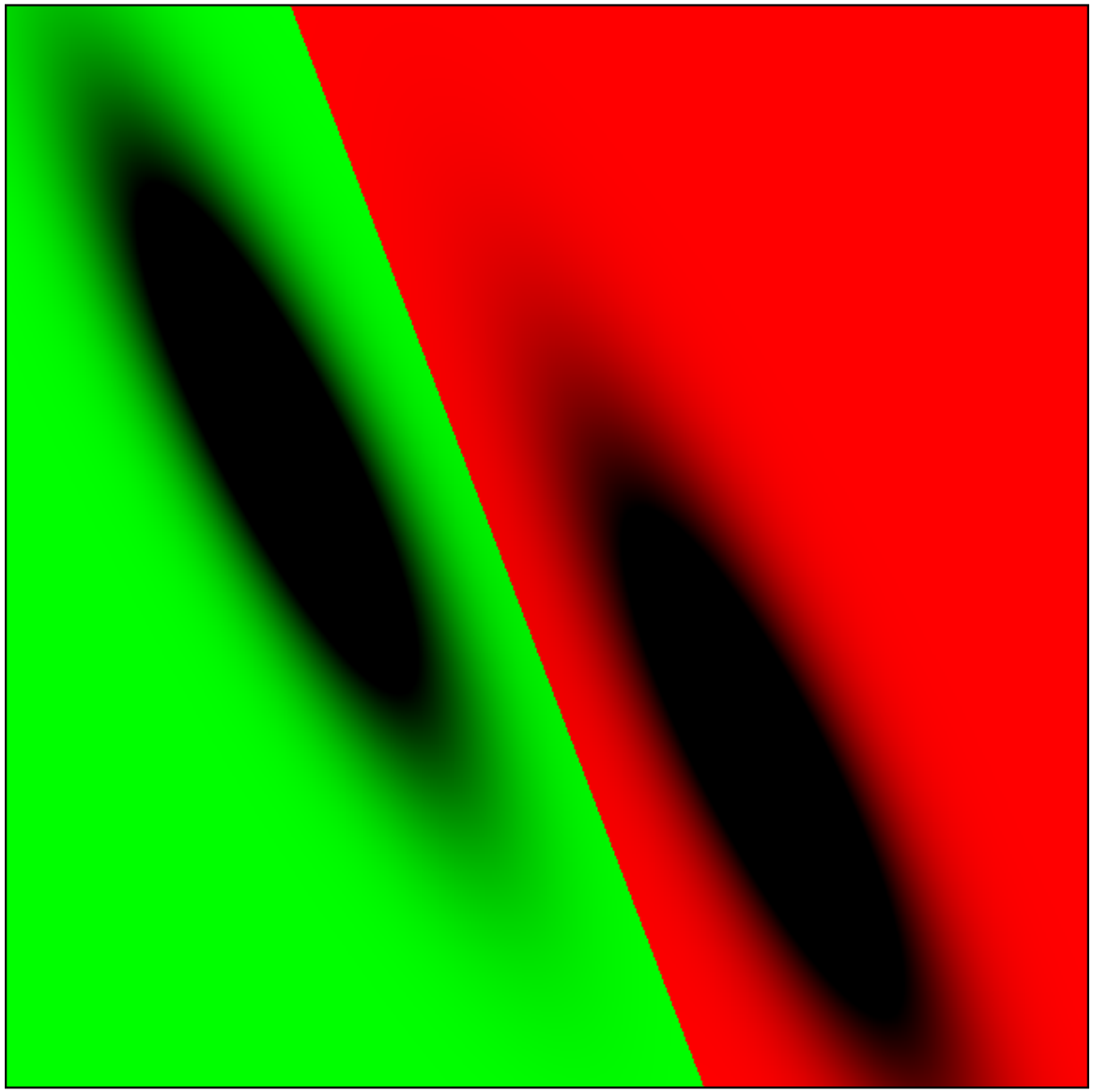
$n = 10^2$

With 100 training points and $\eta = 10^{-1}$.

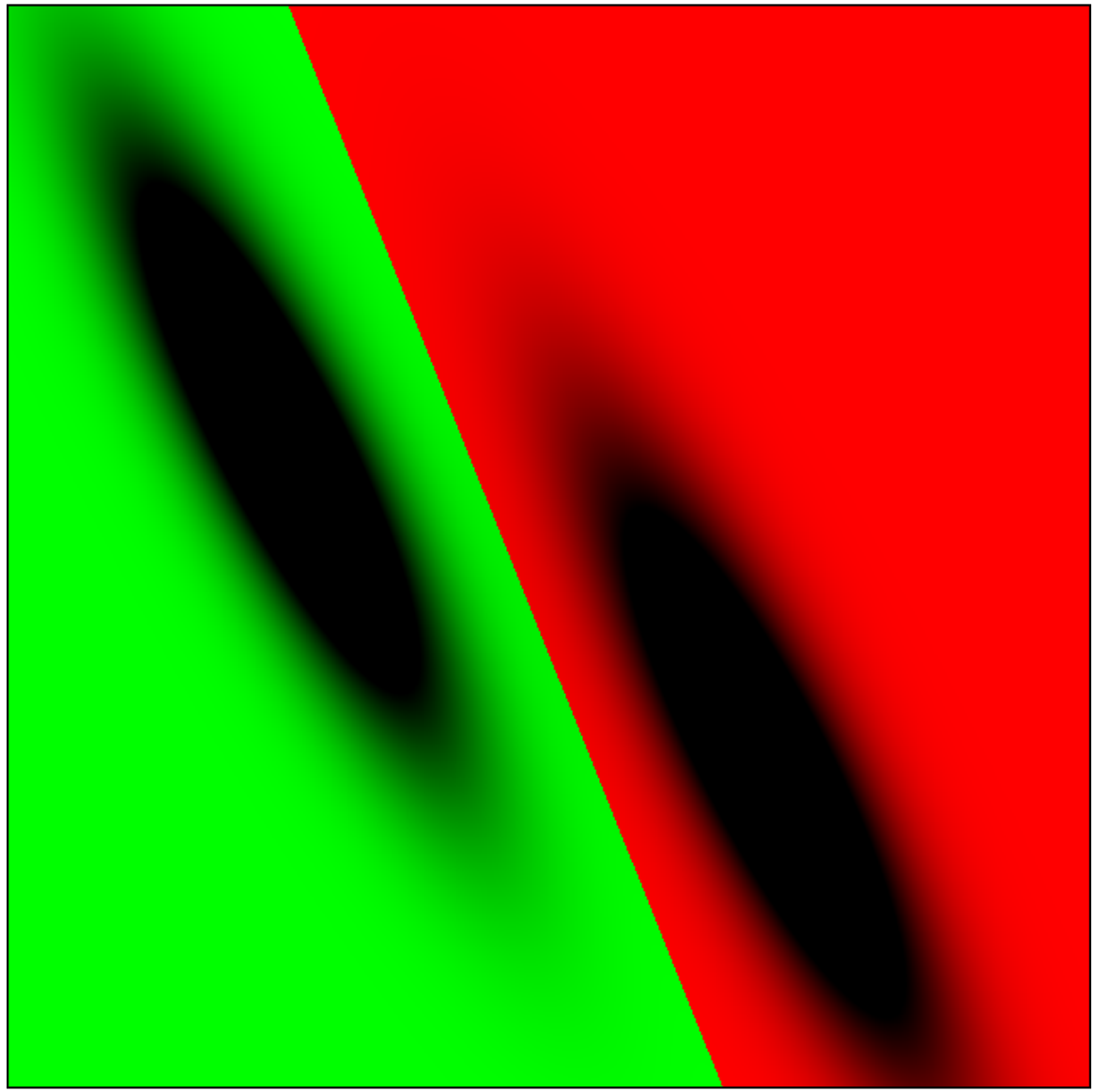


$n = 10^3$

With 100 training points and $\eta = 10^{-1}$.



$n = 10^4$



LDA

Deep learning

3.6. Back-propagation

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

We want to train an MLP by minimizing a loss over the training set

$$\mathscr{L}(w, b) = \sum_n \ell(f(x_n; w, b), y_n).$$

To use gradient descent, we need the expression of the gradient of the per-sample loss

$$\ell_n = \ell(f(x_n; w, b), y_n)$$

with respect to the parameters, e.g.

$$\frac{\partial \ell_n}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial \ell_n}{\partial b_i^{(l)}}.$$

For clarity, we consider a single training sample x, and introduce $s^{(1)}, \ldots, s^{(L)}$ as the summations before activation functions.

$$x^{(0)} = x \xrightarrow{w^{(1)},b^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \xrightarrow{w^{(2)},b^{(2)}} s^{(2)} \xrightarrow{\sigma} \ldots \xrightarrow{w^{(L)},b^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)} = f(x; w, b).$$

Formally we set $x^{(0)} = x$,

$$\forall l = 1, \ldots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right), \end{cases}$$

and we set the output of the network as $f(x; w, b) = x^{(L)}$.

This is the **forward pass**.

The core principle of the back-propagation algorithm is the "chain rule" from differential calculus:

$$(g \circ f)' = (g' \circ f) f'.$$

The linear approximation of a composition of mappings is the product of their individual linear approximations.

This generalizes to longer compositions and higher dimensions

$$J_{f_N \circ f_{N-1} \circ \cdots \circ f_1}(x) = J_{F_N}(f_{N-1}(\ldots(x))) \ldots J_{f_3}(f_2(f_1(x))) J_{f_2}(f_1(x)) J_{f_1}(x)$$

where $J_f(x)$ is the Jacobian of f at x, that is the matrix of the linear approximation of f in the neighborhood of x.

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)}, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$ with

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$ with

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

we have

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}}$$

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$ with

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

we have

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right),$$

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$ with

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

we have

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right),$$

And since $x_j^{(l-1)}$ influences ℓ only through the $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$ with

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

we have

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right),$$

And since $x_j^{(l-1)}$ influences ℓ only through the $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

we have

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial x_j^{(l-1)}}$$

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$ with

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

we have

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right),$$

And since $x_j^{(l-1)}$ influences ℓ only through the $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

we have

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}.$$

Derivatives w.r.t the activations

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $s_i^{(l)}$ influences ℓ only through $x_i^{(l)}$ with

$$x_i^{(l)} = \sigma(s_i^{(l)}),$$

we have

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right),$$

And since $x_j^{(l-1)}$ influences ℓ only through the $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

we have

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}.$$

Derivatives w.r.t the weights an biases

$$x^{(l-1)} \xrightarrow{w^{(l)}, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $w_{i,j}^{(l)}$ and $b_i^{(l)}$ influences ℓ only through $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

Derivatives w.r.t the weights an biases

$$x^{(l-1)} \xrightarrow{w^{(l)}, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $w_{i,j}^{(l)}$ and $b_i^{(l)}$ influences ℓ only through $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

we have

$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial w_{i,j}^{(l)}}$$

Derivatives w.r.t the weights an biases

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $w_{i,j}^{(l)}$ and $b_i^{(l)}$ influences ℓ only through $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

we have

$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)},$$

Derivatives w.r.t the weights an biases

$$x^{(l-1)} \xrightarrow{w^{(l)},\, b^{(l)}} s^{(l)} \xrightarrow{\sigma} x^{(l)}$$

Since $w_{i,j}^{(l)}$ and $b_i^{(l)}$ influences ℓ only through $s_i^{(l)}$ with

$$s_i^{(l)} = \sum_j w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)},$$

we have

$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)},$$

$$\frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}}.$$

To summarize: we can compute $\frac{\partial \ell}{\partial x_i^{(L)}}$ from the definition of ℓ, and recursively **propagate backward** the derivatives of the loss w.r.t the activations with

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right)$$

and

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}.$$

To summarize: we can compute $\frac{\partial \ell}{\partial x_i^{(L)}}$ from the definition of ℓ, and recursively **propagate backward** the derivatives of the loss w.r.t the activations with

$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma'\left(s_i^{(l)}\right)$$

and

$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i \frac{\partial \ell}{\partial s_i^{(l)}} w_{i,j}^{(l)}.$$

And then compute the derivatives w.r.t the parameters with

$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)},$$

and

$$\frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}}.$$

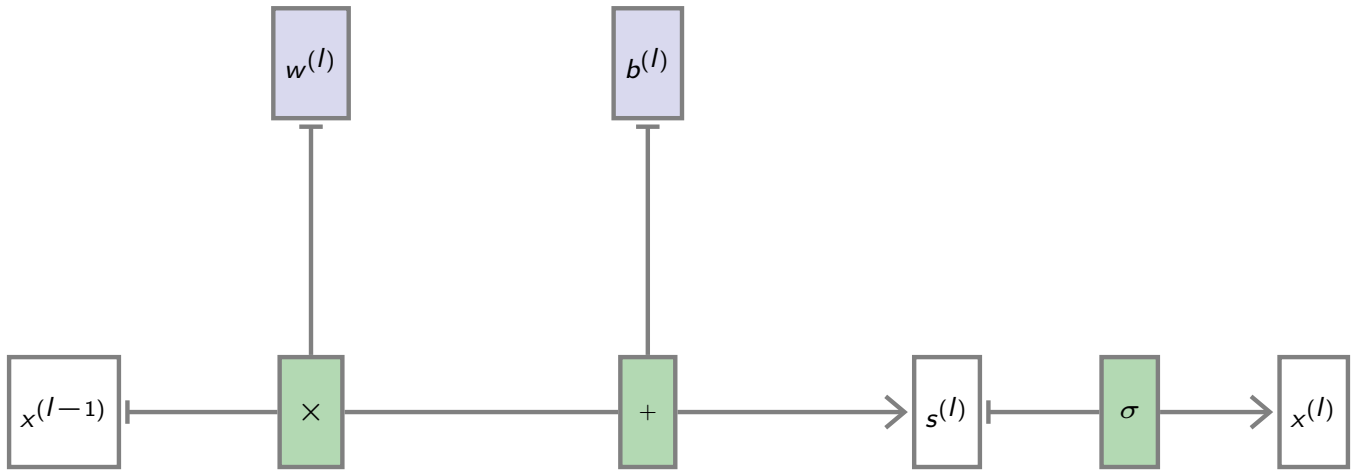
This is the **backward pass**.

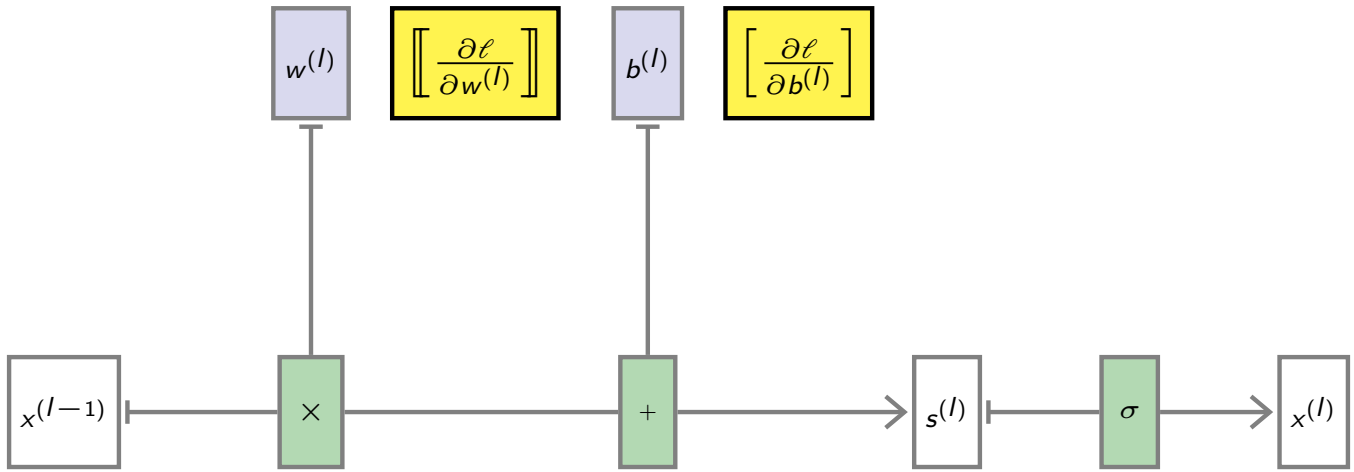
To write in tensorial form we will use the following notation for the gradient of a loss $\ell : \mathbb{R}^N \to \mathbb{R}$,

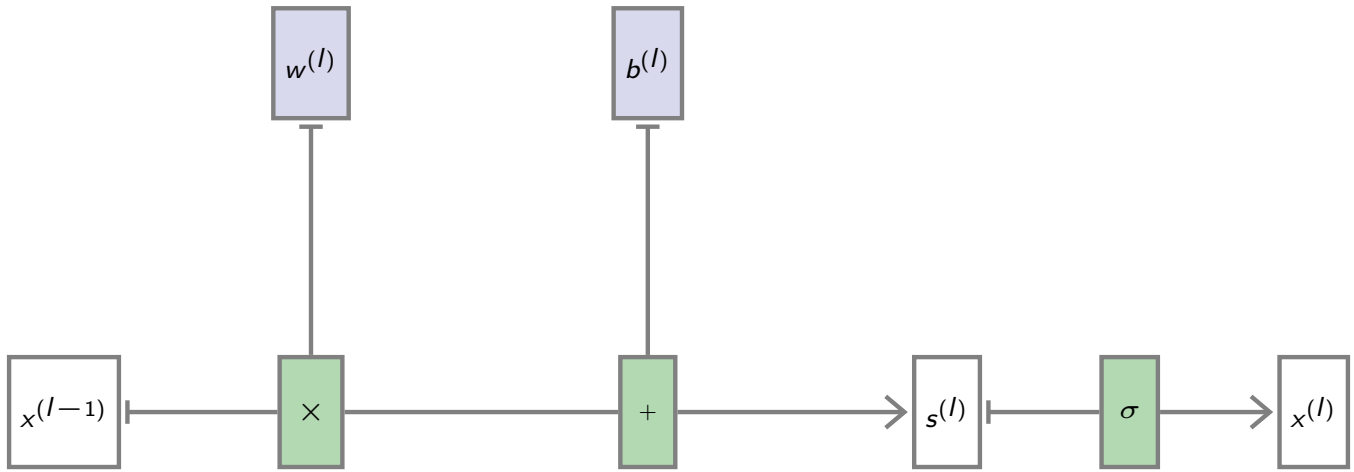
$$\left[\frac{\partial \ell}{\partial x}\right] = \begin{pmatrix} \frac{\partial \ell}{\partial x_1} \\ \vdots \\ \frac{\partial \ell}{\partial x_N} \end{pmatrix},$$

and if $\psi : \mathbb{R}^{N \times M} \to \mathbb{R}$, we will use the notation

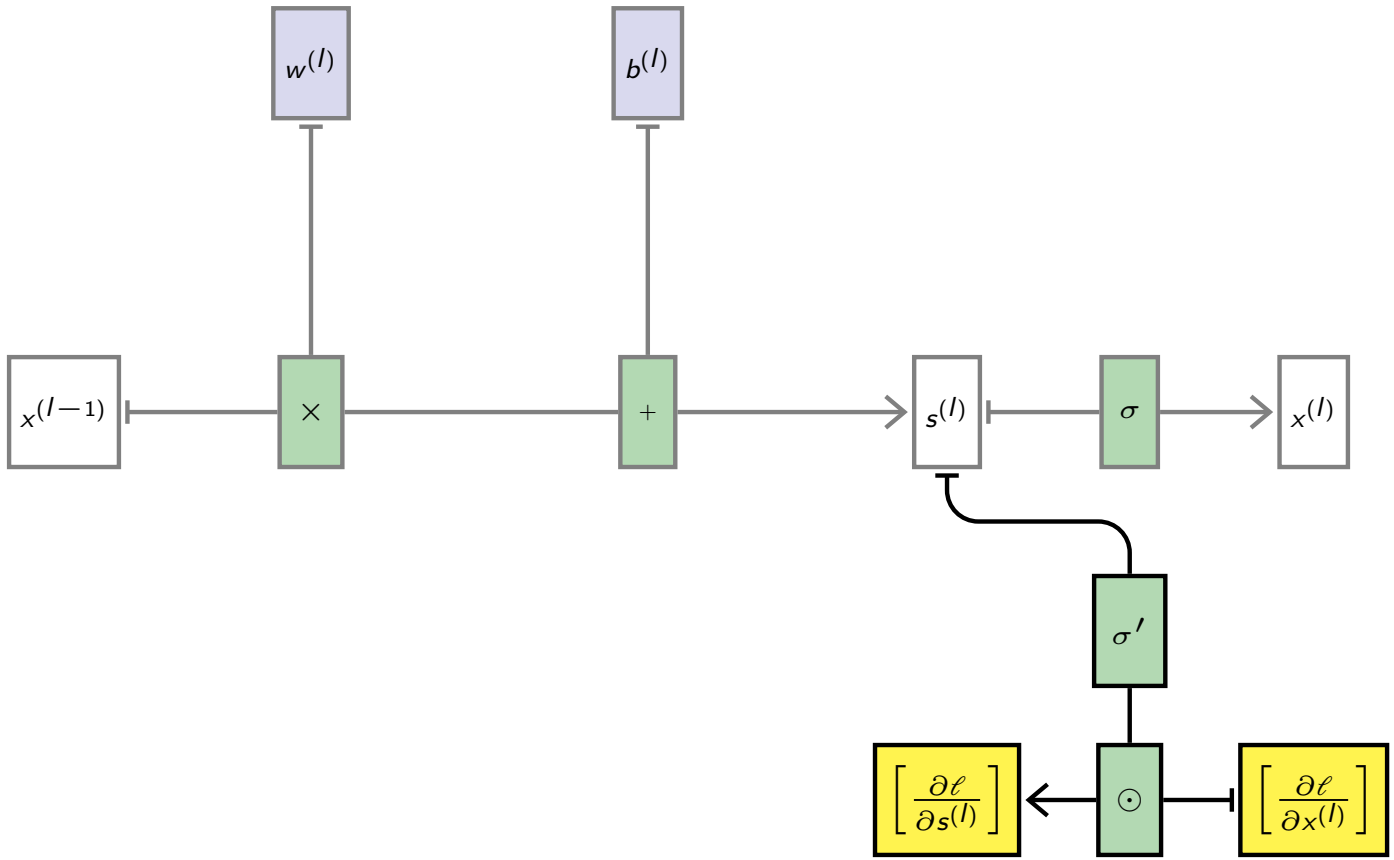
$$\left[\!\left[\frac{\partial \psi}{\partial w}\right]\!\right] = \begin{pmatrix} \frac{\partial \psi}{\partial w_{1,1}} & \cdots & \frac{\partial \psi}{\partial w_{1,M}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi}{\partial w_{N,1}} & \cdots & \frac{\partial \psi}{\partial w_{N,M}} \end{pmatrix}.$$


$w^{(l)}$
$b^{(l)}$
$x^{(l-1)}$
$\times$
$+$
$s^{(l)}$
σ
$x^{(l)}$

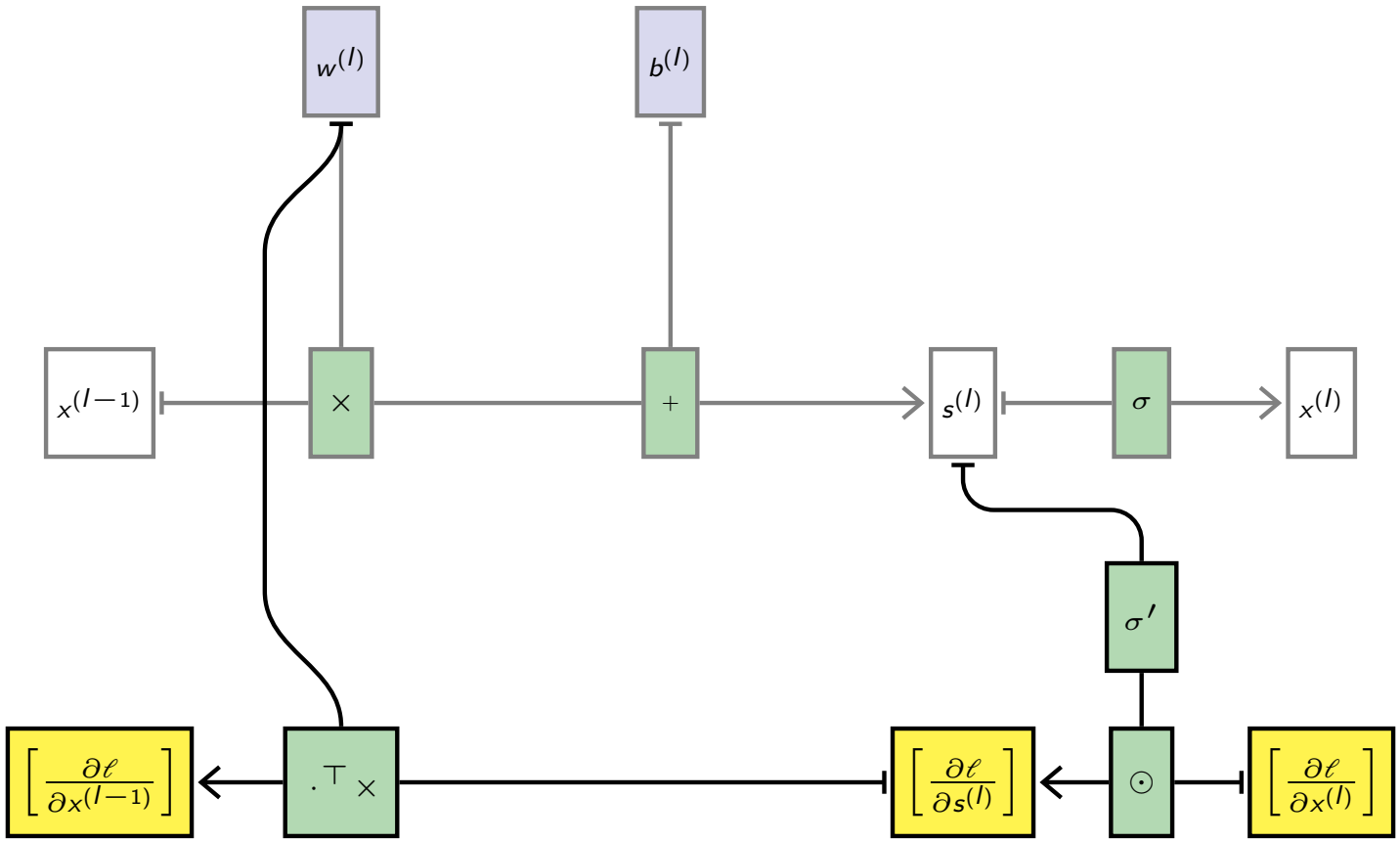

$w^{(l)}$
$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}} \right]\!\right]$
$b^{(l)}$
$\left[\frac{\partial \ell}{\partial b^{(l)}} \right]$
$x^{(l-1)}$
$\times$
$+$
$s^{(l)}$
σ
$x^{(l)}$



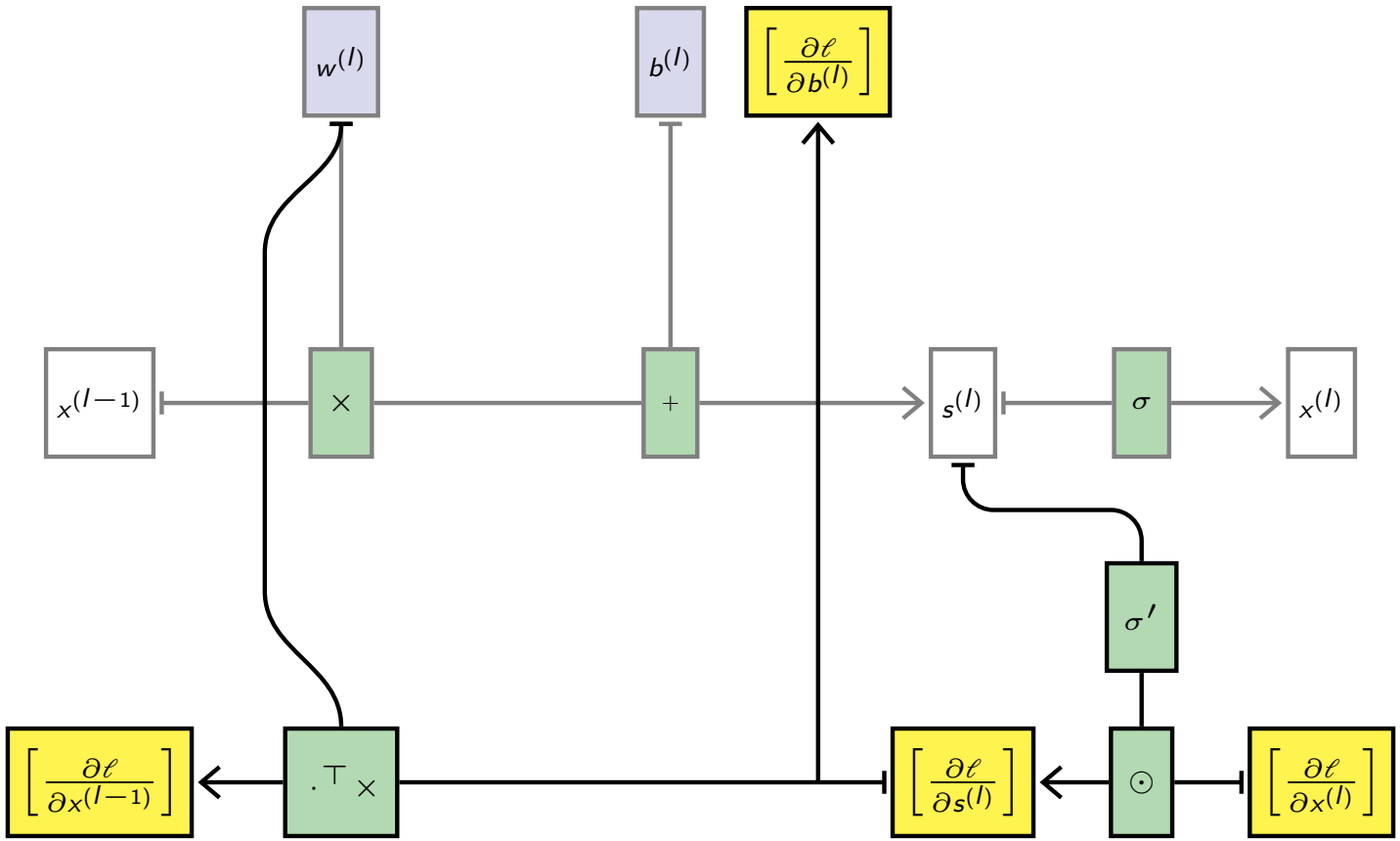
$$\left[\frac{\partial \ell}{\partial x^{(l)}}\right]$$



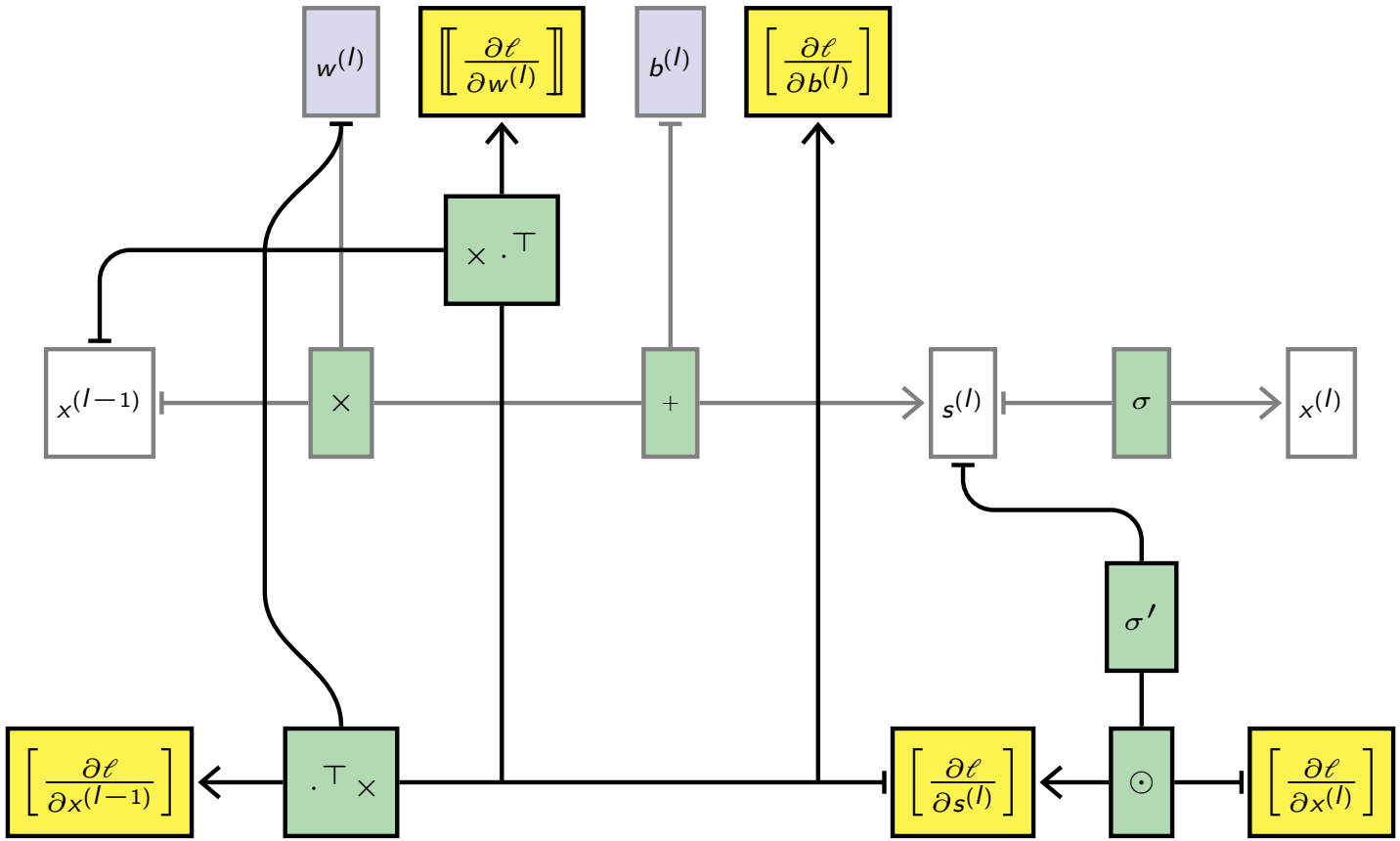
$$\frac{\partial \ell}{\partial s_i^{(l)}} = \frac{\partial \ell}{\partial x_i^{(l)}} \sigma' \left(s_i^{(l)} \right)$$



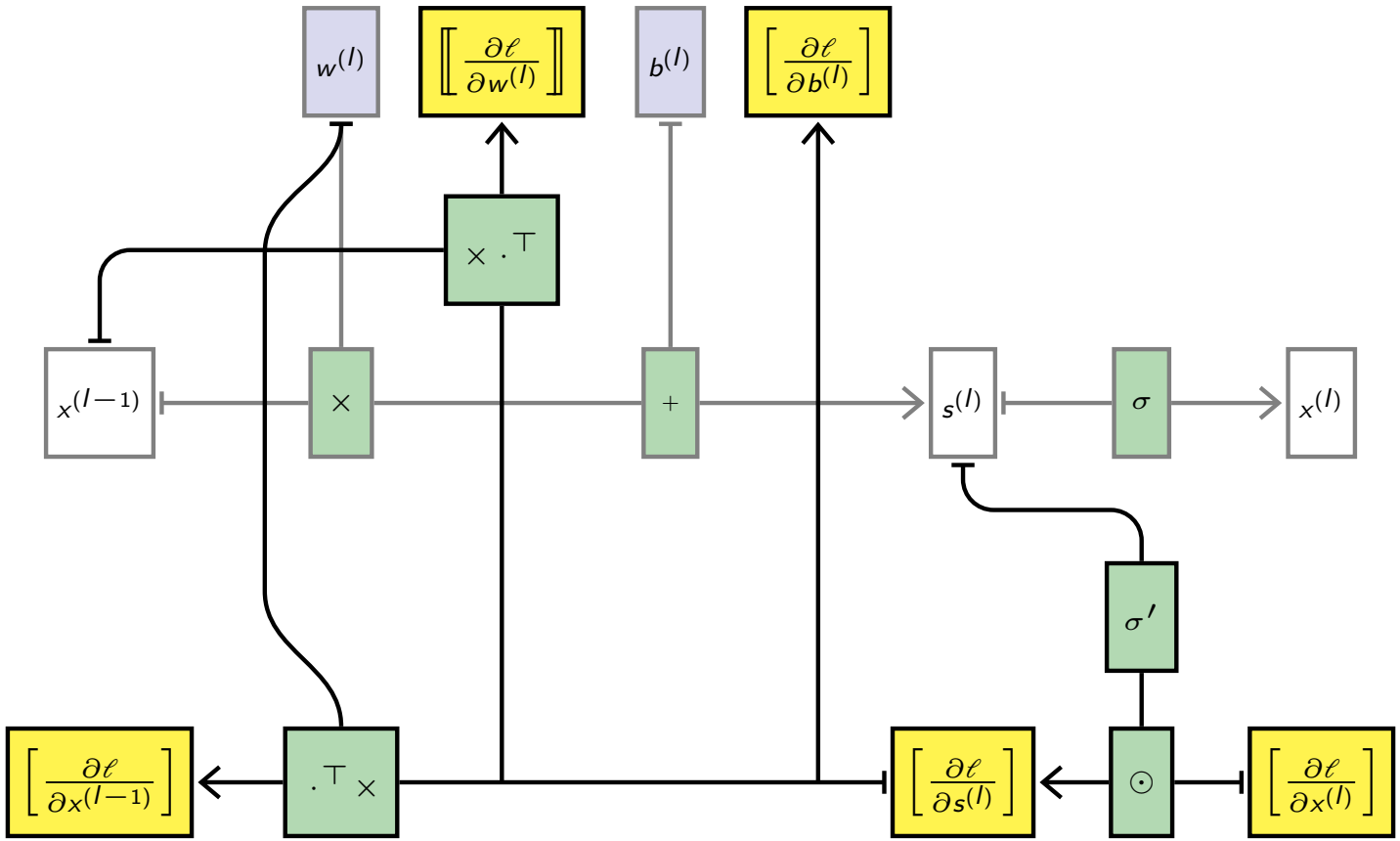
$$\frac{\partial \ell}{\partial x_j^{(l-1)}} = \sum_i w_{i,j}^{(l)} \frac{\partial \ell}{\partial s_i^{(l)}}$$



$$\frac{\partial \ell}{\partial b_i^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}}$$



$$\frac{\partial \ell}{\partial w_{i,j}^{(l)}} = \frac{\partial \ell}{\partial s_i^{(l)}} x_j^{(l-1)}$$


$w^{(l)}$
$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}} \right]\!\right]$
$b^{(l)}$
$\left[\frac{\partial \ell}{\partial b^{(l)}} \right]$
$\times \, .^\top$
$x^{(l-1)}$
$\times$
$+$
$s^{(l)}$
σ
$x^{(l)}$
σ'
$\left[\frac{\partial \ell}{\partial x^{(l-1)}} \right]$
$.^\top \times$
$\left[\frac{\partial \ell}{\partial s^{(l)}} \right]$
$\odot$
$\left[\frac{\partial \ell}{\partial x^{(l)}} \right]$

Forward pass

Compute the activations.

$$x^{(0)} = x, \quad \forall l = 1, \ldots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right) \end{cases}$$

Forward pass

Compute the activations.

$$x^{(0)} = x, \quad \forall l = 1, \ldots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right) \end{cases}$$

Backward pass

Compute the derivatives of the loss w.r.t. the activations.

$$\begin{cases} \left[\frac{\partial \ell}{\partial x^{(L)}}\right] \text{ from the definition of } \ell \\ \text{if } l < L, \left[\frac{\partial \ell}{\partial x^{(l)}}\right] = \left(w^{(l+1)}\right)^\top \left[\frac{\partial \ell}{\partial s^{(l+1)}}\right] \end{cases} \qquad \left[\frac{\partial \ell}{\partial s^{(l)}}\right] = \left[\frac{\partial \ell}{\partial x^{(l)}}\right] \odot \sigma'\left(s^{(l)}\right)$$

Compute the derivatives of the loss w.r.t. the parameters.

$$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right] = \left[\frac{\partial \ell}{\partial s^{(l)}}\right] \left(x^{(l-1)}\right)^\top \qquad \left[\frac{\partial \ell}{\partial b^{(l)}}\right] = \left[\frac{\partial \ell}{\partial s^{(l)}}\right].$$

Forward pass

Compute the activations.

$$x^{(0)} = x, \quad \forall l = 1, \ldots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right) \end{cases}$$

Backward pass

Compute the derivatives of the loss w.r.t. the activations.

$$\begin{cases} \left[\frac{\partial \ell}{\partial x^{(L)}}\right] \text{ from the definition of } \ell \\ \text{if } l < L, \left[\frac{\partial \ell}{\partial x^{(l)}}\right] = \left(w^{(l+1)}\right)^\top \left[\frac{\partial \ell}{\partial s^{(l+1)}}\right] \end{cases} \qquad \left[\frac{\partial \ell}{\partial s^{(l)}}\right] = \left[\frac{\partial \ell}{\partial x^{(l)}}\right] \odot \sigma'\left(s^{(l)}\right)$$

Compute the derivatives of the loss w.r.t. the parameters.

$$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right] = \left[\frac{\partial \ell}{\partial s^{(l)}}\right] \left(x^{(l-1)}\right)^\top \qquad \left[\frac{\partial \ell}{\partial b^{(l)}}\right] = \left[\frac{\partial \ell}{\partial s^{(l)}}\right].$$

Gradient step

Update the parameters.

$$w^{(l)} \leftarrow w^{(l)} - \eta \left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right] \qquad b^{(l)} \leftarrow b^{(l)} - \eta \left[\frac{\partial \ell}{\partial b^{(l)}}\right]$$

In spite of its hairy formalization, the backward pass is a simple algorithm: apply the chain rule again and again.

As for the forward pass, it can be expressed in tensorial form. Heavy computation is concentrated in linear operations, and all the non-linearities go into component-wise operations.

Without tricks, we have to keep in memory all the activations computed during the forward pass.

Regarding computation, since the costly operation for the forward pass is

$$s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)}$$

and for the backward

$$\left[\frac{\partial \ell}{\partial x^{(l)}}\right] = \left(w^{(l+1)}\right)^\top \left[\frac{\partial \ell}{\partial s^{(l+1)}}\right]$$

and

$$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right] = \left[\frac{\partial \ell}{\partial s^{(l)}}\right] \left(x^{(l-1)}\right)^\top,$$

the rule of thumb is that the backward pass is twice more expensive than the forward one.

The end