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.

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.

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

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

its gradient is the mapping

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

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.

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 $\mathscr{L}$.

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

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

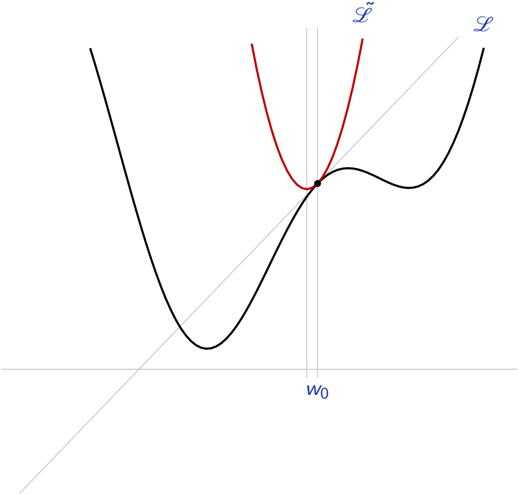
The resulting iterative rule, which goes to the minimum of the approximation at the current location, takes the form:

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

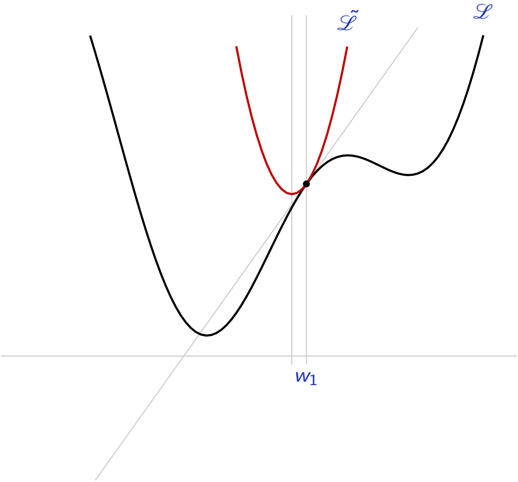
which corresponds intuitively to "following the steepest descent".

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

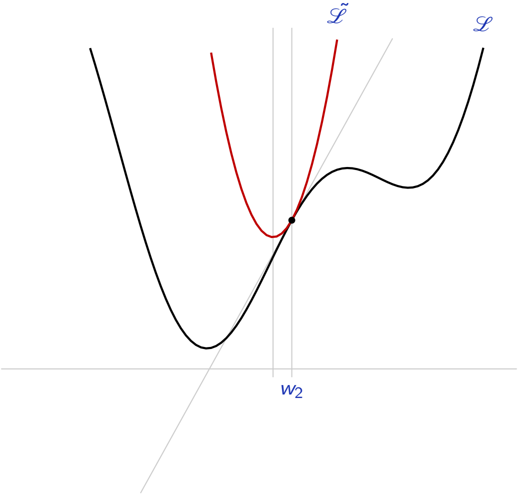
$\eta = 0.125$



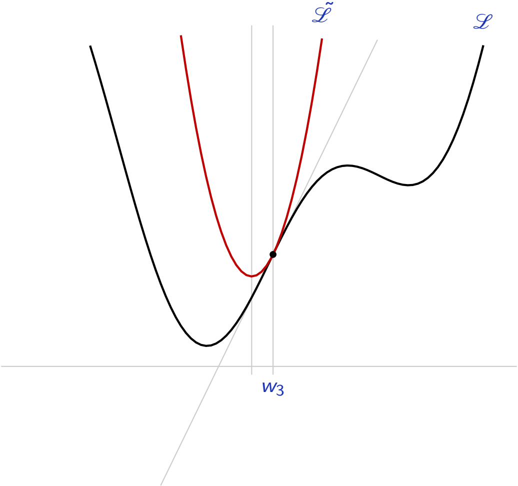
$\eta = 0.125$



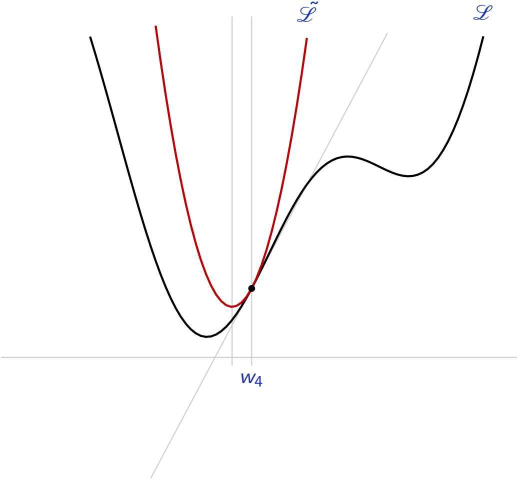
$\eta = 0.125$



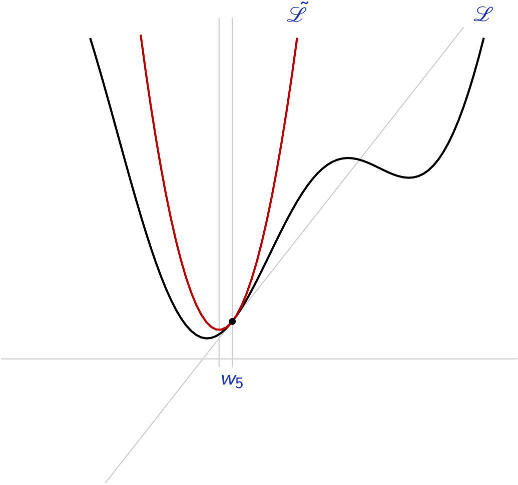
$\eta = 0.125$



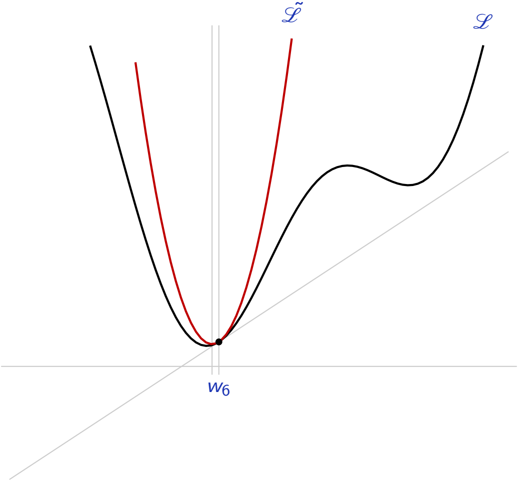
$$\eta = 0.125$$



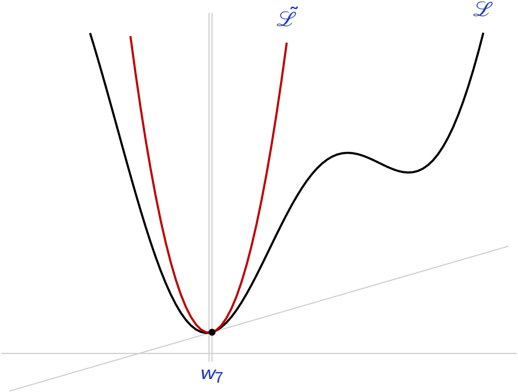
$\eta = 0.125$



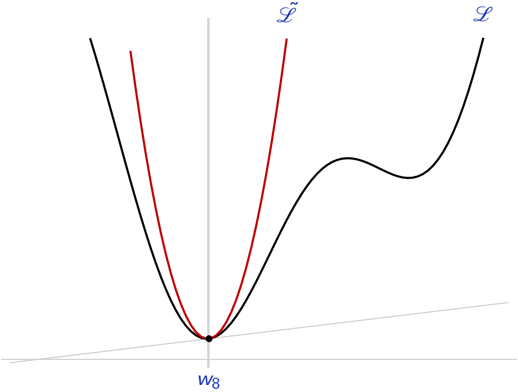
$\eta = 0.125$



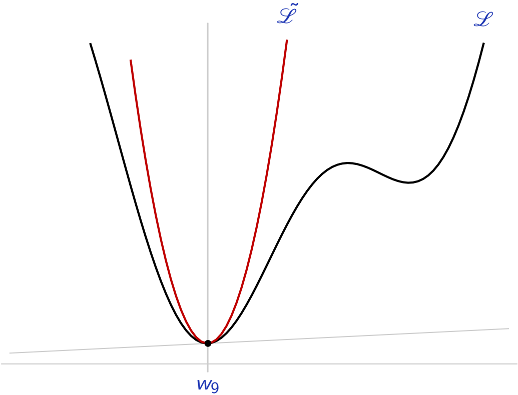
$\eta = 0.125$



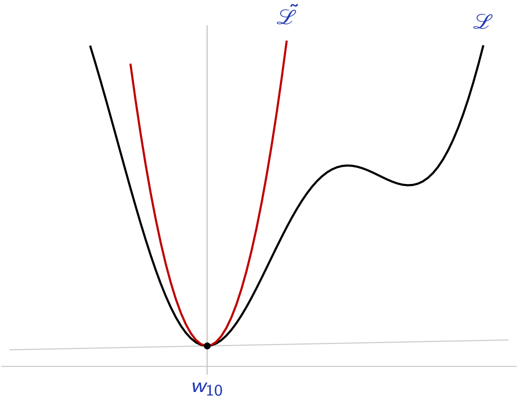
$\eta = 0.125$



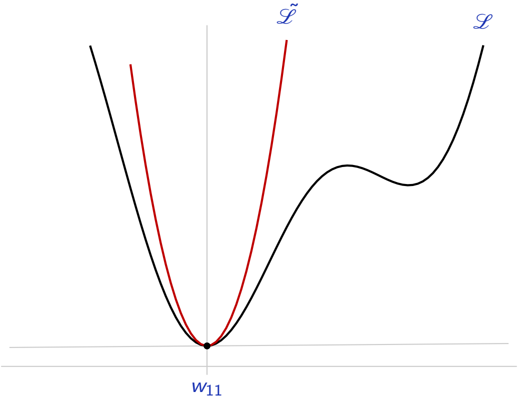
$\eta = 0.125$



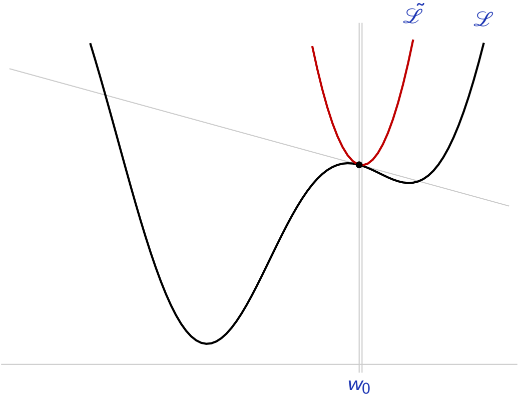
$\eta = 0.125$



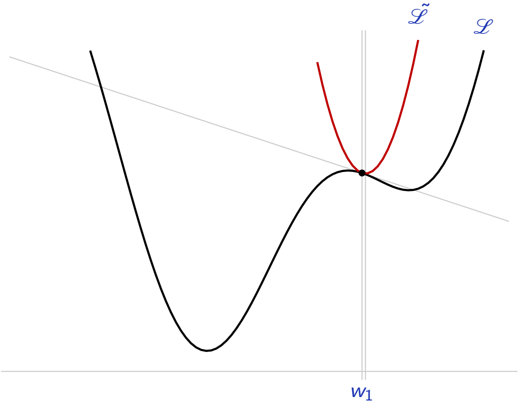
$\eta = 0.125$



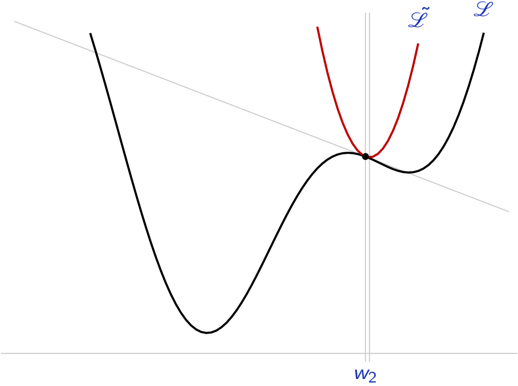
$\eta = 0.125$



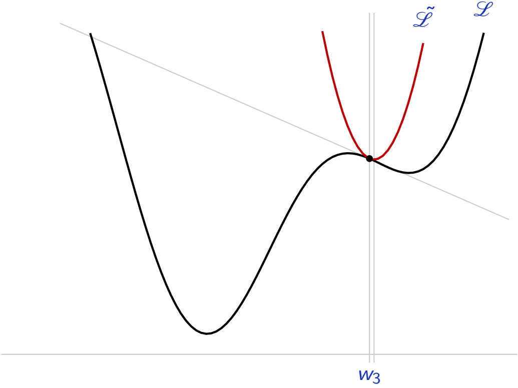
$\eta = 0.125$



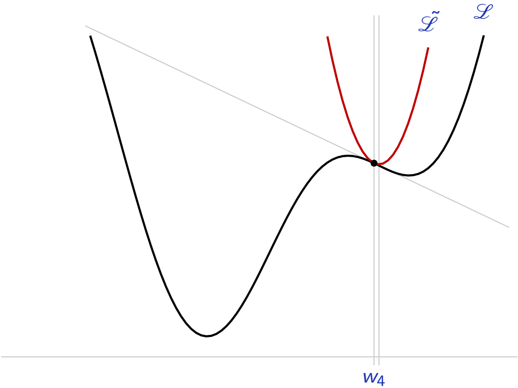
$\eta = 0.125$



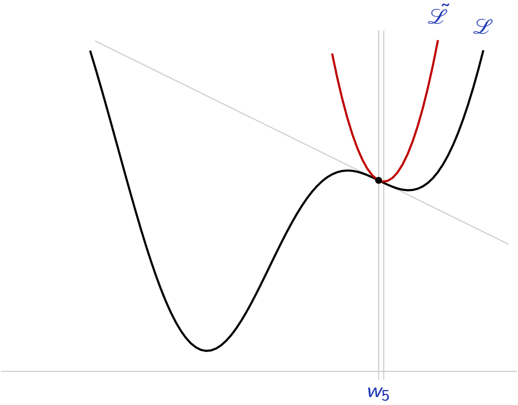
$\eta = 0.125$



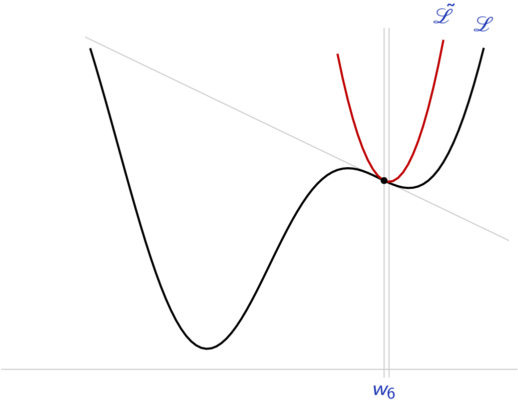
$\eta = 0.125$



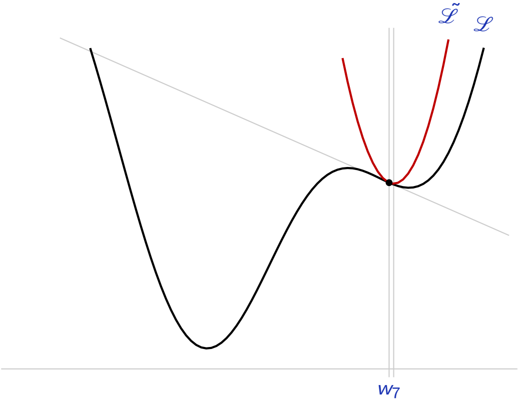
$\eta = 0.125$



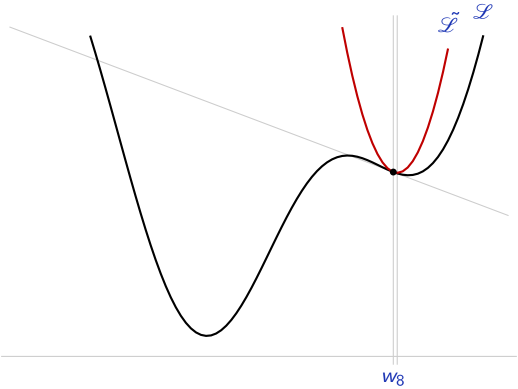
$\eta = 0.125$



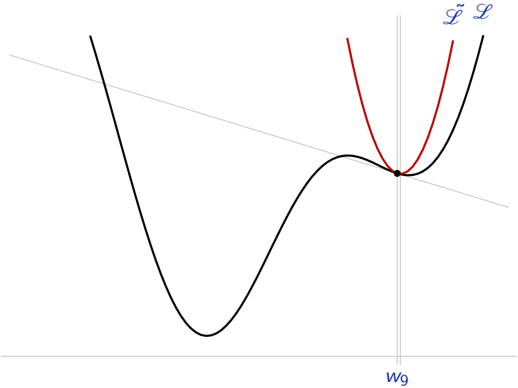
$\eta = 0.125$



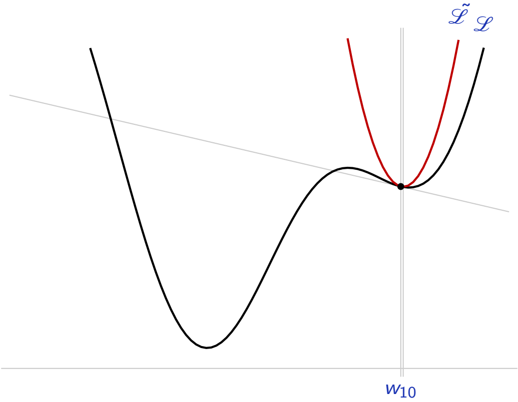
$\eta = 0.125$



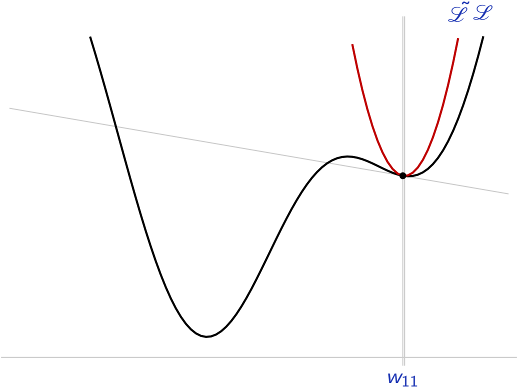
$\eta = 0.125$



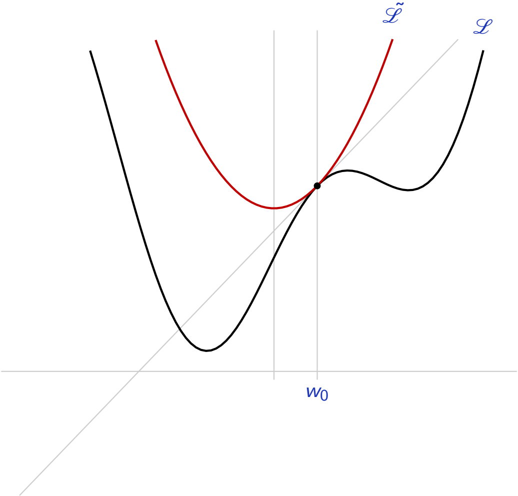
$\eta = 0.125$



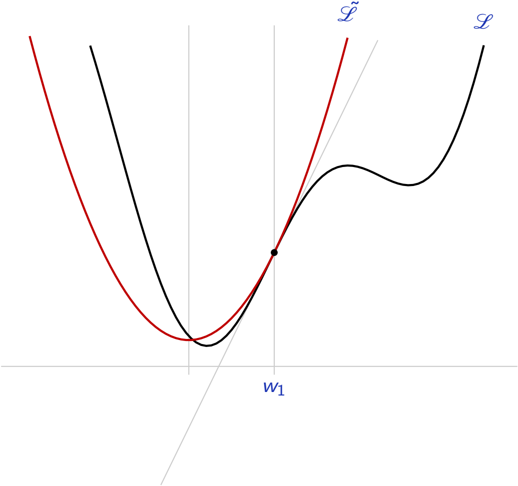
$\eta = 0.125$



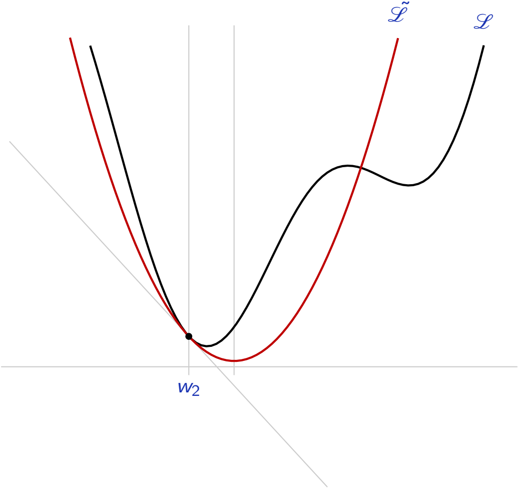
$\eta = 0.5$



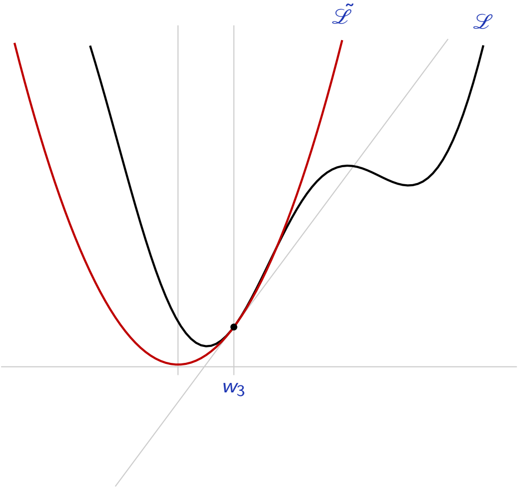
$\eta = 0.5$



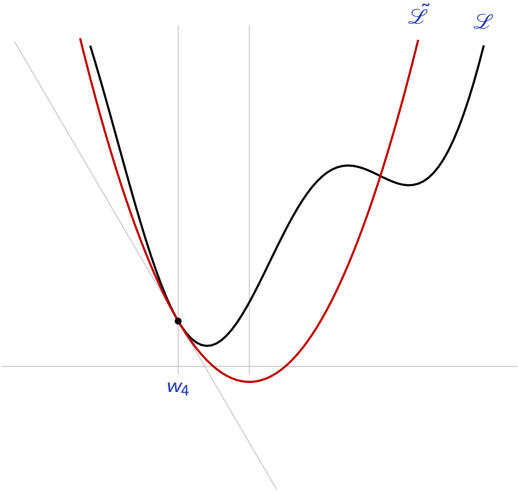
$\eta = 0.5$



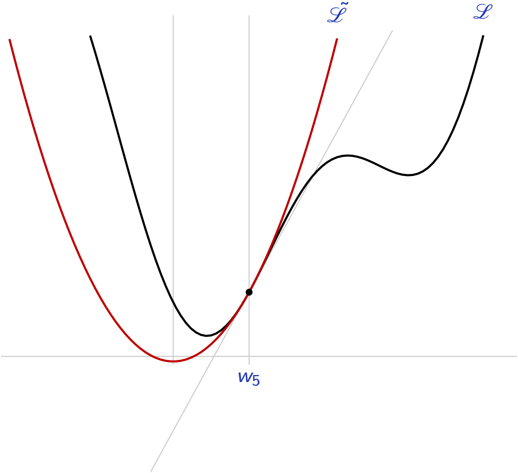
$\eta = 0.5$



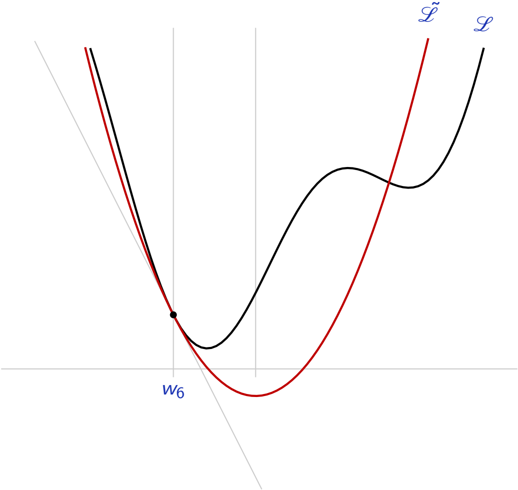
$\eta = 0.5$



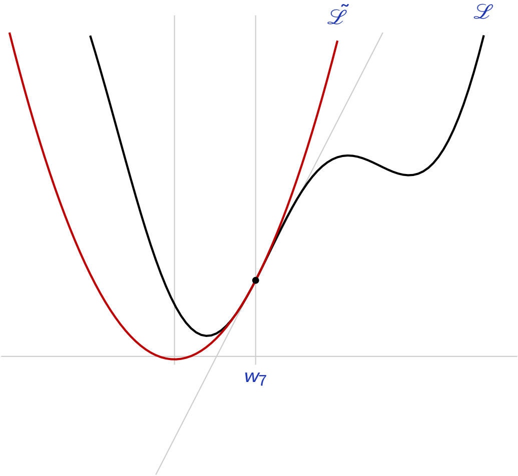
$\eta = 0.5$



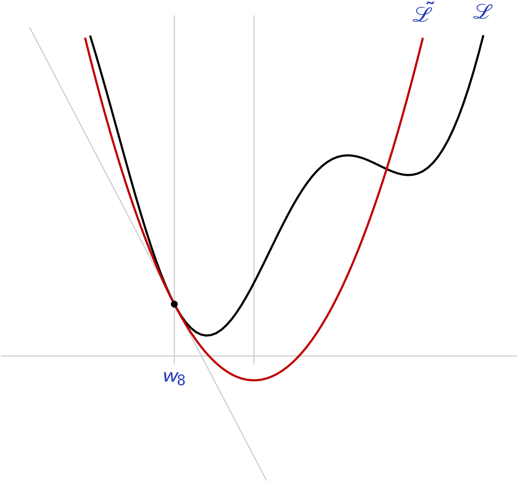
$\eta = 0.5$



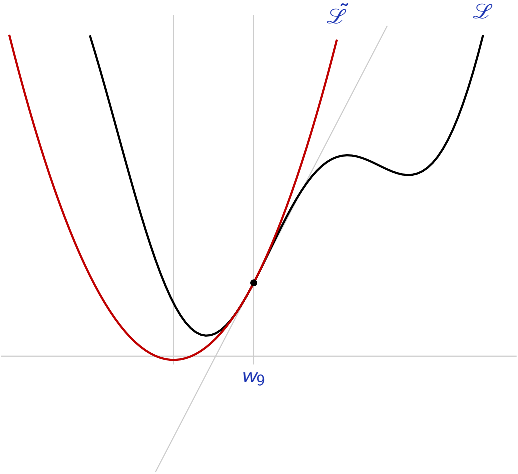
$\eta = 0.5$



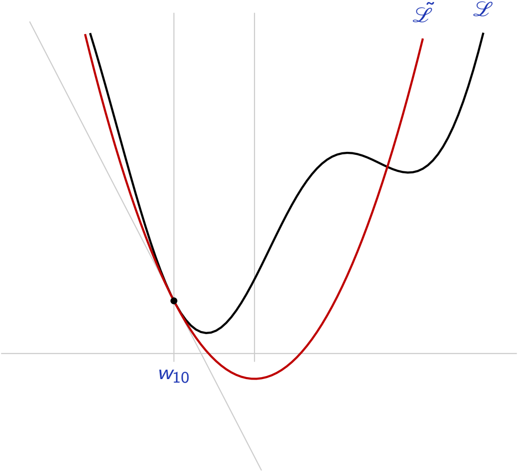
$\eta = 0.5$



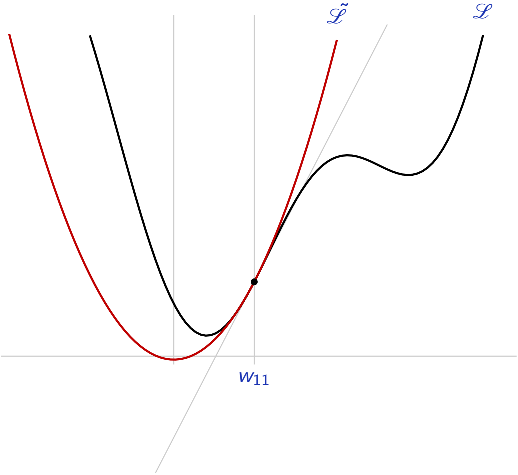
$\eta = 0.5$

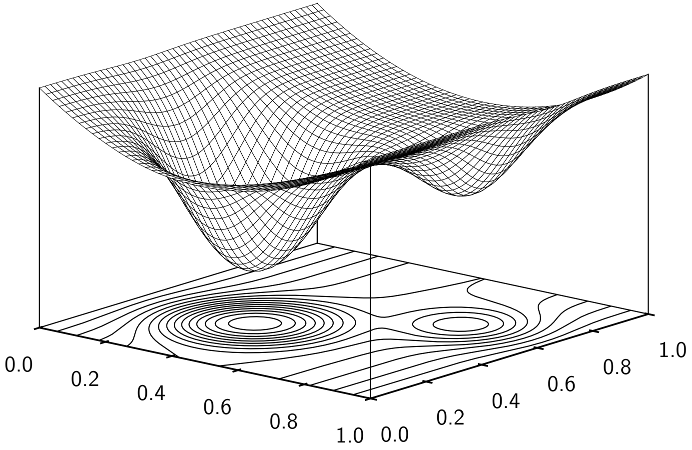


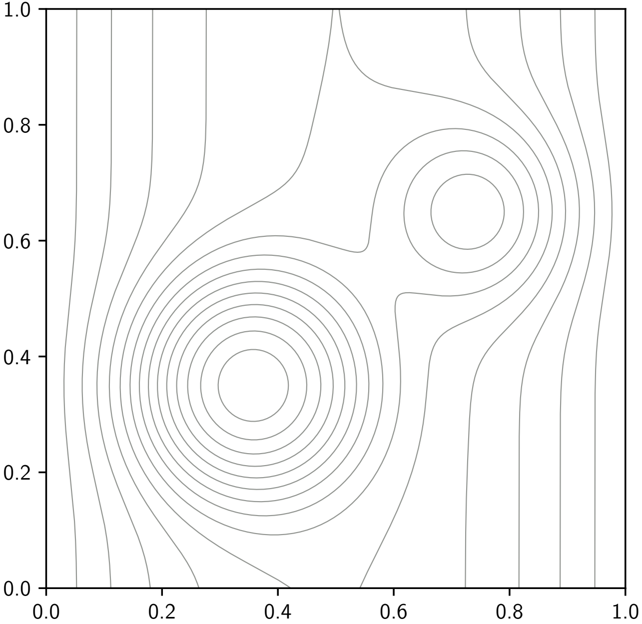
$\eta = 0.5$

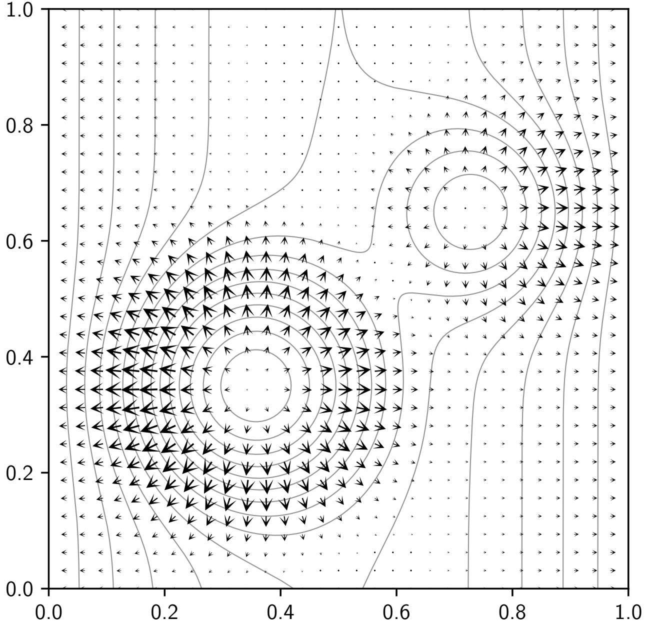


$\eta = 0.5$

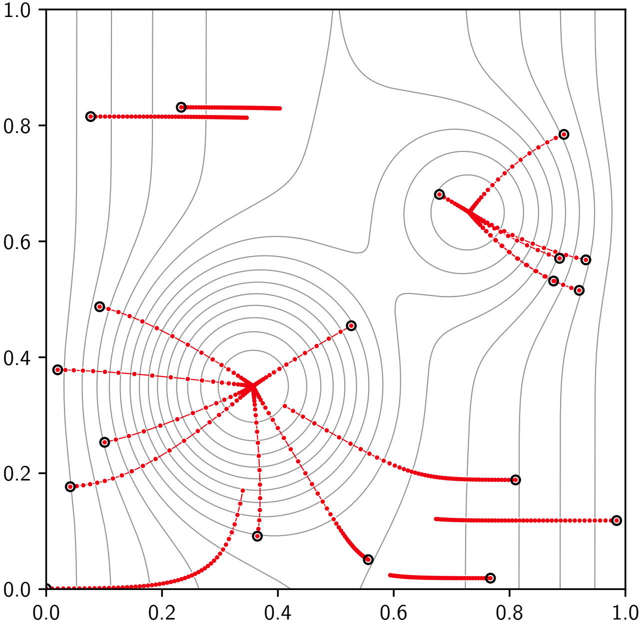



0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0


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

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

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

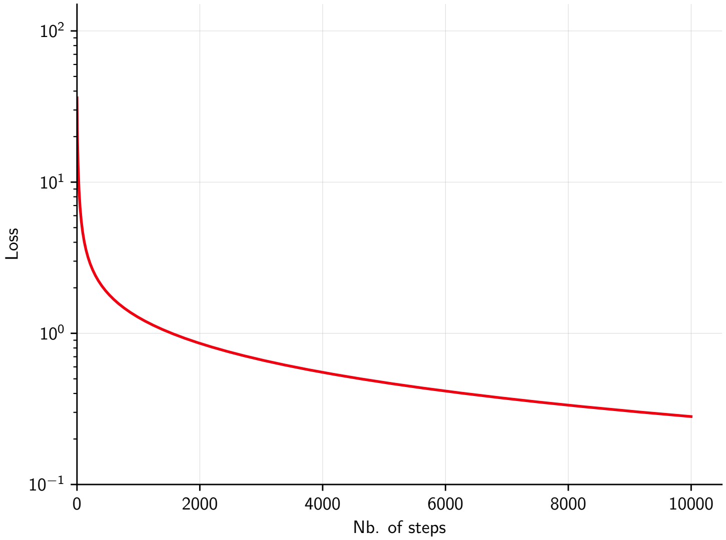
which can be implemented as

```
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)
```

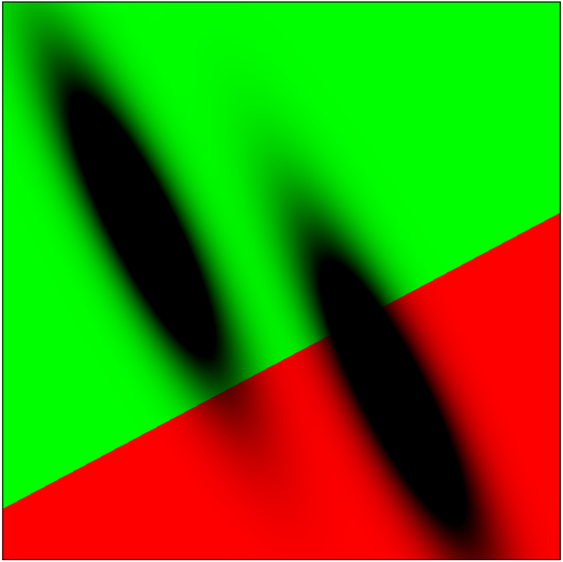
and the gradient descent as

```
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
```

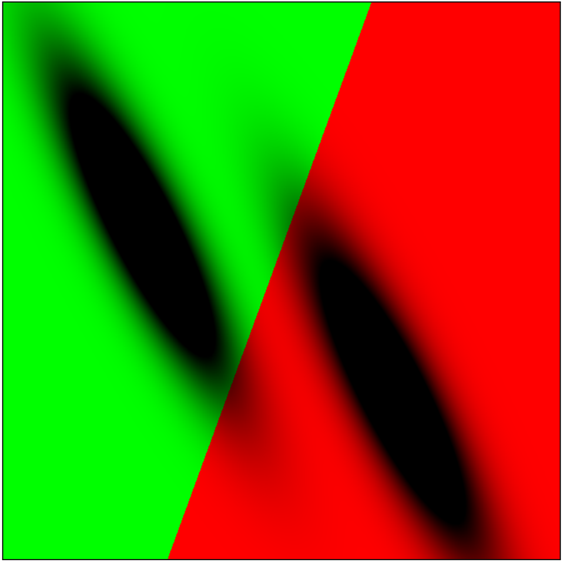

10^2
10^1
10^0
10^{-1}
Loss
0
2000
4000
6000
8000
10000
Nb. of steps

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



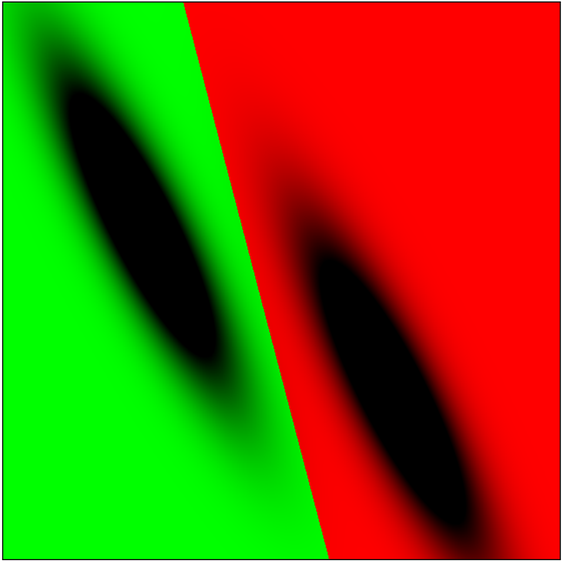
$n = 0$

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



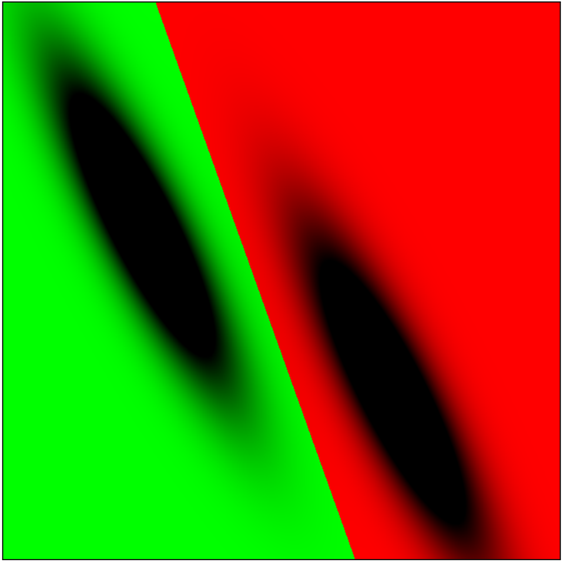
$n = 10$

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



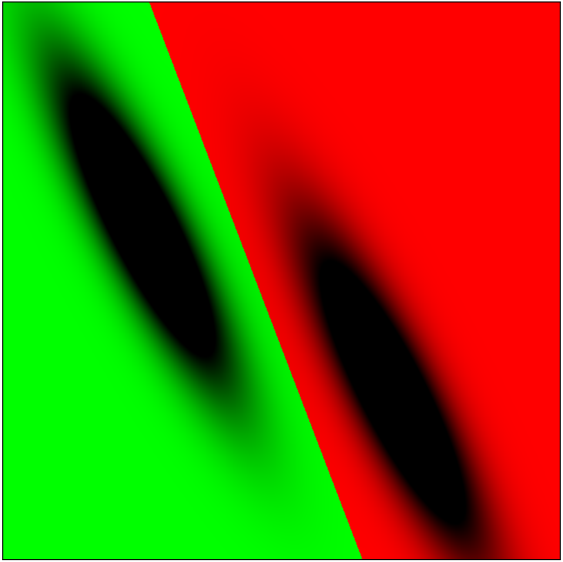
$n = 10^2$

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

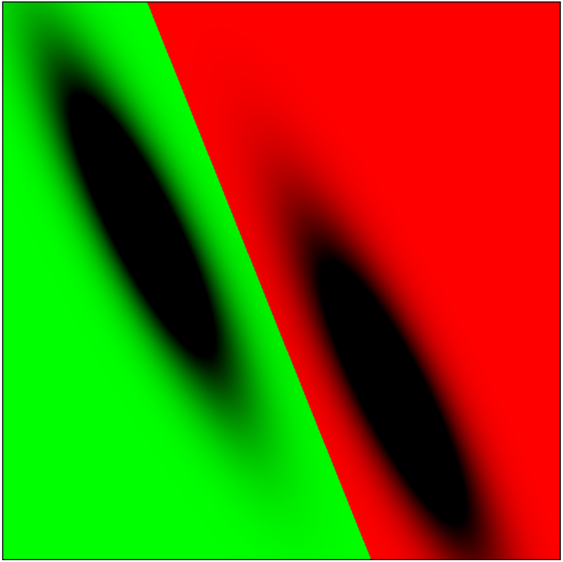


$n = 10^3$

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



$n = 10^4$



LDA

The end