Introduction to Deep Learning

Copernicus Master on Digital Earth

Lecture 2: Neurons, neural networks and back-propagation

Prof. Nicolas Courty
ncourty@irisa.fr

Today

Explain and motivate the basic constructs of neural networks.

- From linear discriminant analysis to logistic regression
- Stochastic gradient descent
- From logistic regression to the multi-layer perceptron
- Vanishing gradients and rectified networks
- Universal approximation theorem

Today

Explain and motivate the basic constructs of neural networks.

- What is a neuron ? A little bit of history
- From linear discriminant analysis to logistic regression
- Refresher on math derivatives/gradient
- Stochastic gradient descent

Neural networks

Threshold Logic Unit (TLU)

The Threshold Logic Unit (McCulloch and Pitts, 1943) was the first mathematical model for a neuron. Assuming Boolean inputs and outputs, it is defined as:

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

- What are the parameters w_i and b needed to implement
  - $\text{or}(a, b) =$
  - $\text{and}(a, b) =$
  - $\text{not}(a) =$

Threshold Logic Unit (TLU)

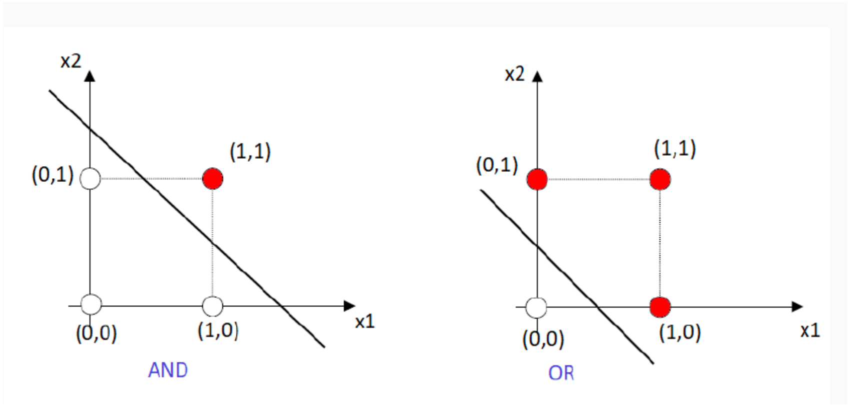
The Threshold Logic Unit (McCulloch and Pitts, 1943) was the first mathematical model for a neuron. Assuming Boolean inputs and outputs, it is defined as:

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

- What are the parameters w_i and b needed to implement
    - $\mathrm{or}(a, b) = 1_{\{a+b-0.5\geq 0\}}$
    - $\mathrm{and}(a, b) = 1_{\{a+b-1.5\geq 0\}}$
    - $\mathrm{not}(a) = 1_{\{-a+0.5\geq 0\}}$
- can you also find $\mathrm{xor}(a, b) =$?

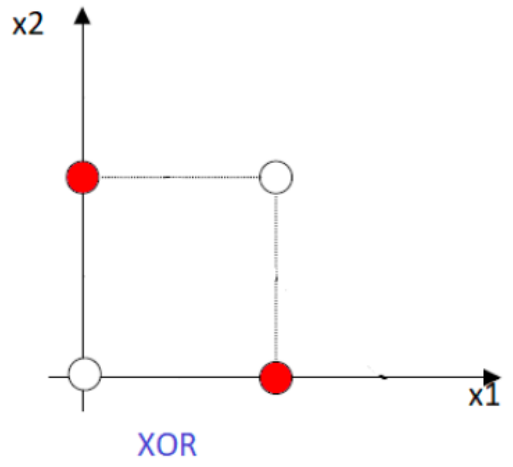
Threshold Logic Unit (TLU)

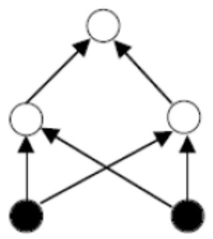
- single TLU is limited in its capacity
- why does the TLU cannot perform this task?
- To see let's plot the OR and AND in the two dimensional space



- Basically, TLU finds a decision boundary to separate the binary outcomes
- Likewise can you plot XOR in a $2D$ space ?

Threshold Logic Unit (TLU)





- we need two lines to separate the outcomes
- a single TLU is limited in its capacity as the complexity of the model grows. We need more TLUs to solve the task
- XOR problem can be solved using five TLUs. Please find the solution !

Threshold Logic Unit (TLU)

- $\mathrm{xor}(a, b) = \mathrm{or}(\mathrm{and}(\mathrm{not}(a), b), \mathrm{and}(a, \mathrm{not}(b))$

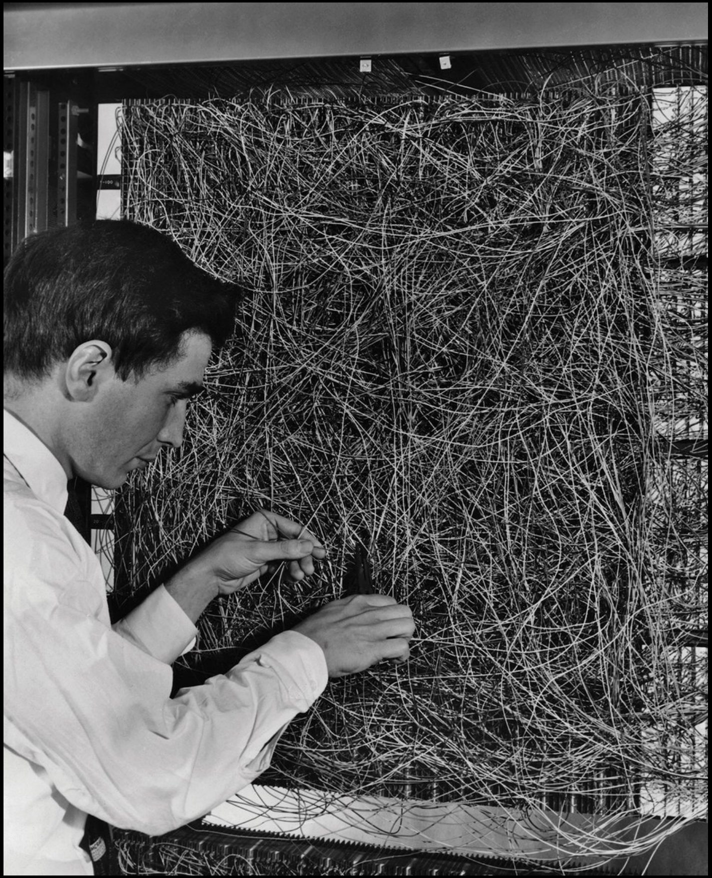
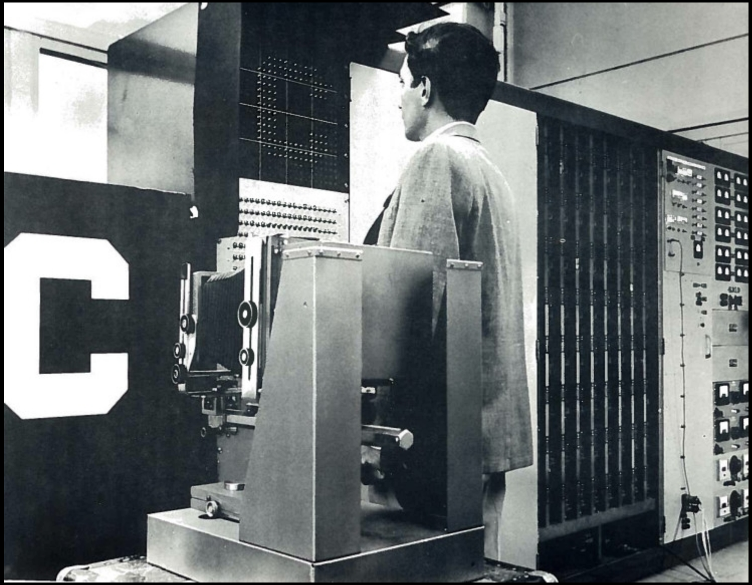
Therefore, any Boolean function can be built with such units.

Perceptron

The perceptron (Rosenblatt, 1957) is very similar, except that the inputs are real:

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

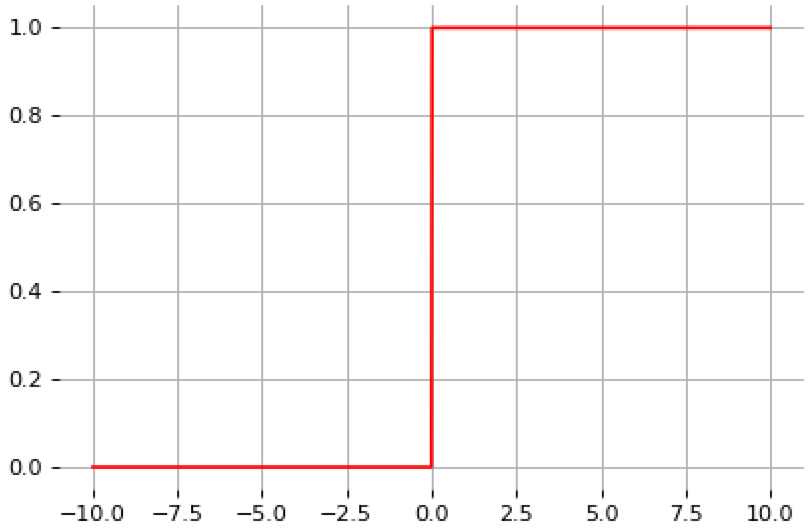
This model was originally motivated by biology, with w_i being synaptic weights and x_i and f firing rates.

The Mark I Percetron (Frank Rosenblatt).

Let us define the (non-linear) activation function:

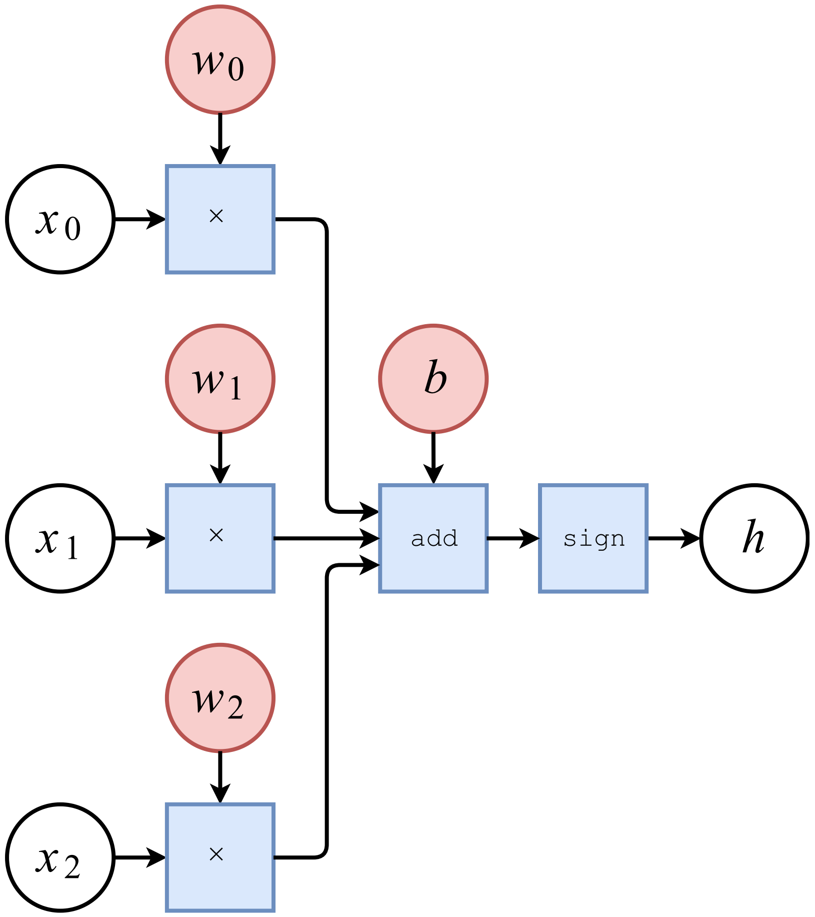
$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



The perceptron classification rule can be rewritten as

$$f(\mathbf{x}) = \text{sign}(\sum_i w_i x_i + b).$$

Computational graphs



The computation of

$$f(\mathbf{x}) = \text{sign}(\sum_i w_i x_i + b)$$

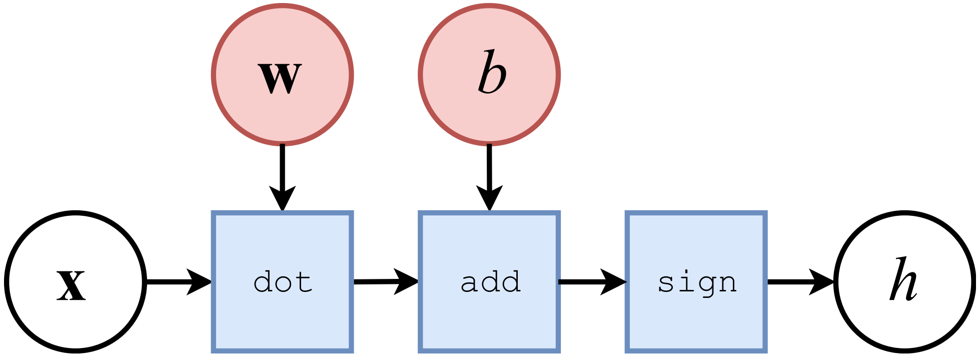
can be represented as a computational graph where

- white nodes correspond to inputs and outputs;
- red nodes correspond to model parameters;
- blue nodes correspond to intermediate operations.

In terms of tensor operations, f can be rewritten as

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b),$$

for which the corresponding computational graph of f is:



Linear Discriminant Analysis

Consider training data $(\mathbf{x}, y) \sim P(X, Y)$, with

- $\mathbf{x} \in \mathbb{R}^p$,
- $y \in \{0, 1\}$.

Assume class populations are Gaussian, with same covariance matrix Σ (homoscedasticity):

$$P(\mathbf{x}|y) = \frac{1}{\sqrt{(2\pi)^p|\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_y)^T \Sigma^{-1} (\mathbf{x} - \mu_y)\right)$$

Using the Bayes' rule, we have:

$$\begin{aligned}
P(Y=1|\mathbf{x}) &= \frac{P(\mathbf{x}|Y=1)P(Y=1)}{P(\mathbf{x})} \\
&= \frac{P(\mathbf{x}|Y=1)P(Y=1)}{P(\mathbf{x}|Y=0)P(Y=0)+P(\mathbf{x}|Y=1)P(Y=1)} \\
&= \frac{1}{1+\frac{P(\mathbf{x}|Y=0)P(Y=0)}{P(\mathbf{x}|Y=1)P(Y=1)}}.
\end{aligned}$$

Using the Bayes' rule, we have:

$$\begin{aligned}
P(Y=1|\mathbf{x}) &= \frac{P(\mathbf{x}|Y=1)P(Y=1)}{P(\mathbf{x})} \\
&= \frac{P(\mathbf{x}|Y=1)P(Y=1)}{P(\mathbf{x}|Y=0)P(Y=0)+P(\mathbf{x}|Y=1)P(Y=1)} \\
&= \frac{1}{1+\frac{P(\mathbf{x}|Y=0)P(Y=0)}{P(\mathbf{x}|Y=1)P(Y=1)}}.
\end{aligned}$$

It follows that with

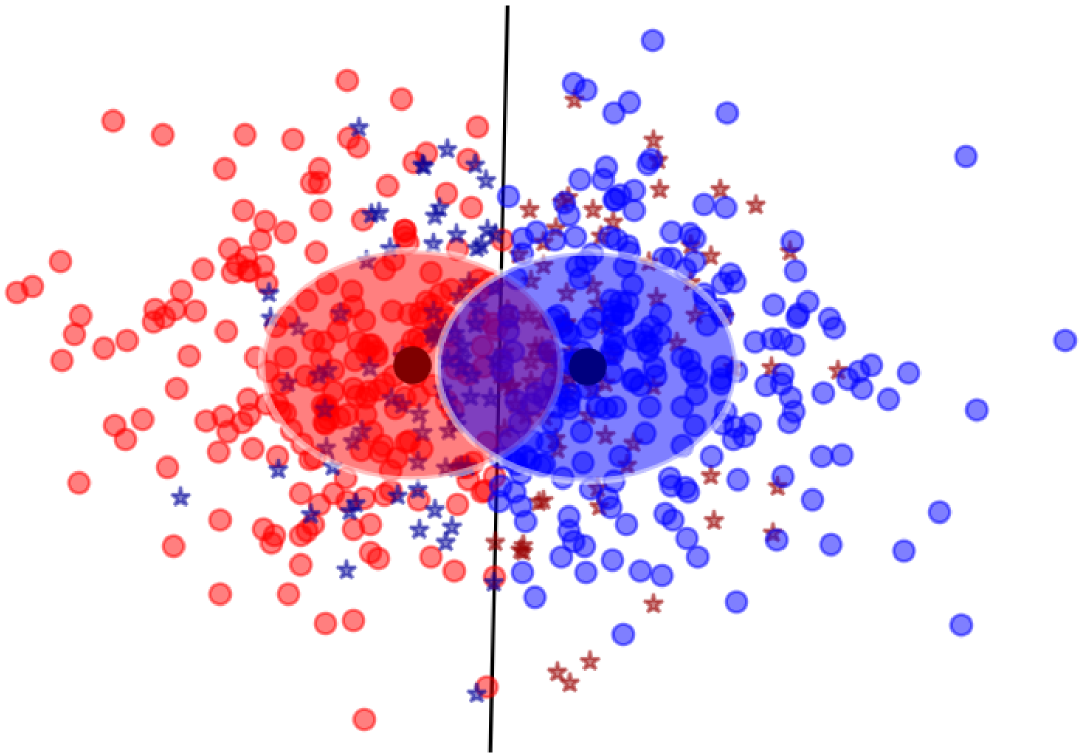
$$\sigma(x) = \frac{1}{1+\exp(-x)},$$

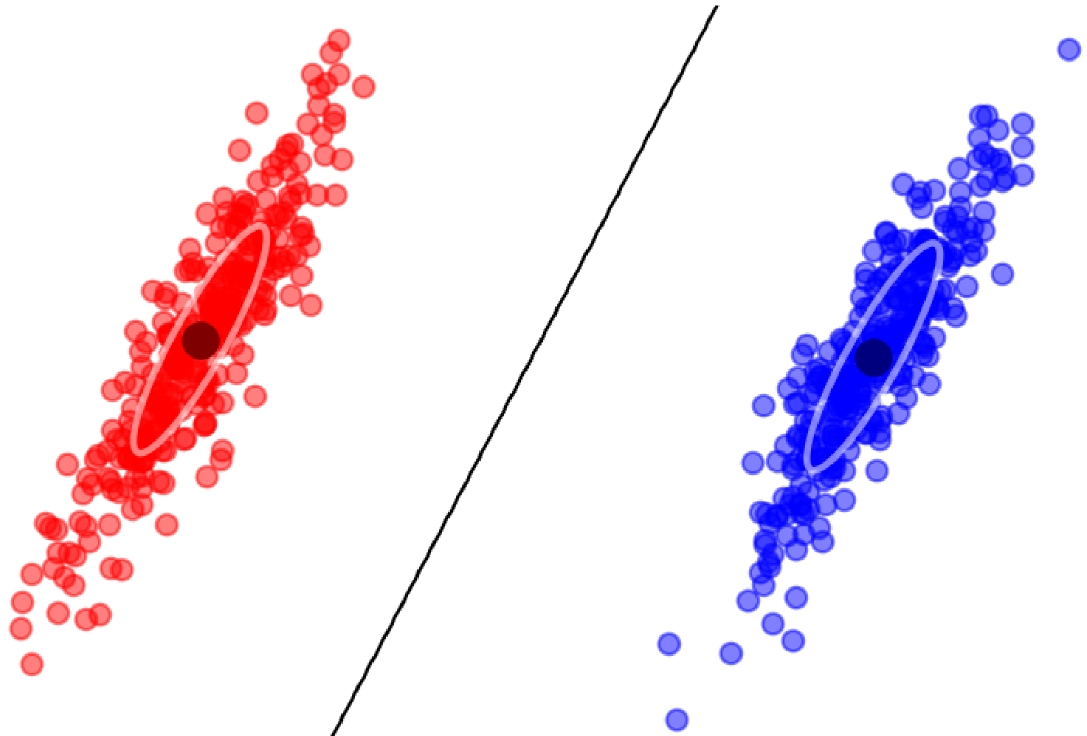
we get

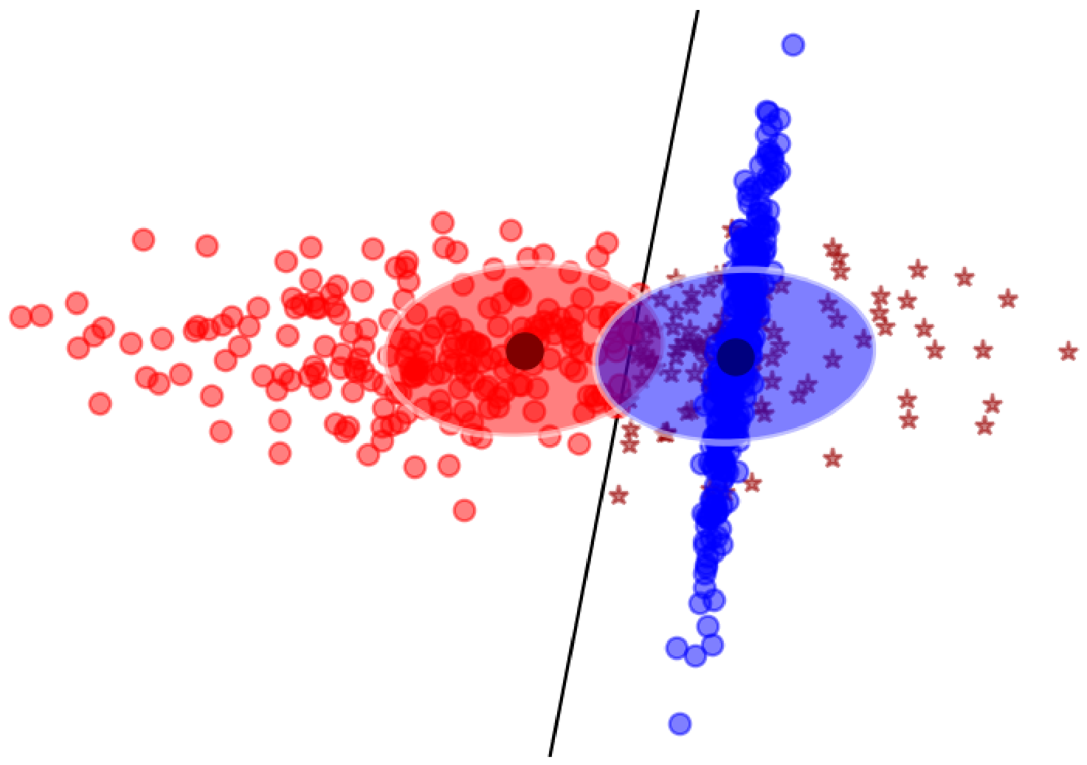
$$P(Y=1|\mathbf{x}) = \sigma\left(\log\frac{P(\mathbf{x}|Y=1)}{P(\mathbf{x}|Y=0)} + \log\frac{P(Y=1)}{P(Y=0)}\right).$$

Therefore,

$$
\begin{aligned}
&P(Y=1|\mathbf{x}) \\
&= \sigma\left(\log\frac{P(\mathbf{x}|Y=1)}{P(\mathbf{x}|Y=0)} + \underbrace{\log\frac{P(Y=1)}{P(Y=0)}}_{a}\right) \\
&= \sigma\left(\log P(\mathbf{x}|Y=1) - \log P(\mathbf{x}|Y=0) + a\right) \\
&= \sigma\left(-\frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1) + \frac{1}{2}(\mathbf{x}-\mu_0)^T\Sigma^{-1}(\mathbf{x}-\mu_0) + a\right) \\
&= \sigma\left(\underbrace{(\mu_1-\mu_0)^T\Sigma^{-1}}_{\mathbf{w}^T}\mathbf{x} + \underbrace{\frac{1}{2}(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1) + a}_{b}\right) \\
&= \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)
\end{aligned}
$$



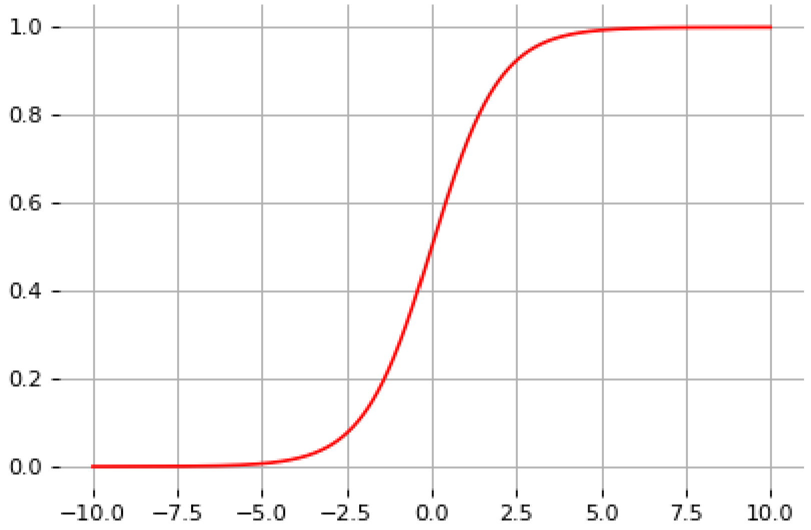




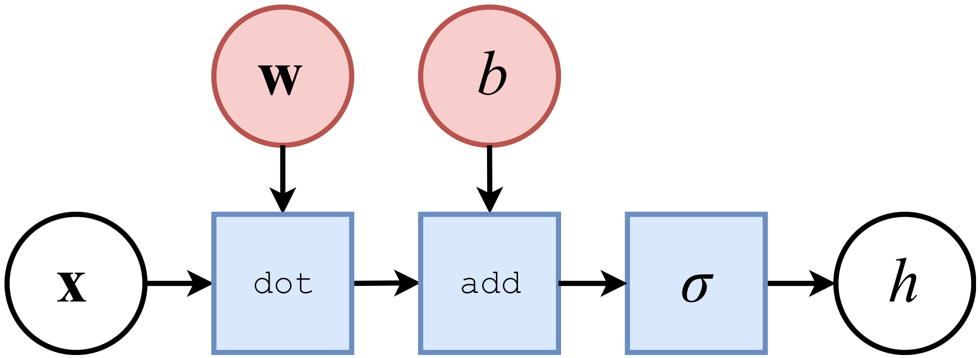
Note that the sigmoid function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

looks like a soft heavyside:



Therefore, the overall model $f(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{w}^T\mathbf{x} + b)$ is very similar to the perceptron.



This unit is the lego brick of all neural networks!

Logistic regression

Same model

$$P(Y = 1|\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$$

as for linear discriminant analysis.

But,

- ignore model assumptions (Gaussian class populations, homoscedasticity);
- instead, find $\mathbf{w}, b$ that maximizes the likelihood of the data.

We have,

$$
\begin{aligned}
&\arg\max_{\mathbf{w},b} P(\mathbf{d}|\mathbf{w},b) \\
&= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} P(Y=y_i|\mathbf{x}_i,\mathbf{w},b) \\
&= \arg\max_{\mathbf{w},b} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} \sigma(\mathbf{w}^T\mathbf{x}_i+b)^{y_i}(1-\sigma(\mathbf{w}^T\mathbf{x}_i+b))^{1-y_i} \\
&= \arg\min_{\mathbf{w},b} \underbrace{\sum_{\mathbf{x}_i,y_i\in\mathbf{d}} -y_i\log\sigma(\mathbf{w}^T\mathbf{x}_i+b)-(1-y_i)\log(1-\sigma(\mathbf{w}^T\mathbf{x}_i+b))}_{\mathcal{L}(\mathbf{w},b)=\sum_i \ell(y_i,\hat{y}(\mathbf{x}_i;\mathbf{w},b))}
\end{aligned}
$$

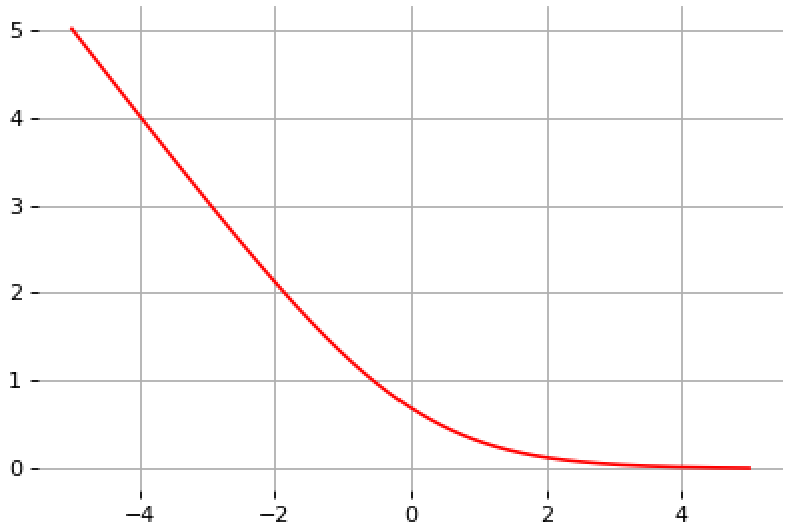
This loss is an instance of the cross-entropy

$$H(p,q)=\mathbb{E}_p[-\log q]$$

for $p=Y|\mathbf{x}_i$ and $q=\hat{Y}|\mathbf{x}_i$.

When Y takes values in $\{-1, 1\}$, a similar derivation yields the logistic loss

$$\mathcal{L}(\mathbf{w}, b) = -\sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \log \sigma \left(y_i (\mathbf{w}^T \mathbf{x}_i + b) \right).$$



- In general, the cross-entropy and the logistic losses do not admit a minimizer that can be expressed analytically in closed form.
- However, a minimizer can be found numerically, using a general minimization technique such as gradient descent.

Math refresher: derivatives

Math refresher: derivatives

One can obtain useful information about a function f by looking at its derivative

$$f'(x) = \frac{df(x)}{dx} = \lim_{\Delta \to 0} \frac{f(x+\Delta) - f(x)}{\Delta}$$

- the sign of the derivative is the direction of increase of f
    - positive: $f(x)$ increases as x increases
    - negative: $f(x)$ decreases as x increases
- the absolute value of the derivative is the rate of change
- instead of d, we will use the ∂ sign

Math refresher: derivatives

Useful derivatives (to know instantly)

- let a and n be constants, then:

$$\frac{\partial a}{\partial x} = 0$$
$$\frac{\partial x^n}{\partial x} = nx^{n-1}$$
$$\frac{\partial \log(x)}{\partial x} = \frac{1}{x}$$
$$\frac{\partial \exp(x)}{\partial x} = \exp(x)$$

Math refresher: derivatives

One can obtain derivatives of composite functions using the following rules

- let a and n be constants, then:

$$\begin{aligned}
\frac{\partial a f(x)}{\partial x} &= a \frac{\partial f(x)}{\partial x} \\
\frac{\partial f(x)^n}{\partial x} &= n f(x)^{n-1} \frac{\partial f(x)}{\partial x} \\
\frac{\partial \exp(f(x))}{\partial x} &= \exp(f(x)) \frac{\partial f(x)}{\partial x} \\
\frac{\partial \log(f(x))}{\partial x} &= \frac{1}{f(x)} \frac{\partial f(x)}{\partial x}
\end{aligned}$$

Math refresher: derivatives

For more complex combinations

$$\frac{\partial g(x) + h(x)}{\partial x} = \frac{\partial g(x)}{\partial x} + \frac{\partial h(x)}{\partial x}$$
$$\frac{\partial g(x)h(x)}{\partial x} = \frac{\partial g(x)}{\partial x}h(x) + g(x)\frac{\partial h(x)}{\partial x}$$
$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}$$

Math refresher: derivatives

- example 1: $f(x) = 3x^4$

$$\frac{\partial f(x)}{\partial x} = \frac{\partial 3x^4}{\partial x} = 3\frac{\partial x^4}{\partial x} = 12x^3$$

- example 2: $f(x) = \exp(\frac{x^2}{3})$

$$\begin{aligned}\frac{\partial f(x)}{\partial x} &= \frac{\partial \exp\left(\frac{x^2}{3}\right)}{\partial x} = \exp\left(\frac{x^2}{3}\right)\frac{\partial \frac{x^2}{3}}{\partial x} \\ &= \frac{1}{3}\exp\left(\frac{x^2}{3}\right)\frac{\partial x^2}{\partial x} = \frac{2}{3}\exp\left(\frac{x^2}{3}\right)x\end{aligned}$$

- example 3: $f(x) = x\exp(x)$

$$\begin{aligned}\frac{\partial f(x)}{\partial x} &= \frac{\partial x}{\partial x}\exp(x) + x\frac{\partial \exp(x)}{\partial x} \\ &= \exp(x) + x\exp(x)\end{aligned}$$

Math refresher: partial derivatives

What happens when the function f is a multi-variate function ?

- The derivative now depends on all the parameters

We then consider the partial derivatives, i.e. the derivatives of the function with respect to one parameter, the others being considered fixed

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta \to 0} \frac{f(x+\Delta, y) - f(x,y)}{\Delta}$$
$$\frac{\partial f(x,y)}{\partial y} = \lim_{\Delta \to 0} \frac{f(x, y+\Delta) - f(x,y)}{\Delta}$$

Math refresher: partial derivatives

- example 1: $f(x, y) = \frac{x^2}{y}$

$$\frac{\partial f(x,y)}{\partial x} = \frac{2x}{y} \qquad \frac{\partial f(x,y)}{\partial y} = \frac{-x^2}{y^2}$$

- example 2: $f(\mathbf{x}) = \frac{\exp(x_2)}{\exp(x_1)+\exp(x_2)+\exp(x_3)}$
  - computing $\frac{\partial f(\mathbf{x})}{\partial x_1}$ is equivalent to compute the derivatives of $h(x) = \frac{a}{\exp(x)+b}$ where a and b are constants

$$\begin{aligned}\frac{\partial h(x)}{\partial x} &= \frac{\partial}{\partial x}\frac{a}{\exp(x)+b} = a\frac{\partial}{\partial x}\frac{1}{\exp(x)+b}\\ &= \frac{-a}{(\exp(x)+b)^2}\frac{\partial}{\partial x}(\exp(x)+b)\\ &= \frac{-a\exp(x)}{(\exp(x)+b)^2}\end{aligned}$$

  - finally $\frac{\partial f(\mathbf{x})}{\partial x_1} = \frac{-\exp(x_2)\exp(x_1)}{(\exp(x_1)+\exp(x_2)+\exp(x_3))^2}$

Math refresher: partial derivatives

Chain derivation

It is always possible to write the derivative of $f(x)$ by using an intermediate function $g(x)$

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial g(x)} \frac{\partial g(x)}{\partial x}$$

If $f(x)$ can be defined by a set of intermediate functions $g_i(x)$, then

$$\frac{\partial f(x)}{\partial x} = \sum_i \frac{\partial f(x)}{\partial g_i(x)} \frac{\partial g_i(x)}{\partial x}$$

This is called the chain rule.

Math refresher: partial derivatives

Chain derivation

- example: $f(x) = 4\exp(x) + 3(1+x)^3$
- let's take $g_1(x) = \exp(x)$ and $g_2(x) = 1 + x$
- such that $f(x) = 4g_1(x) + 3g_2(x)^3$ We have the following partial derivatives

$$\begin{aligned} \frac{\partial f(x)}{\partial g_1(x)} &= 4 & \frac{\partial g_1(x)}{\partial x} &= \exp(x) \\ \frac{\partial f(x)}{\partial g_2(x)} &= 9g_2(x)^2 & \frac{\partial g_2(x)}{\partial x} &= 1 \end{aligned}$$

and finally

$$\frac{\partial f(x)}{\partial x} = 4\exp(x) + 9g_2(x)^2 = 4\exp(x) + 9(1+x)^2$$

Math refresher: partial derivatives

Gradient

We call gradient of a function $f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^n$ the vector $\nabla f(\mathbf{x})$ which components are the partial derivatives of f with respect to the compoent x_i of $\mathbf{x}$

- example: $f(x, y) = \frac{x^2}{y}$

$$\begin{aligned}\nabla f(x, y) &= \left[\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right] \\ &= \left[\frac{2x}{y}, \frac{-x^2}{y^2}\right]\end{aligned}$$

Back to gradient descent

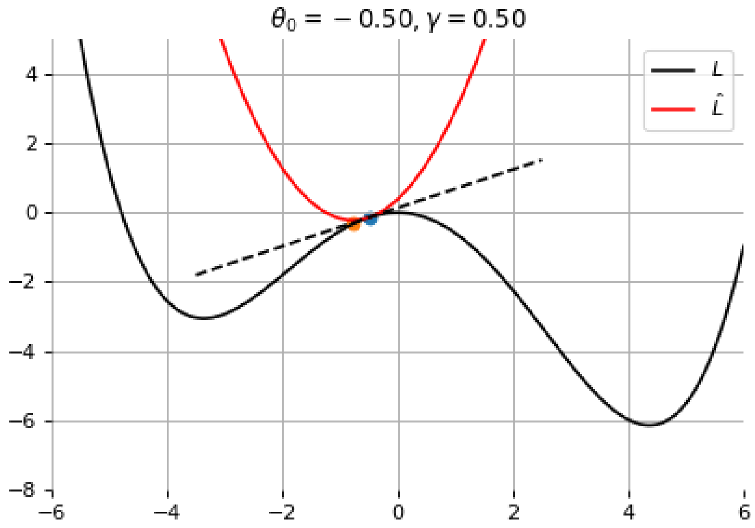
Gradient descent

Let $\mathcal{L}(\theta)$ denote a loss function defined over model parameters θ (e.g., $\mathbf{w}$ and b).

To minimize $\mathcal{L}(\theta)$, gradient descent uses local linear information to iteratively move towards a (local) minimum.

For $\theta_0 \in \mathbb{R}^d$, a first-order approximation around θ_0 can be defined as

$$\hat{\mathcal{L}}(\theta_0 + \epsilon) = \mathcal{L}(\theta_0) + \epsilon^T \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{2\gamma}||\epsilon||^2.$$



A minimizer of the approximation $\hat{\mathcal{L}}(\theta_0 + \epsilon)$ is given for

$$\begin{aligned}\nabla_\epsilon \hat{\mathcal{L}}(\theta_0 + \epsilon) &= 0 \\ &= \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{\gamma}\epsilon,\end{aligned}$$

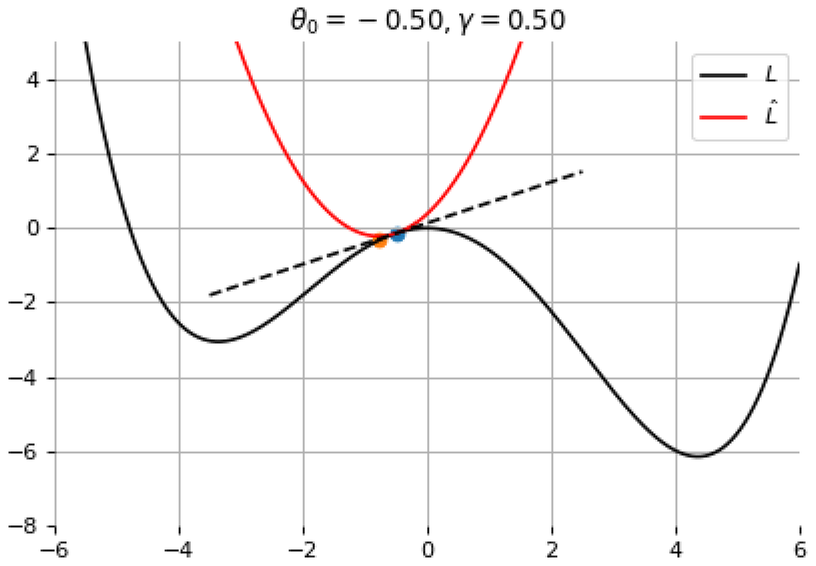
which results in the best improvement for the step $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$.

Therefore, model parameters can be updated iteratively using the update rule

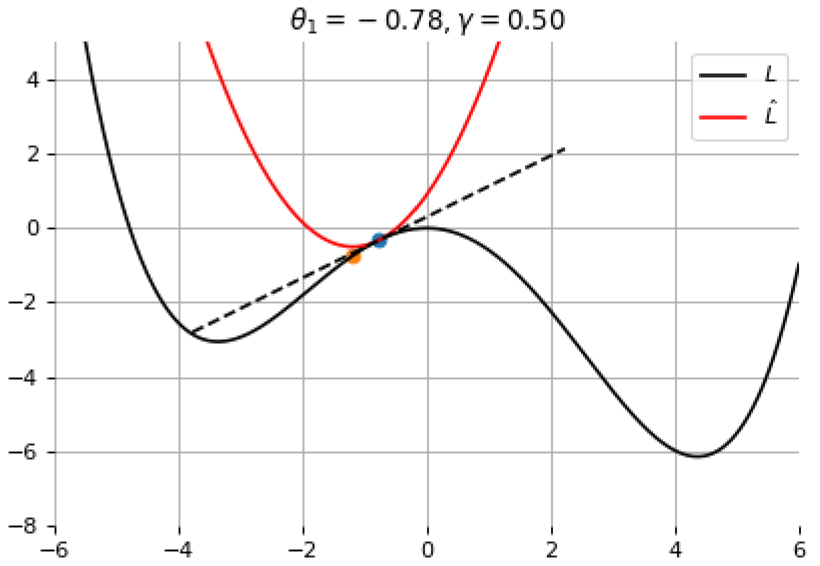
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t),$$

where

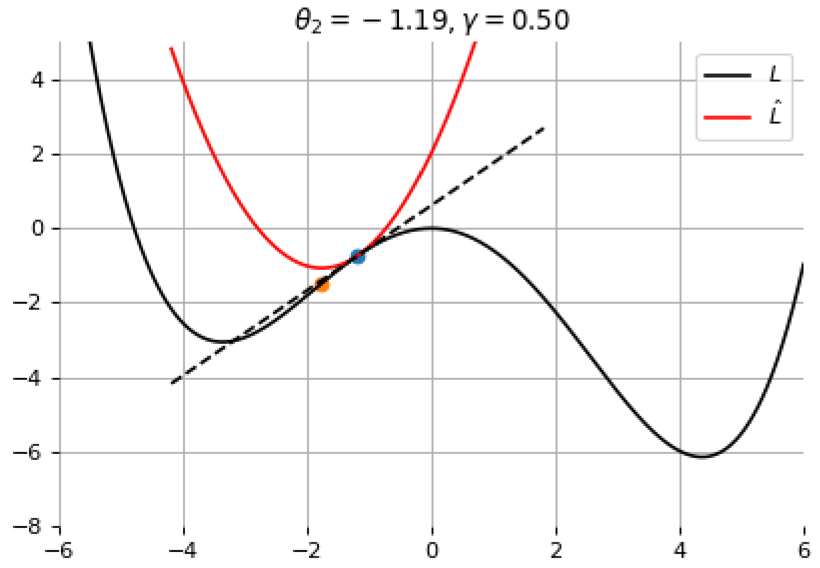
- θ_0 are the initial parameters of the model;
- γ is the learning rate;
- both are critical for the convergence of the update rule.



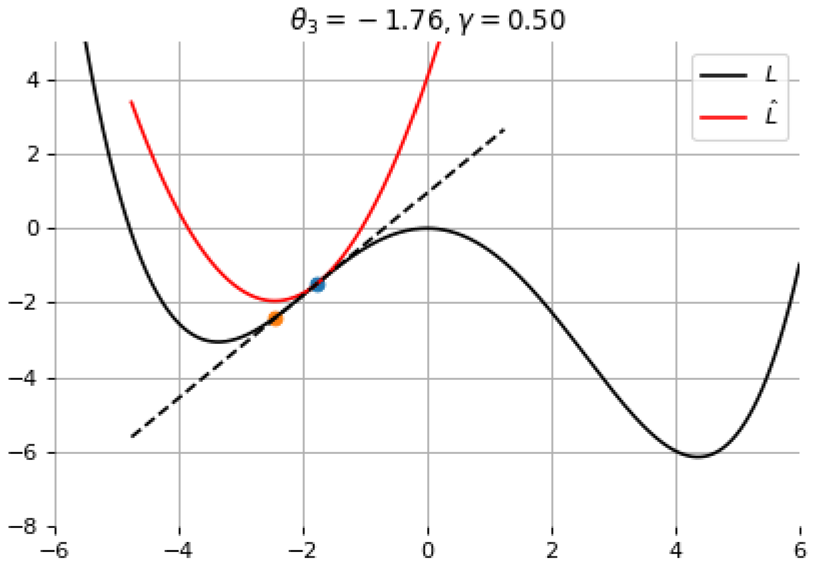
Example 1: Convergence to a local minima



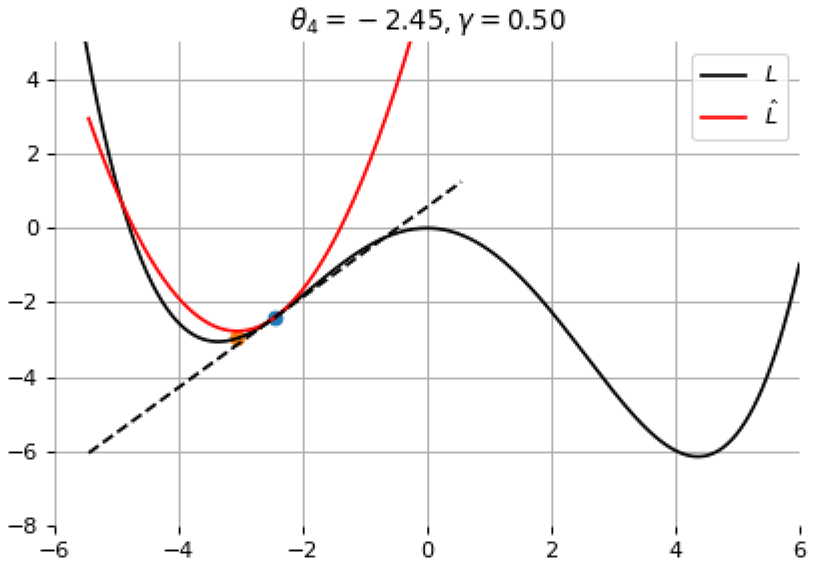
Example 1: Convergence to a local minima



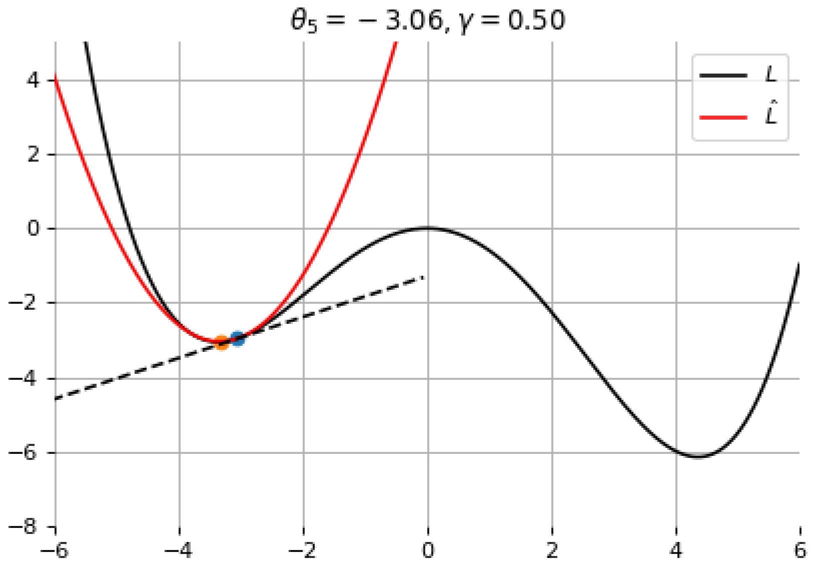
Example 1: Convergence to a local minima



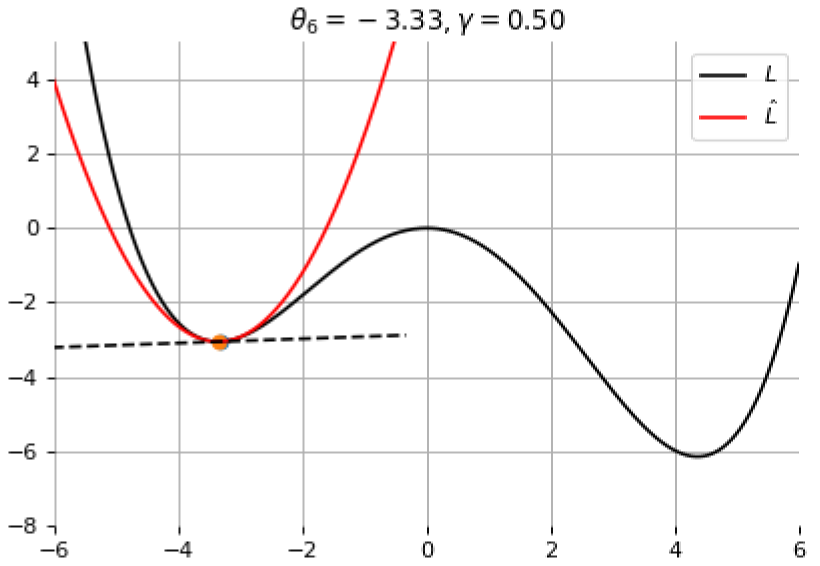
Example 1: Convergence to a local minima



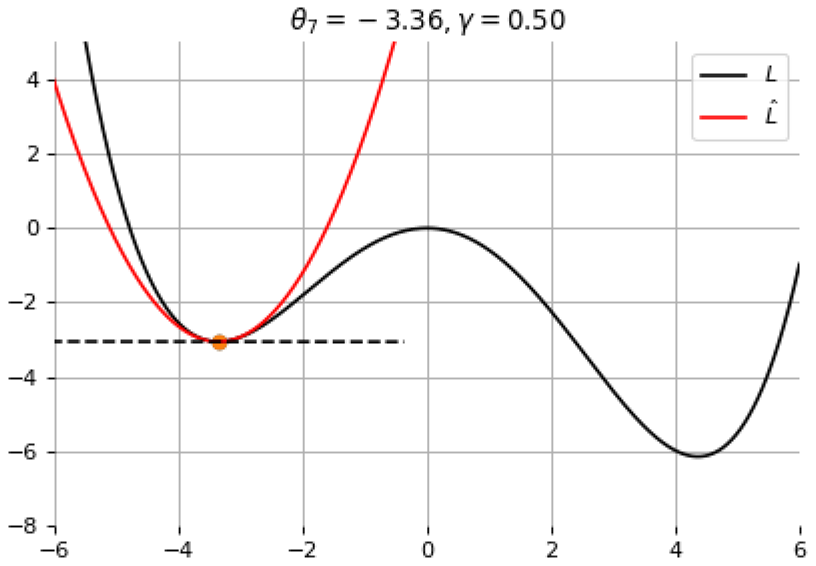
Example 1: Convergence to a local minima



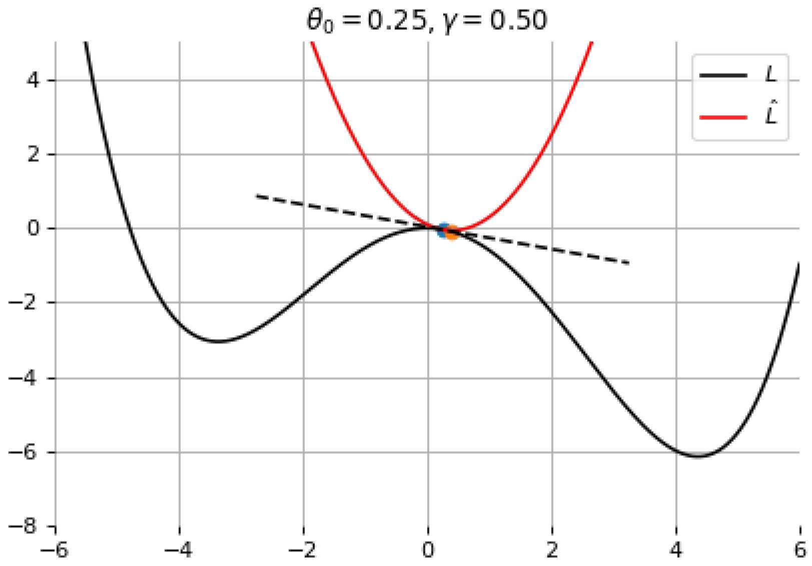
Example 1: Convergence to a local minima



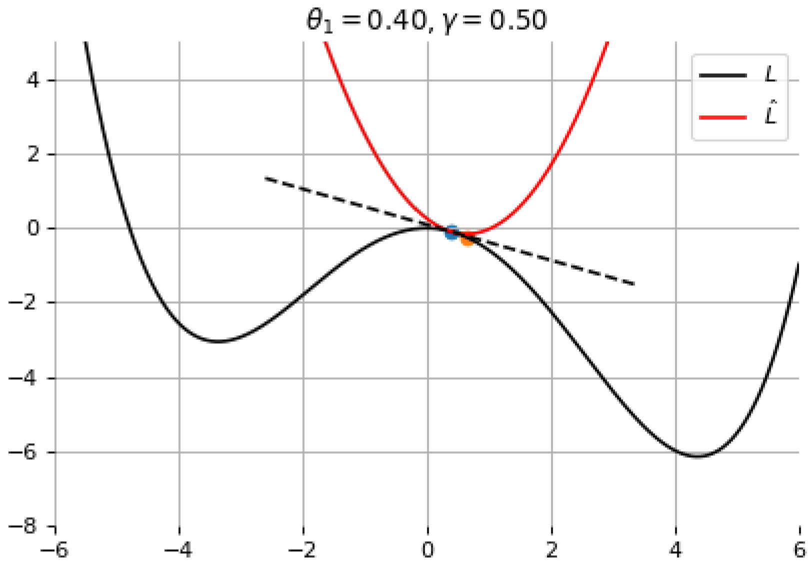
Example 1: Convergence to a local minima



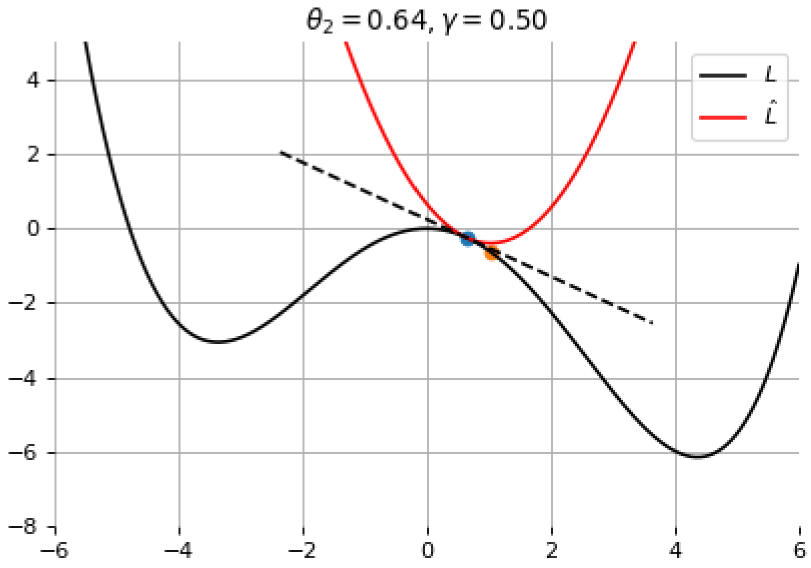
Example 1: Convergence to a local minima



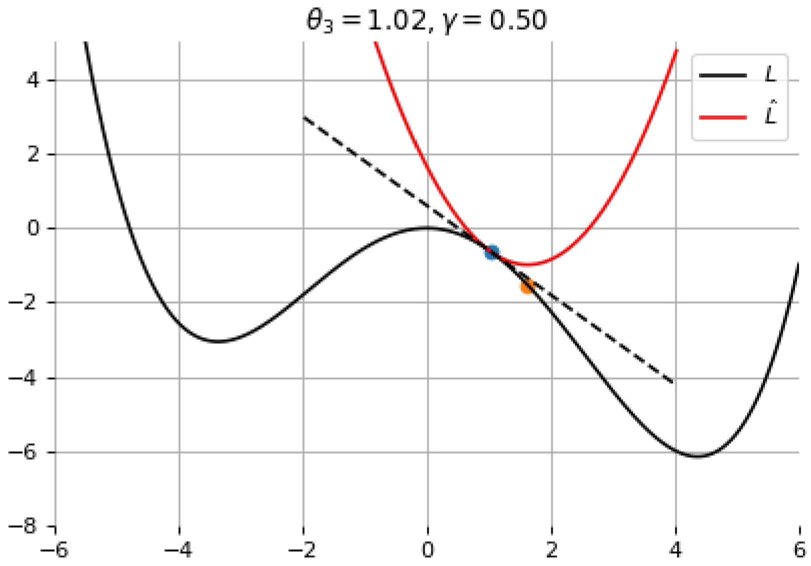
Example 2: Convergence to the global minima



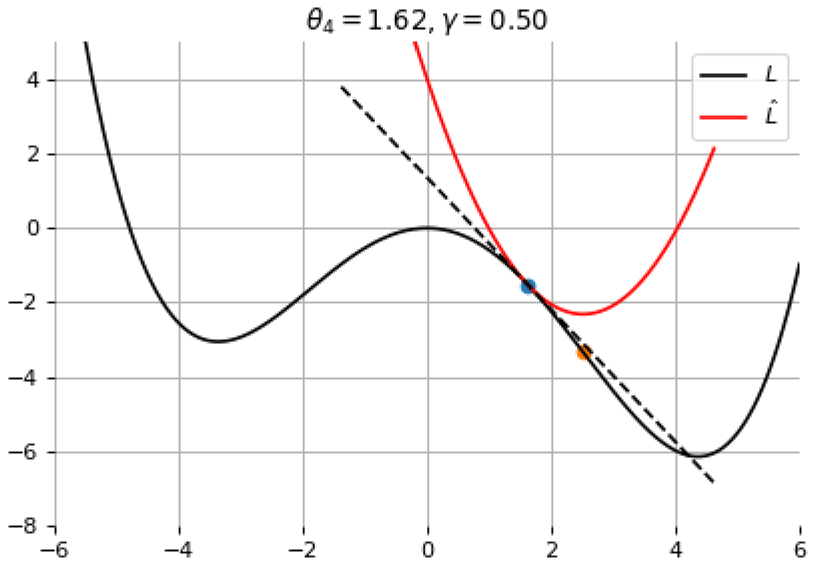
Example 2: Convergence to the global minima



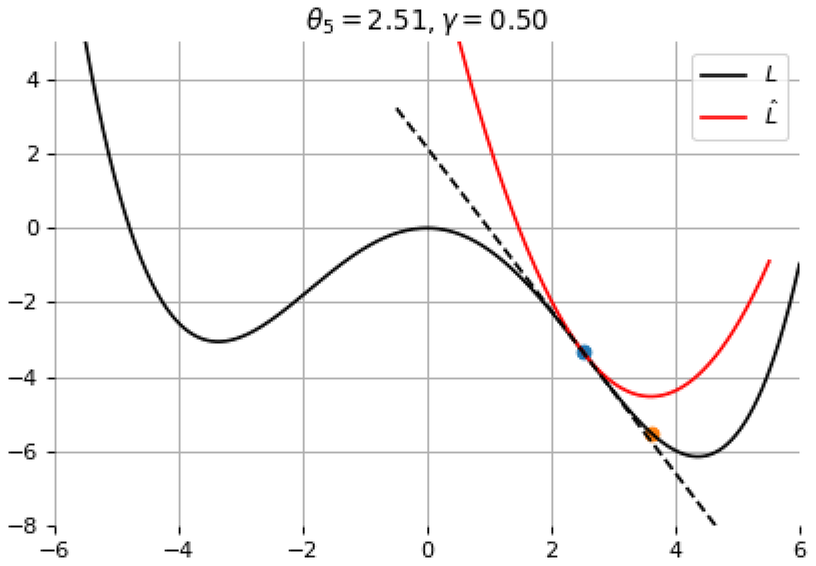
Example 2: Convergence to the global minima



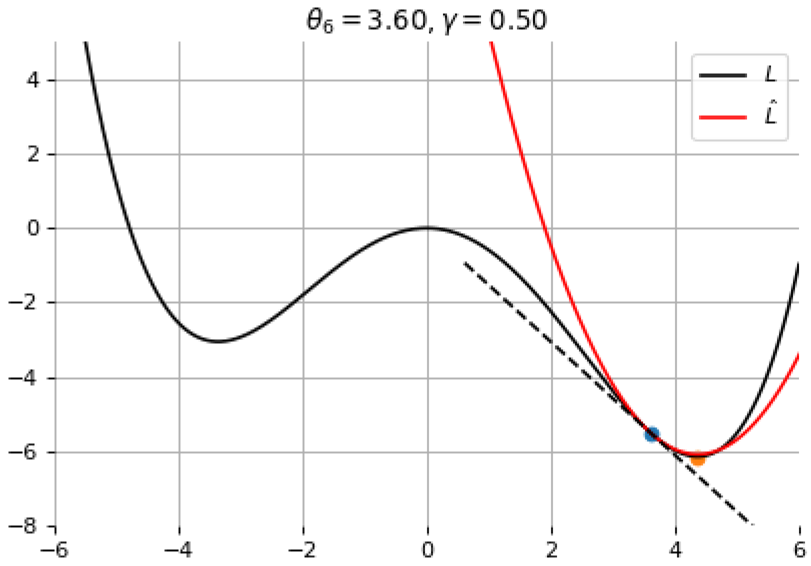
Example 2: Convergence to the global minima



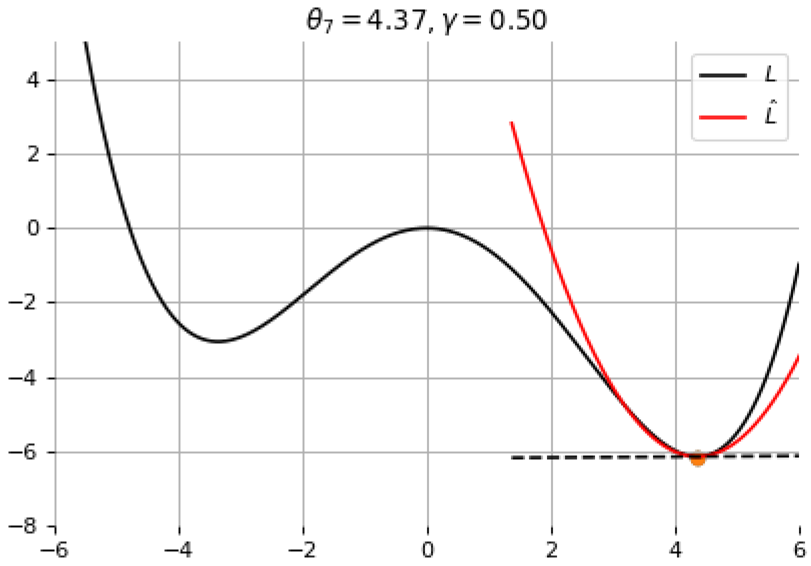
Example 2: Convergence to the global minima



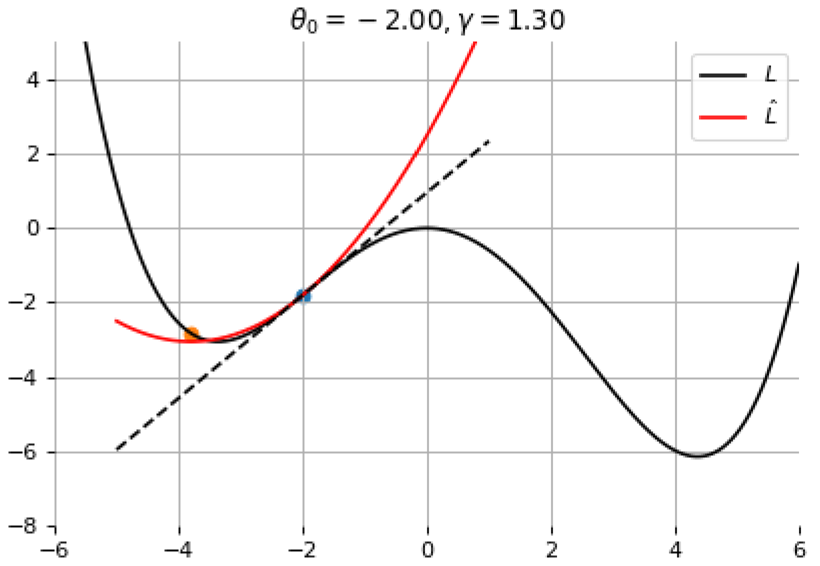
Example 2: Convergence to the global minima



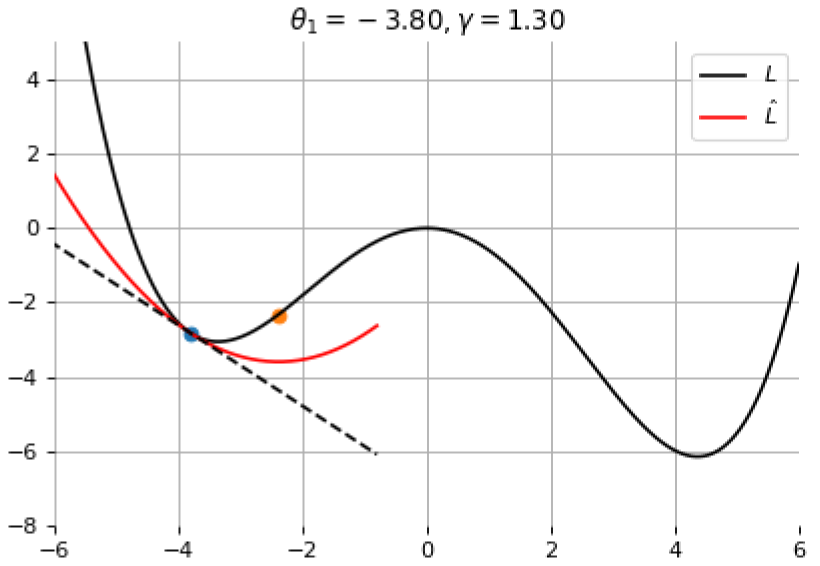
Example 2: Convergence to the global minima



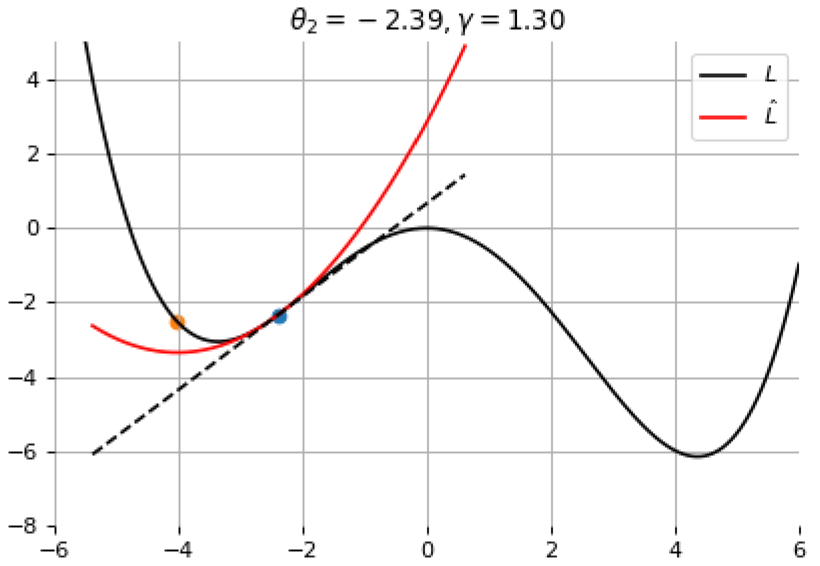
Example 2: Convergence to the global minima



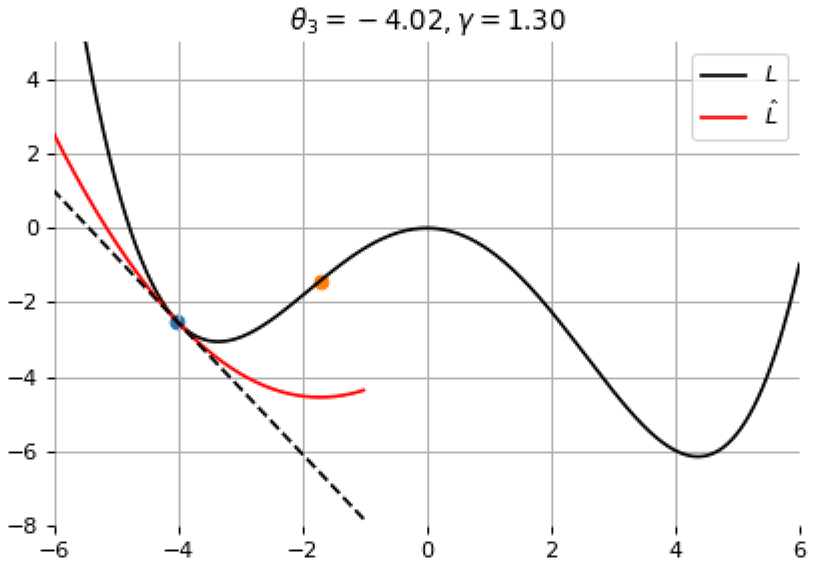
Example 3: Divergence due to a too large learning rate



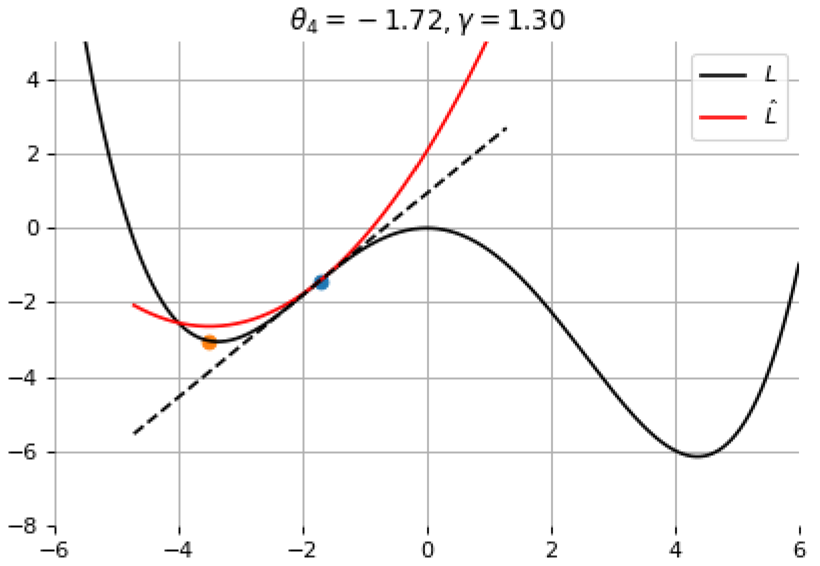
Example 3: Divergence due to a too large learning rate



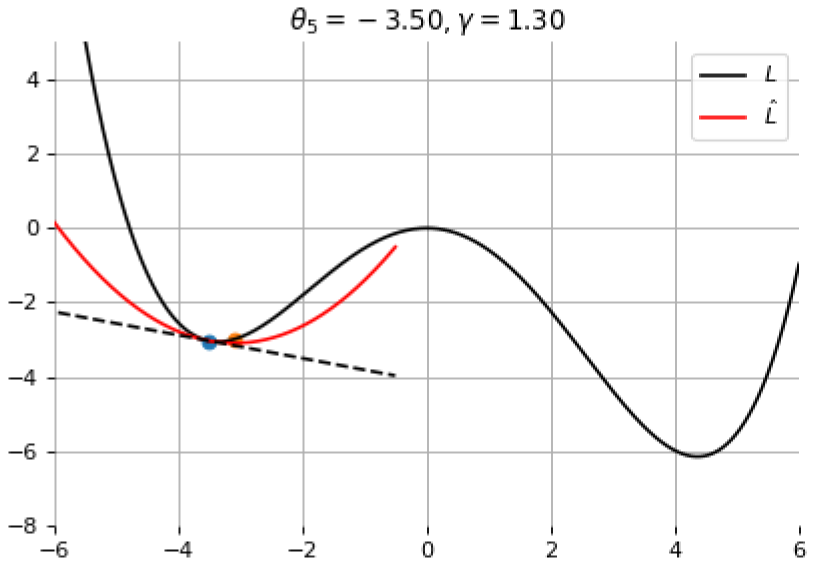
Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate



Example 3: Divergence due to a too large learning rate

Stochastic gradient descent

In the empirical risk minimization setup, $\mathcal{L}(\theta)$ and its gradient decompose as

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \ell(y_i, f(\mathbf{x}_i; \theta))$$
$$\nabla \mathcal{L}(\theta) = \frac{1}{N} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \nabla \ell(y_i, f(\mathbf{x}_i; \theta)).$$

Therefore, in batch gradient descent the complexity of an update grows linearly with the size N of the dataset.

More importantly, since the empirical risk is already an approximation of the expected risk, it should not be necessary to carry out the minimization with great accuracy.

Instead, stochastic gradient descent uses as update rule:

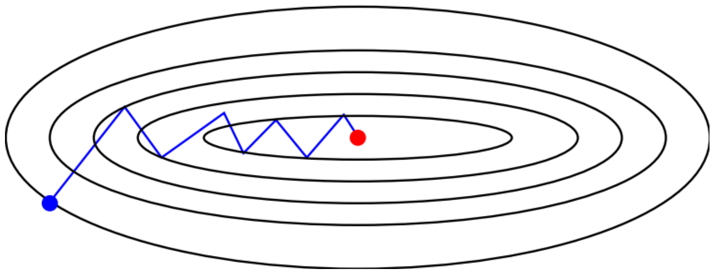
$$\theta_{t+1} = \theta_t - \gamma \nabla \ell(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t))$$

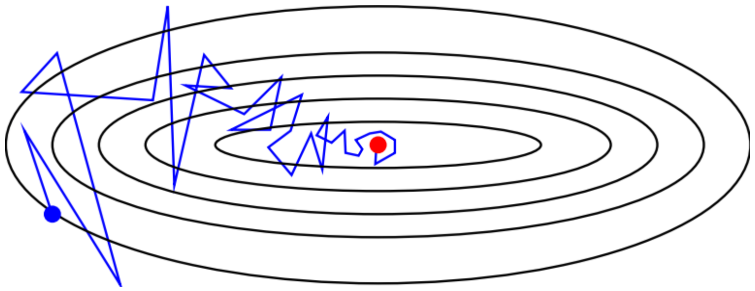
- Iteration complexity is independent of N.
- The stochastic process $\{\theta_t | t = 1, ...\}$ depends on the examples $i(t)$ picked randomly at each iteration.

Instead, stochastic gradient descent uses as update rule:

$$\theta_{t+1} = \theta_t - \gamma \nabla \ell(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t))$$

- Iteration complexity is independent of N.
- The stochastic process $\{\theta_t | t = 1, ...\}$ depends on the examples $i(t)$ picked randomly at each iteration.


Batch gradient descent


Stochastic gradient descent

Why is stochastic gradient descent still a good idea?

- Informally, averaging the update

$$\theta_{t+1} = \theta_t - \gamma \nabla \ell(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t))$$

  over all choices $i(t+1)$ restores batch gradient descent.

- Formally, if the gradient estimate is unbiased, e.g., if

$$\begin{aligned}\mathbb{E}_{i(t+1)}[\nabla \ell(y_{i(t+1)}, f(\mathbf{x}_{i(t+1)}; \theta_t))] &= \frac{1}{N} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \nabla \ell(y_i, f(\mathbf{x}_i; \theta_t)) \\ &= \nabla \mathcal{L}(\theta_t)\end{aligned}$$

  then the formal convergence of SGD can be proved, under appropriate assumptions (see references).

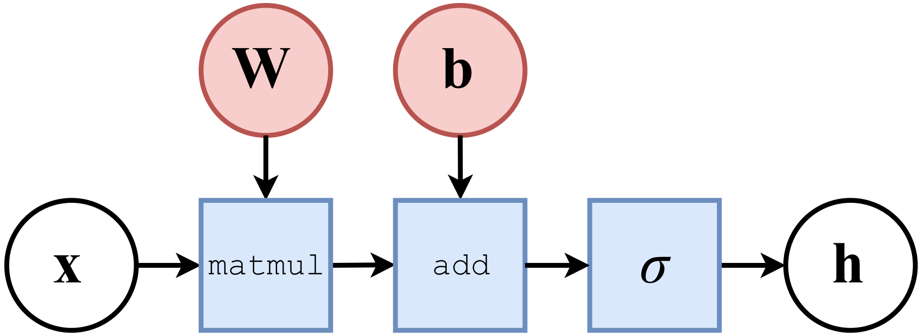
Layers

So far we considered the logistic unit $h = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$, where $h \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{w} \in \mathbb{R}^p$ and $b \in \mathbb{R}$.

These units can be composed in parallel to form a layer with q outputs:

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

where $\mathbf{h} \in \mathbb{R}^q$, $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{W} \in \mathbb{R}^{p\times q}$, $b \in \mathbb{R}^q$ and where $\sigma(\cdot)$ is upgraded to the element-wise sigmoid function.



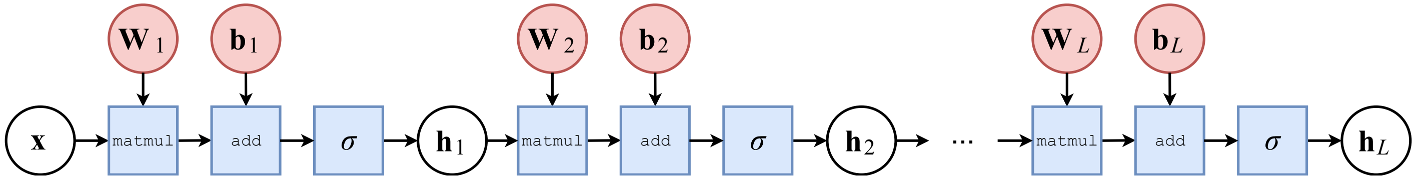
Multi-layer perceptron

Similarly, layers can be composed in series, such that:

$$\begin{aligned}
\mathbf{h}_0 &= \mathbf{x} \\
\mathbf{h}_1 &= \sigma(\mathbf{W}_1^T \mathbf{h}_0 + \mathbf{b}_1) \\
&... \\
\mathbf{h}_L &= \sigma(\mathbf{W}_L^T \mathbf{h}_{L-1} + \mathbf{b}_L) \\
f(\mathbf{x};\theta) = \hat{y} &= \mathbf{h}_L
\end{aligned}$$

where θ denotes the model parameters $\{\mathbf{W}_k, \mathbf{b}_k, ... | k = 1, ..., L\}$.

This model is the multi-layer perceptron, also known as the fully connected feedforward network.


$\mathbf{W}_1$
$\mathbf{b}_1$
$\mathbf{W}_2$
$\mathbf{b}_2$
$\mathbf{W}_L$
$\mathbf{b}_L$
$\mathbf{x}$
matmul
add
σ
$\mathbf{h}_1$
matmul
add
σ
$\mathbf{h}_2$
...
matmul
add
σ
$\mathbf{h}_L$

Classification

- For binary classification, the width q of the last layer L is set to 1, which results in a single output $h_L \in [0,1]$ that models the probability $P(Y=1|\mathbf{x})$.
- For multi-class classification, the sigmoid action σ in the last layer can be generalized to produce a (normalized) vector $\mathbf{h}_L \in [0,1]^C$ of probability estimates $P(Y=i|\mathbf{x})$.

  This activation is the Softmax function, where its i-th output is defined as

$$\text{Softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{C}\exp(z_j)},$$

  for $i = 1, ..., C$.

Regression

The last activation σ can be skipped to produce unbounded output values $h_L \in \mathbb{R}$.

Automatic differentiation

To minimize $\mathcal{L}(\theta)$ with stochastic gradient descent, we need the gradient $\nabla_\theta \ell(\theta_t)$.

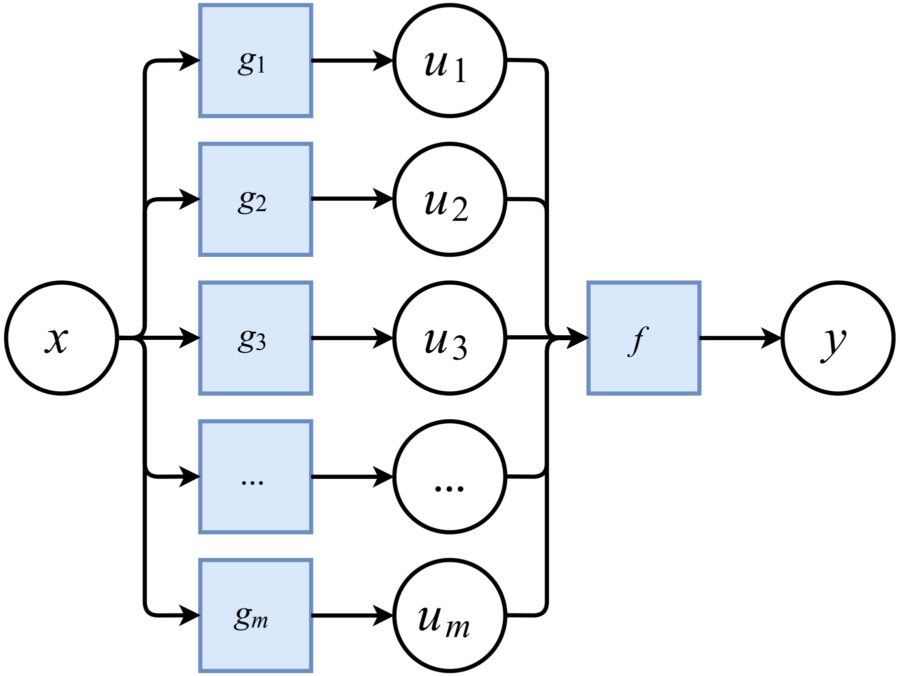
Therefore, we require the evaluation of the (total) derivatives

$$\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{W}_k}, \frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{b}_k}$$

of the loss ℓ with respect to all model parameters $\mathbf{W}_k$, $\mathbf{b}_k$, for $k = 1, ..., L$.

These derivatives can be evaluated automatically from the computational graph of ℓ using automatic differentiation.

Chain rule



Let us consider a 1-dimensional output composition $f \circ g$, such that

$$\begin{aligned} y &= f(\mathbf{u}) \\ \mathbf{u} &= g(x) = (g_1(x), ..., g_m(x)). \end{aligned}$$

The chain rule states that $(f \circ g)' = (f' \circ g)g'$.

For the total derivative, the chain rule generalizes to

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \sum_{k=1}^{m} \frac{\partial y}{\partial u_k} \underbrace{\frac{\mathrm{d}u_k}{\mathrm{d}x}}_{\text{recursive case}}$$

Reverse automatic differentiation

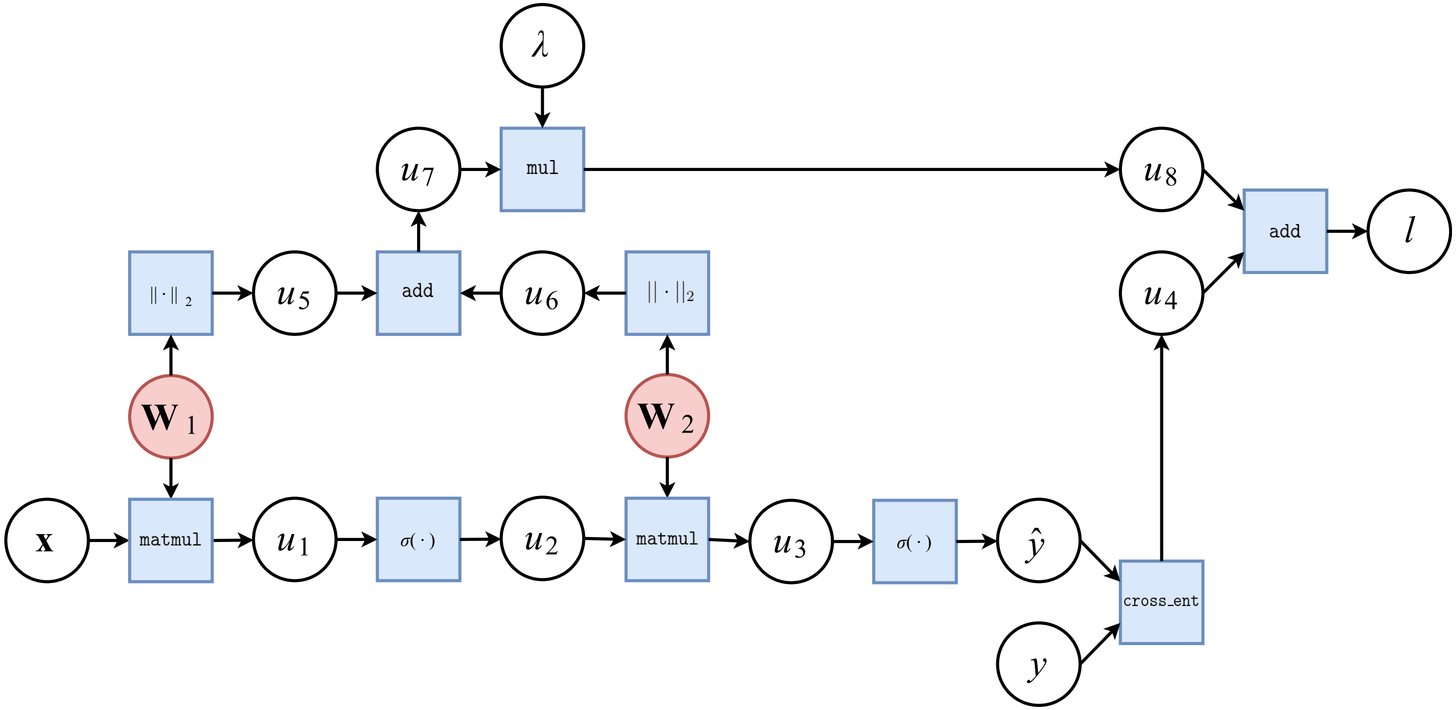
- Since a neural network is a composition of differentiable functions, the total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph.
- The implementation of this procedure is called reverse automatic differentiation.

Let us consider a simplified 2-layer MLP and the following loss function:

$$f(\mathbf{x}; \mathbf{W}_1, \mathbf{W}_2) = \sigma\left(\mathbf{W}_2^T \sigma\left(\mathbf{W}_1^T \mathbf{x}\right)\right)$$
$$\ell(y, \hat{y}; \mathbf{W}_1, \mathbf{W}_2) = \mathrm{cross\_ent}(y, \hat{y}) + \lambda\left(||\mathbf{W}_1||_2 + ||\mathbf{W}_2||_2\right)$$

for $\mathbf{x} \in \mathbb{R}^p$, $y \in \mathbb{R}$, $\mathbf{W}_1 \in \mathbb{R}^{p \times q}$ and $\mathbf{W}_2 \in \mathbb{R}^q$.

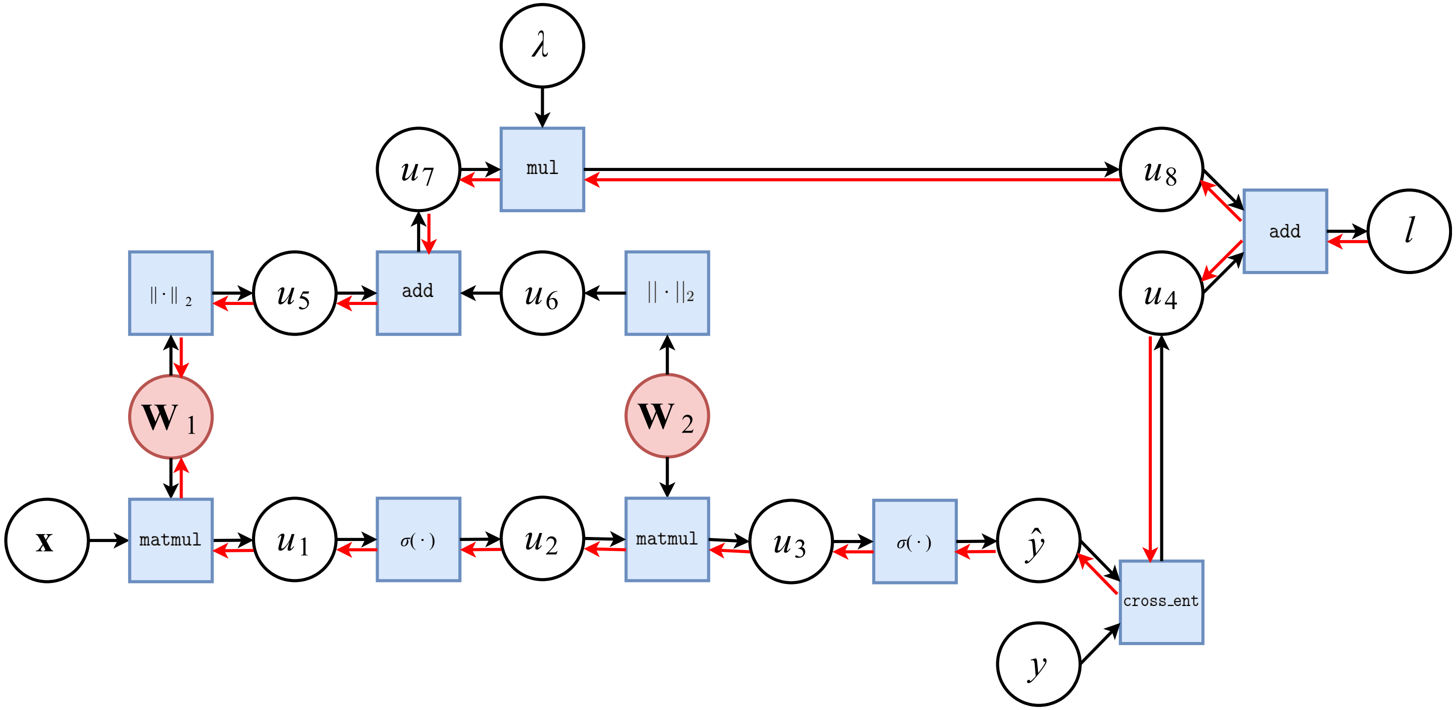
In the forward pass, intermediate values are all computed from inputs to outputs, which results in the annotated computational graph below:

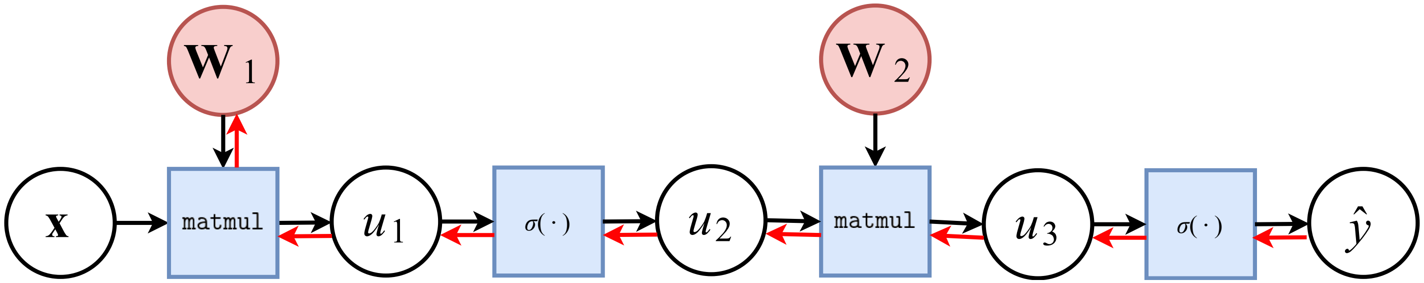


The total derivative can be computed through a backward pass, by walking through all paths from outputs to parameters in the computational graph and accumulating the terms. For example, for $\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{W}_1}$ we have:

$$\frac{\mathrm{d}\ell}{\mathrm{d}\mathbf{W}_1} = \frac{\partial\ell}{\partial u_8}\frac{\mathrm{d}u_8}{\mathrm{d}\mathbf{W}_1} + \frac{\partial\ell}{\partial u_4}\frac{\mathrm{d}u_4}{\mathrm{d}\mathbf{W}_1}$$

$$\frac{\mathrm{d}u_8}{\mathrm{d}\mathbf{W}_1} = \dots$$





Let us zoom in on the computation of the network output $\hat{y}$ and of its derivative with respect to $\mathbf{W}_1$.

- Forward pass: values u_1, u_2, u_3 and $\hat{y}$ are computed by traversing the graph from inputs to outputs given $\mathbf{x}$, $\mathbf{W}_1$ and $\mathbf{W}_2$.
- Backward pass: by the chain rule we have

$$\begin{aligned}\frac{\mathrm{d}\hat{y}}{\mathrm{d}\mathbf{W}_1} &= \frac{\partial \hat{y}}{\partial u_3}\frac{\partial u_3}{\partial u_2}\frac{\partial u_2}{\partial u_1}\frac{\partial u_1}{\partial \mathbf{W}_1} \\ &= \frac{\partial \sigma(u_3)}{\partial u_3}\frac{\partial \mathbf{W}_2^T u_2}{\partial u_2}\frac{\partial \sigma(u_1)}{\partial u_1}\frac{\partial \mathbf{W}_1^T u_1}{\partial \mathbf{W}_1}\end{aligned}$$

Note how evaluating the partial derivatives requires the intermediate values computed forward.

- This algorithm is also known as backpropagation.
- An equivalent procedure can be defined to evaluate the derivatives in forward mode, from inputs to outputs.
- Since differentiation is a linear operator, automatic differentiation can be implemented efficiently in terms of tensor operations.

Deep learning

Recent advances and model architectures in deep learning are built on a natural generalization of a neural network: a graph of tensor operators, taking advantage of

- the chain rule
- stochastic gradient descent
- convolutions
- parallel operations on GPUs.

This does not differ much from networks from the 90s, as covered in Today's lecture.

This generalization allows to compose and design complex networks of operators, possibly dynamically, dealing with images, sound, text, sequences, etc. and to train them end-to-end.

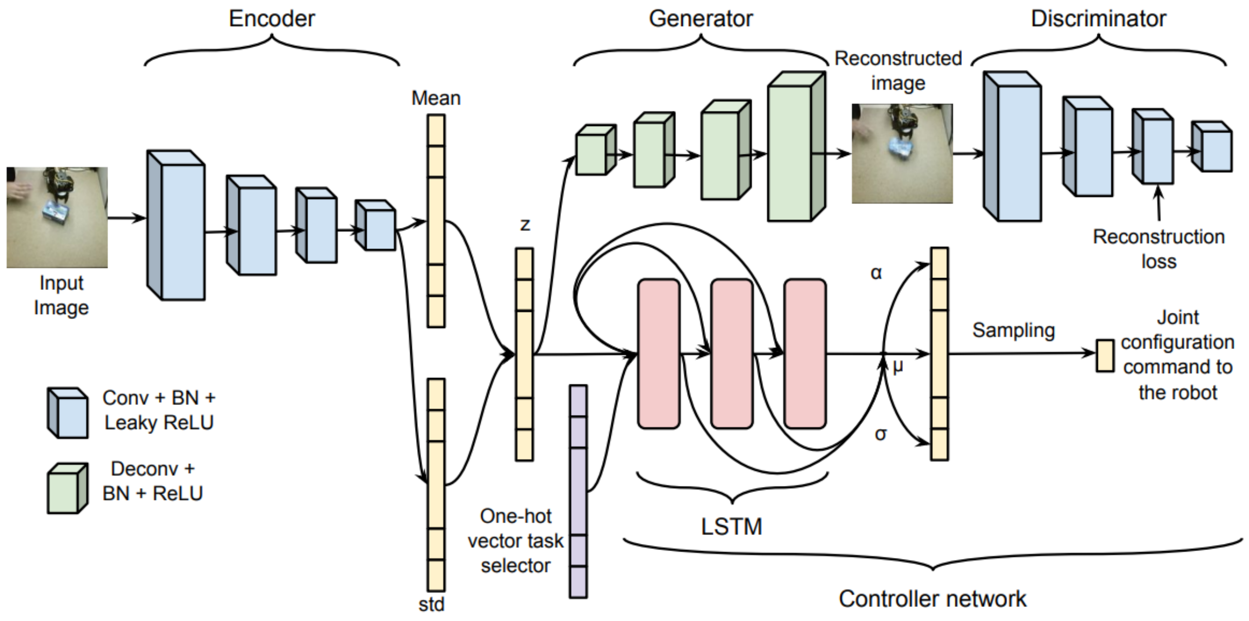


Fig. 2: Our proposed architecture for multi-task robot manipulation learning. The neural network consists of a controller network that outputs joint commands based on a multi-modal autoregressive estimator and a VAE-GAN autoencoder that reconstructs the input image. The encoder is shared between the VAE-GAN autoencoder and the controller network and extracts some shared features that will be used for two tasks (reconstruction and controlling the robot).

The end.

References

- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.
- Bottou, L., & Bousquet, O. (2008). The tradeoffs of large scale learning. In Advances in neural information processing systems (pp. 161-168).
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533.