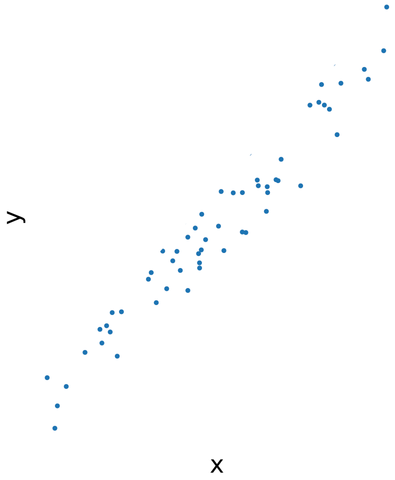
Hands-on Introduction to Deep Learning with PyTorch

Training a simple model

Rafael Sarmiento and Rocco Meli
ETHZürich / CSCS

Lugano, February 28th - March 1st 2024

How to find a y_i given a x_i that's not in the data?



- We select a linear **model** which is appropriate for our dataset and problem

$$y = mx + n$$

$$y = mx + n$$

- We select a linear **model** which is appropriate for our dataset and problem
- How can we use the data to adjust our **model's parameters** (m and n) to minimize prediction error and ensure accuracy within an acceptable margin?

$$y = mx + n$$

$$L = \frac{1}{N}\sum_{i}^{N}(\hat{y}_i - y_i)^2$$

- We select a linear **model** which is appropriate for our dataset and problem
- How can we use the data to adjust our **model's parameters** (m and n) to minimize prediction error and ensure accuracy within an acceptable margin?
- We select a suitable **loss function**
- A loss function quantifies the difference between predicted and actual values, serving as a measure of how well the model performs on a task

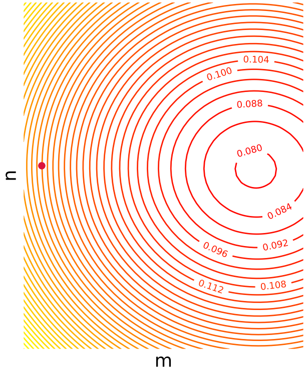
$$y = mx + n$$

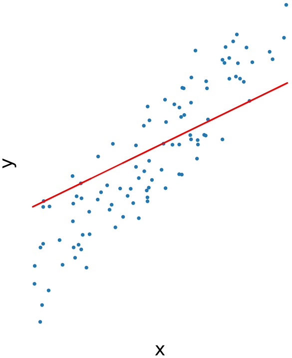
$$L = \frac{1}{N}\sum_{i}^{N}(\hat{y}_i - y_i)^2$$

$$L = \frac{1}{N}\sum_{i}^{N}(mx_i + n - y_i)^2$$

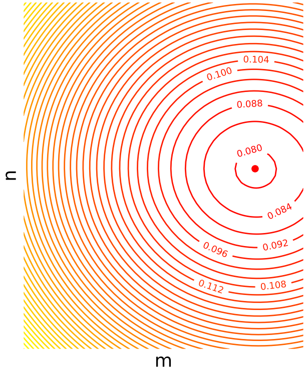
- We select a linear **model** which is appropriate for our dataset and problem
- How can we use the data to adjust our **model's parameters** (m and n) to minimize prediction error and ensure accuracy within an acceptable margin?
- We select a suitable **loss function**
- A loss function quantifies the difference between predicted and actual values, serving as a measure of how well the model performs on a task
- The loss is a function of both the data and the model parameters. Our objective is to minimize the loss function by adjusting the model parameters (**training the model**)

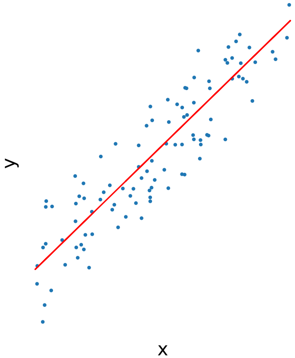
We need to choose an optimizer



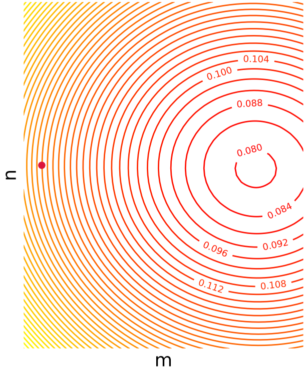


We need to choose an optimizer



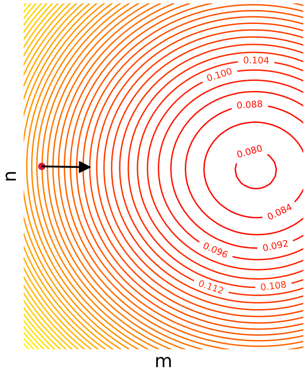


Gradient Descent



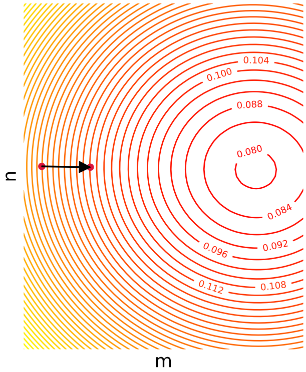
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ in the data $\{x, y\}$ (**forward pass**)

Gradient Descent



- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ in the data $\{x, y\}$ (**forward pass**)
- Compute the gradients of the loss function with respect to the parameters of the model $\frac{\partial L}{\partial W}\Big|_{\{x,y\}}$ (**backpropagation**)

Gradient Descent

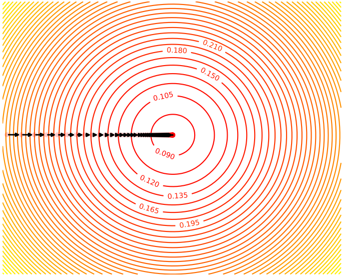


- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ in the data $\{x, y\}$ (**forward pass**)
- Compute the gradients of the loss function with respect to the parameters of the model $\frac{\partial L}{\partial W}\Big|_{\{x,y\}}$ (**backpropagation**)
- Set a **learning rate** (η) and update the model parameters $W_t = W_{t-1} - \eta \frac{\partial L}{\partial W}\Big|_{\{x,y\}_{t-1}}$
- Iterate until the minimization of the loss function converges

Notes on the forward pass and backpropagation

- The computation of the gradients (backpropagation) is done via **Automatic Differentiation**. The chain rule is used to provide exact derivatives of complex functions composed of elementary operations
- During the forward pass, the values of all intermediate operations and its derivatives with respect to the model parameters are stored
- During backpropagation, the gradients of the loss function with respect to the model parameters are computed recursively from the stored values via the chain rule
- PyTorch, TensorFlow, Jax and other deep learning packages provide automatic differentiation

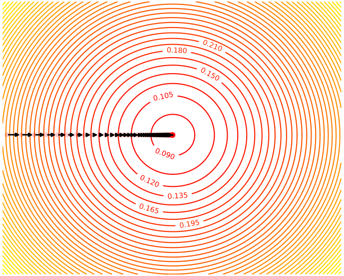
<Stochastic> Gradient Descent



Gradient
Descent

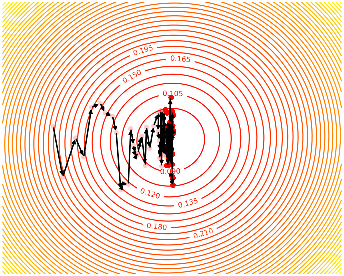
`batch_size = training_set_size`

<Stochastic> Gradient Descent



Gradient
Descent

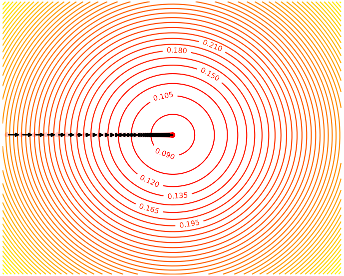
`batch_size = training_set_size`



Stochastic Gradient
Descent

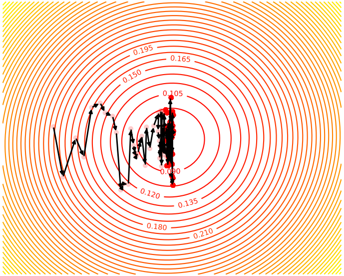
`batch_size = 1`

<Stochastic> Gradient Descent



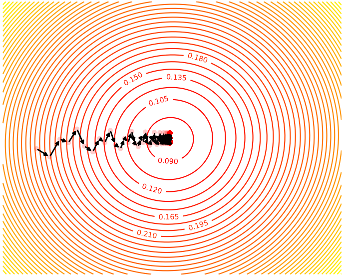
Gradient
Descent

`batch_size = training_set_size`



Stochastic Gradient
Descent

`batch_size = 1`



Minibatch Stochastic Gradient
Descent

`1 < batch_size < training_set_size`

[lab] Simple Stochastic Gradient Descent

Let's run the notebook `sgd/linear_model_sgd.ipynb`. There we use an unidimensional linear model to understand the trajectories of the SGD.

Let's try different batch sizes and learning rates to see how the SGD trajectory changes.

Summary and Remarks

- Loss function surfaces exhibit high complexity with numerous local minima. Model training primarily focuses on locating suitable local minima rather than striving for the global minimum
- Here we saw an example of **regression** which together with **classification** are the main types of problems in **supervised learning**
- **Unsupervised learning** and **reinforcement learning** are other important categories in machine learning problems
- For simplicity we didn't do any **testing** or **validation** of our model, but that's an important part of training models

Thank you for your attention!

ETH zürich
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre