EPFL

Introduction to Pytorch

IPEO Exercise 6

Devis TUIA
Thien-Anh Nguyen
Manon Béchaz
Giacomo May
EPFL-ECEO

October 18, 2024

École
polytechnique
fédérale de
Lausanne

Welcome to Pytorch Exercises 6-8

In the next four exercises, we will focus on
Deep Learning with Pytorch

IPEO exercise 6	Introduction to Pytorch with Logistic Regression
IPEO exercise 7	Image Classification with CNNs
IPEO exercise 8	Model Training and Regularization
IPEO exercise 9	Semantic Segmentation

Welcome to Pytorch Exercises 6-8

In the next four exercises, we will focus on
Deep Learning with Pytorch

IPEO exercise 6	Introduction to Pytorch with Logistic Regression
IPEO exercise 7	Image Classification with CNNs
IPEO exercise 8	Model Training and Regularization
IPEO exercise 9	Semantic Segmentation

Welcome to Pytorch Exercises 6-8

In the next four exercises, we will focus on
Deep Learning with Pytorch

IPEO exercise 6	Introduction to Pytorch with Logistic Regression
IPEO exercise 7	Image Classification with CNNs
IPEO exercise 8	Model Training and Regularization
IPEO exercise 9	Semantic Segmentation

Welcome to Pytorch Exercises 6-8

In the next four exercises, we will focus on
Deep Learning with Pytorch

IPEO exercise 6	Introduction to Pytorch with Logistic Regression
IPEO exercise 7	Image Classification with CNNs
IPEO exercise 8	Model Training and Regularization
IPEO exercise 9	Semantic Segmentation

Pytorch is Numpy with gradients

NumPy

- package for numerical operations in Python
- first released 1995
- basis for several classical machine learning packages, like scikit-learn

PyTorch

- package for deep learning in Python
- first released 2016
- basis for deep learning and de-facto standard library for scientific applications
- **extends numpy with automatic differentiation and gradients**
- **allows working with GPUs**

Pytorch is Numpy with gradients

NumPy

- package for numerical operations in Python
- first released 1995
- basis for several classical machine learning packages, like scikit-learn

PyTorch

- package for deep learning in Python
- first released 2016
- basis for deep learning and de-facto standard library for scientific applications
- **extends numpy with automatic differentiation and gradients**
- **allows working with GPUs**

Pytorch is Numpy with gradients

NumPy

- package for numerical operations in Python
- first released 1995
- basis for several classical machine learning packages, like scikit-learn

PyTorch

- package for deep learning in Python
- first released 2016
- basis for deep learning and de-facto standard library for scientific applications
- extends numpy with automatic differentiation and gradients
- allows working with GPUs

Pytorch is Numpy with gradients

NumPy

- package for numerical operations in Python
- first released 1995
- basis for several classical machine learning packages, like scikit-learn

PyTorch

- package for deep learning in Python
- first released 2016
- basis for deep learning and de-facto standard library for scientific applications
- **extends numpy with automatic differentiation and gradients**
- allows working with GPUs

Pytorch is Numpy with gradients

NumPy

- package for numerical operations in Python
- first released 1995
- basis for several classical machine learning packages, like scikit-learn

PyTorch

- package for deep learning in Python
- first released 2016
- basis for deep learning and de-facto standard library for scientific applications
- **extends numpy with automatic differentiation and gradients**
- **allows working with GPUs**

Automatic Differentiation

Image **I** of N gray pixels i_i

$$\mathbf{I} = [i_1, i_2, i_3, \ldots, i_N]$$

```
from torch import tensor
I = tensor([[1],[0.8]...],
    requires_grad=True)
```

Automatic Differentiation

Image **I** of N gray pixels i_i

$$\mathbf{I} = [i_1, i_2, i_3, \ldots, i_N]$$

some function $\mathbf{I} \rightarrow s$, e.g., summation

$$s = \sum_{i=1}^{N} i_i$$

```
from torch import tensor
I = tensor([[1],[0.8]...],
    requires_grad=True)
```

summation function

```
s = I.sum()
```

Automatic Differentiation

Image $\mathbf{I}$ of N gray pixels i_i

$$\mathbf{I} = [i_1, i_2, i_3, \dots, i_N]$$

some function $\mathbf{I} \rightarrow s$, e.g., summation

$$s = \sum_{i=1}^{N} i_i$$

analytic gradient calculation
"effect of one pixel on the sum"

$$\frac{\partial s}{\partial i_1} = \sum_{i=1}^{N} i_i \frac{\partial}{\partial i_1} = 1$$

```
from torch import tensor
I = tensor([[1],[0.8]...],
    requires_grad=True)
```

summation function

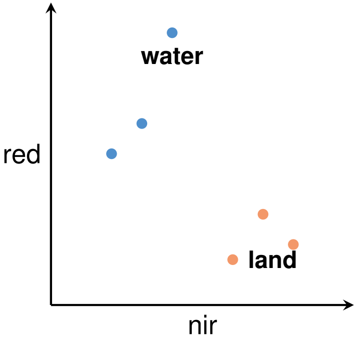
```
s = I.sum()
```

automatic differentiation

```
s.backward() # backprop
# display grad
I.grad[0]
> 1.
```

Logistic Regression

Objective: map an input pixel $\mathbf{x} = (x_{\text{green}}, x_{\text{red}}, x_{\text{nir}}) \rightarrow y$ probability of "water" [0,1]

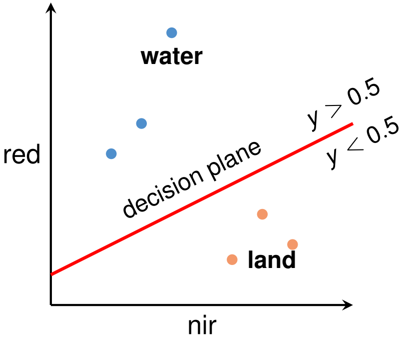


Logistic Regression

Objective: map an input pixel $\mathbf{x} = (x_{\text{green}}, x_{\text{red}}, x_{\text{nir}}) \rightarrow y$ probability of "water" [0,1]

logistic regression model $f_{\mathbf{A},b}(\mathbf{x})$

$$y = f_{\mathbf{A},b}(\mathbf{x}) = \sigma(\underbrace{\mathbf{A}^{\mathsf{T}}\mathbf{x} + b}_{\text{decision plane}})$$



Logistic Regression

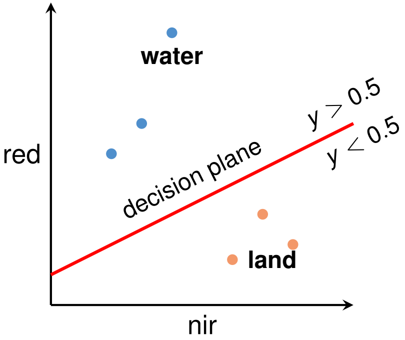
Objective: map an input pixel $\mathbf{x} = (x_{\text{green}}, x_{\text{red}}, x_{\text{nir}}) \to y$ probability of "water" [0,1]

logistic regression model $f_{\mathbf{A},b}(\mathbf{x})$

$$y = f_{\mathbf{A},b}(\mathbf{x}) = \sigma(\underbrace{\mathbf{A}^\mathsf{T}\mathbf{x} + b}_{\text{decision plane}})$$

with parameters slope $\mathbf{A} = (a_{\text{green}}, a_{\text{red}}, a_{\text{nir}})$ and bias b of the decision plane

and a "squishing" sigmoid function $\sigma \mapsto [0, 1]$



Logistic Regression

How do we find a good decision plane $\mathbf{A}, b$?

We solve the optimization objective

$$\mathbf{A}, b = \arg_{\mathbf{A},b} \min \sum_{i=1}^{M} \mathcal{L}(f_{\mathbf{A},b}(\mathbf{x}_i), y_\mathrm{i})$$

in words:

- we find parameters $\mathbf{A}, b$
- that minimize ($\arg\min$) the error $\sum_{i=1}^{M} \mathcal{L}(f_{\mathbf{A},b}(\mathbf{x}_i), y_\mathrm{i})$ of
- model output $f_{\mathbf{A},b}(\mathbf{x}_i)$
- with ground truth y_i over a training dataset of M examples.

we implement $\arg\min$ with gradient descent in PyTorch

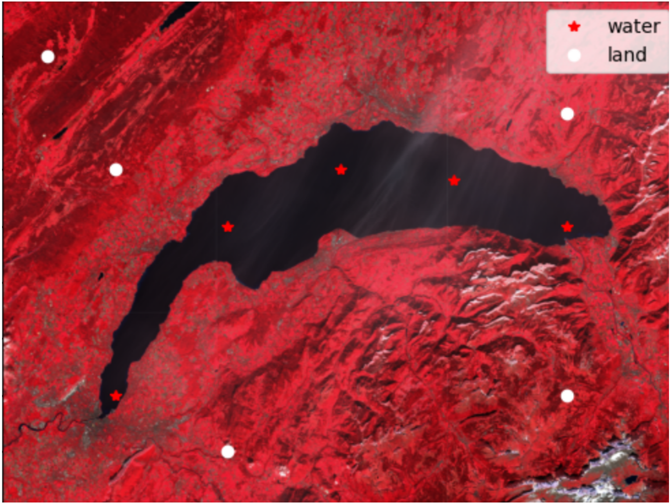
Gradient descent

Idea, we start with random **A**, b and update parameters iteratively

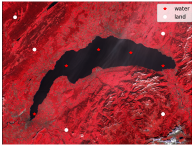
$$\mathbf{A} \leftarrow \mathbf{A} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{A}}$$
$$b \leftarrow b - \eta \frac{\partial \mathcal{L}}{\partial b}.$$

- we can think of the gradient $\frac{\partial \mathcal{L}}{\partial b}$ in words: "what **change** in b **changes** the loss $\mathcal{L}$".
- the learning rate η defines the step size and is an important hyper-parameter.

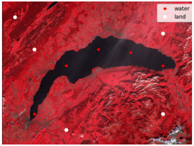
Water Classification

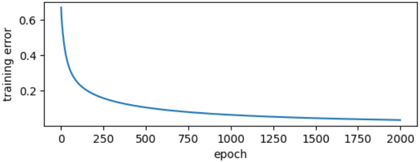


Land-water classification



Land-water classification

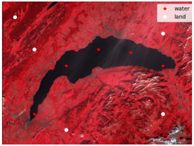


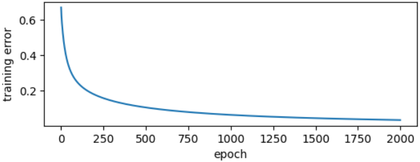


training data

optimize decision plane parameters

Land-water classification



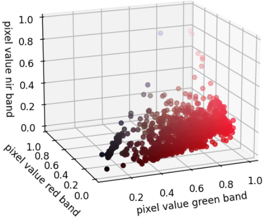


training data

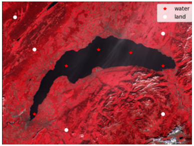
optimize decision plane parameters

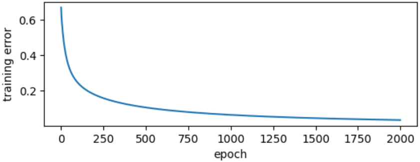
all pixels

feature space: green - red - nir



Land-water classification

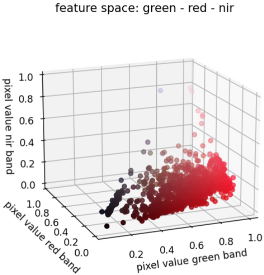


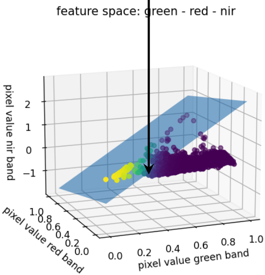


training data

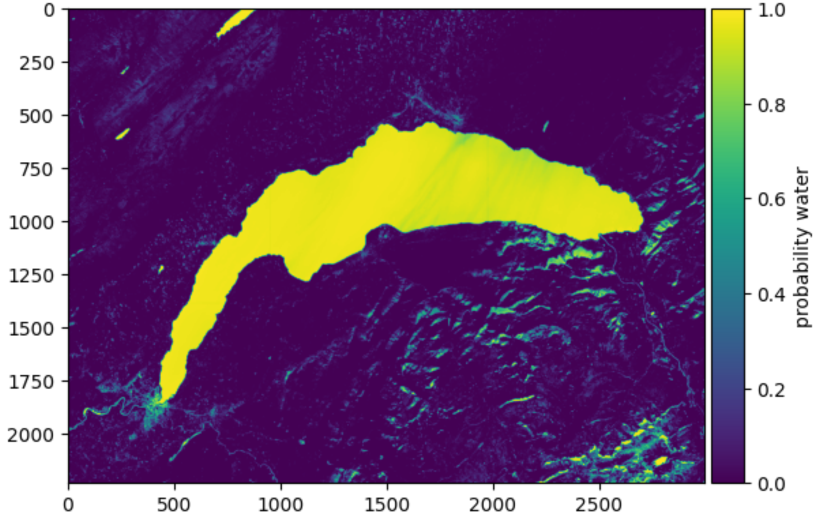
all pixels

optimize decision plane parameters





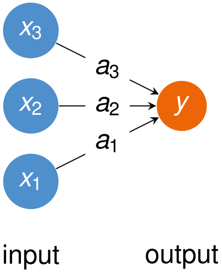
Expected Result



Outlook: Why logistic regression?

Logistic regression corresponds to a single linear layer

$$\sigma(\mathbf{Ax} + \mathbf{b})$$



Multi-layer perceptron (neural net with dense layers)

$$\sigma(\mathbf{A}_1\mathbf{x} + \mathbf{b}_1) \circ \sigma(\mathbf{A}_2\mathbf{x} + \mathbf{b}_2)$$