Introduction to Deep Learning (CS474)

Lecture 6

Rupam

Outline

- Mechanics of Learning-PART I
    - Introduction
    - Example
        - Application of Linear Model.

Introduction

- We can argue that **learning from data** presumes the underlying model is not engineered to solve a specific problem and is instead capable of **approximating** a much wider family of functions.

- We're interested in models that are not engineered for solving a specific narrow task, but that can be automatically adapted to specialize themselves for <u>any one of many similar tasks</u> using input and output pairs.

- **PyTorch** is designed to make it easy to create models for which the derivatives of the fitting error, with respect to the parameters, can be expressed analytically.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Introduction: Overview of Learning Process

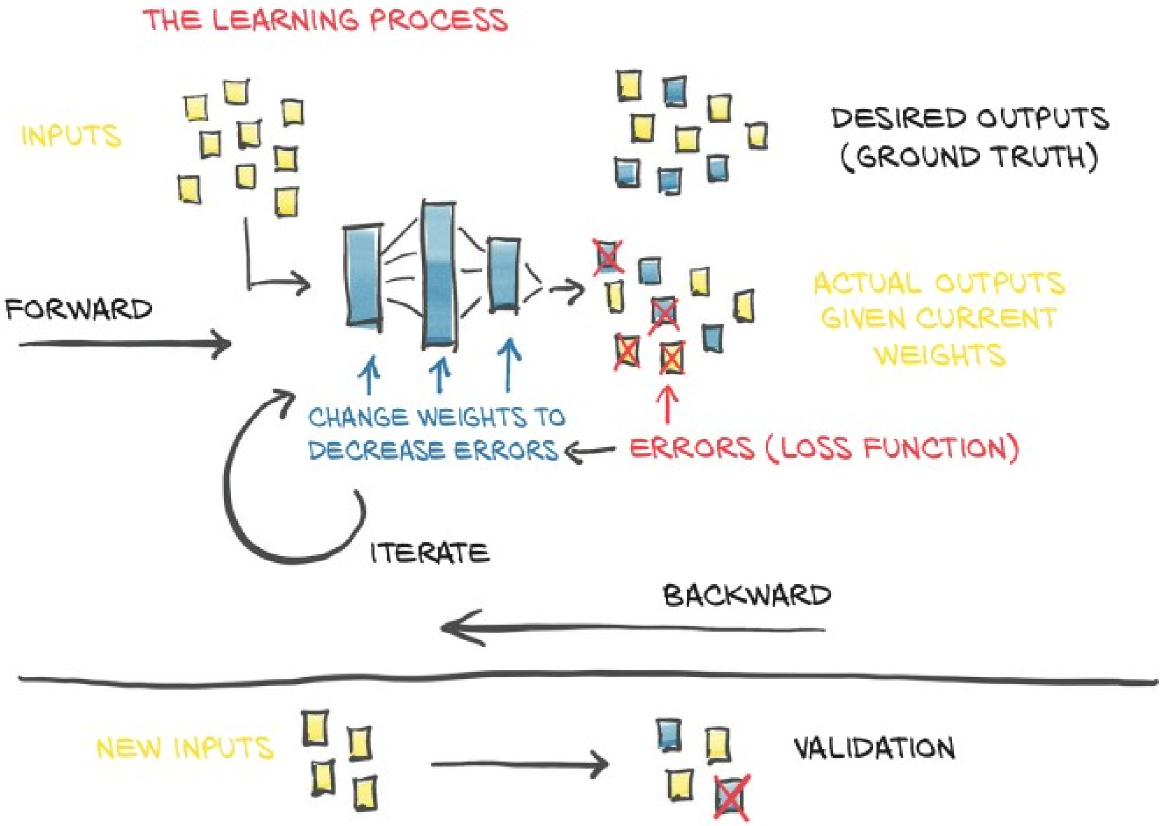


Image credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Example

```python
import numpy as np
import torch

# t_c values are temperatures in Celsius
# t_u values are our unknown units

t_c = [0.5,  14.0, 15.0, 28.0, 11.0,  8.0,  3.0, -4.0,  6.0, 13.0, 21.0]
t_u = [35.7, 55.9, 58.2, 81.9, 56.3, 48.9, 33.9, 21.8, 48.4, 60.4, 68.4]

# For convenience, we've already put the data into tensors.

t_c = torch.tensor(t_c)
t_u = torch.tensor(t_u)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

- The ***two*** may be ***linearly*** related—that is, multiplying **`t_u`** by a factor and adding a constant, we may get the temperature in Celsius (up to an error that we omit):

  `t_c = w * t_u + b`

- We'll see how well the final model performs.

- We chose to name **`w`** and **`b`** after **weight** and **bias**, two very common terms for *linear scaling* and the *additive constant.*

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

- We have a **model** with some unknown parameters.

- We need to **estimate** those parameters so that the error between predicted outputs and measured values is as low as possible.

- We notice that we still need to exactly define a measure of the error. Such a measure, which we refer to as the loss function, should be high if the error is high and should ideally be as low as possible for a perfect match.

- Our **optimization process** should therefore aim at finding **`w`** and **`b`** so that the **loss function** is at a minimum.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

- Continuing with the earlier *notebook*.

```
# Our model!
def model(t_u, w, b):
    return w * t_u + b

# Our loss function!
def loss_fn(t_p, t_c):
    squared_diffs = (t_p - t_c)**2
    return squared_diffs.mean()
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

- Note that we are building a **tensor** of differences, taking their square **element-wise**, and finally producing a scalar loss function by averaging all of the elements in the resulting tensor. It is a mean square loss.

- Continuing with the earlier *notebook*.

```
# Initializing parameters!
w = torch.ones(())
b = torch.zeros(())

t_p = model(t_u, w, b)
t_p
```

```
tensor([35.7000, 55.9000, 58.2000, 81.9000, 56.3000, 48.9000, 33.9000, 21.8000,
        48.4000, 60.4000, 68.4000])
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

- Now, we can print the loss as well!

- Continuing with the earlier *notebook*.

```
# Loss calculation!

loss = loss_fn(t_p, t_c)
loss
```

```
tensor(1763.8846)
```

- How do we estimate **w** and **b** such that the loss reaches a **minimum**?

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

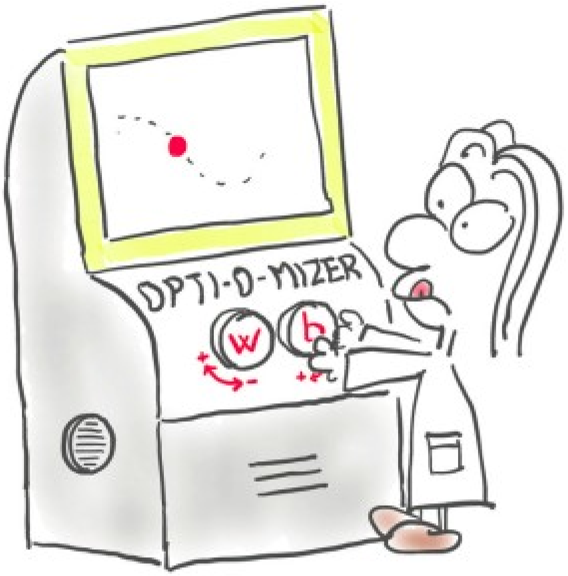


Figure 1 A cartoon depiction of the optimization process, where a person with knobs for w and b searches for the direction to turn the knobs that makes the loss decrease

Image credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

Decreasing Loss:

- **Gradient descent** is not that different from the scenario we just described.

- The idea is to compute the rate of change of the loss with respect to each parameter, and modify each parameter in the direction of decreasing loss.

```
delta = 0.1

loss_rate_of_change_w = \
    (loss_fn(model(t_u, w + delta, b), t_c) -
     loss_fn(model(t_u, w - delta, b), t_c)) / (2.0 * delta)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Example: Application of Linear Model

- We typically should scale the rate of change by a small factor.

- This scaling factor has many names; the one we use in machine learning is learning_rate.

```
learning_rate = 1e-2

print(learning_rate)

w = w - learning_rate * loss_rate_of_change_w
```

```
0.01
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

- This represents the basic **parameter-update step** for **gradient descent.**

```
# Same thing can be done for b

loss_rate_of_change_b = \
    (loss_fn(model(t_u, w, b + delta), t_c) -
     loss_fn(model(t_u, w, b - delta), t_c)) / (2.0 * delta)

b = b - learning_rate * loss_rate_of_change_b
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

```python
#Computing the derivative

def dloss_fn(t_p, t_c):
    dsq_diffs = 2 * (t_p - t_c) / t_p.size(0)
    return dsq_diffs

# APPLYING THE DERIVATIVES TO THE MODEL

def dmodel_dw(t_u, w, b):
    return t_u

def dmodel_db(t_u, w, b):
    return 1.0

# DEFINING THE GRADIENT FUNCTION

def grad_fn(t_u, t_c, t_p, w, b):
    dloss_dtp = dloss_fn(t_p, t_c)
    dloss_dw = dloss_dtp * dmodel_dw(t_u, w, b)
    dloss_db = dloss_dtp * dmodel_db(t_u, w, b)
    return torch.stack([dloss_dw.sum(), dloss_db.sum()])
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

- We now have everything in place to optimize our parameters.
- Starting from a tentative value for a parameter, we can **iteratively** apply updates to it for a fixed number of iterations, or until w and b stop changing.
- There are several stopping criteria; for now, we'll stick to a fixed number of iterations.
- We call a training iteration during which we update the parameters for all of our training samples an **epoch**.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model

```
# Training Loop

def training_loop(n_epochs, learning_rate, params, t_u, t_c):
    for epoch in range(1, n_epochs + 1):
        w, b = params

        t_p = model(t_u, w, b)  # <1>
        loss = loss_fn(t_p, t_c)
        grad = grad_fn(t_u, t_c, t_p, w, b)  # <2>

        params = params - learning_rate * grad

        print('Epoch %d, Loss %f' % (epoch, float(loss))) # <3>

    return params
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

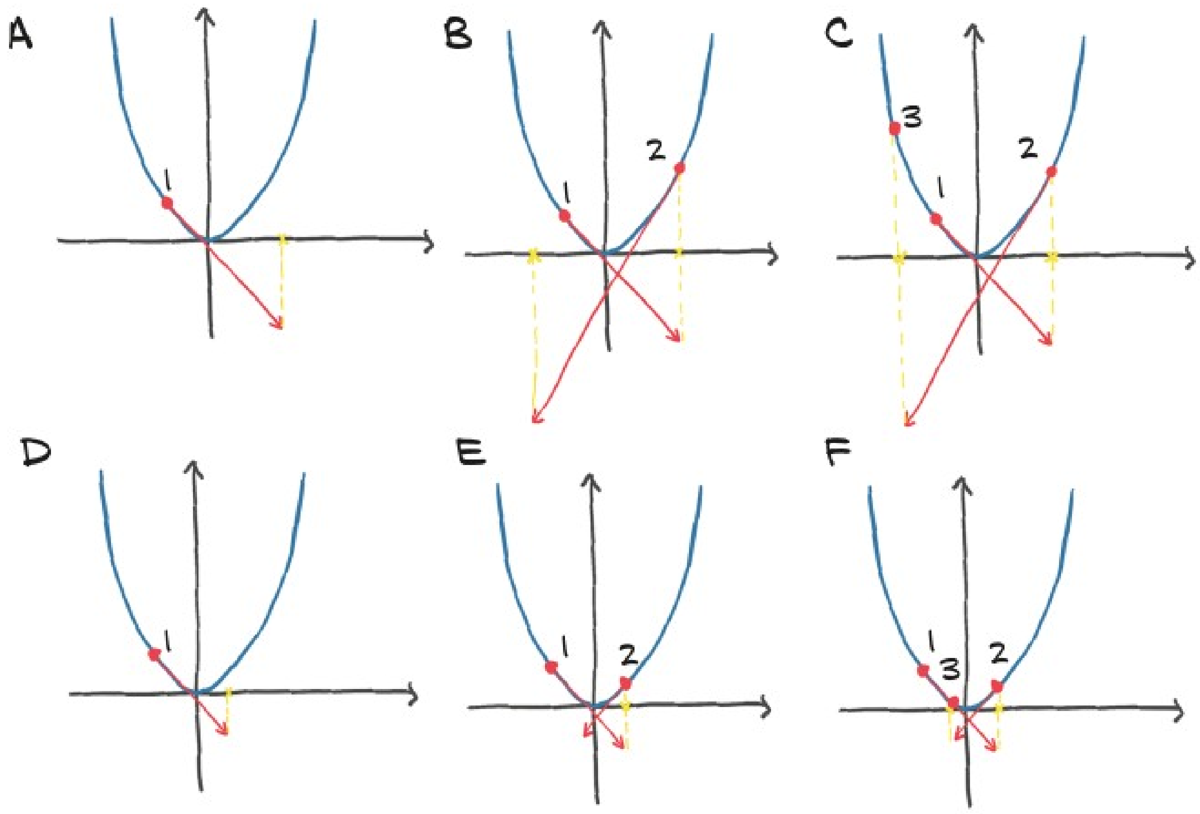
Example: Application of Linear Model

```
# Invoking Training Loop

training_loop(
n_epochs = 100,
learning_rate = 1e-2,
params = torch.tensor([1.0, 0.0]),
t_u = t_u,
t_c = t_c)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example: Application of Linear Model



Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

References

- All the contents present in the slides are taken from various online resources. Due credit is given in the respective slides.These slides are used for *academic* purposes only.