Introduction to Deep Learning (CS474)

Lecture 12

Rupam

Outline

- Mechanics of Learning-PART VI
  - PyTorch `nn` module

Rupam

Introduction

- In the last lecture, we have seen PyTorch nn module.
- There's one last step left to take: replacing our linear model with a neural network as our approximating function.
- We said earlier that using a neural network will not result in a higher-quality model, since the process underlying our calibration problem was fundamentally linear.
- However, it's good to make the leap from linear to neural network in a controlled environment so we won't feel lost later.

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

Rupam

Replacing Linear Model

- We are going to keep everything else fixed, including the loss function, and only redefine model .

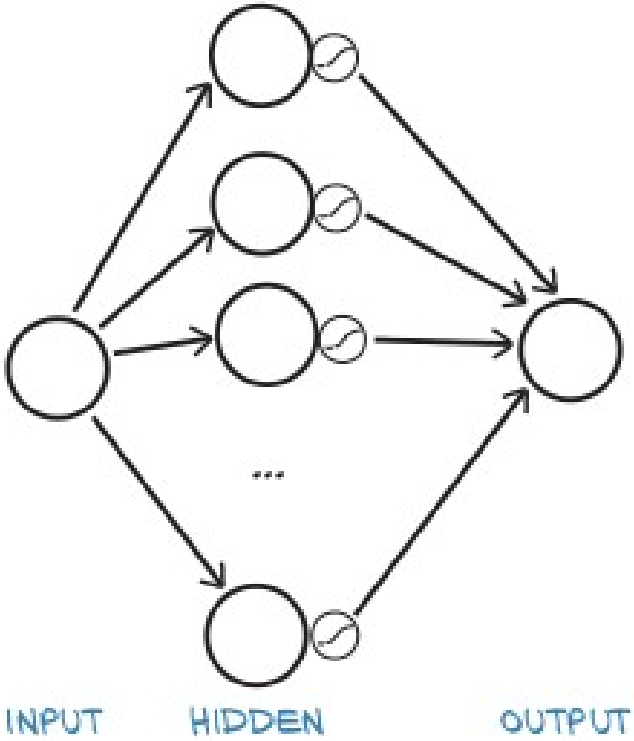
- Let's build the simplest possible neural network: a linear module, followed by an activation function, feeding into another linear module.

- the input and output of the model are both of size 1 (they have one input and one output feature).

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

Rupam

Replacing Linear Model



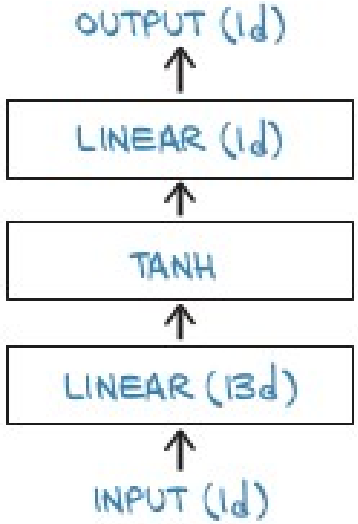


Figure 1

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

Rupam

Replacing Linear Model

- There is no standard way to depict neural networks.

- Figure 1 shows two ways that seem to be somewhat prototypical: the left side shows how our network might be depicted in basic introductions, whereas a style similar to that on the right is often used in the more advanced literature.

- It is common to make diagram blocks that roughly correspond to the neural network modules PyTorch offers (though sometimes things like the Tanh activation layer are not explicitly shown).

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

Rupam

Example

```
import numpy as np
import torch
import torch.optim as optim
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]
t_c = torch.tensor(t_c).unsqueeze(1)
t_u = torch.tensor(t_u).unsqueeze(1)
t_u.shape

n_samples = t_u.shape[0]
n_val = int(0.2 * n_samples)

shuffled_indices = torch.randperm(n_samples)

train_indices = shuffled_indices[:-n_val]
val_indices = shuffled_indices[-n_val:]

train_indices, val_indices

t_u_train = t_u[train_indices]
t_c_train = t_c[train_indices]

t_u_val = t_u[val_indices]
t_c_val = t_c[val_indices]

t_un_train = 0.1 * t_u_train
t_un_val = 0.1 * t_u_val
```

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

Rupam

Example

Continuing with the earlier *notebook*:

```
import torch.nn as nn

seq_model = nn.Sequential(
            nn.Linear(1, 13), # <1>
            nn.Tanh(),
            nn.Linear(13, 1)) # <2>
seq_model
```

```
Sequential(
  (0): Linear(in_features=1, out_features=13, bias=True)
  (1): Tanh()
  (2): Linear(in_features=13, out_features=1, bias=True)
)
```

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

Rupam

Example

- **nn** provides a simple way to concatenate modules through the **nn.Sequential** container.

- The end result is a model that takes the inputs expected by the first module specified as an argument of nn.Sequential, passes intermediate outputs to subsequent modules, and produces the output returned by the last module.

- The model fans out from 1 input feature to 13 hidden features, passes them through a tanh activation, and linearly combines the resulting 13 numbers into 1 output feature.

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

Rupam

Example

- Calling `model.parameters()` will collect weight and bias from both the first and second linear modules.

- It's instructive to inspect the parameters in this case by printing their shapes.

```
[param.shape for param in seq_model.parameters()]
```

```
[torch.Size([13, 1]), torch.Size([13]), torch.Size([1, 13]), torch.Size([1])]
```

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

Rupam

Example

- These are the tensors that the optimizer will get.
- Again, after we call `model.backward()`, all parameters are populated with their `grad`, and the optimizer then updates their values accordingly during the `optimizer.step()` call.
- A few notes on parameters of nn.Modules .
- When inspecting parameters of a model made up of several submodules, it is handy to be able to identify parameters by name.
- There's a method for that, called `named_parameters`.

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

Rupam

Example

- Continuing with the earlier *notebook*:

```
for name, param in seq_model.named_parameters():
    print(name, param.shape)
```

```
0.weight torch.Size([13, 1])
0.bias torch.Size([13])
2.weight torch.Size([1, 13])
2.bias torch.Size([1])
```

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

Rupam

Example

- Continuing with the earlier *notebook*:

```
from collections import OrderedDict

seq_model = nn.Sequential(OrderedDict([
    ('hidden_linear', nn.Linear(1, 8)),
    ('hidden_activation', nn.Tanh()),
    ('output_linear', nn.Linear(8, 1))
]))

seq_model
```

```
Sequential(
  (hidden_linear): Linear(in_features=1, out_features=8, bias=True)
  (hidden_activation): Tanh()
  (output_linear): Linear(in_features=8, out_features=1, bias=True)
)
```

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

Rupam

Example

- Continuing with the earlier *notebook*:

```
for name, param in seq_model.named_parameters():
    print(name, param.shape)
```

```
hidden_linear.weight torch.Size([8, 1])
hidden_linear.bias torch.Size([8])
output_linear.weight torch.Size([1, 8])
output_linear.bias torch.Size([1])
```

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

Rupam

Example

- Continuing with the earlier *notebook*:
- You need to write the function for the training loop! (It is similar to our last lecture note)

```
optimizer = optim.SGD(seq_model.parameters(), lr=1e-3)

training_loop(
    n_epochs = 5000,
    optimizer = optimizer,
    model = seq_model,
    loss_fn = nn.MSELoss(),
    t_u_train = t_un_train,
    t_u_val = t_un_val,
    t_c_train = t_c_train,
    t_c_val = t_c_val)
```

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

Rupam

Example

- Continuing with the earlier *notebook*:

```
print('output', seq_model(t_un_val))
print('answer', t_c_val)
print('hidden', seq_model.hidden_linear.weight.grad)
```

```
Epoch 1, Training loss 104.2794, Validation loss 427.5994
Epoch 1000, Training loss 5.4566, Validation loss 51.7407
Epoch 2000, Training loss 3.0486, Validation loss 26.1902
Epoch 3000, Training loss 2.1646, Validation loss 16.8295
Epoch 4000, Training loss 1.7993, Validation loss 12.2945
Epoch 5000, Training loss 1.6467, Validation loss 10.0024
output tensor([[24.4679],
        [15.7434]], grad_fn=<AddmmBackward>)
answer tensor([[28.],
        [13.]])
hidden tensor([[ 0.0018],
        [ 0.0263],
        [-0.0152],
        [-0.0020],
        [-0.0525],
        [ 0.0154],
        [ 0.0313],
        [ 0.0193]])
```

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.