Quiz 3

December 5, 2025

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import jax
import jax.numpy as jnp
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.utils.data
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
import pickle
```

[8]:
```
import os
os.getcwd()
```

[8]:
```
'/Users/jakegraham'
```

[19]:
```
# 1 (quadratic regression)

ts, ys = pickle.load(open('/Users/jakegraham/Downloads/
 ↪quadratic_regression_data.pkl', 'rb'))

def f(theta,x):
    a,b,c = theta
    return a*x**2 + b*x + c

def initial_parameters():
    return jnp.array([0., 0., 0.])

def mse(theta,x,y):
    return jnp.mean((f(theta,x)-x)**2)

def sgd_step(theta, x, y, lr):
    loss, grads = jax.value_and_grad(mse)(theta, x, y)
    theta = theta - lr*grads
    return theta, loss
```

```
theta = initial_parameters()
n_epochs, eta = 10000, 1e-2

for epoch in range(n_epochs):
    theta, loss = sgd_step(theta, ts, ys, eta)

print(f"predicted theta is a = {theta[0]}, b = {theta[1]}, c = {theta[2]}")
```

```
predicted theta is a = 4.791501851286739e-05, b = 0.9998172521591187, c =
0.0001303233002545312
```

[28]:
```
# 2 (supervised learning with NN)

ts_torch = torch.tensor(ts, dtype=torch.float32).reshape(-1, 1)
ys_torch = torch.tensor(ys, dtype=torch.float32).reshape(-1, 1)
N, n_epochs = 25, 10000
epochs_to_plot = [0, 100, 500, 1000, 5000, 9999]


class MLRegression(nn.Module):
    def __init__(self, N):
        super().__init__()

        self.fc1 = nn.Linear(in_features=1, out_features=N)
        self.fc2 = nn.Linear(in_features=N, out_features=1)

    def forward(self, t):
        t = self.fc1(t)
        t = F.relu(t)
        t = self.fc2(t)

model = MLRegression(N)
optimizer = torch.optim.SGD(model.parameters(), lr=1e-2)
criterion = nn.MSELoss()

for epoch in range(n_epochs):
    model.train()
    pred = model(ts_torch)
    loss = criterion(pred, ys_torch)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch in epochs_to_plot:
        print(f"Loss at epoch {epoch} = {loss}")
```

```
model.eval()
with torch.no_grad():
  ys_prediction = ys_prediction.detach().numpy()


plt.plot(ts, ys, label='true data')
plt.plot(ts, ys_prediction, label='model prediction')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[28], line 28
     26 model.train()
     27 pred = model(ts_torch)
---> 28 loss = criterion(pred, ys_torch)
     30 optimizer.zero_grad()
     31 loss.backward()

File /opt/anaconda3/envs/sci-dev/lib/python3.11/site-packages/torch/nn/modules/
 ↪module.py:1775, in Module._wrapped_call_impl(self, *args, **kwargs)
   1773     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1774 else:
-> 1775     return self._call_impl(*args, **kwargs)

File /opt/anaconda3/envs/sci-dev/lib/python3.11/site-packages/torch/nn/modules/
 ↪module.py:1786, in Module._call_impl(self, *args, **kwargs)
   1781 # If we don't have any hooks, we want to skip the rest of the logic in
   1782 # this function, and just call forward.
   1783 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1784         or _global_backward_pre_hooks or _global_backward_hooks
   1785         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1786     return forward_call(*args, **kwargs)
   1788 result = None
   1789 called_always_called_hooks = set()

File /opt/anaconda3/envs/sci-dev/lib/python3.11/site-packages/torch/nn/modules/
 ↪loss.py:634, in MSELoss.forward(self, input, target)
    630 def forward(self, input: Tensor, target: Tensor) -> Tensor:
    631     """
    632     Runs the forward pass.
    633     """
--> 634     return F.mse_loss(input, target, reduction=self.reduction)
```

```
File /opt/anaconda3/envs/sci-dev/lib/python3.11/site-packages/torch/nn/
 ↪functional.py:3853, in mse_loss(input, target, size_average, reduce,␣
 ↪reduction, weight)
   3841 if has_torch_function_variadic(input, target, weight):
   3842     return handle_torch_function(
   3843         mse_loss,
   3844         (input, target, weight),
   (…)   3850         weight=weight,
   3851     )
-> 3853 if not (target.size() == input.size()):
   3854     warnings.warn(
   3855         f"Using a target size ({target.size()}) that is different to th␣
 ↪input size ({input.size()}). "
   3856         "This will likely lead to incorrect results due to broadcasting ␣
 ↪"
   3857         "Please ensure they have the same size.",
   3858         stacklevel=2,
   3859     )
   3861 if size_average is not None or reduce is not None:

AttributeError: 'NoneType' object has no attribute 'size'
```

[23]:

```
Object `detach` not found.
```

[ ]: