Demo - Generalization and Overfitting Parabola

October 15, 2019

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import random
from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

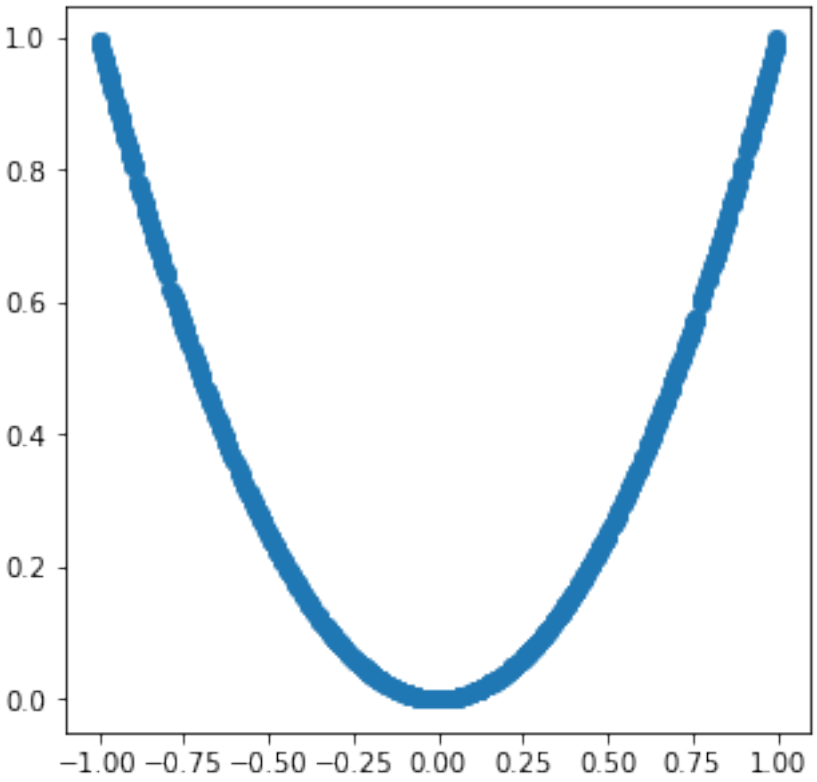
[2]:
```
random.seed(1337)
np.random.seed(1337)
torch.manual_seed(1337)
```

[2]:
```
<torch._C.Generator at 0x11b3ac770>
```

[3]:
```
# generator function (parabolic true function!)
def generate_data(N):
    generator = lambda x : x**2
    x = np.random.rand(N) * 2 - 1
    y = generator(x)
    return x, y
```

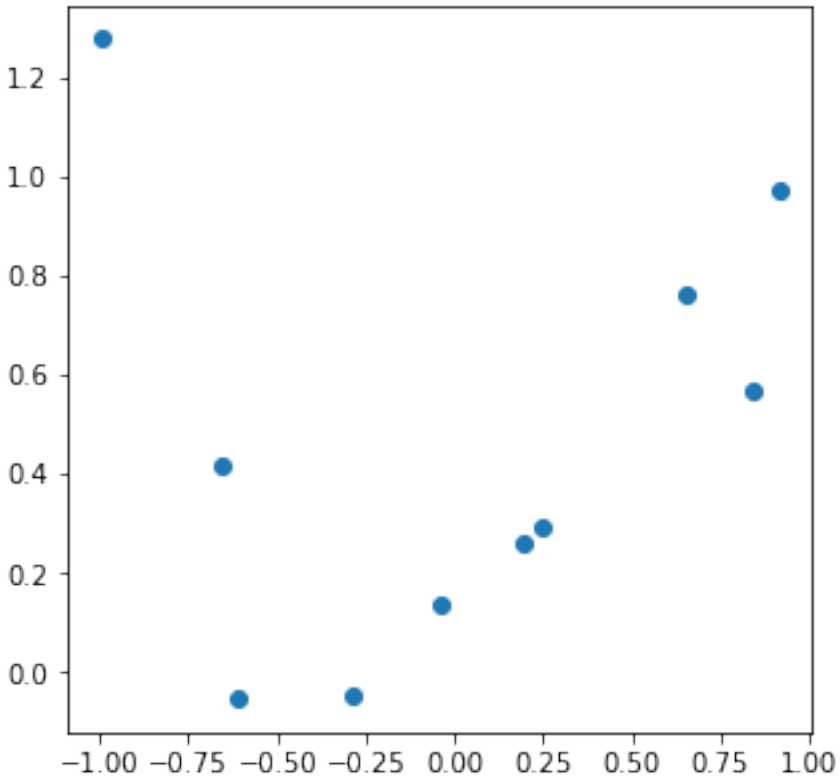
[4]:
```
# generate 1000 data points for testing.
N_test = 1000
x_test, y_test = generate_data(N_test)
```

[5]:
```
plt.figure(figsize=(5,5))
plt.plot(x_test, y_test, 'o')
plt.show()
```



[6]:
```
# generate 10 data points for training.
N_train = 10
x_train, y_train = generate_data(N_train)
y_train = y_train + np.random.randn(N_train) * 0.2
```

[7]:
```
plt.figure(figsize=(5,5))
plt.plot(x_train, y_train, 'o')
plt.show()
```



---

[8]:
```
from torchvision import datasets, transforms
```

[9]:
```
from torch.utils.data import TensorDataset
```

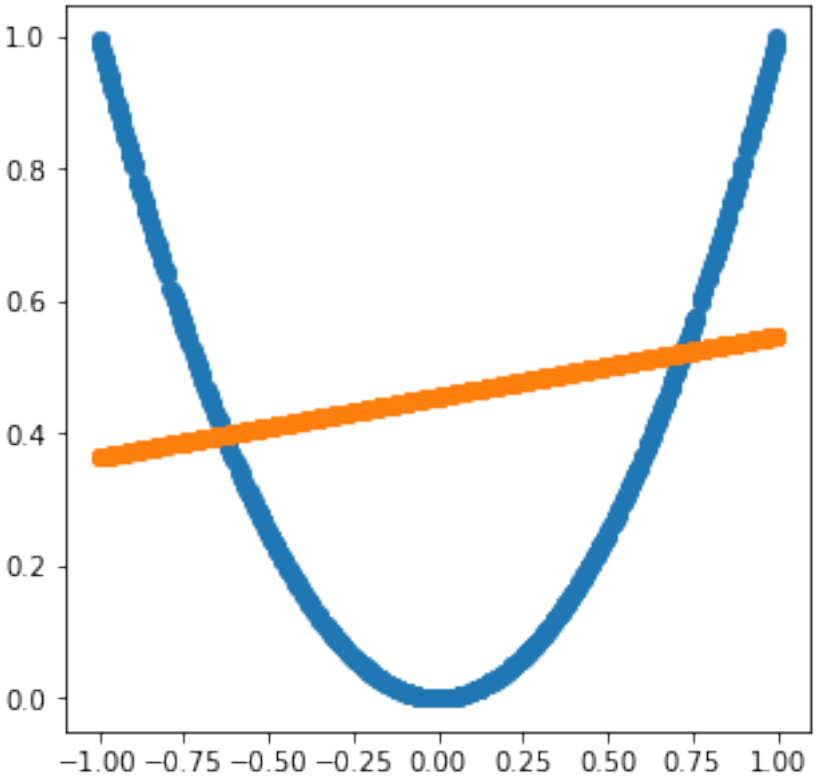
[10]:
```
X_train = torch.from_numpy(x_train).float().unsqueeze(1)
Y_train = torch.from_numpy(y_train).float().unsqueeze(1)
train_dataset = TensorDataset(X_train,Y_train)
```

[11]:
```
X_test = torch.from_numpy(x_test).float().unsqueeze(1)
Y_test = torch.from_numpy(y_test).float().unsqueeze(1)
test_dataset = TensorDataset(X_test, Y_test)
```

[12]:
```
class Linear(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Linear(1, 1)

    def forward(self, data):
        return self.fc(data)

model = Linear()
optimizer = optim.SGD(model.parameters(), lr=1e-1)
```

```
for i in tqdm(range(1000)):
    model.train()
    optimizer.zero_grad()
    Y_pred = model(X_train)
    loss = F.mse_loss(Y_pred, Y_train)
    loss.backward()
    optimizer.step()

y_pred = model(X_test).squeeze(1).detach().numpy()
plt.figure(figsize=(5,5))
plt.plot(x_test, y_test, 'o')
plt.plot(x_test, y_pred, 'o')
plt.show()
```

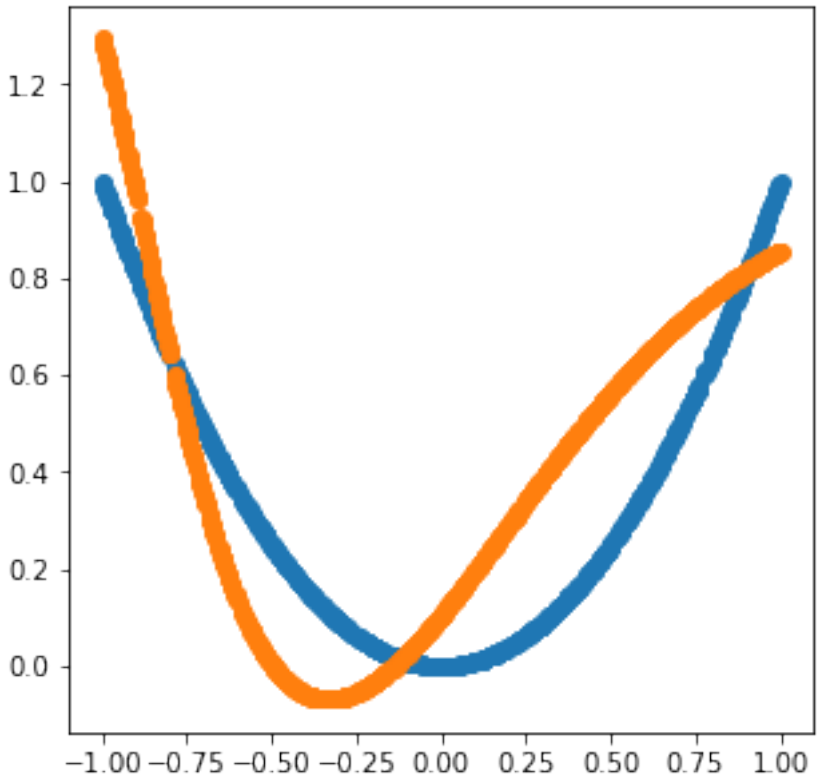
```
100%|| 1000/1000 [00:00<00:00, 3125.97it/s]
```



[13]:
```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Sequential(
            nn.Linear(1, 10),
```

```
            nn.Tanh(),
            nn.Linear(10, 10),
            nn.Tanh(),
            nn.Linear(10, 1))

    def forward(self, data):
        return self.fc(data)

model = MLP()
optimizer = optim.SGD(model.parameters(), lr=1e-1)

for i in tqdm(range(1000)):
    model.train()
    optimizer.zero_grad()
    Y_pred = model(X_train)
    loss = F.mse_loss(Y_pred, Y_train)
    loss.backward()
    optimizer.step()

y_pred = model(X_test).squeeze(1).detach().numpy()
plt.figure(figsize=(5,5))
plt.plot(x_test, y_test, 'o')
plt.plot(x_test, y_pred, 'o')
plt.show()
```

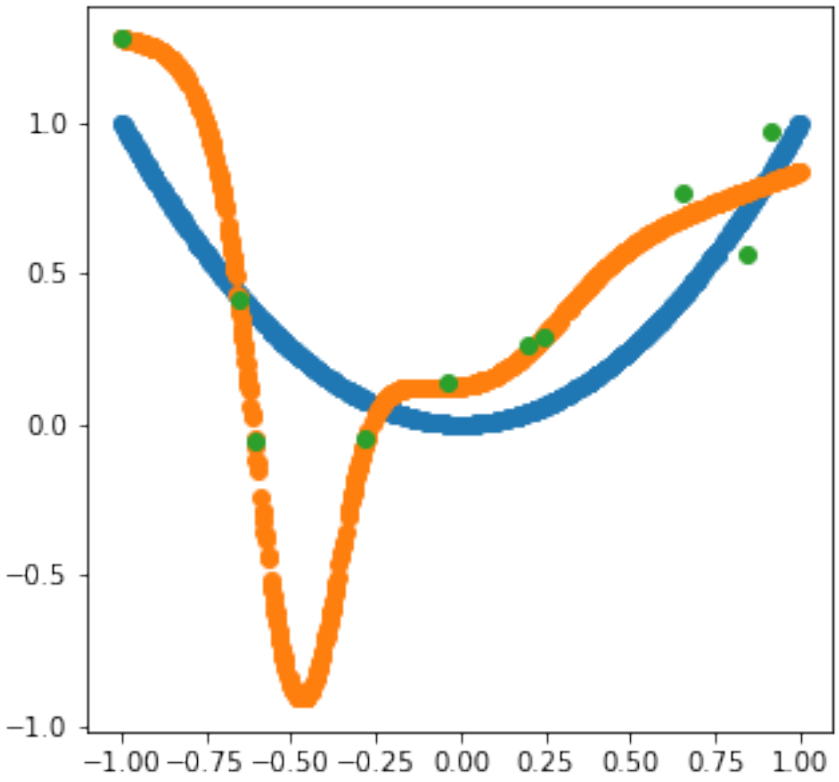
```
100%|| 1000/1000 [00:00<00:00, 1016.57it/s]
```



[14]:
```
class BigMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Sequential(
            nn.Linear(1, 100),
            nn.Tanh(),
            nn.Linear(100, 100),
            nn.Tanh(),
            nn.Linear(100, 100),
            nn.Tanh(),
            nn.Linear(100, 100),
            nn.Tanh(),
            nn.Linear(100, 100),
            nn.Tanh(),
            nn.Linear(100, 1))

    def forward(self, data):
        return self.fc(data)

model = BigMLP()
optimizer = optim.SGD(model.parameters(), lr=1e-1)
```

```
for i in tqdm(range(10000)):
    model.train()
    optimizer.zero_grad()
    Y_pred = model(X_train)
    loss = F.mse_loss(Y_pred, Y_train)
    loss.backward()
    optimizer.step()

y_pred = model(X_test).squeeze(1).detach().numpy()
plt.figure(figsize=(5,5))
plt.plot(x_test, y_test, 'o')
plt.plot(x_test, y_pred, 'o')
plt.plot(x_train ,y_train, 'o')
plt.show()
```

```
100%|| 10000/10000 [00:14<00:00, 712.15it/s]
```



[ ]:

[ ]: