Raken Putra Athallah
1103204186

Pytorch Neural Network Classification

Jaringan saraf (Neural Network) adalah model komputasi yang terinspirasi oleh struktur dan fungsi jaringan saraf biologis dalam otak manusia. Ini terdiri dari unit pengolahan informasi kecil yang disebut neuron atau node yang terhubung dalam lapisan-lapisan. Setiap koneksi antara neuron memiliki bobot, dan selama pelatihan, model ini belajar untuk menyesuaikan bobot-bobot ini berdasarkan data yang diberikan.

Jaringan saraf digunakan untuk memodelkan dan menemukan pola kompleks dalam data, membuatnya sangat efektif dalam tugas-tugas seperti klasifikasi, regresi, pengenalan pola, dan pemrosesan bahasa alami. Ada berbagai jenis arsitektur jaringan saraf, termasuk jaringan saraf feedforward, jaringan saraf konvolusional (CNN) untuk pengolahan gambar, dan jaringan saraf rekurent (RNN) untuk tugas yang melibatkan urutan data, seperti teks atau waktu.

Arsitektur dari sebuah jaringan saraf untuk klasifikasi umumnya terdiri dari komponen-komponen berikut, dan nilai-nilai spesifiknya tergantung pada sifat masalahnya:

1. **Bentuk Layer Masukan (in_features):**
   - Klasifikasi Biner: Sama dengan jumlah fitur dalam data masukan.
   - Klasifikasi Multikelas: Sama seperti dalam klasifikasi biner.
2. **Layer Tersembunyi:**
   - Khusus masalah, dengan minimum 1 lapisan tersembunyi dan maksimum yang bervariasi.
   - Untuk klasifikasi multikelas, arsitekturnya sama seperti klasifikasi biner.
3. **Neuron per Layer Tersembunyi:**
   - Khusus masalah, umumnya berkisar antara 10 hingga 512.
   - Serupa untuk klasifikasi biner dan multikelas.
4. **Bentuk Layer Keluaran (out_features):**
   - Klasifikasi Biner: 1 (mewakili satu kelas atau yang lain).
   - Klasifikasi Multikelas: 1 per kelas (misalnya, 3 untuk foto makanan, orang, atau anjing).
5. **Aktivasi Layer Tersembunyi:**
   - Biasanya ReLU (Rectified Linear Unit), tetapi fungsi aktivasi lain dapat digunakan berdasarkan masalahnya.

6. **Aktivasi Layer Keluaran:**

- Klasifikasi Biner: Aktivasi Sigmoid (torch.sigmoid di PyTorch).
- Klasifikasi Multikelas: Aktivasi Softmax (torch.softmax di PyTorch).

7. **Fungsi Kerugian (Loss Function):**

- Klasifikasi Biner: Binary Crossentropy (torch.nn.BCELoss di PyTorch).
- Klasifikasi Multikelas: Cross-entropy (torch.nn.CrossEntropyLoss di PyTorch).

8. **Optimizer:**

- Optimizer umum yang digunakan termasuk SGD (Stochastic Gradient Descent) dan Adam. Berbagai opsi tersedia di torch.optim.

Pilihan spesifik untuk komponen-komponen ini dapat bervariasi tergantung pada masalah yang dihadapi. Arsitektur umum ini memberikan titik awal, dan penyesuaian dapat dilakukan berdasarkan karakteristik data dan sifat tugas klasifikasi.

1. **Membuat Classification Data**

   Membuat data menggunakan fungsi make_circles dari pustaka scikit-learn untuk membuat dataset sintetis yang berisi sampel-sampel berbentuk dua lingkaran yang saling terkait. Dataset ini dapat digunakan untuk menjelaskan atau menguji model klasifikasi pada tugas-tugas yang memerlukan pemahaman terhadap hubungan non-linier antara fitur dan label.

```python
from sklearn.datasets import make_circles


# Make 1000 samples
n_samples = 1000

# Create circles
X, y = make_circles(n_samples,
                    noise=0.03, # a little bit of noise to the dots
                    random_state=42) # keep random state so we get the same values
```

```python
print(f"First 5 X features:\n{X[:5]}")
print(f"\nFirst 5 y labels:\n{y[:5]}")
```

```
First 5 X features:
[[ 0.75424625  0.23148074]
 [-0.75615888  0.15325888]
 [-0.81539193  0.17328203]
 [-0.39373073  0.69288277]
 [ 0.44220765 -0.89672343]]

First 5 y labels:
[1 1 1 1 0]
```

```python
# Make DataFrame of circle data
import pandas as pd
circles = pd.DataFrame({"X1": X[:, 0],
    "X2": X[:, 1],
    "label": y
})
circles.head(10)
```

	X1	X2	label
0	0.754246	0.231481	1
1	-0.756159	0.153259	1
2	-0.815392	0.173282	1
3	-0.393731	0.692883	1
4	0.442208	-0.896723	0
5	-0.479646	0.676435	1
6	-0.013648	0.803349	1
7	0.771513	0.147760	1
8	-0.169322	-0.793456	1
9	-0.121486	1.021509	0

2. **Mengubah data menjadi tensor dan melakukan split train set dan test set**

Kita harus mengubah data kita menjadi tensors dikarenakan saat ini data yang kita miliki masih berupa numpy array dan pytorch membutuhkan pytorch tensor untuk dapat bekerja. Kemudian melakukan split train set dan test set setelah mengubah data menjadi tensors

```
# Turn data into tensors
# Otherwise this causes issues with computations later on
import torch
X = torch.from_numpy(X).type(torch.float)
y = torch.from_numpy(y).type(torch.float)

# View the first five samples
X[:5], y[:5]
```

```
(tensor([[ 0.7542,  0.2315],
         [-0.7562,  0.1533],
         [-0.8154,  0.1733],
         [-0.3937,  0.6929],
         [ 0.4422, -0.8967]]),
 tensor([1., 1., 1., 1., 0.]))
```

```
# Split data into train and test sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2, # 20% test, 80% train
                                                    random_state=42) # make the random split reproducible

len(X_train), len(X_test), len(y_train), len(y_test)
```

```
(800, 200, 800, 200)
```

3. **Membuat Model**

Tahap berikutnya yang dilakukan adalah membuat model neural network sederhana untuk masalah klasifikasi

```
# 1. Construct a model class that subclasses nn.Module
class CircleModelV0(nn.Module):
    def __init__(self):
        super().__init__()
        # 2. Create 2 nn.Linear layers capable of handling X and y input and output shapes
        self.layer_1 = nn.Linear(in_features=2, out_features=5) # takes in 2 features (X), produces 5 features
        self.layer_2 = nn.Linear(in_features=5, out_features=1) # takes in 5 features, produces 1 feature (y)

    # 3. Define a forward method containing the forward pass computation
    def forward(self, x):
        # Return the output of layer_2, a single feature, the same shape as y
        return self.layer_2(self.layer_1(x)) # computation goes through layer_1 first then the output of layer_1 goes through layer_2

# 4. Create an instance of the model and send it to target device
model_0 = CircleModelV0().to(device)
model_0
```

```
CircleModelV0(
  (layer_1): Linear(in_features=2, out_features=5, bias=True)
  (layer_2): Linear(in_features=5, out_features=1, bias=True)
)
```

4. **Setup loss function and optimizer**

Sama seperti chapter sebelumnya kita perlu membuat loss function dan juga optimizer terhadap model yang telah kita buat untuk mendapatkan hasil yang lebih optimal dari model yang kita miliki

```
# Create a loss function
# loss_fn = nn.BCELoss() # BCELoss = no sigmoid built-in
loss_fn = nn.BCEWithLogitsLoss() # BCEWithLogitsLoss = sigmoid built-in

# Create an optimizer
optimizer = torch.optim.SGD(params=model_0.parameters(),
                            lr=0.1)
```

5. **Train model**

Setelah membuat loss function dan optimizer kita dapat membuat training loop untuk model yang telah kita buat

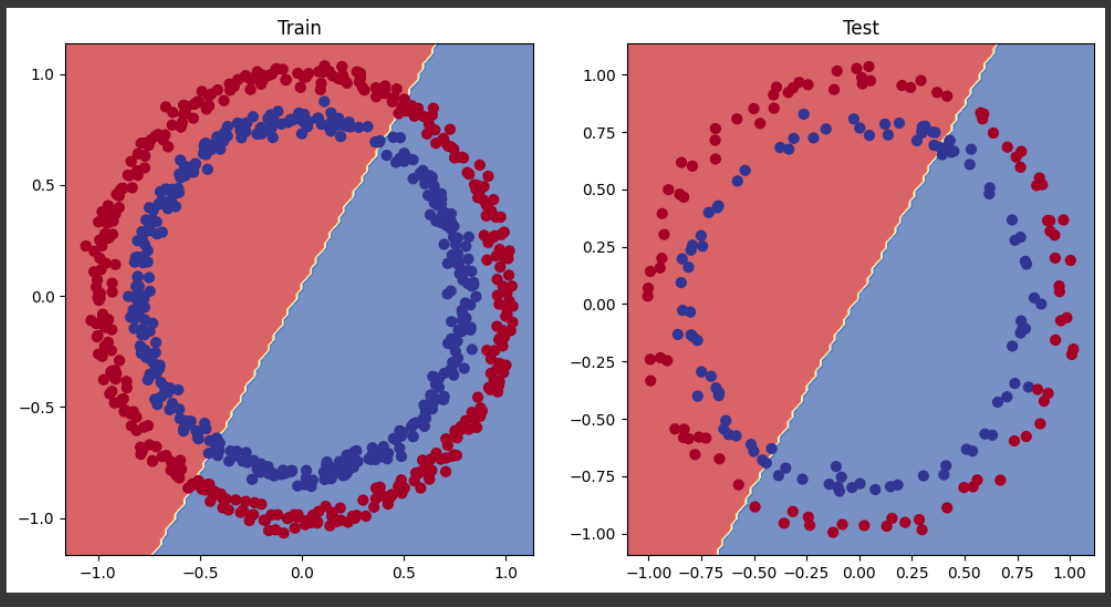
```
torch.manual_seed(42)

# Set the number of epochs
epochs = 100

# Put data to target device
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)

# Build training and evaluation loop
for epoch in range(epochs):
    ### Training
    model_0.train()

    # 1. Forward pass (model outputs raw logits)
    y_logits = model_0(X_train).squeeze() # squeeze to remove extra `1` dimensions, this won't work unless model and data are on same device
    y_pred = torch.round(torch.sigmoid(y_logits)) # turn logits -> pred probs -> pred labls

    # 2. Calculate loss/accuracy
    # loss = loss_fn(torch.sigmoid(y_logits), # Using nn.BCELoss you need torch.sigmoid()
    #                y_train)
    loss = loss_fn(y_logits, # Using nn.BCEWithLogitsLoss works with raw logits
                   y_train)
    acc = accuracy_fn(y_true=y_train,
                      y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_0.eval()
    with torch.inference_mode():
        # 1. Forward pass
        test_logits = model_0(X_test).squeeze()
        test_pred = torch.round(torch.sigmoid(test_logits))
        # 2. Caculate loss/accuracy
        test_loss = loss_fn(test_logits,
                            y_test)
        test_acc = accuracy_fn(y_true=y_test,
                               y_pred=test_pred)

    # Print out what's happening every 10 epochs
    if epoch % 10 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% | Test loss: {test_loss:.5f}, Test acc: {test_acc:.2f}%")
```

```
Epoch: 0 | Loss: 0.70548, Accuracy: 50.00% | Test loss: 0.70129, Test acc: 50.00%
Epoch: 10 | Loss: 0.69854, Accuracy: 50.00% | Test loss: 0.69548, Test acc: 50.00%
Epoch: 20 | Loss: 0.69582, Accuracy: 44.75% | Test loss: 0.69347, Test acc: 45.00%
Epoch: 30 | Loss: 0.69470, Accuracy: 47.62% | Test loss: 0.69284, Test acc: 48.50%
Epoch: 40 | Loss: 0.69418, Accuracy: 48.25% | Test loss: 0.69270, Test acc: 50.00%
Epoch: 50 | Loss: 0.69390, Accuracy: 49.00% | Test loss: 0.69273, Test acc: 49.50%
Epoch: 60 | Loss: 0.69372, Accuracy: 49.12% | Test loss: 0.69281, Test acc: 49.00%
Epoch: 70 | Loss: 0.69359, Accuracy: 50.25% | Test loss: 0.69290, Test acc: 48.50%
Epoch: 80 | Loss: 0.69349, Accuracy: 50.12% | Test loss: 0.69300, Test acc: 47.50%
Epoch: 90 | Loss: 0.69340, Accuracy: 50.88% | Test loss: 0.69310, Test acc: 47.00%
```

6. Evaluasi Model

Setelah membuat train loop untuk model yang telah dibuat, kita melakukan evaluasi terhadap model kita setelah dilakukan training. Pada proses ini kita dapat melakukan evaluasi dengan memvisualisasikan data untuk mempermudah proses evaluasi model

```
# Plot decision boundaries for training and test sets
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Train")
plot_decision_boundary(model_0, X_train, y_train)
plt.subplot(1, 2, 2)
plt.title("Test")
plot_decision_boundary(model_0, X_test, y_test)
```



7. Non Linearity Pada Model

Jika kita memiliki data non linear akan tetapi kita tidak menggunakan fungsi non linearity pada model kita seperti relu dan sigmoid maka terkadang akan berpengaruh pada hasil training dari model kita. Oleh karena itu non linearity model harus disesuaikan dengan jenis data kita apakah termasuk linear atau tidak berikut adalah contoh hasil training dari model tanpa menggunakan fungsi non linearity dan model dengan fungsi linearity

```
# Plot decision boundaries for training and test sets
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Train")
plot_decision_boundary(model_1, X_train, y_train)
plt.subplot(1, 2, 2)
plt.title("Test")
plot_decision_boundary(model_1, X_test, y_test)
```