Task 1: Classification

- Create a Neural Network model
- Define optimization procedure
- Train Classifier
- Evaluate model on test set

In [1]:

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np
from utils import NoisyFashionMNIST

%matplotlib inline
def show(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1,2,0)), interpolation='nearest')
```

Dataset

In [2]:

```python
transform=transforms.Compose([
        transforms.ToTensor()])

train_dataset = datasets.FashionMNIST("./data", train = True, download=True, transform=transform)
test_dataset = datasets.FashionMNIST("./data", train = False, download=True, transform=transform)


idx_to_class = {v: k for k, v in train_dataset.class_to_idx.items()}
```

In [3]:

```python
x,y = train_dataset[1]
idx_to_class[y]
```

Out[3]:

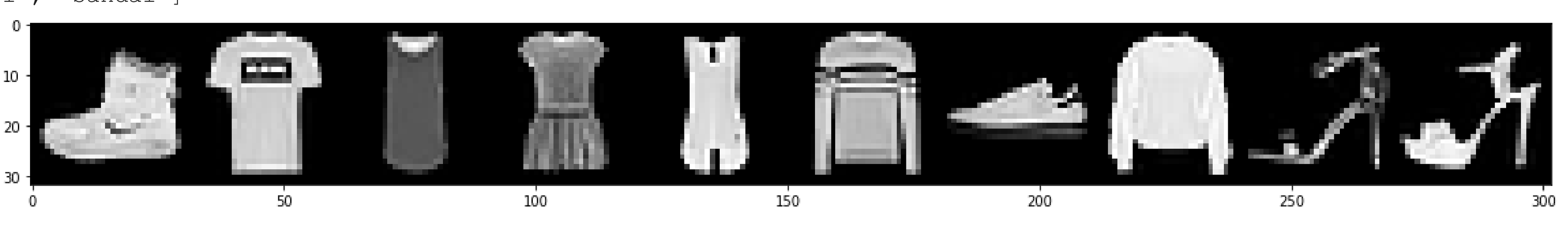
```
'T-shirt/top'
```

In [4]:

```python
x = [train_dataset[i][0] for i in range(10)]
labels = [idx_to_class[train_dataset[i][1]] for i in range(10)]
print(labels)

plt.figure(figsize=(20,10))
show(torchvision.utils.make_grid(x, nrow=10))
plt.show()
```

```
['Ankle boot', 'T-shirt/top', 'T-shirt/top', 'Dress', 'T-shirt/top', 'Pullover', 'Sneaker', 'Pullover', 'Sanda
l', 'Sandal']
```



In [5]:

```python
class network(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(784,256)
        self.layer2 = nn.Linear(256,128)
        self.layer3 = nn.Linear(128,64)
        self.layer4 = nn.Linear(64,10)
        self.dropout = nn.Dropout(0.2)

    def forward(self,x):
        x = x.view(x.shape[0],-1)
        x = self.dropout(F.relu(self.layer1(x)))
        x = self.dropout(F.relu(self.layer2(x)))
        x = self.dropout(F.relu(self.layer3(x)))
        x = F.log_softmax(self.layer4(x),dim=1)
        return x

from torch.optim.lr_scheduler import StepLR

def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()), end='\r')

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
    test_loss, correct, len(test_loader.dataset),
    100. * correct / len(test_loader.dataset)), end='\r')


train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64)
test_loader = torch.utils.data.DataLoader(test_dataset,  batch_size=64)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = network().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

scheduler = StepLR(optimizer, step_size=5, gamma=0.1)

for epoch in range(1,10 + 1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
    scheduler.step()
```

```
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.705907
Test set: Average loss: 0.4501, Accuracy: 8348/10000 (83%)
Train Epoch: 2 [59520/60000 (99%)]      Loss: 0.579034
Test set: Average loss: 0.4084, Accuracy: 8502/10000 (85%)
Train Epoch: 3 [59520/60000 (99%)]      Loss: 0.486624
Test set: Average loss: 0.3830, Accuracy: 8612/10000 (86%)
Train Epoch: 4 [59520/60000 (99%)]      Loss: 0.520283
Test set: Average loss: 0.3840, Accuracy: 8603/10000 (86%)
Train Epoch: 5 [59520/60000 (99%)]      Loss: 0.629630
Test set: Average loss: 0.3824, Accuracy: 8607/10000 (86%)
Train Epoch: 6 [59520/60000 (99%)]      Loss: 0.406237
Test set: Average loss: 0.3363, Accuracy: 8800/10000 (88%)
Train Epoch: 7 [59520/60000 (99%)]      Loss: 0.316511
Test set: Average loss: 0.3336, Accuracy: 8807/10000 (88%)
Train Epoch: 8 [59520/60000 (99%)]      Loss: 0.371744
Test set: Average loss: 0.3331, Accuracy: 8819/10000 (88%)
Train Epoch: 9 [59520/60000 (99%)]      Loss: 0.400399
Test set: Average loss: 0.3299, Accuracy: 8839/10000 (88%)
Train Epoch: 10 [59520/60000 (99%)]     Loss: 0.338608
Test set: Average loss: 0.3286, Accuracy: 8845/10000 (88%)
```

Task 2: Train Autoencoder

- Create a Neural Network model
- Define optimization procedure
- Train Classifier
- Evaluate model on test set

In [6]:

```python
train_dataset = NoisyFashionMNIST("./data", True)
test_dataset = NoisyFashionMNIST("./data", False)
```

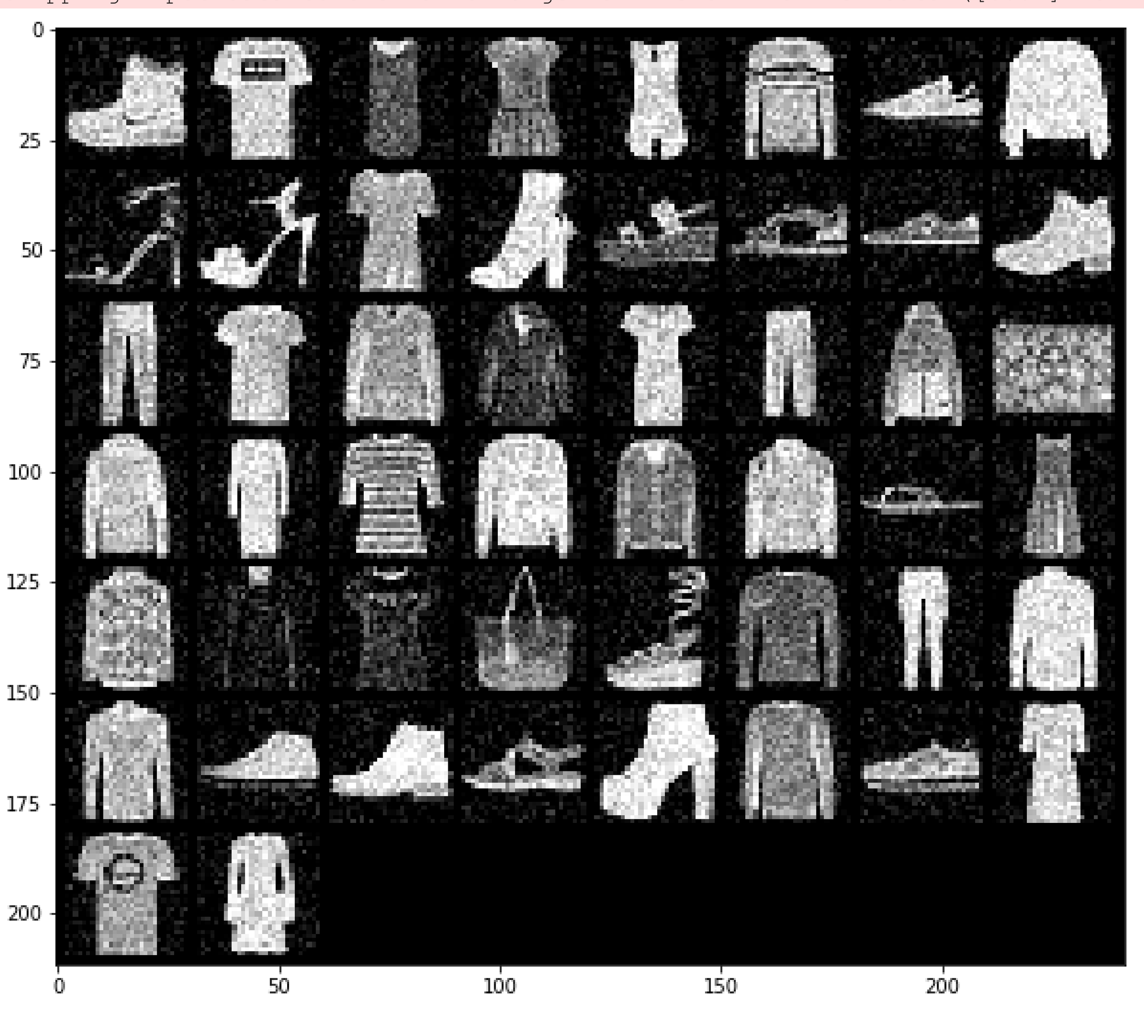
In [26]:

```python
x = [train_dataset[i][0] for i in range(50)]
y = [train_dataset[i][1] for i in range(50)]

plt.figure(figsize=(10,10))
show(torchvision.utils.make_grid(x))
plt.show()

plt.figure(figsize=(10,10))
show(torchvision.utils.make_grid(y))
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



In [25]:

```python
class AE(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = torch.nn.Sequential(
            torch.nn.Linear(28 * 28, 128),
            torch.nn.ReLU(),
            torch.nn.Linear(128, 64),
            torch.nn.ReLU(),
            torch.nn.Linear(64, 36),
            torch.nn.ReLU(),
            torch.nn.Linear(36, 18),
            torch.nn.ReLU(),
            torch.nn.Linear(18, 9)
        )
        self.decoder = torch.nn.Sequential(
            torch.nn.Linear(9, 18),
            torch.nn.ReLU(),
            torch.nn.Linear(18, 36),
            torch.nn.ReLU(),
            torch.nn.Linear(36, 64),
            torch.nn.ReLU(),
            torch.nn.Linear(64, 128),
            torch.nn.ReLU(),
            torch.nn.Linear(128, 28 * 28),
            torch.nn.Sigmoid()
        )
    def forward(self, x):
        encode = self.encoder(x)
        decode = self.decoder(encode)
        return decode

def train(model, device, train_loader, optimizer, epoch):
    model.train()
    loss_func = torch.nn.MSELoss()
    for batch_idx, (data,target) in enumerate(train_loader):

        data,target = data.to(device),target.to(device)
        data = data.reshape(-1,28*28)
        output = model(data)
        loss = loss_func(output, target.reshape(-1,28*28))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()), end='\r')

def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    mse = 0  # I didn't find a good measurement to evaluate the performance of autoencoder so I use mean square
    loss_func = torch.nn.MSELoss()
    with torch.no_grad():
        for batch_idx,(data,target) in enumerate(test_loader):
            data,target = data.to(device),target.to(device)
            data = data.reshape(-1,28*28)
            target = target.reshape(-1,28*28)
            output = model(data)
            test_loss += loss_func(output, target)
            mse += ((output-target)**2/(28*28)).sum()

    print('\nTest set: Average loss: {:.4f}, mean square error: {}\n'.format(test_loss.item()/156,mse.item()/15


train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64)
test_loader = torch.utils.data.DataLoader(test_dataset,  batch_size=64)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = AE().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

scheduler = StepLR(optimizer, step_size=5, gamma=0.1)

for epoch in range(1,10+1):
    train(model, device, train_loader, optimizer, epoch)
    test(model, device, test_loader)
    scheduler.step()
```

```
Train Epoch: 1 [59520/60000 (99%)]      Loss: 0.028735
Test set: Average loss: 0.0320, mean square error: 2.037291697966747
Train Epoch: 2 [59520/60000 (99%)]      Loss: 0.024060
Test set: Average loss: 0.0272, mean square error: 1.727043934357472
Train Epoch: 3 [59520/60000 (99%)]      Loss: 0.021906
Test set: Average loss: 0.0245, mean square error: 1.5603382404033954
Train Epoch: 4 [59520/60000 (99%)]      Loss: 0.021034
Test set: Average loss: 0.0233, mean square error: 1.4812427422939203
Train Epoch: 5 [59520/60000 (99%)]      Loss: 0.019782
Test set: Average loss: 0.0222, mean square error: 1.4096292349008412
Train Epoch: 6 [59520/60000 (99%)]      Loss: 0.019361
Test set: Average loss: 0.0216, mean square error: 1.3712111253004808
Train Epoch: 7 [59520/60000 (99%)]      Loss: 0.019304
Test set: Average loss: 0.0214, mean square error: 1.362516354291867
Train Epoch: 8 [59520/60000 (99%)]      Loss: 0.019188
Test set: Average loss: 0.0213, mean square error: 1.3541761545034556
Train Epoch: 9 [59520/60000 (99%)]      Loss: 0.019158
Test set: Average loss: 0.0212, mean square error: 1.345805437136919
Train Epoch: 10 [59520/60000 (99%)]     Loss: 0.019002
Test set: Average loss: 0.0211, mean square error: 1.33967531644381
```