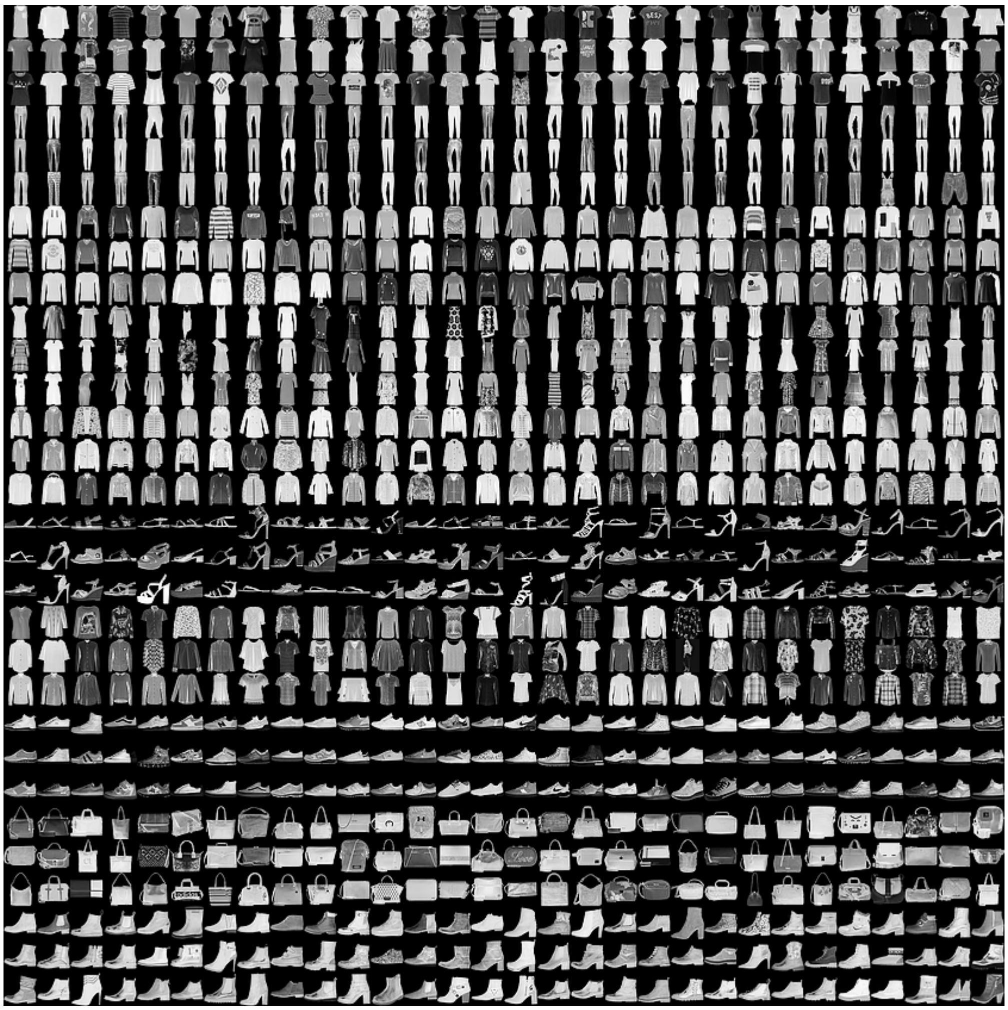
Classifying Fashion-MNIST

Now it's your turn to build and train a neural network. You'll be using the Fashion-MNIST dataset (https://github.com/zalandoresearch/fashion-mnist), a drop-in replacement for the MNIST dataset. MNIST is actually quite trivial with neural networks where you can easily achieve better than 97% accuracy. Fashion-MNIST is a set of 28x28 greyscale images of clothes. It's more complex than MNIST, so it's a better representation of the actual performance of your network, and a better representation of datasets you'll use in the real world.



In this notebook, you'll build your own neural network. For the most part, you could just copy and paste the code from Part 3, but you wouldn't be learning. It's important for you to write the code yourself and get it to work. Feel free to consult the previous notebooks though as you work through this.

First off, let's load the dataset through torchvision.

In [1]:
```
import torch
from torchvision import datasets, transforms
import helper

# Define a transform to normalize the data
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.
5, 0.5))])
# Download and load the training data
trainset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True, tr
ain=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=Tru
e)

# Download and load the test data
testset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True, tra
in=False, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=True)
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-
labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-i
mages-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-l
abels-idx1-ubyte.gz
Processing...
Done!
```

Here we can see one of the images.

In [2]:
```
image, label = next(iter(trainloader))
helper.imshow(image[0,:]);
```

Building the network

Here you should define your network. As with MNIST, each image is 28x28 which is a total of 784 pixels, and there are 10 classes. You should include at least one hidden layer. We suggest you use ReLU activations for the layers and to return the logits or log-softmax from the forward pass. It's up to you how many layers you add and the size of those layers.

In [23]:

```
# TODO: Define your network architecture here
from torch import nn, optim
import torch.nn.functional as F

class Network(nn.Module):
    def __init__(self):
        super().__init__()

        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 32)
        self.fc5 = nn.Linear(32, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = F.log_softmax(self.fc5(x), dim=1)

        return x
```

Train the network

Now you should create your network and train it. First you'll want to define [the criterion (http://pytorch.org/docs/master/nn.html#loss-functions)](http://pytorch.org/docs/master/nn.html#loss-functions) (something like `nn.CrossEntropyLoss`) and [the optimizer (http://pytorch.org/docs/master/optim.html)](http://pytorch.org/docs/master/optim.html) (typically `optim.SGD` or `optim.Adam`).

Then write the training code. Remember the training pass is a fairly straightforward process:

- Make a forward pass through the network to get the logits
- Use the logits to calculate the loss
- Perform a backward pass through the network with `loss.backward()` to calculate the gradients
- Take a step with the optimizer to update the weights

By adjusting the hyperparameters (hidden units, learning rate, etc), you should be able to get the training loss below 0.4.

In [26]:
```python
# TODO: Create the network, define the criterion and optimizer
learning_rate = 0.005

model = Network()
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), learning_rate)
```

In [27]:
```python
# TODO: Train the network here
epochs = 10
for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)

        optimizer.zero_grad()
        output = model.forward(images)

        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    else:
        print(f"training loss {e}: {running_loss/len(trainloader)}")
```

```
training loss 0: 0.5511114772861955
training loss 1: 0.41583031705066337
training loss 2: 0.37926690796735696
training loss 3: 0.3623951860645941
training loss 4: 0.3478105309913789
training loss 5: 0.3279548869776065
training loss 6: 0.31715129823413996
training loss 7: 0.3163430090588547
training loss 8: 0.3132802788049046
training loss 9: 0.29613090787869273
```

In [35]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import helper

# Test out your network!

dataiter = iter(testloader)
images, labels = dataiter.next()
img = images[0]
# Convert 2D image to 1D vector

#img = img.resize_(1, 784)
img = img.view(img.shape[0], -1)

# TODO: Calculate the class probabilities (softmax) for img

ps = torch.exp(model(img))

# Plot the image and probabilities
helper.view_classify(img.resize_(1, 28, 28), ps, version='Fashion')
```