#Apa yang akan kita bahas

Mari kita mulai dengan memasukkan apa yang kita bahas ke dalam kamus untuk referensi nanti.

```
what_were_covering = {1: "data (prepare and load)",
    2: "build model",
    3: "fitting the model to data (training)",
    4: "making predictions and evaluating a model (inference)",
    5: "saving and loading a model",
    6: "putting it all together"
}
```

Kita akan mendapatkan torch, torch.nn (nn adalah singkatan dari jaringan saraf dan paket ini berisi blok penyusun untuk membuat jaringan saraf di PyTorch) dan matplotlib.

```
import torch
from torch import nn # nn contains all of PyTorch's building blocks
for neural networks
import matplotlib.pyplot as plt

# Check PyTorch version
torch.__version__
```

```
{"type":"string"}
```

Mari kita buat data kita sebagai garis lurus.

```
# Create *known* parameters
weight = 0.7
bias = 0.3

# Create data
start = 0
end = 1
step = 0.02
X = torch.arange(start, end, step).unsqueeze(dim=1)
y = weight * X + bias

X[:10], y[:10]
```

```
(tensor([[0.0000],
         [0.0200],
         [0.0400],
         [0.0600],
         [0.0800],
         [0.1000],
         [0.1200],
         [0.1400],
         [0.1600],
```

```
         [0.1800]]),
 tensor([[0.3000],
         [0.3140],
         [0.3280],
         [0.3420],
         [0.3560],
         [0.3700],
         [0.3840],
         [0.3980],
         [0.4120],
         [0.4260]]))
```

Kita dapat membuatnya dengan memisahkan tensor X dan y.

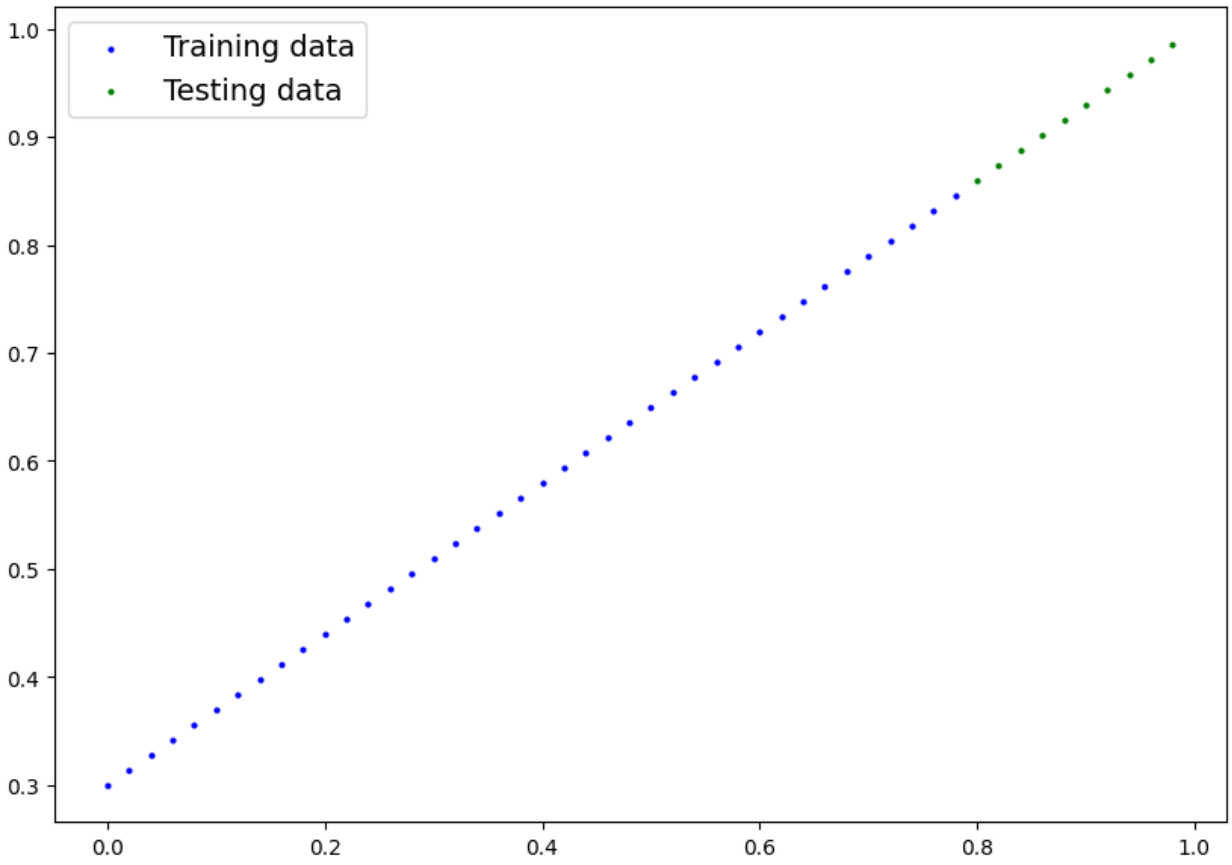
```
# Create train/test split
train_split = int(0.8 * len(X)) # 80% of data used for training set,
20% for testing
X_train, y_train = X[:train_split], y[:train_split]
X_test, y_test = X[train_split:], y[train_split:]

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

```
(40, 40, 10, 10)
```

Mari buat fungsi untuk memvisualisasikannya.

```
def plot_predictions(train_data=X_train,
                     train_labels=y_train,
                     test_data=X_test,
                     test_labels=y_test,
                     predictions=None):
  """
  Plots training data, test data and compares predictions.
  """
  plt.figure(figsize=(10, 7))

  # Plot training data in blue
  plt.scatter(train_data, train_labels, c="b", s=4, label="Training
data")

  # Plot test data in green
  plt.scatter(test_data, test_labels, c="g", s=4, label="Testing
data")

  if predictions is not None:
    # Plot the predictions in red (predictions were made on the test
data)
    plt.scatter(test_data, predictions, c="r", s=4,
label="Predictions")
```

```
  # Show the legend
  plt.legend(prop={"size": 14});

plot_predictions();
```



Mari kita replikasi model regresi linier standar menggunakan PyTorch murni

```
# Create a Linear Regression model class
class LinearRegressionModel(nn.Module): # <- almost everything in
PyTorch is a nn.Module (think of this as neural network lego blocks)
    def __init__(self):
        super().__init__()
        self.weights = nn.Parameter(torch.randn(1, # <- start with
random weights (this will get adjusted as the model learns)
                                                dtype=torch.float), #
<- PyTorch loves float32 by default
                                       requires_grad=True) # <- can we
update this value with gradient descent?)

        self.bias = nn.Parameter(torch.randn(1, # <- start with random
bias (this will get adjusted as the model learns)
                                             dtype=torch.float), # <-
```

```
PyTorch loves float32 by default
                                    requires_grad=True) # <- can we update
this value with gradient descent?))

    # Forward defines the computation in the model
    def forward(self, x: torch.Tensor) -> torch.Tensor: # <- "x" is
the input data (e.g. training/testing features)
        return self.weights * x + self.bias # <- this is the linear
regression formula (y = m*x + b)
```

#Memeriksa konten model PyTorch

Sekarang kita sudah menyelesaikannya, mari kita buat instance model dengan kelas yang telah kita buat dan periksa parameternya menggunakan .parameters().

```
# Set manual seed since nn.Parameter are randomly initialzied
torch.manual_seed(42)

# Create an instance of the model (this is a subclass of nn.Module
that contains nn.Parameter(s))
model_0 = LinearRegressionModel()

# Check the nn.Parameter(s) within the nn.Module subclass we created
list(model_0.parameters())
```

```
[Parameter containing:
 tensor([0.3367], requires_grad=True),
 Parameter containing:
 tensor([0.1288], requires_grad=True)]
```

Kita juga bisa mendapatkan status (isi model) dari model menggunakan .state_dict()

```
# List named parameters
model_0.state_dict()
```

```
OrderedDict([('weights', tensor([0.3367])), ('bias',
tensor([0.1288]))])
```

Mari kita membuat beberapa prediksi.

```
# Make predictions with model
with torch.inference_mode():
    y_preds = model_0(X_test)

# Note: in older PyTorch code you might also see torch.no_grad()
# with torch.no_grad():
#   y_preds = model_0(X_test)
```

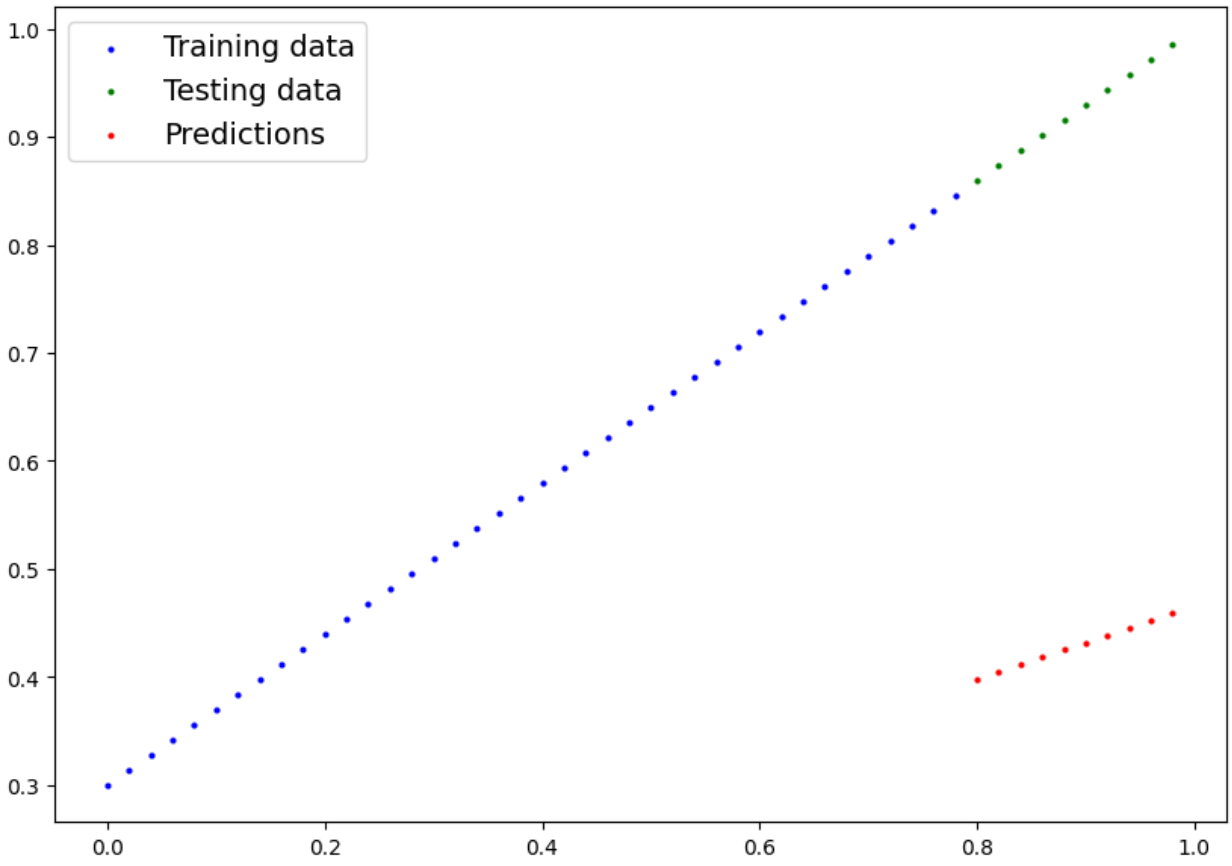
Kami telah membuat beberapa prediksi, mari kita lihat seperti apa

```
# Check the predictions
print(f"Number of testing samples: {len(X_test)}")
print(f"Number of predictions made: {len(y_preds)}")
print(f"Predicted values:\n{y_preds}")
```

```
Number of testing samples: 10
Number of predictions made: 10
Predicted values:
tensor([[0.3982],
        [0.4049],
        [0.4116],
        [0.4184],
        [0.4251],
        [0.4318],
        [0.4386],
        [0.4453],
        [0.4520],
        [0.4588]])
```

Prediksi kita masih berupa angka pada sebuah halaman, mari kita visualisasikannya dengan fungsi plot_predictions() yang kita buat di atas.

```
plot_predictions(predictions=y_preds)
```



```
y_test - y_preds
```

```
tensor([[0.4618],
        [0.4691],
        [0.4764],
        [0.4836],
        [0.4909],
        [0.4982],
        [0.5054],
        [0.5127],
        [0.5200],
        [0.5272]])
```

#Membuat fungsi kerugian dan pengoptimal di PyTorch

```
# Create the loss function
loss_fn = nn.L1Loss() # MAE loss is same as L1Loss

# Create the optimizer
optimizer = torch.optim.SGD(params=model_0.parameters(), # parameters
of target model to optimize
                            lr=0.01) # learning rate (how much the
optimizer should change parameters at each step, higher=more (less
stable), lower=less (might take a long time))
```

#Membuat loop pengoptimalan di PyTorch

Mari kita gabungkan semua hal di atas dan latih model kita selama 100 epoch (melewati data) dan kita akan mengevaluasinya setiap 10 epoch.

```
torch.manual_seed(42)

# Set the number of epochs (how many times the model will pass over
the training data)
epochs = 100

# Create empty loss lists to track values
train_loss_values = []
test_loss_values = []
epoch_count = []

for epoch in range(epochs):
    ### Training

    # Put model in training mode (this is the default state of a
model)
    model_0.train()

    # 1. Forward pass on train data using the forward() method inside
    y_pred = model_0(X_train)
```

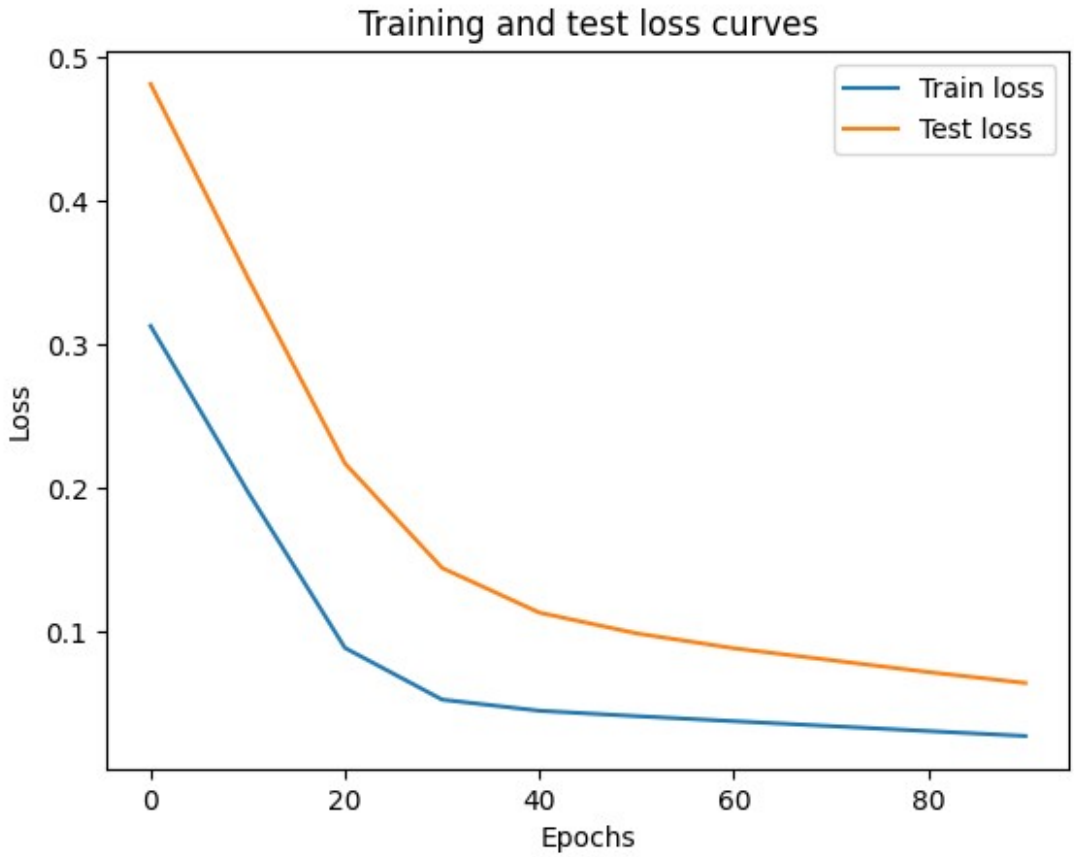
```
    # print(y_pred)

    # 2. Calculate the loss (how different are our models predictions
to the ground truth)
    loss = loss_fn(y_pred, y_train)

    # 3. Zero grad of the optimizer
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Progress the optimizer
    optimizer.step()

    ### Testing

    # Put the model in evaluation mode
    model_0.eval()

    with torch.inference_mode():
      # 1. Forward pass on test data
      test_pred = model_0(X_test)

      # 2. Caculate loss on test data
      test_loss = loss_fn(test_pred, y_test.type(torch.float)) #
predictions come in torch.float datatype, so comparisons need to be
done with tensors of the same type

      # Print out what's happening
      if epoch % 10 == 0:
            epoch_count.append(epoch)
            train_loss_values.append(loss.detach().numpy())
            test_loss_values.append(test_loss.detach().numpy())
            print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test
Loss: {test_loss} ")
```

```
Epoch: 0 | MAE Train Loss: 0.31288138031959534 | MAE Test Loss:
0.48106518387794495
Epoch: 10 | MAE Train Loss: 0.1976713240146637 | MAE Test Loss:
0.3463551998138428
Epoch: 20 | MAE Train Loss: 0.08908725529909134 | MAE Test Loss:
0.21729660034179688
Epoch: 30 | MAE Train Loss: 0.053148526698350906 | MAE Test Loss:
0.14464017748832703
Epoch: 40 | MAE Train Loss: 0.04543796554207802 | MAE Test Loss:
0.11360953003168106
Epoch: 50 | MAE Train Loss: 0.04167863354086876 | MAE Test Loss:
0.09919948130846024
Epoch: 60 | MAE Train Loss: 0.03818932920694351 | MAE Test Loss:
```

```
0.08886633068323135
Epoch: 70 | MAE Train Loss: 0.03476089984178543 | MAE Test Loss:
0.0805937647819519
Epoch: 80 | MAE Train Loss: 0.03132382780313492 | MAE Test Loss:
0.07232122868299484
Epoch: 90 | MAE Train Loss: 0.02788739837706089 | MAE Test Loss:
0.06473556160926819
```

Sepertinya kerugian kita semakin berkurang di setiap zaman, mari kita plot untuk mengetahuinya.

```
# Plot the loss curves
plt.plot(epoch_count, train_loss_values, label="Train loss")
plt.plot(epoch_count, test_loss_values, label="Test loss")
plt.title("Training and test loss curves")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend();
```



Mari kita periksa .state_dict() model kita untuk melihat seberapa dekat model kita dengan nilai asli yang kita tetapkan untuk bobot dan bias.

```
# Find our model's learned parameters
print("The model learned the following values for weights and bias:")
print(model_0.state_dict())
print("\nAnd the original values for weights and bias are:")
print(f"weights: {weight}, bias: {bias}")
```

```
The model learned the following values for weights and bias:
OrderedDict([('weights', tensor([0.5784])), ('bias',
tensor([0.3513]))])

And the original values for weights and bias are:
weights: 0.7, bias: 0.3
```

Membuat prediksi dengan model PyTorch terlatih

(inferensi)

Dua item pertama memastikan semua perhitungan dan pengaturan berguna yang digunakan PyTorch di belakang layar selama pelatihan tetapi tidak diperlukan untuk inferensi dimatikan (ini menghasilkan perhitungan yang lebih cepat). Dan yang ketiga memastikan Anda tidak mengalami kesalahan lintas perangkat.

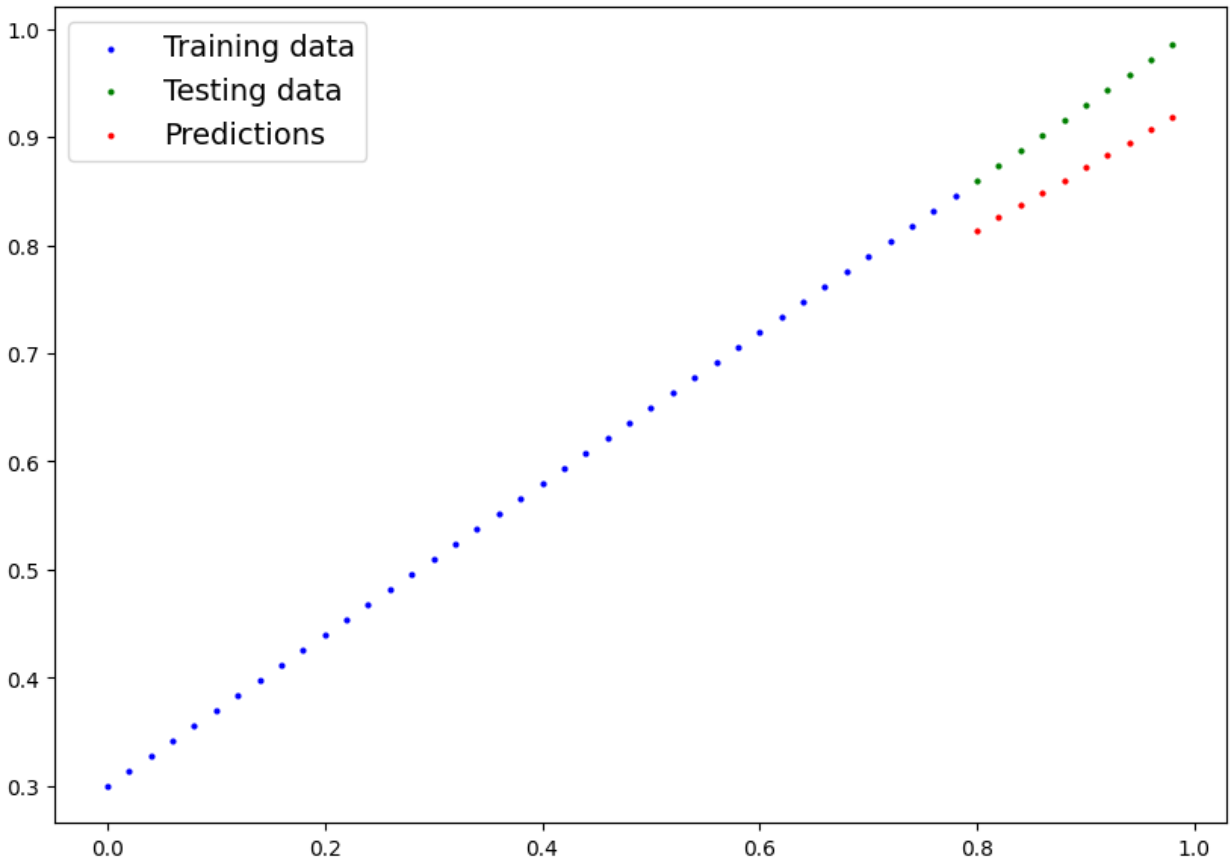
```
# 1. Set the model in evaluation mode
model_0.eval()

# 2. Setup the inference mode context manager
with torch.inference_mode():
  # 3. Make sure the calculations are done with the model and data on
the same device
  # in our case, we haven't setup device-agnostic code yet so our data
and model are
  # on the CPU by default.
  # model_0.to(device)
  # X_test = X_test.to(device)
  y_preds = model_0(X_test)
y_preds
```

```
tensor([[0.8141],
        [0.8256],
        [0.8372],
        [0.8488],
        [0.8603],
        [0.8719],
        [0.8835],
        [0.8950],
```

```
        [0.9066],
        [0.9182]])
```

Kami telah membuat beberapa prediksi dengan model terlatih kami, sekarang bagaimana tampilannya?

```
plot_predictions(predictions=y_preds)
```



#Menyimpan dan memuat model PyTorch

Cara yang disarankan untuk menyimpan dan memuat model untuk inferensi (membuat prediksi) adalah dengan menyimpan dan memuat state_dict() model.

```
from pathlib import Path

# 1. Create models directory
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True, exist_ok=True)

# 2. Create model save path
MODEL_NAME = "01_pytorch_workflow_model_0.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME
```

```
# 3. Save the model state dict
print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model_0.state_dict(), # only saving the state_dict()
only saves the models learned parameters
           f=MODEL_SAVE_PATH)
```

```
Saving model to: models/01_pytorch_workflow_model_0.pth
```

```
# Check the saved file path
!ls -l models/01_pytorch_workflow_model_0.pth
```

```
-rw-r--r-- 1 root root 1680 Jan  2 01:20
models/01_pytorch_workflow_model_0.pth
```

#Memuat state_dict() model PyTorch yang disimpan

kami menggunakan metode fleksibel untuk menyimpan dan memuat state_dict() saja, yang pada dasarnya merupakan kamus parameter model.

```
# Instantiate a new instance of our model (this will be instantiated
with random weights)
loaded_model_0 = LinearRegressionModel()

# Load the state_dict of our saved model (this will update the new
instance of our model with trained weights)
loaded_model_0.load_state_dict(torch.load(f=MODEL_SAVE_PATH))
```

```
<All keys matched successfully>
```

Sekarang untuk menguji model yang dimuat, mari kita lakukan inferensi dengannya (membuat prediksi) pada data pengujian.

```
# 1. Put the loaded model into evaluation mode
loaded_model_0.eval()

# 2. Use the inference mode context manager to make predictions
with torch.inference_mode():
    loaded_model_preds = loaded_model_0(X_test) # perform a forward
pass on the test data with the loaded model
```

Sekarang kita telah membuat beberapa prediksi dengan model yang dimuat, mari kita lihat apakah prediksi tersebut sama dengan prediksi sebelumnya.

```
# Compare previous model predictions with loaded model predictions
(these should be the same)
y_preds == loaded_model_preds
```

```
tensor([[True],
        [True],
        [True],
```

```
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True]])
```

#Menyatukan semuanya

Kita akan mulai dengan mengimpor perpustakaan standar yang kita butuhkan.

```
# Import PyTorch and matplotlib
import torch
from torch import nn # nn contains all of PyTorch's building blocks
for neural networks
import matplotlib.pyplot as plt

# Check PyTorch version
torch.__version__
```

```
{"type":"string"}
```

Sekarang mari kita mulai membuat perangkat kode kita agnostik dengan mengatur device="cuda" jika tersedia, jika tidak maka akan default ke device="cpu".

```
# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")
```

```
Using device: cuda
```

Terakhir, kita akan menggunakan nilai X, serta nilai bobot dan bias untuk membuat y menggunakan rumus regresi linier (y = bobot * X + bias).

```
# Create weight and bias
weight = 0.7
bias = 0.3

# Create range values
start = 0
end = 1
step = 0.02

# Create X and y (features and labels)
X = torch.arange(start, end, step).unsqueeze(dim=1) # without
unsqueeze, errors will happen later on (shapes within linear layers)
y = weight * X + bias
X[:10], y[:10]
```

```
(tensor([[0.0000],
         [0.0200],
         [0.0400],
         [0.0600],
         [0.0800],
         [0.1000],
         [0.1200],
         [0.1400],
         [0.1600],
         [0.1800]]),
 tensor([[0.3000],
         [0.3140],
         [0.3280],
         [0.3420],
         [0.3560],
         [0.3700],
         [0.3840],
         [0.3980],
         [0.4120],
         [0.4260]]))
```

Sekarang kita punya beberapa data, mari kita bagi menjadi set pelatihan dan pengujian.

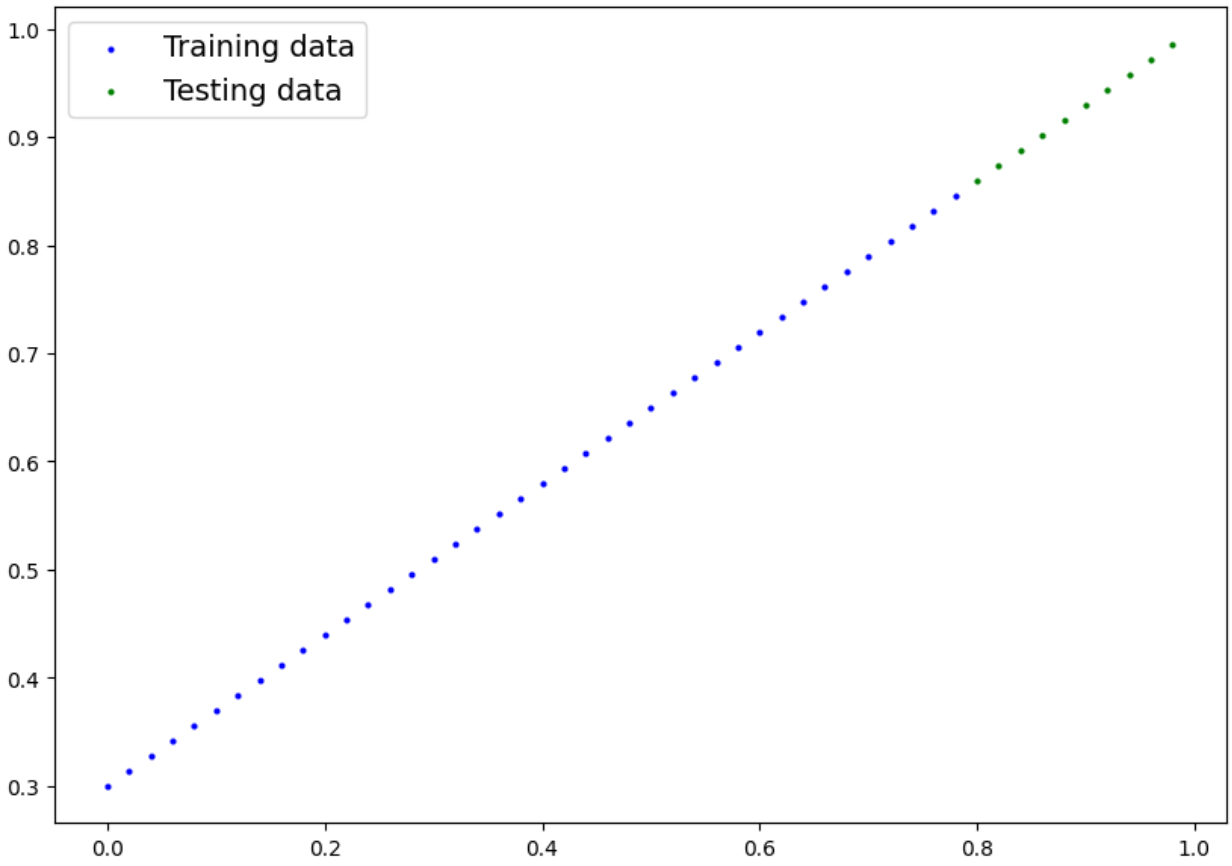
Kami akan menggunakan pemisahan 80/20 dengan 80% data pelatihan dan 20% data pengujian.

```
# Split data
train_split = int(0.8 * len(X))
X_train, y_train = X[:train_split], y[:train_split]
X_test, y_test = X[train_split:], y[train_split:]

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

```
(40, 40, 10, 10)
```

mari kita visualisasikan untuk memastikan semuanya terlihat baik-baik saja.

```
# Note: If you've reset your runtime, this function won't work,
# you'll have to rerun the cell above where it's instantiated.
plot_predictions(X_train, y_train, X_test, y_test)
```



#Membangun model linier PyTorch

```
# Subclass nn.Module to make our model
class LinearRegressionModelV2(nn.Module):
    def __init__(self):
        super().__init__()
        # Use nn.Linear() for creating the model parameters
        self.linear_layer = nn.Linear(in_features=1,
                                      out_features=1)

    # Define the forward computation (input data x flows through
nn.Linear())
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.linear_layer(x)

# Set the manual seed when creating the model (this isn't always need
but is used for demonstrative purposes, try commenting it out and
seeing what happens)
torch.manual_seed(42)
model_1 = LinearRegressionModelV2()
model_1, model_1.state_dict()
```

```
(LinearRegressionModelV2(
   (linear_layer): Linear(in_features=1, out_features=1, bias=True)
```

```
 ),
 OrderedDict([('linear_layer.weight', tensor([[0.7645]])),
              ('linear_layer.bias', tensor([0.8300]))]))
```

Kita dapat mengubah perangkat tempat objek PyTorch kita menggunakan .to(device).

Pertama mari kita periksa perangkat model saat ini.

```
# Check model device
next(model_1.parameters()).device
```

```
device(type='cpu')
```

Mari kita ubah menjadi di GPU (jika tersedia).

```
# Set model to GPU if it's availalble, otherwise it'll default to CPU
model_1.to(device) # the device variable was set above to be "cuda" if
available or "cpu" if not
next(model_1.parameters()).device
```

```
device(type='cuda', index=0)
```

#Pelatihan

Mari kita gunakan fungsi yang sama yang kita gunakan sebelumnya, nn.L1Loss() dan torch.optim.SGD().

Kita harus meneruskan parameter model baru (model.parameters()) ke pengoptimal agar dapat menyesuaikannya selama pelatihan.

```
# Create loss function
loss_fn = nn.L1Loss()

# Create optimizer
optimizer = torch.optim.SGD(params=model_1.parameters(), # optimize
newly created model's parameters
                            lr=0.01)
```

Jika Anda memerlukan pengingat tentang langkah-langkah loop pelatihan PyTorch, lihat di bawah.

```
torch.manual_seed(42)

# Set the number of epochs
epochs = 1000

# Put data on the available device
# Without this, error will happen (not all model/data on device)
X_train = X_train.to(device)
```

```
X_test = X_test.to(device)
y_train = y_train.to(device)
y_test = y_test.to(device)

for epoch in range(epochs):
    ### Training
    model_1.train() # train mode is on by default after construction

    # 1. Forward pass
    y_pred = model_1(X_train)

    # 2. Calculate loss
    loss = loss_fn(y_pred, y_train)

    # 3. Zero grad optimizer
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Step the optimizer
    optimizer.step()

    ### Testing
    model_1.eval() # put the model in evaluation mode for testing
(inference)
    # 1. Forward pass
    with torch.inference_mode():
        test_pred = model_1(X_test)

        # 2. Calculate the loss
        test_loss = loss_fn(test_pred, y_test)

    if epoch % 100 == 0:
        print(f"Epoch: {epoch} | Train loss: {loss} | Test loss:
{test_loss}")
```

```
Epoch: 0 | Train loss: 0.5551779866218567 | Test loss:
0.5739762187004089
Epoch: 100 | Train loss: 0.006215683650225401 | Test loss:
0.014086711220443249
Epoch: 200 | Train loss: 0.0012645035749301314 | Test loss:
0.013801801018416882
Epoch: 300 | Train loss: 0.0012645035749301314 | Test loss:
0.013801801018416882
Epoch: 400 | Train loss: 0.0012645035749301314 | Test loss:
0.013801801018416882
Epoch: 500 | Train loss: 0.0012645035749301314 | Test loss:
0.013801801018416882
Epoch: 600 | Train loss: 0.0012645035749301314 | Test loss:
```

```
0.013801801018416882
Epoch: 700 | Train loss: 0.0012645035749301314 | Test loss:
0.013801801018416882
Epoch: 800 | Train loss: 0.0012645035749301314 | Test loss:
0.013801801018416882
Epoch: 900 | Train loss: 0.0012645035749301314 | Test loss:
0.013801801018416882
```

Mari kita periksa parameter yang telah dipelajari model kita dan bandingkan dengan parameter asli yang kita kode keras.

```
# Find our model's learned parameters
from pprint import pprint # pprint = pretty print, see:
https://docs.python.org/3/library/pprint.html
print("The model learned the following values for weights and bias:")
pprint(model_1.state_dict())
print("\nAnd the original values for weights and bias are:")
print(f"weights: {weight}, bias: {bias}")
```

```
The model learned the following values for weights and bias:
OrderedDict([('linear_layer.weight', tensor([[0.6968]],
device='cuda:0')),
             ('linear_layer.bias', tensor([0.3025],
device='cuda:0'))])

And the original values for weights and bias are:
weights: 0.7, bias: 0.3
```

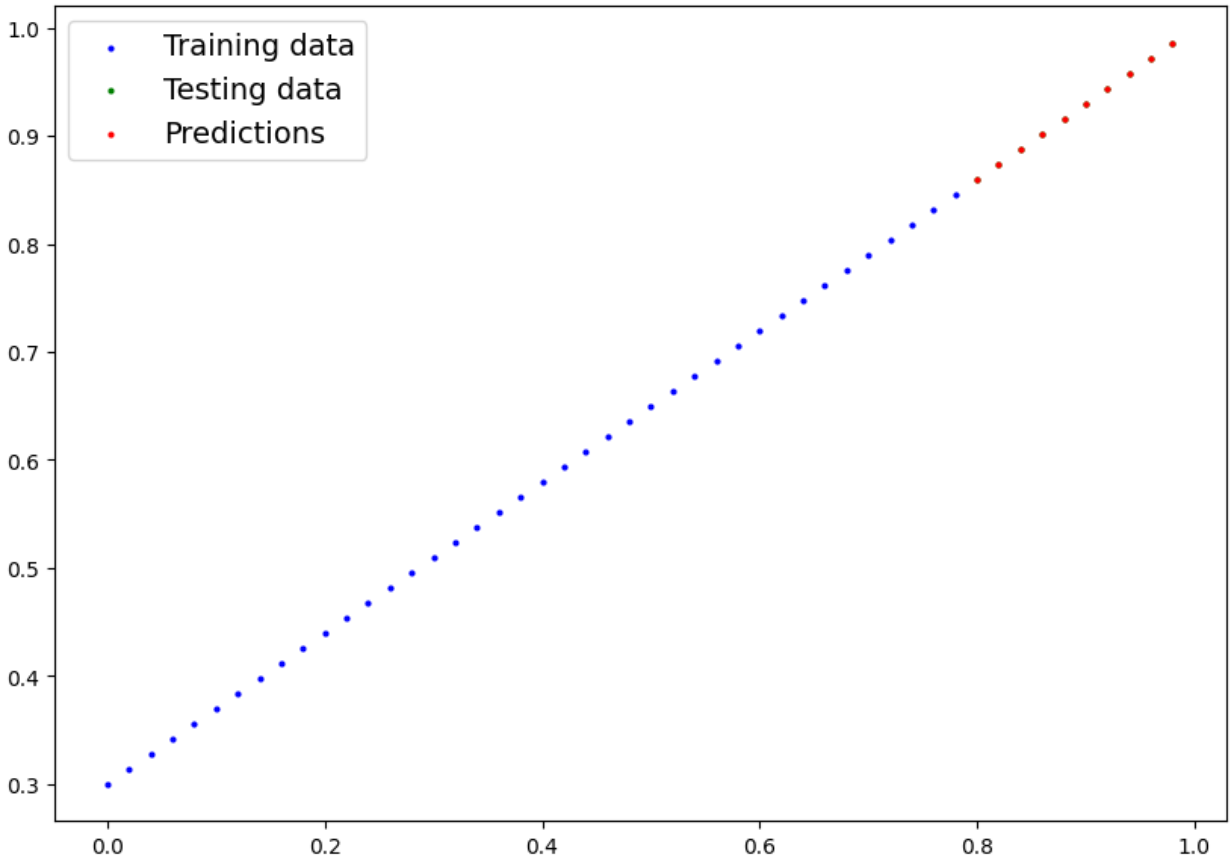
Sekarang kita memiliki model terlatih, mari aktifkan mode evaluasinya dan buat beberapa prediksi

```
# Turn model into evaluation mode
model_1.eval()

# Make predictions on the test data
with torch.inference_mode():
    y_preds = model_1(X_test)
y_preds
```

```
tensor([[0.8600],
        [0.8739],
        [0.8878],
        [0.9018],
        [0.9157],
        [0.9296],
        [0.9436],
        [0.9575],
        [0.9714],
        [0.9854]], device='cuda:0')
```

Sekarang mari kita buat prediksi model kita.

```
# plot_predictions(predictions=y_preds) # -> won't work... data not on CPU

# Put data on the CPU and plot it
plot_predictions(predictions=y_preds.cpu())
```



mari simpan ke file agar dapat digunakan nanti.

```
from pathlib import Path

# 1. Create models directory
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True, exist_ok=True)

# 2. Create model save path
MODEL_NAME = "01_pytorch_workflow_model_1.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME

# 3. Save the model state dict
print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model_1.state_dict(), # only saving the state_dict()
```

```
only saves the models learned parameters
           f=MODEL_SAVE_PATH)

Saving model to: models/01_pytorch_workflow_model_1.pth
```

Dan untuk memastikan semuanya berfungsi dengan baik, mari kita muat kembali.

```
# Instantiate a fresh instance of LinearRegressionModelV2
loaded_model_1 = LinearRegressionModelV2()

# Load model state dict
loaded_model_1.load_state_dict(torch.load(MODEL_SAVE_PATH))

# Put model to target device (if your data is on GPU, model will have
to be on GPU to make predictions)
loaded_model_1.to(device)

print(f"Loaded model:\n{loaded_model_1}")
print(f"Model on device:\n{next(loaded_model_1.parameters()).device}")

Loaded model:
LinearRegressionModelV2(
  (linear_layer): Linear(in_features=1, out_features=1, bias=True)
)
Model on device:
cuda:0
```

Sekarang kita dapat mengevaluasi model yang dimuat untuk melihat apakah prediksinya sesuai dengan prediksi yang dibuat sebelum menabung.

```
# Evaluate loaded model
loaded_model_1.eval()
with torch.inference_mode():
    loaded_model_1_preds = loaded_model_1(X_test)
y_preds == loaded_model_1_preds

tensor([[True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True]], device='cuda:0')
```