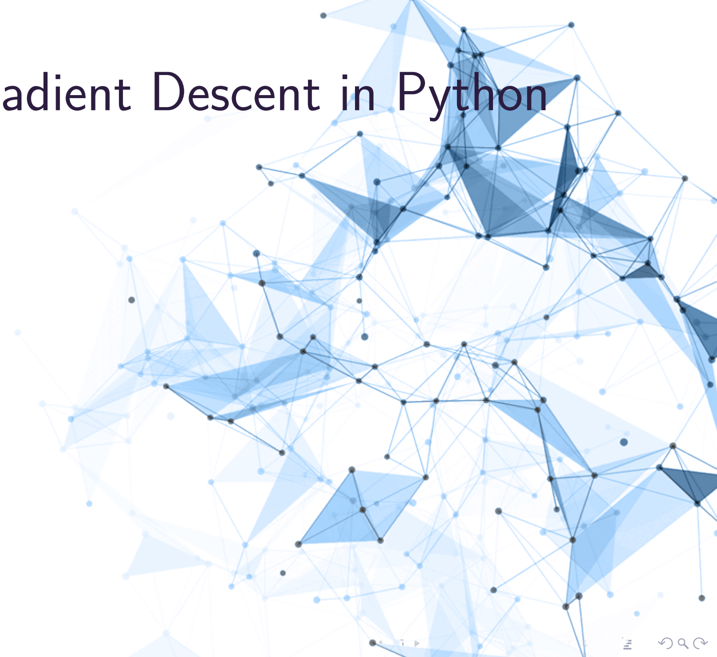
Implementation of Gradient Descent in Python



Weiguan Wang
Shanghai University

Overview

1. Automatic differentiation and gradient
2. Gradient descent to fit a linear model
3. Keras API for training networks

Automatic differentiation and gradient

The core function of Deep Learning libraries such as Tensorflow and Pytorch is arguably the automatic differentiation. The library needs to remember all operations in forward pass, and in the backward pass, it traverses all these operations to compute gradient.

Automatic differentiation and gradient

Tensorflow provides the tf.GradientTape API for automatic differentiation. As the name indicates, the "Tape" records all relevant operations inside it.

```
import tensorflow as tf
x = tf.Variable(3.0)
with tf.GradientTape() as tape:
  y = x**2
```

Automatic differentiation and gradient

Then the gradient of y with respect to x is calculated with `GradientTape.gradient(target, sources)`

```
# dy = 2x * dx
dy_dx = tape.gradient(y, x)
dy_dx.numpy()
# output is 6.0
```

Automatic differentiation and gradient

The gradient can also be computed with respect to model parameters.

```
layer = tf.keras.layers.Dense(1, activation='linear')
x = tf.constant(np.random.standard_normal(size=(100, 1)), dtype='float32')
y_true = 2. * x + 1  # assume the true model is 2*x + 1

with tf.GradientTape() as tape:
# Forward pass
    y_pred = layer(x)
    loss = tf.reduce_mean((y_true - y_pred)**2)

# Calculate gradients with respect to every trainable variable
grad = tape.gradient(loss, layer.trainable_variables)


# Output for grad:
# [<tf.Tensor: shape=(1, 1), dtype=float32, numpy=array([[-0.65564555]], dtype=float32)>,
# <tf.Tensor: shape=(1,), dtype=float32, numpy=array([-1.905393], dtype=float32)>]
"""
```

Fit a linear model with gradient descent

The estimation of a linear model has a closed-from solution that does not need gradient descent. Here we use a linear model to check if gradient descent gives the same parameters as the analytical solution.

Gradient descent to fit a linear model

This can be achieved by simply adding a training loop for gradient descent.

```
layer = tf.keras.layers.Dense(1, activation='linear')
x = tf.constant(np.random.standard_normal(size=(100, 1)), dtype='float32')
y_true = 2. * x + 1  # assume the true model is 2*x + 1
learning_rate = 0.01

for _ in range(1000):
    with tf.GradientTape() as tape:
    # Forward pass
        y_pred = layer(x)
        loss = tf.reduce_mean((y_true - y_pred)**2)

    # Calculate gradients with respect to every trainable variable
    grad = tape.gradient(loss, layer.trainable_variables)
    for i in range(2):
        layer.trainable_variables[i].assign_sub(learning_rate * grad[i])
```

Keras API

More concise syntax is available with Tensorflow.Keras API

```
# Build fully connected network by stacking the layers
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(10)
])

# Compile model by equiping it with optimizer and loss function
model.compile(optimizer='SGD', loss='mse')

# Start fitting the model
model.fit(x, y_true, epochs=1000)
```

Thanks for your attention!