Computer Vision 2: Exercise 2 (16.4.2019)

Training a neural network using Tensorflow

We train a neural network to do a simple linear regression task. Solving this task will give you an idea of how to manage a training process in Tensorflow.

Background

We are given a set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ of n training data points, where $x_i \in \mathbb{R}$ is the ith input sample, and $y_i \in \mathbb{R}$ is the ith output sample. Our objective is to learn a function $f : \mathbb{R} \to \mathbb{R}$ from $\mathcal{D}$. In linear regression, the function f is of the form

$$f(x; w, b) = wx + b, \tag{1}$$

where $w \in \mathbb{R}$ and $b \in \mathbb{R}$ are *parameters* of the function that may be learned. For a given input x the function f produces a predicted output $\hat{y}$ dependent on the parameters. The quality of the prediction can be measured by a loss function $L(w, b)$. For the linear regression problem, we use the average mean squared loss, which for dataset $\mathcal{D}$ is

$$L(w, b) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f(x_i; w, b)\right)^2 .$$

The learning problem is to find the best values of the parameters w and b, denoted w^* and b^*, that minimize the loss:

$$w^*, b^* = \operatorname*{argmin}_{w,b} L(w, b).$$

As a neural network, the linear regression (Eq. (1)) can be reprsented as a single neuron with a bias term. The term w is the input weight of the neuron, and b is the bias term of the neuron. An example is shown in Figure 1. Note that we use the representation where the bias term can be viewed as an additional weight parameter with a constant 1 input.

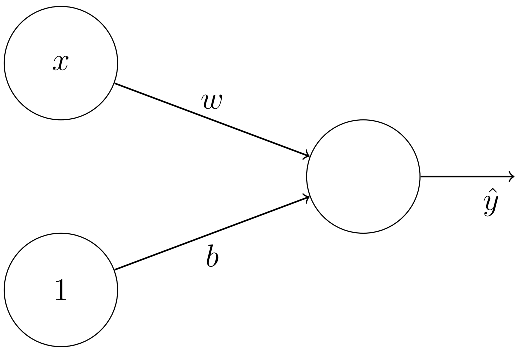

Figure 1: A neuron representing Equation (1).

We can apply the idea of gradient descent to iteratively optimize w and b. Suppose we start from some values w_0, b_0 of the parameters. We first assign some learning rate $\alpha > 0$. We calculate the partial derivative of the loss function with respect to the parameters:

$$\frac{\partial}{\partial w} L(w, b), \quad \frac{\partial}{\partial b} L(w, b).$$

The loss function *decreases fastest* if one goes from, w_0 (or b_0) in the direction of the negative partial derivative of w_0 (or of b_0). To update the parameters, at iteration $k \geq 1$ we apply

$$w_{k+1} = w_k - \alpha \left. \frac{\partial}{\partial w} L(w, b) \right|_{w_k, b_k}$$

$$b_{k+1} = b_k - \alpha \left. \frac{\partial}{\partial w} L(w, b) \right|_{w_k, b_k},$$

where the notation $\frac{\partial}{\partial w}L(w,b)\big|_{w_k,b_k}$ denotes evaluation of the partial derivative at (w_k, b_k).

Linear regression in Tensorflow

A `tf.Variable` is a tensor that both retains its value over multiples `session.run()` calls and can be updated. It is used for bookkeeping variables, and most importantly to store learnable parameters of a machine learning model. We use `tf.Variable` to represent the two parameters w and b of our linear regression model, and learn their values.

- Download the two data files `linreg_x.npy` and `linreg_y.npy`, which contain the training data samples $\mathcal{D}$. The arrays are of shape $(n, 1)$.
- Set up a computational graph with the following specifications:
  - Placeholders for the inputs x and targets y, with shape $(n, 1)$.
  - Create a `tf.Variable` for w and b both. Set their initial values to a value drawn randomly from a normal distribution $N(0, 1)$.
  - Define an output as `y_predicted = w*x + b`.
- Set up a training operation by following these steps:
  - Define a loss function using `tf.losses.mean_squared_error`. The loss should compare `y_predicted` and the target placeholder `y`.
  - Define a gradient descent optimizer with learning rate $\alpha = 0.1$. Use the function `tf.train.GradientDescentOptimizer`.
  - Define a minimizer operation by calling `minimize_op = optimer.minimize(loss)` where `optimizer` is your gradient descent optimizer, and `loss` is your loss function.
- After you have finished these steps, start a `tf.Session`, and use a `FileWriter` to write the computational graph. Examine the output using TensorBoard and see that you can localize all the parts you added.
- When you are done, add the following code block inside the `tf.Session` block and then run the code to start training:

```
1 sess.run(tf.global_variables_initializer())
2 for k in range(100):
3   _, l, wk, bk = sess.run([minimize_op, loss, w, b], {xp: x, yp: y})
4   print('Iteration {:d}: Loss {:f}, w = {:f}, b = {:f}'.format(k, l,wk,bk))
```

  Explanation of the lines:

  1. To use any variables in a computational graph, they must be initialized. This line calls a global variables initializer which will assign a value to all variables.
  2. Start a training loop with 100 iterations...
  3. On each iteration, call out minimizer operation to update the parameters. Also evaluate the values of the loss, and the variables w and b. Feed in our data x and y through the placeholders `xp` and `yp`, respectively. We ignore the output value of `minimize_op` by saving it to `_`.
  4. Display formatted output about the training progress.

Finally, to visualize the result using the parameters from the last training step add:

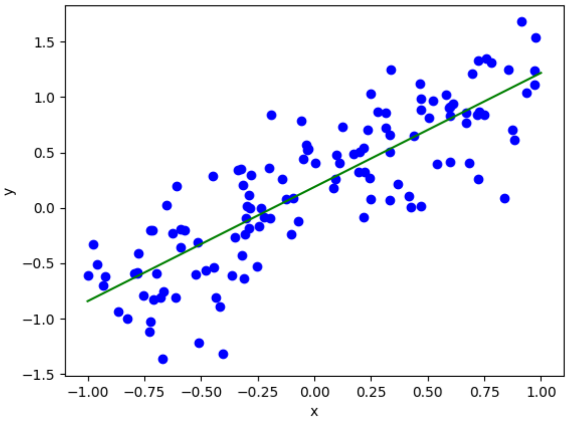

Figure 2: Linear regression result.

```python
import matplotlib.pyplot as plt
xs = np.linspace(-1.0, 1.0, num=20)
ys = wk*xs + bk
plt.plot(x, y, 'bo')
plt.plot(xs,ys,'g')
plt.ylabel('y')
plt.xlabel('x')
plt.show()
```

The result should look similar to Figure 2.

Monitoring training through Tensorboard

Later when the network you train becomes more complicated, it is no longer feasible to monitor everything through the terminal. We add a simple monitoring and logging, so we can view training progress through Tensorboard.

- After the definition of your computational graph, but before the `tf.Session` block, add a summary that tracks the loss by adding a line:

  ```python
  l_summary = tf.summary.scalar(name="loss", tensor=loss)
  ```

  This will create a summary with a name "loss" that stores values of the tensor `loss`.

- Create a similar summary for the variables `w` and `b`.

- Inside your training loop over `k`, add a line as follows that evaluates all summaries:

  ```python
  ls, ws, bs = sess.run([l_summary, w_summary, b_summary], {xp: x,
      yp: y})
  ```

  Additionally, add a line for each summary that writes the summary to your Event file:

  ```python
  writer.add_summary(ls, global_step=k) # writes loss summary
  ```

  Here, `writer` refers to your `FileWriter` object. Replicate the above for the summaries for `w` and `b`.

- Run your code, and open Tensorboard to visualize the summaries and graph.