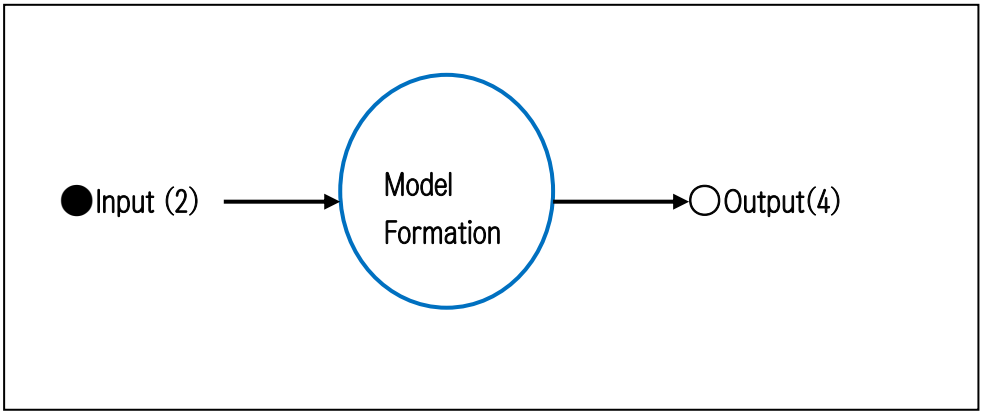
Episode 33 (AI Programming)

Deep Learning Programming

Tanuki! Before we get into programming, let's review what we discussed in Episode 32.
As shown in the figure below, the computer forms a model on its own to get from the input value (2) to the output value (4). The model formed may be additive, multiplicative, or a completely different method. This formed model is what is called a black box. Naturally, it is not always possible to create a model that approaches the correct answer from a single set of data [2→4]. Therefore, it is necessary to train the model using many different patterns of data such as [2→4], [3→6], and [4→8] ・・・・・. By learning, a robust model will be formed. This is what it means to utilize and learn from big data on the Internet. After the model has been formed, if we set an unlearned input value of, say, [8] and obtain an output value close to 16, such as 15.98654, then we have successfully programmed deep learning to create the model.



Is the algorithm of the model made a black box? If it's impossible to know the algorithm of the model, fox, tell me how to make it learn deep learning!

Tanuki, you got it. Explain the following diagram.

Deep learning training is initially given a random (meaning appropriate) value, the predicted value (2.6: manual is also possible), as shown in the figure below.

Then calculate the loss.

loss ＝ ２．６ (Predicted value) － ４ (correct)

The next value is chosen (by the program) by the function to reduce the loss (in absolute value).

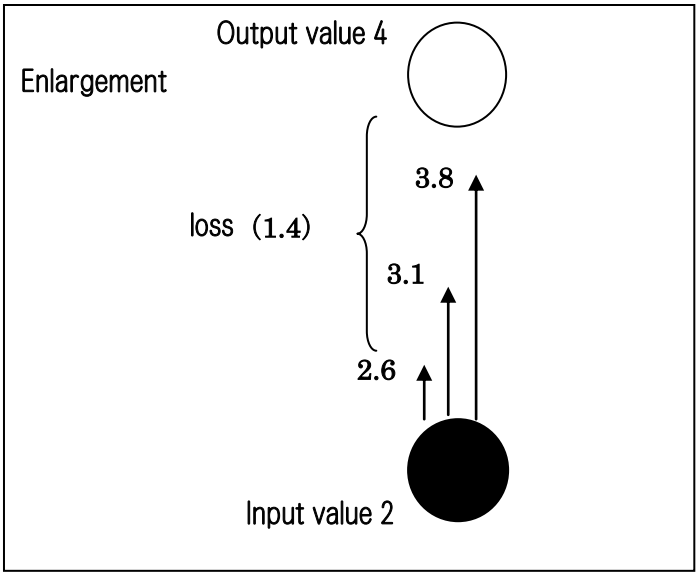
loss ＝ ３．１ (Predicted value) － ４ (correct)

The same is true for the following values

loss ＝ ３．８ (Predicted value) － ４ (correct)

This is how it will be. The function used to reduce this loss is called the optimization function or activation function. There is a wide variety of optimization functions, including least squares and stochastic gradient descent. It is the job of the deep learning programmer to select the best optimization function for learning and to form models for individual tasks. The programmer must be familiar with the business and also have the mathematical skills to be able to choose the optimization function.

So, what happens when we put the explanation so far into the form of a deep learning program and run it?



Kitsune, I'm starting to get the picture, but who created the black box algorithm?

It was first developed by Google (now Alphabet) in the U.S., famous for cat image recognition and Alpha Go. That Google provides the framework (which can be thought of as a library) called Tensorflow, which it utilized for image recognition, as open source. Many Japanese deep learning programmers are using it. You can also use it to get an overview of deep learning.

Now, let's try an example deep learning program to solve a "simple example problem" based on the explanation of deep learning so far (how to make it learn, save the learning, and recall the saved learning). The following program is written in the Python language with an embedded framework called Tensorflow.

First, an overview of deep learning programming.

It is important to get a general idea of deep learning programming.

This is true for programs for image recognition as well.

(Summary)

- Set of input and output (correct) data
  (e.g.) Input: Load multiple cat images.
  Output (correct): Cat
- Data optimization (learning), i.e., model creation
  (e.g.) In the case of an image, optimization is performed on a pixel-by-pixel basis with minimal loss.
- New data is entered and checked against the expected output (correct answer) (expressed as % probability of being close to the correct answer).
  (e.g.) Input still images of various animals and display the probability that the image is of a cat as a percentage.

The actual program is shown on the next page.

Example of "Simple Example" program: file (keisan.py)

```
import tensorflow as tf
import numpy as np

input_dim=1
output_dim=1

x = tf.placeholder("float",[None, input_dim])
w = tf.Variable(tf.random_uniform([input_dim, output_dim], -1.0, 1.0))
b = tf.Variable(tf.zeros([output_dim]))
y = w * x + b

t = tf.placeholder("float",[None, output_dim])

loss = tf.reduce_mean(tf.square(y - t))

train_step = tf.train.MomentumOptimizer(0.01,0.97).minimize(loss)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for i in range(100):
    batch_xs = np.array([[2.],[3.],[4.],[5.]])
    batch_ys = np.array([[4.],[6.],[8.],[10.]])

    sess.run(train_step, feed_dict={x: batch_xs, t:batch_ys})
    print(i,sess.run(y, feed_dict={x: batch_xs, t:batch_ys}))

print("gakusyugo-yosokuchi")

print(sess.run(w) * 8 + sess.run(b))
```

[Explanation of the example program "keisan.py"].

To begin, I would like you to look at ①,②,③... as a general section within "keisan.py".

(1) Import of frameworks (sensorflow, numpy). Use "as tf" and "as np" abbreviations for sensorflow and numpy because of their long characters.

(2) Simple variable specification and numeric assignment. The 1 in the input and 1 in the output mean one dimension that is then used as a parameter.

(3) The most important part describing the learning process. This is where the optimization and activation functions are described. What functions are used for learning is a matter of skill for programmers who are good at mathematics.

. x is the input data to be trained (2, 3, 4, 5). The .placeholder() method is closely related to the feed_dict in ⑤.

· t is the output (correct) data for x (4, 6, 8, 10).
The placeholder() method is closely related to the feed_dict in ⑤.

• w is the weight, and the initial value is a random number with an appropriate value.
For this problem, it can be thought of as the slope of a straight line.

• b is the bias. In this problem, the intercept with the vertical axis.

• The expected value of y is determined from w and b. It is a linear equation.

• LOSS is the smallest average value taken by the least-squares method. This method is often used in the determination of earthquake epicenters.

• The train_step encompasses all of the formulas in ③ and is the most important part of the optimization function.
It is optimized by an optimization class called MomentumOptimizer() (Optimizer by momentum method).
There is another optimization class called GradientDescentOptimizer().
The specific method of the processing content of this function is in the realm of mathematics.

(4) Spell (?) : These three lines follow the Tensorflow mechanism.
Tensorflow follows the rule of creating a session, initializing variables in the session, and executing them. Otherwise, an error will occur, so these three lines are a must.

(5) Input values (2, 3, 4, 5) are set as mini-batch batch_xs (array), and output values (4, 6, 8, 10), which are correct answers, are set in batch_ys and trained 100 times. The results are shown in the execution results below.
The result is [3.90966535],[5.92648077],[7.94329643],[9.96011162]) at the 100th run (99), indicating that the correct answer (4, 6, 8, 10) is approaching.

(6) After learning 100 times, an unknown value (8) is set and tested to see if it can predict 16. The result was [16.01055908]. I have to say, well done.

Tanuki! The next step is to show the results of running this program.
Let's let it learn 100 times and see how it approaches the correct answers [4, 6, 8, 10]! Since there are so many times, we omit the 3rd through the 98th times.
However, as you can see from the 99th study, the more times you study, the closer you get to the correct answer, but on the other hand, you may diverge and move away from the correct answer. It seems that there is an appropriate number of times to study.

Execution Result

```
Ubun@14-32:~$ python keisan.py          execution (e.g. program)
(0, array([[ 0.78011608],     1st trial          correct  4
        [ 1.08909798],                           correct  6
        [ 1.39807975],                           correct  8
        [ 1.70706165]], dtype=float32))          correct  10
(1,  array([[ 3.15634394],      Second trial
        [ 4.51723194],
        [ 5.87811947],
        [ 7.239007  ]], dtype=float32))
  :
(98,  array([[  4.49484825],     Trial 99 th
        [  6.77231359],
        [  9.04977989],
        [ 11.32724571]],  dtype=float32))
(99, array([[ 3.90966535],      100th trial        correct  4
        [ 5.92648077],                           correct  6
        [ 7.94329643],                           correct  8
        [ 9.96011162]], dtype=float32))          correct  10
```

The program sets a new value of "8" at the end to test whether it has been learned correctly.
I'm not sure if doubling it to 16 is the correct answer.
The result is shown on the right.
Isn't it a beautiful result?

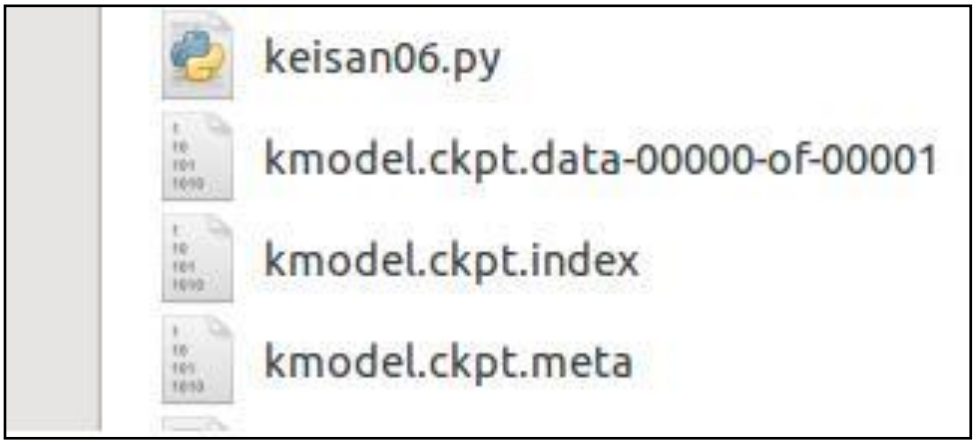
```
gakusyugo-yosokuchi
   After learning: 2 times the predicted value of 8

[[ 16.01055908]]              correct  16
```

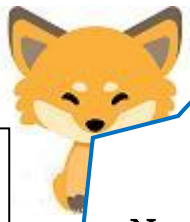
Tanuki! You said before that if you can pass the parameters of the regression line (curve) to the AI app after it has been trained, then you can run the AI app on your phone without the need for further training.
That's what we're going to talk about now.
The first step is to save the parameters after the study. In order to save them, the program must be slightly edited as follows. The edited program is named "keihozon.py".

[Improvements were made to the model saving program.]「keihozon.py」

```
    :   Up to this point, do the same as ①,②,③, and ④ in "keisan.py".
    :
hozon = tf.train.Saver()    #  Addition (Definition)
for i in range(100):
    batch_xs = np.array([[2.],[3.··
    batch_ys = np.array([[4.],[6.··
    sess.run(train_step, feed_dic··     # No change
    print(i,sess.run(y, feed_dict··
hozon.save(sess,"kmodel.ckpt")     #   Add (Save)
```

When "keihozon.py" is executed, three files (kmodel.ckpt~), which are trained models, are created in the current directory (the location of keihozon.py) as shown below. These files are in binary format, so you cannot look inside them. They are truly a black box.





Now edit the program to read the "saved training model" in the opposite direction. This means that the three files (kmodel.ckpt ~) in the above figure are to be read. The name of the edited program is "keiread.py". If the loaded training model is not available, there is no point in saving it. In addition, IT compan ies that provide AI cannot make a profit unless they can sell their own trained models in the same framework. So, of course, they need to test if it is usable after loading. The test was to set a value of "9" and see how close it came to the result of "18". The result is shown on the left.

Program: Example of "keiread.py

1

2

3

4 of "keisan.py" are used as they are.

Delete 5 in the learning section and add the following two lines in the learning model loading.

```
hozon = tf.train.Saver()
hozon.restore(sess,"kmodel.ckpt")
```

Edit 6 as follows.

```
print("gakusyugo-yosokuchi")
print(sess.run(w) * 9 + sess.run(b))
```

Tanuki, I'll explain the program: "keiread.py" a little more.
Since ① is importing sensorflow, ② is defining variables, ③ is how to make it learn, and ④ is allocating a session area specific to sensorflow, it is necessary in its original form. Since the training model is imported, the training part ⑤ is no longer necessary. Since ⑥ is the part to check if the value doubles by giving a new value, a new value of "9" is given and checked if it approaches "18".

As you can guess from the results, the defined variable names (w, b) and the last obtained values of w (weight) and b (bias) by learning are stored in the session area, and they are saved as a training model in the "kmodel.ckpt" file by the "save" method. The results were as follows.

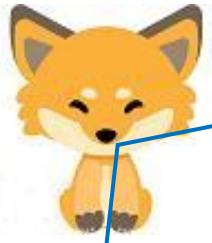
[Execution Result]

```
Ubun@14-32:~$ python keiread.py  ←  execution (e.g. program)

gakusyugo-yosokuchi

[[ 18.0175972]]  ←  Learning is passed on and values close to double of 9 are
                    displayed.

Ubun@14-32:~$
```

That's great, Kitsune!
You explained it in three parts: keisan.py, a program to train deep learning, keihozon.py, a program to save parameters, and keiread.py, a program to read and run the saved parameters, so I understand Ira AI.
Now I know better that I have to learn the Python language well.

Tanuki, I'll give you an overview of the Python language.
Tensorflow, a deep learning framework from Google, is available in C++ and Python. However, some people give up studying deep learning just by hearing the word "C++," which is a compiler language, and think it is too complicated. But don't worry. You can learn it in Python, an interpreter language. In fact, it is better to think of Tensorflow as being provided for Python.
Guido Van Rossum, the Dutch mathematician and programmer who developed Python, apparently also worked at Google. Furthermore, there is a version of Python called iPython that can be entered while checking for errors at each instruction.

Initially, to understand deep learning programs, it is necessary to study the Python language, but understanding Python alone does not lead to an understanding of deep learning. As mentioned above, a change of mindset is necessary.
Python is capable of performing matrix calculations and other mathematical operations taught in high school using only the standard library, but by incorporating various packages as a framework, it can be treated as a language for advanced scientific and technical calculations. The advantage is that the results of the calculations can be immediately graphed and visually confirmed simply by setting the parameters for the vertical and horizontal axes.
Other third-party packages can be installed to develop network-related programming, such as programs that specify still and moving images to be collected, pick them up from the Internet and use them as big data, and web applications (linking with HTML and creating server-side scripts).
However, in my opinion, in order to understand Python, it is necessary to have a solid foundation of language learning such as assembler, C, and Java. This is because you will encounter constructors, instances, and objects, which are terms used in object-oriented languages. Arrays (= matrices) are also used, including lists and many other types of arrays.



Next, I would like to share my thoughts on deep learning.
Deep learning does not = AI. Deep learning is a method of building AI. However, it is now regarded as the most powerful and effective method.
The simple deep learning program "keisan.py" used here is programming with one neuron layer. Increasing the number of neuron layers (synaptic connections in human terms) to two or three means creating multiple parts of "keisan.py" ③. For example, in "keisan.py", if the number of neuron layers is increased to two, two weights (w) and two biases (b) will be created: w1 and w2, and b1 and b2. It is natural to say that the output expected value (y) is the input value to the next neuron layer. If we use the same optimization and activation functions as in the previous layer, the rest of the script will be the same. However, it is not true to say that more layers will produce more accurate evaluation values. In many cases, they diverge. As we understand deep learning, we realize that it is impossible to predict gambling games such as numbers even if we use the input values and output values (results) of the moment as data. Unless, of course, there is some kind of trend in the gambling object. If not, the predicted values will diverge. Similarly, in the case of predicting stock prices, AI (e.g., robo-advisors) will be powerful in predicting stock prices if they follow a trend, but will be unable to predict catastrophic events such as the Lehman Shock and will lose a lot of money. But catastrophic events are unpredictable even to humans, so it may be more profitable to rely on robo-advisors that have learned various patterns (input & output values). Neural networks include a form that simply increases the number of neuron layers, a recurrent neural network that returns the output value in the middle to the previous value and uses it as the input value, and a convolutional neural network that divides a still image (input data) used in image recognition into smaller parts (called kernels) in pixel units in order to obtain highly accurate evaluation values.
However, no matter what kind of neural network algorithm is used, I think that ③ in "keisan.py", which has only about 20 lines, is the basis of the idea of deep learning.