05.1 CHAPTER 5: CACCULATING NEURAL NETWORW WITH LOSS

- With a randomly initialized model, we train the model over time. To train model we tweak weight and bias to improve model accuracy & confidence.

- Loss function —
- We calculate how much the error the model has.
- Loss function also referred as the cost function, is the algorithm that quantifies how wrong the model is.
- Since loss is a metric, model's error we ideally want it to be 0.

Categorical cross entropy Loss —
- Categorical cross entropy is explicitly used to compare a "ground-truth" / reality probability (y or "targets") and model predicted distribution ($\hat{y}$ or predictions).

Categorical cross entropy of Y (actual) and $\hat{y}$ (predicted) is $\boxed{L_i = -\sum y \times \log(\hat{y})}$

where L denotes Loss value, y represent actual and $\hat{y}$ represent predicted.

- In our case, we have a classification model that returns a probability distribution over all the outputs. Cross entropy compares two probability distributions.
So that is why it is called Categorical cross entropy loss.

$\underset{\uparrow \text{ Target Variable}}{\text{Categorical}}$ $\underbrace{\text{cross entropy}}_{\text{compare actual predicted}}$ $\underset{\uparrow \text{ Calculate loss}}{\text{loss}}$

Example — Softmax output of $[0.7, 0.1, 0.2]$ and target $[1, 0, 0]$ sugges 3 target class and softmax has output probability.

$$L_1 = -\sum y \times \log(\hat{y}) = -(1 \times \log(0.7) + (0 \times \log(0.1)) + (0 \times \log(0.2)))$$
$$= -(-0.35 + 0 + 0) = 0.35$$

In python, import math   softmax-output = [0.7, 0.1, 0.2]   target = [1,0,0]

```
loss = -(math.log(softmax-output[0]) * target[0] + math.log(softmax-output[1])
        * target[1] + math.log(softmax-output[2]) * target[2]

print(loss)  # 0.35
```

So we can say when loss = 0, model is perfect. Indirectly probability should be 1.

print (math.log $\underbrace{(\text{softmax-output}) \times \cancel{\log}\ \text{target}}_{\text{It should be 1, so that output is 0}}$) = math.log (1) = 0. so here loss = 0.

So what is log?

- Log is short for logarithim and is defined as the solution for x term in an equation of the form $a^x = b$.

  For example, $10^x = 100$ can be solved by log : $\log_{10}(100) = 2$.

  This log property is extremely useful when e (Euler number or ~2.71828) is used in base.

- Logarithim with e as its base is referred to natural logarithim, natural log or simply log.

  Can be written as $\ln(x) = \log(x) = \log_e(x)$

  So something like $e^x = b$ for example $e^x = 5.2$ is solved by $\log(5.2)$.

```
import numpy as np.
b = 5.2
print (np.log(b)) # 1.648.
```

```
import math.
print (math.e ** 1.648)
# 5.19999 ≃ 5.2   so both are same.
```

- Consider a scenario with a neural network that perform classification between three classes. and neural network classifies in batch of three. After running softmax activation function with a batch of 3 samples and 3 classes (dog, cat, humans)

```
softmax_output = np.array([[0.7, 0.1, 0.2],
                           [0.1, 0.5, 0.4],
                           [0.02, 0.9, 0.08]])

class_targets = [0,1,1]   # dog, cat, cat
```

In class_targets, first value is 0 means softmax output distribution intended was 0th index of [0.7, 0.1, 0.2], the model has 0.7 confidence that this observation is dog.

For second, second value is 1, so 1st index of [0.1, 0.5, 0.4], so model has 0.5 confidence it is cat.

Third, same as second.

dog	cat	humans
0.7	0.1	0.2
0.1	0.5	0.4
0.02	0.9	0.08

```
for targ_ind, distribution in zip (class_targets, softmax_outputs):
    print (distribution [targ_ind])     #: 0.7, 0.5, 0.9
```

Othe way to write is

```
~~pr~~ print (softmax_outputs [range (len (softmax_outputs)), class_targets])
# [0.7 0.5 0.9], same output return.
```

05·3 Finally, we want to average loss per batch to have an idea how our model is doing after training.

arithmetic mean : sum(iterable) / len(iterable)

So in python, neg_log = -np.log (softmax_output [range(len (softmax outputs)), class_targets.

average_loss = np.mean (neg_log)

print (average_loss) # 0.385

One hot encoded

- One hot encoded is where all the values except for one, are zeroes and the correct label's position is filled with 1.

Last example softmax_output	→	One hot encoded
[0.7, 0.1, 0.2]		[1, 0, 0]
[0.1, 0.5, 0.4]		[0 1 0]
[0.02, 0.9, 0.08]		[0 1 0]

For log(0) it will give error. i.e, -np.log(0) will give error. Normally, log(0) is undefined. $y = \log(x)$ then $e^y = x$ in this case $y = \log(0)$ is same as $e^y = 0$. e raised to any power is positive number and there is no y resulting $e^y = 0$. This means log(0) is undefined and equal to very big number. so -np.log(0) = inf #infinite.

then np.exx(-np.inf) = 0.0

So we could add a very small value to the confidence to prevent it from being zero.

For example, 1e-7 -np.log (1e-7) # 16.118

Adding a very small value, to the confidence at its far edge will insignificant impact the result, but leads to 2 additional issues.

First is when Confidence Value is 1 → -np.log (1 + 1e-7) # $-9.999e^{-8} \simeq 0$

When model is fully correct in prediction and predicted all correct label, 1. but negative value ~~(1e-17)~~ (1e-7) is shifting its confidence even if it is very small. Ideal in this case should be 0.

-np.log (1 - 1e-7) # 1.0000494e-7 ≃ 0

So this will prevent being exactly 0 making it very small value

Therefore we will clip values from both sides by same number, 1e-7 in our case.

y_pred_clipped = np.clip (y_pred, $1e^{-7}$, $1-1e^{-7}$)

Categorical Cross Entropy Loss Class

```
#Common Loss class
class Loss:
    # Calculate the data and regularization losses, model output and reality.
    def calculate (self, output, y):
        # Calculate sample loss
        sample_losses = self.forward (output, y)
        #Calculate mean loss
        data_loss = np.mean (sample_losses)
        return data_loss
```

Since we implemented both probability and one-hot we need to first check whether its probability or one-hot.

So if targets are single dimensional (like a list) it is probability and if it is of 2 dimensions then it is one-hot. 2 means list of list.

Sample –

```
softmax_output = np.array([[0.7, 0.1, 0.2],
                           [0.1, 0.5, 0.4],
                           [0.02, 0.9, 0.08]])

class_targets = np.array([[1, 0, 0],
                          [0, 1, 0],
                          [0, 1, 0]])

# for probability
if len(class_targets.shape) == 1:
    correct_confidences = softmax_outputs[range(len(softmax_output)), class_target]
# if One-hot
elif len(class_targets.shape) == 2:
correct_confidences = np.sum(softmax_outputs * class_targets, axis=1)

# Losses
neg_log = -np.log(correct_confidences)
average_loss = np.mean(neg_log)
```

when ==2, we multiply confidence by target, zeroing out all values except the one at correct labels, performing a sum along row axis (axis 1).

Accuracy Calculation →

- While loss is a useful metric for optimizing model, Accuracy is also a good measure.
- Accuracy often describe how often the largest confidence is correct in terms of fraction.

Model output	Actual	Accuracy
[0.7, 0.2, 0.1],	[0,	✓ [0.7 (pred) = 0.7 Actual]
[0.5, 0.1, 0.4]	1	× [0.5 (pred) = 0.1 Actual]
[0.02, 0.9, 0.08]	1]	✓ [0.9 (pred) = 0.9 Actual]

~~Loss =~~

= 2/3 = 0.66, Accuracy = 0.66

~~Loss = 1 - 0.66 = 0.34~~