

Metrics

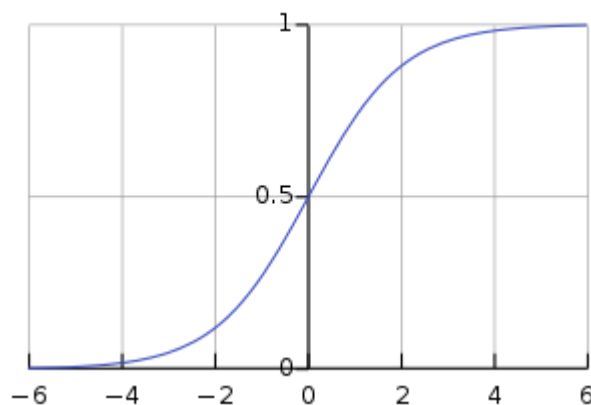
Discover the most commonly used metrics for evaluating a neural network.

Chapter Goals:

- Convert logits to probabilities
- Obtain model predictions from probabilities
- Calculate prediction accuracy

A. Sigmoid

As discussed in the previous chapter, our model outputs logits based on the input data. These logits represent real number mappings from probabilities. Therefore, if we had the inverse mapping we could obtain the original probabilities. Luckily, we have exactly that, in the form of the [sigmoid function](#).



The above plot shows a sigmoid function graph. The x-axis represents logits, while the y-axis represents probabilities.

Note the horizontal asymptotes at $y = 0$ and $y = 1$.

Using the sigmoid function (`tf.nn.sigmoid` in TensorFlow), we can extract probabilities from the logits. In binary classification, i.e. `output_size = 1`, this

represents the probability the model gives to the input data being labeled 1. A probability closer to 0 or 1 means the model is pretty sure in its prediction, while a probability closer to 0.5 means the model is unsure (0.5 is equivalent to a random guess of the label's value).

B. Predictions and accuracy

A probability closer to 1 means the model is more sure that the label is 1, while a probability closer to 0 means the model is more sure that the label is 0. Therefore, we can obtain model predictions just by rounding each probability to the nearest integer, which would be 0 or 1. Then our prediction accuracy would be the number of correct predictions divided by the number of labels.

In TensorFlow, there is a function called `tf.reduce_mean` which produces the overall mean of a tensor's numeric values. We can use this to calculate prediction accuracy as the mean number of correct predictions across all input data points.

We use these metrics to evaluate how good our model is both during and after training. This way we can experiment with different computation graphs, such as different numbers of neurons or layers, and find which structure works best for our model.

Time to Code!

The coding exercise for this chapter focuses on obtaining predictions and accuracy based on model logits.

In the backend, we've loaded the `logits` tensor from the previous chapter's `model_layers` function. We'll now obtain probabilities from the `logits` using the sigmoid function.

Set `probs` equal to `tf.nn.sigmoid` applied to `logits`.

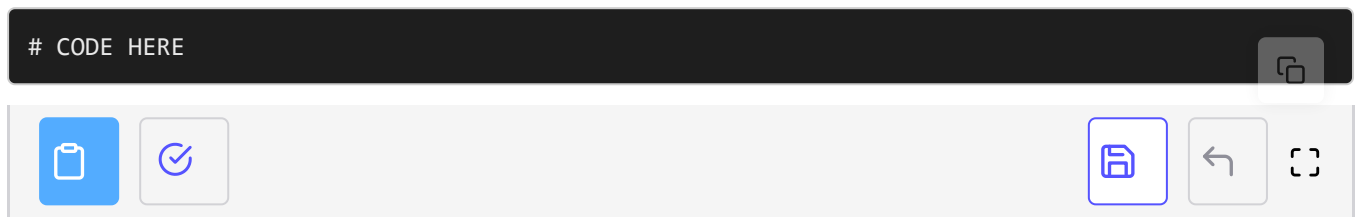


We can calculate label predictions by rounding each probability to the nearest

integer (0 or 1). We'll use `tf.round` to first round the probabilities to 0.0 or 1.0.

Set `rounded_probs` equal to `tf.round` with `probs` as the lone argument.

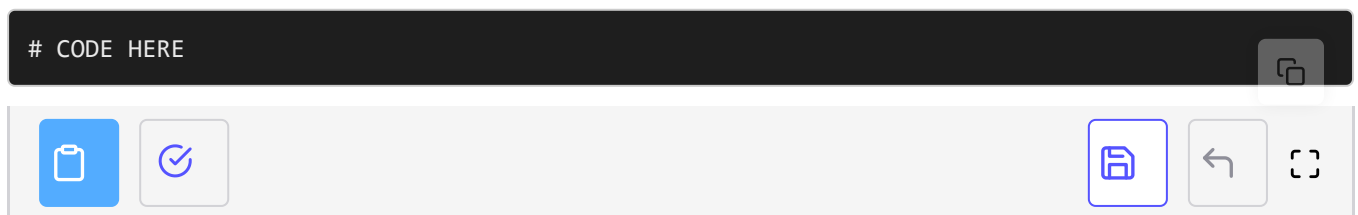
```
# CODE HERE
```



The problem with `rounded_probs` is that it's still type `tf.float32`, which doesn't match the type of the `labels` placeholder. This is an issue, since we need the types to match to compare the two. We can fix this problem using `tf.cast`.

Set `predictions` equal to `tf.cast` with `rounded_probs` as the first argument and data type `tf.int32` as the second argument.

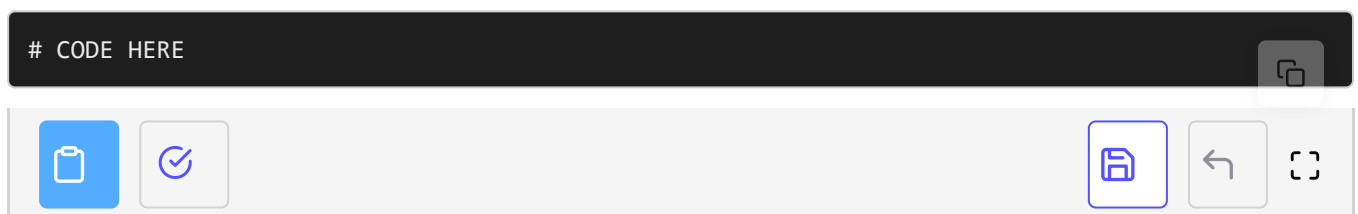
```
# CODE HERE
```



The final metric we want is the accuracy of our predictions. We'll directly compare `predictions` to `labels` by using `tf.equal`, which returns a tensor that has `True` at each position where our prediction matches the label and `False` elsewhere. Let's call this tensor `is_correct`.

Set `is_correct` equal to `tf.equal` with `predictions` as the first argument and `labels` as the second argument.

```
# CODE HERE
```



The neat thing about `is_correct` is that the number of `True` values divided by the number of total values in the tensor gives us our accuracy. We can use `tf.reduce_mean` to do this calculation. We just need to cast `is_correct` to type `tf.float32`, which converts `True` to 1.0 and `False` to 0.0.

Set `is_correct_float` equal to `tf.cast` with `is_correct` as the first argument and data type `tf.float32` as the second argument.

Set `accuracy` equal to `tf.reduce_mean` applied to `is_correct_float`.

```
# CODE HERE
```

