Chengwei Zhang Follow
Programmer and maker. Love to write deep learning articles.| Website: DLology.com | Check my Dev profile: https://sourcerer.io/Tony607
Feb 10 · 5 min read

How to Multi-task learning with missing labels in Keras

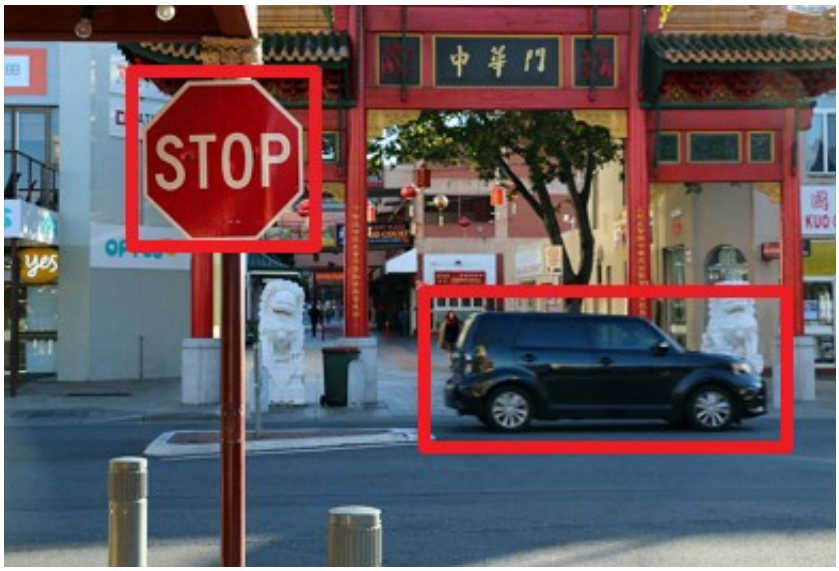
(Comments)

Multi-task learning enables us to train a model to simultaneously do several tasks.

For example, given a photo was taken by a self-driving car, we want to detect different things in the image. **Stop sign, traffic lights, cars** etc.

Without multi-task learning, we have to train model for each object we want to detect and with one output either the target object is detected or not.

But with multi-task learning, we can have one model trained only once to detect if any of the target objects are detected by having 3 output labels.



The model input is an image and the output has 3 labels, with 1 meaning a specific object is detected.

$$y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \begin{matrix} \text{Stop sign} \\ \text{traffic lights} \\ \text{cars} \end{matrix}$$

For model trained on dataset like images, training one model to do multiple tasks performs better than models trained separately to detect objects separately since lower level images features learned during training could be shared between all objects types.

Another benefit of multi-tasking learning is it allows the training data output to be partially labeled. Let's say instead of labeling previous 3 objects, we want the human labeler to labels 3 additional different objects in all given images, pedestrians, cyclists, roadblocks. He/She

may eventually get tired and didn't bother to label whether or not there's a stop sign or whether or not there's a roadblock.

So the labeled training output could look like this, where we indicate unlabeled as **"-1"**.

$$Y = \begin{bmatrix} 1 & 1 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & -1 & 0 & 1 \end{bmatrix} \begin{matrix} \text{Stop sign} \\ \text{traffic lights} \\ \text{cars} \\ \text{pedestrians} \\ \text{cyclists} \\ \text{roadblocks} \end{matrix}$$

So how can we train our model with the dataset like this?

The key is the loss function we want to **"mask"** labeled data. Meaning for unlabeled output, we don't consider when computing of the loss function.

Multi-task learning Demo

Let's walk through a concrete example to train a Keras model that can do multi-tasking. For demo purpose, we build our own toy datasets since it is simple to train and visualize the result.

```
N =100000
X = np.random.rand(N,2)
# place holder for Y
Y = np.ones((N,3))
Y[:,0] = X[:, 1] <= 0.5
Y[:,1] = X[:, 0] >= 0.5
Y[:,2] = X[:, 0] + X[:, 1] > 1
```

Here we randomly generate 100,000 data points in 2D space. Each axis is in the range between 0 to 1.

For the output Y, we have 3 labels in the following logic

$$y_0 = \begin{cases} 1, x_1 \leq 0.5 \\ 0, x_1 > 0.5 \end{cases}$$

$$y_1 = \begin{cases} 1, x_0 \geq 0{,}5 \\ 0, x_0 < 0.5 \end{cases}$$

$$y_2 = \begin{cases} 1, x_0 + x_1 > 1 \\ 0, x_0 + x_1 \leq 1 \end{cases}$$

We will build a model to discover such relation between X and Y,

To make the problem more complicated, we will simulate the labeler to drop some of the output labels.

```
# <ask for missing label.
mask_value = -1
# Drop 2% y0.
Y[: int(N*0.020), 0] = mask_value
# Drop 0.7% y1.
Y[int(N*0.018): int(N*0.0025), 1] = mask_value
# Drop 1.1% y2.
Y[int(N*0.024): int(N*0.0035), 2] = mask_value
```

Let's build a simple model with 4 layers,

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(10, input_dim=2, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(3, activation='sigmoid'))
```

Here is the important part, where we define our custom loss function to "**mask**" only labeled data.

The **mask** will be a tensor to store 3 values for each training sample whether the label is not equal to our **mask_value** (-1),

Then during computing the categorical cross-entropy loss, we only compute those masked losses.

```
from keras import backend as K
def masked_loss_function(y_true, y_pred):
    mask = K.cast(K.not_equal(y_true, mask_value),
K.floatx())
    return K.categorical_crossentropy(y_true * mask, y_pred
* mask)

model.compile(loss=masked_loss_function, optimizer='adam',
metrics=['accuracy'])
```

Training is simple, let's first reserve the last 3000 generated data for the final evaluation test.

And split the rest data into 90% for train and 10% for dev during training.

```
# split into 90% for train and 10% for dev
X_train, X_dev, y_train, y_dev =
train_test_split(X[:-3000], Y[:-3000], test_size=0.9,
```

```
random_state=seed)

history = model.fit(X_train, y_train, validation_data=
(X_dev,y_dev),
                    epochs=2000, batch_size=5000)
```

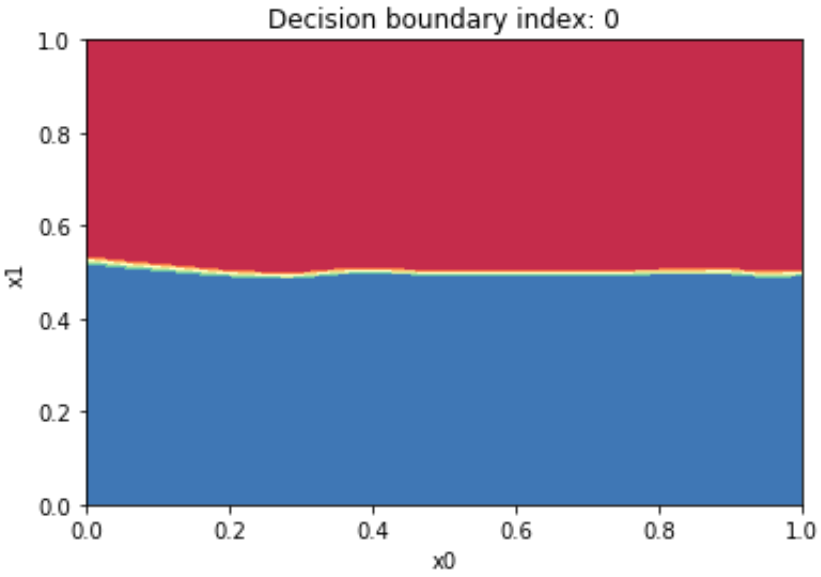
After training for 2000 epochs, let's check the model performance with our reserved evaluation test data.

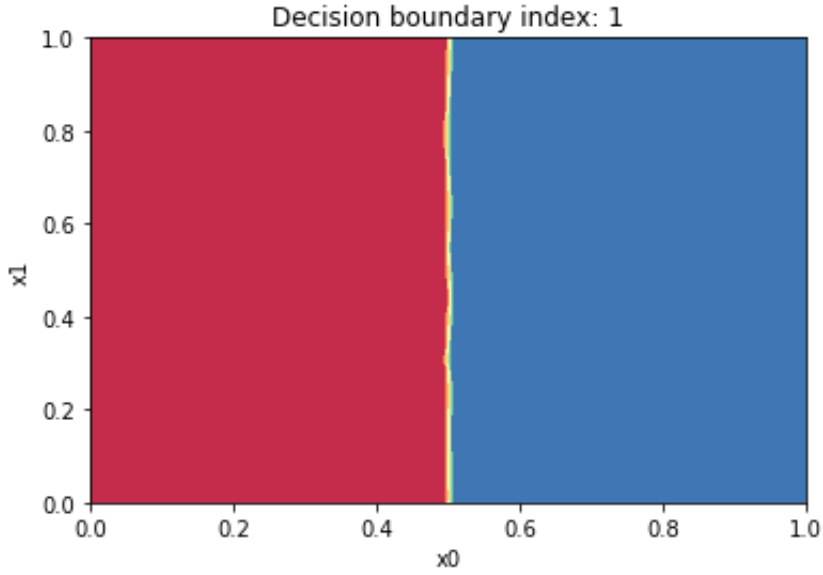
```
model.evaluate(X[-3000:],Y[-3000:])
```

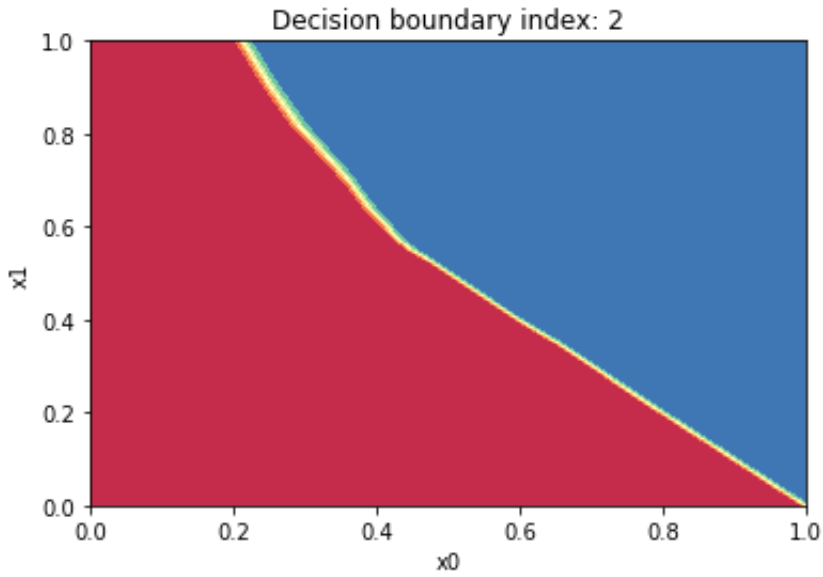
The loss/accuracy is

```
0.908/0.893
```

To help visualize what the model is thinking, let's plot its decision boundary for each of our 3 labels.







Looks like our model figured out the logic between X and Y ;)

If you are not convinced of the effectiveness of our custom loss function. Let's compare them side by side.

To disable our custom loss function, simply change the loss function back to the default `'categorical_crossentropy'` like this.

```
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
```

Then run the model training and evaluation again.

It finally evaluated accuracy is only around 0.527 which is much worse than our previous model with custom loss function.

```
0.909/0.527
```

Check out the source code on my GitHub repo.

Summary and Further Thought

With Multi-task learning, we can train the model on a set of tasks what could benefit from having shared lower-level features.

Usually, the amount of data you have for each task is quite similar.

Some degree of missing labels in training data is not a problem, which can be dealt with a custom loss function to mask only labeled data.

We can do so much more, one potential dataset that came to my mind is the (MBTI) Myers-Briggs Personality Type Dataset.

The Input is the text a given person posted.

There are 4 output labels.

- Introversion (I)—Extroversion (E)
- Intuition (N)—Sensing (S)
- Thinking (T)—Feeling (F)

- Judging (J)—Perceiving (P)

We can treat each one as a binary label.

We can allow the labeler to leave any personality type unlabeled for a given person's post.

The model should still be able to figure out the relationship between the input and output with our custom loss function.

If you have tried this, leave a comment below and let us know if it works.

. . .

Originally published at www.dlology.com.