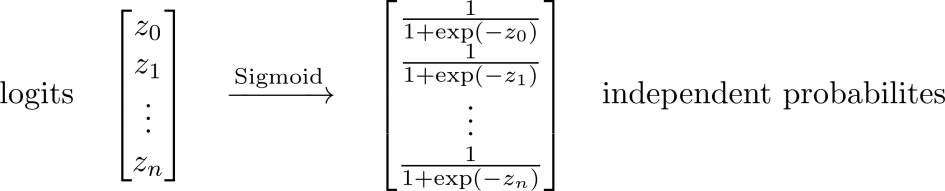
**[Guide To Multi-Class Multi-Label Classification With Neural Networks In Python](https://www.depends-on-the-definition.com/guide-to-multi-label-classification-with-neural-networks/" \o "Guide to multi-class multi-label classification with neural networks in python)**



Often in machine learning tasks, you have multiple possible labels for one sample that are not mutually exclusive. This is called a multi-class, multi-label classification problem. Obvious suspects are image classification and text classification, where a document can have multiple topics. Both of these tasks are well tackled by neural networks. A famous python framework for working with neural networks is [keras](http://www.keras.io/). We will discuss how to use keras to solve this problem. If you are not familiar with keras, check out the excellent documentation.

In [1]:

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense

Using TensorFlow backend.

To begin with, we discuss the general problem and in the next post, I show you an example, where we assume a classification problem with 5 different labels. This means we are given n samples

\[X = \{x_1, \dots, x_n\}\]

and labels

\[y = \{y_1, \dots, y_n\}\]

with y_i\in \{1,2,3,4,5\}. We use a simple neural network as an example to model the probability P(c_j|x_i) of a class c_i given sample x_i. We then estimate out prediction as

\[\hat{y}_i = \text{argmax}_{j\in \{1,2,3,4,5\}} P(c_j|x_i).\]

Now we set up a simple neural net with 5 output nodes, one output node for each possible class.

In [2]:

nn = Sequential()

nn.add(Dense(10, activation="relu", input\_shape=(10,)))

nn.add(Dense(5))

**Multi-class classification**

Now the important part is the choice of the output layer. The usual choice for multi-class classification is the softmax layer. The softmax function is a generalization of the logistic function that “squashes” a K-dimensional vector \mathbf{z} of arbitrary real values to a K-dimensional vector \sigma(\mathbf{z}) of real values in the range [0, 1] that add up to 1.

In [3]:

**import** **math**

**def** softmax(z):

z\_exp = [math.exp(i) **for** i **in** z]

sum\_z\_exp = sum(z\_exp)

**return** [i / sum\_z\_exp **for** i **in** z\_exp]

Assume our last layer (before the activation) returns the numbers z = [1.0, 2.0, 3.0, 4.0, 1.0]. Every number is the value for a class. Lets see what happens if we apply the softmax activation.

In [4]:

z = [1.0, 2.0, 3.0, 4.0, 1.0]

softmax(z)

Out[4]:

[0.031062774127550943,

0.0844373744524495,

0.22952458061688552,

0.623912496675563,

0.031062774127550943]

So we would predict class 4. But let’s understand what we model here. Using the softmax activation function at the output layer results in a neural network that models the probability of a class c_j as multinominal distribution.

\[P(c_j|x_i) = \frac{\exp(z_j)}{\sum_{k=1}^5 \exp(z_k)}.\]

A consequence of using the softmax function is that the probability for a class is not independent from the other class probabilies. This is nice as long as we only want to predict a single label per sample.

**Multi-class mulit-label classification**

But now assume we want to predict multiple labels. For example what object an image contains.  
Say, our network returns

\[z = [-1.0, 5.0, -0.5, 5.0, -0.5]\]

for a sample (e.g. an image).

In [5]:

z = [-1.0, 5.0, -0.5, 4.7, -0.5]

softmax(z)

Out[5]:

[0.0014152405960574873,

0.5709488061694115,

0.002333337273878307,

0.4229692786867745,

0.002333337273878307]

By using softmax, we would clearly pick class 2 and 4. But we have to know how many labels we want for a sample or have to pick a threshold. This is clearly not what we want. If we stick to our image example, the probability that there is a cat in the image should be independent of the probability that there is a dog. Both should be equally likely.

A common activation function for binary classification is the sigmoid function

\[\sigma(z) = \frac{1}{1 + \exp(-z)}\]

for z\in \mathbb{R}.

In [6]:

**def** sigmoid(z):

**return** [1 / (1 + math.exp(-n)) **for** n **in** z]

In [7]:

z = [-1.0, 5.0, -0.5, 5.0, -0.5]

sigmoid(z)

Out[7]:

[0.2689414213699951,

0.9933071490757153,

0.3775406687981454,

0.9933071490757153,

0.3775406687981454]

With the sigmoid activation function at the output layer the neural network models the probability of a class c_j as bernoulli distribution.

\[P(c_j|x_i) = \frac{1}{1 + \exp(-z_j)}.\]

Now the probabilities of each class is independent from the other class probabilities. So we can use the threshold 0.5 as usual. This is exactly what we want. So we set the output activation.

In [8]:

nn = Sequential()

nn.add(Dense(10, activation="relu", input\_shape=(10,)))

nn.add(Dense(5, activation="sigmoid"))

To make this work in keras we need to compile the model. An important choice to make is the loss function. We use the binary\_crossentropy loss and not the usual in multi-class classification used categorical\_crossentropy loss. This might seem unreasonable, but we want to penalize each output node independently. So we pick a binary loss and model the output of the network as a independent Bernoulli distributions per label.

In [9]:

nn.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

To get everything running, you now need to get the labels in a “multi-hot-encoding”. A label vector should look like

\[l = [0, 0, 1, 0, 1]\]

if class 3 and class 5 are present for the label. We will see how to do this in the next post, where we will try to [classify movie genres by movie posters](http://depends-on-the-definition.com/classifying-genres-of-movies-by-looking-at-the-poster-a-neural-approach/) or [this post](http://depends-on-the-definition.com/classify-toxic-comments-on-wikipedia/) about a kaggle challenge applying this. Note that you can view image segmentation, like in [this post](https://www.depends-on-the-definition.com/unet-keras-segmenting-images/), as a extreme case of multi-label classification.