Fastai Lesson 10a review

Joseph Catanzarite
Fastai Deep Learning From The Foundations
TWiML Study Group Meetup
Saturday, 8/17/2019

Overview of Lesson 10a

- Softmax discussion
- Notebooks:
  - 04_callbacks
  - 05_anneal
  - 05a_foundations
  - 05b_early_stopping
- We’ll stop at 1:00:40 in Lesson 10 video
- We’ll cover the rest of Lesson 10 next week

Softmax definition and implementation

Softmax

Cross-entropy loss is the usual loss function for multi-label classification problems. Since cross-entropy loss is computed from the probabilities of the predicted classes, we must first convert the output activations to probabilities; this is accomplished by applying the softmax function to the output activations.

Softmax is defined by:

$$\mathrm{softmax(x)}_i = \frac{e^{x_i}}{e^{x_0} + e^{x_1} + \cdots + e^{x_{n-1}}}$$

or more concisely:

$$\mathrm{softmax(x)}_i = \frac{e^{x_i}}{\sum_{0 \leq j \leq n-1} e^{x_j}}$$

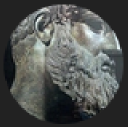
In the above formulas, i indexes the output activation node.

In practice, we will need the log of the softmax when we calculate the loss.

In [10]:

```
# compute log(softmax)
def log_softmax(x):

    return (x.exp()/(x.exp().sum(-1,keepdim=True))).log()
```



jcatanza Joseph Catanzarite 9 10d

Found it! @jeremy discusses `Softmax` starting at 44:38 in Lesson 10 video, and ending at 52:44. He's discussing the `entropy_example.xlsx` spreadsheet and the section labelled **Softmax** in the `05a_foundations.ipynb` notebook.

Two key points @jeremy makes are that `Softmax` operates under the assumption that *each data point belongs to exactly one of the classes*, and that `Softmax` works well when these assumptions are satisfied.

However, the assumptions are **not** satisfied for
(1) **multi-class, multi-label** problems where a data point can be a member of more than one class (i.e. have more than one label), or
(2) **missing label** problems where the identified classes do not provide a complete representation of the data, i.e. there are data points that belong to none of the classes.

So what to do about these cases?

@jeremy shows empirically that for **multiclass, multilabel** problems a better approach is to create a **binary classifier** for each of the classes.

When to use a binary classifier for each class instead of softmax

L6 =H6/(1+H6)

	A	B	C	D	E	F	G	H	I	J	K	L
1		Image 1					Image 2				**binomial**	**binary**
2		**output**	**exp**	**softmax**			**output**	**exp**	**softmax**		Image 1	Image 2
3	cat	0.02	1.02	0.08		cat	-1.42	0.24	0.08		0.51	0.19
4	dog	-2.49	0.08	0.01		dog	-3.93	0.02	0.01		0.08	0.02
5	plane	-1.75	0.17	0.01		plane	-3.19	0.04	0.01		0.15	0.04
6	fish	2.07	7.93	0.62		fish	0.63	1.87	0.62		0.89	0.65
7	building	1.25	3.49	0.27		building	-0.19	0.82	0.27		0.78	0.45
8			12.70	1.00				3.00	1.00			
9												

What does softmax do when two samples have weights that differ by a constant?

Remember that the weights are not yet normalized before **Softmax** is applied. What happens when the weights of the two pixels differ by a constant offset? i.e. What happens when each pixel 2 weight is d units different from the corresponding pixel 1 weight, where d is a constant?

When you transform the weights by exponentiation, a **constant offset** d between two sets of weights becomes a **multiplicative factor**.

To see this, suppose that the weights of pixel 1 are

$$w_1, w_2$$

and the weights of pixel 2 differ from those of pixel 1 by an offset d, so that they are

$$w_1 + d, w_2 + d$$

What does softmax do when two samples have weights that differ by a constant? cont'd

The exponentiated weights for pixel 1 are

> $\exp w_1, \exp w_2$

And the exponentiated weights for pixel 2 are

> $\exp(w_1 + d), \exp(w_2 + d) = k \exp w_1, k \exp w_2,$

where $k = \exp d$.

Now for each pixel, **Softmax** normalizes the exponentiated weights by their sum, and since

> the weights of pixel 2 **are proportional to** those of pixel 1 by a multiplicative factor of k,

the **normalized exponentiated weights** will be the same for the two pixels. This is exactly what happens in the example Jeremy discusses in Lesson 10.

For **missing label** problems, @jeremy says that some practitioners have tried
(A) adding a category for **none-of-the-above**, or alternately
(B) doubling the number categories by adding categories for **not(each class)**.

However, he says that both of these approaches are terrible, dumb and wrong, because it can be difficult to capture features that describe these ‘negative’ categories.

While I agree that the ‘negative class’ features could be hard to capture, I’m not convinced that either of the approaches (A) and (B) are wrong, since in each case, the classes satisfy the `Softmax` assumptions.

Case (A): if you can learn what features are present in a certain **class K**, you also know that when these features are absent, the data is not likely to be a member of **class K**. This means that learning to recognize **class K** is implicitly learning to recognize **class not(K)**.

Case (B) I’d argue that **none-of-the-above**ness *can* be learned with enough examples.

So I don’t see anything wrong with these approaches to handle the case of missing classes.

To summarize, `Softmax` works well when its assumptions are satisfied, and gives wrong or misleading results otherwise. An example of the former case: `Softmax` works well in language modeling when you are asking “what’s the next word?” An example of the latter case is when there are missing classes and you **don’t** account for this situation by using, say approach A or B above; in this case the output probabilities are entirely bogus. **Multiclass, multilabel** problems provide another example where `Softmax` is the wrong approach, because the class probabilities do not sum to one.

Review notebooks