Information Bottleneck Theory

Our goal here is twofold. First, to understand what the information bottleneck theory is saying on a conceptual level. This involves ideas and interpreting some equations. But we also want to understand the basics of how the relevant quantities are computed. This is the second goal, and it will be accomplished with the help of a simple toy problem.

Consider a discrete probability distribution Y. We can think of each of the potential values of Y as a class label. Each class could be a different type of animal, for example. Out in the world, our experience of Y is mediated by another random variable X. To continue with the concrete example, if Y is the class label of an animal, X is an image of an animal. This means X is a high dimensional vector whereas Y is univariate. Our goal is to learn a model that takes in a realization of X and predicts a class label $\hat{Y}$. Thinking of this model as a deep neural network, there is an intermediate layer T which is a deterministic function of the input $f(X)$, governed by parameters (weights and biases) p. If T is low-dimensional, meaning a vector with dimension much smaller than X, this is a compression step and the parameters p define an encoding $X \to T$. The state of the intermediate layer is mapped to a prediction $\hat{Y}$ through another deterministic function $g(T)$ governed by parameters q. See the figure.

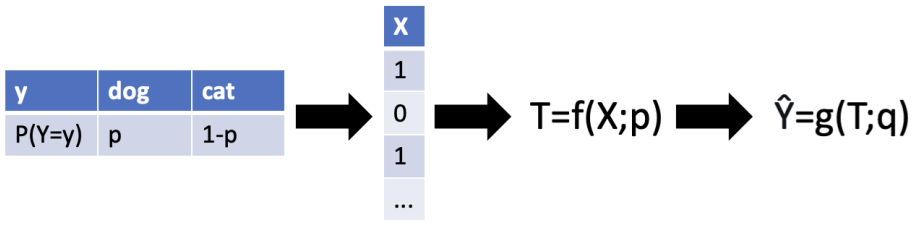

Figure 5: Vectors of pixel values X correspond in some unknown way to class labels Y. We try to learn a mapping from images to class labels which is accurate on the training data and generalizes to unseen images.

The simplest way to learn the parameters p and q for the classification model is to note that $\hat{Y} = (g \circ f)(X)$, and compute the parameters through the following optimization problem:

$$p^*, q^* = \underset{p,q}{\operatorname{argmin}} \; E\left[\left(\hat{Y}(p,q) - Y\right)^2\right]$$

This is just the mean-squared error between the predicted and the actual class labels. Nothing stops this algorithm from "memorizing" the data, which

means that it may not perform well on unseen images. This is also called overfitting. Our intuition is that preventing overfitting will lead to better generalization of the learned model. We might expect that there is some abstract feature essential to the class labels that defines why one image is a dog and another is a cat. Getting at this abstract thing should actually require ignoring a lot of other features. The background of the image shouldn't matter, nor should the size of the animal in the picture (it could be taken from close up or far away). The model should learn "dog-ness" vs. "cat-ness," rather than learning that all dogs in the training data happen to have some superficial feature in common.

The information bottleneck theory is a potential answer to the question of how a model learns to generalize. The following optimization problem is at the heart of the information bottleneck theory:

$$\underset{p}{\text{argmin}}\ I(X;T) - \beta I(T;Y)$$

Remember that the parameters p control the relation between X and T. The parameter β is a scalar that controls the relative weighting of the two terms in the objective, and $I(A;B)$ is the mutual information between two random variables A and B. The mutual information can be conceptualized as a (symmetric) measure of the extent to which knowing the value of one variable decreases uncertainty about the other. What does it mean for this quantity to be minimized? Roughly, it means that the hidden layer T becomes only loosely related to X (driving the first term down), but still retains information about Y (making the second term big). The intuition is that the irrelevant details of the input are forgotten as the input is processed through the preceding layers of the neural network, but the details relevant for the prediction of Y are retained. The dimensionality reduction layer T is thus the "bottleneck."

If we use the colloquial definition of information to interpret the two mutual information terms in the objective, this makes sense. The hidden layer loses information about the details of the input X, but keeps the information relevant to the true output Y. When the superfluous details of X are neglected, we expect overfitting to be limited and generalization to be good. Though helpful, this is some serious hand-waving. In order to further clarify (and operationalize) this objective, we can use the chain rule for mutual information to write

$$I(X;T) - \beta I(T;Y) = H(X) - H(X|T) - \beta H(Y) + \beta H(Y|T)$$

where $H(\cdot)$ is the information entropy. Noting that X and Y come from the data, their entropies do not depend on the optimization variables p. This means the minimization problem can be rewritten as

$$\underset{p}{\text{argmin}}\ \beta H(Y|T) - H(X|T)$$

These conditional entropies are easier to interpret than the mutual information. The objective is at a minimum when $H(Y|T)$ is small and when $H(X|T)$ is big. The first term is small when all data with a given class label Y are mapped onto the same "latent" state T. In other words, for a given T, there is little uncertainty in the corresponding class label. The second term is big when many inputs map onto a few number of latent states T. The first term controls prediction accuracy, the second term controls compression, and β controls how much accuracy is weighted over compression. This is the essence of the information bottleneck: the tradeoff between prediction accuracy and compression.

Framing things in terms of conditional entropies, a measure of average uncertainty in one variable given another, helps think more clearly about this problem. But it is instructive to play around with an example. It seems that the information bottleneck objective is best suited as a tool to understand how deep neural networks can learn to generalize, not as an objective function to train a neural network. In other words, it is used to make sense of the training process, which is carried out with some other loss function. Despite apparently not being realistic for training neural networks, using the information bottleneck objective as a loss to train a simple learning algorithm is pedagogically useful.

Y (labels)	X (input data)
3	0000
2	0001
0	0010
1	0100
$\vdots$	$\vdots$

Table 1: Made-up data for the information bottleneck toy problem. The data X (a proxy for images) consists of all 16 binary sequences of length 4. Each sequence corresponds to some label $Y = 0, 1, 2, 3$. We want to demonstrate how the information bottleneck objective is computed given a data set of this sort and explore what sort of mappings $X \to T$ perform well with the given objective function.

A snapshot of the data for the toy problem is shown in the table. As before, we can write $T = f(X; p)$ where p is a discrete parameter that controls how the different realizations of X are mapped onto the discrete states of T. Notice that the information bottleneck objective has nothing to say about the agreement of the predictions $\hat{Y}$ with the true labels Y. This is a bit strange. It seems that this objective mostly deals with good encoding of the data where "good" is defined by a trade-off between accuracy and compression. Presumably, a decoder could be learned in a second step if we have a good encoding in hand. We will take the latent representation T to be the prediction $\hat{Y}$ in our example. This means that $T = \hat{Y} = 0, 1, 2, 3$. Basically, we want to see if we recover the

true relationship between the data and its labels by minimizing the information bottleneck objective with the latent representation being the prediction itself. The objective is then

$$\underset{p}{\text{argmin}}\ \beta H(Y|\hat{Y}(p)) - H(X|\hat{Y}(p))$$

In practice, there is no clear way to write the dependence of the prediction on a parameter p. What we will do is randomly generate classification rules $X \to \hat{Y}$ and observe the value of this objective for different rules. This is why we can think abstractly of p as a "discrete parameter." There are a finite number of ways to map the data to predictions. I agree, things are getting a bit weird. Figuring out exactly what is happening at this point is an exercise left to the reader.

A brief sketch of how the conditional entropies are computed is in order before getting to the punchline. Say that we have randomly generated a classification rule. This means that each X in the input data is randomly assigned to a prediction $\hat{Y}$. This might look something like

$$X = 0000 \to \hat{Y} = 1, \quad X = 1001 \to \hat{Y} = 3, \quad X = 1111 \to \hat{Y} = 2, \quad \dots$$

The first term in the loss can be computed as

$$H(Y|\hat{Y}) = \sum_i p(\hat{Y} = i) H(Y|\hat{Y} = i)$$

$$H(Y|\hat{Y} = i) = -\sum_j p(Y = j|\hat{Y} = i) \log p(Y = j|\hat{Y} = i)$$

The conditional distribution in the second line is computed simply by using the relative frequencies of the true labels Y corresponding to each predicted label $\hat{Y} = i$. The second term in the loss is

$$H(X|\hat{Y}) = \sum_i p(\hat{Y} = i) H(X|\hat{Y} = i) = \sum_i p(\hat{Y} = i) \log N_i$$

where N_i is the number of binary sequences classified to $\hat{Y} = i$. This relies on the assumption that each of the input data points is equally likely. Now that we know how to compute the objective, we can look at its values for different classification schemes.

We can run the problem with 1000 randomly generated classification schemes for different values of β. The minimum value of the information bottleneck objective obtained over the 1000 runs will be called $z(\beta)$. With the given data set, we have $z_{ref} = -2.11$ for the classification scheme which perfectly reproduces the input data. We can use this to see how the randomly generated classification schemes fair in comparison to the "exact solution." See the table.

β	1E-2	1E-1	1E0	1E1
$z(\beta)$	-2.44	-2.37	-1.52	5.81

Table 2: Lowest information bottleneck objective taken from 1000 runs of randomly generated classification schemes for different values of the trade-off parameter β.

We see from the table that when compression is prioritized over prediction accuracy (small β), at least one of the randomly generated classification schemes does better than the exact solution. This is because the random classification schemes occasionally produce compressions of the data where some of the T values are not used. When one or more of the potential states of T is ignored, the data is compressed down to fewer states. If some of the labels Y occur more frequently than others, it is possible to do this compression without significant loss of prediction error. When β is large, none of the random schemes do better than perfect prediction. This makes sense, and generally illustrates how the information bottleneck theory quantifies a trade-off between prediction accuracy and compression of the data.