Zhiping Xiu

1): Reference to *A Brief Introduction to Boosting*

A):

Each weak W_t hypothesis is constructed by D_t, after each round, q portion of the D_t is flipped over, causing Wt take some penalty of accuracy (for each succeeding round). So this will making the Wt less and less accurate, which will causes the final hypothesis less accurate.

B):

One thought might improve the AdaBoost for giving situation, consider that:
Each Wt hypothesis is becoming less and less accurate with each iteration. Our algorithm should present such penalty by assigning a penalty value to a_t after every iteration. The older weak learner will take bigger penalty, and it will eventually converge to 0.5. (this can be illustrated by a program in my assignment folder ~ python p1.py)

Or formally, add the purple step to the original algorithm:

- constructing D_t:
  - $D_1(i) = \frac{1}{m}$
  - given D_t and h_t:

$$D_{t+1} = \frac{D_t}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t}{Z_t} \cdot \exp(-\alpha_t \cdot y_i \cdot h_t(x_i))$$

where: Z_t = normalization constant

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) > 0$$

$\varepsilon_i = \varepsilon_i * (1-q) + (1-\varepsilon_i) * q$ for i from 1 to t.

- final hypothesis: $H_{\text{final}}(x) = \text{sgn}\left(\sum_t \alpha_t h_t(x)\right)$

C):

Reason:
h_t in each iteration is learned based on D_t, and since the data in D_t is going to mutate after each iteration, our confidence towards D_t are also decreasing after each iteration. We can reflect such decreasing by apply a penalty to error rate in each iteration. Since the older iteration will take more mutation, the result will be more inaccurate, thus we can trust their result less.

To illustrate this penalty better, I will go with an example: say we have an original dataset with all data labeled 1 (the other label is 0, but currently they are all labeled one), and we have a learner which can correctly divide all data (acc 100%, I know this is not very possible, but this can make the example simpler). So, after first round, q portion of data changed, so even though the first learner can have acc 100%, the actual accuracy can only be 1-q. After 10 round, those accuracy will decrease further, and eventually converge to 50%. (I will illustrate why 50% in next paragraph, and a testing script.)

And by setting $\varepsilon_i = \varepsilon_i * (1-q) + (1-\varepsilon_i) * q$ for i from 1 to t we can apply penalty to each error rate in every iteration, and the older error rate take more penalty, this penalty will lead the accuracy eventually converge to 0.5, leading to a α close to 0.

Such penalty method can be illustrated by an example script in my assignment folder, test it using, in this example, accuracy is getting punished in each iteration, and the accuracy is eventually converge to 0.5.

```
~ python p1.py
```

See next page for following questions:

Zhiping Xiu

2): Reference to *Improving Generalization with Active Learning*

A):

The idea of active learning is, only ask for supervised information when necessary. For example, when we are not confident about current result gotten from machine learner.

Some time we have a lot of unlabeled training example but few labeled example, and labeled example are **scarce** or **expensive** to get. For example, considering about movie recommendation problem, we have lot of unlabeled users data and movie data, but only few labeled user and movie pair (only few movie are labeled by the user). And getting those data could be expensive (someone has to watch the movie), at this time, we can use active learning to training our learner.

B):

A learner will exam the information already given and determining a region of uncertainty, which is an area in the domain where the learner believe the misclassification is still possible, the learner then asks for examples exclusively from that region, this sampling approach is called **selective sampling**.

More specifically, we can maintain two subsets of the version space, S and G, respectively stands for **most specific consistent concepts** and **most general concepts**. In other word, we are enough confident that samples inside S should be all positive, samples outside G should be all negative, and we are uncertain about the samples between S and G (SΔG).

So, we draw sample from SΔG, and query the actual value of this sample, if this sample is positive, we enlarge S to cover this sample, if this sample is negative, we shrink G to exclude this sample. Either way we can reduce the area of SΔG, thus enlarge the space of certainty.

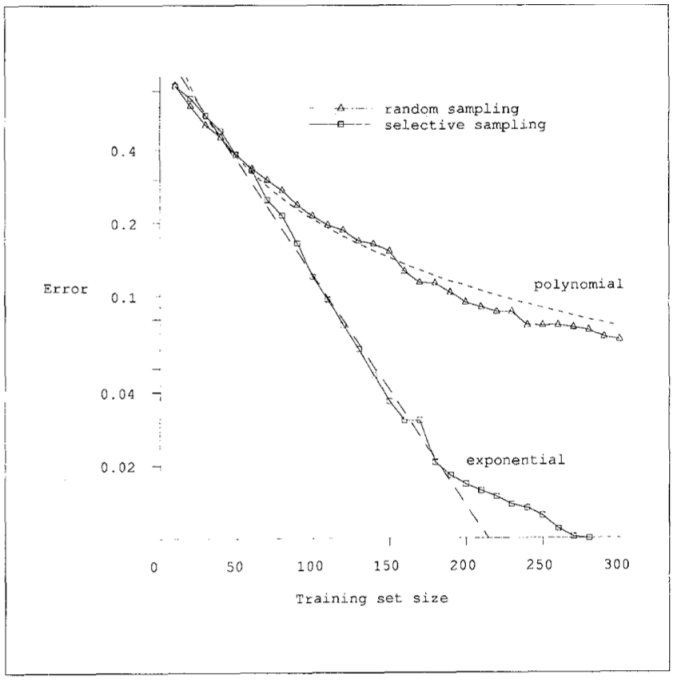
C):

According to the paper,

For selective sampling SG neural network:
Indicating a good fit to the exponential curve (up until the error drops below 1.5%), $\varepsilon = e^{a * m + b}$ for a = - 0.0218 and b = 0.071, with coefficient of determination of $r^2 = 0.995$.

For random sampling:
Indicating a roughly fit to the polynomial curve, $\varepsilon = (a * b + b)^{-1}$ for a = 0.0514 and b = - 0.076, with coefficient of determination of $r^2 = 0.987$.

Table 12 from the paper visualized both curve, as we can see from the picture, error rate of selective sampling decrease with steeper slope with the increase examples, compared to random sampling.



3): Reference to *Classifier Technology and the Illusion of Progress*

A): Thesis:

In real problem, some time simple methods out perform or at least as competitive as the sophisticated ones. Difference in performance may be caused by some other sources, which is not considered in the classical supervised classification paradigm. And argue that the progress made on theoretical level, currently translate poorly to practical level.

B): Evidence to back up their argument:

The argument is demonstrated in three steps:

1: Large gain in predictive accuracy in classification are gotten using relatively simple models at the start of the process, the gain of substitute process decreased with the process taken further. And the further gain beyond that attained by simple models is achieved from trivial aspects.
2: In most of the real classification problem, data points is the design set are not randomly drawn from the same distribution as the data points to which the classifier will be applied. Caused that the classifier accuracy is based on a false assumption about the data we aimed to observe.
3: When constructing classification rules, some assumptions and choices are not appropriately made, which may give misleading impressions to future classifier performance.

This three steps is described further with example in Section 2-4.

C): Do you agree, why or why not?

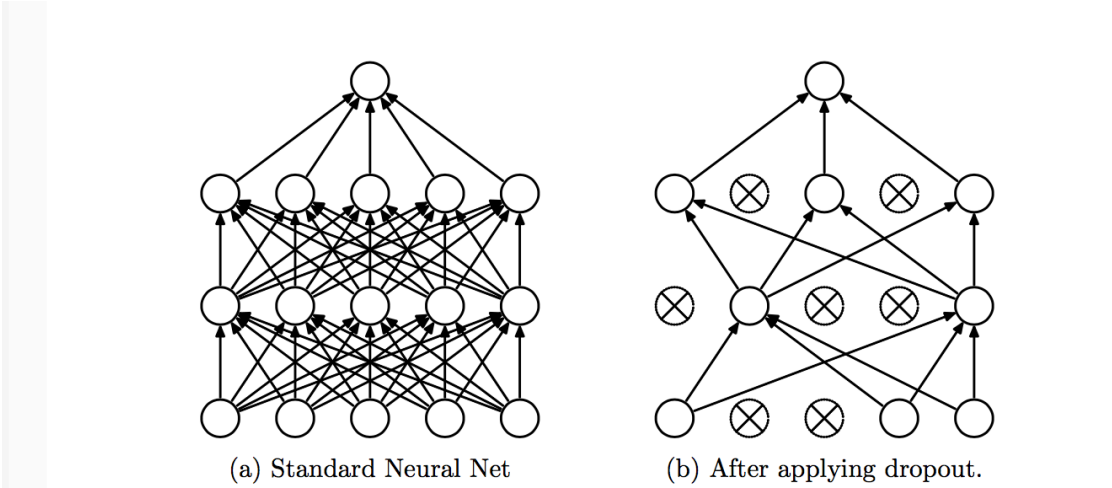
I partially agree with the idea, we should keep things simple, but we should also realize simple thing has its own limitation.

What does this paper point out is reasonable, we did make impropriate assumption towards the data when we are trying to use a machine learner to interpret them. And I believe it is also reasonable to realize that some sophisticated machine leaner, especially those who perform like ‘black box’, such as SVM and DNN, may sometime focused on learning trivial aspects instead of learning the more general truth of the data, like some simple learner do.

But I also believe that there are also somethings that simple model cannot discover. Because of their limited representative power, and since most of the sometimes human cannot distinguish which model can fit the data perfectly, making assumptions and build sophisticated machine learner and trying to use them to discover implicit attribute that we could not see seems to be the only logical thing we can do to make progress.

4): Reference to *Dropout: A Simple Way to Prevent Overfitting*

A): Dropout is a regularization method aimed to prevent neural network from overfitting. This can be achieved by randomly drop nodes from neural network, along with it's connections. A chart from the paper *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* can illustrate this idea perfectly. By removing some node from the network, we can prevent nodes from co-adapting too much with each other than we want (which may lead to overfitting).



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

B): Model combination in machine learning means combining different machine learning model to get a final result. From my perspective, boosting can be seen as one type of model combination. The different machine learning algorithm is expected to have different architecture, or trained on different data.

The advantage of model combination is, since the final model combined different machine learning algorithm (or trained on different data), the different model can compensate with each other for different situation (usually close to boundary).

C): Section 6 applied Dropout on different algorithm, on different dataset, and shown the results from experiment. The chart on the right is partial of the results from the paper:

The results were compared based on error rate, the error rate is reduced after applying Dropout to the original algorithm. Thus the result shown the evidence that Dropouts improves generalization.

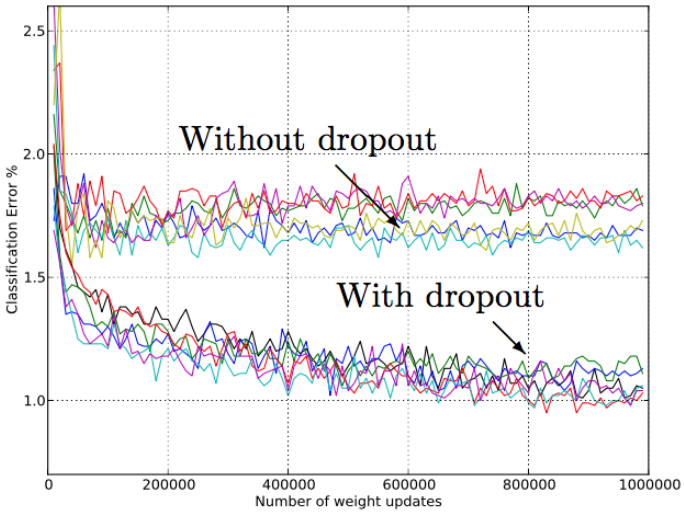


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

D): According to 6.4 section of the paper, Bayesian neural nets weight each model according to the prior and how well the model fit the data.

Bayesian neural nets are useful for solving problems in domains where data is scarce. But training bayesian neural nets consume more computational power than Dropout (mainly slower), especially when it is scaled to very large network sizes.