Summary

2017/7/20

I. Paper Review

1. Bayesian Approach

- MacKay, D. J. (1992). The evidence framework applied to classification networks. Neural computation, 4(5), 720-736

Bayesian methods have strong assumption on data distribution and is really computationally expensive. Despite its solid math explanation power, this method is not a good fit for deep neural network. Details of this method will be skipped.

2. Delta Approach

- Hwang, J. G., & Ding, A. A. (1997). Prediction intervals for artificial neural networks. Journal of the American Statistical Association, 92(438), 748-757
- Chryssolouris, G., Lee, M., & Ramsey, A. (1996). Confidence interval prediction for neural network models. IEEE Transactions on neural networks, 7(1), 229-232

Delta approach analyzes the properties of neural network as a nonlinear regression function $f(x)$ with a very good properties, and using nonlinear regression analysis, derive an asymptotic result for prediction value and construct Confident Interval and Prediction Interval based on that.

Given

$$y = g_\theta(x) + \epsilon$$

where $g_\theta(x) = \alpha_0 + \sum_{i=1}^{k} \alpha_i f(\beta_i^t x + \beta_{i0})$, $f(x) = \frac{1}{1+e^{-x}}$ and ϵ are i.i.d with mean 0 and variance σ^2.

$$\sqrt{n}(\hat{\theta} - \theta^*) \longrightarrow N(0, \sigma^2 V(\theta^*))$$

$$\sqrt{n}[g_{\hat{\theta}}\ (x_{n+1})\ - g_{\theta^*}(x_{n+1})] \longrightarrow N(0, \sigma^2 (\nabla_\theta g_\theta(x_{n+1}))^t\ \Sigma^{-1}\ \nabla_\theta g_\theta(x_{n+1}))$$

Therefore, the confident interval and prediction interval should be:

$$g_{\hat{\theta}}\ (x_{n+1}) \pm t_{1-\alpha/2, n-(d+2)k-1} \hat{\sigma} \sqrt{S(\hat{\theta})}$$

$$g_{\hat{\theta}}\ (x_{n+1}) \pm t_{1-\alpha/2, n-(d+2)k-1} \hat{\sigma} \sqrt{1 + S(\hat{\theta})}$$

where

$$\hat{\sigma^2} = \frac{1}{n - (d+2)k - 1} \sum_{i=1}^{n} (Y_i - g_{\hat{\theta}}\ (x_{n+1}))^2$$

and

$$S(\hat{\theta}) = \frac{1}{n}\left\{\nabla_\theta g_\theta(x_{n+1}))^t\ \hat{\Sigma}^{-1}\ \nabla_\theta g_\theta(x_{n+1})\right\}_{\theta=\hat{\theta}}$$

and

$$\hat{\Sigma}^{-1} = \frac{1}{n}\sum_{i=1}^{n}\left[\nabla_\theta g_\theta(x_{n+1})\,\nabla_\theta g_\theta(x_{n+1})^t\right]_{\theta=\hat{\theta}}$$

Delta methods also have strong assumptions on data distribution and the properties of activation function we use. As soon as we use some different activation functions other than sigmoid, and using training tachniques like dropout, we cannot ensure the conditions hold for delta approach. Also, it is highly computationally expensive, and not a good fit for deep neural network.

3. Bootstrapping Approach

- Heskes, T. (1997). Practical confidence and prediction intervals. In Advances in neural information processing systems (pp. 176-182)
- Carney, J. G., Cunningham, P., & Bhagwan, U. (1999, July). Confidence and prediction intervals for neural network ensembles. In Neural Networks, 1999. IJCNN'99. International Joint Conference on (Vol. 2, pp. 1215-1218). IEEE
- Zio, E. (2006). A study of the bootstrap method for estimating the accuracy of artificial neural networks in predicting nuclear transient processes. IEEE Transactions on Nuclear Science, 53(3), 1460-1478

Bootstrapping method is the most common technique for the construction of CIs and PIs. We assume that:

$$y = f(x) + \epsilon(x)$$

where $\epsilon \sim N(0, \chi^2(x))$.

First, N training datasets are resampled from the original dataset with sample size n. On each training dataset, we train a single neural network, the output of each neural network is denoted as $f^{(j)}(x_i)$. As the estimate of our ensemble networks, we take the average:

$$\hat{f}(x_i) = \frac{1}{N}\sum_{j=1}^{N} f^{(j)}(x_i)$$

By assuming a Gaussian distribution $P(f(x_i) \mid \hat{f}(x_i))$, we can estimate its variance by:

$$\hat{\sigma}^2(x_i) = \frac{1}{N-1}\sum_{i=1}^{N}\left[f^{(j)}(x_i) - \hat{f}(x_i)\right]^2$$

Therefore, the confident interval should be:

$$\hat{f}(x_i) - t_{1-\alpha/2;\,N-2}\hat{\sigma}(x_i) \le f(x_i) \le \hat{f}(x_i) + t_{1-\alpha/2;\,N-2}\hat{\sigma}(x_i)$$

As for the prediction interval,

$$s^2(x_i) = [y_i - \hat{f}(x_i)]^2 = [f(x_i) - \hat{f}(x_i)]^2 + \chi^2(x_i)$$

We can now try to estimate a model $\hat{\chi}^2(x)$ to fit the remaining residuals:

$$r^2(x_i) \equiv max\left([y_i - \hat{f}(x_i)]^2 - \hat{\sigma}^2(x_i),\ 0\right)$$

using negative log likelihood as cost function:

$$L \equiv -\sum_{i=1}^{n} log\left[\frac{1}{\sqrt{2\pi\chi^2(x_i)}} exp\left(-\frac{r^2(x_i)}{2\chi^2(x_i)}\right)\right]$$

Instead of using some new data to fit $\chi^2(x_i)$, we can utilize our bootstrap procedure. Since each time we bootstrap, there are about $\frac{1}{e} \approx 36.8\%$ data not included in the resample dataset. Rewrite $\hat{f}(x_i)$:

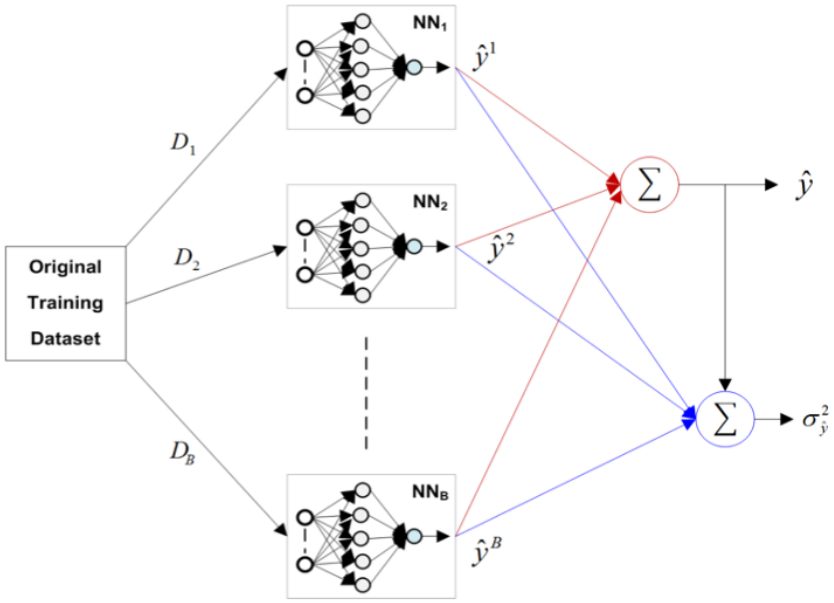
$$m(x_i) = \frac{\sum_{j=1}^{N} q^{(j)}\, f^{(j)}(x_i)}{\sum_{j=1}^{N} q^{(j)}}$$

where $q^{(j)} = 1$ if x_i is in the jth bootstrap dataset and $q^{(j)} = 0$ otherwise. Using $m(x_i)$ and $r^2(x_i)$ derived from $m(x_i)$:

$$r^2(x_i) \equiv max\left([y_i - m(x_i)]^2 - \hat{\sigma}^2(x_i),\ 0\right)$$

we can train a neural network and get an estimation $\hat{\chi}^2(x_i)$. Therefore, we can compute $\hat{s}^2(x_i)$. Then, the prediction interval for a new point x_{new} should be:

$$\hat{f}(x_{new}) - t_{1-\alpha/2;\,N-2}\hat{s}(x_{new}) \leq f(x_{new}) \leq \hat{f}(x_{new}) + t_{1-\alpha/2;\,N-2}\hat{s}(x_{new})$$



Bootsrapping approach is simpler than Delta and Bayesion approaches without calculate matrices and derivatives, but also computationally expensive since $N + 1$ models are required in total. When it comes to DNN, the computation is even more costly. Besides, it really depends on the unbiased property of NN models. With biased estimator, total variance will be underestimated and resulting in narrow PIs.

4. LUBE (Lower Upper Bound Estimation) Approach

- Khosravi, A., Nahavandi, S., Creighton, D., & Atiya, A. F. (2011). Lower upper bound estimation method for construction of neural network-based prediction intervals. IEEE Transactions on Neural Networks, 22(3), 337-346
- Quan, H., Srinivasan, D., & Khosravi, A. (2014). Short-term load and wind power forecasting using neural network-based prediction intervals. IEEE transactions on neural networks and learning systems, 25(2), 303-315

LUBE constructs lower and upper bound $L(\cdot)$ and $U(\cdot)$ as a function of X_i. First we present some measures for quantitative assessment of PIs. PI coverage probability(PICP):

$$PICP = \frac{1}{n}\sum_{i=1}^{n} c_i$$

where $c_i = 1$ if $y_i \in [L(X_i), U(X_i)]$, otherwise $y_i = 0$. And Normalized Mean Prediction Interval Width (NWPIW):

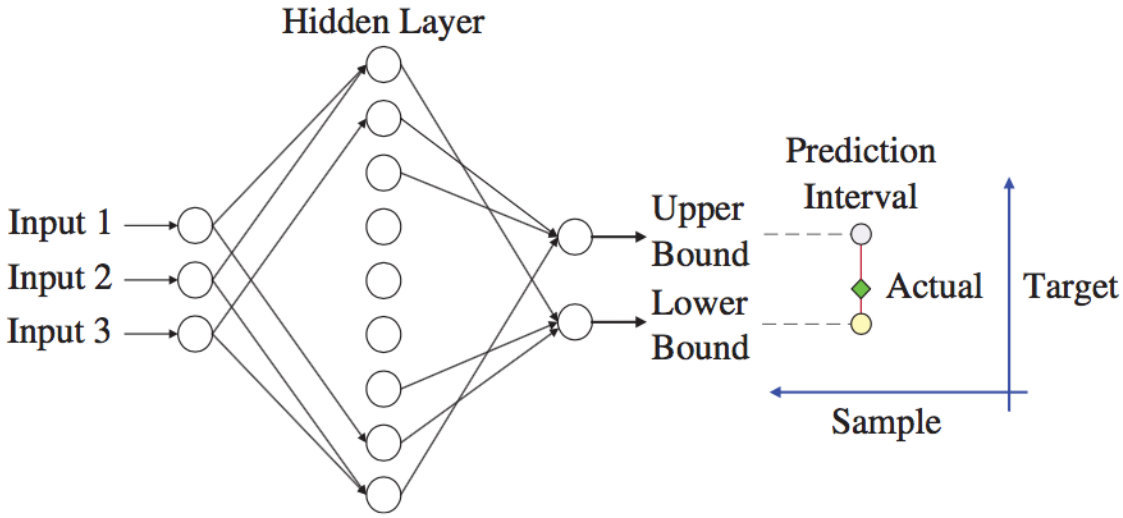
$$NMPIW = \frac{\sum_{i=1}^{n} U(X_i) - L(X_i)}{nR}$$

where R is the range of underlying target. Finally, Coverage Width-based Criterion (CWC) combined these two measures together:

$$CWC = NWPIW(1 + PICPe^{-\eta(PICP-\mu)})$$

where μ, η are two hyperparameters determining how much penalty is assigned to low coverage of PIs.

Using $CWC(X)$ as cost function, we can estimate lower and upper bound for prediction intervals through the Neural Network with a following structure.



LUBE method is inspired by Mean-Variance Estimation approach. It does not provide a point estimator, while we want to obtain point estimator as well as a prediction interval.

5. Mean-Variance Estimation

- Nix, D. A., & Weigend, A. S. (1994, June). Estimating the mean and variance of the target probability distribution. In Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference On (Vol. 1, pp. 55-60). IEEE

Mean-Variance Estimation train the neural network with a modulated cost function that output both mean and variance estimation. We assume that:

$$y = f(x) + \epsilon(x)$$

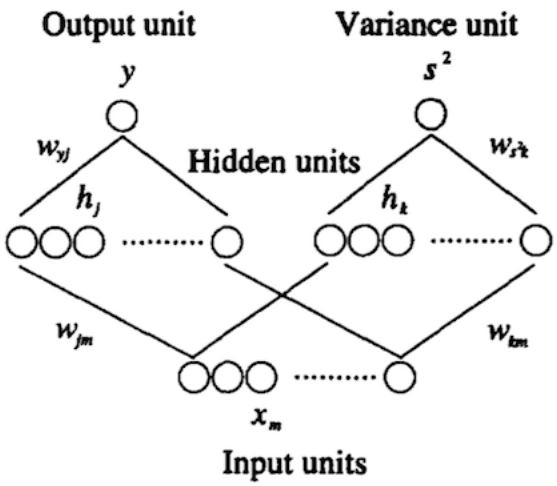
where $\epsilon \sim N(0, \sigma^2(x))$, and the conditional distribution of y on x is:

$$P(y_i|x_i) = \frac{1}{\sqrt{2\pi\sigma^2(x_i)}} exp\left\{-\frac{[y_i - f(x_i)]^2}{2\sigma^2(x_i)}\right\}$$

Inspired by this normal conditional distribution, our cost function is:

$$C(x) = \sum_{i=1}^{n} \frac{1}{2}\left(\frac{[y_i - f(x_i)]^2}{\sigma^2(x_i)} + ln\left[\sigma^2(x_i)\right]\right)$$

Using the above cost function and the neural network below we can estimate mean and variance, as well as construct prediction intervals.



Mean-Variance Estimation is another popular estimation approach. It is not that computationally costly(compared to Delta, Bayesian, Bootstrapping) and does not alter the NN point estimation structure(compared to LUBE). However, it does depend on our assumption about conditional distribution $P(y_i|x_i)$.

6. Summary

- Khosravi, A., Nahavandi, S., Creighton, D., & Atiya, A. F. (2011). Comprehensive review of neural network-based prediction intervals and new advances. IEEE Transactions on neural networks, 22(9), 1341-1356

7. Training Techniques(Updated)

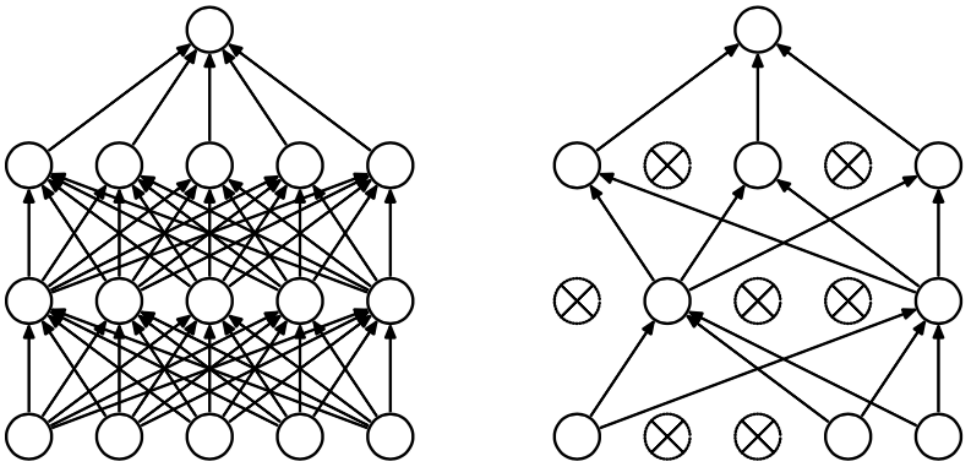
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958

If relationship between input and output are complicated, there might be many different solutions to model the limited amount training data if our network has enough capacibility, however, most of these solutions will perform poorly on test set, since weights of network are tuned to work together to fit the training data.

With unlimited computation, the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by its posterior probability given the training data.

Using this "mean" network s exactly equivalent to taking the geometric mean of the probability distributions over labels predicted by all 2^N possible networks.

In order to avoid overfitting, we use dropout method, which can be seen as an extreme form of bagging in which each model is trained on a single case, but parameters are strongly regularized by sharing with other models. Dropout provides a way of approximately combining exponentially many different neural network architectures efficiently. The term "dropout" refers to dropping out units (hidden and visible) in a neural network. See a diagram of dropout below.



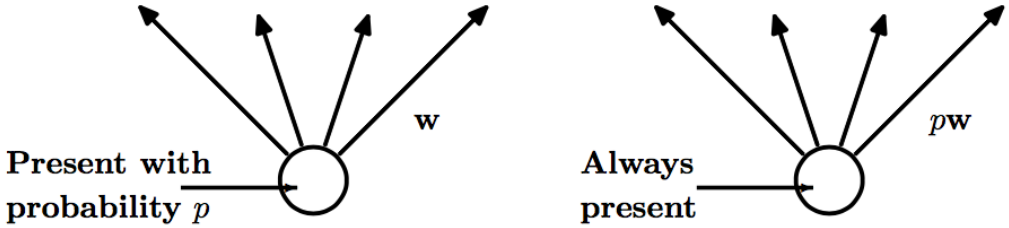
(a) Standard Neural Net (b) After applying dropout.

A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

In practice, use a L2-norm constraint instead of a penalty, which allows a larger learning rate at

the beginnin and thus gives a far more thorough search of the weight-space. Also, we can also do dropout on input pixels.

While doing preditions on test data, the unit is always present and the weights are multiplied by p. In this way, the output at test time is same as the expected output at training time.

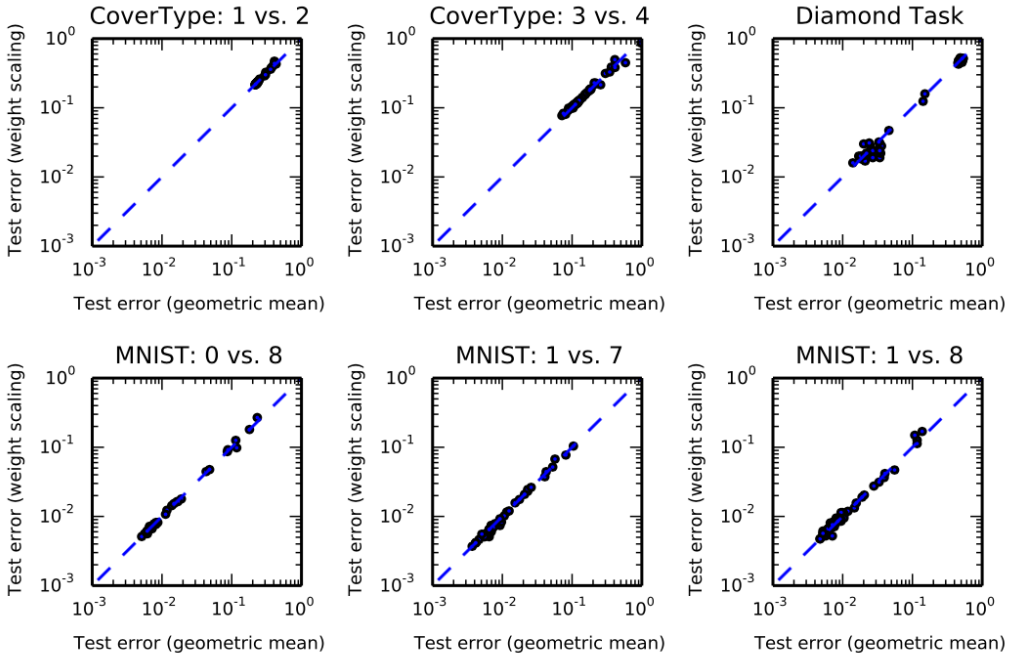


(a) At training time (b) At test time

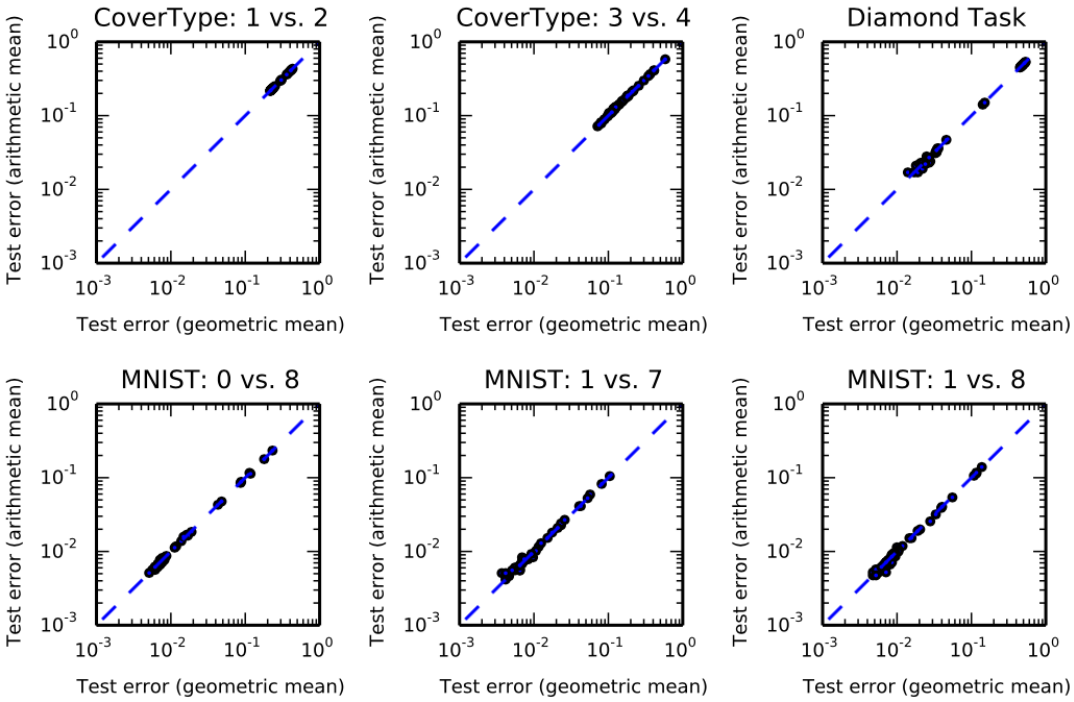
Standard backpropagation learning builds up brittle co-adaptations that work for the training data but do not generalize to unseen data. Random dropout breaks up these co-adaptations by making the presence of any particular hidden unit unreliable. Dropout technique was found to improve the performance of neural nets. **It has benn shown that dropout achieves its regularization effect in linear, sigmoud as well as ReLU hidden layer networks**.

- Baldi, P., & Sadowski, P. J. (2013). Understanding dropout. In Advances in Neural Information Processing Systems (pp. 2814-2822).
- Warde-Farley, D., Goodfellow, I. J., Courville, A., & Bengio, Y. (2013). An empirical analysis of dropout in piecewise linear networks. arXiv preprint arXiv:1312.6197

weight Scaling & geometric mean: Taking the geometric mean is equivalent to arithmatically averaging all values of the input to the sigmoid output unit. Warde-Farley *et al.* (2013) trained 50 networks with dropout, the netwroks were small enough that, for each network, exhausive enumeration over all 2^N subnetwork is possible. As shown below, the weight scaling is a good proxy of geometric mean.



geometric mean & arithmatic mean: Arithmatic mean is more often used for ensemble methods. As shown below, by exhaustively searching all subnetworks, geometric mean and arithmatic mean perform similarly.



Now we have a full and clear view that weight scaling used in dropout trick is a good proxy to estimate arithmatic mean of ensemble networks. Maths detials can be found in Baldi *et al.* (2013).

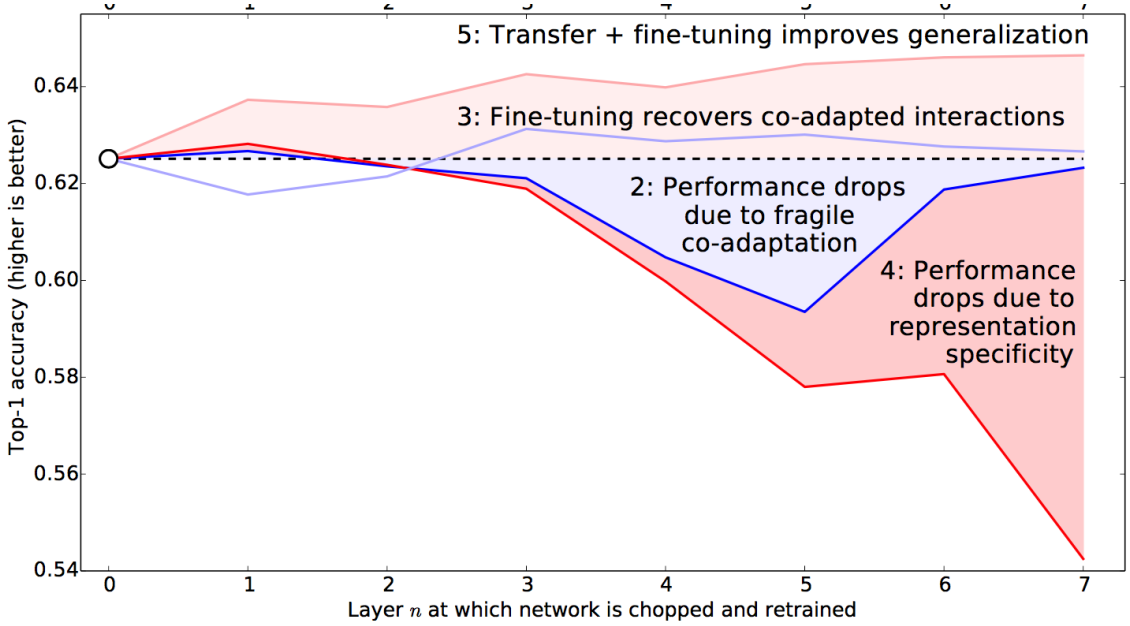
$$p_{weight} \approx p_{geo} \approx p_{arith}$$

8. Transfer Learning(Updated)

- Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks?. In Advances in neural information processing systems (pp. 3320-3328)

For deep neural networks, the first few layers tend to learn some stanford features for similar dataset. The observations raise several questions:

1. Can we measure degree to which a particular layer is general or specific?
2. Does the transition from general to specific occur suddenly or spread out smoothly?
3. Where does the transition happen: front, middle or rear part of networks?



Experiment details are ignored. From the result above, the first few layer tend to learn some common features and are easy to transfer. However, the middle layers have a co-adaption phenomenon, features interact with each other in a complex way and cannot be relearned by upper layers alone. With fine tuning, we can fix this performance drop by fine tuning. Fine tuning works even better together with transfer. Train a network on one dataset A, then transfer to another dataset B, with fine tuning tricks, we can preform even better than training a network directly on B.

However, this boosting of generalization performance will drop if dataset A and B are not so similiar.

II. Ideas

1. Bootstrap

basic idea

Using Bootstrap method, $N = 50$ training datasets are resampled from original dataset with sample size m: $\{x_1^{(i)}...x_m^{(i)}\}$ for $i = 1...N$.

Train $N = 50$ deep neural networks with same structure using $\{x_j^{(i)}\}$ seperately, each DNN get an estimation function $f^{(i)}(x)$. Then, for a new data point, we can make a point estimation through:

$$\overline{f(x_{new})} = \frac{\sum_{i=1}^{N} f^{(i)}(x_{new})}{N}$$

Estimation of variance is:

$$\sigma^2 = \frac{\sum_{i=1}^{N}[\ f^{(i)}(x_{new}) - \overline{f(x_{new})}\ ]^2}{N-1}$$

And the prediction interval should be:

$$\overline{f(x_{new})} - t_{1-\alpha/2;\,N-2}\ \sigma \leq y \leq \overline{f(x_{new})} + t_{1-\alpha/2;\,N-2}\ \sigma$$

Pros and cons

- Computationally Expensive, have to compute m times than before, computation quantity should be $O(mN)$
- Good idea regardless computational problem

Experiment Result

```
Number of Networks: 50
Test accuracy, each individual model: max = 0.98600018, min = 0.97900009
Test accuracy on ensemble model: 0.99040
Test accuracy on wide PI: 0.80323
Test accuracy on narrow PI: 0.99761
```

Potential Improvement

- Fix a fraction of data, for example $\alpha = 50\%$ data, train a DNN using these data, then do bootstrap on other data and train $N = 50$ DNNs based on this DNN. In this way, our computation quantity should be $O(\alpha m + (1-\alpha)mN)$. However, the variance is implicit.
- Inspired from dropout tricks to avoid enormous computation(Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958).

Instead of training data on the whole big neural network, we can randomly dropout some units and train this "thinned" neural network on a bootstrap data. Then we combine N "thinned" neural network together.

2. Division

Basic idea

Divdie data into $N = 50$ parts randomly. Train $N = 50$ deep neural networks with same structure using different parts of data, each DNN get an estimation function $f^{(i)}(x)$. Then, for a new data point, we can make a point estimation through:

$$\overline{f(x_{new})} = \frac{\sum_{i=1}^{N} f_i(x_{new})}{N}$$

Estimation of variance is:

$$\sigma^2 = \frac{\sum_{i=1}^{N}[\ f_i(x_{new}) - \overline{f(x_{new})}\ ]^2}{N-1}$$

And the prediction interval should be:

$$\overline{f(x_{new})} - t_{1-\alpha/2;\,N-2}\ \frac{\sigma}{N-1} \leq y \leq \overline{f(x_{new})} + t_{1-\alpha/2;\,N-2}\ \frac{\sigma}{N-1}$$

Pros and cons(Updated)

- Computationally cheap, computation quantity should be $O(N)$, same with the original point estimation one
- If dataset size m is large enough that

$$\lim_{N\to\infty,\, m\to\infty} \frac{m}{N} \to \infty$$

then this method should be the best since we can have a result converges to its expectation. However, our assumption here is our model has an unbiased prediction of true probability $p(y = 1|x)$, or at least this bias is not large to be a concern.

With limited data, each network is trained on a small piece, with lack of data, each network is very likely to be biased, and $E\left[p(y = 1|x)\right] - E\left[\hat{p}(y = 1|x)\right]$ cannot be ignored.

Experiment Result

```
Number of Networks: 20
Test accuracy, each individual network: max = 0.9673, min = 0.9083
Test accuracy on ensemble model: 0.97670
Test accuracy on wide PI: 0.79788
Test accuracy on narrow PI: 0.99536
```

```
Number of Networks: 50
Test accuracy, each individual network: max = 0.9671, min = 0.9377
Test accuracy on ensemble model: 0.97750
Test accuracy on wide PI: 0.79896
Test accuracy on narrow PI: 0.99646
```

3. Estimate Mean and Variance together

basic idea

Instead of directly constructing conditional distribution $p(y|x;\theta)$, we construct the neural network to representing a function $f(x;\theta)$, the outputs of this function are not direct predictions of the value y.

$f(x;\theta) = \omega$ provides the parameters for a distribution over y. Therefore, our loss function should be

$$J(\theta) = -log\, p(y;\, \omega(x))$$

This is what Mean-Variance Estimation approach (Nix and Weigend 1994) actually did. In that case,

$$p(y;\, \omega(x)) = p(y;\, f(x), \sigma^2(x)) = \frac{1}{\sqrt{2\pi\sigma^2(x)}} exp\left\{-\frac{[y - f(x)]^2}{2\sigma^2(x)}\right\}$$

and loss function is exactly:

$$J(\theta) = -log\, p(y;\, \omega(x)) = \sum_{i=1}^{n} \frac{1}{2}\left(\frac{[y_i - f(x_i)]^2}{\sigma^2(x_i)} + ln\,[\sigma^2(x_i)]\right)$$

In a classification problem, we need a different assumption on $p(y = k;\, \omega(x))$ other than a normal. One of the options is:

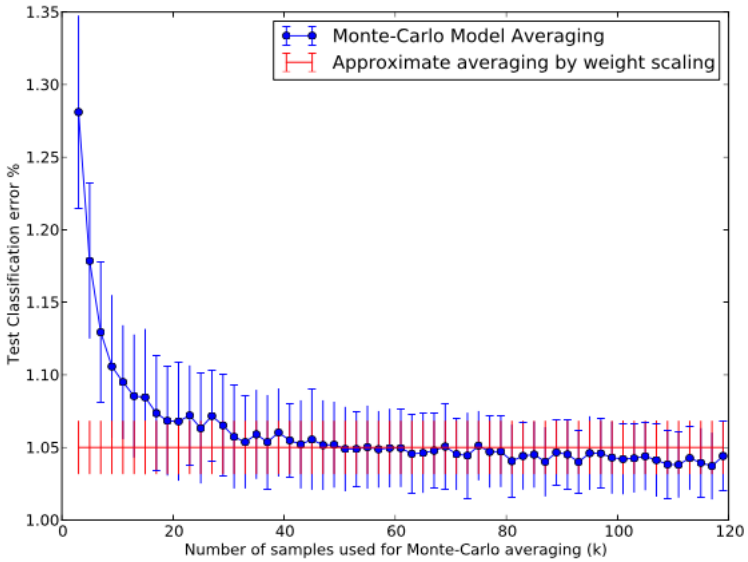
$$P(y = 1|x) \sim N\left(p_1(x), \sigma^2(x)\right)$$

$$P(y = 0|x) \sim N\left(1 - p_1(x), \sigma^2(x)\right)$$

4. Dropout

basic idea

Inspired from dropout tricks (Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958). After training the DNN using dropout approach, instead of multiply dropout probability p on DNN's weights, we do a Monte-Carlo. As the result below shows, a Monte-Carlo average can converge to the weight scaling.



We can construct reduced networks with part of parameters with a dropout rate p for each parameter. If parameter number is n, then we actually do a bootstrap on 2^n network, each produce a $\hat{y}_j = f_i(x_j)$ given sample x_j.

Repeat the procedure $N = 50$ times, we can calculate $f_1(x_{new}), f_2(x_{new}), ..., f_N(x_{new})$, and our first and second order estimatiors are:

$$\overline{f(x_{new})} = \frac{\sum_{i=1}^{N} f_i(x_{new})}{N}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N} [\, f_i(x_{new}) - \overline{f(x_{new})}\,]^2}{N-1}$$

Experiment Result

```
Number of Networks: 50
dropout rate = 0.75
Test accuracy, each individual network: max = 0.9809, min = 0.9762
Test accuracy on newtork with dropout rate = 0: 0.9840
Test accuracy on ensemble model: 0.9839
Test accuracy on wide PI: 0.76715
Test accuracy on narrow PI: 0.99485
```