Comparative study : Dropout as a Bayesian Approximation and Batch Normalization

Ahmed MAZARI Hafed RHOUMA Laurent CETINSOY

March 2, 2017

Abstract

In this articles study summary, we study the importance of regularization in deep learning models. Two regularization techniques are provided : dropout and batch normalization. Up to know, dropout remains the most popular choice for simplicity . Besides, batch normalization outperforms state of art performance in computer vision and eliminates the need of dropout. However, dropout offers insight into the model uncertainty of the deep neural network when it is performed during testing and it can be seen as a bayesian Approximation.We first give a general introduction to over-fitting and regularization. Then, we show how dropout captures model uncertainty and how batch normalization fix the input distribution, and allows deep learning models to learn faster (fast regularization). After that, we discuss the results obtains by both method in different application domains. Finally, we give our intuition about it and our perspectives.

1 Google scholar citation

Dropout as a Bayesian approximation: Representing model uncertainty in deep learning : appeared in 2015 and cited 84 times.
Batch normalization: Accelerating deep network training by reducing internal covariate shift : appeared in 2015 and cited 1124 times.

2 Introduction

Regularization is an important tool in deep learning theory and practice. Since the recent resurgence of deep learning, dropout has been the most common choice thanks to its effectiveness and simplicity, although recently Batch Normalization is starting to draw much attention. We know overfitting happens when the number of trainable parameters is much larger than the amount of data. Thus, we may argue that when we have enough data to fit into current models, we will no longer need these annoying regularization techniques. But at that time, what we should do is to increase the capacity of the network, and we will need regularization again. Although deep models may not be utilizing all its capacity, so is the case with human brain, which clearly has huge numbers of parameters compared to the data that it receives.

3 Problem definition

When training a deep neural network, regularization is an important because the network has a billion of parameters to learn and given the limited training data over-fitting is likely to happen. The two articles [1, 2] tackle the problem of over-fitting from different perspectives. Dropout [3] and batch normalization [2] can be seen as a regularizers that control the penalty for model complexity.
Over-fitting is a common problem in machine learning which characterized by a high variance between error rate of training set and test set. Minimizing training set error does not necessarily minimize test set error as it's depicted in figure 1.
Standard deep learning models don't provide information on model uncertainty comparing to bayesian model which are able to capture more information. However, they are computationally expensive. [1] argues that when dropout is performed during test time, it offers an acute insight into model uncertainty of deep architecture. Specifically, the variance of the label prediction is

$$Var(y) = \tau^{-1} I_D + \frac{1}{T}\sum_{t=1}^{T} \hat{y}^*(x^*, W_1^t, ..., W_l^t)^T \hat{y}^*(x^*, W_1^t, ..., W_l^t) - \mathbb{E}_q(y^*|x^*)(y^*)^T \mathbb{E}_q(y^*|x^*)(y^*) \quad (1)$$

which equals the sample variance of T stochastic forward passes through the neural network plus the inverse model precision τ, which is a constant that is computable. As a result, dropout is not only useful during training but also

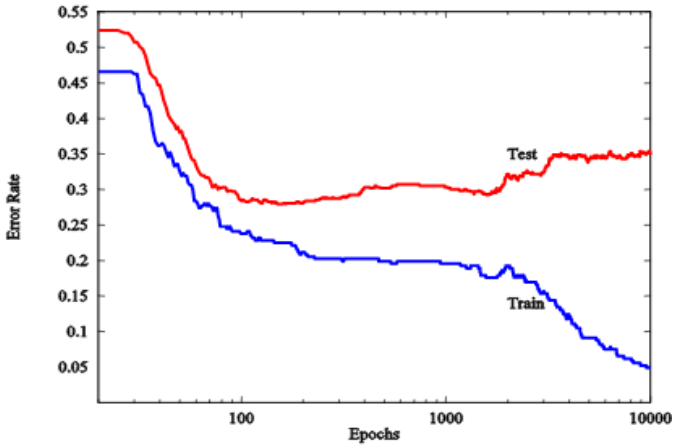


Figure 1: Phenomena of Over-fitting

for testing.
Batch normalization [2] is another option for regularization. It's based upon the idea that learning is easier when the distribution of the inputs is fixed. As it has been proved that the network training converges faster if the inputs are normalized and decorrelated [4, 5]. Thus, we should try to fix the activation distribution of all intermediate layers since the following layer can be seen as a network which is built upon the activations of the current layer.

3.1 Dropout

The dropout technique relies on the idea of introducing stochasticity into neural network by dropping random units out. In other terms, when training with dropout, a randomly selected subset of activations are set to zero within each layer. It can be seen as a network sampling from the full deep neural architecture as it is illustrated in figure 2. Algorithmically :

- During each training iteration, drop each unit in a layer out with probability p (usually set to 0.5) ,which means randomly dropping half of the units in the units in the current layer).
- During testing, multiply all the outgoing weights of the units by p and make final prediction using all units.

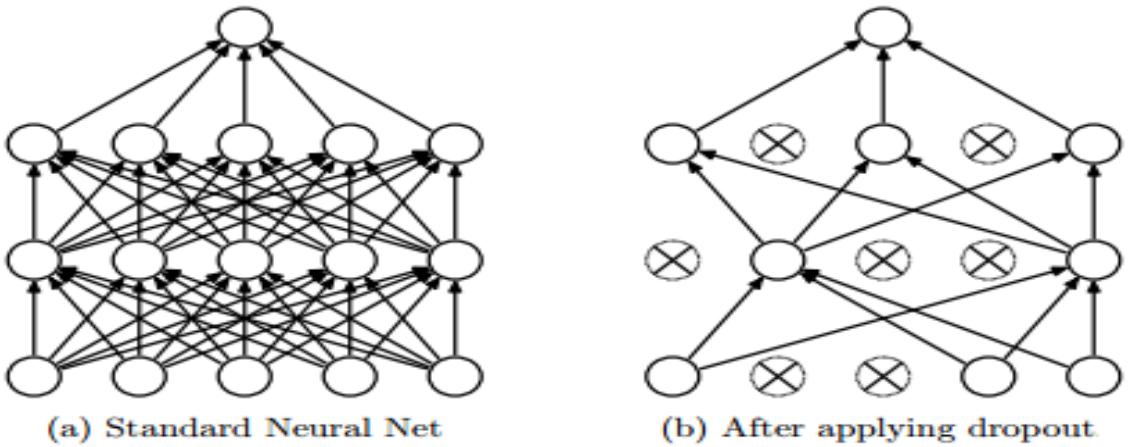
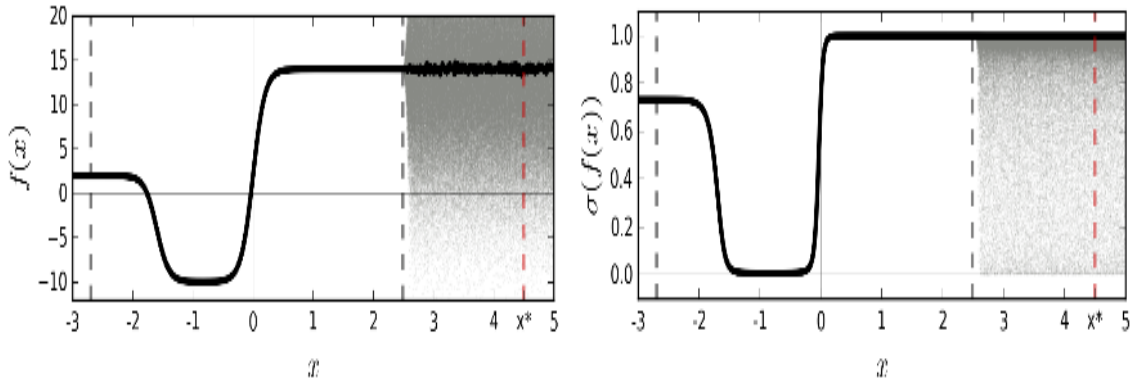


Figure 2: Dropout model

3.1.1 Dropout as a bayesian approximation

Deep learning architectures don't capture uncertainty of the models. In other term, we can't show what are our models know and what they don't know. We can ask the following question : to what extent our model is sure about the prediction ?
For instance, in classification, predictive probabilities obtained at the end (softmax output) are erroneously interpreted as model confidence [1]. A model can be uncertain in its prediction even with a high softmax output as illustrated in figure 3.



(a) Arbitrary function $f(\mathbf{x})$ as a function of data $\mathbf{x}$ (softmax *input*) (b) $\sigma(f(\mathbf{x}))$ as a function of data $\mathbf{x}$ (softmax *output*)

Figure 3: A sketch of softmax input and output for an idealized binary classification problem. Training data is given between the dashed gray lines. Function point estimate is shown with a solid line. Function uncertainty is shown with a shaded area. Marked with a dashed red line is a point x^* far from the training data. Ignoring function uncertainty, point x^* is classified as class 1 with probability 1. Passing a point estimate of a function (solid line a) through a softmax (solid line b) results in the extrapolation with unjustified high confidence for points far from the training data. However passing the distribution (shared area a) through a softmax (shaded area b) better reflects classification uncertainty far from the training data

In practice measuring the uncertainty of a model is important. For instance, if a model might return a result with high uncertainty, we might decide to pass the input to a human for classification (nuclear power plan with a system responsible for critical infrastructure) [1].
Bayesian rule can reason about model uncertainty but it's intractable in practice, it's computationally expensive. However, deep learning models(arbitrary depth and non-linearities) with dropout is mathematically equivalent to an approximation of a Gaussian process.
The dropout objective function minimizes the $Kullback - Leibler$ divergence between the approximate distribution and the posterior of a deep Gaussian process

$$KL(q_\theta(\omega)||P(\omega|X,Y)) \tag{2}$$

with ω parameters of the model.such that the posterior

$$P(\omega|X,Y) = \frac{P(Y|\omega,X)P(\omega)}{P(Y|X)} \tag{3}$$

and the approximate distribution of

$$P(\omega|X,Y) \tag{4}$$

is $q_\theta(\omega)$ which is defined as $W_i = M_i.diag([z_{i,j}]_{j=1}^{K_i})$ where $z_{i,j} = \text{Bernouli}(p_i)$. Given some probabilities p_i and matrices M_i as variational parameters. The binary variable $z_{i,j} = 0$ corresponds then to unit j in layer $i-1$ being dropped out as an input to layer i. The variational distribution $q(\omega)$ is highly multi-modal, inducing strong joint correlations over the rows of the matrices W_i (which correspond to the frequencies in sparse spectrum Gaussian process approximation) [1]. Minimizing Kullback-Leibler in equation 2 is identical to minimize

$$l(\theta) = -\int q_\theta(\omega) log P(Y|X,\omega) dw + KL(q_\theta(\omega)||p(\omega)) \tag{5}$$

such that $P(Y|X,\omega)$ is the likelihood and $p(\omega)$ is the prior. However the posterior $P(\omega|X,Y)$ term in the predictive distribution given new input x^* is intractable

$$P(y^*|x^*,X,Y) = \int P(y^*|x^*,\omega)P(\omega|X,Y)d\omega \tag{6}$$

Then we can approximate this predictive distribution by

$$\int P(y^*|x^*,\omega)q_\theta(\omega)d\omega \tag{7}$$

The link between deep neural network with dropout and bayesian models can be expressed by the fact that we put a prior distribution over the weights

$$p(W_{ik}) \propto \exp^{-\frac{1}{2}W_{ik}^T W_{ik}} \tag{8}$$

then the network output will be a random variable

$$f(x,\omega) = W_L\sigma(...W_2\sigma(W_1x + b_1)..) \tag{9}$$

for which we apply a softmax such that in context of classification we have

$$P(y|x,\omega) = softmax(f(x,\omega)) \tag{10}$$

But the core problem is how to evaluate the posterior $P(\omega|X,Y)$?
The integral of the equation 5 can be approximated with Monte Carlo integration $\hat{\omega} = q_\theta(\omega)$ then equation 5 becomes

$$\hat{l}(\theta) = -logP(Y|X,\hat{\omega}) + KL(q_\theta(\omega)||p(\omega)) \tag{11}$$

[1] shows that $l(\hat{\theta}) = l(\theta)$ converge to the same optima.
Thus, implementing the approximate inference is the same as implementing dropout in neural network where q_θ at the test time propagates the mean $\mathbb{E}(W_i) = p_iM_i$. q_θ has strong correlations between function frequencies and it's independent across output dimensions. Finally, in this way we can combine deep learning models with bayesian technique in practical way as it's developed in the equations above and get an uncertainty estimate in the network.

3.2 Batch normalization

In batch normalization, after every batch, the output of all neurons is normalized to zero-mean and unit variance. It makes the network significantly more robust to bad initialization. So, during each training iteration, we only observe a mini-batch, which prohibits computing the normalization exactly because it's computationally expensive and require access to the entire training set. Thus, the idea is treat the scaling factor γ and bias β as trainable parameters. The batch normalization module is depicted in figure 4

Mini-batch mean: $$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

Mini-batch variance: $$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Normalize: $$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

Scale and shift: $$y_i \leftarrow \gamma\widehat{x}_i + \beta$$

Figure 4: batch normalization steps

It can be shown [2] that the cost function is differentiable with respect to γ and β.
Batch normalization allows much higher learning rate, less careful initialization, and acts as a regularizer. The internal covariate shift is responsible of the slowness in training deep network. Covariate shift refers to the change in the input distribution to a learning system. The input to each layer is affected by parameters in all the input layers. So even small changes to the network get amplified down the network. This leads to change in the input distribution to internal layers of the deep network. In [2], it alleviates this issue using the batch normalization . It is well established that networks converge faster if the inputs have been whitened (example zero mean, unit variances) and are decorrelated and internal covariate shift leads to just the opposite [2].
The problem that deep network encounter is the different distribution of training and testing set and the activation functions reach soon their saturated regime. For instance, we consider a single node which takes a single dimensional input and applies a learned affine function and then passes that to a sigmoid non-linearity. It happens, that the input to a non-linearity is in its saturated regime. As a result, the output of the non-linearity becomes useless because the output is going to be always the same and the gradient passing back is going to be very small. However, if the input is produced by other layers ,the parameters of the model keep changing, the input to the affine transform keep changing over time, thus it's difficult to cope with adjusting the parameter (a and b) of the affine transform $(ax + b)$ in order to keep the non-linearity functioning (not in it's saturated regime). So, in order to mitigates the effect of changing inputs distribution, we can think about careful initialization of weight such as suggested by (Bengio,glorot 2013), setting small learning rates which prevent the distribution to non-linearity from drifting too fast and use ReLu to avoid saturated regime.

Deep neural networks suffer from generalization when test and training set have different distribution this is why some domain adaptation is required to adjust the parameters of the network. Taking two stacked activation functions in the network,their parameters keep changing since their inputs distribution is changing. As a consequence, continuous domain adaption is required to make sure that the network learns. Unfortunately, it takes time and its slows the training of the network.
In order to circumvent covariate shift, they [2] normalized each activation using its expectation and variance as follow

$$x \to \frac{x - E[x]}{\sqrt{Var[x]}} \tag{12}$$

However, variance and expectation depends on the model parameters. So, normalization should participate in gradient optimization such that $\frac{\partial E[x]}{\partial \theta}$ and $\frac{\partial Var[x]}{\partial \theta}$. Doing that, we can use mini-batch statistics to approximate the statistics over the entire data. In figure 4 (γ : standard deviation and β : the mean), we finally scale and shift which ensures that batch normalization don't reduce the expressivity of the model and be able to represent the density. Thus, we are able to recover the original input with some approximation. In this way, we reduce the internal Covariate shift.
During the inference with normalization we use mini-batches because we don't want the output of an example to depend on other examples. We replace the mini-batch statistics by population statistics. In other term, having trained a model with batch normalization, we compute the expectation and the variance of each activation we are normalizing and at the inference time, we'll normalize by subtracting by the population expectation and divide by standard deviation of the population as depicted in figure 5

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \quad \Rightarrow \quad \widehat{x} \leftarrow \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

Figure 5: inference with batch normalization

When we combine batch normalization with convolutional networks, we don't do only normalization on the examples of mini-batch but also over the nodes. In other terms, when batch normalization is added to convolutional layer, the layer still remains convolutional. We add batch normalization between the affine transform and non-linearity which gives us a likely symmetric distribution. It turns out, applying normalization in convolutional network in this way, the resulting transform ends up being invariant to the scale of the weight matrix W.

4 Experiments and results

4.1 Dropout

In order to asses their model and its properties of uncertainty estimates, they tested it on different tasks : regression, classification, predictive performance and reinforcement learning.
In **regression** tasks, they trained different models on the CO_2 dataset where they used 4 or 5 hidden layers and 1024 hidden units with either RELU non-linearities or TanH non-linearities in each network and use dropout probabilities of either 0.1 or 0.2. The results are depicted in figure 6. In 6 (a) shows that for standard dropout without assessing the model uncertainty using 5 layer RELU. This model predicts value 0 for point x^* with a high confidence. In counterpart,(b) shows the results obtained from a Gaussian process with a squared exponential covariance function. However, monte carlo dropout 6 (c) represents this uncertainty, it declares that the predictive value might be 0 but the model is uncertain to a certain order. It's obvious because dropout's uncertainty draws its properties from the Gaussian process in which different covariance functions correspond to different uncertainty estimates [1]. We noticed also that the models in 6 have an incorrect predictive mean, the increased standard deviation expresses the models' uncertainty about the point.
We noticed that the uncertainty is increasing far from the data for the RELU model, whereas for the TanH model it stays bounded because TanH saturates whereas RELU does not.
In **reinforcement** learning task, agent tries to learn to avoid transitioning into states with low rewards, and to pick actions that lead to better states instead. So, uncertainty information in this context allows to an agent to decide when to exploit rewards and when to explore the environment. Relying on the uncertainty estimates given by a dropout Q-network we can use Thompson sampling (*Thompson*, 1933) in order to converge faster than epsilon greedy. To do so, they simulated an agent (with 5 actions) in 2D world with 9 eyes pointing in different angles ahead, where each eye can sense a single pixel intensity of 3 colors. The environment consists of red circles which give the agent a positive reward and green circles which result in a negative reward. The agent is rewarded for not looking at white

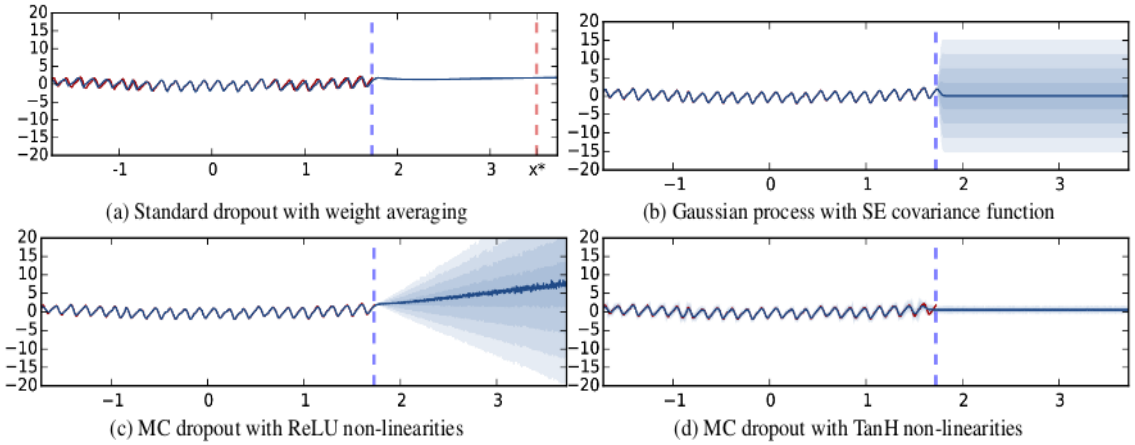


Figure 6: Predictive mean and uncertainties on the Mauna Loa CO 2 concentrations dataset, for various models.

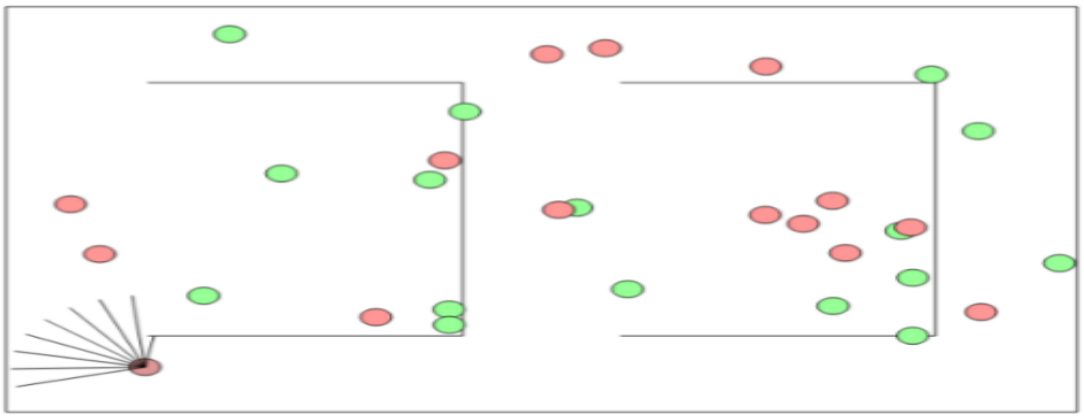


Figure 7: reinforcement learning problem used in the experiments. The agent is in the lower left part of the maze,facing north-west.

walls and walking in straight line as it's depicted in figure 7. In figure 8 , a log plot of the average reward obtained by epsilon greedy and the Thompson sampling using uncertainty (to make use of the dropout Q-network's uncertainty estimates) is shown .We Noticed that Thompson sampling gets reward better than epsilon greedy which takes more batches to achieve the same performance of Thompson sampling. However, Thompson sampling stops improving after 1K batches because it still sample random moves, whereas epsilon greedy only exploits it at this stage [1]

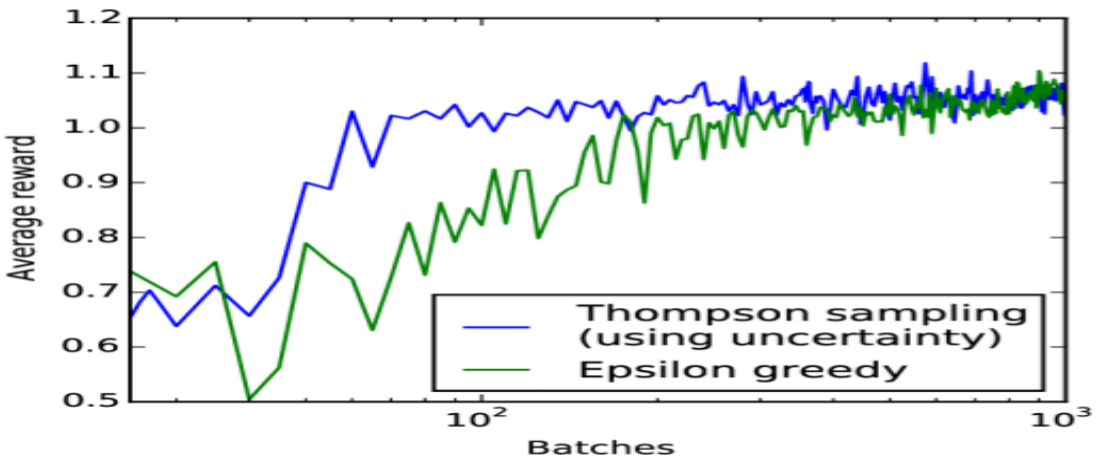


Figure 8: Log plot of average reward obtained by both epsilon greedy and Thompson sampling (in blue), as a function of the number of batches

4.2 Batch normalization

In order to evaluate the effect of internal covariate shift on training and the ability of batch normalization to combat it, they considered a neural network of 3 fully-connected hidden layers with 100 activations each with sigmoid non-linearity and softmax output on MNIST handwritten digits. Figure 9 on the **left** shows the evolution of the distributions of the inputs to a typical sigmoid as the training proceeds. We notice that its distribution change significantly over time, both in their mean and the variance.

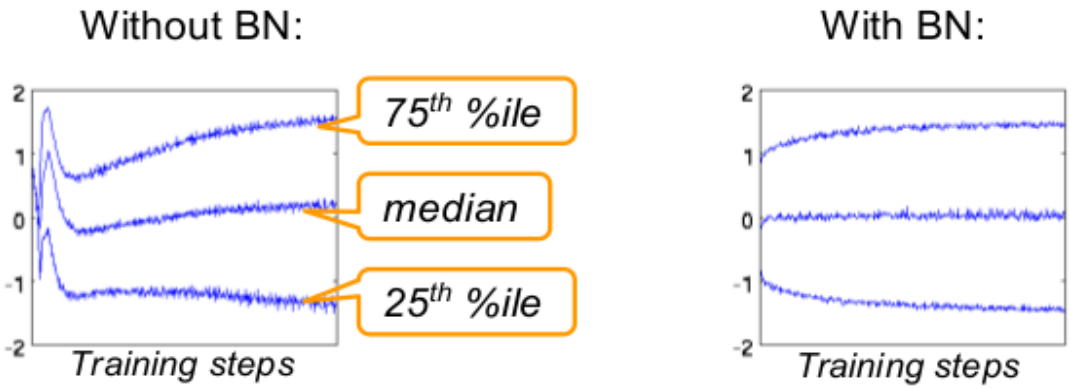


Figure 9: The evolution of input distributions to a typical sigmoid , over the course of training (100k steps)

Figure 9 on the **right**, shows that when we add batch normalization, the distributions becomes more stable. Thus, covariate shift is reduced by batch normalization. Figure 10 confirms that neural network with batch normalization performs better in terms of the accuracy of the model.

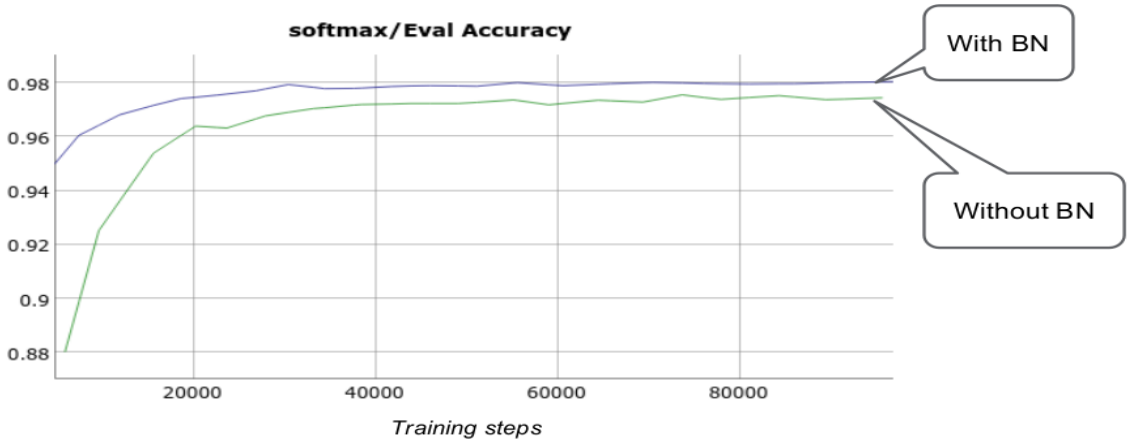


Figure 10: The test accuracy of the MNIST network trained with and without Batch Normalization

A second experiment on large scale computer vision task : ImageNet classification (Russalovsky et al.,2014)) using a variant of inception (CNN uses convolutions kernels of multiple sizes)network (Szegedy et al.,2014) is done by applying batch normalization (30% extra computation cost) at each convolutional layer. By just adding batch normalization to inception as depicted in 11, we get the same accuracy as the baseline model (state of art performance) but with batch normalization this results is reached by more than half less steps than in the baseline. We notice that with/ without batch normalization the accuracy rate remains the same, but with batch normalization the learning rates is higher, it's increased by a factor of 30. Furthermore, when using batch normalization and removing dropout, we improve validation accuracy. Thus, batch normalization acts as regularizer.

5 Perspective and open problems

Deep neural network is an heuristic-based path. It's not theoretically grounded. Thus, we don't have guarantees about generalization. Each architecture is specific to a given problem. A small change in parameters can drastically change the results. Deep neural networks requires a billion of training examples and are computationally expensive since we have a billion of parameter to fine-tune.

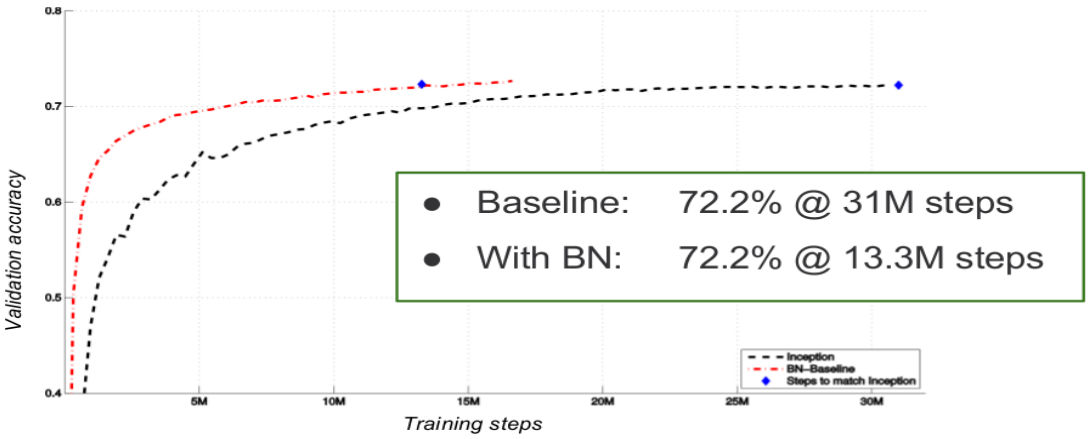


Figure 11: Inception deep convolutional network with and without batch normalization

In order to build a robust models which are insensible to perturbation (noise) and to transformation, we need to be able to built invariant models. With a mathematically grounded framework, we can get rid of fine-tuning a set of hyper-parameters such as :

- Dropout : Because it's difficult to find the appropriate probability to drop
- The nature of connexion between layers : fully-connected, sparse, dropout, dropconnect
- The number of hidden layers and hidden units

Recently, we noticed that mathematical advances is unveiling the nature of deep network. In [7], Stepahane Mallat gives a mathematical framework to explain the hierarchy, the nature of interaction between linear and non-linear operators and how to build group invariants (to translation, rotation, elastic deformation) from group theory perspective. Group invariants are important to build robust architecture (insensible to hyper-parameter) without a loss of generality. Moreover, following the state of art in optimization Yann Olliver and al. comes up with invariants principles [8, 9] for covariance matrix adaptation evolutionary strategy relying on a Riemmanian Manifold which is equipped with a fisher metric (the only invariant metric in this statistical manifold). This progress alleviates the need of computing the inverse of the Hessian matrix (so expensive in high dimension), it can be replaced by the Fisher matrix (covariance matrix) that can be approximated using monte carlo integration. Using this kind of mathematically grounded approaches, we can get rid of dropout and do a strong optimization.
Extend the following studies can be tempting to come up with a theoretical founding of deep learning [10, 11, 12] from group theory and Probabilistic models standpoints . Besides, many progress from statistical physics of disordered systems have been made to explain the nature of deep architectures from Hamiltonian mechanics, Ising model, chaos theory and the violation of the second law of thermodynamic [13, 14].
In conclusion we reckon that the study of group theory, Riemannian geometry, information theory, probabilistic models, complexity theory, functional analysis in high dimensional and statistical physics of disordered system jointly can lead us toward a theory of deep learning.

References

[1] Yarin Gal, Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. ICML 2016

[2] Sergey Ioffe, Christian Szegedy.Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.2015

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014.

[4] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller, "Neural networks-tricks of the trade," Springer Lecture Notes in Computer Sciences, vol. 1524, no. 5-50, p. 6, 1998.

[5] S. Wiesler and H. Ney, "A convergence analysis of log-linear training," in Advances in Neural Information Processing Systems, pp. 657–665, 2011.

[6] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in Advances in neural information processing systems, pp. 2654–2662, 2014.

[7] Stephane Mallat. Understanding deep convolutional networks. 2016

[8] Yann Ollivier, Ludovic Arnold, Anne Auger, Nikolaus Hansen. Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles.2011

[9] Yann Ollivier. Riemannian metrics for neural networks I: Feedforward networks. Information and inference, 2015.

[10] Ankit B. Patel, Tan Nguyen, Richard G. Baraniuk. A Probabilistic Theory of Deep Learning.2015

[11] Pankaj Mehta, David J. Schwab. An exact mapping between the Variational Renormalization Group and Deep Learning.2014

[12] Henry W. Lin , Max Tegmark. Why does deep and cheap learning work so well?.2016

[13] Madhu Advani, Subhaneil Lahiri, Surya Ganguli. Statistical mechanics of complex neural systems and high dimensional data.2013

[14] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics.2015