Regularization for Deep Learning

Changyou Chen

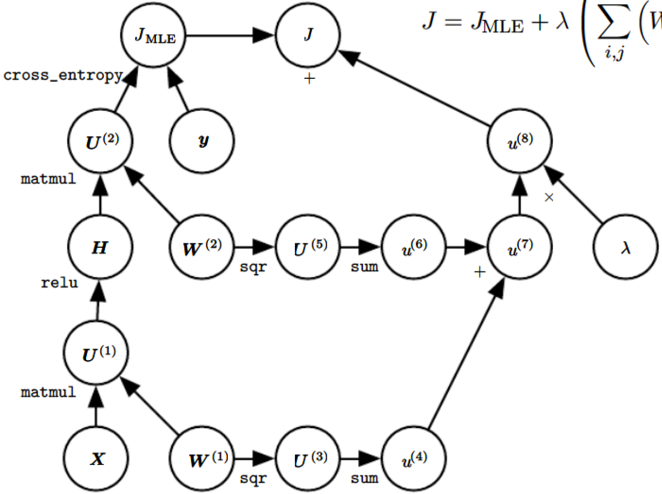
Department of Computer Science and Engineering
Universitpy at Buffalo, SUNY
changyou@buffalo.edu

Feburary 26, 2019

Example: Regularization in FNN Forward Propagation Graph

Total cost:

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} \left(W_{i,j}^{(1)} \right)^2 + \sum_{i,j} \left(W_{i,j}^{(2)} \right)^2 \right)$$



Definition

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Importance of Regularization

- Overly complex family does not necessarily include the target function, true data generating process, or even an approximation.
- Most deep learning applications are where true data generating process is outside the model family:
    - complex domains of images, audio sequences and text generation process may involve entire universe, can not be fully described by our model.
    - need to choose a model that best approximates the data.

Regularization Techniques

- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

Regularization Techniques

- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

Norm Penalty

1. Regularized objective function:

$$\tilde{J}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) = J(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) + \alpha \Omega(\boldsymbol{\theta}) \ ,$$

   where $\boldsymbol{\theta}$ denotes both weights **W** and biases **b**.

2. Different choices of the parameter norm Ω can result in different solutions preferred.
3. Typically no penalty for biases:
   - each bias controls only a single variable without data interaction, we do not induce too much variance on it.
4. Reasonable to use the same α for all weights to avoid expensive tuning.

L^2 parameter Regularization

$$\Omega(\theta) = \frac{1}{2} \|\mathbf{W}\|_2^2$$

1. Simplest and most common used.
2. Drives weights closer to the origin.
3. Called weight decay; some communities also called ridge regression or Tikhonov regularization.
4. Gradient:

$$\nabla_{\mathbf{W}} \tilde{J}(\mathbf{W}; \mathbf{X}, \mathbf{y}) = {\color{red}\alpha\, \mathbf{W}} + \nabla_{\mathbf{W}} J(\mathbf{W}; \mathbf{X}, \mathbf{y})$$

5. Equivalent to MAP Bayesian estimation with Gaussian prior.

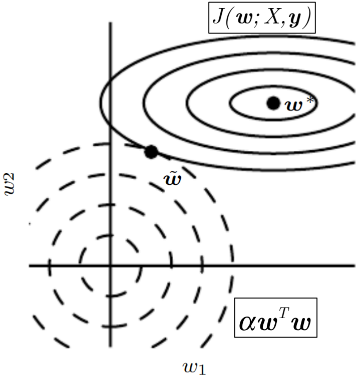


Figure: Choose the one that has the smallest L^2-norm.

L^1 parameter Regularization

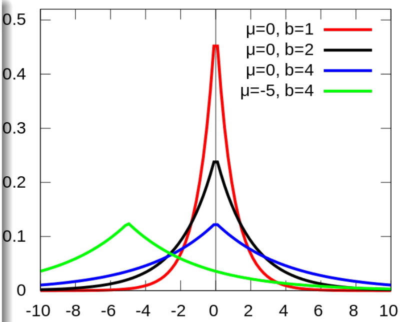
$$\Omega(\boldsymbol{\theta}) = \|\mathbf{W}\|_1 = \sum_i |W_i|$$

Encourages sparsity, equivalent to MAP Bayesian estimation with a Laplace prior.

Laplace Distribution

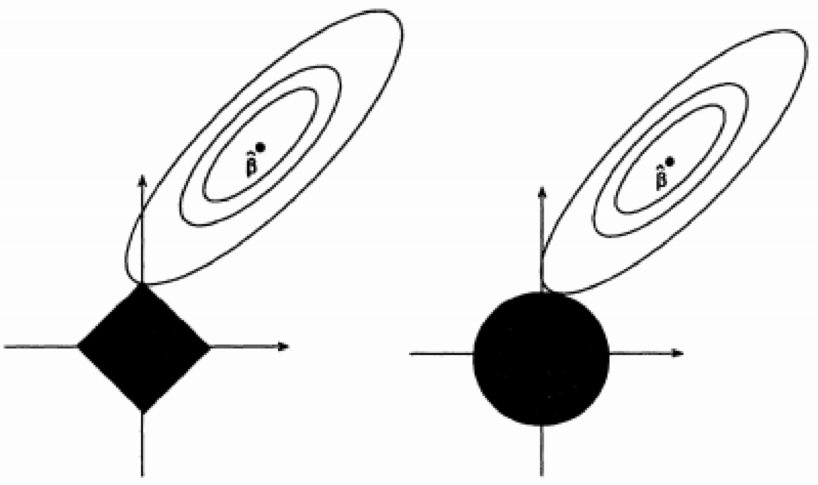
A random variable x has a Laplace distribution with parameters (μ, b) if its probability density function is

$$p(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$
$$= \frac{1}{2b} \begin{cases} \exp\left(-\frac{x-\mu}{b}\right), & \text{if } x \le \mu \\ \exp\left(-\frac{\mu-x}{b}\right), & \text{if } x > \mu \end{cases}$$



Why is L^1 Sparse?

L^1 regularizer has a better chance to touch the objective function at zero!



Regularization Techniques

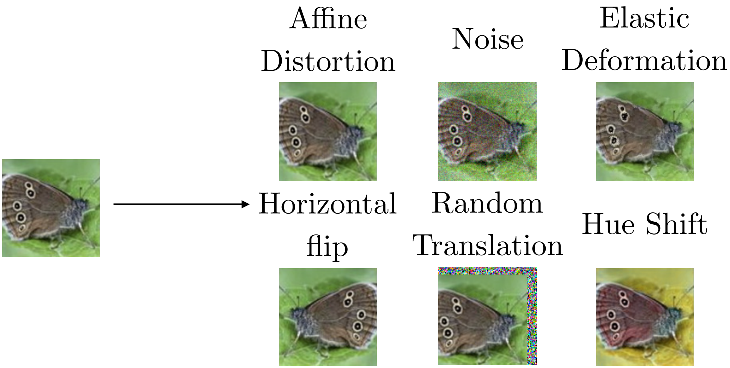
- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

More data is better

1. Best way to make an ML model to generalize better is to train it on more data.
2. In practice, the amount of data is limited.
3. Get around the problem by creating fake data.
4. For some ML tasks it is straightforward to create fake data:
   - For classification: generate new samples $(\mathbf{x}, y)$ just by transforming inputs $\mathbf{x}$.
   - This approach is not easily generalized to other problems such as density estimation problem, because it is not possible to generate new data without solving density estimation.
   - Generative adversarial net (GAN) is also an effective way for data augmentation.

Injecting noise

1. Injecting noise into the input of a neural network can be seen as data augmentation.
2. To improve robustness of neural networks, train them with random noise applied to their inputs, *e.g.*, denoising autoencoder.
3. Noise can also be applied to hidden units, *e.g.*, Dropout.



Regularization Techniques

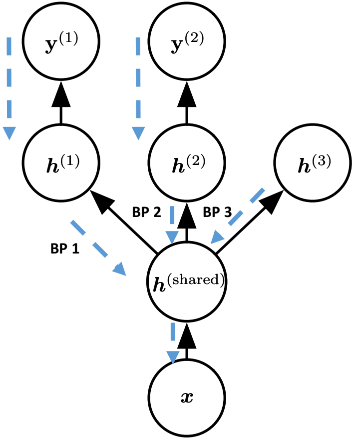
- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

Sharing Parameters over Tasks

- Multi-task learning is a way to improve generalization by pooling the examples out of several tasks:
  - can be seen as some kind of data augmentation.

Example of Multi-task Learning

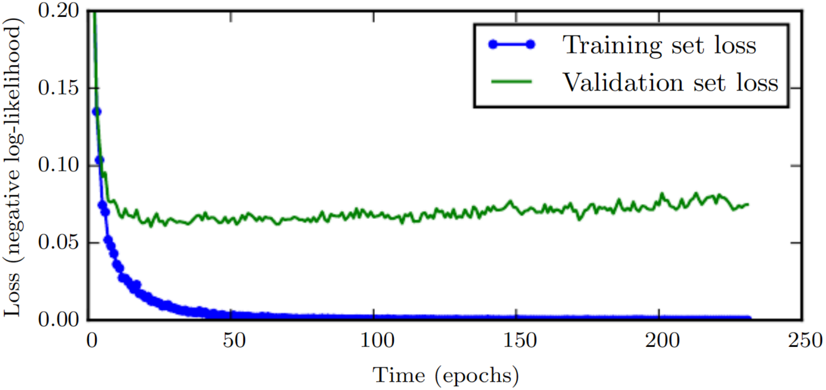
1. Common input but different target random variables:
   - task-specific parameters $h^{(1)}$ and $h^{(2)}$ can be learned on top of those yielding a shared representation $h^{(\text{shared})}$.
2. In the unsupervised learning context, some of the top level factors are associated with no outputs, *e.g.*, $h^{(3)}$.
3. These are factors that explain some of the input variations but not relevant for predicting $h^{(1)}$, $h^{(2)}$.



Regularization Techniques

- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

Learning Curves



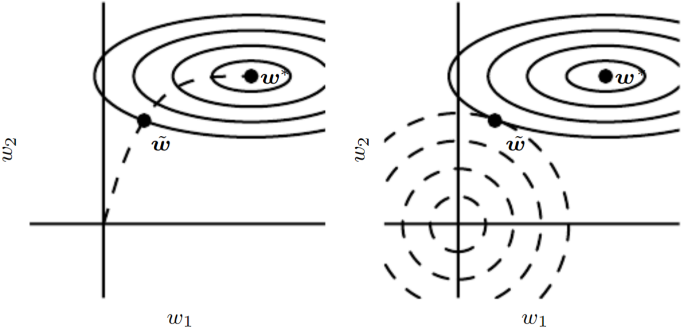
Shows how negative log-likelihood loss changes over time (indicated as no. of Training iterations over the data set, or epochs).
In this example, we train a maxout network on MNIST.
Training objectve decreases consistently over time, but validation set average Loss eventually begins to increase again for ming an asymmetric U shape 4

Early Stopping: Saving Parameters

1. We can thus obtain a model with better validation set error (and typically better test error) by saving the one at the point of time with the lowest validation set error.
2. Every time the error on the validation set improves, we store a copy of the model parameters.
3. When the training algorithm terminates, we return these parameters, rather than the latest one.

Very often used in practice in deep learning!

Early Stopping vs. L^2 Regularization



Two weights, Solid contour lines: contours of negative log-likelihood

Left: dashed lines indicates trajectory of SGD. Rather than stopping at point $\boldsymbol{w}^*$ that minimizes cost, early stopping results in an earlier point in trajectory

Right: dashed circles indicate contours of L^2 penalty which causes the minimum of the total cost to lie nearer the origin than the minimum of the the unregularized cost

Regularization Techniques

- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

Parameter Dependency

1. We want to model dependencies between model parameters:
   - Parameter tying: certain parameters should be close to one another, *e.g.*, two models map the input to two different but related outputs, we want the two models close.
   - Parameter sharing: forces sets of parameters to be equal
     - In an CNN, significant reduction in the memory footprint of a model.
2. L^2 penalty for parameter tying:
   - just add an additional regularized term for two model parameters $\mathbf{w}^A$ and $\mathbf{w}^B$:

$$\Omega\left(\mathbf{w}^A, \mathbf{w}^B\right) = \left\|\mathbf{w}^A - \mathbf{w}^B\right\|^2$$

3. Represent the loss with the same parameter for parameter sharing.

Regularization Techniques

- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

What is Bagging?

1. Short form of “bootstrap aggregrating”.
2. A technique for reducing generalization error by combining several models:
   - Different models are trained separately.
   - All the models vote on the output for test examples.
3. This strategy for machine learning is referred to as model averaging.
   - Techniques employing this strategy are known as ensemble methods.

Why does Bagging Work?

1. Train k regression models separately, each with squared error ϵ_i.
2. Assume $\mathbb{E}[\epsilon_i^2] = v$, $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
3. It can be shown that the variance of the average error, assuming independence ($c = 0$) and $\mathbb{E}[\epsilon_i] = 0$, decreases linearly with ensemble size.

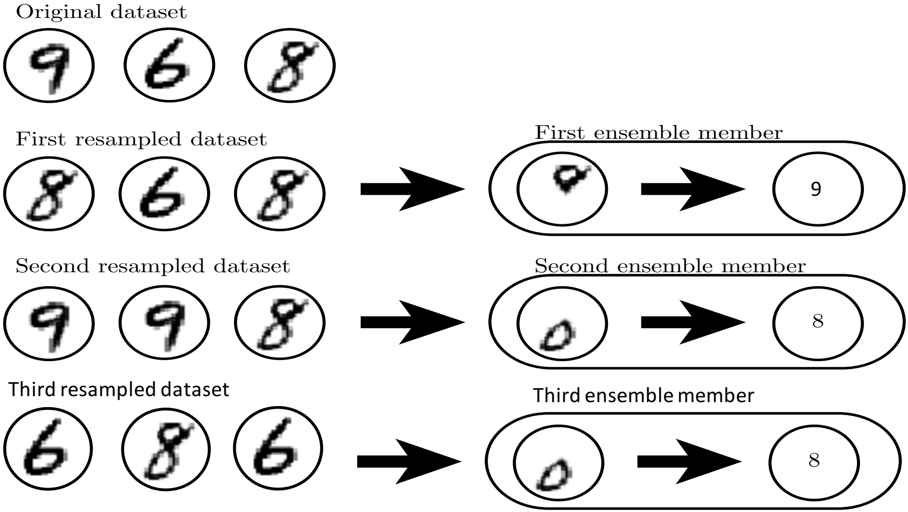
Why does Bagging Work?

1. Train k regression models separately, each with squared error ϵ_i.
2. Assume $\mathbb{E}[\epsilon_i^2] = v$, $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
3. It can be shown that the variance of the average error, assuming independence ($c = 0$) and $\mathbb{E}[\epsilon_i] = 0$, decreases linearly with ensemble size.

$$
\begin{aligned}
\mathrm{Var}(\frac{1}{k}\sum_i \epsilon_i) &= \mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] - \left(\mathbb{E}\left[\frac{1}{k}\sum_i \epsilon_i\right]\right)^2 \\
&= \mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j\right)\right] \\
&= \frac{1}{k}v + \frac{k-1}{k}c = \frac{1}{k}v
\end{aligned}
$$

Bagging

- To ensure independence, we resample data sets independently for each model for training.



Regularization Techniques

- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

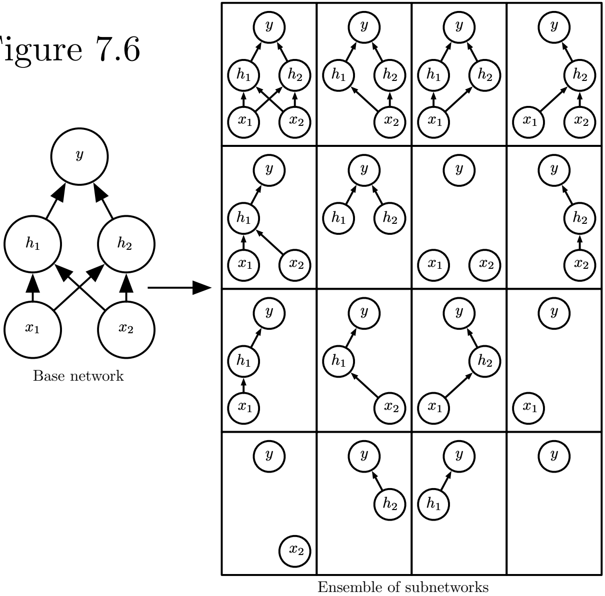
What is Dropout?

1. An inexpensive but powerful method of regularizing a broad family of models:
   - A practical way of bagging applied to neural networks with weights sharing.
   - Practical to apply bagging to very many large neural networks.
2. It trains an ensemble of all subnetworks formed by removing non-output units from an underlying base network.

Dropout

- Resulting in many networks with no path from input to output.
- Problem insignificant with large networks.
- All subnetworks share the weights.

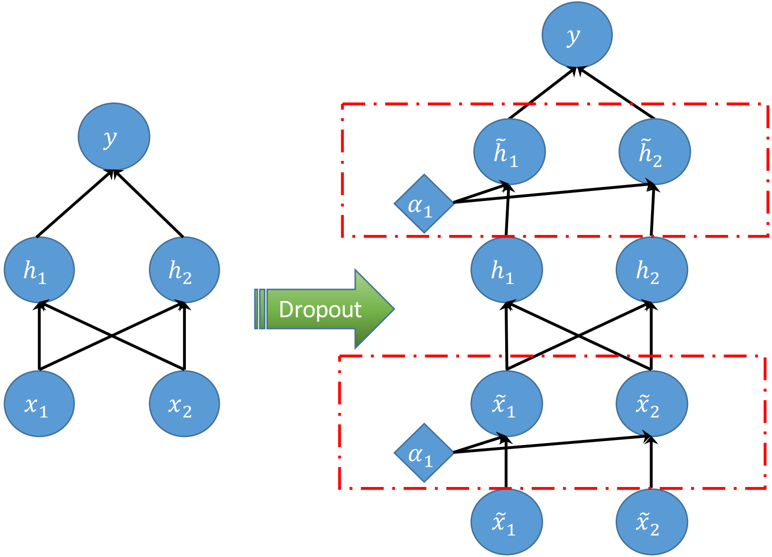
Figure 7.6



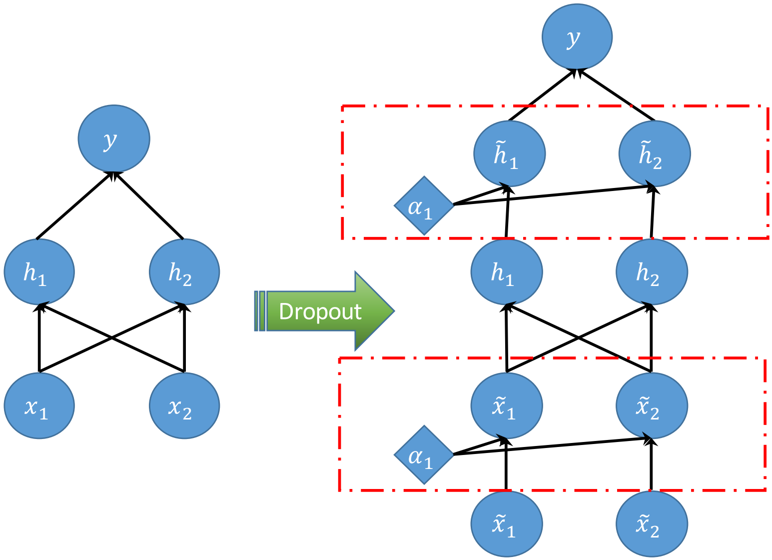
Dropout Training

1. Use minibatch based learning algorithm that takes small steps such as SGD.
2. At each step, randomly sample a binary mask μ:
   - probability of including a unit is a hyperparameter, *e.g.*, 0.5 for hidden units and 0.8 for input units.
3. Node-wise multiplication of the binary mask with the original network.
4. Run forward and backward propagation as usual on the resulting network.

BP with Dropout: Implemented as Additional Layers



BP with Dropout: Implemented as Additional Layers



How to backpropagate gradients in the Dropout layer?

Prediction

1. Submodel defined by mask vector μ defines a probability distribution $p(y|\mathbf{x}, \mu)$.
2. Arithmetic mean over all masks is: $\sum_\mu p(y|\mathbf{x}, \mu)p(\mu)$, where $p(\mu)$ is the probability of generating the sample mask μ.
3. Intractable to evaluate due to an exponential number of masks.
4. Use geometric mean rather than arithmetic mean for prediction: $\left(\prod_\mu p(y|\mathbf{x}, \mu)\right)^{1/2^d}$.

Prediction: Geometric Mean Approximation

Geometric mean inequality

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} \leq \frac{\sum_{i=1}^{n} x_i}{n}$$

- Given two numbers, x and y, their geometric mean is $\sqrt{xy}$.
- Approximate it as the mean of the coefficients and the mean of the exponents:

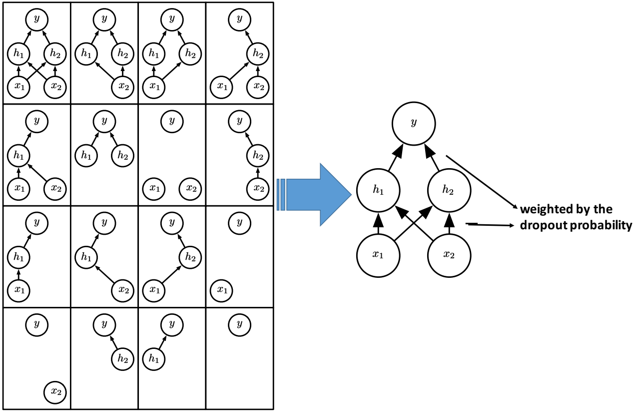
Example

Let $x = 4 \times 10^2$, $y = 7 \times 10^6$. Then approximate $\sqrt{xy}$ as:

$$\sqrt{xy} = 5.29 \times 10^4 \approx \frac{4+7}{2} \times 10^{(2+6)/2} = 5.5 \times 10^4 \ .$$

Prediction: Geometric Mean Approximation

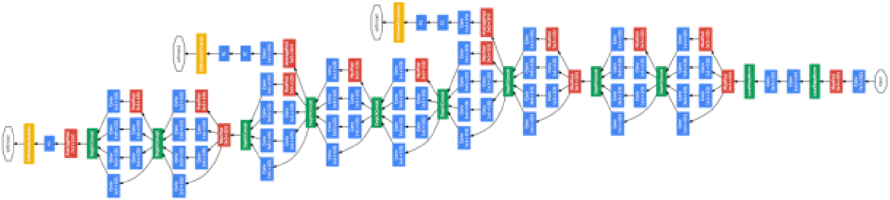
- Approximate the geometric mean of Dropout by evaluating $p(y|\mathbf{x}, \mu)$ in one model: the model without dropout, but with the weights going out of unit *i* multiplied by the probability of including unit *i*:
  - motivation is to capture the right expected value of the output from that unit.



Batch Normalization

- Norm regularization.
- Data augmentation.
- Multi-task learning.
- Early stopping.
- Parameter sharing.
- Bagging.
- Dropout.
- Batch normalization.

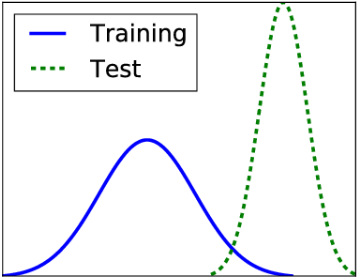
Internal Covariance Shift



- Change of distributions in activation across layers.
- Change in optimal learning rate $\Rightarrow$ need really small steps.

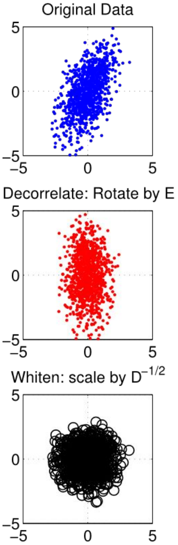
Covariate Shift

- Training and test input follow different distributions.
- But functional relation remains the same.



Solution 1: Decorrelation and Whitening

- Benefit:
  - Transform training and testing onto a space where they have same distribution.
- Issues:
  - Computationally expensive to calculate covariance matrices for every layer.
  - Not work for stochastic algorithms.



Solution 2: Batch Normalization

- Normalize distribution in each layer across each minibatch to $N(0, 1)$.
- Learn the scale and shift parameter.
- Parameters are differentiable via chain rule.

$$\mu_{\mathcal{B}} \to \frac{1}{m}\sum_i \mathbf{x}_i, \qquad \text{//mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \to \frac{1}{m}\sum_i (\mathbf{x}_i - \mu_{\mathcal{B}})^2, \qquad \text{//mini-batch variance}$$

$$\hat{\mathbf{x}}_i \to \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{//normalization}$$

$$\mathbf{y}_i \to \gamma \hat{\mathbf{x}}_i + \beta \triangleq \mathrm{BN}_{\gamma,\beta}(\mathbf{x}_i), \qquad \text{//scale and shift}$$

Solution 2: Batch Normalization

- Normalize distribution in each layer across each minibatch to $N(0, 1)$.
- Learn the scale and shift parameter.
- Parameters are differentiable via chain rule.

$$\mu_{\mathcal{B}} \rightarrow \frac{1}{m}\sum_i \mathbf{x}_i, \qquad \text{//mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \rightarrow \frac{1}{m}\sum_i (\mathbf{x}_i - \mu_{\mathcal{B}})^2, \qquad \text{//mini-batch variance}$$

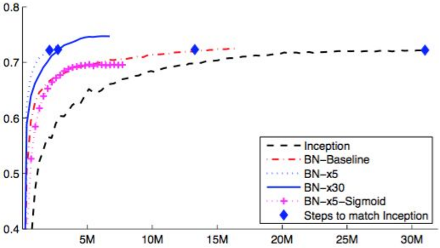
$$\hat{\mathbf{x}}_i \rightarrow \frac{\mathbf{x}_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{//normalization}$$

$$\mathbf{y}_i \rightarrow \gamma\hat{\mathbf{x}}_i + \beta \triangleq \mathrm{BN}_{\gamma,\beta}(\mathbf{x}_i), \qquad \text{//scale and shift}$$

- At test time, $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are running average of values seen during training.

Inception Net on ImageNet

- Faster convergence (30X).
- Similar accuracies.



Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Quiz: Why does Bagging Work?

1. Train k regression models separately, each with squared error ϵ_i.
2. Assume $\mathbb{E}[\epsilon_i^2] = v$, $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
3. **Prove**: The variance of the average error, assuming independence ($c = 0$) and $\mathbb{E}[\epsilon_i] = 0$, decreases linearly with ensemble size k.