04.10.2024

Statistical Methods in AI (CS7.403)

Lecture-16: CNN/Deep Learning - 2

Ravi Kiran (ravi.kiran@iiit.ac.in)

https://ravika.github.io

@vikataravi

Center for Visual Information Technology (CVIT)
IIIT Hyderabad

Regularization Strategies

1. Parameter Norm Penalties
2. Norm Penalties as Constrained Optimization
3. Regularization and Under-constrained Problems
4. Data Set Augmentation
5. Noise Robustness
6. Semi-supervised learning
7. Multi-task learning
8. Early Stopping
9. Parameter tying and parameter sharing
10. Sparse representations
11. Bagging and other ensemble methods
12. Dropout
13. Adversarial training
14. Tangent methods

Overfitting in Deep Neural Nets

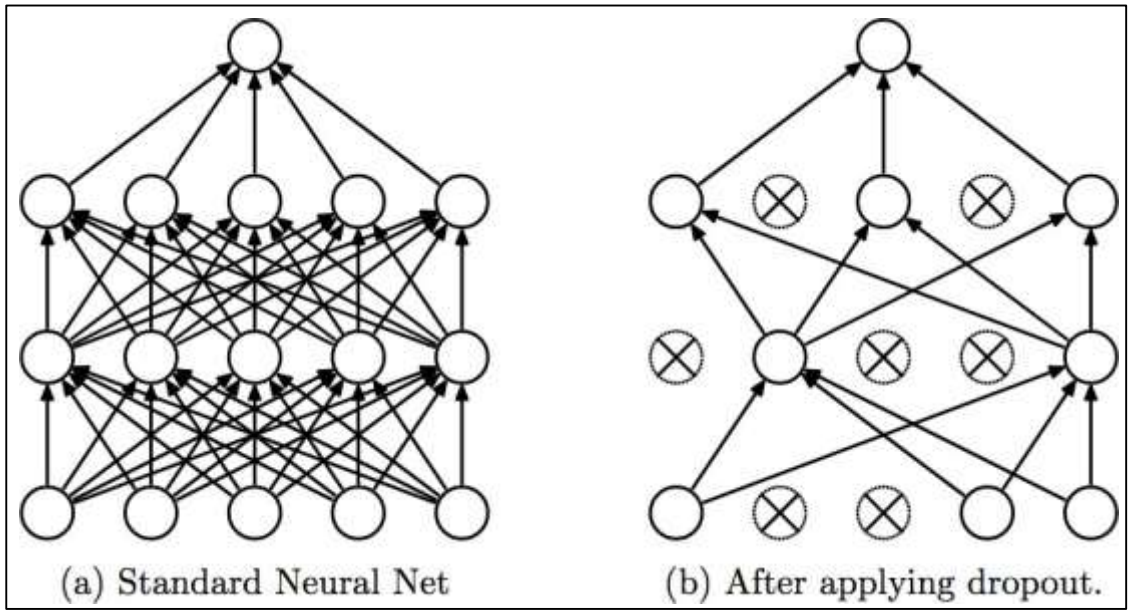
- Deep nets have many non-linear hidden layers
  - Making them very expressive to learn complicated relationships between inputs and outputs
  - But with limited training data, many complicated relationships will be the result of training noise
    - So they will exist in the training set and not in test set even if drawn from same distribution
- Many methods developed to reduce overfitting
  - Early stopping with a validation set
  - Weight penalties (L^1 and L^2 regularization)
  - Soft weight sharing

Regularization with unlimited computation

- Best way to regularize a fixed size model is:
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
    - This would be the Bayesian approach
- Dropout does this using considerably less computation
  - By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout Neural Net

- A simple way to prevent neural net overfitting



Drop hidden and visible units from net, i.e., temporarily remove it from the network with all input/output connections. Choice of units to drop is random, determined by a probability p, chosen by a validation set, or equal to 0.5

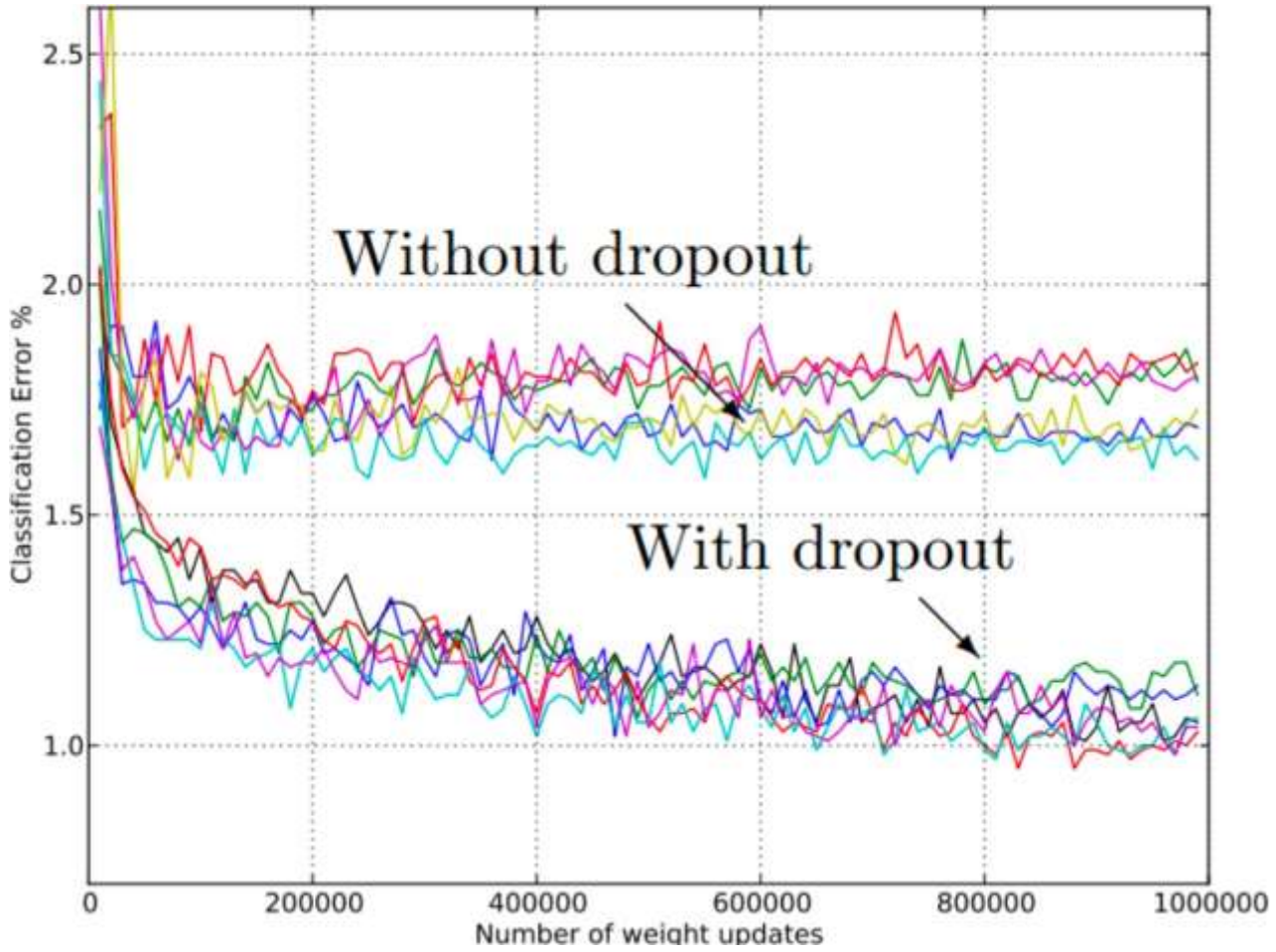
(a) A standard neural net with two hidden layers

(b) A thinned net produced by applying dropout, crossed units have been dropped

Bernoulli Distribution

$$f(x|p) = \begin{cases} p & x = 1 \\ 1-p & x = 0 \end{cases}$$

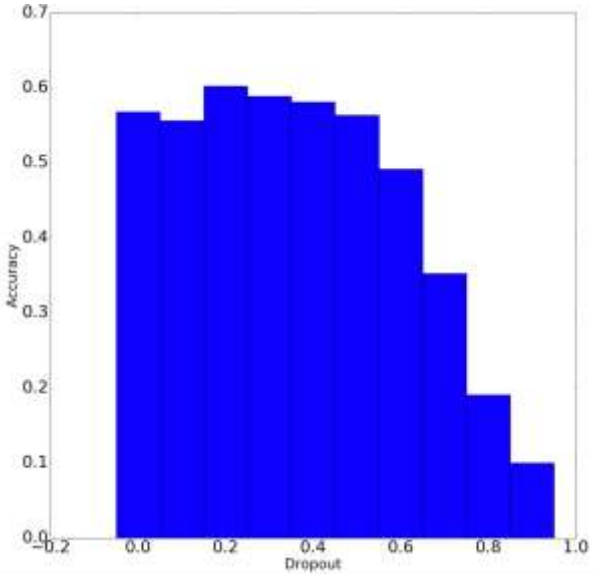
Performance with/without Dropout



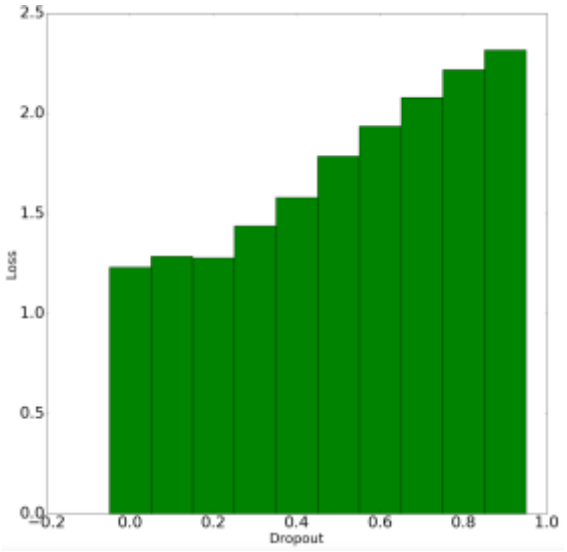
Dependence on p

Deep net in Keras
Validate on CIFAR-10 dataset
Network built had three convolution layers of size 64, 128 and 256 followed by two densely connected layers of size 512 and an output layer dense layer of size 10

Accuracy vs dropout

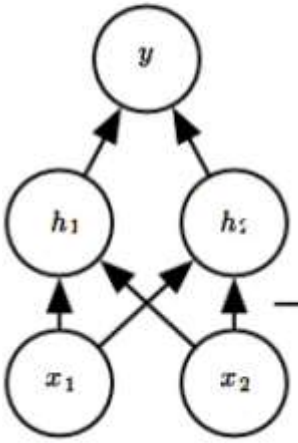


Loss vs dropout

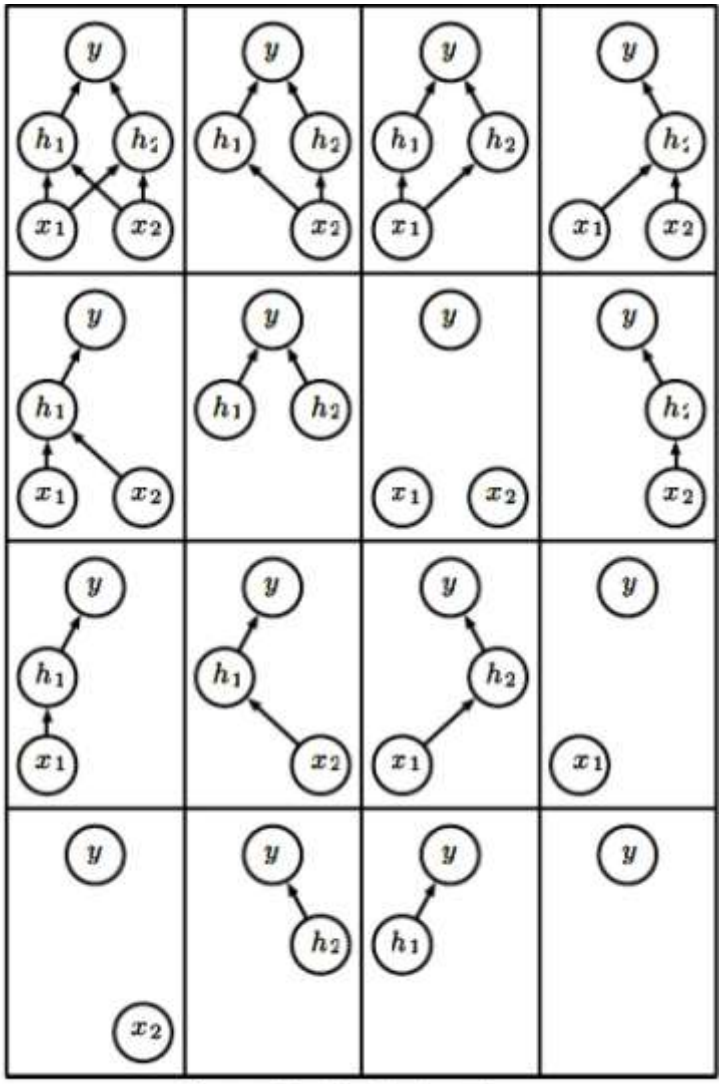


Dropout as an ensemble method

- Remove non-output units from base network.
- Remaining 4 units yield 16 networks



Base network



Ensemble of subnetworks

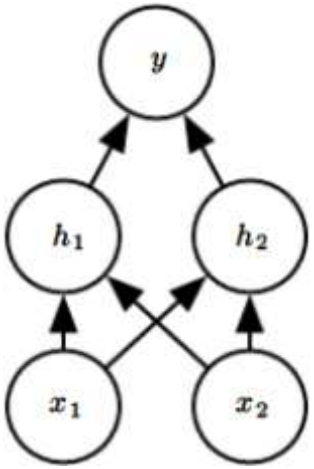
- Here many networks have no path from input to output
- Problem insignificant with large networks

Mask for dropout training

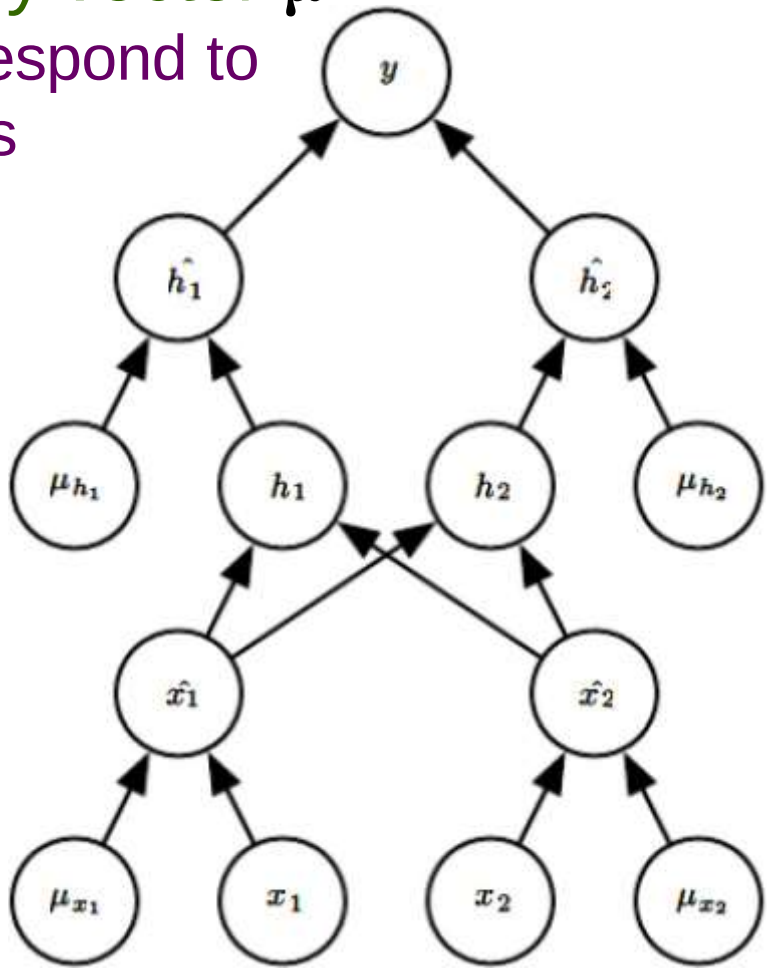
- To train with dropout we use minibatch based learning algorithm that takes small steps such as SGD
- At each step randomly sample a binary mask
  - Probability of including a unit is a hyperparameter
    - 0.5 for hidden units and 0.8 for input units
- We run forward & backward propagation as usual

Forward Propagation with dropout

Feed-forward network



- Network with binary vector $\boldsymbol{\mu}$ whose elements correspond to input and hidden units
- Elements of $\boldsymbol{\mu}$
- With probability of 1 being a hyperparameter
  - 0.5 for hidden
  - 0.8 for input
- Each unit is
  - Multiplied by corresponding mask



- Forward prop as usual
- Equivalent to randomly selecting one of the subnetworks of previous slide

Activations in Dropout layers

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$

Inference time: Dropout is disabled.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Motivation

- The range of values of raw training data often varies widely
  - Example: Has a loan: {0,1}
  - Value of car: $500-$100
- Optimization process are sensitive to normalization
  - E.g. Distance between two points by the Euclidean distance.
  - After, normalization, each feature contributes approximately proportionately to the final distance.
- In general, Gradient descent converges much faster with feature scaling than without it.
- Good practice for numerical stability for numerical calculations, and to avoid ill-conditioning when solving systems of equations.

Common normalizations

Two methods are usually used for rescaling or normalizing data:

- Scaling data all numeric variables to the range [0,1]. One possible formula is given below:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

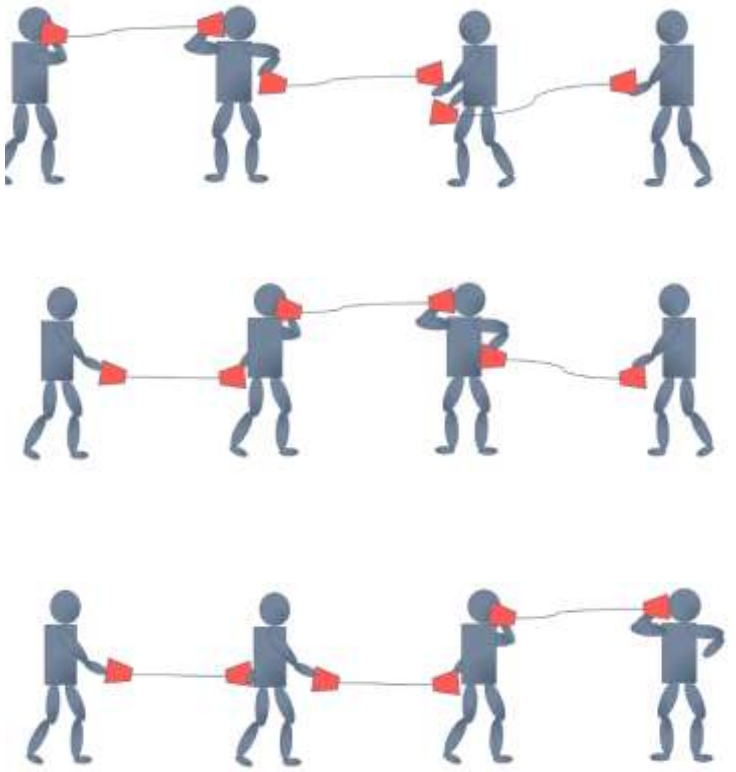
- To have zero mean and unit variance (whitening):

$$x_{new} = \frac{x - \mu}{\sigma}$$

Internal covariate shift:

The cup game example

- 1-2: “go water the plants”
- 2-3: “go water teapans”
- ….
- “goa turkey ounce”
- Would help to have a consistent way of passing messages in a more controlled and standardized (“normalized”) way. e.g: Same volume, same language, etc.



“First layer parameters change and so the distribution of the input to your second layer changes”

Proposed Solution: Batch Normalization (BN)

Batch Normalization (BN): A normalization method/layer for neural networks.

Usually inputs to neural networks are normalized to either the range of [0, 1] or [-1, 1] or to mean=0 and variance=1

BN essentially performs whitening to intermediate layers of the networks.

Naive method: Normalize (whiten) per-neuron activation values (either at beginning of forward pass or while computing gradients).

Why the naïve whitening approach doesn't work?

Naive method: Normalize (whiten) per-neuron activation values (either at beginning of forward pass or while computing gradients).

Issues:

Expressive Capacity: Forcing a strict distribution on the activations could limit what the network can represent.

Why the naïve whitening approach doesn't work?

Naive method: Normalize (whiten) per-neuron activation values.

Issues:

- **Expressive Capacity**: Forcing a strict distribution on the activations could limit what the network can represent.
- **Representational Power**: After several layers of strict normalization, activations across layers might become too similar, losing representational power.

The proposed solution: To add an extra regularization layer

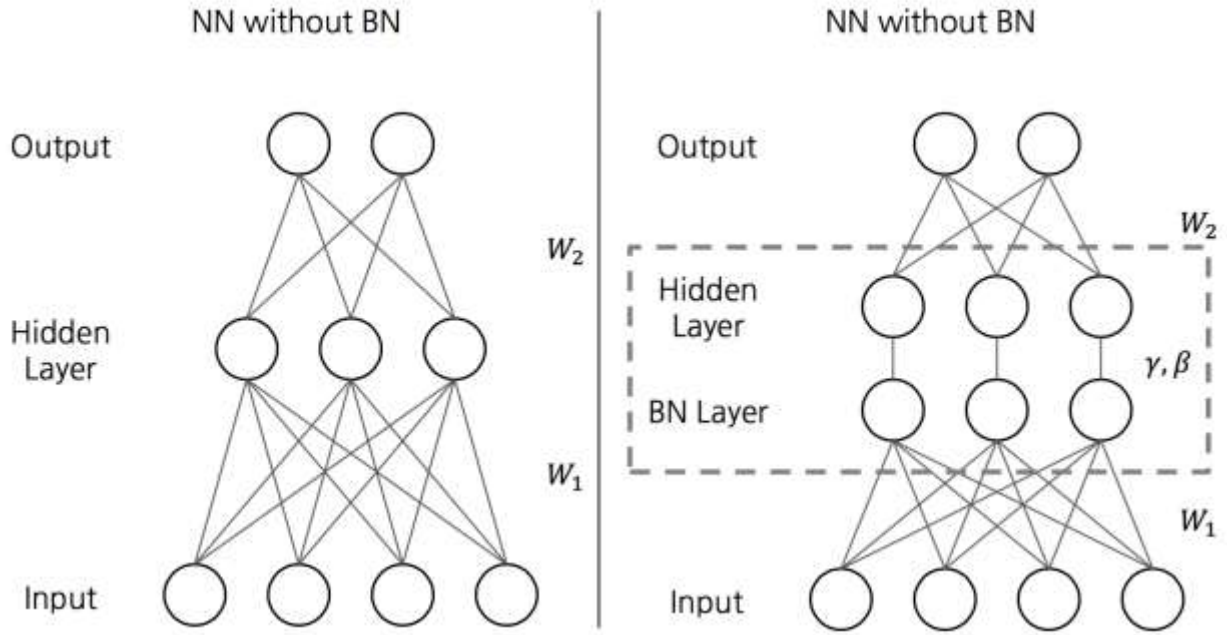
we introduce, for each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}, \beta^{(k)}$, which scale and shift the normalized value:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$

The proposed solution: To add an extra regularization layer

we introduce, for each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}, \beta^{(k)}$, which scale and shift the normalized value:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$



A new layer is added so the gradient can “see” the normalization and make adjustments if needed.

The Batch Transformation: formally from the paper.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The full algorithm as proposed in the paper

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

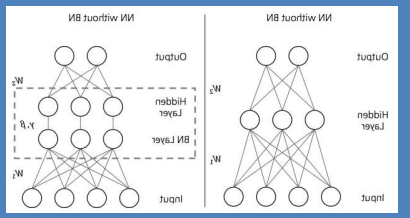
$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Alg 1 (previous slide)

Architecture modification



Note that BN(x) is different during test…

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Vs.

$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

Populations stats vs. sample stats

- In algorithm 1, we are estimating the true mean and variance over the entire population for a given batch.
- When doing inference you're minibatching your way through the entire dataset, you're calculating statistics on a per sample/batch basis. We want our sample statistics to be *unbiased* to population statistics.

	Population Statistics over *N*	Sample/Batch Statistics over *m*
Mean Estimate	$\mu = \frac{1}{N}\sum_i x_i$	$\bar{x} = \frac{1}{m}\sum_i x_i$
Variance Estimate	$\sigma = \frac{1}{N}\sum_i (x_i - \mu)^2$	$\sigma_B = \frac{1}{m-1}\sum_i (x_i - \bar{x})^2$

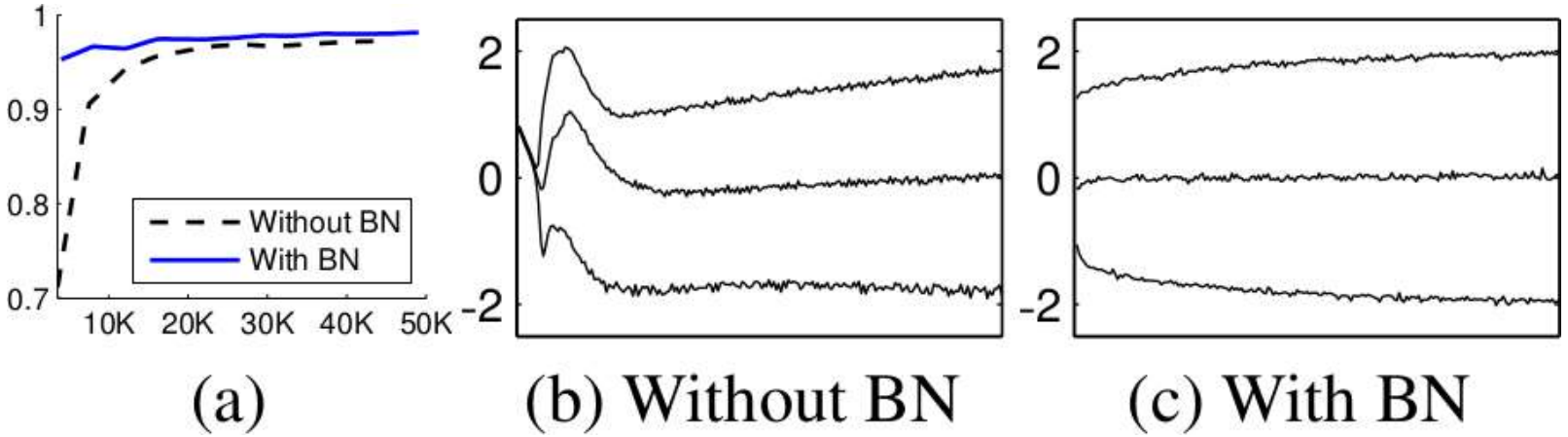
Batch normalization
Why it is good?

- **BN reduces *Covariate Shift*:**
  - Each neuron's activation becomes (more or less) a Gaussian distribution, i.e. its usually not active, sometimes a bit active, rarely very active.
  - Later layers adapting just to new distribution parameters, e.g. new mean and variance values for Gaussian distributions.
- **BN reduces effects of exploding and vanishing gradients**,
  - because every becomes roughly normally distributed.

Batch normalization: Other benefits in practice

- BN reduces training times. (Because of less Covariate Shift, less exploding/vanishing gradients.)
- BN reduces demand for regularization, e.g. dropout or L2 norm.
    - Because the means and variances are calculated over batches and therefore every normalized value depends on the current batch. I.e. the network can no longer just memorize values and their correct answers.)
- BN allows higher learning rates. (Because of less danger of exploding/vanishing gradients.)
- BN enables training with saturating nonlinearities in deep networks, e.g. sigmoid. (Because the normalization prevents them from getting stuck in saturating ranges, e.g. very high/low values for sigmoid.)

Batch normalization: Better accuracy , faster.



BN applied to MNIST (a), and activations of a randomly selected neuron over time (b, c), where the middle line is the median activation, the top line is the 15th percentile and the bottom line is the 85th percentile.

ACCELERATING BN NETWORKS

Batch normalization only not enough!

Increase learning rate.

Remove Dropout.

Shuffle training examples more thoroughly

Reduce the L2 weight regularization.

Accelerate the learning rate decay.

Reduce the photometric distortions.

Useful links

Blog posts

https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b#.s4ftttada

https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

A lecture mentioning the technique:

https://www.youtube.com/watch?v=gYpoJMlgyXA&feature=youtu.be&list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC&t=3078

Paper summaries:

https://github.com/aleju/papers/blob/master/neural-nets/Batch_Normalization.md

https://wiki.tum.de/display/lfdv/Batch+Normalization

https://aresearch.wordpress.com/2015/11/05/batch-normalization-accelerating-deep-network-training-b-y-reducing-internal-covariate-shift-ioffe-szegedy-arxiv-2015/

Q&A:

http://stats.stackexchange.com/questions/215458/what-is-an-explanation-of-the-example-of-why-batch-normalization-has-to-be-done