CS60010: Deep Learning
Spring 2021

Sudeshna Sarkar

Regularization Continued

15 Feb 2021

Solutions for Overfitting

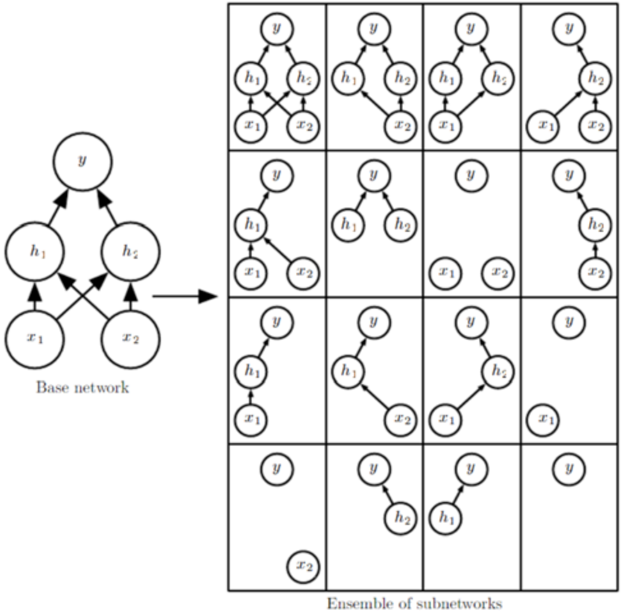
- Regularization
  - L1-norm penalty
  - L2-norm penalty
- Data Augmentation
- Dropout (2012)
  - A method of bagging applied to neural networks
  - An inexpensive but powerful method of regularizing a broad family of models
- Batch Normalization (2015)

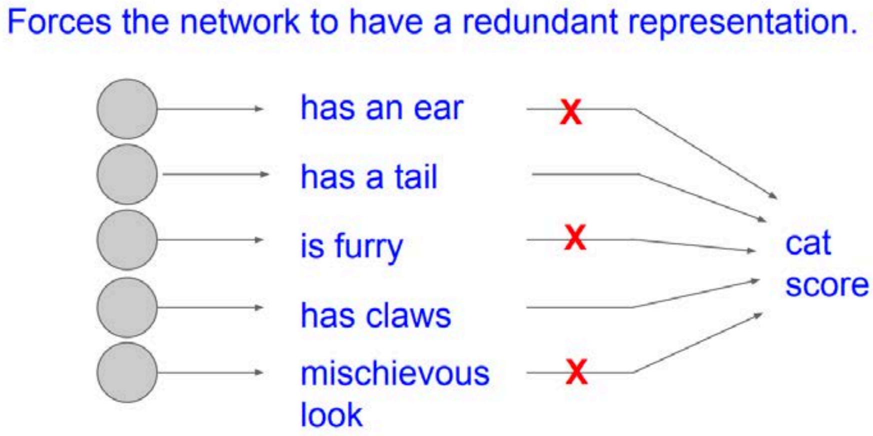
Research in Dropout

- First proposed by G.E. Hinton (2012)
- Became popular by AlexNet (2012)
  - Winner in ILSVRC-2012 (ImageNet challenge)
  - AlexNet outperforms the other models at most 2x
  - CNN model with ReLU, Dropout, Data augmentation, GPU
  - Applied the dropout at Full-Connected layer
- Reinforced by S. Nitish (2014)
  - Strengthen the theoretical background, extend to convolutional layer

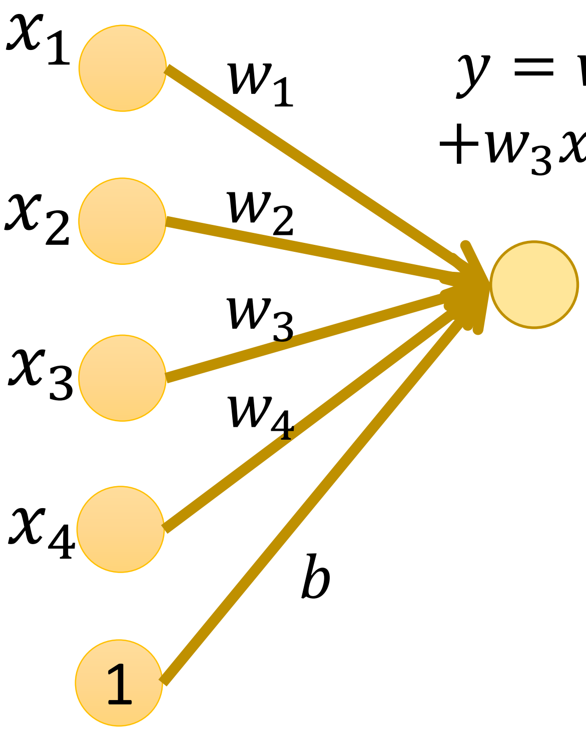
Dropout

- Training with dropout consists of training sub-networks that can be formed by removing non-output units from an underlying base network





Dropout

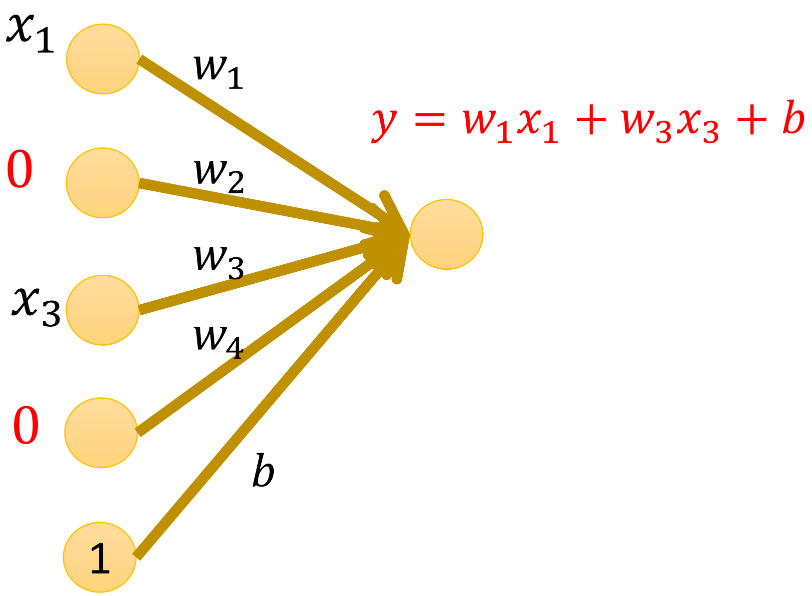


$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

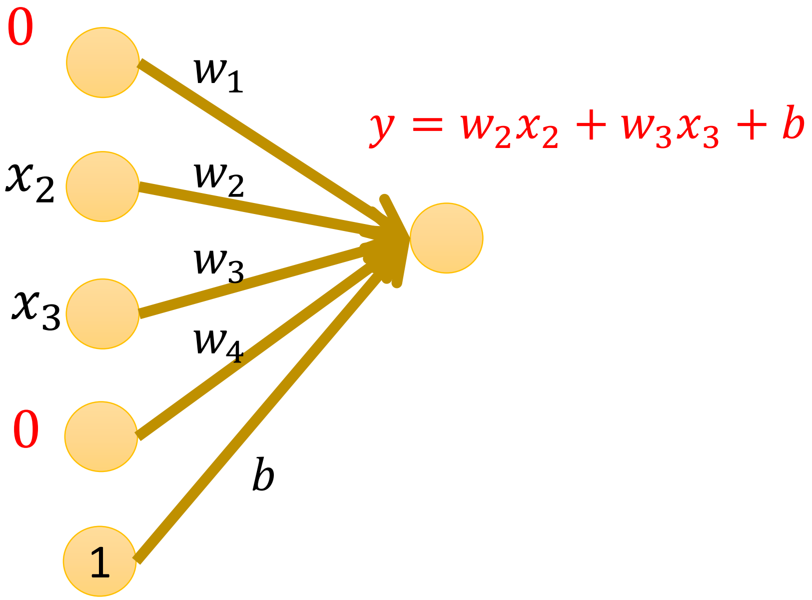
During training

- For each data point
- Randomly set input to 0 with probability 0.5 (or p) – dropout ratio

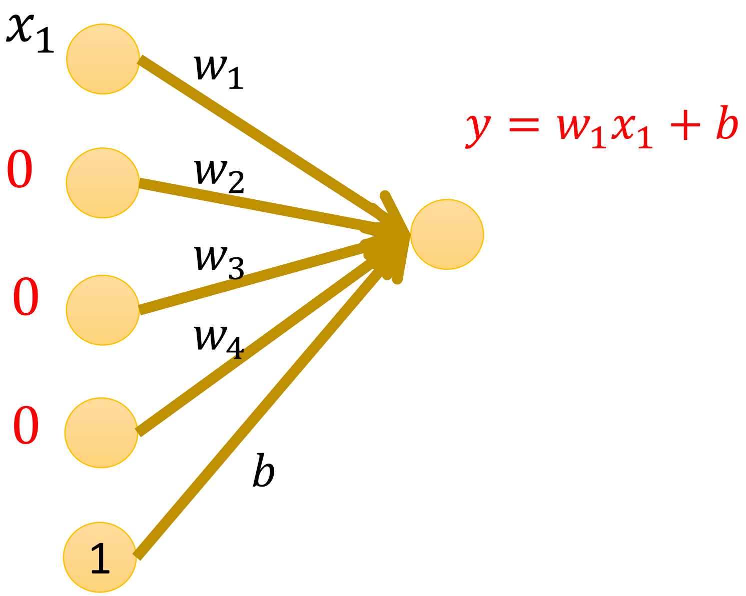
Training



What is Dropout

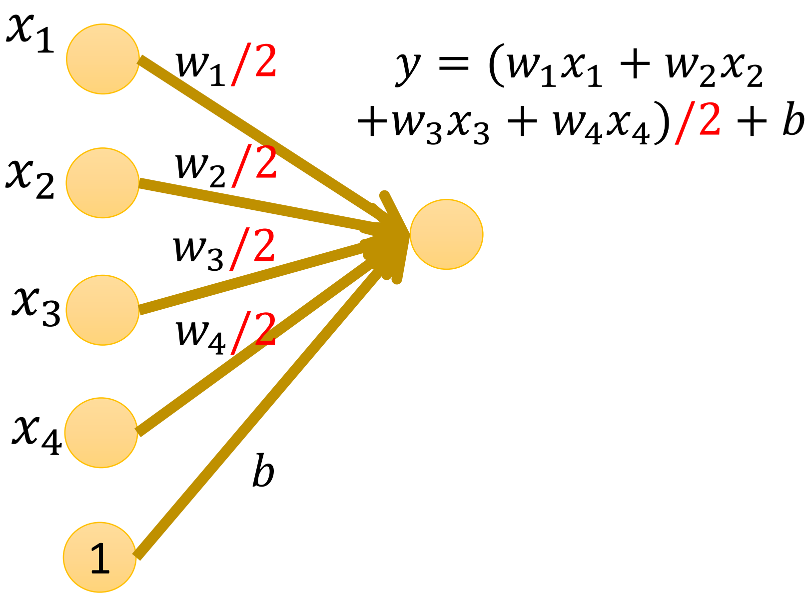


What is Dropout



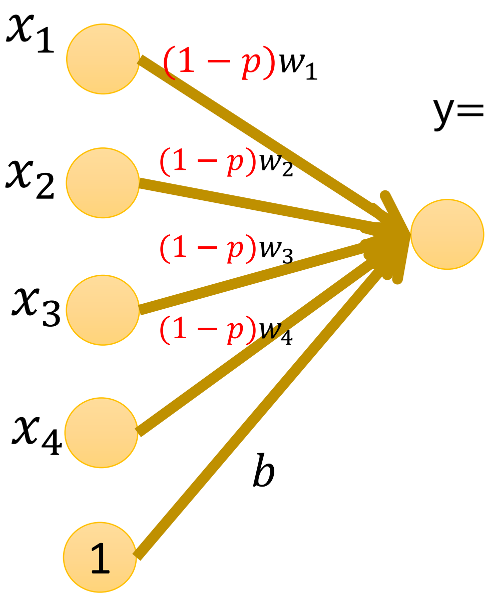
What is Dropout

- During training
  - For each data point
  - Randomly set input to 0 with probability 0.5 “dropout ratio”.
- During testing
  - Halve weights
  - No dropout



What is Dropout

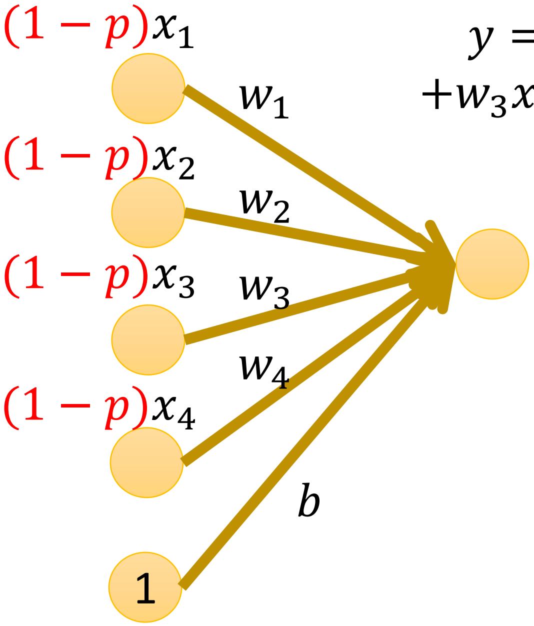
- During training
  - For each data point
  - Randomly set input to 0 with probability p "dropout ratio".
- During testing
  - Multiply weights by $1-p$
  - No dropout



y= $(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4)(1-p) + b$

What is Dropout

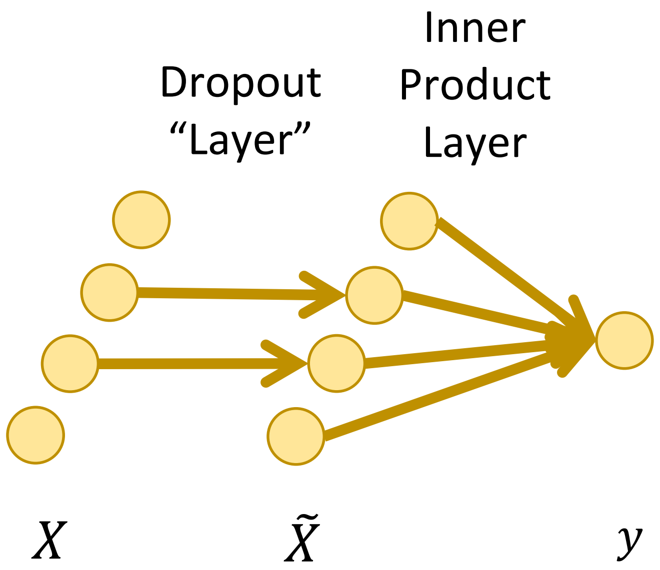
- During training
  - For each data point
  - Randomly set input to 0 with probability p "dropout ratio".
- During testing
  - Multiply data by $1 - p$
  - No dropout



$$y = (w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4)(1-p) + b$$

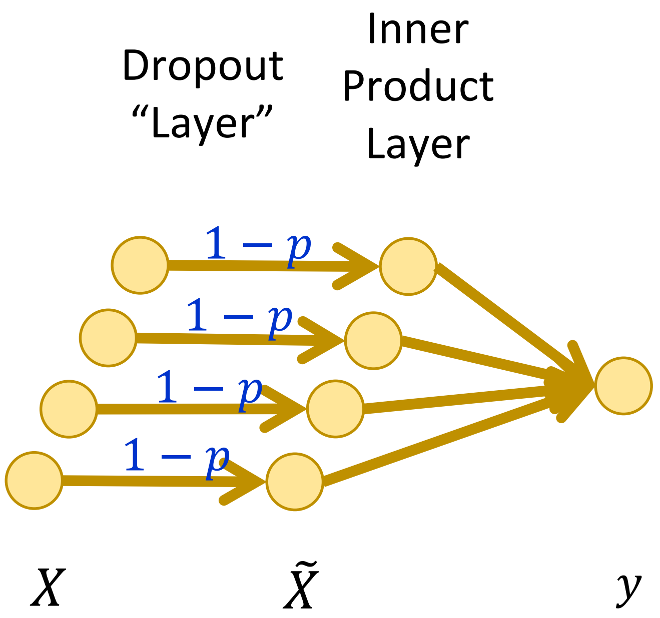
What is Dropout

- During training
  - For each data point
  - Randomly set input to 0 with probability p "dropout ratio".
- During testing
  - Multiply data by $1 - p$
  - No dropout



What is Dropout

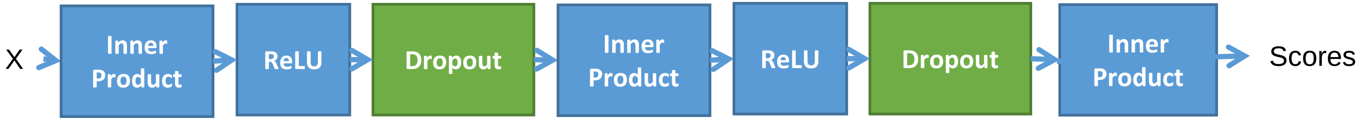
- During training
  - For each data point
  - Randomly set input to 0 with probability p "dropout ratio".
- During testing
  - Multiply data by $1 - p$
  - No dropout



Dropout on MLP

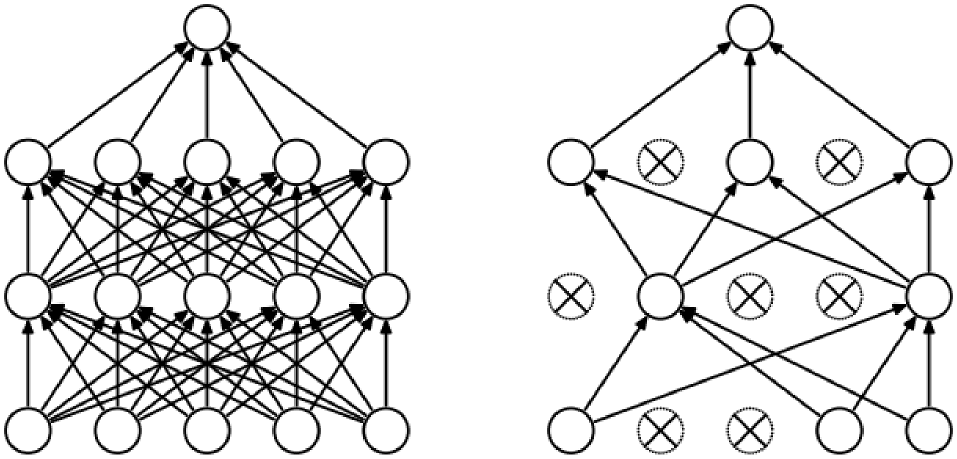
Without dropout

With dropout



Dropout

- Bagging with shared weights
  - Exponentially many architectures



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

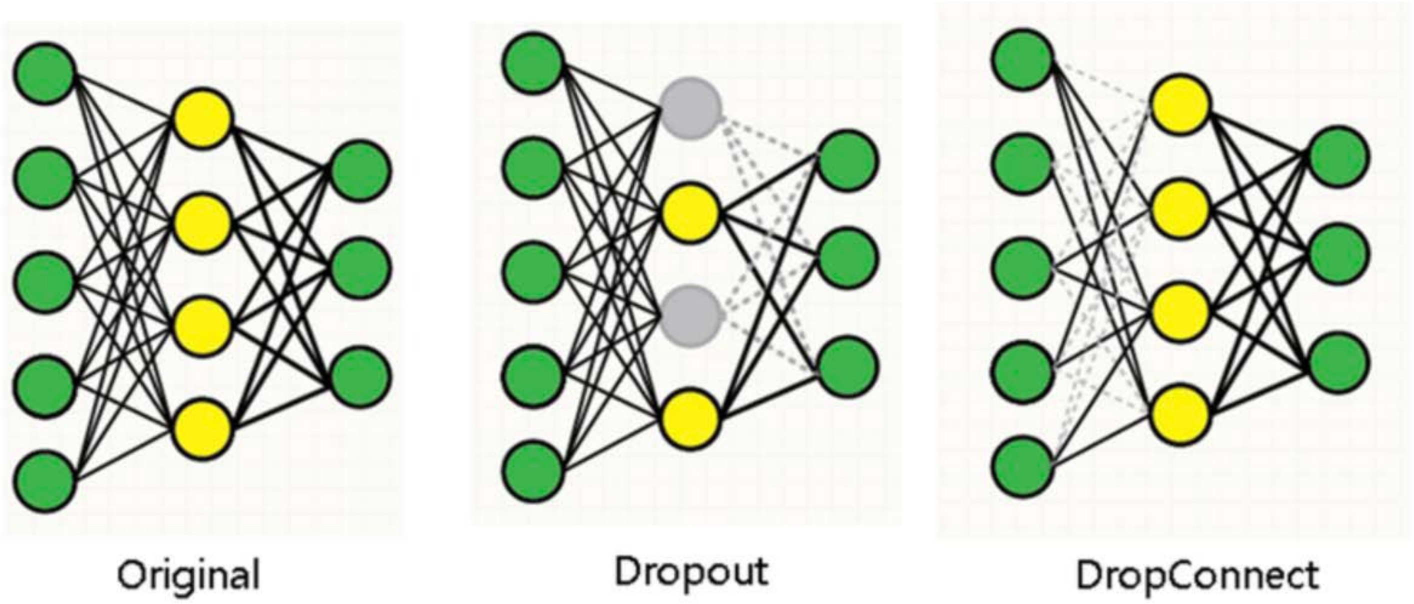
Bagging: reduces generalization error through combining several models

1. Train k different models on k different subsets of training data
2. Have all of the models vote on the output for test examples

DropConnect

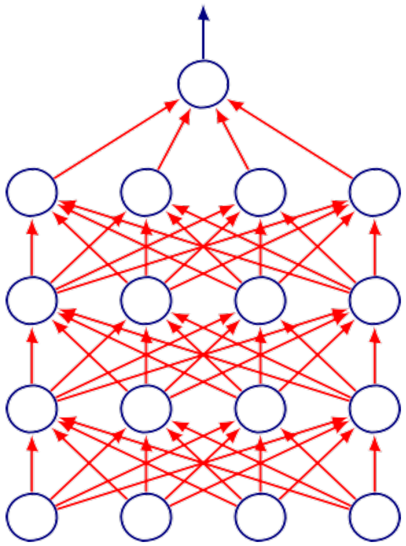
A generalization of Dropout

- **Dropout omit the all connections which related to the unit**
- **DropConnect only omit the connections and leave the unit alive**



Batch Normalization

- A Difficulty in Training Deep Neural Networks
    - A deep model involves composition of several functions



- Distribution of inputs to a layer is changing during training
- Harder to train: smaller learning rate, careful initialization
- Easier if distribution of inputs stayed same
- How to enforce same distribution?

Batch Normalization

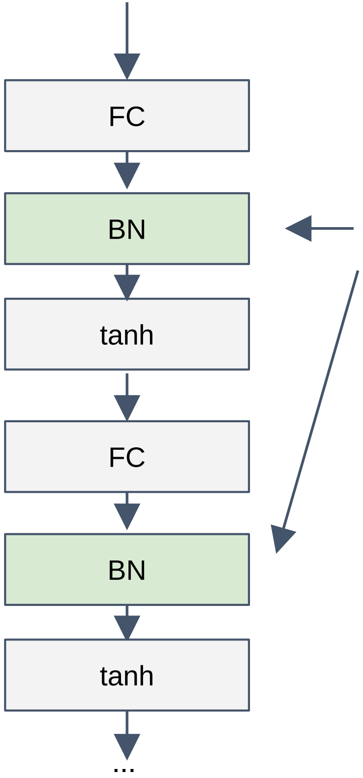
- Batch normalization (batch norm) is a method used to make neural networks faster and more stable through normalization of the input layer by re-centering and re-scaling.
- Consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Ioffe, Sergey; Szegedy, Christian (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".

Batch Normalization

[Ioffe and Szegedy, 2015]



Where to insert, i.e. **before** or after non-linearities?

Differing opinions on this: different options have worked in different models.
Usually inserted after Fully Connected / (or Convolutional) layers, and before nonlinearity.

$$\boxed{\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}}$$

Batch Normalization

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

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

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Batch Normalization

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

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

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

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

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

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

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe