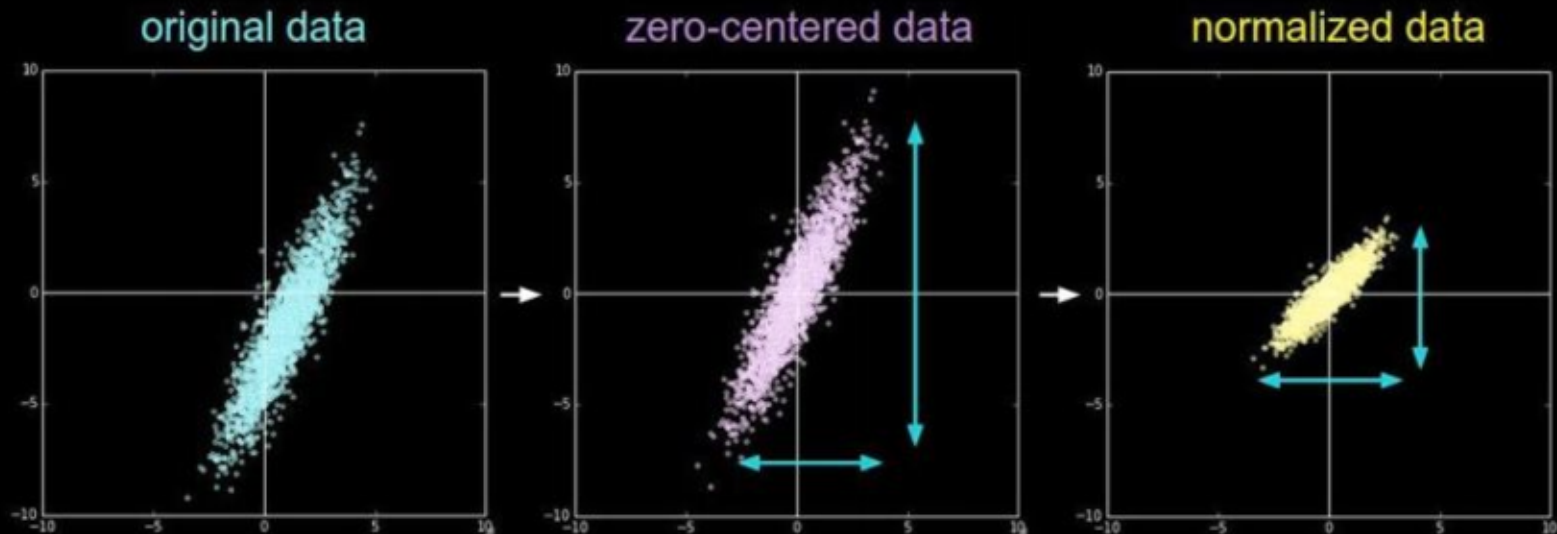
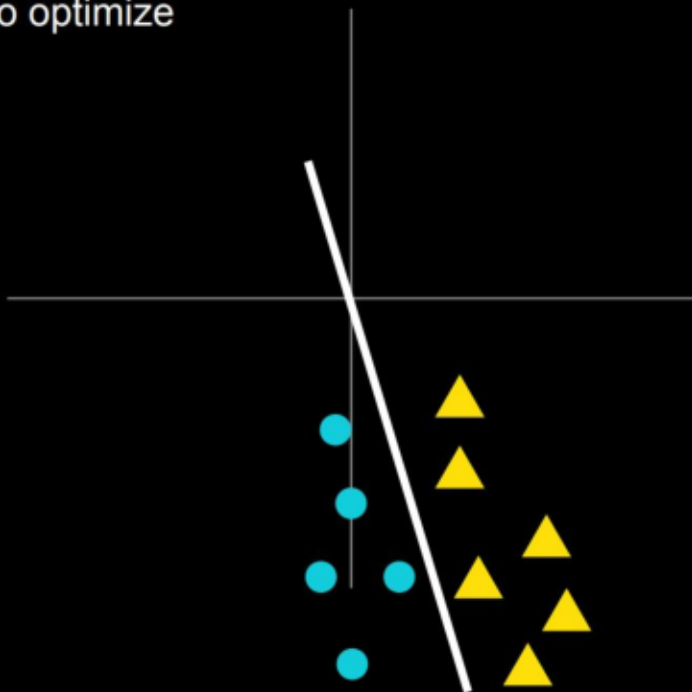


# Data normalization

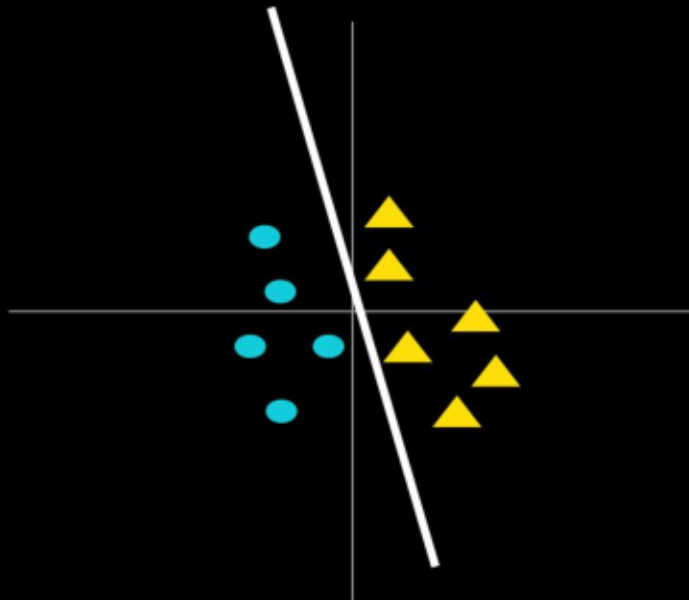


# Data normalization

**Before normalization:** classification loss very sensitive to changes in weight matrix; hard to optimize



**After normalization:** less sensitive to small changes in weights; easier to optimize



- Pitfall: all zero initialization.

# Weights initialization

- Pitfall: all zero initialization.
- Small random numbers.

# Weights initialization

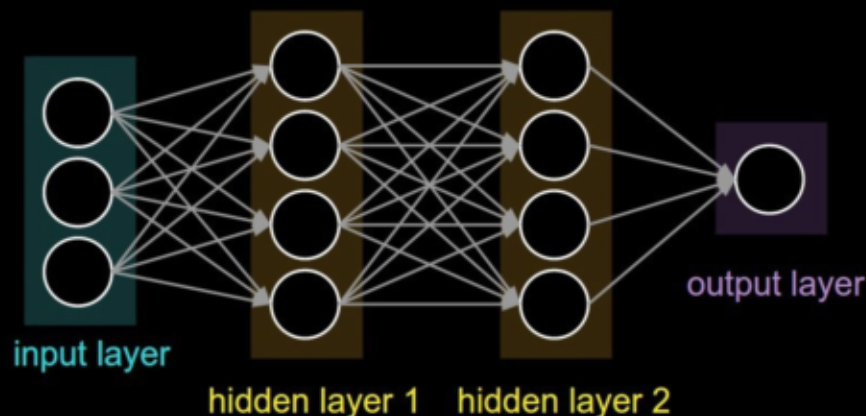
- Pitfall: all zero initialization.
- Small random numbers.
- Calibrated random numbers.

$$\begin{aligned} s &= \sum_i^n w_i x_i \\ \text{Var}(s) &= \text{Var}\left(\sum_i^n w_i x_i\right) \\ &= \sum_i^n \text{Var}(w_i x_i) \\ &= \sum_i^n [E(w_i)]^2 \text{Var}(x_i) + E[(x_i)]^2 \text{Var}(w_i) + \text{Var}(x_i) \text{Var}(w_i) \\ &= \sum_i^n \text{Var}(x_i) \text{Var}(w_i) \\ &= (n \text{Var}(w)) \text{Var}(x) \end{aligned}$$

# Batch normalization

Problem:

- Consider a neuron in any layer beyond first
- At each iteration we tune it's weights towards better loss function
- But we also tune it's inputs. Some of them become larger, some – smaller
- Now the neuron needs to be re-tuned for it's new inputs



TL; DR:

- It's usually a good idea to normalize linear model inputs
- (c) Every machine learning lecturer, ever

- Normalize activation of a hidden layer  
(zero mean unit variance)

$$h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$$

- Update  $\mu_i, \sigma_i^2$  with moving average while training

$$\mu_i := \alpha \cdot \text{mean}_{\text{batch}} + (1 - \alpha) \cdot \mu_i$$

$$\sigma_i^2 := \alpha \cdot \text{variance}_{\text{batch}} + (1 - \alpha) \cdot \sigma_i^2$$



# Batch normalization

Original algorithm  
(2015)

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

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

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

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

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

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

# Batch normalization

Original algorithm  
(2015)

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

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

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

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

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

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

What is this?

# Batch normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots x_m\}$ ;

Parameters to be learned:  $\gamma, \beta$

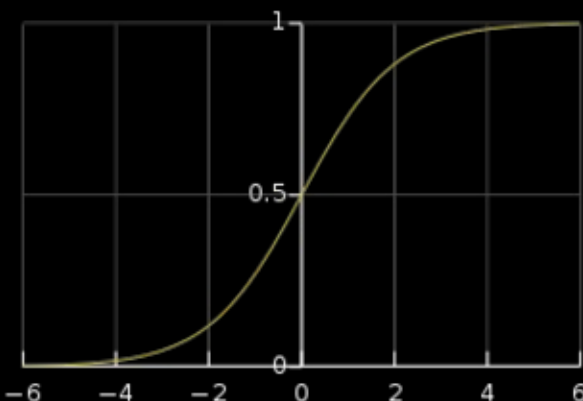
**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

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

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

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



# Batch normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots x_m\}$ ;

Parameters to be learned:  $\gamma, \beta$

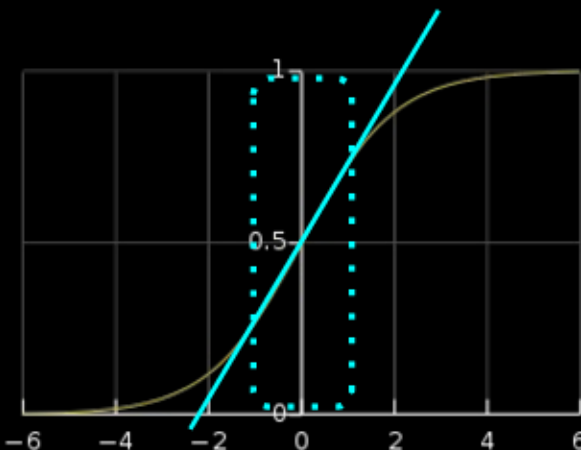
**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

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

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

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



# Batch normalization

Original algorithm  
(2015)

What is this?

This transformation  
should be able to  
represent the identity  
transform.

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

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

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

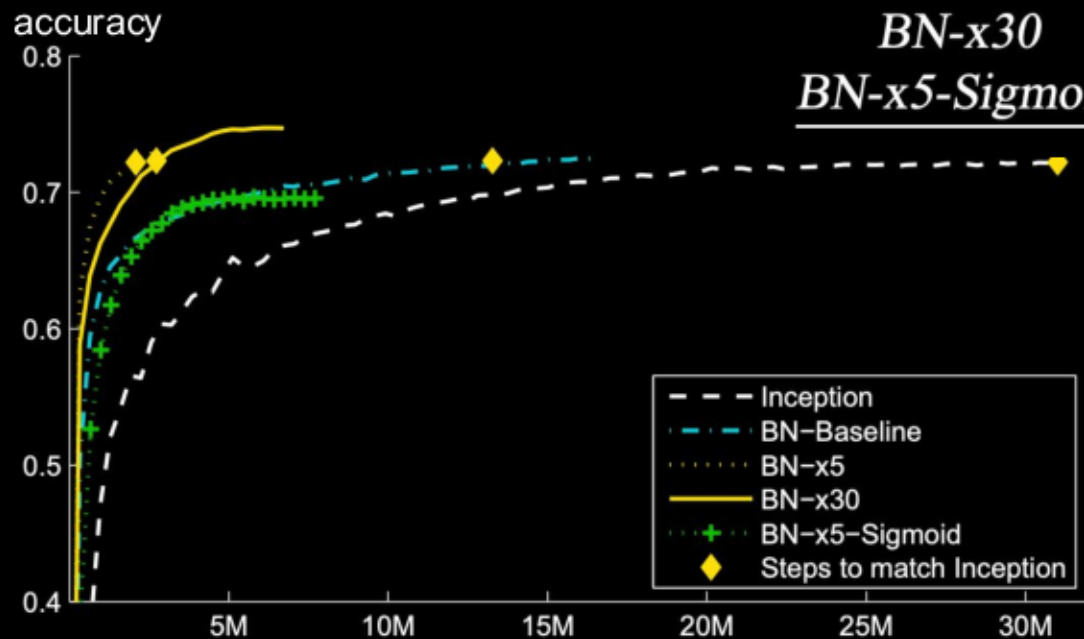
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

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

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

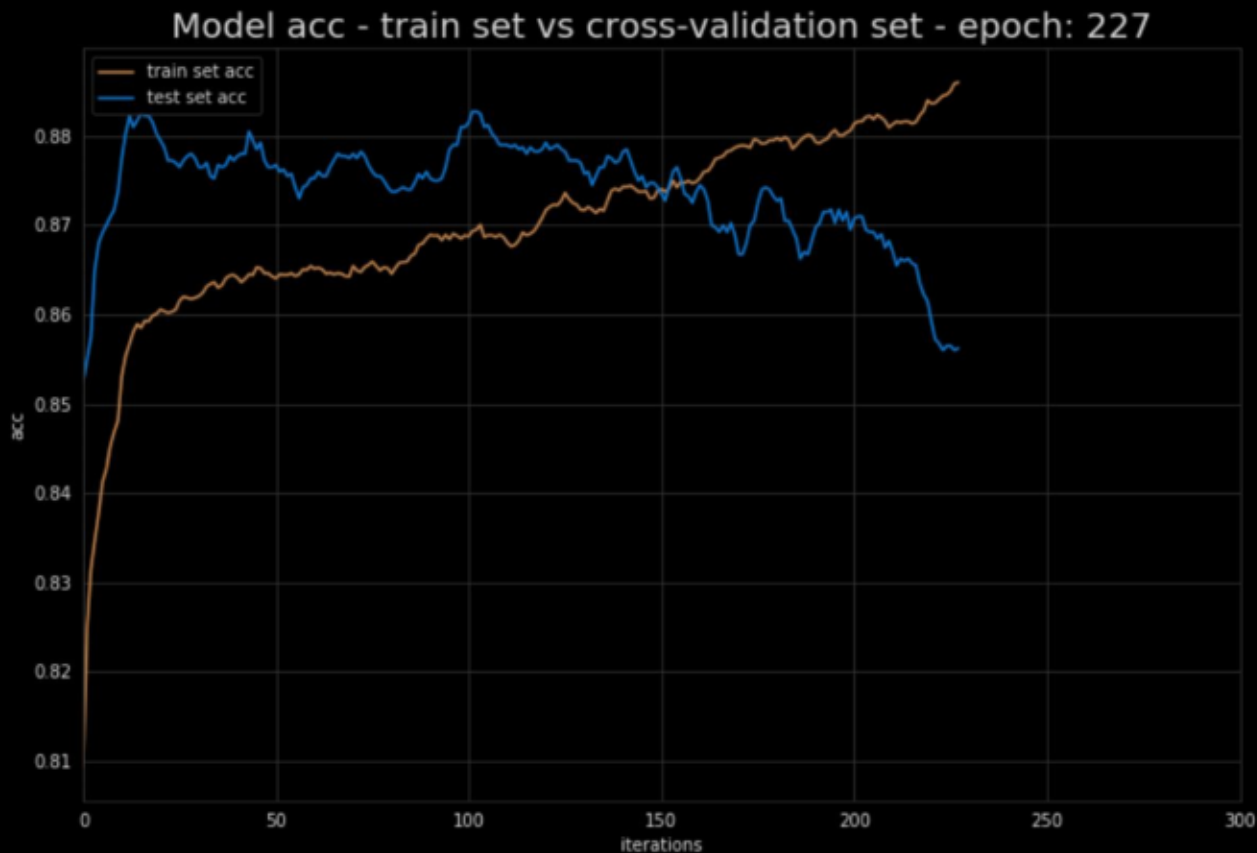
# Batch normalization

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



number of training steps

# Problem: overfitting



$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

Adding some extra term to the loss function.

Common cases:

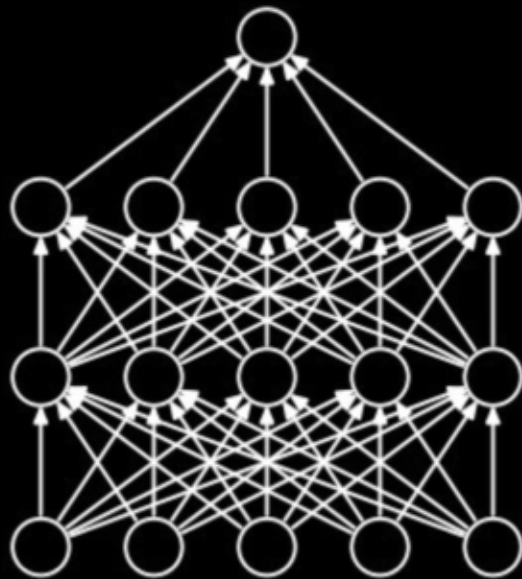
- L2 regularization:  $R(W) = \|W\|_2^2$
- L1 regularization:  $R(W) = \|W\|_1$
- Elastic Net (L1 + L2):  $R(W) = \beta \|W\|_2^2 + \|W\|_1$



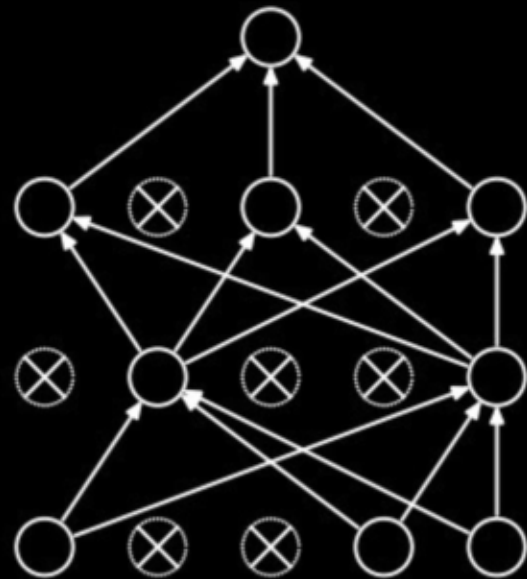
# Regularization: Dropout

Some neurons are “dropped” during training.

Prevents overfitting.



(a) Standard Neural Net

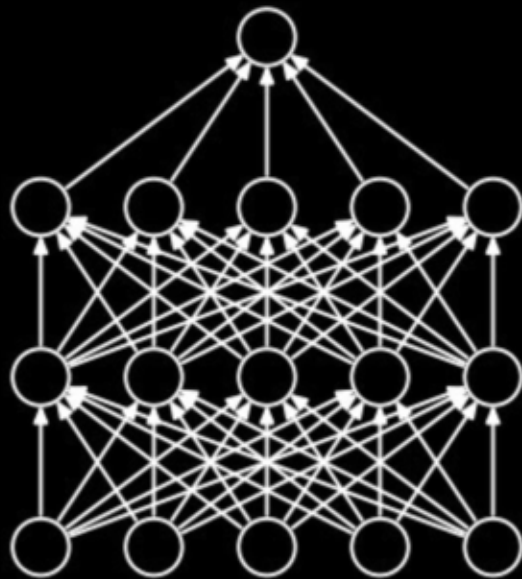


(b) After applying dropout.

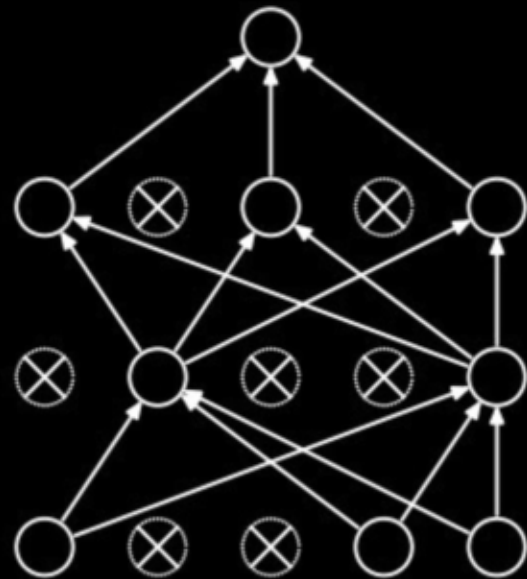
# Regularization: Dropout

Some neurons are “dropped” during training.

Prevents overfitting.



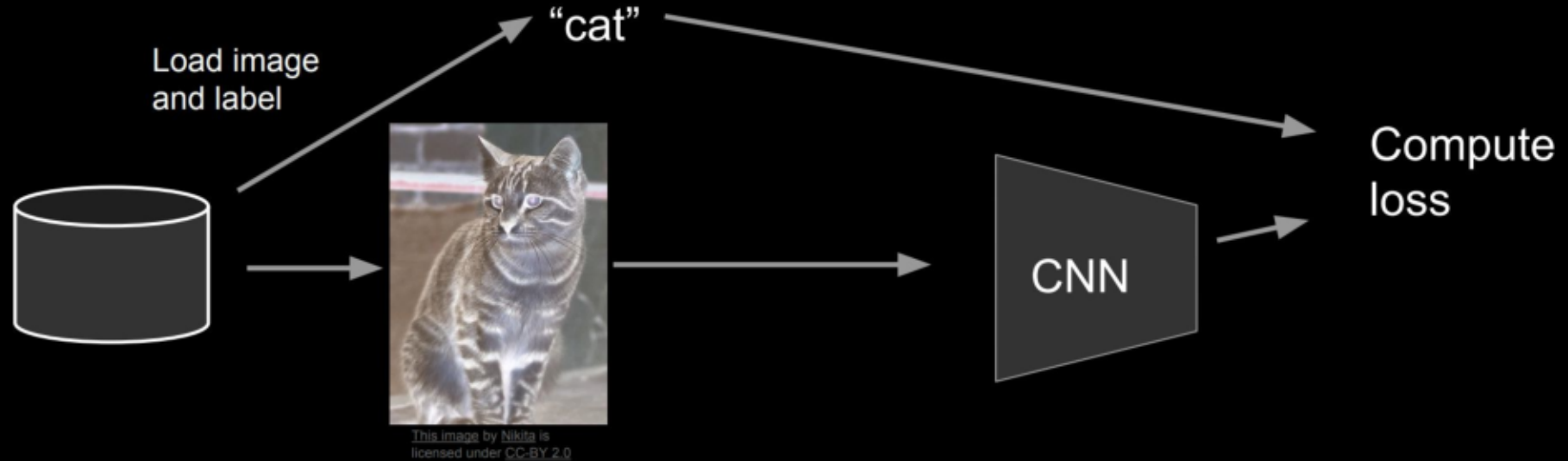
(a) Standard Neural Net



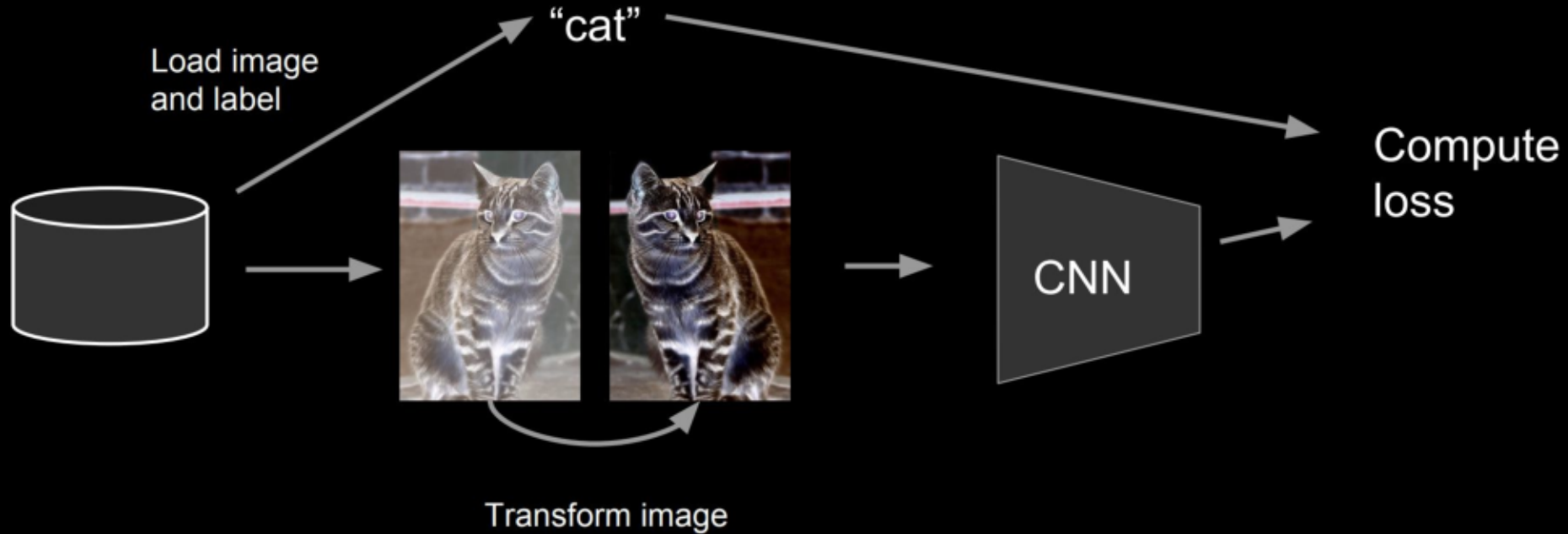
(b) After applying dropout.

Actually, on test case output should be normalized. See sources for more info.

# Regularization: data augmentation



# Regularization: data augmentation



# Sum up: regularization

## Regularization:

- Add some weight constraints
- Add some random noise during train and marginalize it during test
- Add some prior information in appropriate form