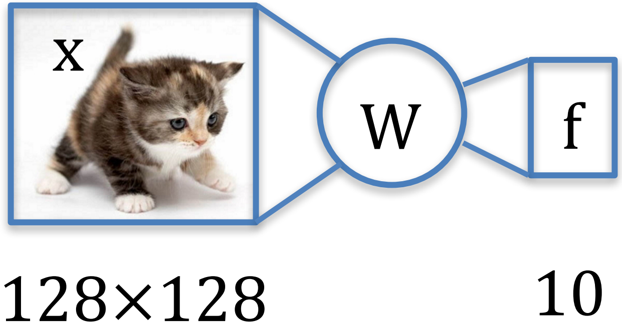
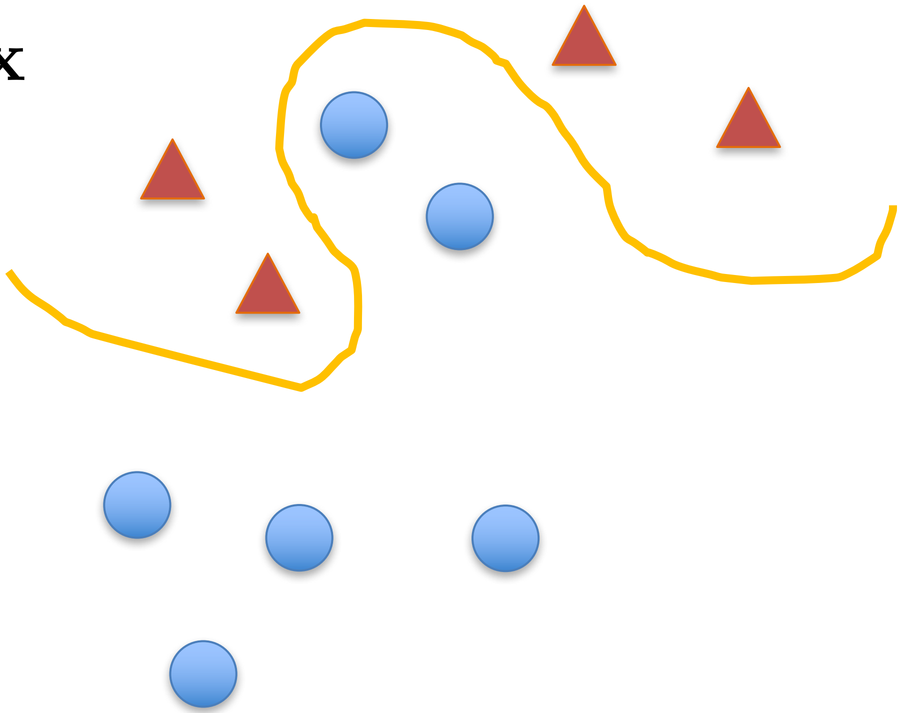
Lecture 5 Recap

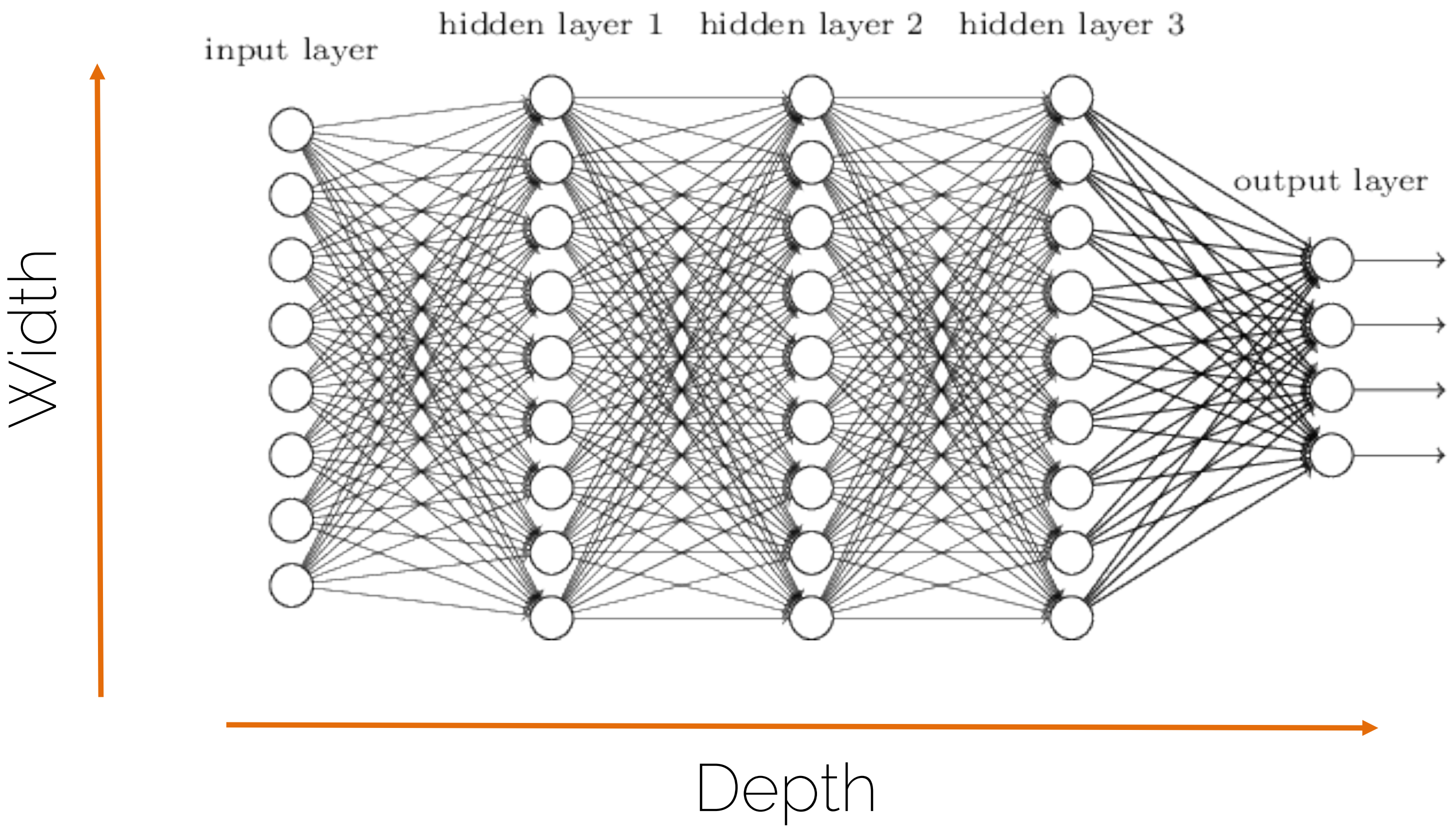
Beyond linear

1-layer network: $f = \mathbf{W}\mathbf{x}$





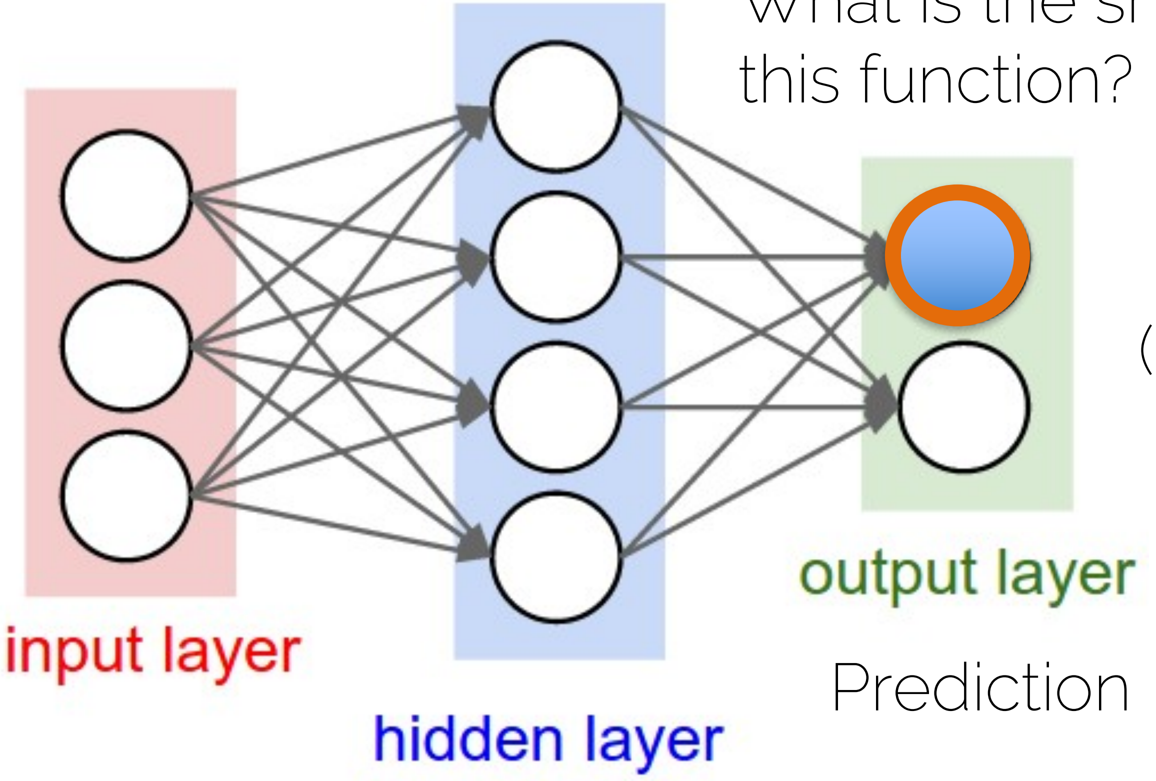
Neural Network



Output functions

Neural networks

What is the shape of this function?

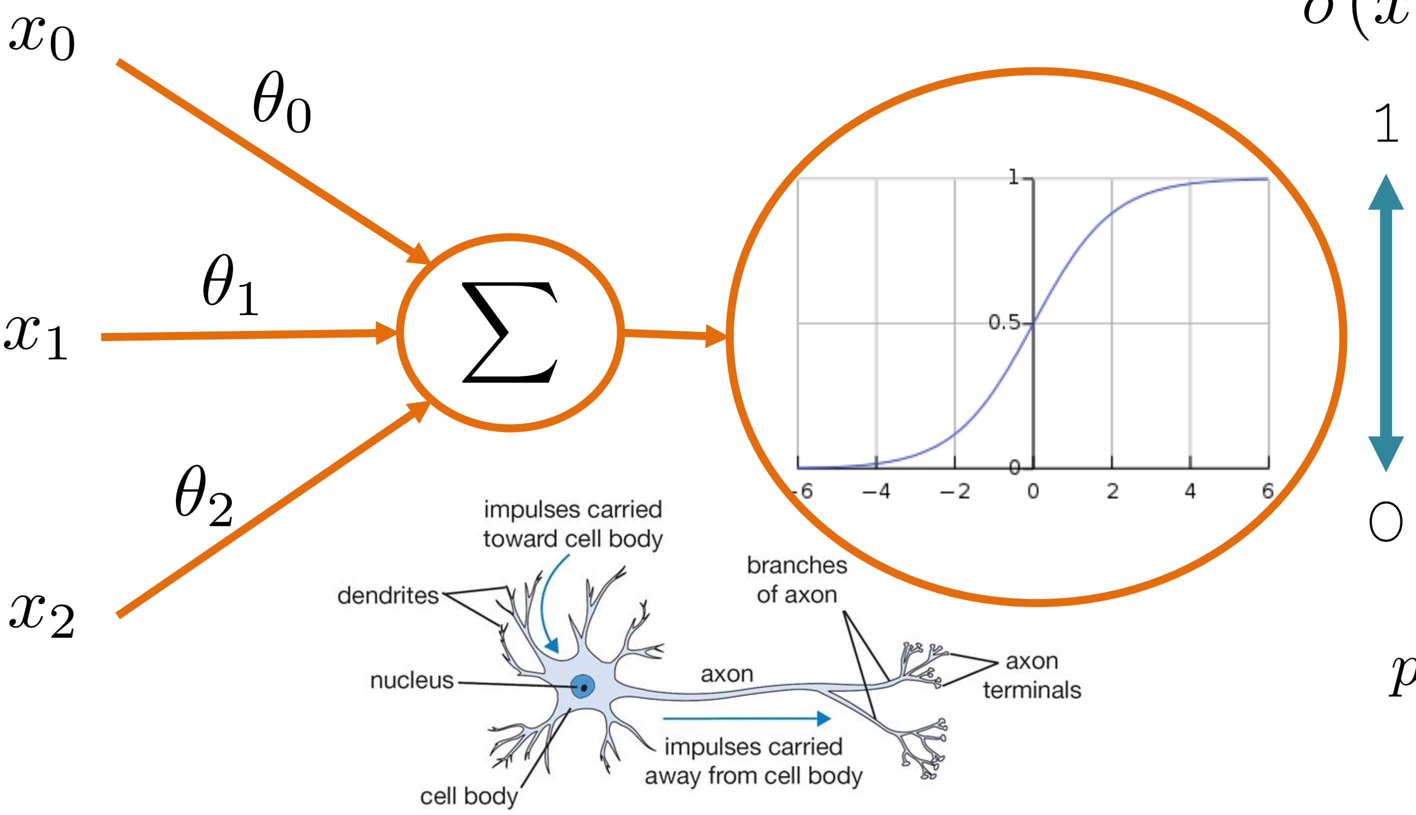


Loss (Softmax, Hinge)

Prediction

Sigmoid for binary predictions

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Can be interpreted as a probability

$$p(y_i = 1|\mathbf{x}_i, \boldsymbol{\theta})$$

Logistic regression

- Optimize using gradient descent

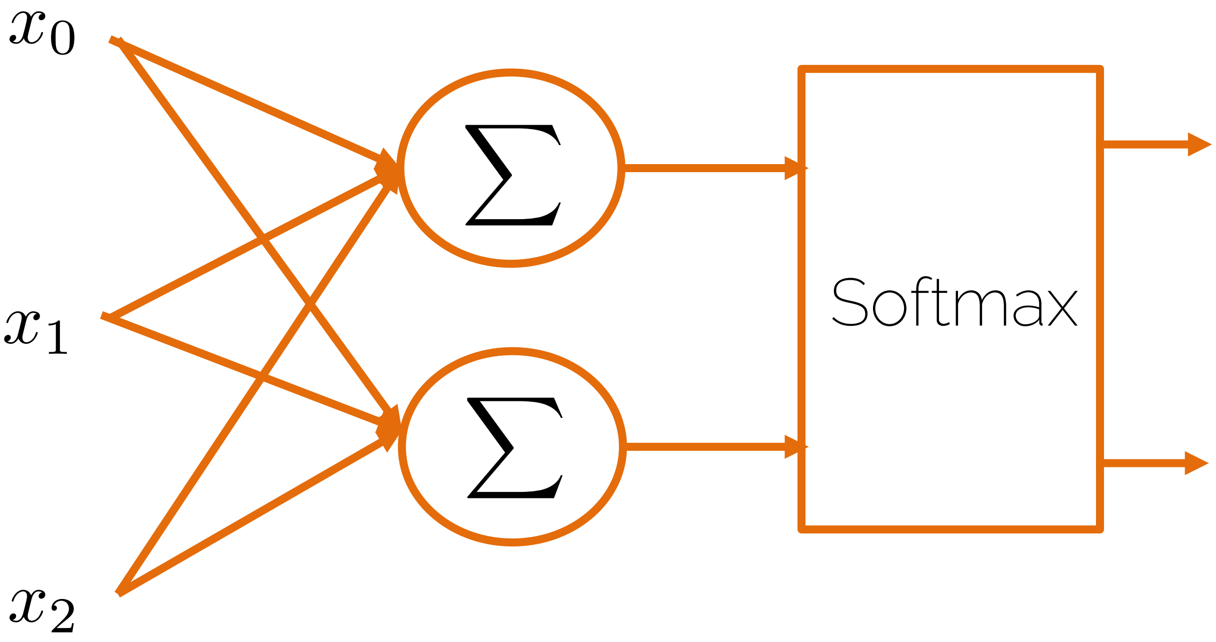
- Saturation occurs only when the model already has the right answer

$$C(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y_i \log(\Pi_i) + (1 - y_i) \log(1 - \Pi_i)$$

Referred to as cross-entropy

Softmax formulation

- What if we have multiple classes?



$$\Pi_1 = \frac{e^{\mathbf{x}_i \boldsymbol{\theta}_1}}{e^{\mathbf{x}_i \boldsymbol{\theta}_1} + e^{\mathbf{x}_i \boldsymbol{\theta}_2}}$$

$$\Pi_2 = \frac{e^{\mathbf{x}_i \boldsymbol{\theta}_2}}{e^{\mathbf{x}_i \boldsymbol{\theta}_1} + e^{\mathbf{x}_i \boldsymbol{\theta}_2}}$$

Softmax formulation

- Softmax

exp

$$p(y_i|\mathbf{x},\boldsymbol{\theta}) = \frac{e^{\mathbf{x}\boldsymbol{\theta}_i}}{\sum_{k=1}^{n} e^{\mathbf{x}\boldsymbol{\theta}_k}}$$

normalize

- Softmax loss (ML)

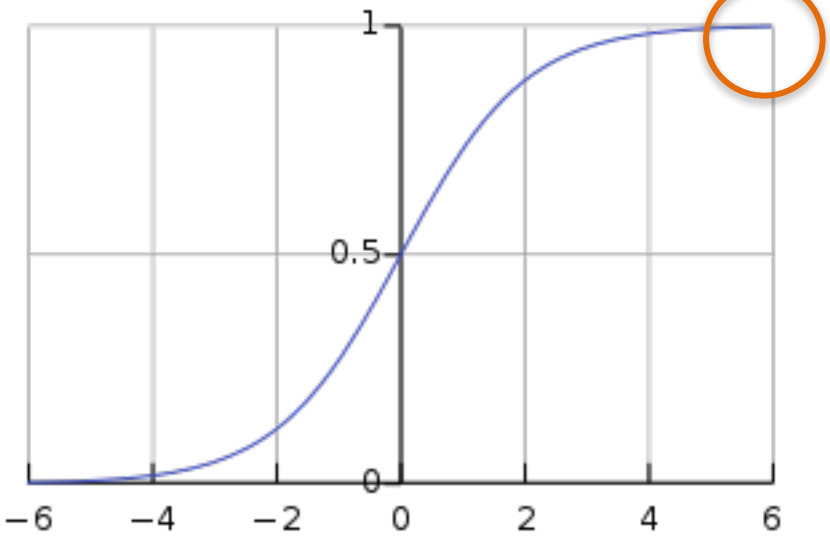
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$$

Activation functions

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Forward

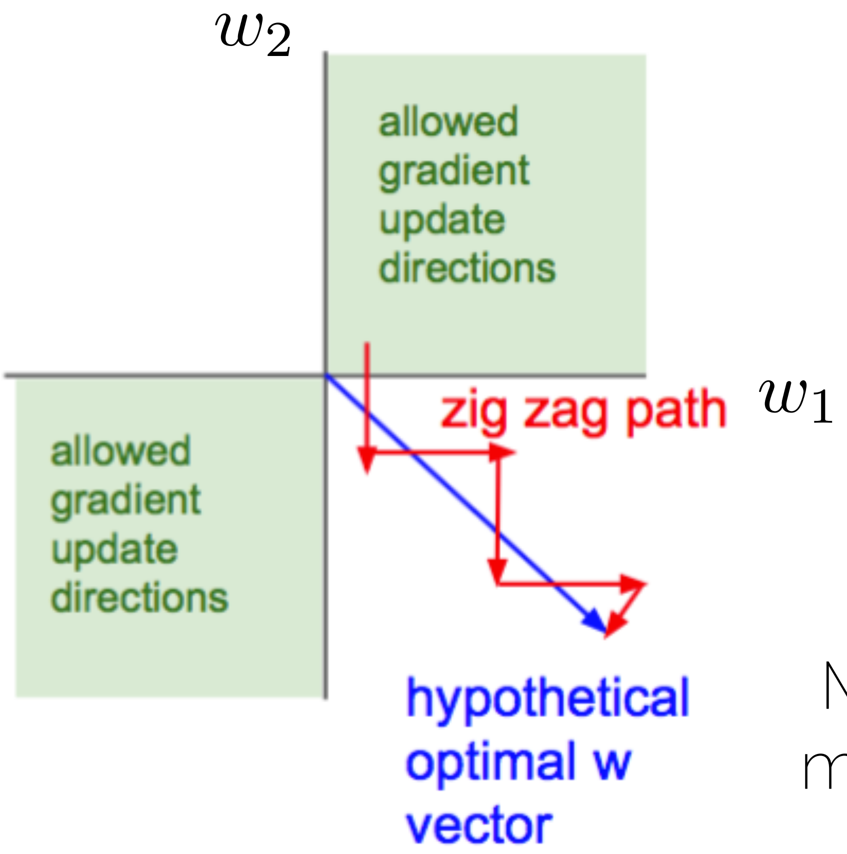


$$x = 6$$

✘ Saturated neurons kill the gradient flow

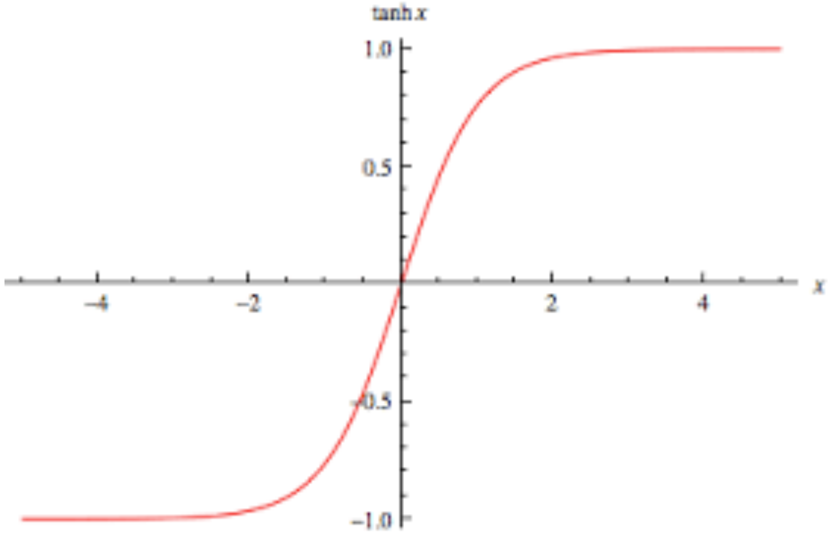
$$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma} \quad \longleftarrow \quad \frac{\partial \sigma}{\partial x} \quad \longleftarrow \quad \frac{\partial L}{\partial \sigma}$$

Problem of positive output



More on zero-mean data later

tanh



✗ Still saturates

✓ Zero-centered

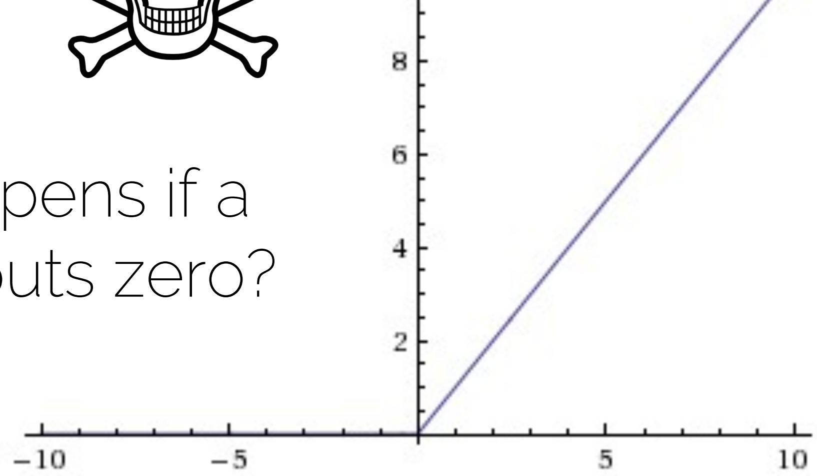
✗ Still saturates

LeCun 1991

Rectified Linear Units (ReLU)

✗ Dead ReLU

What happens if a ReLU outputs zero?



Large and consistent gradients ✓

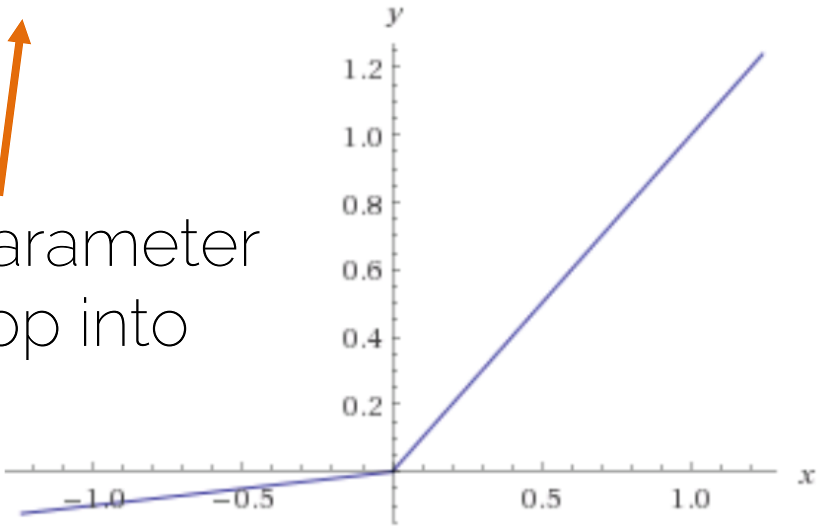
✓ Fast convergence

✓ Does not saturate

Parametric ReLU

$$\sigma(x) = \max(\alpha x, x)$$

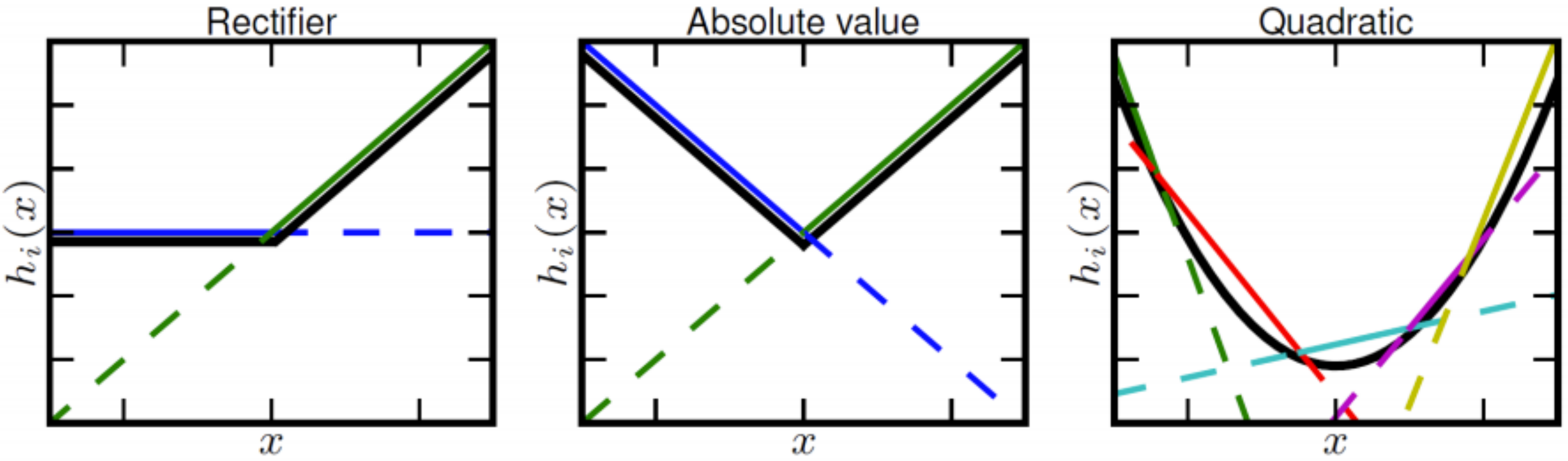
One more parameter
to backprop into



✓ Does not die

He 2015

Maxout units

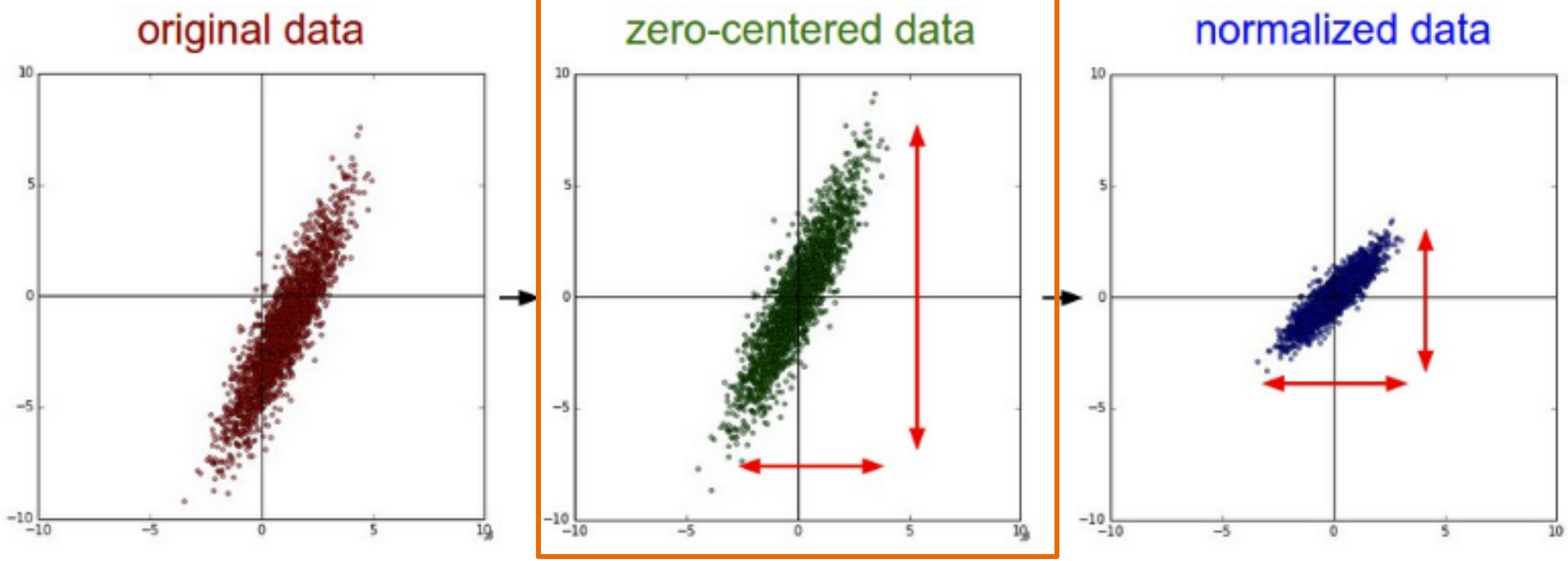


k=2 k=2 k=5

✓ Generalization of ReLUs ✓ Linear regimes ✓ Does not die ✓ Does not saturate

✗ Increase of the number of parameters

Data pre-processing

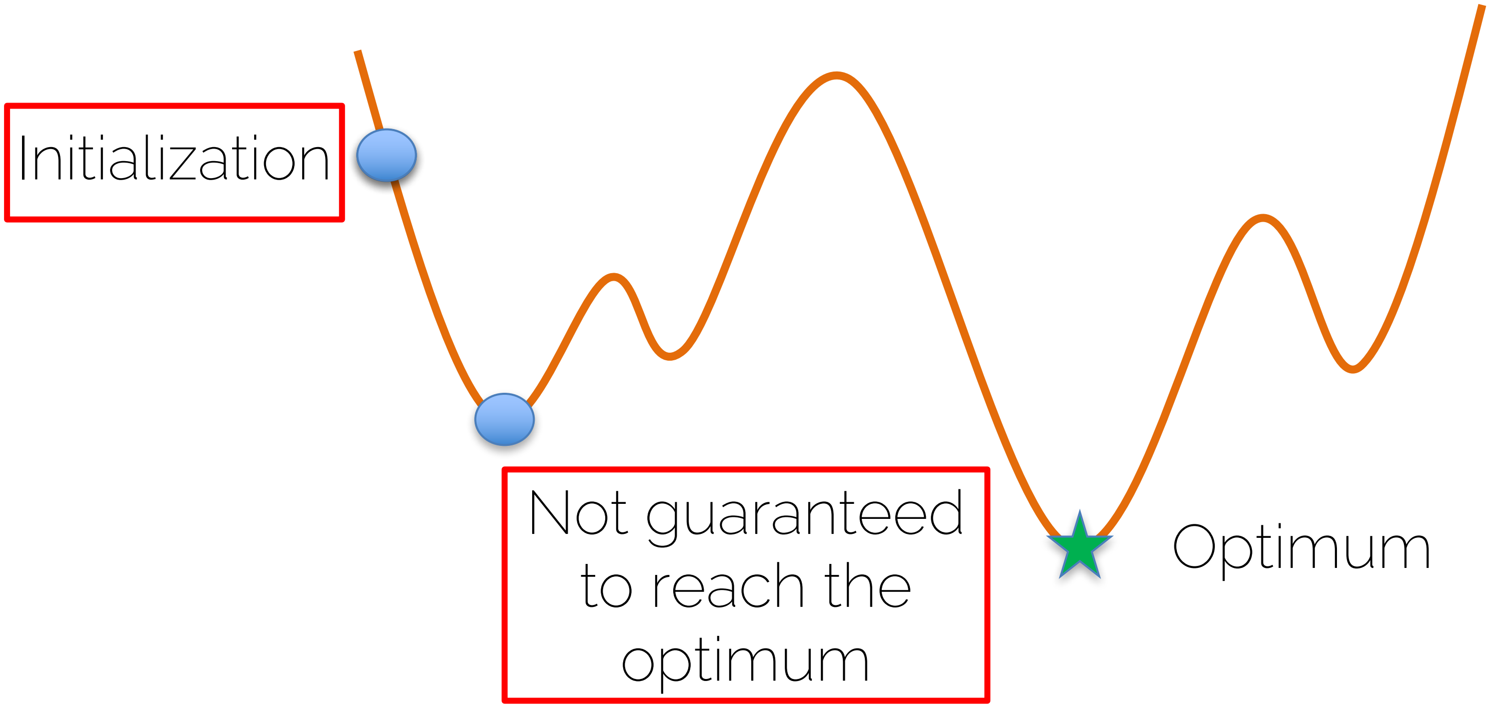


For images subtract the mean image (AlexNet) or per-channel mean (VGG-Net)

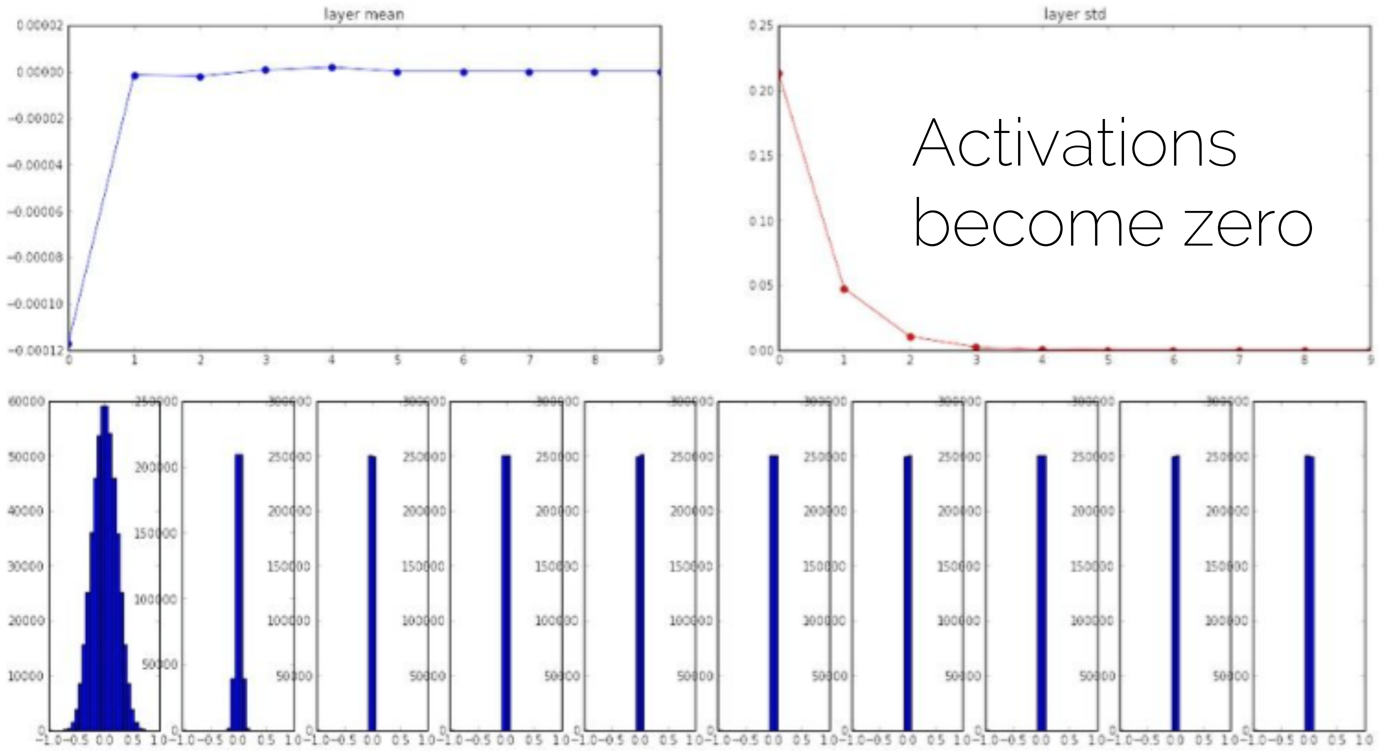
Weight initialization

Initialization is extremely important

$$\mathbf{x}^* = \arg\min f(\mathbf{x})$$



Small random numbers



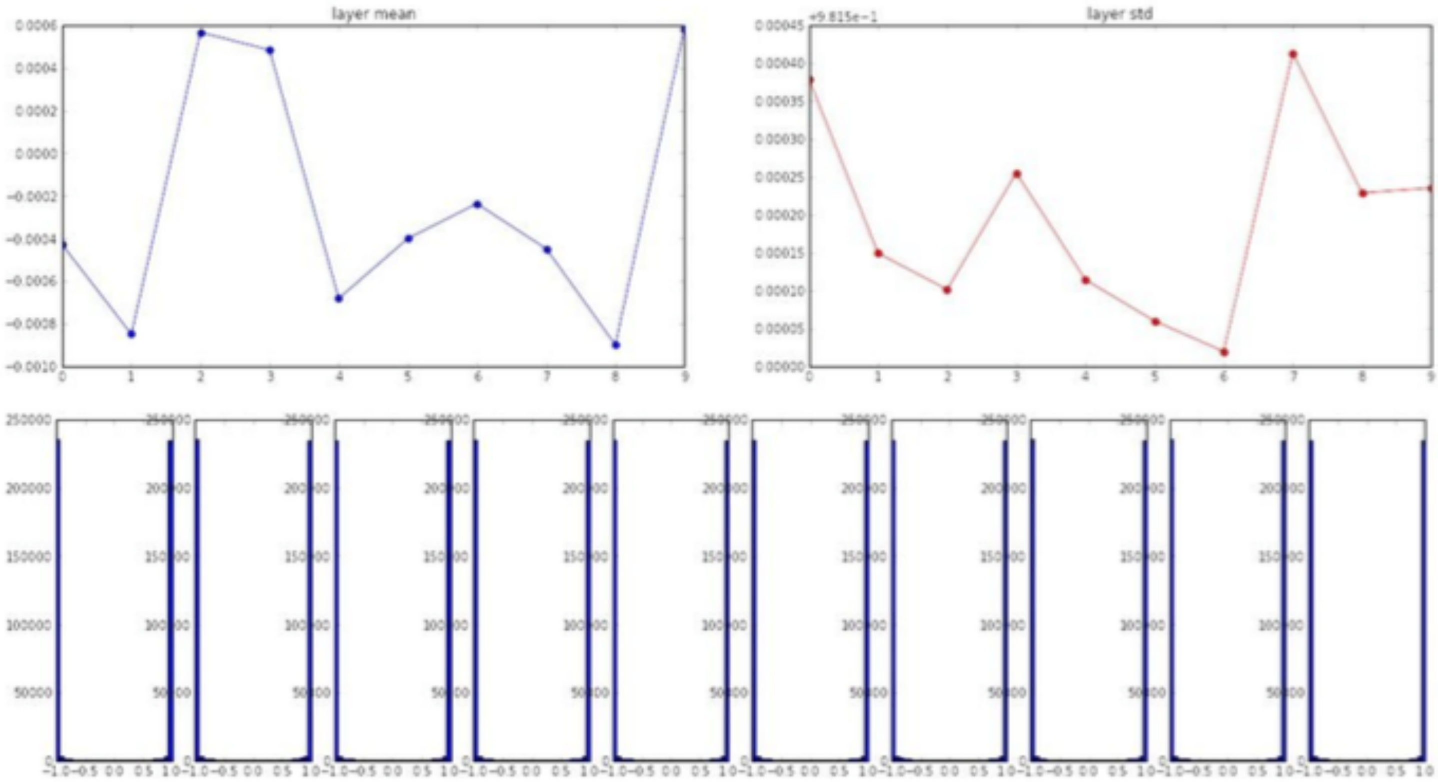
Activations become zero

Input

Last layer

Forward

Big random numbers



Everything is saturated

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_{i}^{n} w_i x_i) = \sum_{i}^{n} \mathrm{Var}(w_i x_i)$$

Glorot 2010

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_i^n w_i x_i) = \sum_i^n \mathrm{Var}(w_i x_i)$$

Independent

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

Zero mean

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\text{Var}(s) = \text{Var}(\sum_i^n w_i x_i) = \sum_i^n \text{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \text{Var}(x_i) + E[(x_i)]^2 \text{Var}(w_i) + \text{Var}(x_i)\text{Var}(w_i)$$

$$= \sum_i^n \text{Var}(x_i)\text{Var}(w_i) = (n\text{Var}(w))\, \text{Var}(x)$$

Identically distributed

Xavier initialization

- Gaussian with zero mean, but what standard deviation?

$$\mathrm{Var}(s) = \mathrm{Var}(\sum_i^n w_i x_i) = \sum_i^n \mathrm{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i) = (n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

Variance gets multiplied by the number of inputs

Xavier initialization

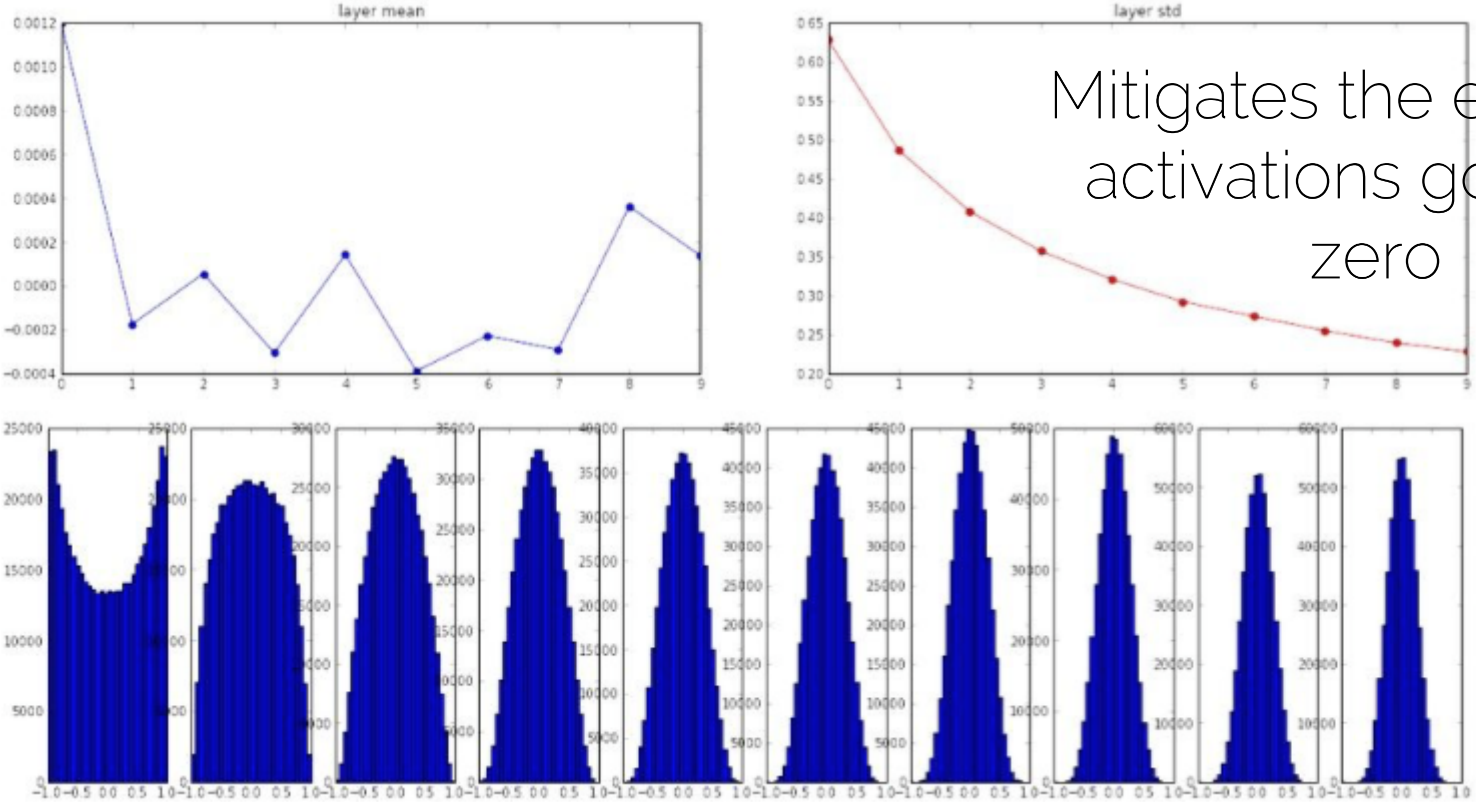
- How to ensure the variance of the output is the same as the input?

$$(n\mathrm{Var}(w))\mathrm{Var}(x)$$

1

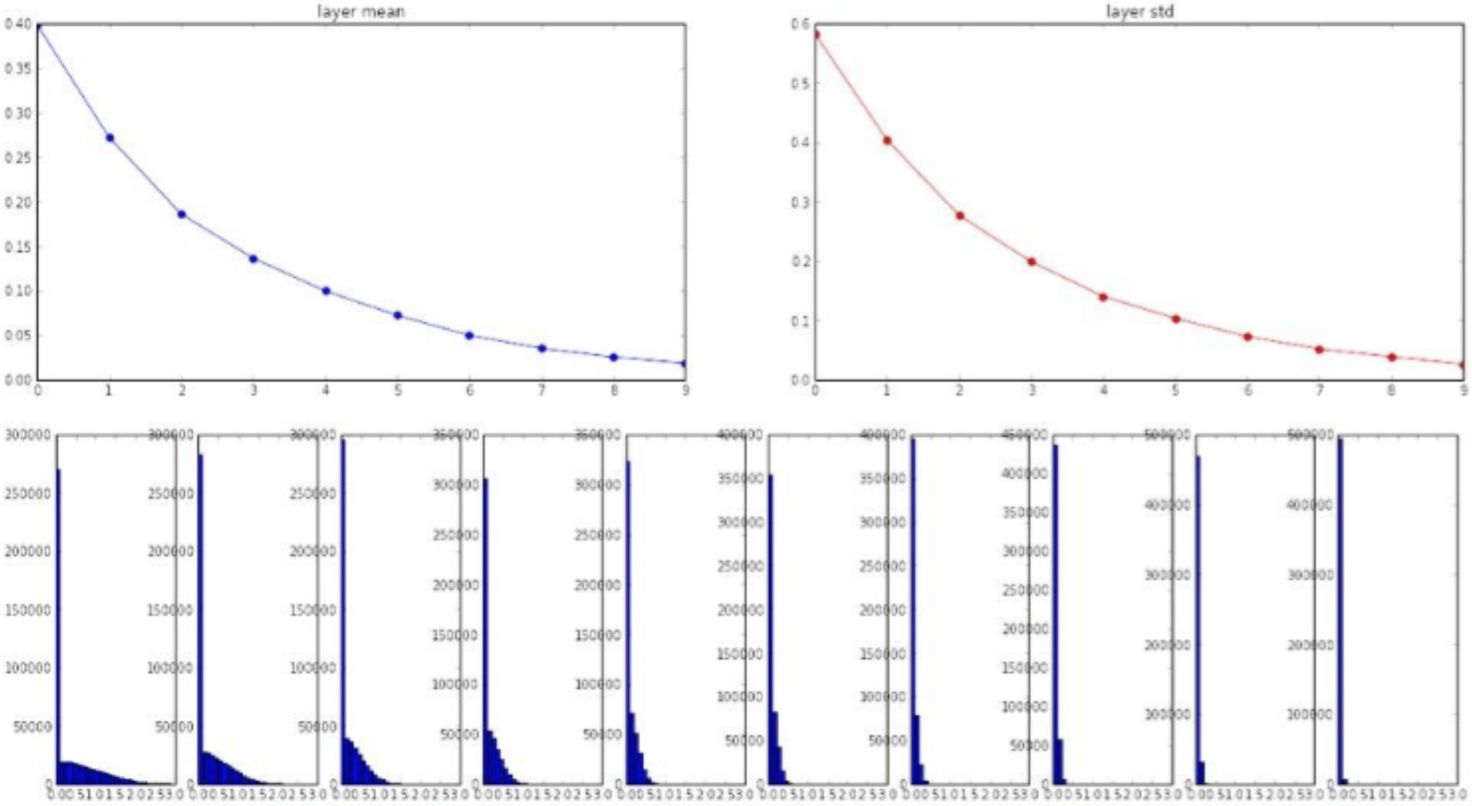
$$Var(w) = \frac{1}{n}$$

Xavier initialization



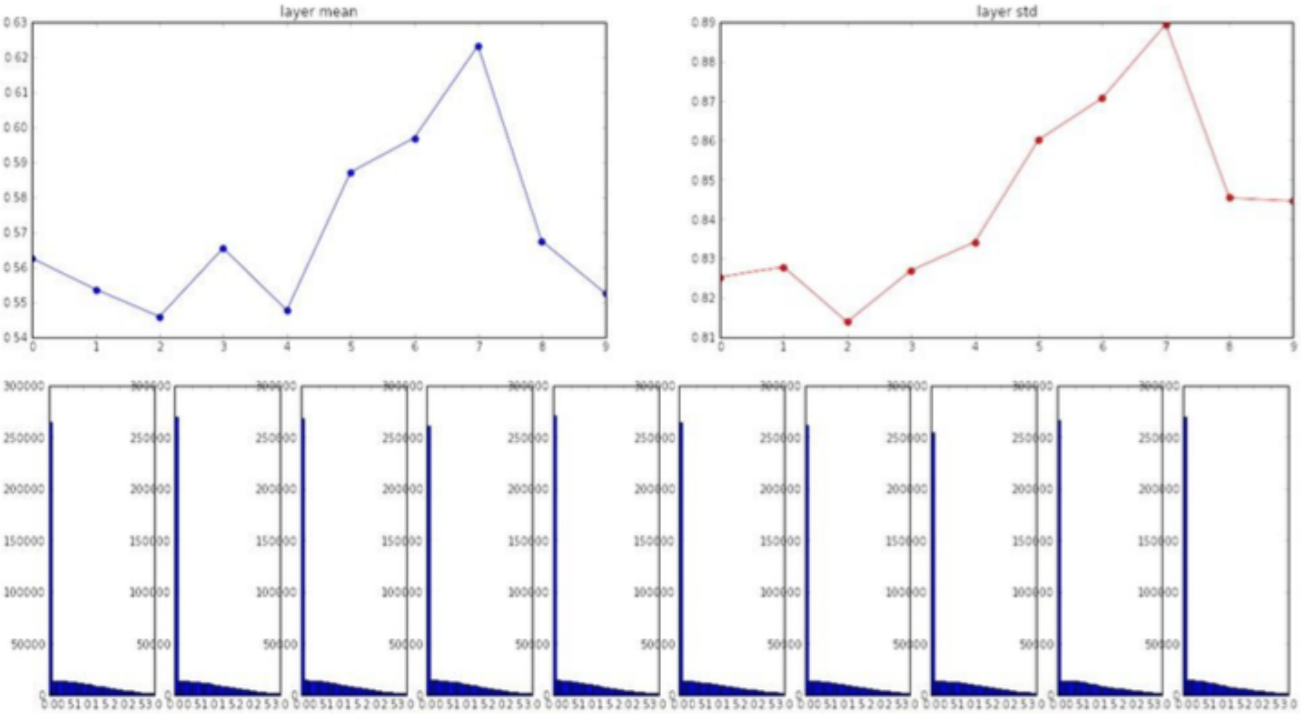
Mitigates the effect of activations going to zero

Xavier initialization with ReLU



ReLU kills half of the data

$$Var(w) = \frac{2}{n}$$

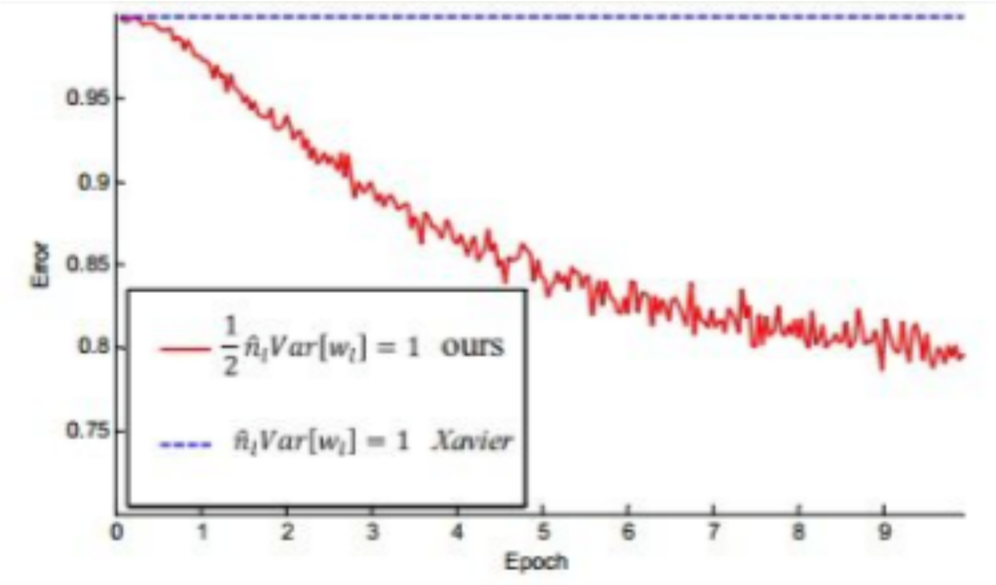


He 2015

ReLU kills half of the data

$$Var(w) = \frac{2}{n}$$

It makes a huge difference!



He 2015

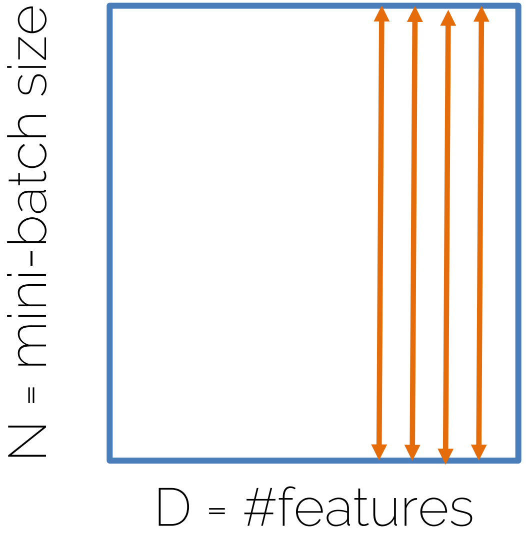
Tips and tricks

- Use ReLU and Xavier/2 initialization

Batch normalization

Batch normalization

- Wish: unit Gaussian activations
- Solution: let's do it



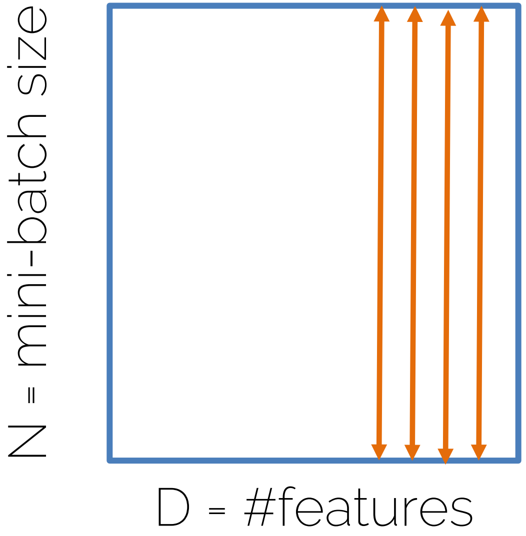
dimension

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

Ioffe and Szegedy 2015

Batch normalization

- In each dimension of the features, you have a unit gaussian



dimension

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

Ioffe and Szegedy 2015

Batch normalization

- In each dimension of the features, you have a unit Gaussian
- Is it ok to treat dimensions separately? Shown empirically that even if features are not decorrelated, convergence is still faster with this method

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

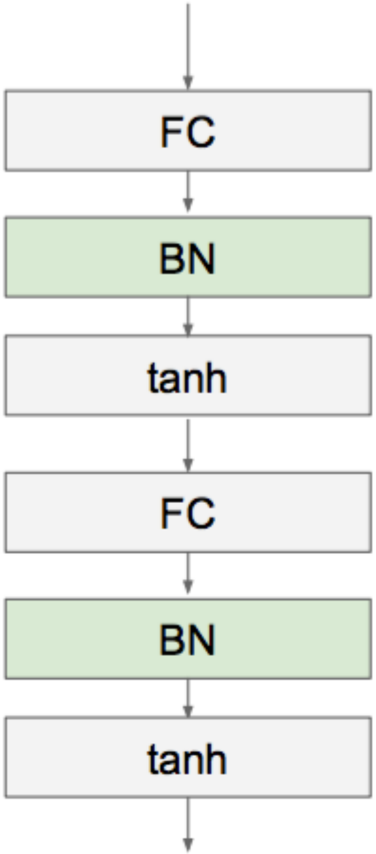
Differentiable function so we can backprop through it….

Ioffe and Szegedy 2015

Batch normalization

- A layer to be applied after Fully Connected (or Convolutional) layers and before non-linear activation functions

- Is it a good idea to have all unit Gaussians before tanh?



Ioffe and Szegedy 2015

Batch normalization

- Normalize

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

- Allow the network to change the range

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

backprop

The network can learn to undo the normalization

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

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

Ioffe and Szegedy 2015

BN for Exercise 2

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

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

Regularization

Regularization

- Any strategy that aims to

Lower validation error

Increasing training error

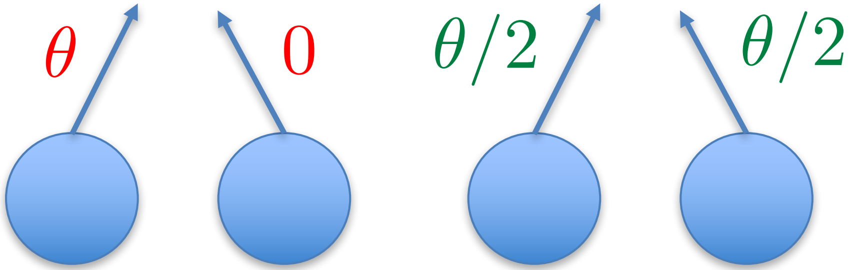
Weight decay

- L^2 regularization

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \epsilon \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_k, \mathbf{x}^i, \mathbf{y}^i) - \lambda \boldsymbol{\theta}_k^T \boldsymbol{\theta}_k$$

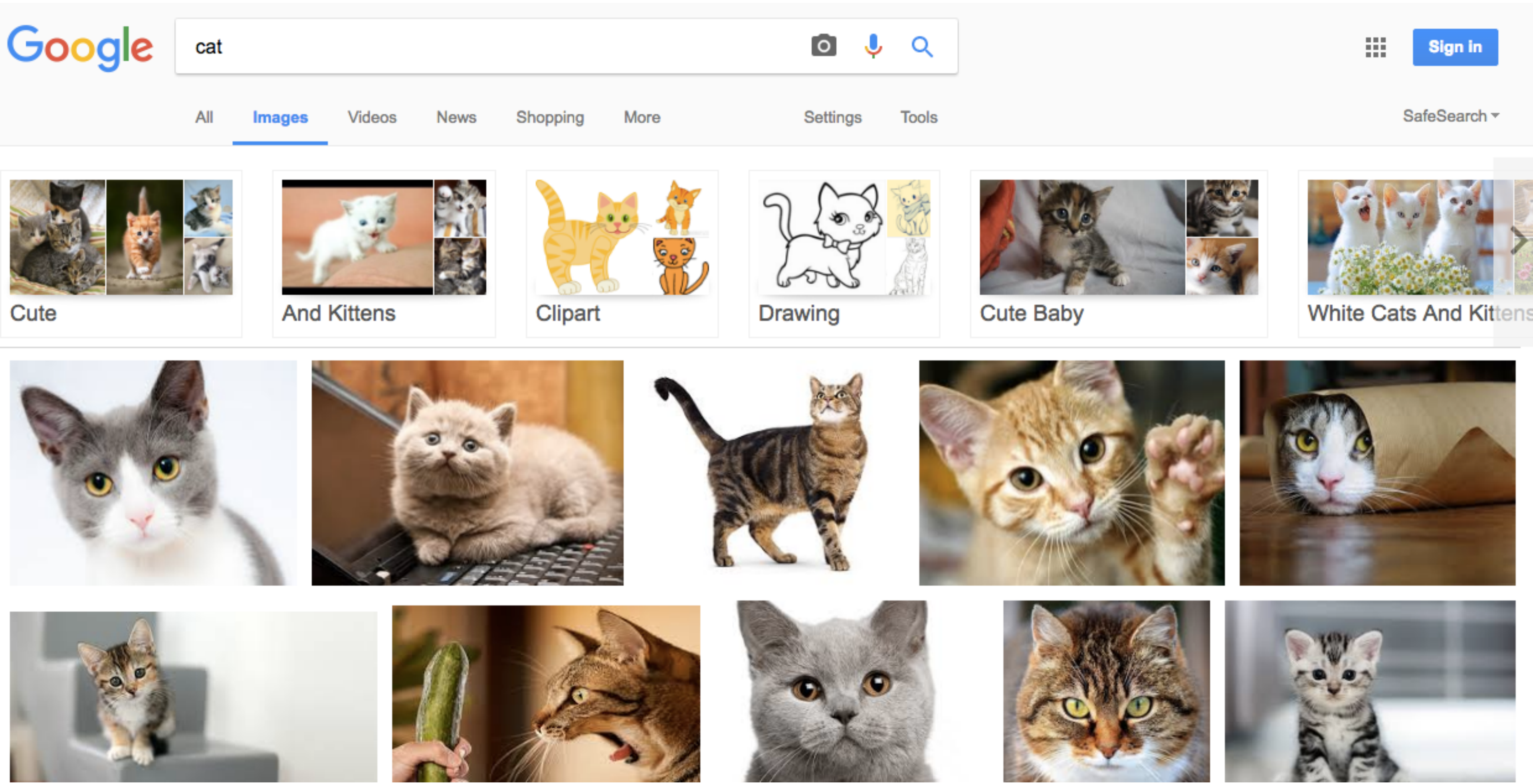
Learning rate — Gradient

- Penalizes large weights
- Improves generalization



Data augmentation

- A classifier has to be invariant to a wide variety of transformations



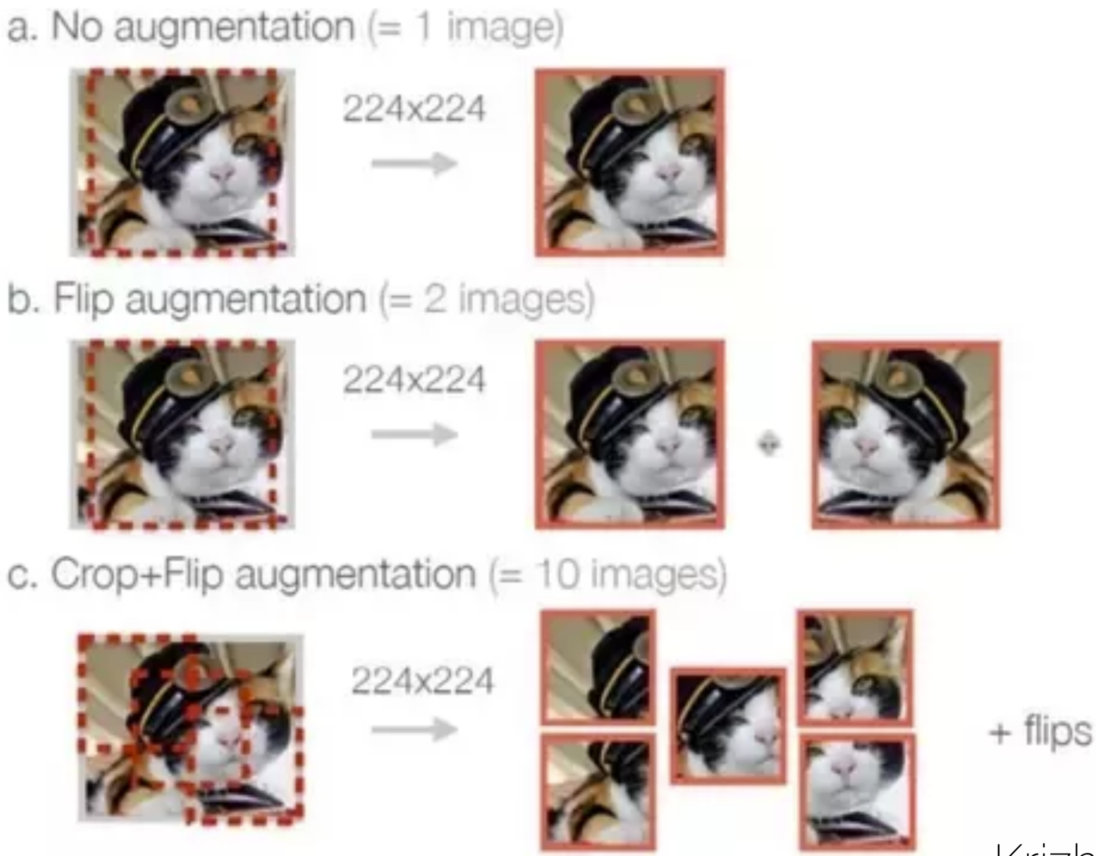
Pose Appearance Illumination

Data augmentation

- A classifier has to be invariant to a wide variety of transformations

- Helping the classifier: generate fake data simulating plausible transformations

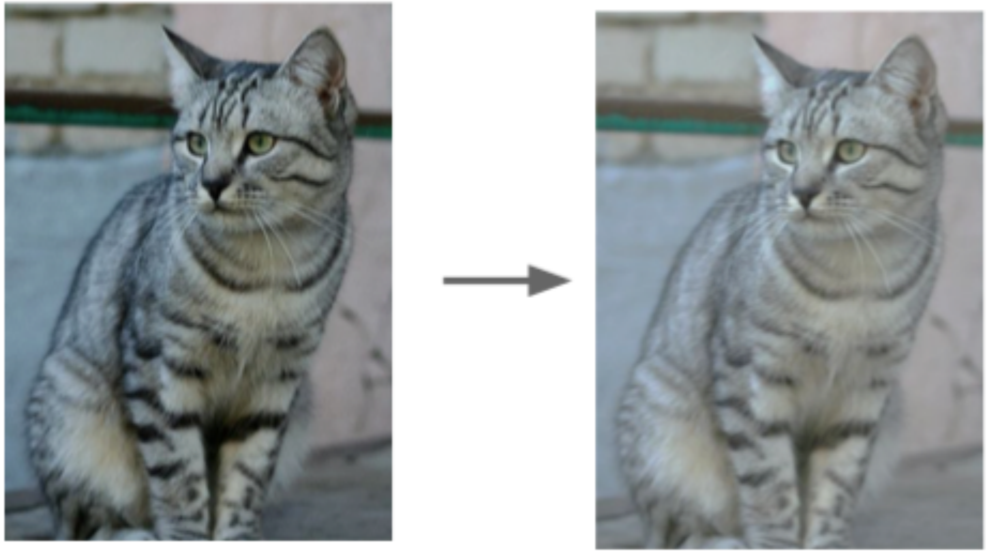
Data augmentation



Krizhevsky 2012

Data augmentation: random crops

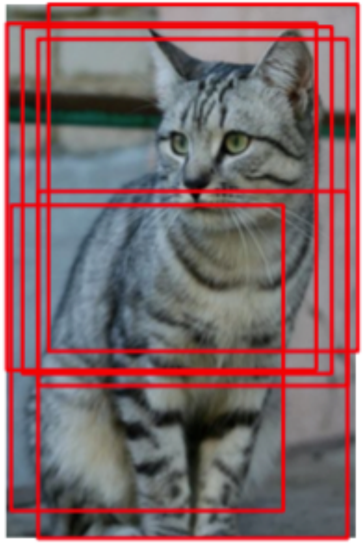
- Random brightness and contrast changes



Krizhevsky 2012

Data augmentation: random crops

- Training: random crops
  - Pick a random L in [256,480]
  - Resize training image, short side L
  - Randomly sample crops of 224x224



- Testing: fixed set of crops
  - Resize image at N scales
  - 10 fixed crops of 224x224: 4 corners + center + flips

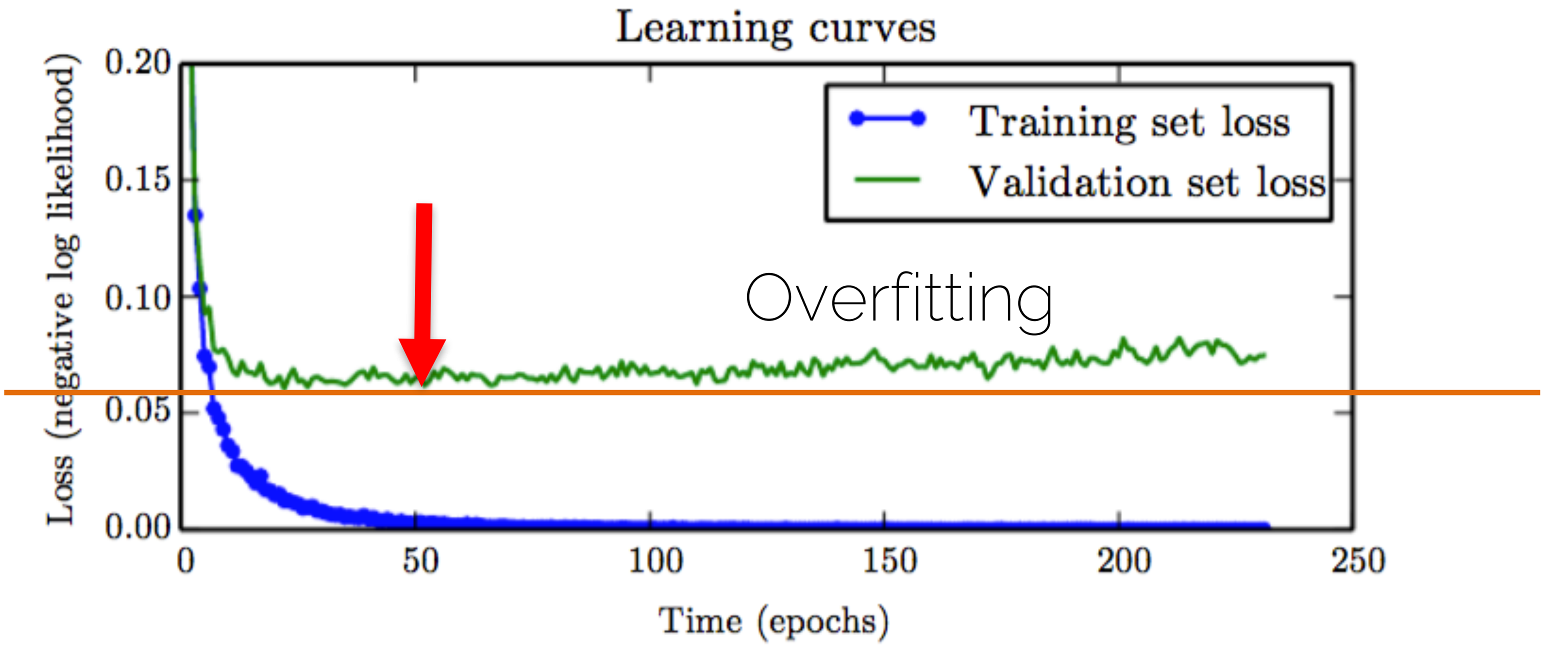
Krizhevsky 2012

Data augmentation

- When comparing two networks make sure to use the same data augmentation!

- Consider data augmentation a part of your network design

Early stopping



Training time is also a hyperparameter

Early stopping

- Easy form of regularization

$$\boldsymbol{\theta}_0 \xrightarrow{\epsilon} \boldsymbol{\theta}_1 \xrightarrow{\epsilon} \boldsymbol{\theta}_2 \qquad \boldsymbol{\theta}_s \qquad \qquad \boldsymbol{\theta}^*$$

$$\underbrace{\boldsymbol{\theta}_0 \quad \cdots \quad \boldsymbol{\theta}_s}_{\tau}$$

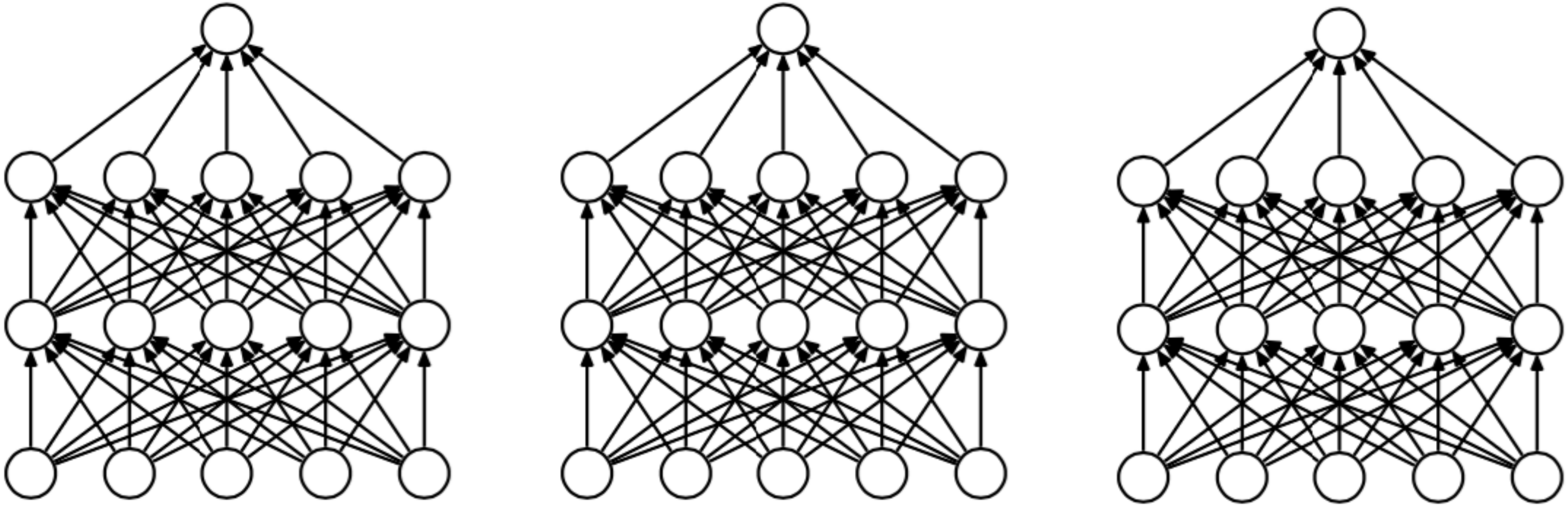
Overfitting

Bagging and ensemble methods

- Train three models and average their results
- Change a different algorithm for optimization or change the objective function
- If errors are uncorrelated, the expected combined error will decrease linearly with the ensemble size

Bagging and ensemble methods

- Bagging: uses k different datasets

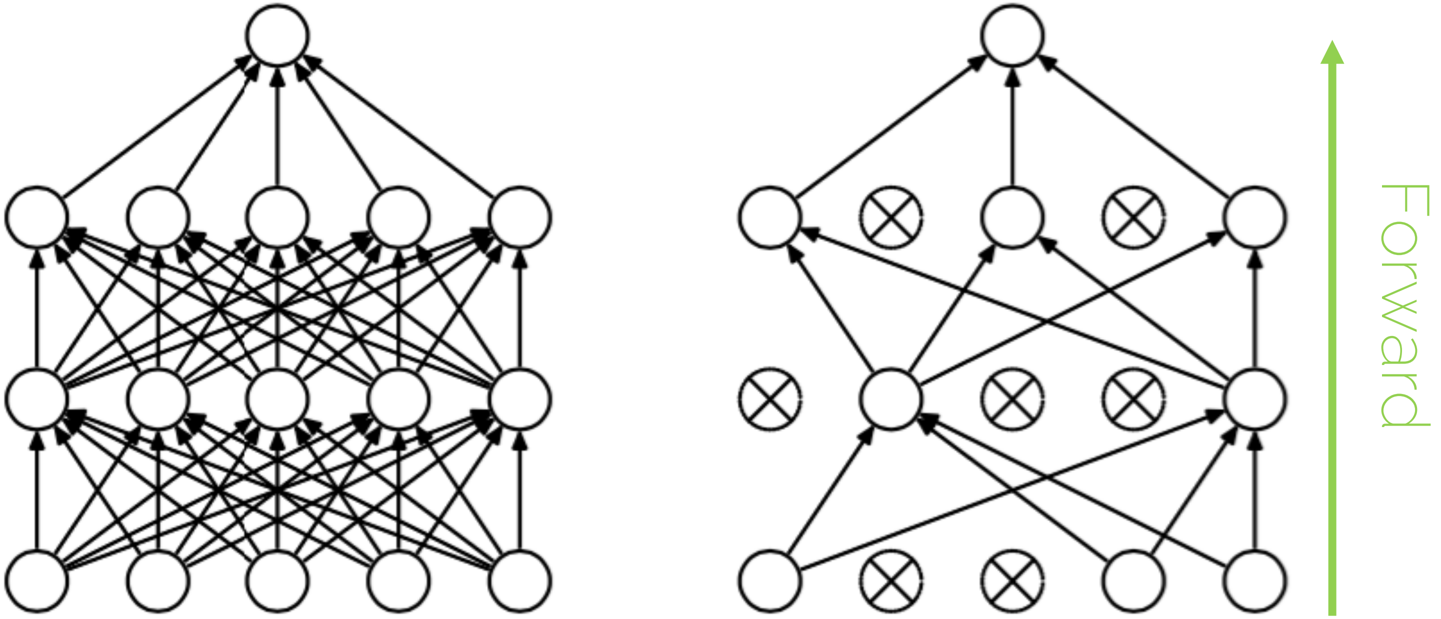


Training Set 1 Training Set 2 Training Set 3

Dropout

Dropout

- Disable a random set of neurons (typically 50%)



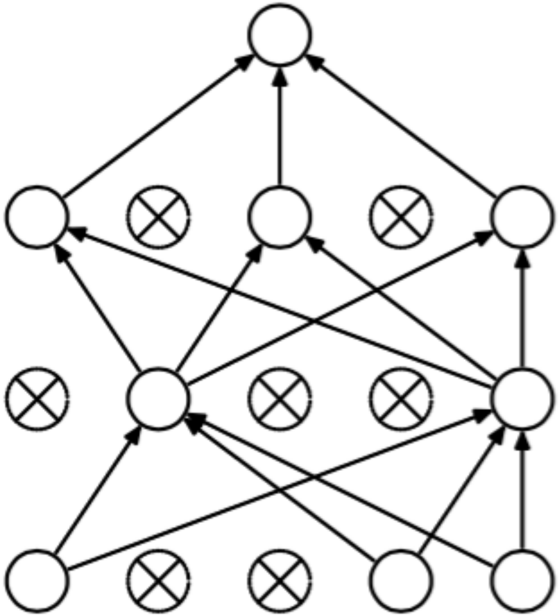
(a) Standard Neural Net (b) After applying dropout.

Srivastava 2014

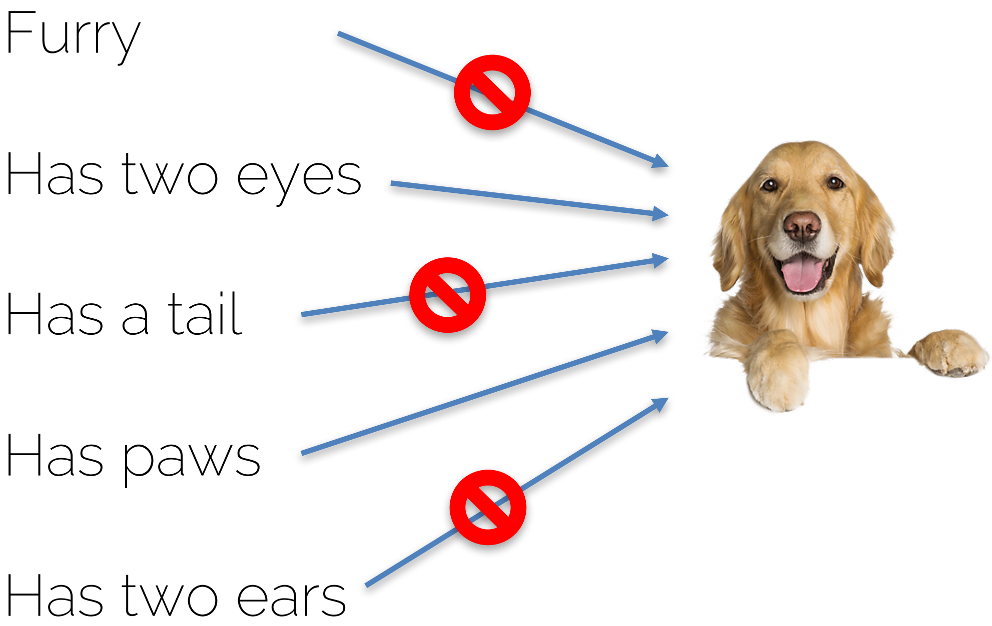
Dropout: intuition

Redundant representations

- Using half the network = half capacity



(b) After applying dropout.



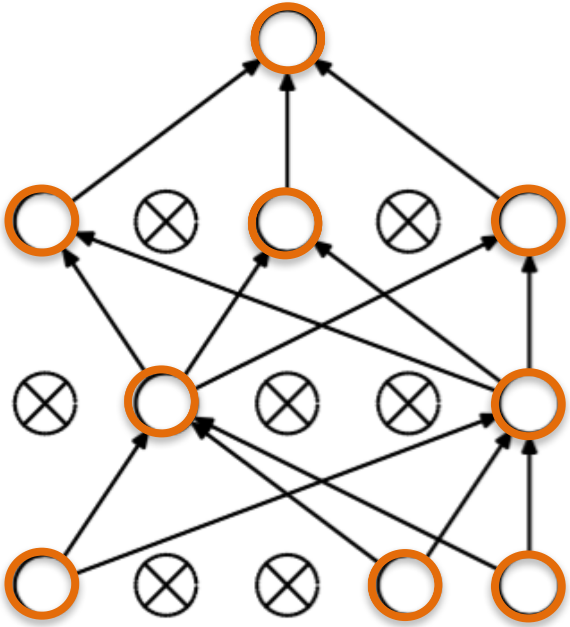
Dropout: intuition

- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- Consider it as model ensemble

Dropout: intuition

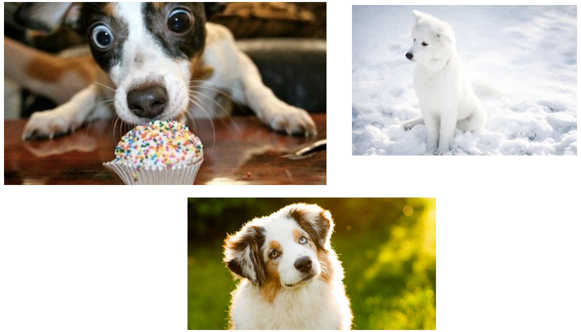
- Two models in one



(b) After applying dropout.

 Model 1

 Model 2



Dropout: intuition

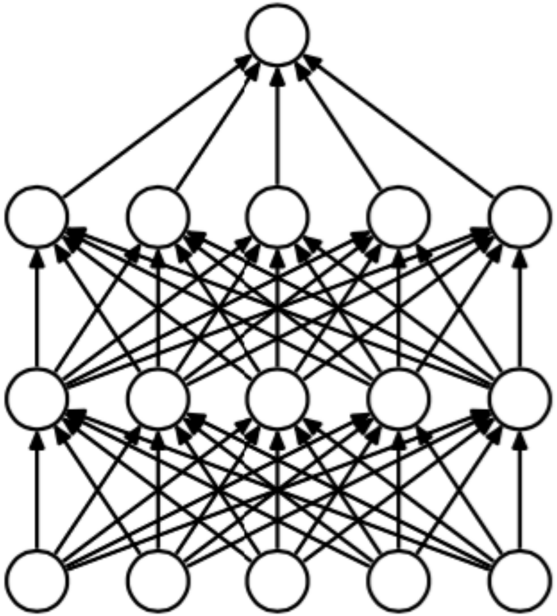
- Using half the network = half capacity
  - Redundant representations
  - Base your scores on more features

- Consider it as two models in one
  - Training a large ensemble of models, each on different set of data (mini-batch) and with SHARED parameters

Reducing co-adaptation between neurons

Dropout: test time

- All neurons are “turned on” – no dropout



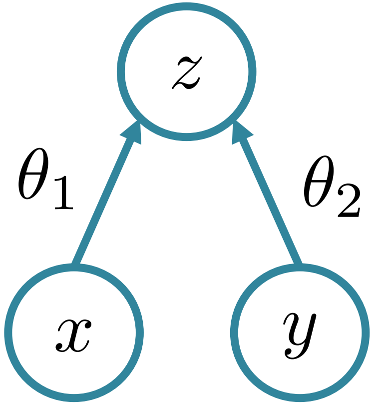
Conditions at train and test time are not the same

Dropout: test time

Dropout probability p=0.5

- Test: $z = \theta_1 x + \theta_2 y$

- Train: $$\mathbf{E}[z] = \frac{1}{4}(\theta_1 0 + \theta_2 0 + \theta_1 x + \theta_2 0 + \theta_1 0 + \theta_2 y + \theta_1 x + \theta_2 y) = \frac{1}{2}(\theta_1 x + \theta_2 y)$$



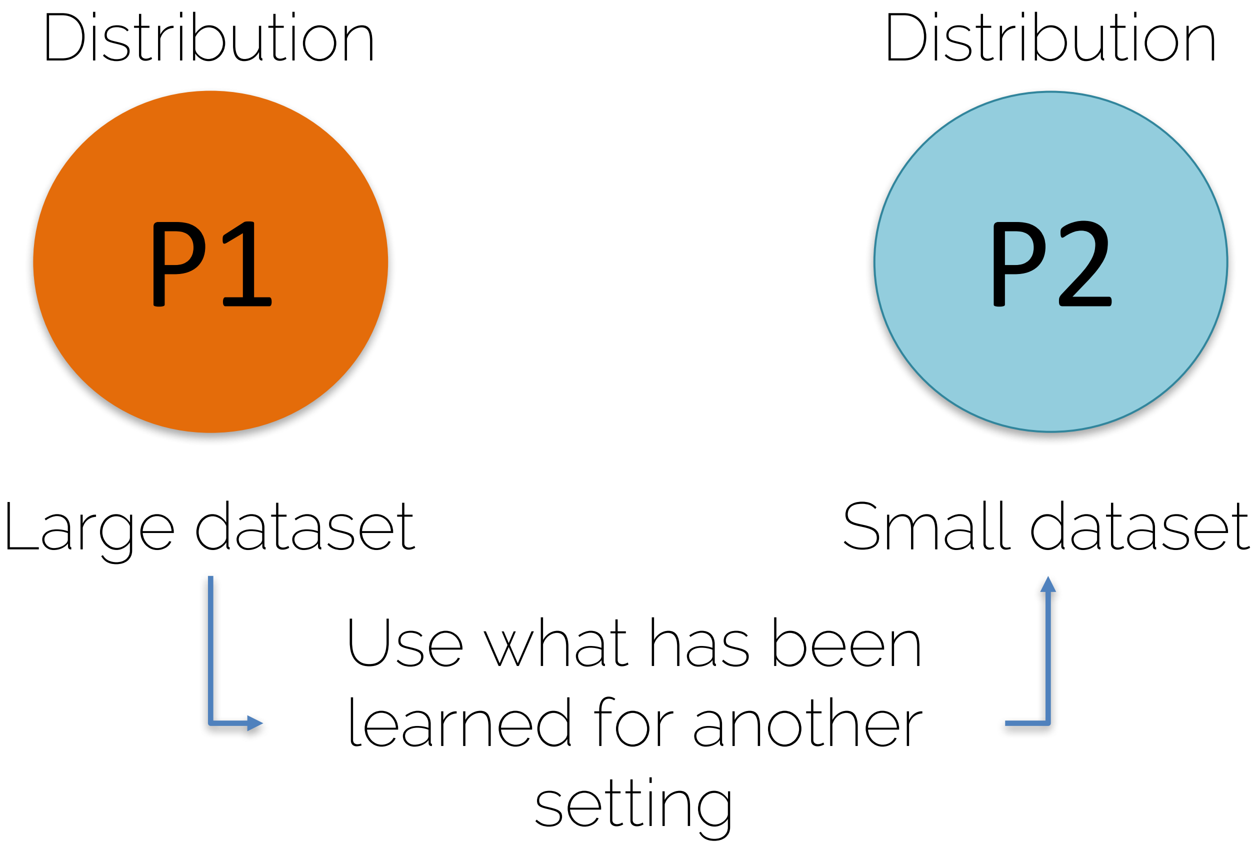
Weight scaling inference rule

Dropout: verdict

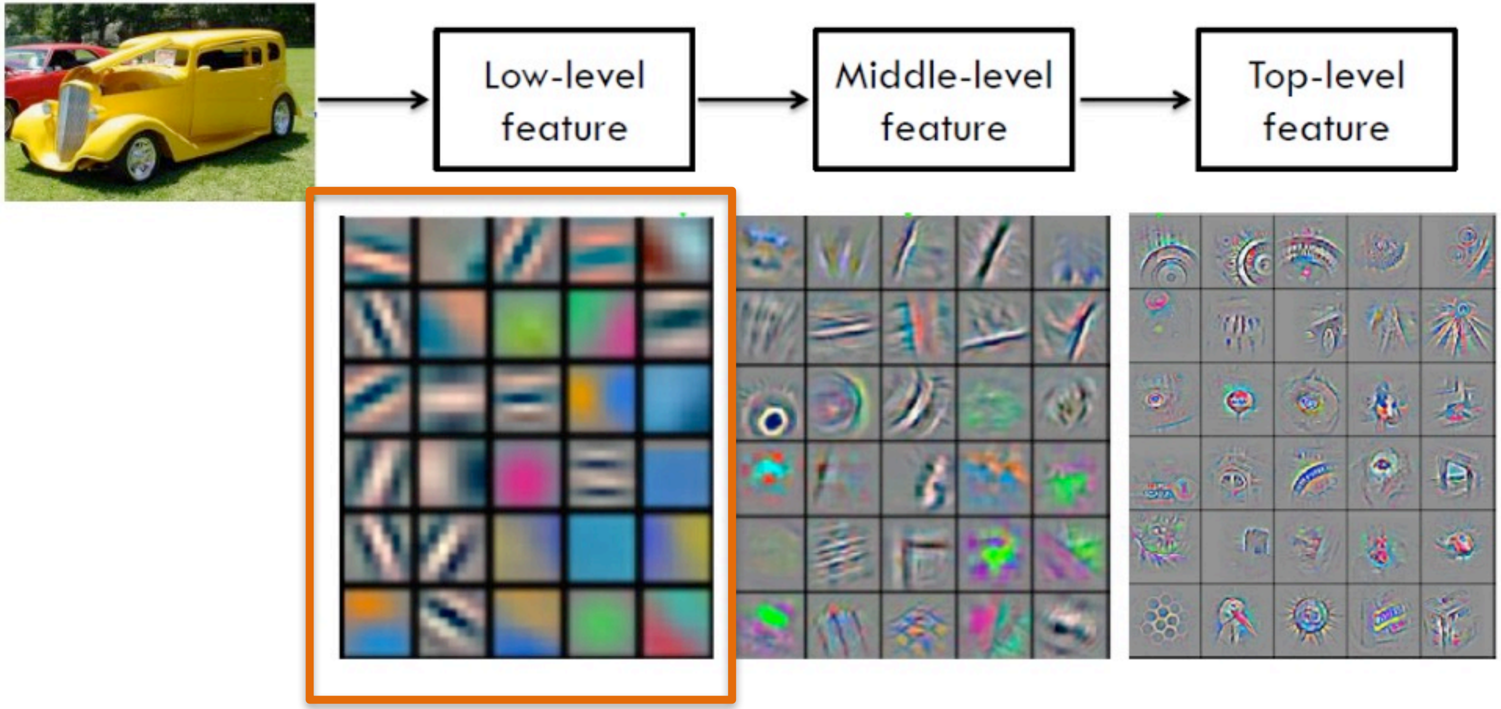
- Efficient bagging method with parameter sharing
- Use it!
- Dropout reduces the effective capacity of a model → larger models, more training time

Transfer learning

Transfer learning



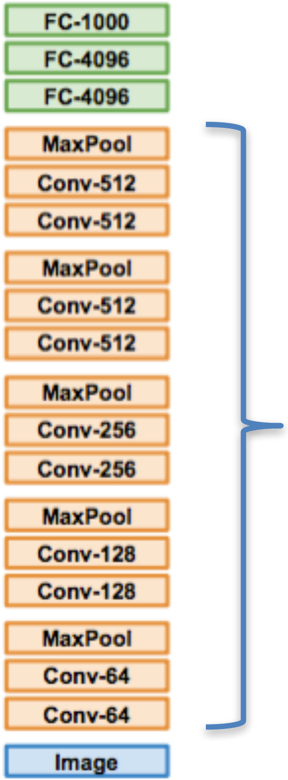
Transfer learning for images



Zeiler and Fergus 2013

Transfer learning

Trained on ImageNet

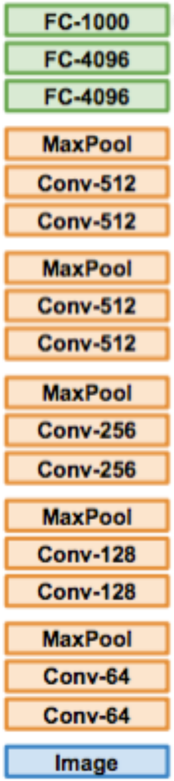


Feature extraction

Donahue 2014, Razavian 2014

Transfer learning

Trained on ImageNet



Decision layers

Parts of an object (wheel, window)

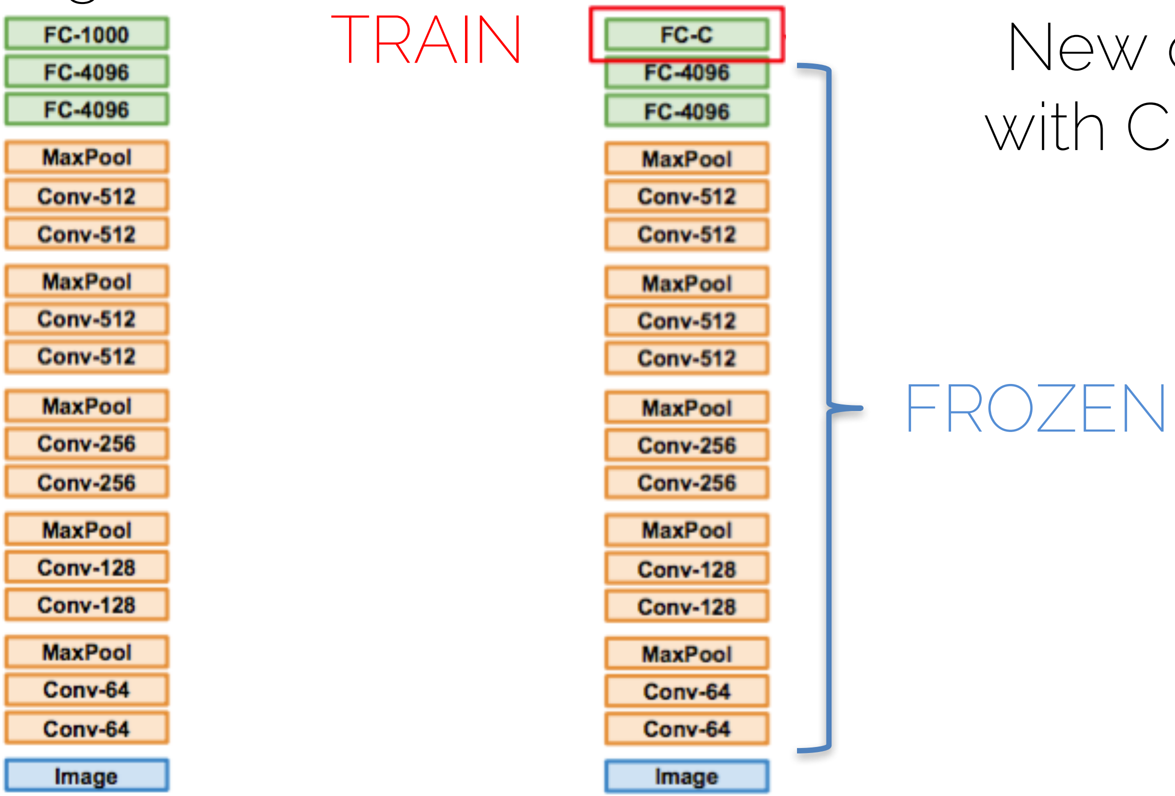
Simple geometrical shapes (circles, etc)

Edges

Donahue 2014, Razavian 2014

Transfer learning

Trained on ImageNet

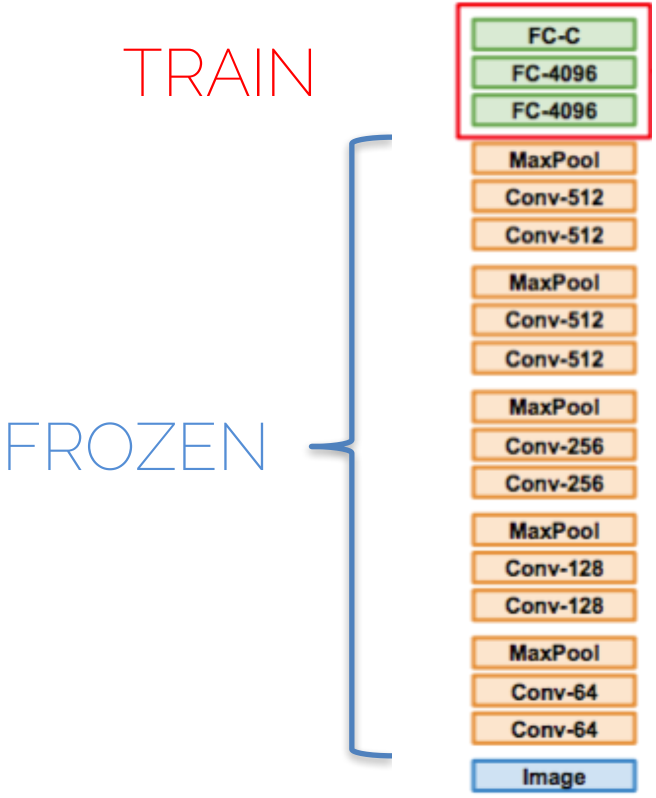


New dataset with C classes

Donahue 2014, Razavian 2014

Transfer learning

If the dataset is big enough train more layers with a low learning rate



Donahue 2014, Razavian 2014

For your projects

- Find a large dataset related to your problem and train your network there

OR

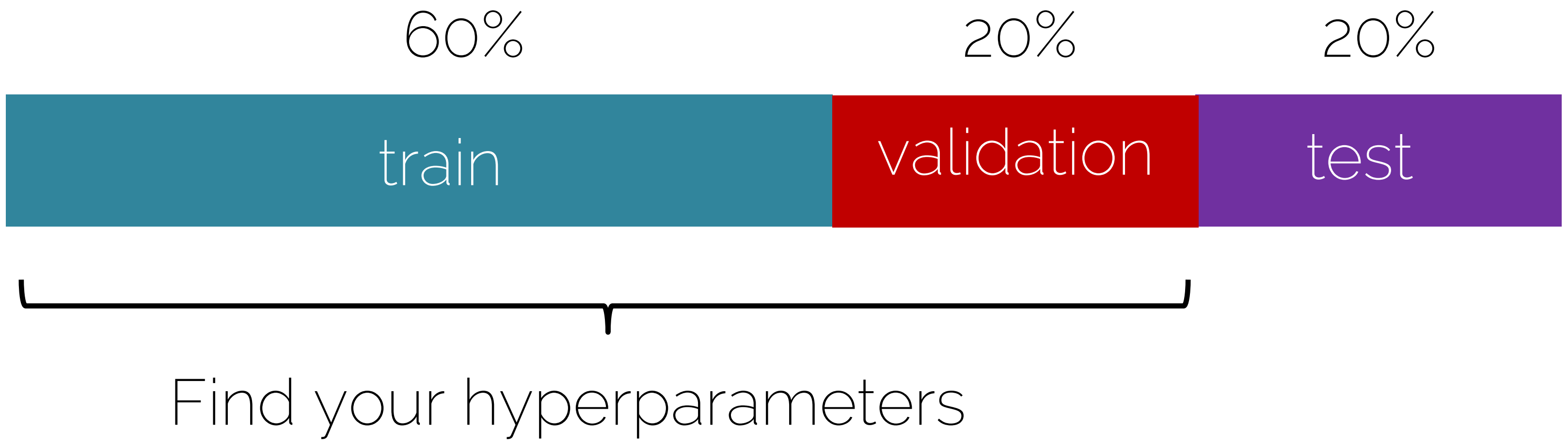
- Take the pre-trained weights from e.g. ImageNet

- Do transfer learning by fine-tuning on you small datasets

Basic recipe for machine learning

Basic recipe for machine learning

- Split your data



Basic recipe for machine learning

- Split your data

Human level error 1%

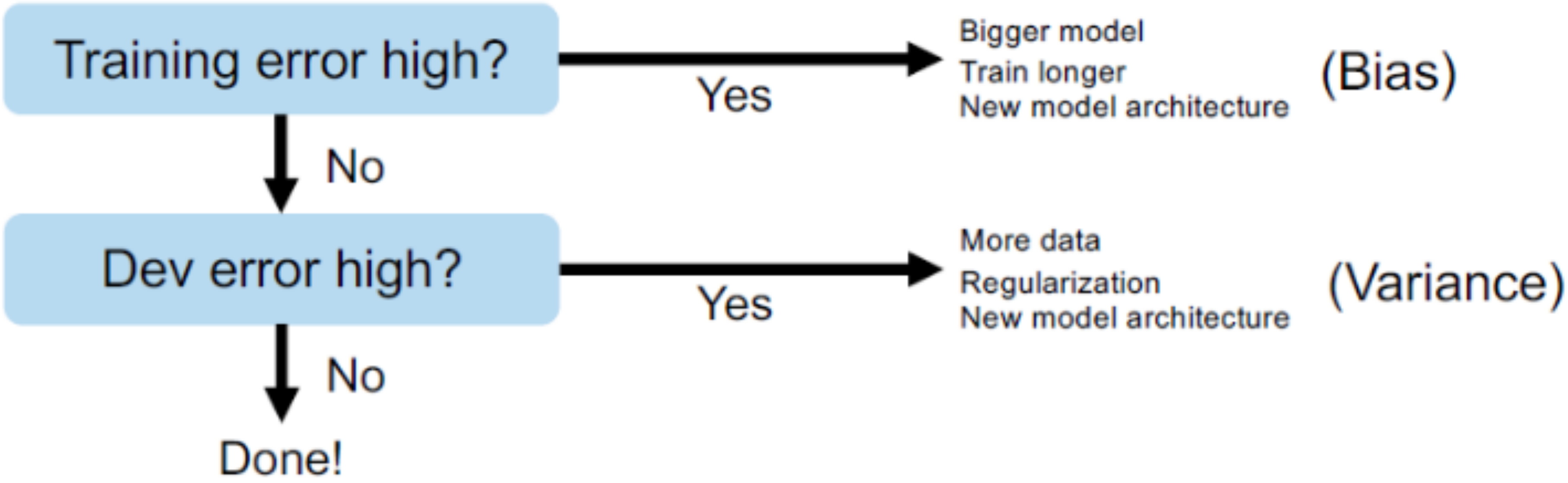
Training set error 5%

Val/Dev set error 8%

Bias (or underfitting)

Variance (overfitting)

Basic recipe for machine learning



Credits: Andrew Ng

Basic recipe for machine learning

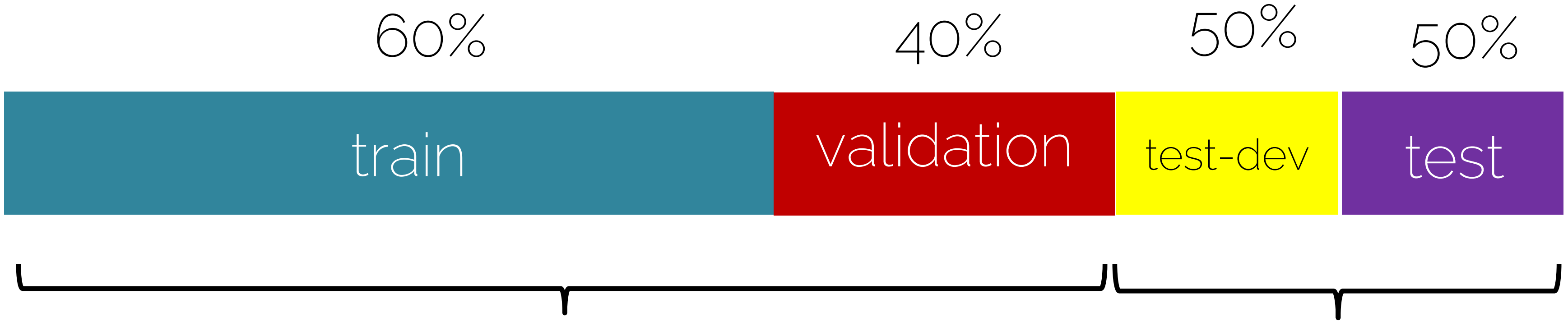
- You train and test do no come from the same source

Training data (e.g. speech data)

Test data (e.g. speech data inside a helicopter)

Basic recipe for machine learning

- dev/val and test set must come from same distribution

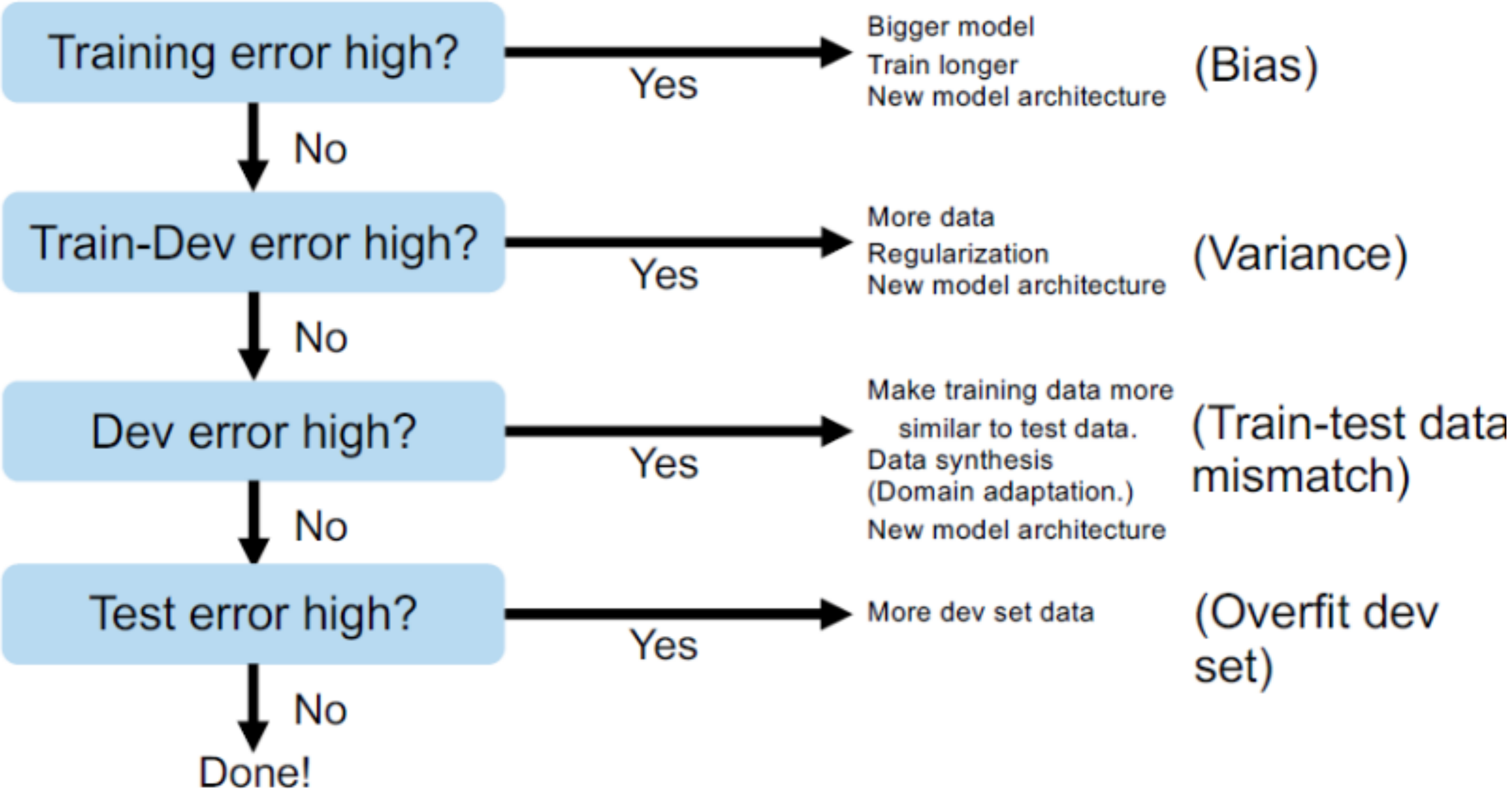


Training data (e.g. speech data)

Test data (e.g. speech data inside a helicopter)

Basic recipe for machine learning

Human level error	...... 1%	
		Bias
Training set error	.... 1.1%	
		Variance
Train-Dev set error	.... 1.5%	
		Data mismatch
Test-Dev set error	.... 8%	
		Overfitting to dev
Test set error	.... 8.5%	



Credits: Andrew Ng

Administrative Things

- Next Thursday June 8th: CNN
- Tomorrow: Solution 2nd exercise, presentation 3rd