11210IPT553000 Deep Learning in Biomedical Optical Imaging

Week4

Improving Deep Neural Networks

Hyperparameter Tuning, Regularization and Optimization

Instructor: Hung-Wen Chen @NTHU, Fall 2023
2023/10/02

SPONSORED BY

DEEP LEARNING INSTITUTE

DEEP LEARNING INSTITUTE
CERTIFIED INSTRUCTOR

Outline of Today's Lecture

- Practical aspects of Deep Learning (Course 2 Week 1)

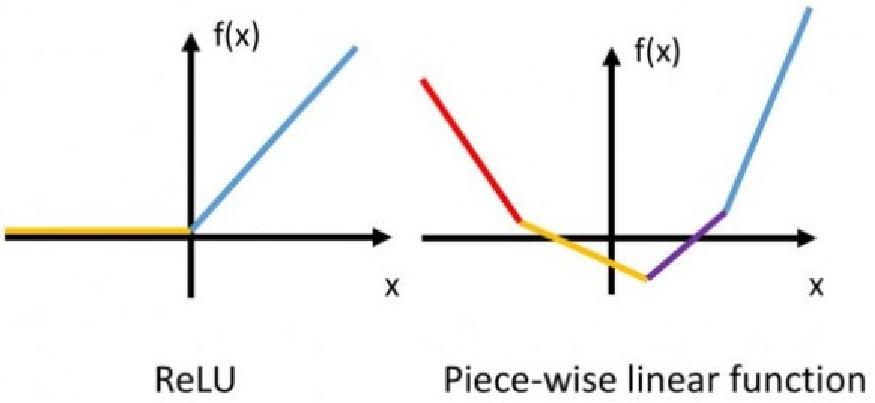
- Lab Practice: Hyperparameter Tuning

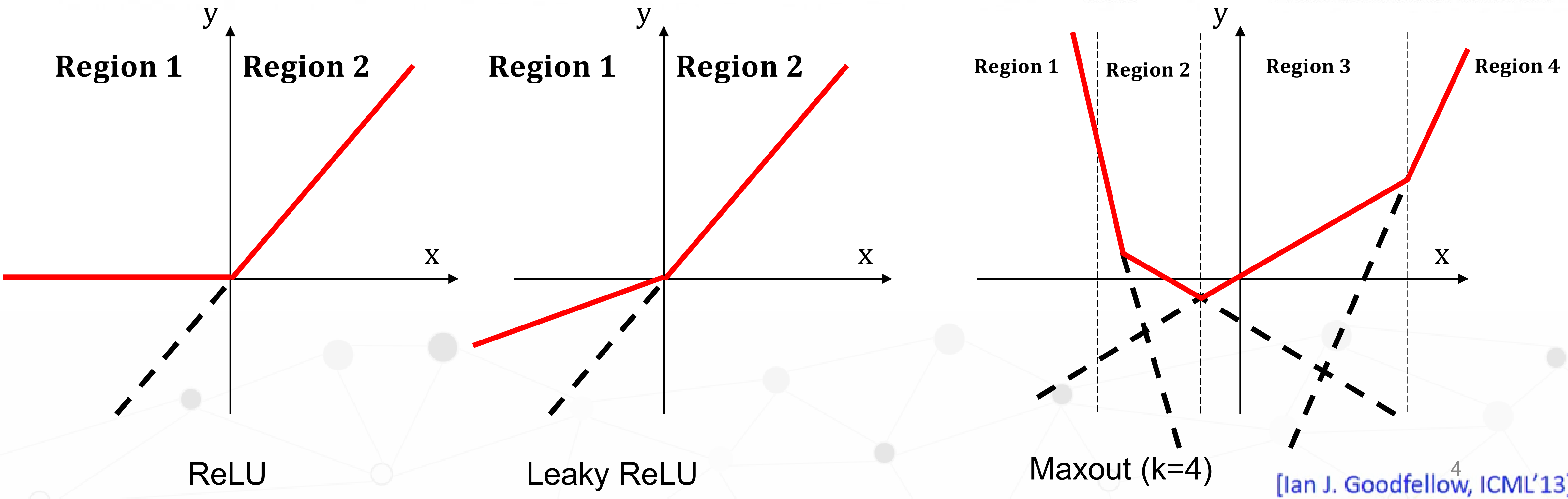
More Activation Functions

Maxout

- The Maxout activation is a generalization of the ReLU and the leaky ReLU functions.
- It is a learnable activation function.
- It is a piecewise linear function that returns the maximum of the inputs, designed to be used in conjunction with the dropout regularization technique.
- But, it doubles the no. of parameter for each return, so there is a trade-off.

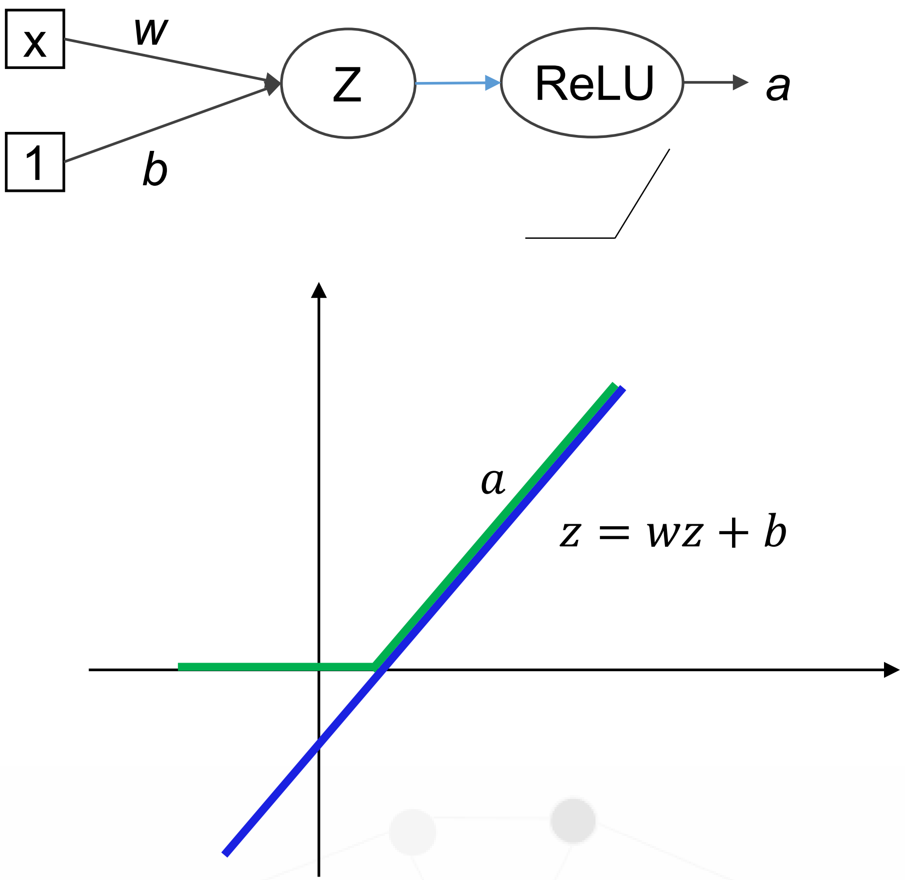
Maxout

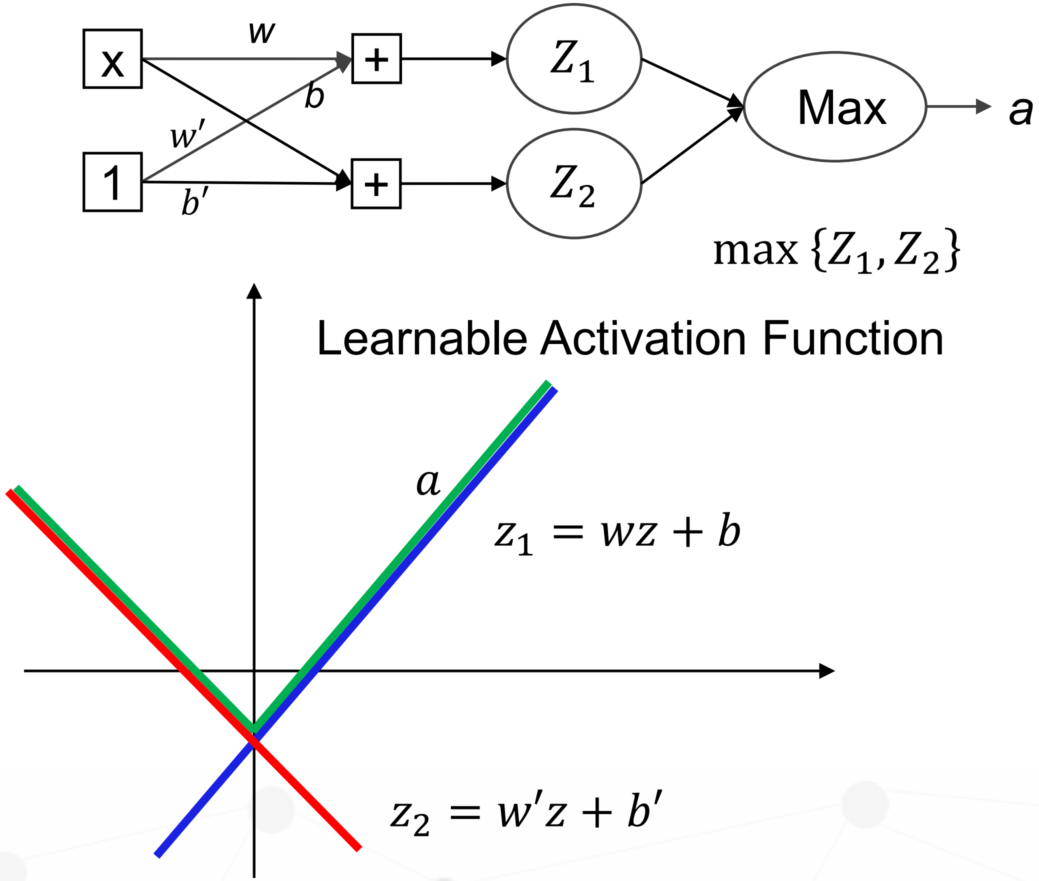




[Ian J. Goodfellow, ICML'13]

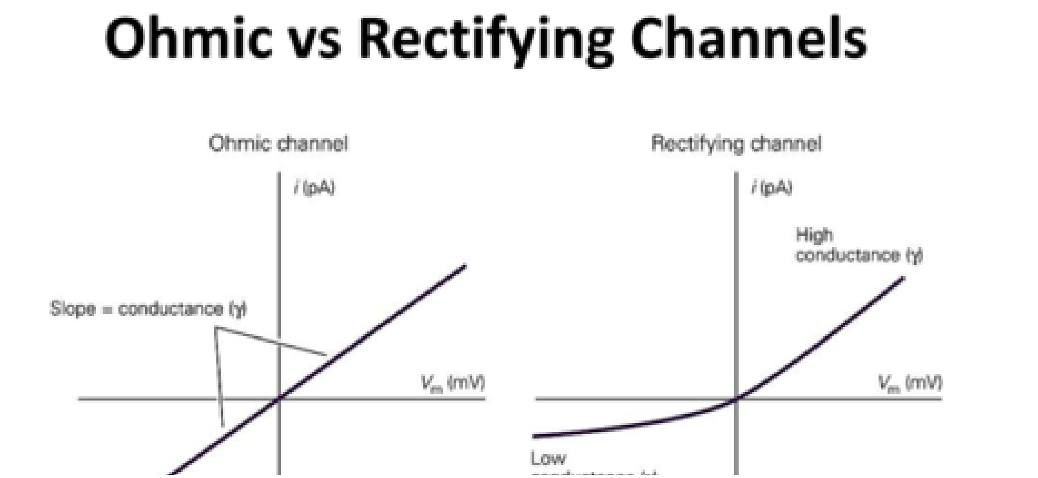
Maxout

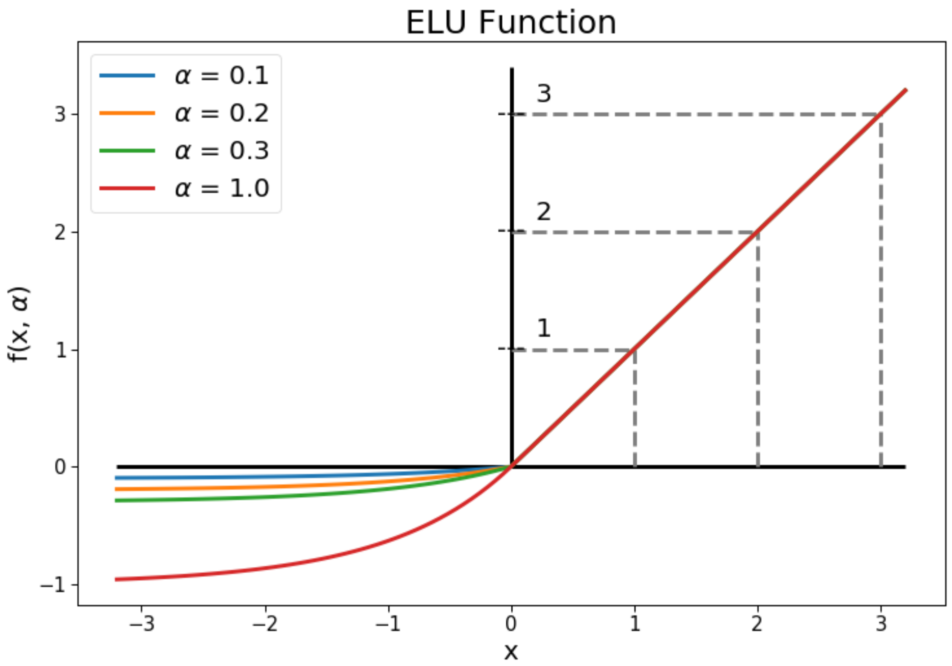

x
1
w
b
Z
ReLU
a
z = wz + b


x
1
w
b
w'
b'
+
+
Z1
Z2
Max
a
max {Z1, Z2}
Learnable Activation Function
z1 = wz + b
z2 = w'z + b'

Activation Function

I-V relationship of ion channels





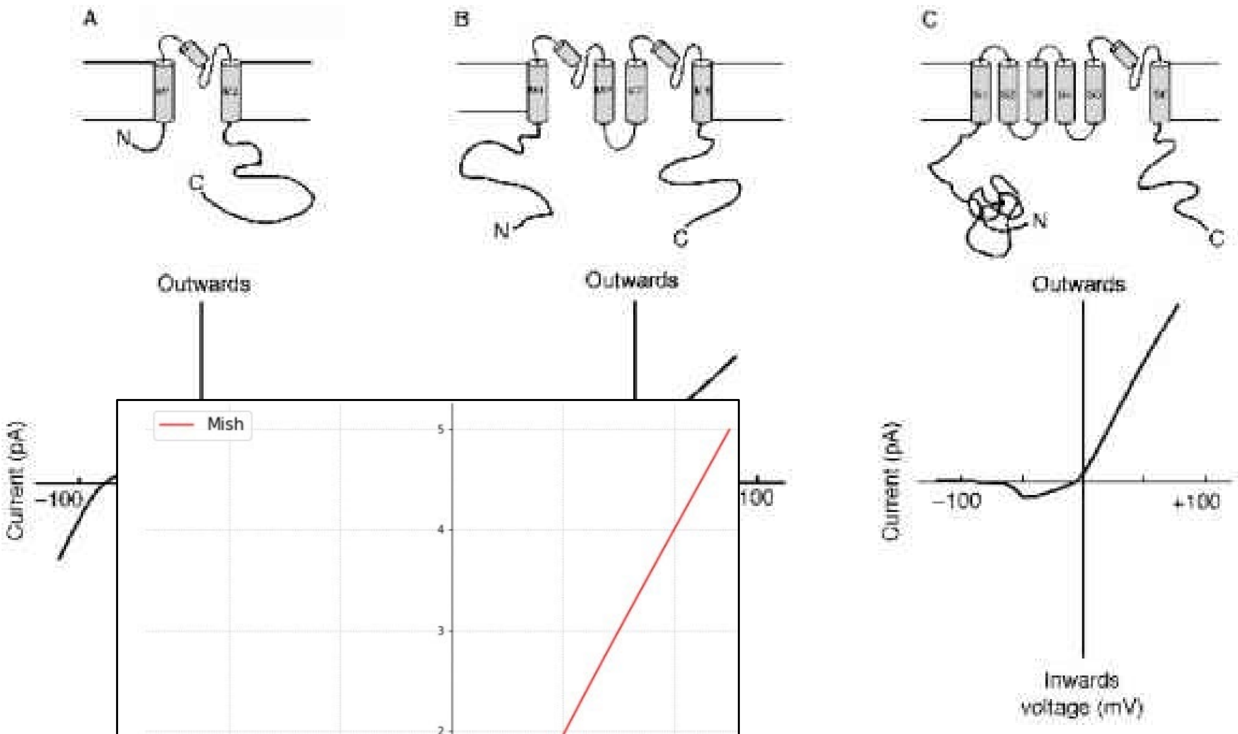


Figure 1. Mish Activation Function

[Mish: A Self Regularized Non-Monotonic Activation Function]

Practical aspects of Deep Learning

Train/dev/test sets

Applied ML is a highly iterative process

Hyperparameters

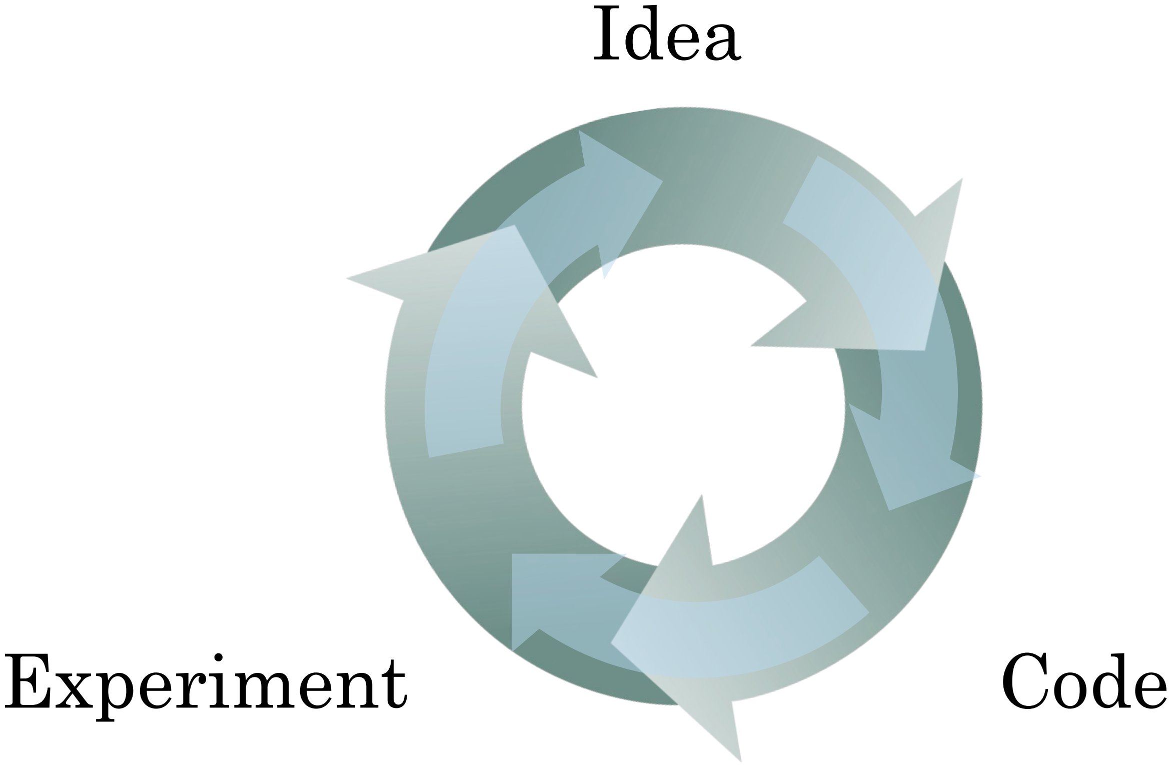
layers

hidden units

learning rates

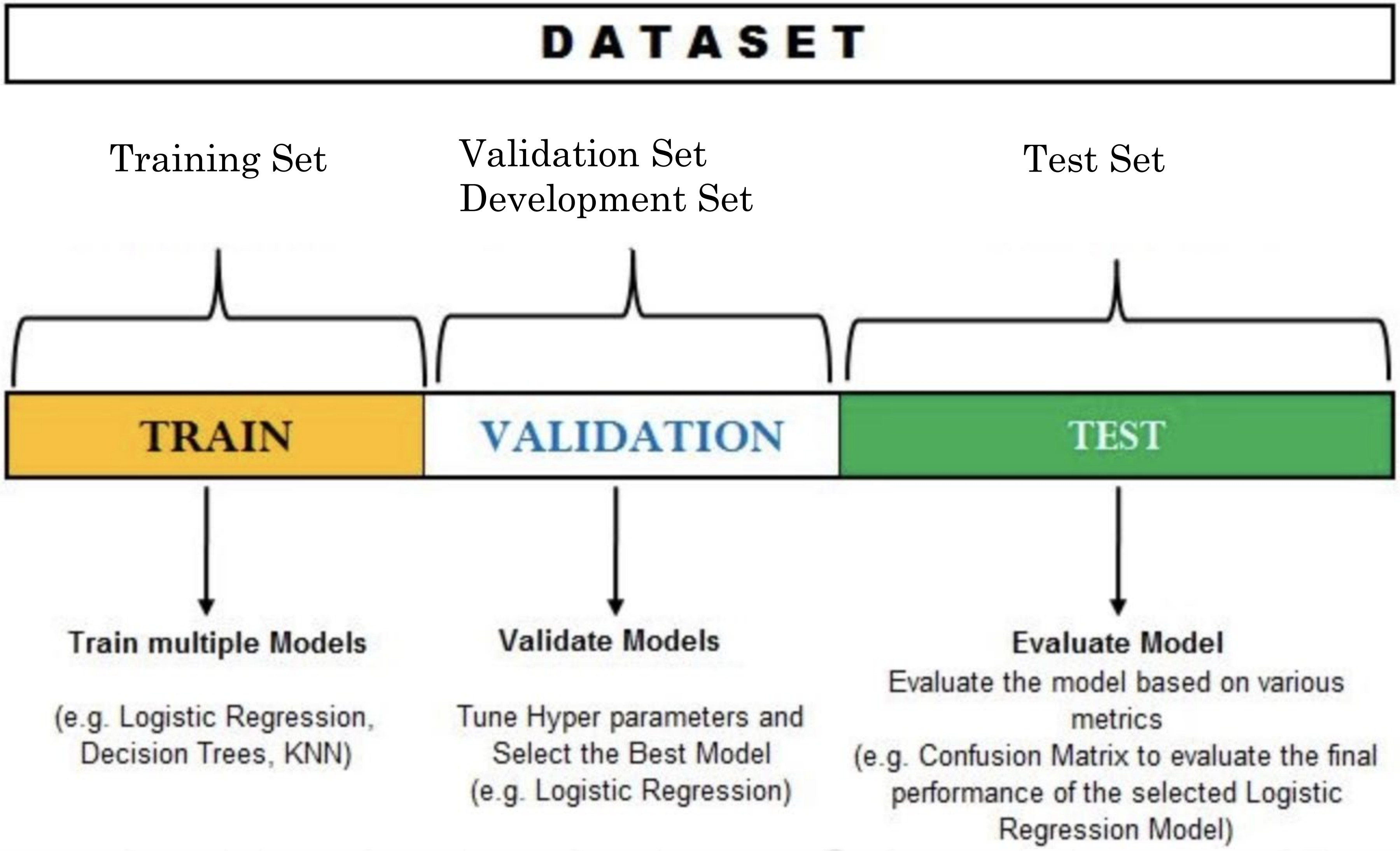
activation functions

…



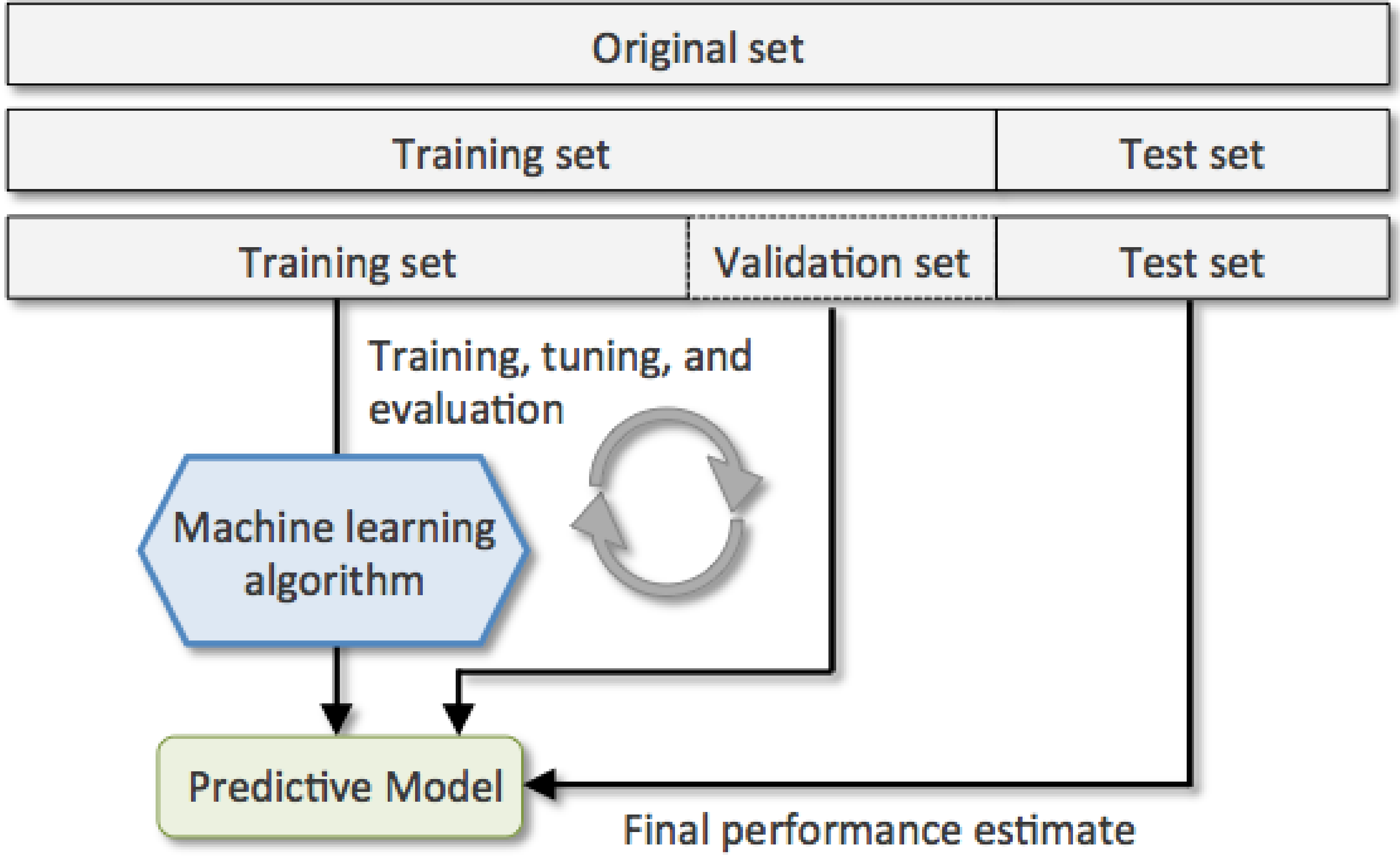
Train/dev/test sets

Train/dev/test sets



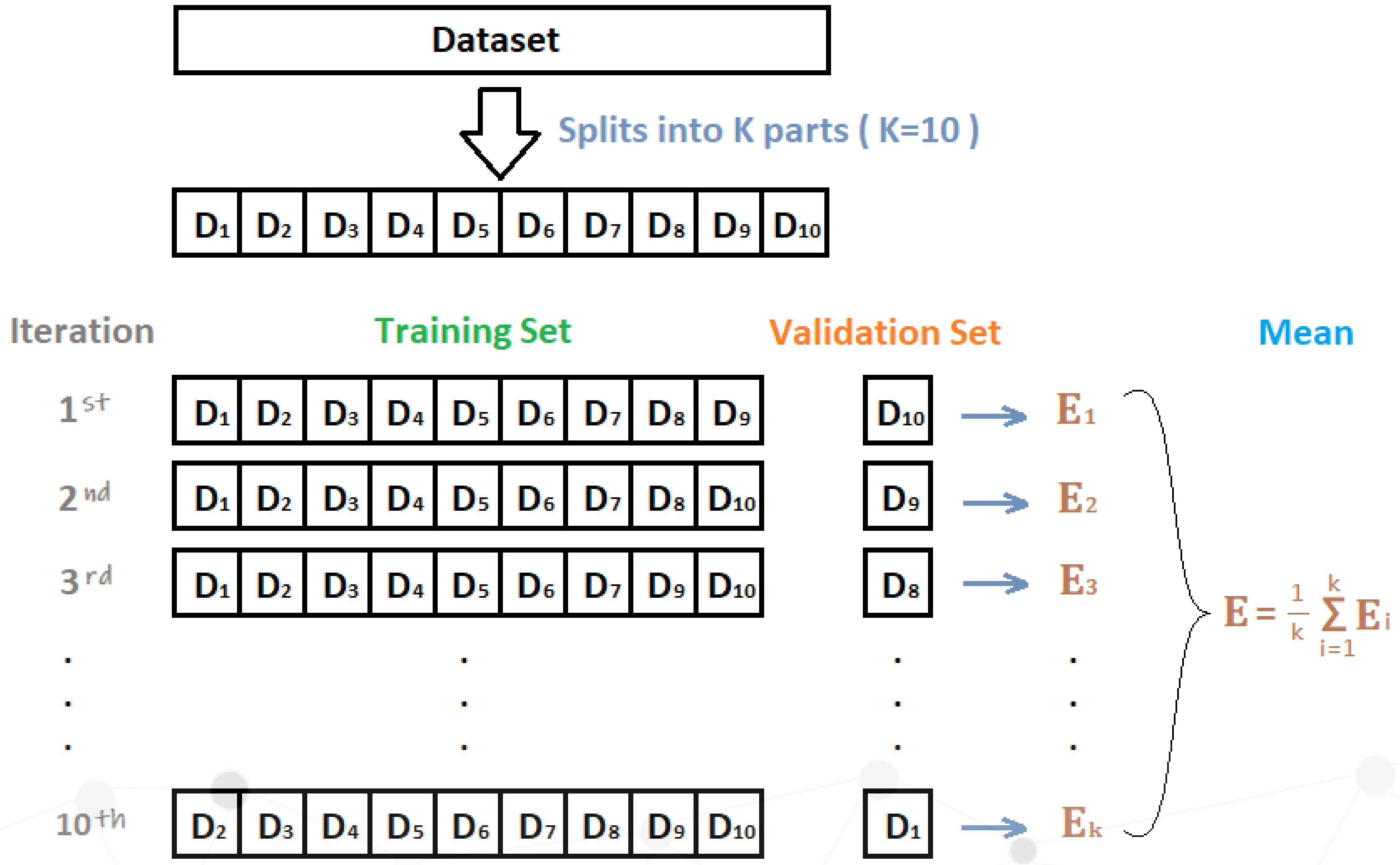
Train/dev/test sets

Hold-out Cross-Validation



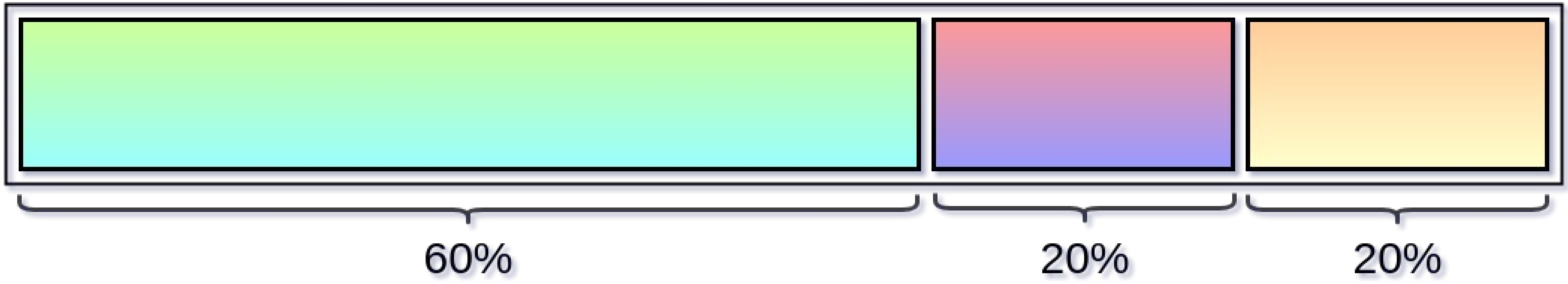
Train/dev/test sets

K-fold Cross-Validation

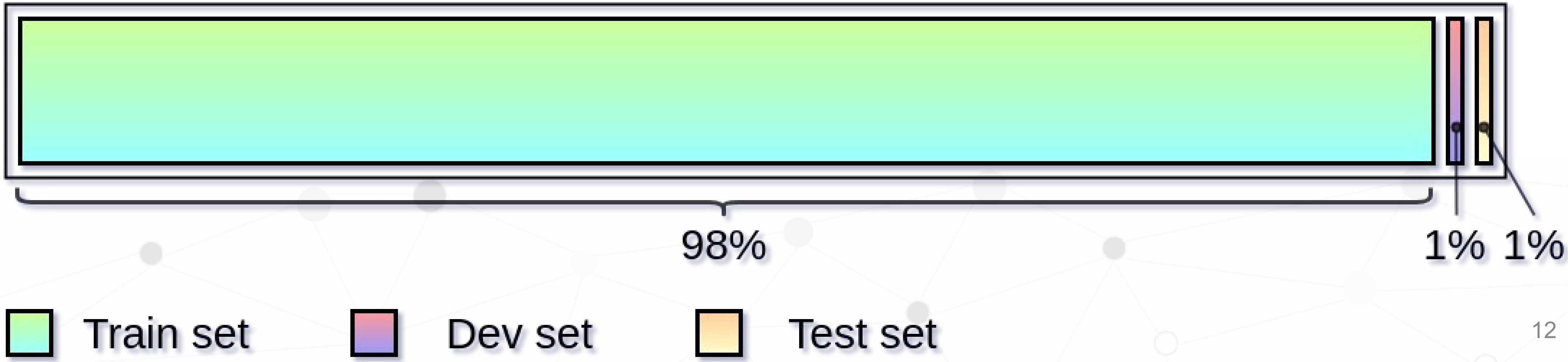


Proportion

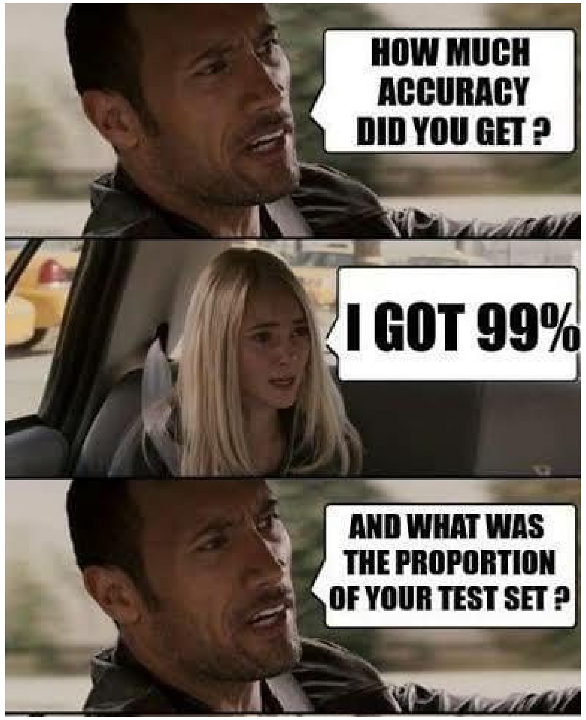
Small dataset (100, 1000, or <10,000)



Big dataset >1,000,000



Train/dev/test sets



Mismatched train/test distribution

Training set:
Cat pictures from webpages

(high-resolution & professional photos)

Dev/test sets:
Cat pictures from users using your app

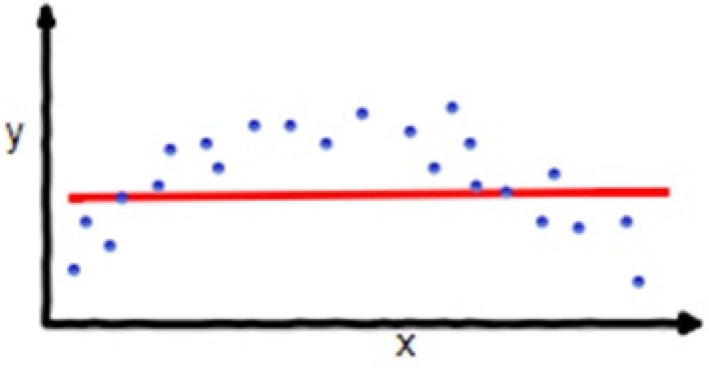
(blurry and low-resolution photos taken by cell phone)

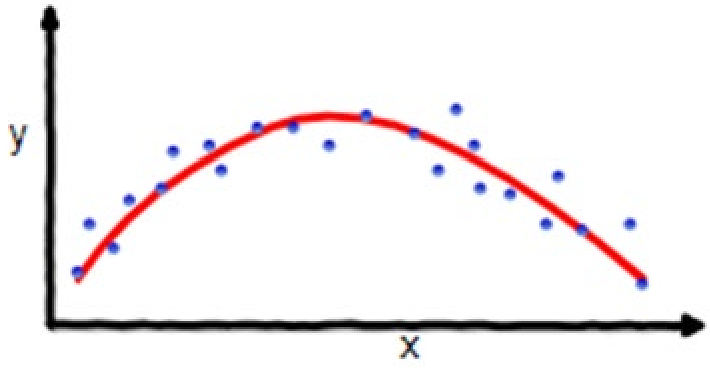
Make sure Dev set and Test set come from the same distribution

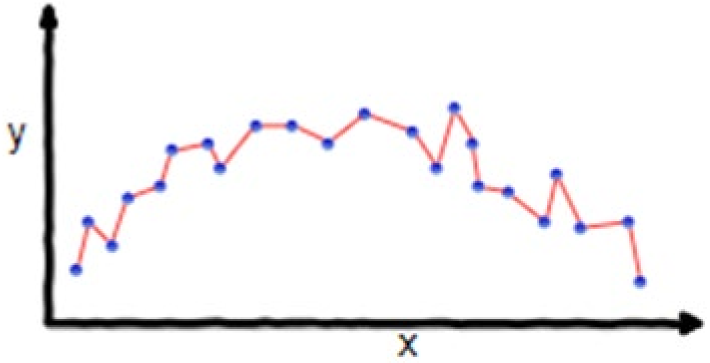
Not having a test set might be okay. (Only dev set.)

Bias and variance

Model







high bias

underfitting

"Just right"

high variance

overfitting

Bias and Variance

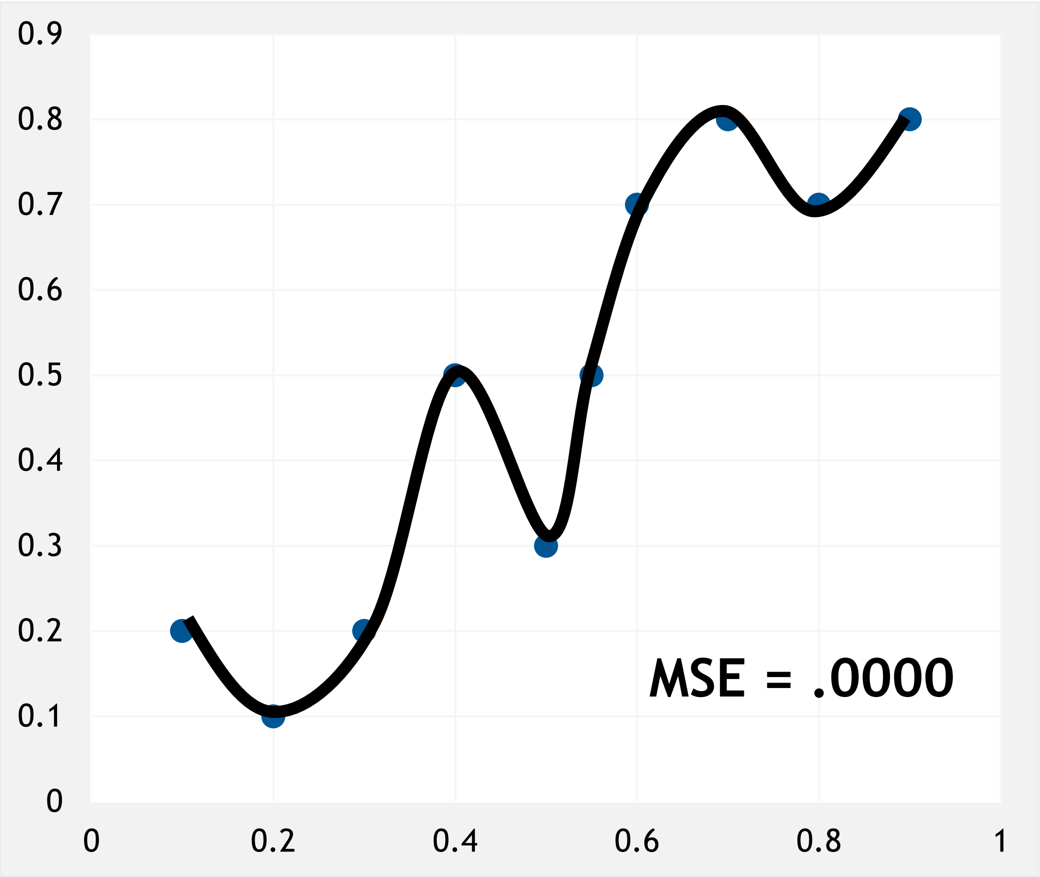
Cat classification

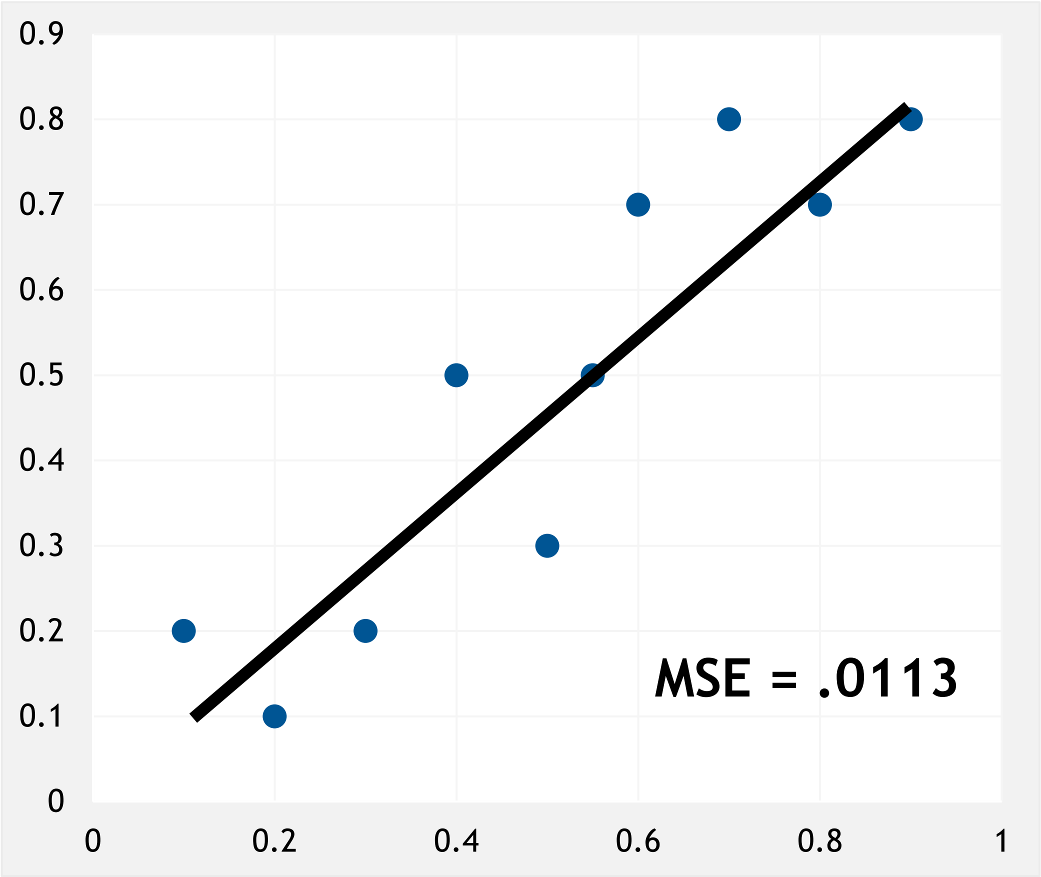
Train set error:	1%	15%	15%	0.5%
Dev set error:	11%	16%	30%	1%
	low bias	high bias	high bias	low bias
	high variance	low variance	high variance	low variance
	overfitting	underfitting		

Human error (Bayes error): ~0%

Overfitting

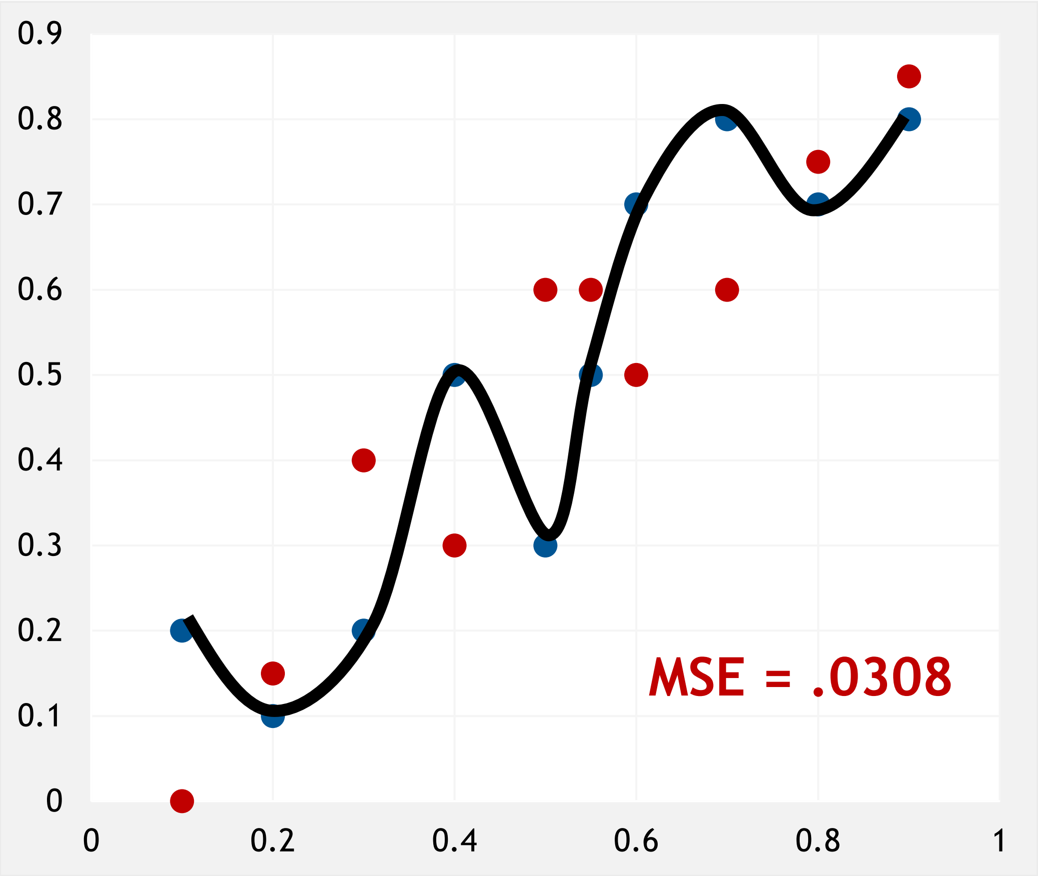
Which Trendline is Better?

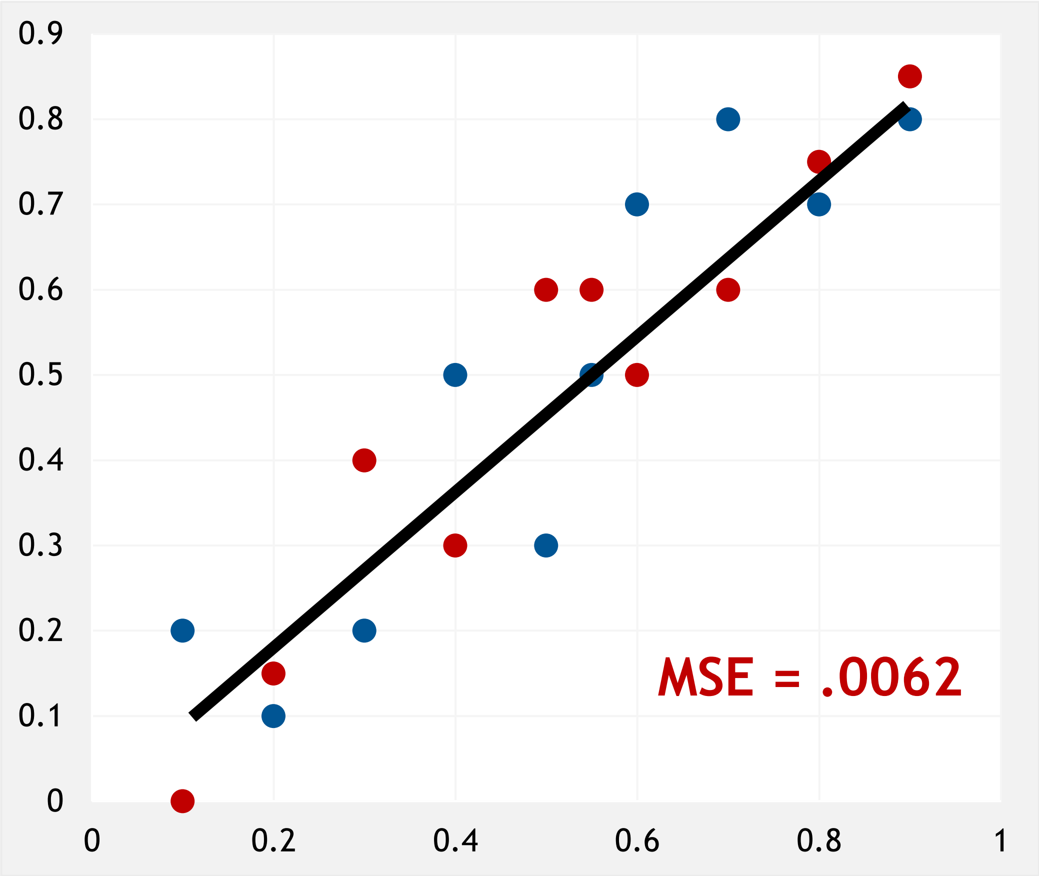




Overfitting

Which Trendline is Better?





TRAINING VS VALIDATION DATA

Avoid memorization

Training data

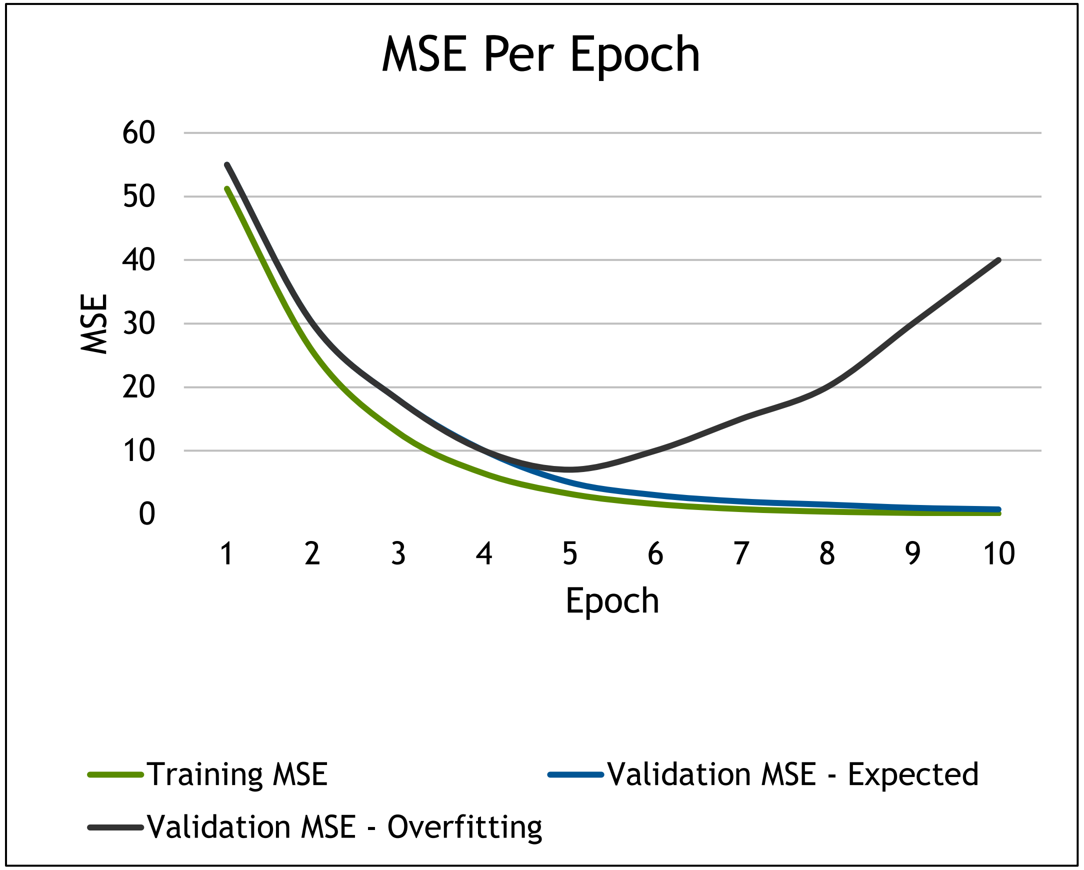
- Core dataset for the model to learn on

Validation data

- New data for model to see if it truly understands (can generalize)

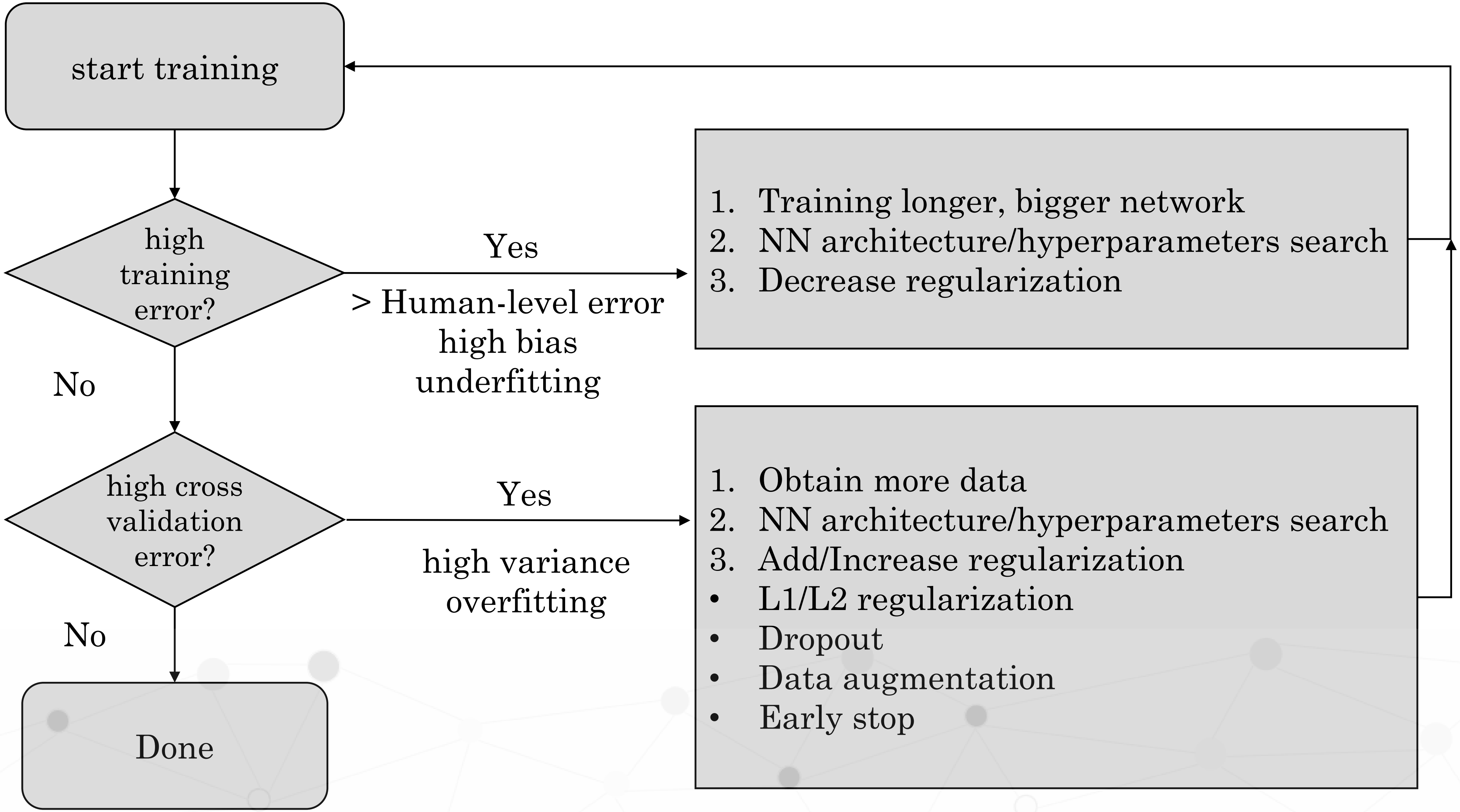
Overfitting

- When model performs well on the training data, but not the validation data (evidence of memorization)
- Ideally the accuracy and loss should be similar between both datasets



Setting Up Your ML Application

Basic “recipe” for machine learning



Regularization

Regularization

How does regularization prevent overfitting?

$$Y = b + \sum w_i x_i$$

λ: regularization parameter

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i \right) \right)^2 \quad \boxed{+ \lambda \sum_n (w_i)^2} \quad \boxed{\cancel{+ \lambda b^2}}$$

Why smooth functions are preferred?

If some noises corrupt input x_i when testing, a smoother function be less influenced

$$+w_i \Delta x_i \qquad +\Delta x_i$$

$$Y = b + \sum w_i x_i$$

How does regularization prevent overfitting?

$$\boldsymbol{L2\ regularization}:\ \|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

$$New\ loss\ function:\ L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2$$

$$Gradient:\ \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda \mathrm{w}$$

$$\text{update: } w^{t+1} \rightarrow w^t - \alpha \frac{\partial L'}{\partial w} = w^t - \alpha \left(\frac{\partial L}{\partial w} + \lambda \mathrm{w}^t \right)$$

$$= \underline{(1 - \alpha\lambda)} w^t - \alpha \frac{\partial L}{\partial w}$$

Weight Decay

How does regularization prevent overfitting?

$\boldsymbol{L1\ regularization}$: $\|\theta\|_1 = |w_1| + |w_2| + \dots$

$New\ loss\ function$: $L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_1$ $\qquad \frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda sgn(w)$

Update: $w^{t+1} \rightarrow w^t - \alpha \frac{\partial L'}{\partial w} = w^t - \alpha \left(\frac{\partial L}{\partial w} + \lambda sgn(\mathrm{w}^t) \right)$

$= w^t - \alpha \frac{\partial L}{\partial w} \underline{- \alpha\lambda sgn(\mathrm{w}^t)}$ always opposite

L2: $(1-\alpha\lambda)w^t - \alpha \frac{\partial L}{\partial w}$

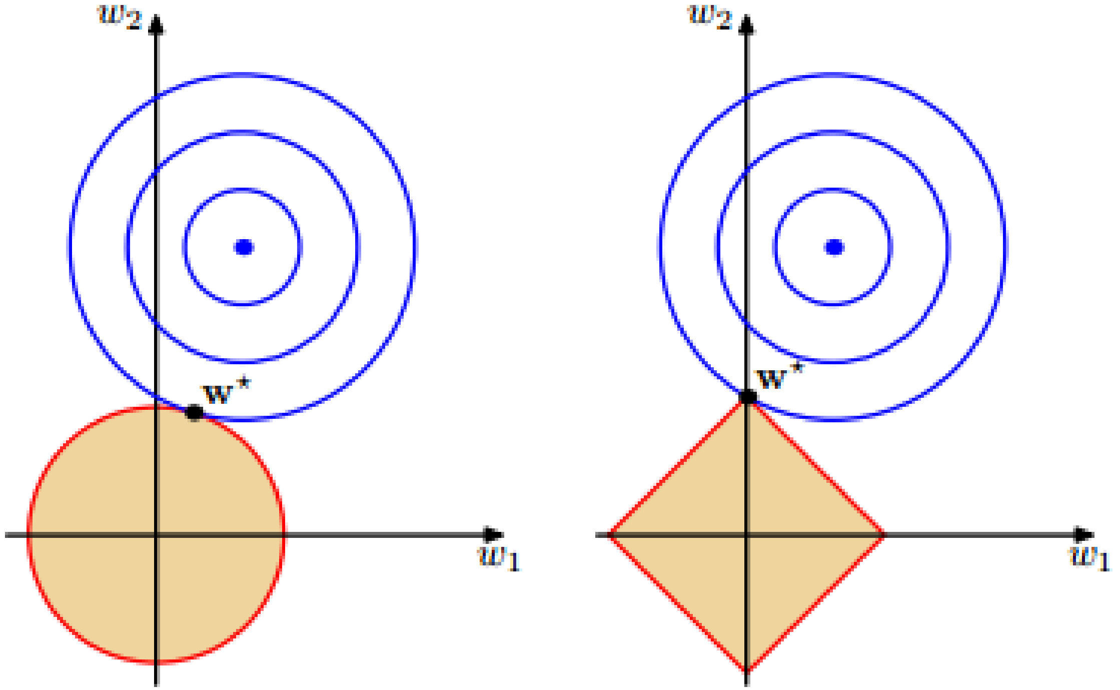
Regularization

How does regularization prevent overfitting?

146 **3. LINEAR MODELS FOR REGRESSION**

Figure 3.4 Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector $\mathbf{w}$ is denoted by $\mathbf{w}^{\star}$. The lasso gives a sparse solution in which $w_1^{\star} = 0$.

L2: $(1-\alpha\lambda)w^t - \alpha\frac{\partial L}{\partial w}$



L2 regularization

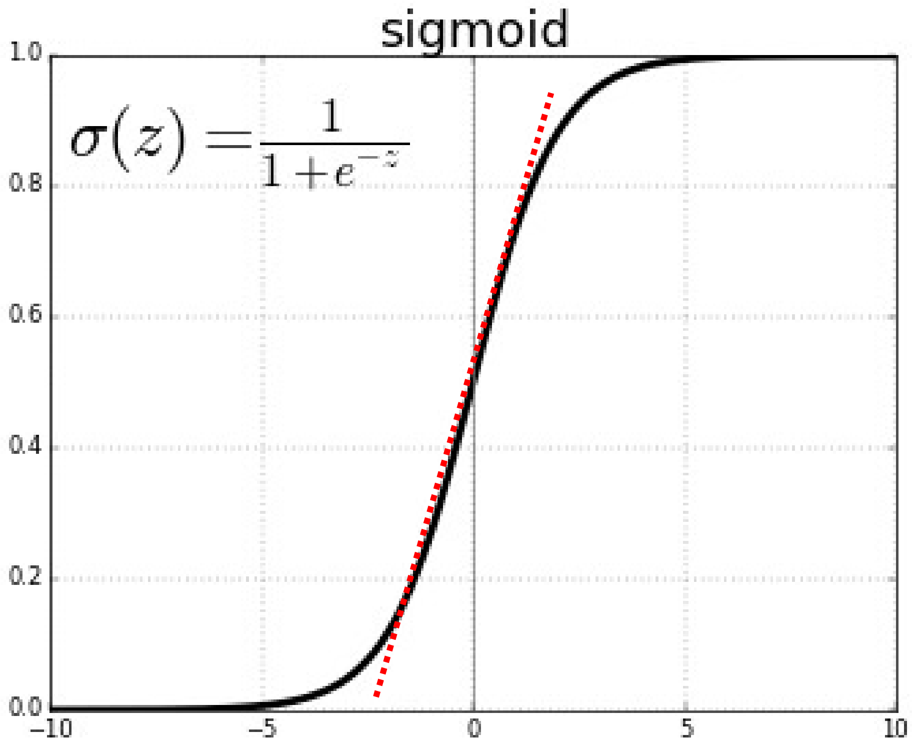
L1 regularization

$L2\ regularization$: $\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \ldots$

$L1\ regularization$: $\|\theta\|_1 = |w_1| + |w_2| + \ldots$

Regularizing Your Neural Network

How does regularization prevent overfitting?



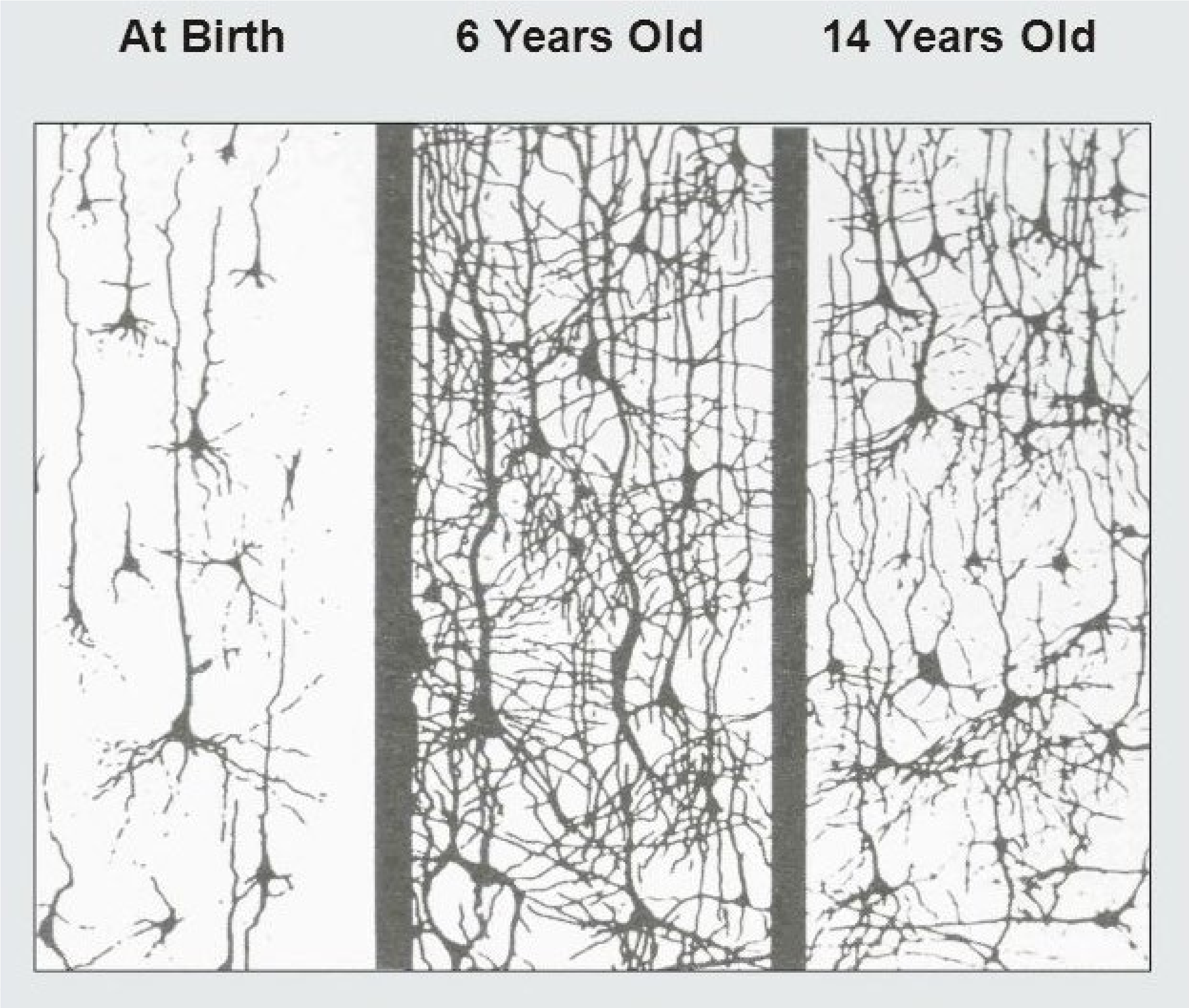
$\lambda \uparrow$ $\quad W^{[l]} \downarrow$ $\quad z^{[1](i)} = W^{[l]}a^{[l-1]} + b^{[l]}$

Every layer becomes ~linear

Dropout Regularization

Weight Decay

Synaptic density



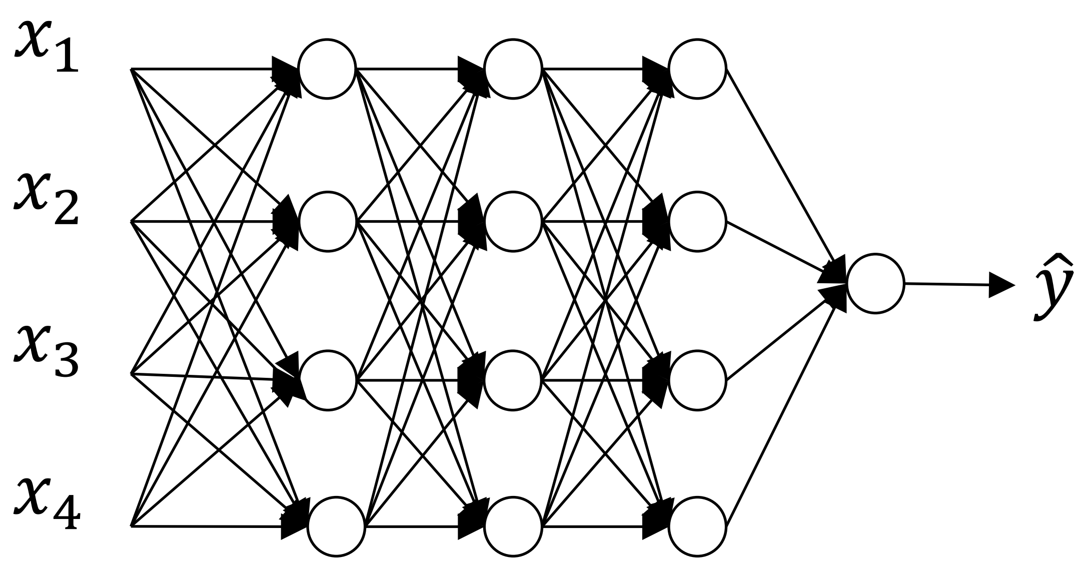
Our brain prunes out the useless links between neurons

Weight Decay is doing the same thing to Machines' brain to improve the performance

Rethinking the Brain, Families and Work Institute, Rima Shore, 1997.

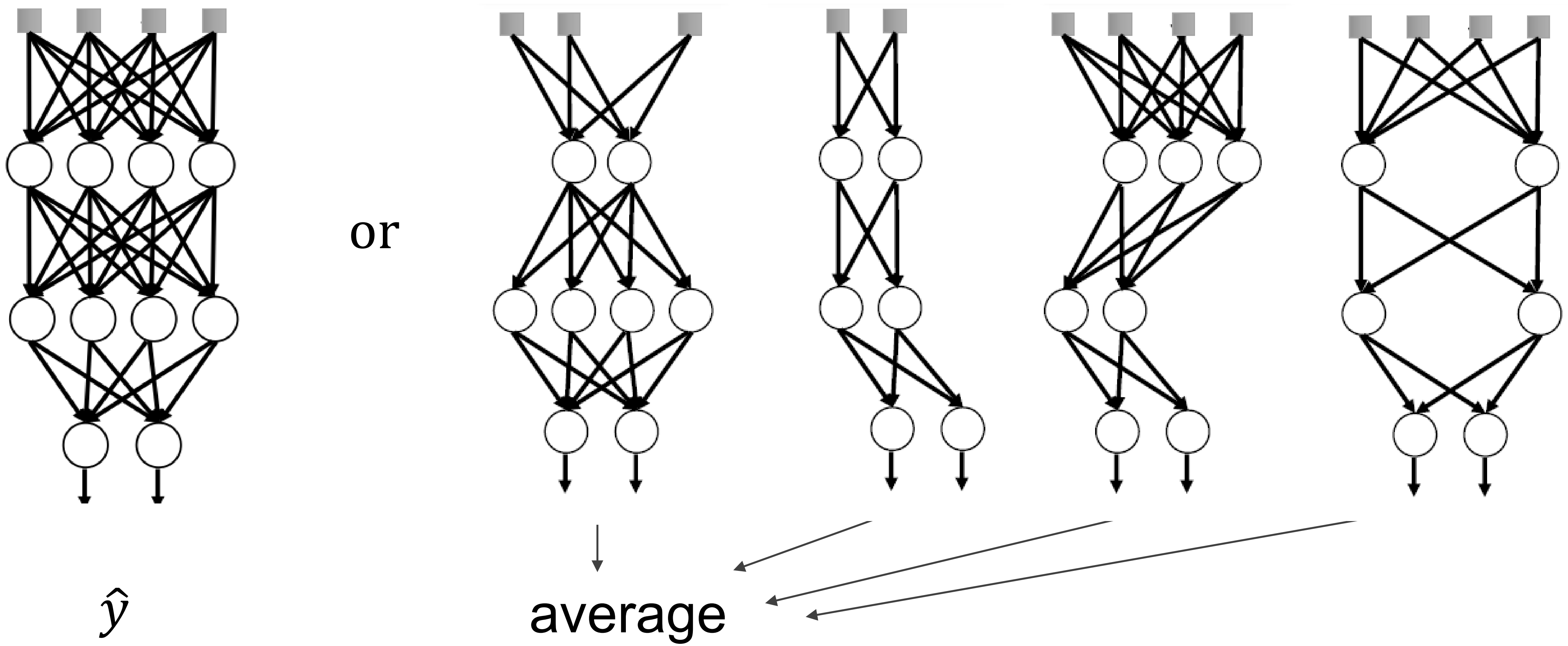
Dropout regularization

A simple way to prevent neural network from overfitting
Randomly set some neurons to zero in the forward pass



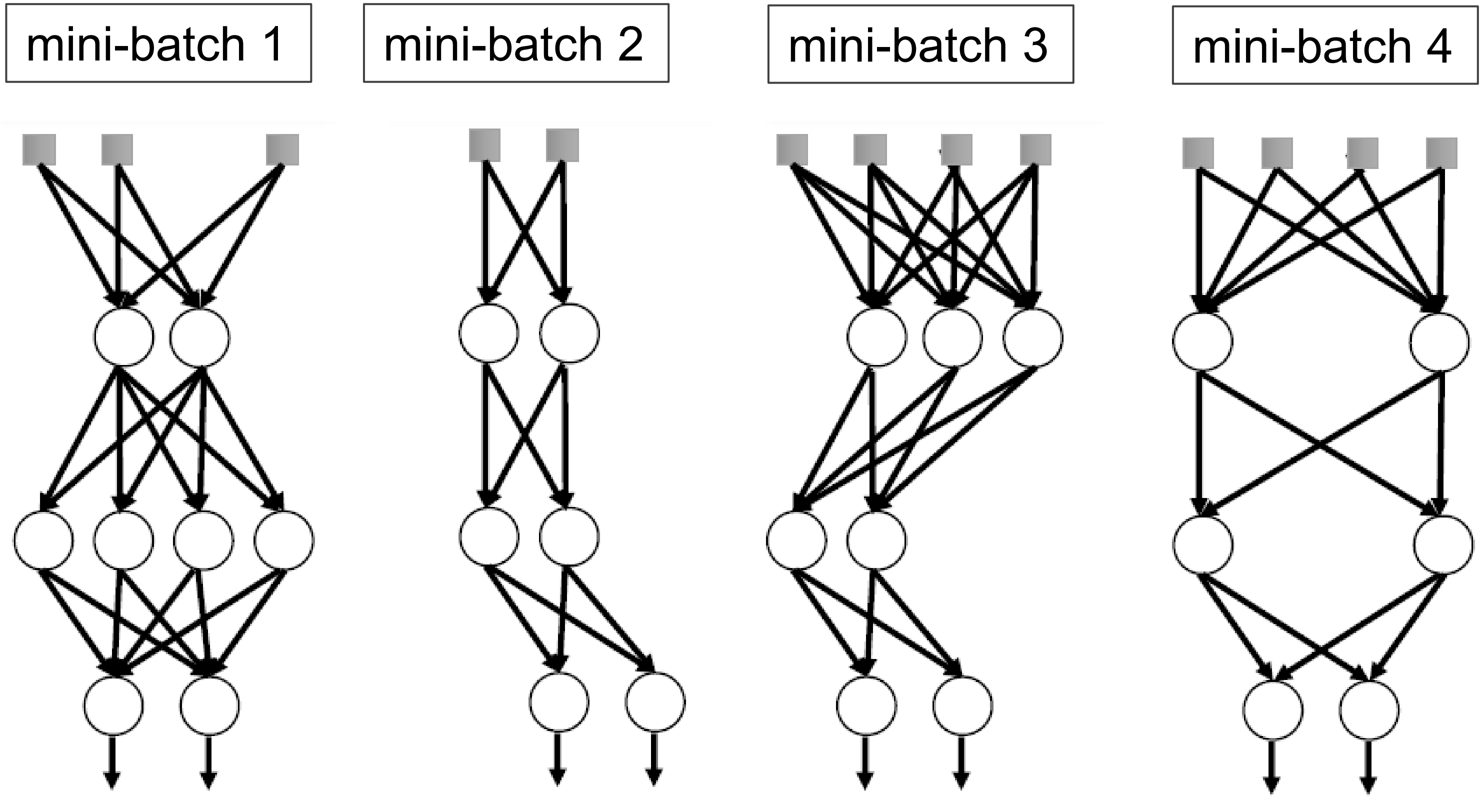
keep_prob = 0.5 0.5 0.5

Making predictions at test time



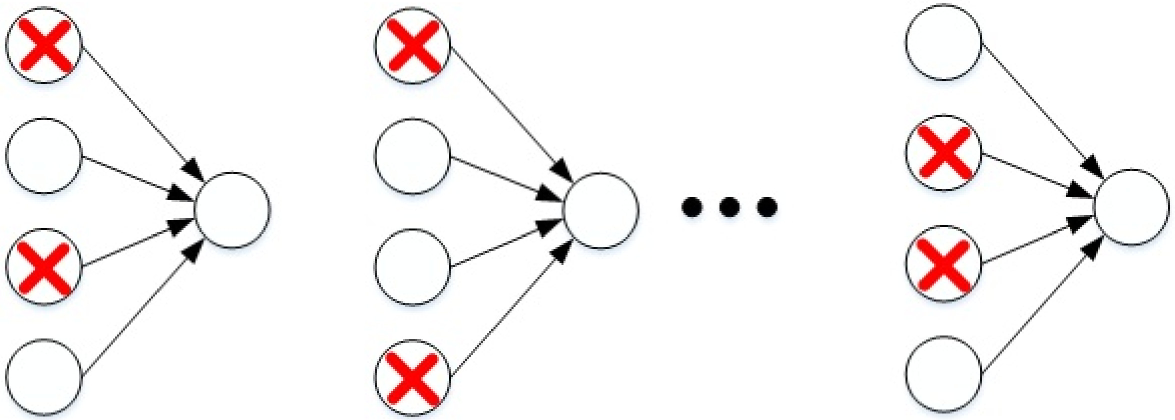
Note: No dropout at test time

Dropout is a kind of ensemble



Why does drop-out work?

Intuition: Can’t rely on any one feature, so have to spread out weights.



Data Augmentation

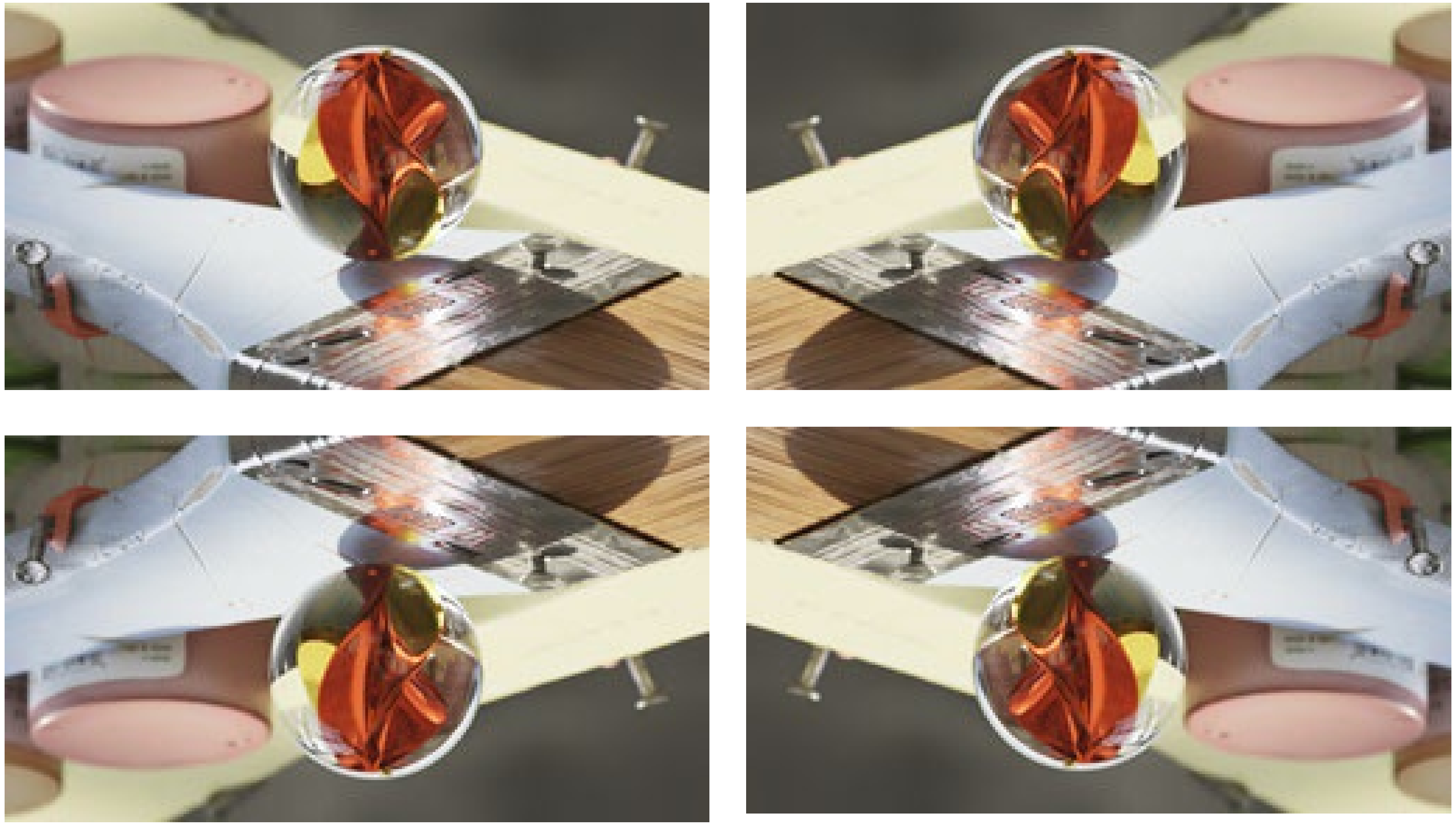
Other regularization methods

Data augmentation

DATA AUGMENTATION

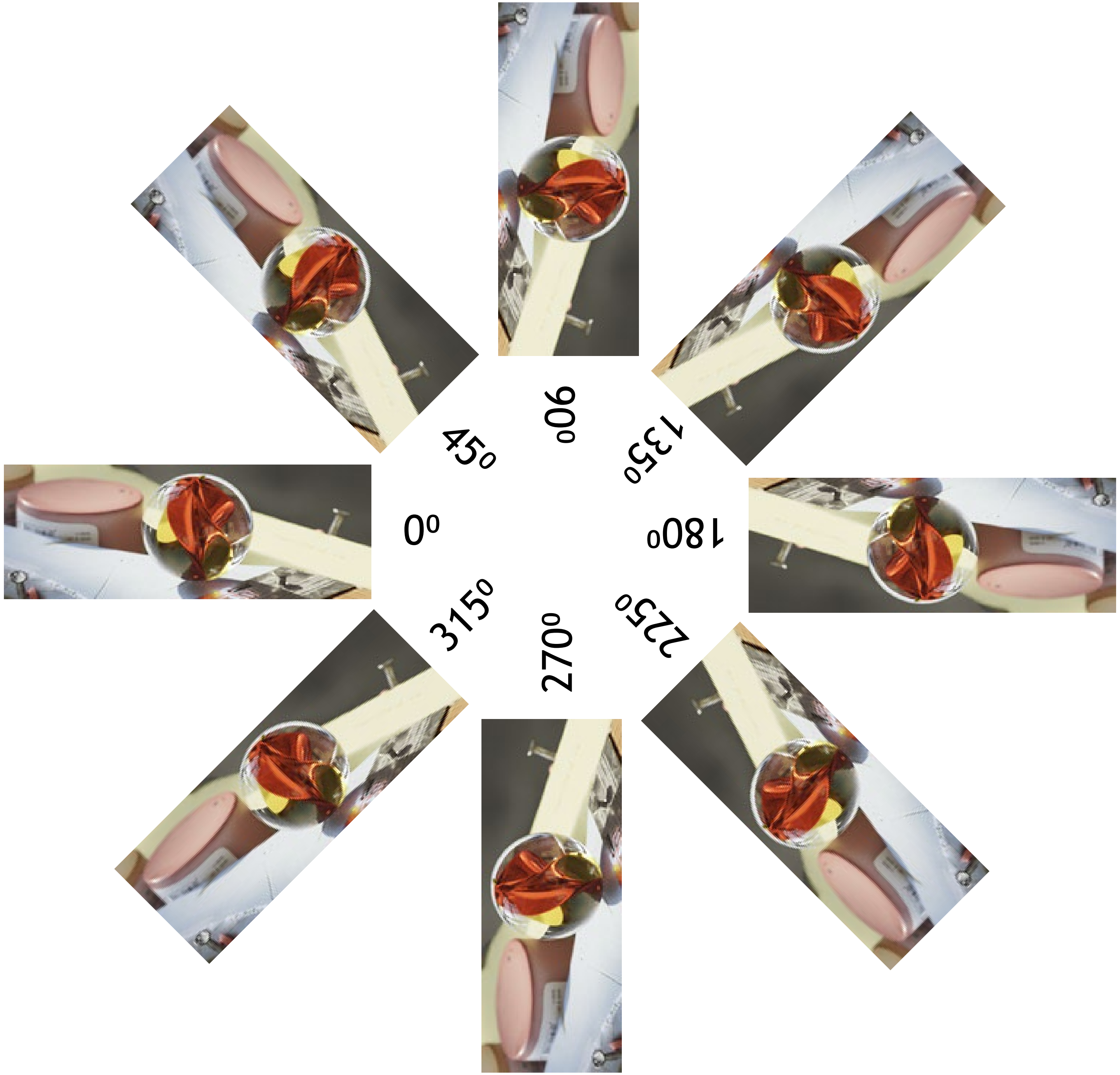
IMAGE FLIPPING

Horizontal Flip

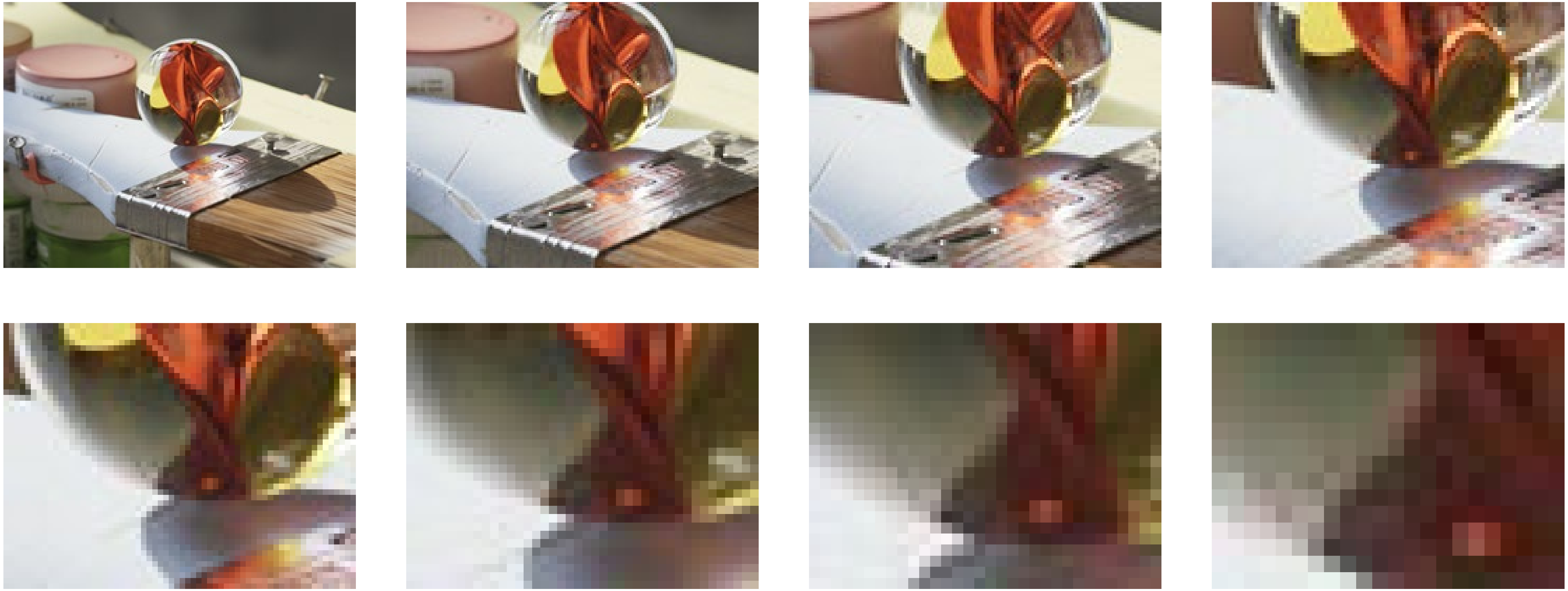


Vertical Flip

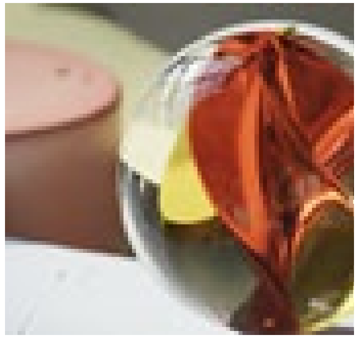
ROTATION

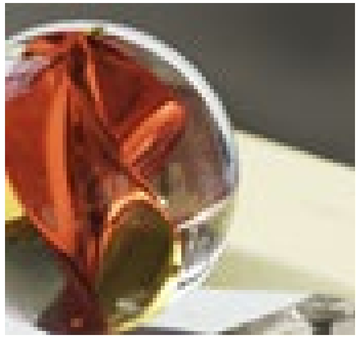


ZOOMING

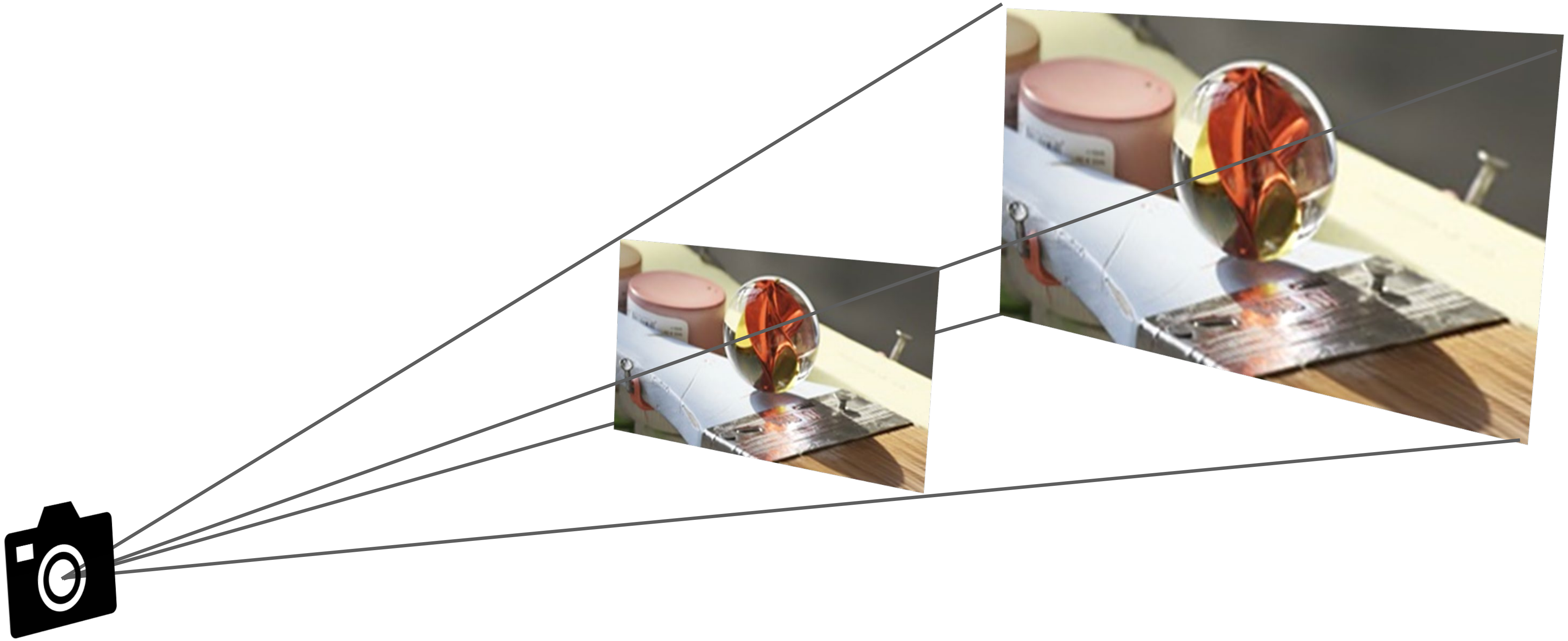


WIDTH AND HEIGHT SHIFTING



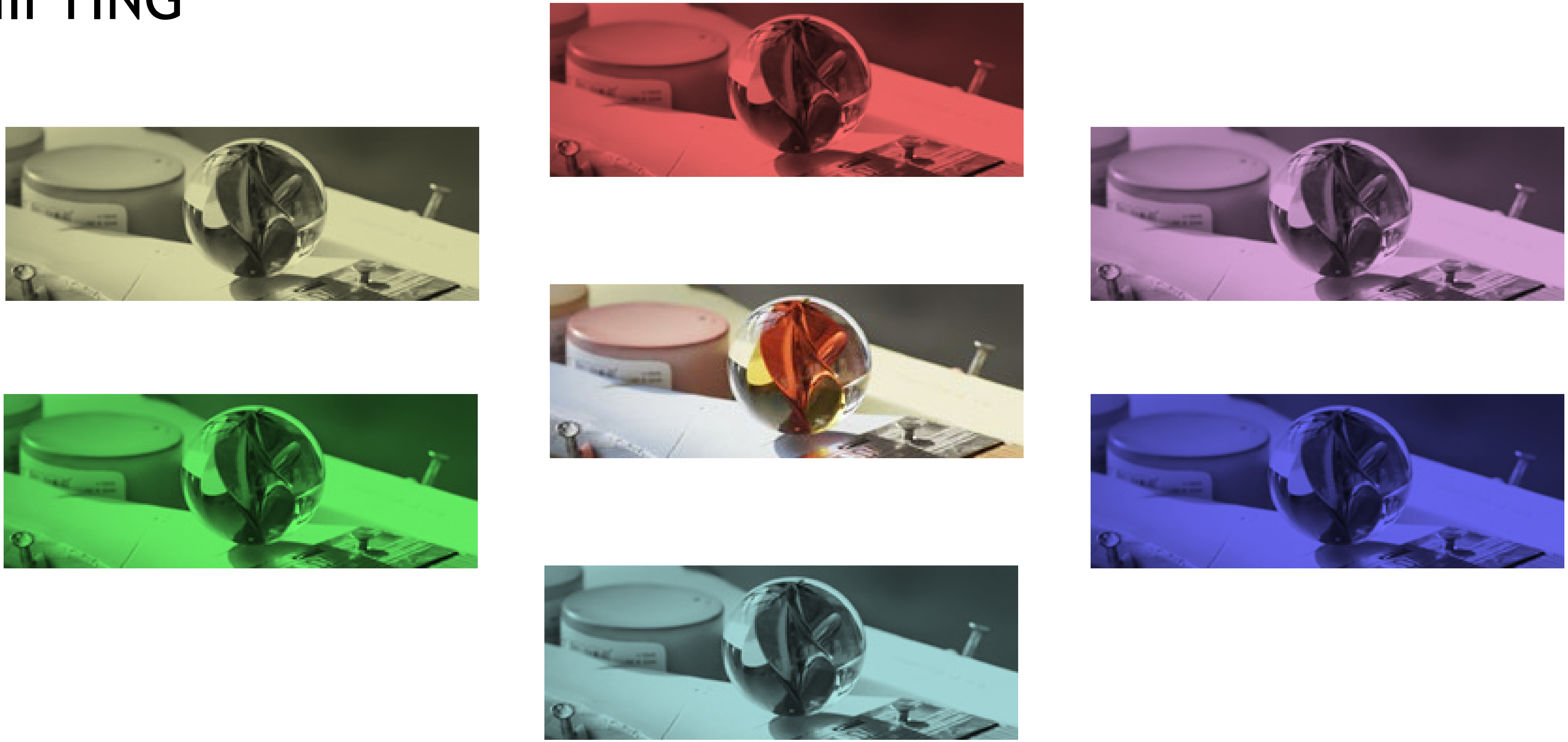


HOMOGRAPHY



BRIGHTNESS

CHANNEL SHIFTING



Other Regularization Methods

Data augmentation

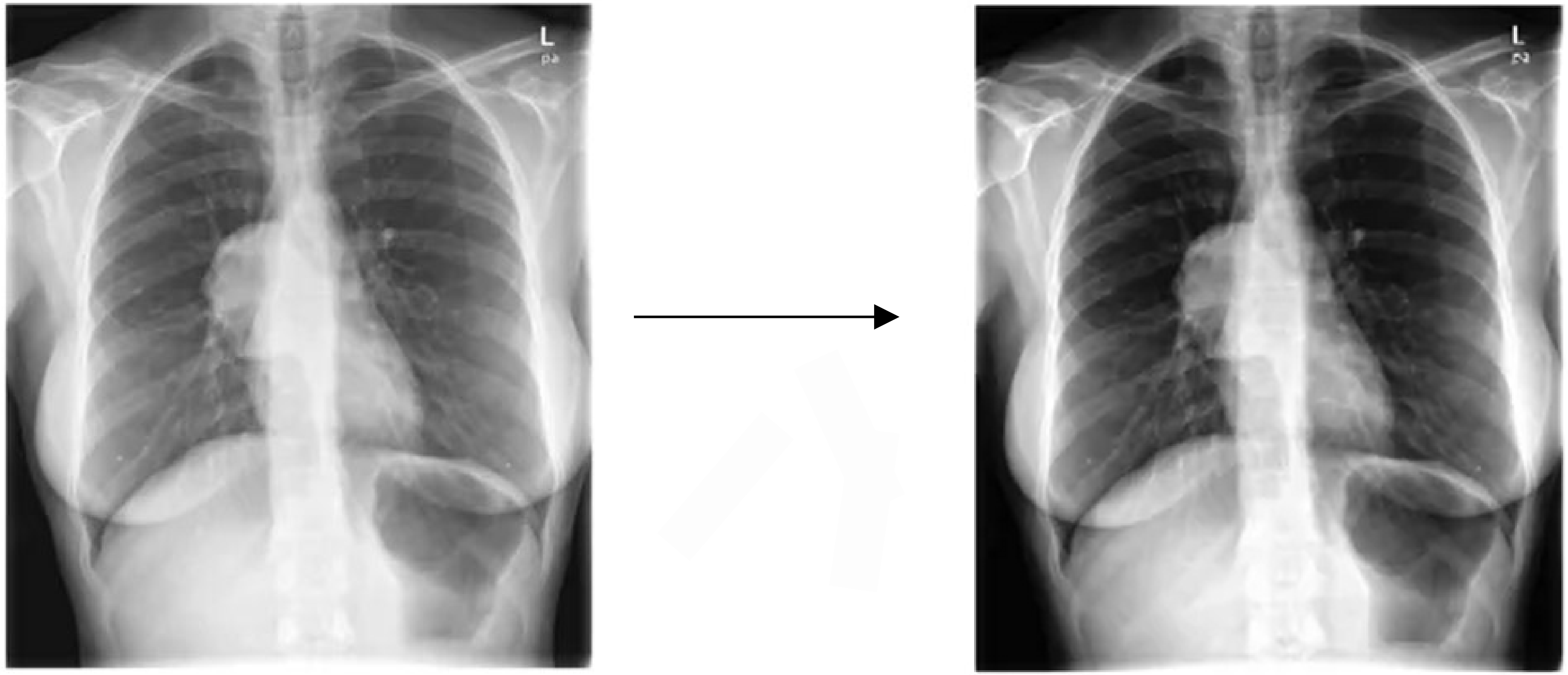
Rotation

Translation

Contrast

Data Augmentation

Do augmentation reflect variation in real world?



Data augmentation

Do augmentation keep the label the same?

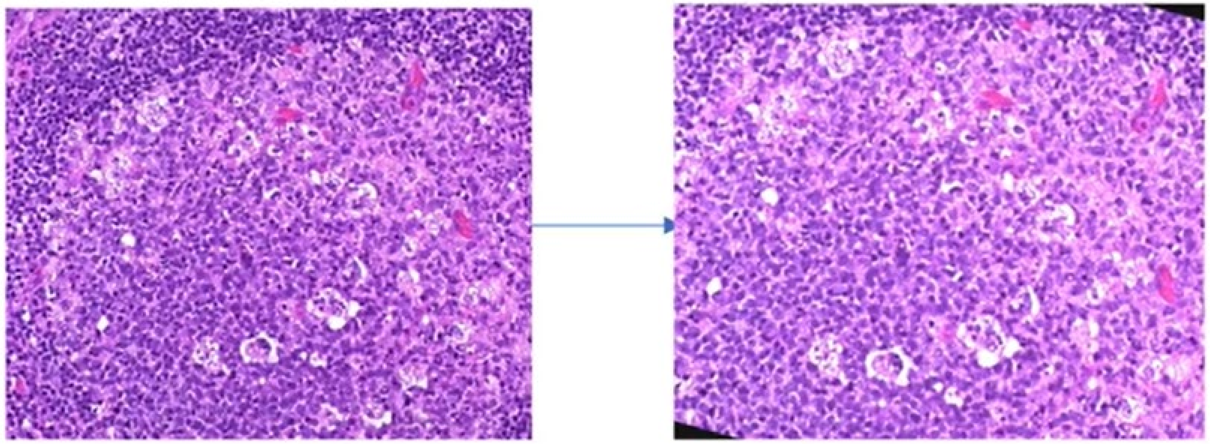
Normal

Normal
Or
Dextrocardia?

Data augmentation

Other augmentation methods

Rotate + Flip

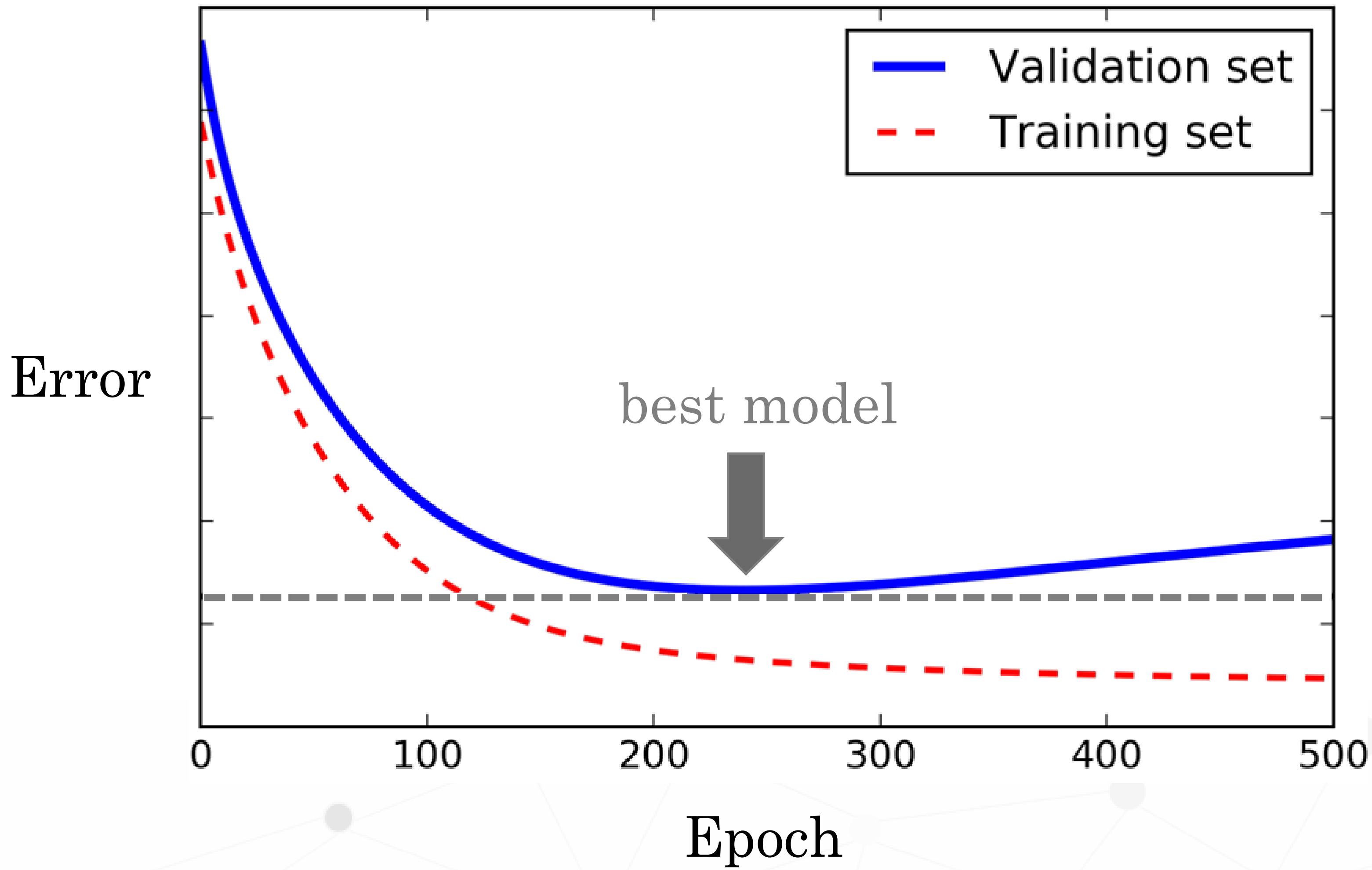


Rotate + Crop + Color Noise

Early Stopping

Other Regularization Methods

Early stopping



- Optimize cost function J(w,b)
- Reduce bias
  - Gradient descent
  - RMSprop, Adam, ……

- Reduce overfitting
- Reduce variance
  - Regularization
  - Getting more data

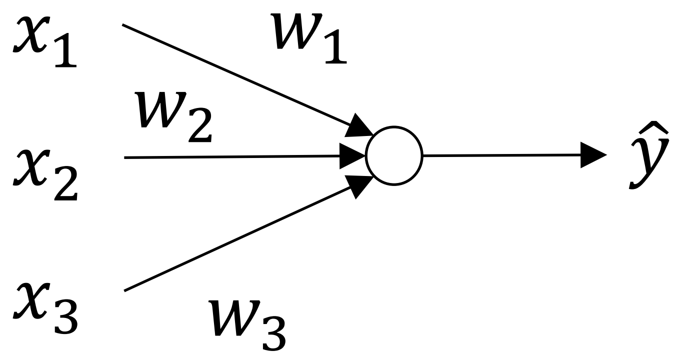
- Orthogonalization

Model Checkpoint

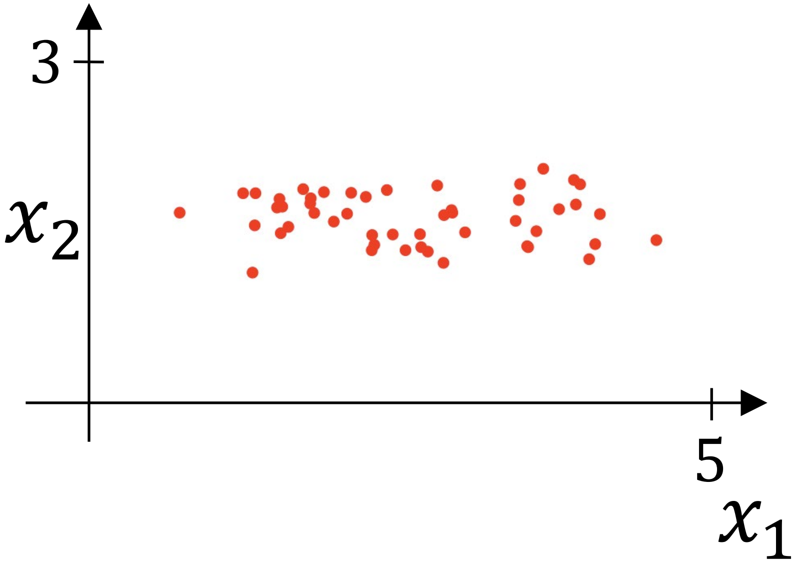
Setting up your optimization problem

Normalizing Inputs

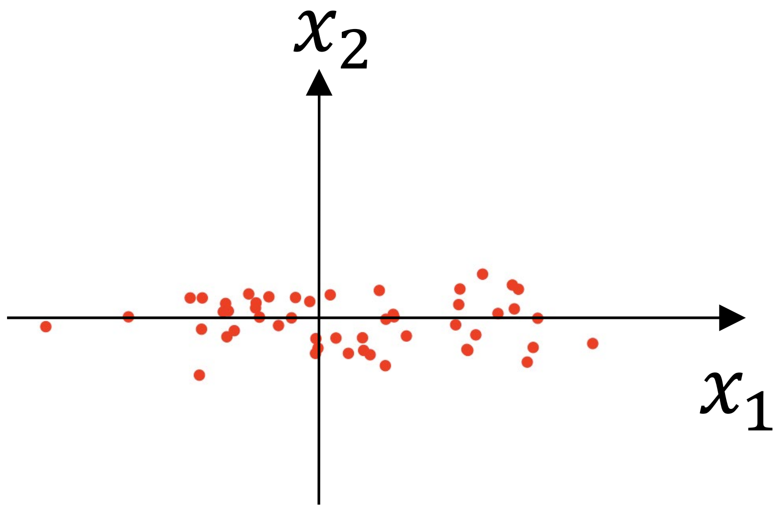
Normalizing Inputs



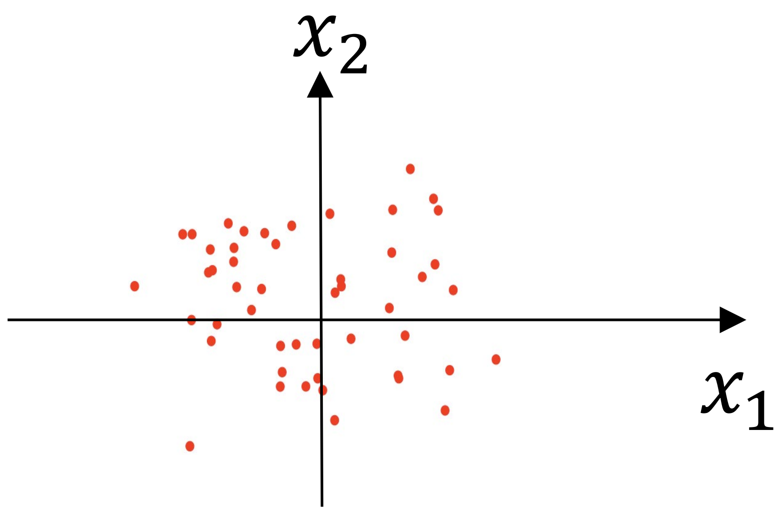
Normalizing training sets to speed up training



X



$X := X - \mu$



$$X := \frac{X - \mu}{\sigma^2}$$

$\mu = 0$
$\sigma^2 = 1$

$$\boxed{\mu = \frac{1}{m}\sum_{i=1}^{m} X^{(i)}}$$

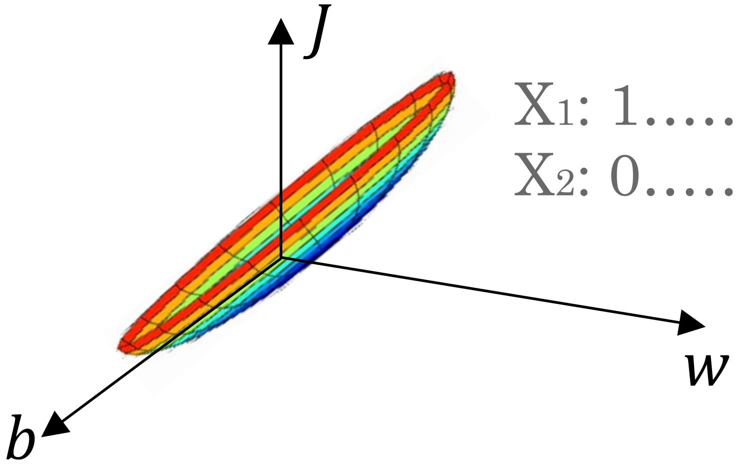
$$\boxed{\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (X^{(i)})^2}$$

Normalizing Inputs

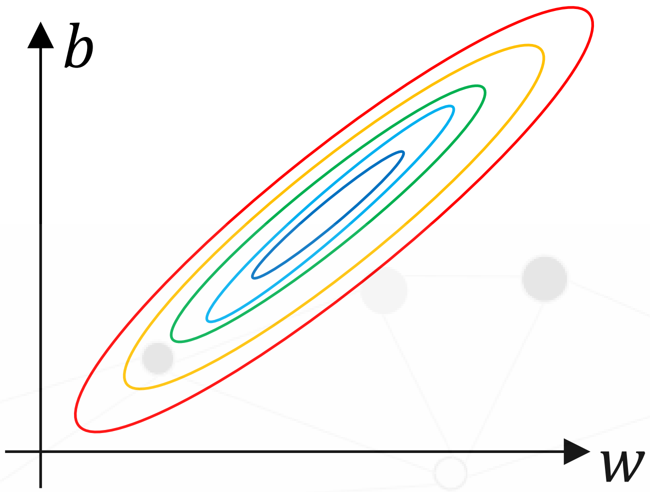
Why normalize inputs?

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\,(\hat{y}^{(i)}, y^{(i)})$$

Unnormalized:

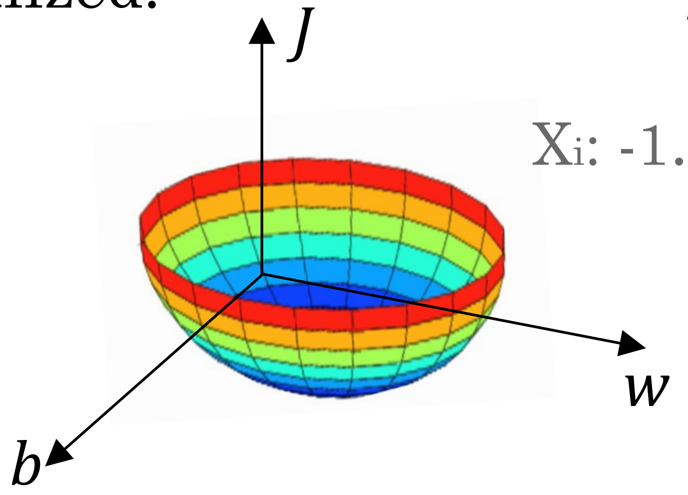


X_1: 1……1000
X_2: 0……0.01

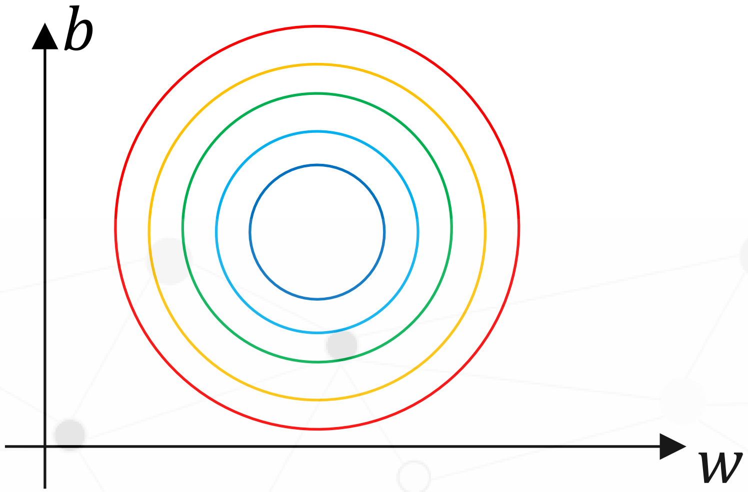


X_1: 0……1
X_2: -1……1
X_3: 1……2

Normalized:



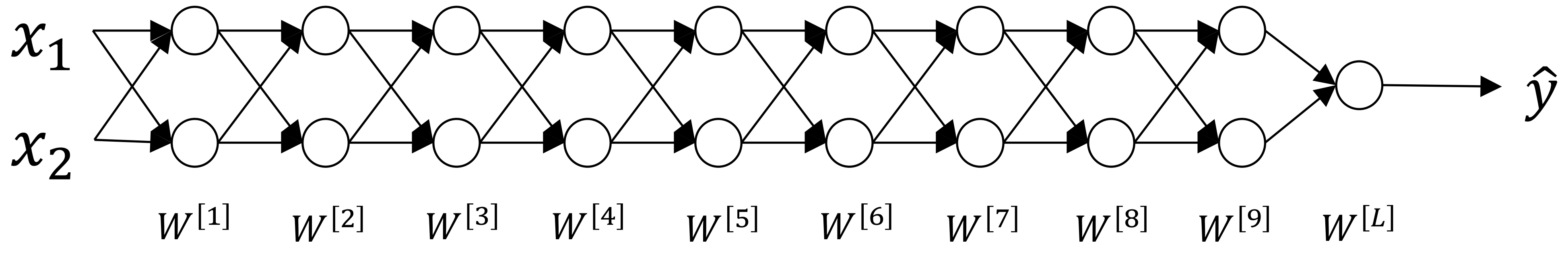
X_i: -1……1



Weight Initialization

Setting Up Your Optimization Problem

Vanishing/exploding gradients



$$g(Z) = Z$$

$$\hat{Y} = W^{[L]} W^{[L-1]} W^{[L-2]} \dots W^{[3]} W^{[2]} W^{[1]} X$$

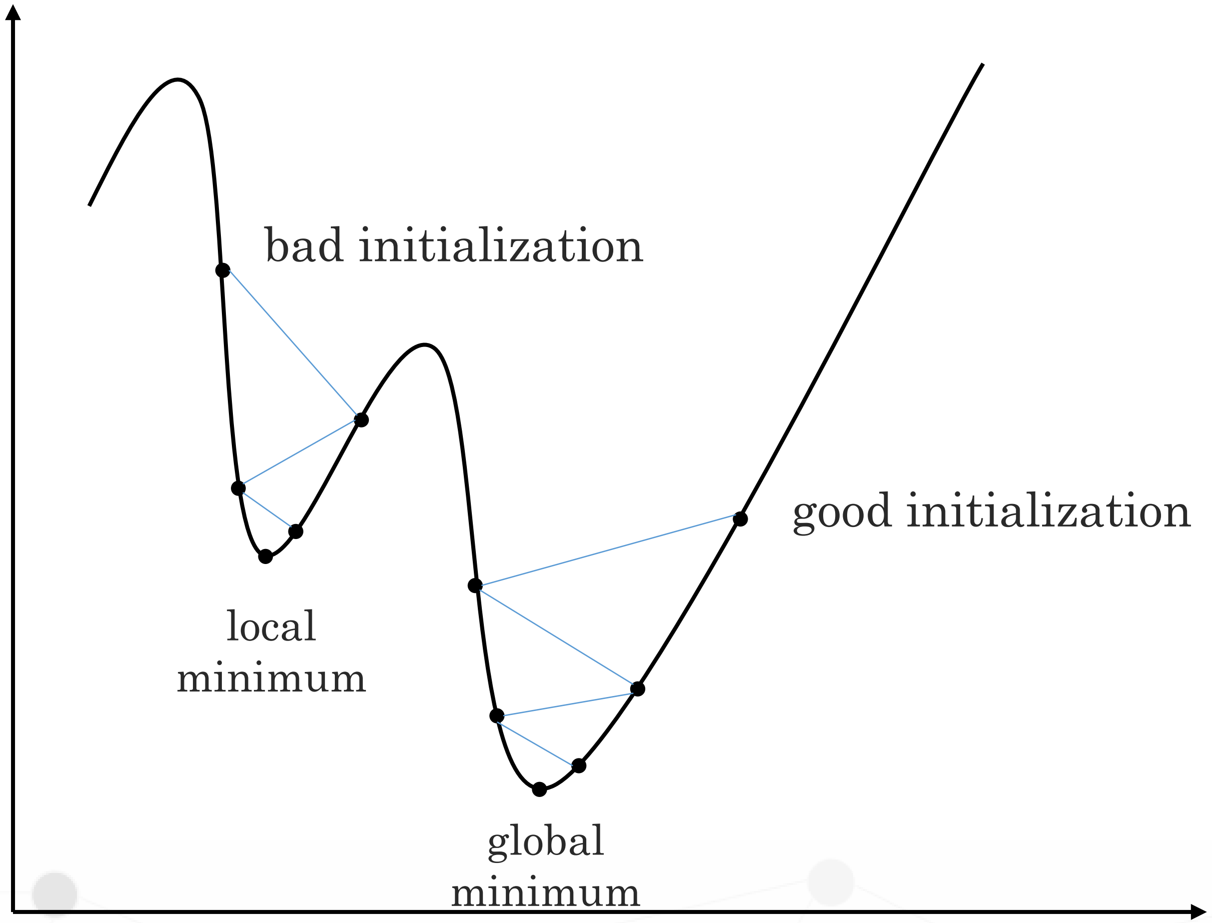
$$w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} > I \qquad w^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} < I$$

$$\hat{y} = W^{[L]} 1.5^{L-1} x \qquad \hat{y} = W^{[L]} 0.5^{L-1} x$$

1.5^L exploding gradients $\qquad$ 0.5^L vanishing gradients

https://www.deeplearning.ai/ai-notes/initialization/

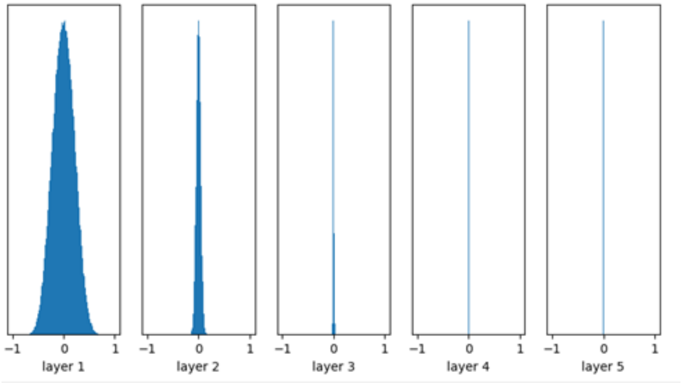
Initialization



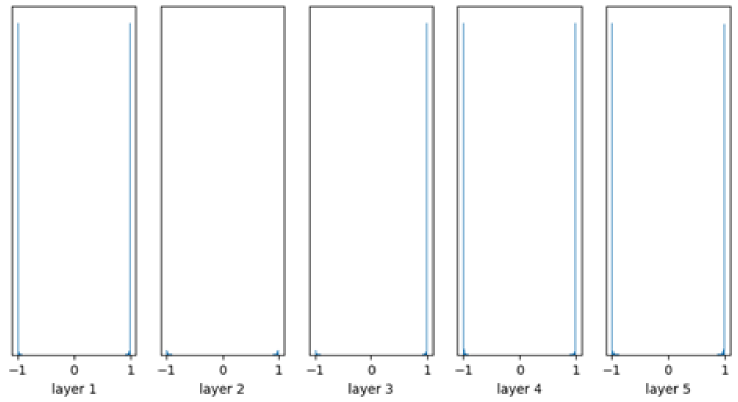
Setting Up Your Optimization Problem

Weight Initialization (6-layer MLP)

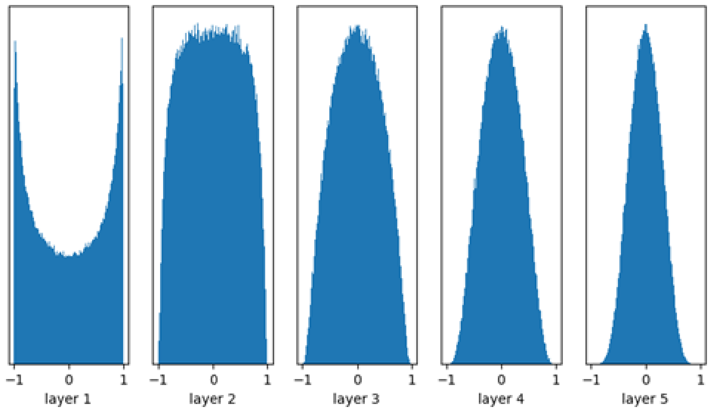
Exp1. (μ = 0， std = 0.01)



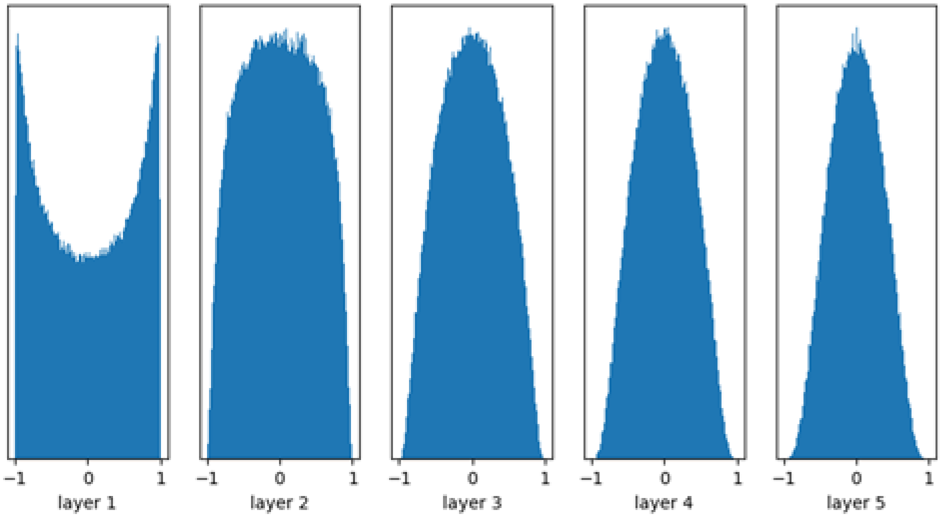
Exp2. (μ = 0， std = 1)



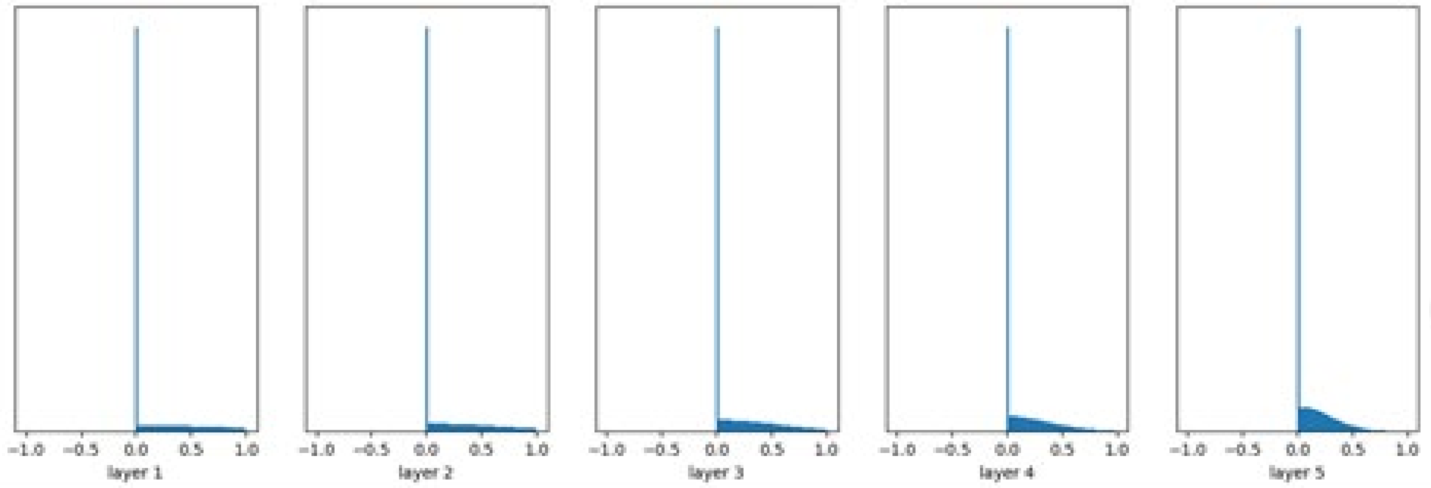
Exp3. (μ = 0， std = 0.05)



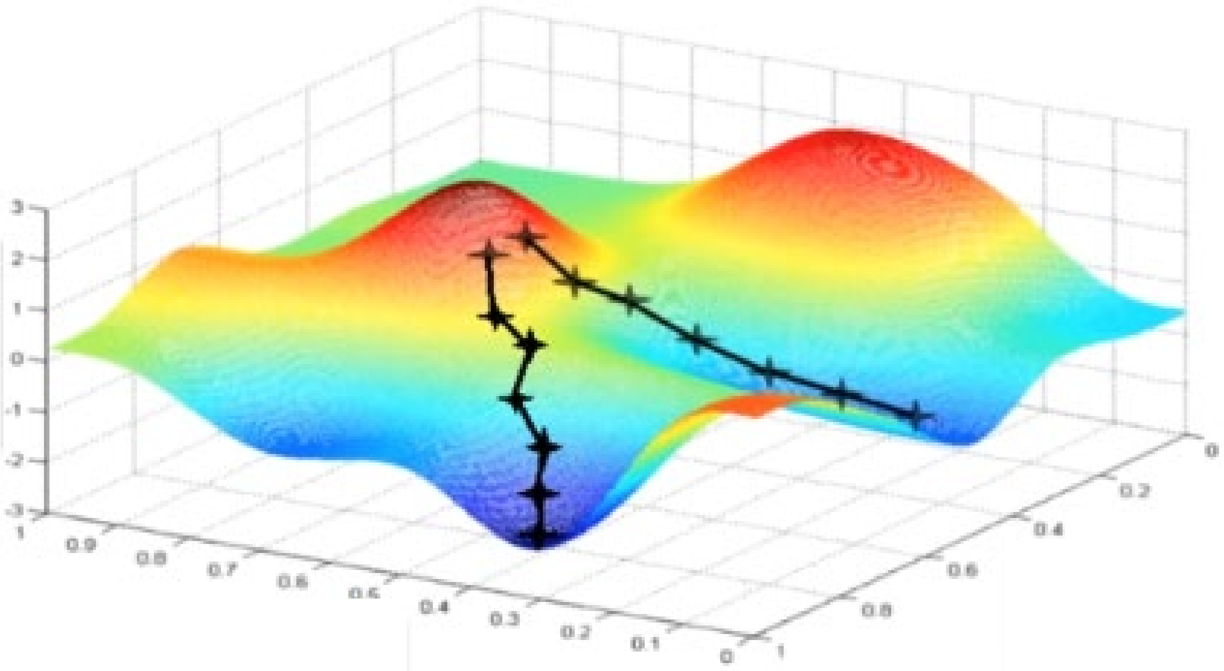
Xavier initialization + tanh



Xavier initialization + ReLU



Weight Initialization



Unlike with convex optimization, it matters where you start!

- Initialize with zero: get stuck at the big saddle point.
- Initialize with constant values: difficult to break symmetries.
- Initialize with very large values: off on the great plateaus. Small gradients, slow convergence.

Zero initialization

Random initialization

He initialization (ReLU)

Xavier initialization (tanh)

$$W^{[l]} \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}})$$

$$b^{[l]} = 0$$

Use **small random values:**

- E.g. zero mean Gaussian noise with constant variance

He et al., Delving Deep into Rectifiers

Next Lecture

- Optimization algorithms (Course 2 Week 2)
- Hyperparameter Tuning (Course 2 Week 3)
- ML Strategy (1) (Course 3 Week 1)
- ML Strategy (2) (Course 3 Week 2)

Next:

Lab Practice

Hyperparameter Tuning