CS7150 Deep Learning

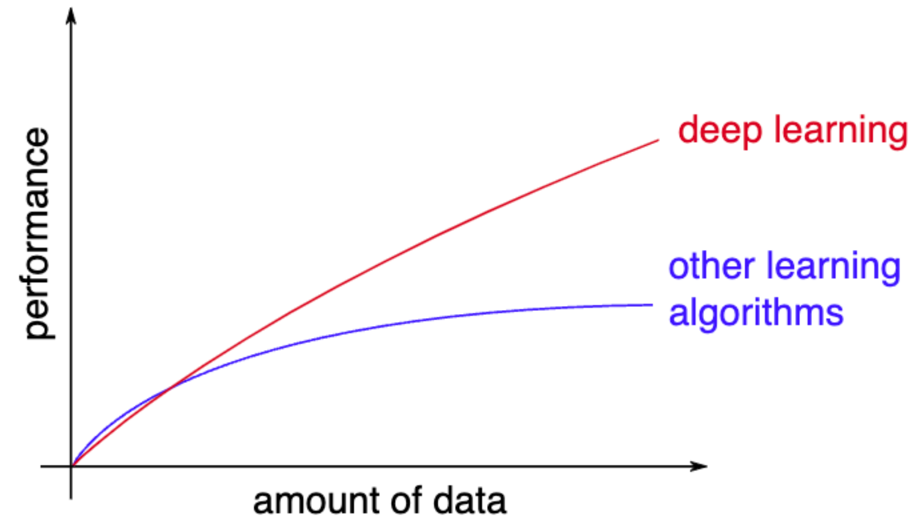
Jiaji Huang

https://jiaji-huang.github.io

01/20/2024

Recap of Last Lecture

- What is DL and why is it useful
- Cool Applications
  - Images, speech, text, robotics, …
- PyTorch
  - Auto-grad
- Linear Algebra, probabilities
  - Linear Least Squares
  - SVD
  - Gradient, Chain rule
  - Gaussian Distribution and its mixture
  - Entropy, Cross Entropy



Agenda

- Machine Learning Paradigms
- Non-Parametric v.s. Parametric Models
- Linear Classifier
- Multi-layer Perceptron (MLP)

Supervised Learning

- Training: learns a model $f: \boldsymbol{x} \longmapsto y$, from data like $\{(\boldsymbol{x}_i, y_i)\}$
- Testing: Use the model to predict y given $\boldsymbol{x}$, i.e. $\hat{y} = f(\boldsymbol{x})$
- Examples:
  - Regression: numeric target

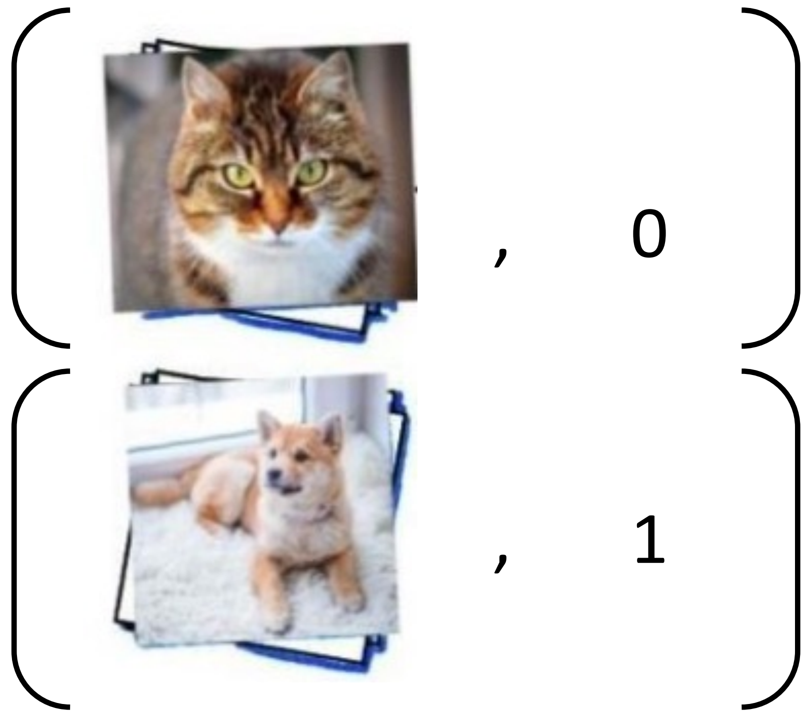
```
data = {
    {4 yr, "Female"} → 3.3 kg,
    {6 yr, "Male"} → 4.5 kg,
    {5 yr 3 mo, "Male"} → 5.1 kg,
    {1 yr 3 mo, "Female"} → 1.7 kg
  };
```

{5 yr, "Female"} →?

Example from this page

Supervised Learning

- Classification: categorical target

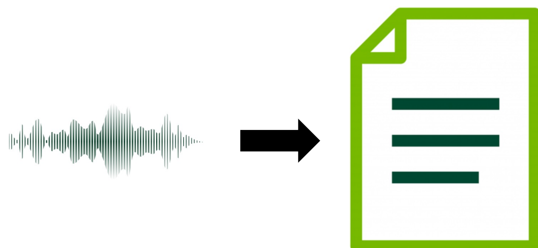


?

Supervised Learning

- Classification with structured target

Speech recognition



Translation

We are attending a deep learning class

我们正在参加深度学习课程

- Classification + regression

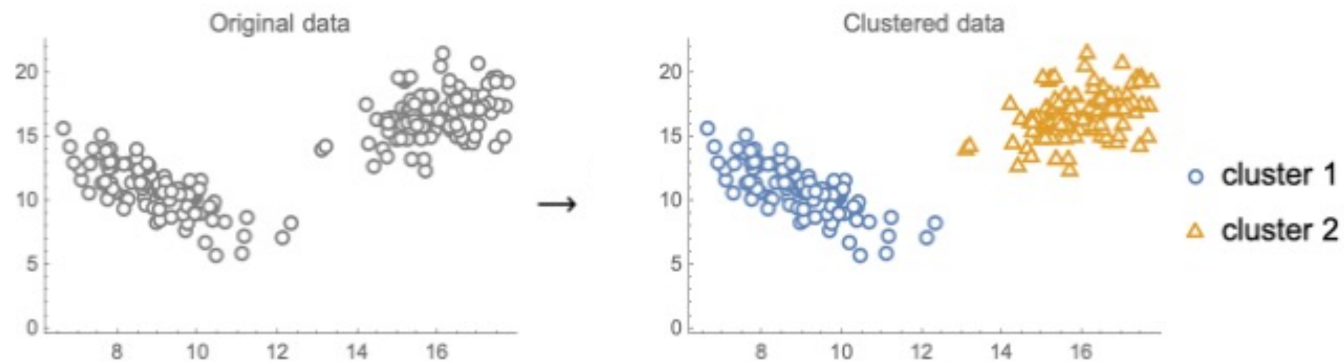
Object detection

→

person → {Rectangle[{0.32, 0.13}, {2.6, 7.8}], ...}
bus → {Rectangle[{7.8, 5.1}, {10, 6.6}], ...}
stoplight → {Rectangle[{1.9, 7.2}, {2.5, 8.9}]}
automobile → {Rectangle[{6.2, 4.5}, {7.2, 5.9}]}
motorcycle → {Rectangle[{9., 1.3}, {9.9, 4.7}]}

Unsupervised Learning

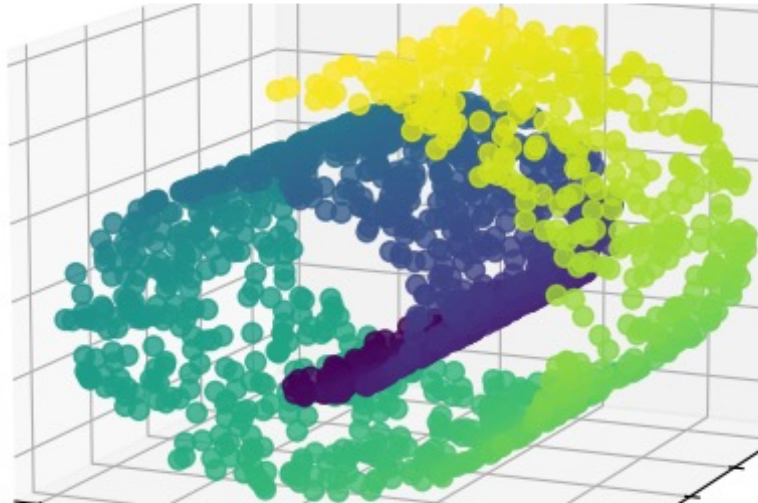
- Given input $\boldsymbol{x}$, learn something about $p_{data}(\boldsymbol{x})$
- Clustering (recap last lecture: infer the latent variable via EM)

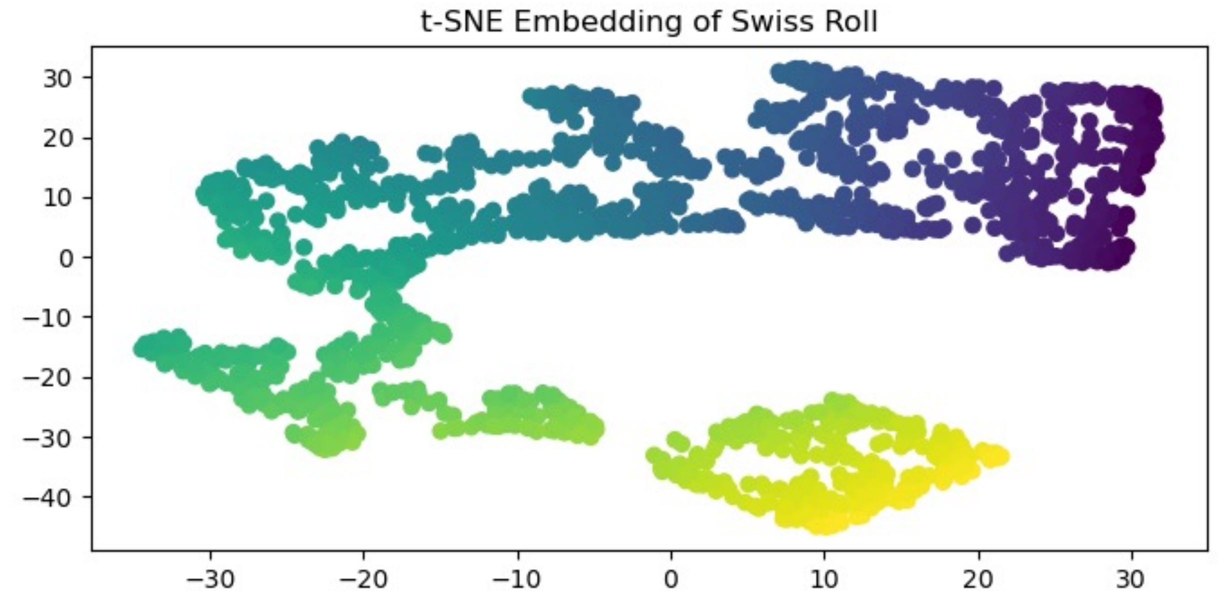


- Generation: sample from $p_{data}(\boldsymbol{x})$

Unsupervised Learning

- Dimension Reduction
  - PCA
  - Non-linear dimension reduction





Example from [sklearn page]

Reinforcement Learning

- Learning by interacting with environment, and
- Maximizing reward

Paradigms can be combined

- Semi-supervised learning
  - Input: $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ and $\{\boldsymbol{x}_j\}$
  - Output: $\boldsymbol{y}_j$'s
  - Jointly modeling $\{\boldsymbol{x}_i\} + \{\boldsymbol{x}_j\}$

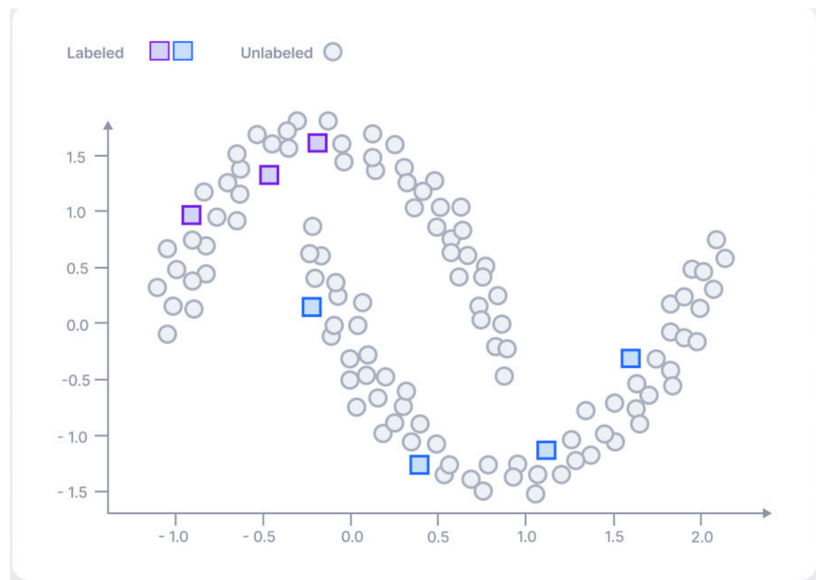
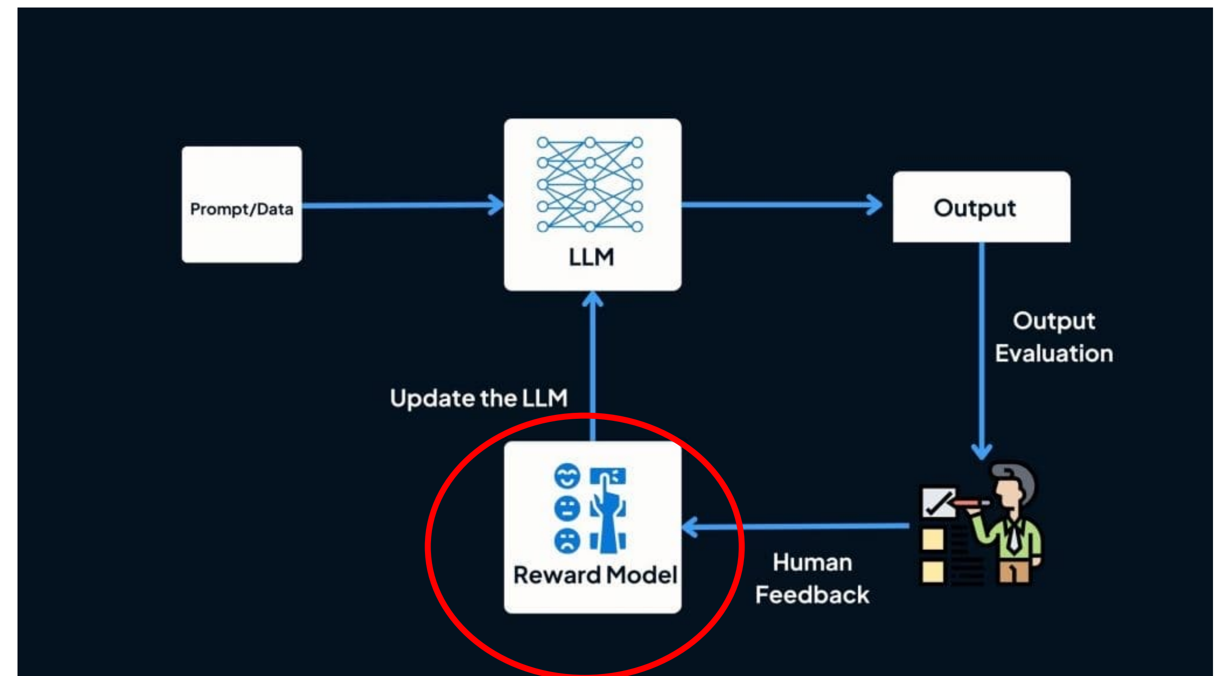


Illustration from here and here

- Reinforcement Learning with Human Feedback (RLHF)



Agenda

- Machine Learning Paradigms
- Non-Parametric v.s. Parametric Models
- Linear Classifier
- Multi-layer Perceptron (MLP)

Let's focus on Supervised Learning in the following

- Training: learns a model $f: \boldsymbol{x} \longmapsto \boldsymbol{y}$, from data like $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$
- Testing: Use the model to predict $\boldsymbol{y}$ given $\boldsymbol{x}$, i.e. $\hat{y} = f(\boldsymbol{x})$

Non-parametric Model

- Doesn't assume a specific form for $f(\boldsymbol{x})$
- Example: Nearest Neighbor Classifier
  - Stores all training data
  - Take k-nearest neighbors
  - Vote on the label by neighbors' labels
  - Question:
    - How to decide k?
    - What distance to use?

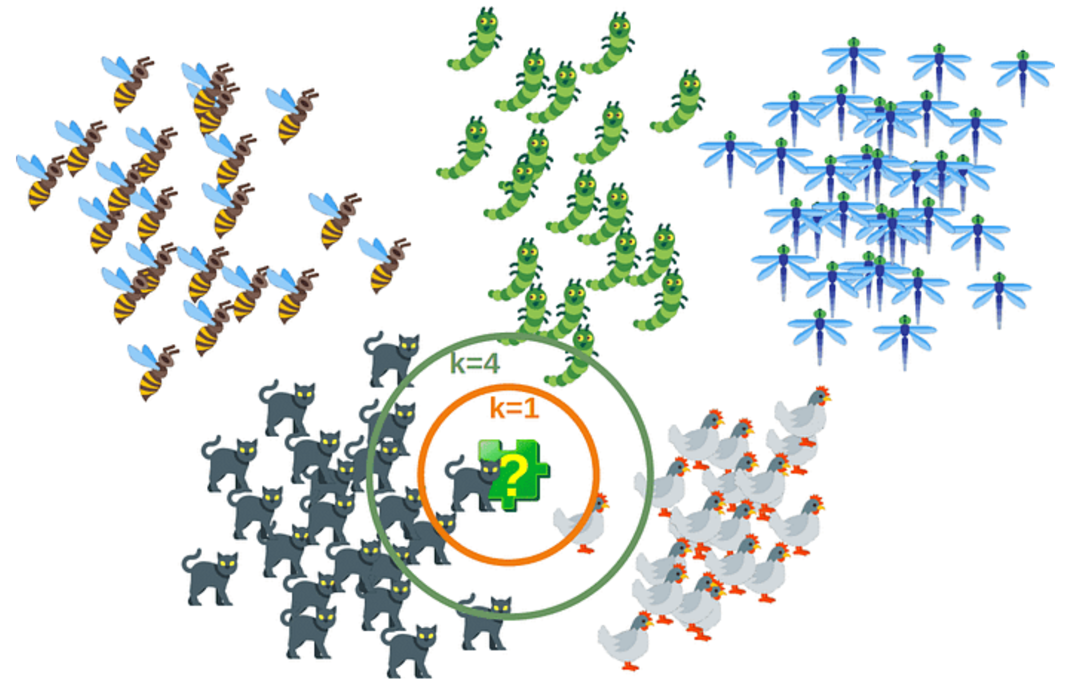


Illustration from [here]()

Hyper-parameters

- The k and choice of distance
- Dangerous idea:
  - Try a bunch, and check which works best for testing
  - As testing data couldn't be access at training
- If we have a validation set, use that
- If not, N-fold cross validation

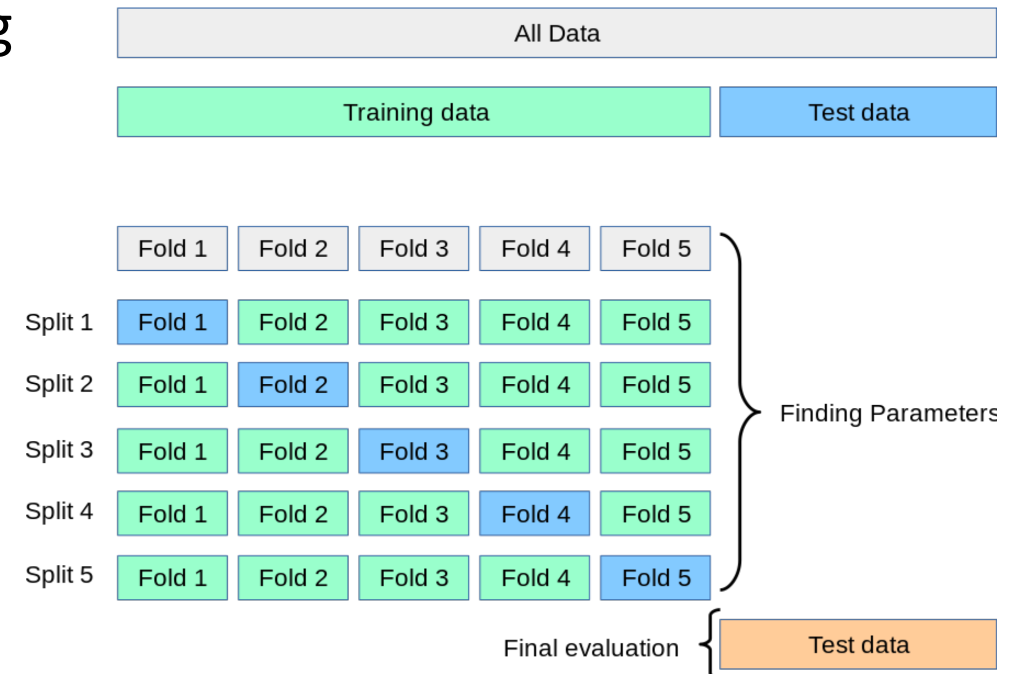


Illustration from sklearn page

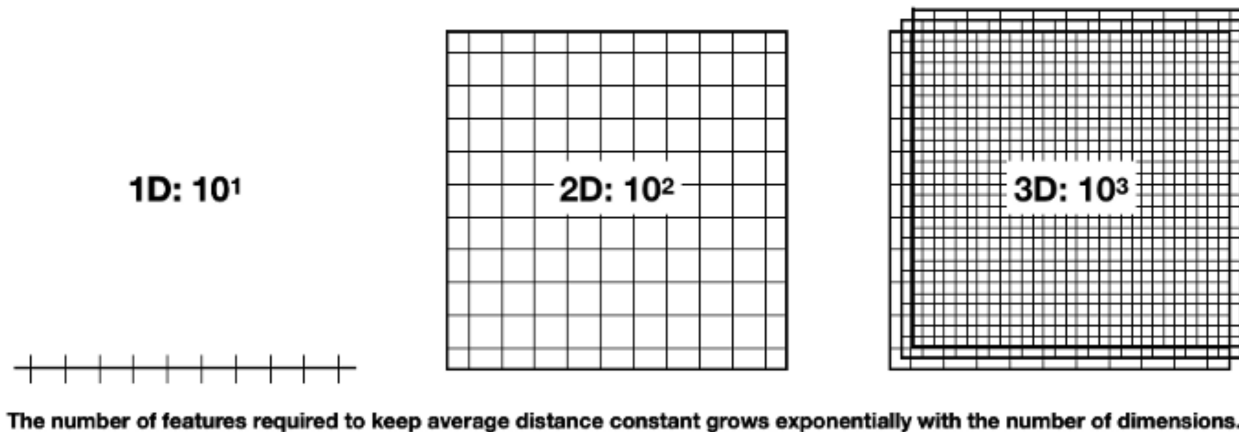
KNN is only for toy problems though

- Consider 128x128 Image classification
  - Flattening the 2D images leads to 16,384 dimensional vector!
- Many irrelevant dimensions
  - E.g.,Background behind the object

Illustration from here and here

KNN is only for toy problems though

- Curse of dimensionality
  - Most volume of hyper-sphere concentrates near its shell
  - Need many training examples to well cover the space



  - There are hubs, very popular nearest neighbors

Illustration from [here]

KNN is only for toy problems though

- Expensive:
  - stores all training data,
  - computational cost $\propto$ #samples $\times$ dimension
  - Resort to approximate NN search
- Dimension Reduction necessary
  - Recall PCA

Parametric Model

- Specify explicit form of $f(\boldsymbol{x}; \boldsymbol{\theta})$ with parameter $\boldsymbol{\theta}$
- Example:
  - linear regression $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$, $\boldsymbol{\theta} = \{\boldsymbol{w}, b\}$
  - Logistic regression
  - Softmax classifier
  - Deep nets

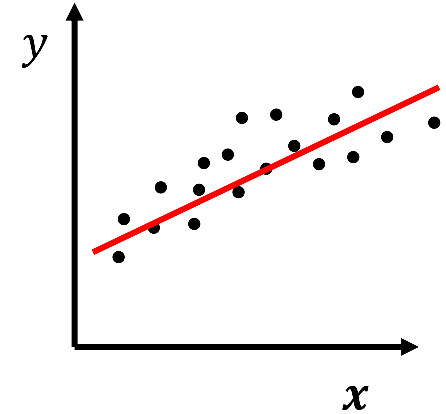
  (Logistic regression, Softmax classifier, Deep nets:) We will talk about them later this lecture
- Train: learn $\boldsymbol{\theta}$ from training samples $\{(\boldsymbol{x}_i, y_i)\}$
- Test: for testing sample $\boldsymbol{x}$, predict using $f(\boldsymbol{x})$

Linear regression

- Parametric form $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$
- Learn $\boldsymbol{w}$ and b from samples $\{(\boldsymbol{x}_i, y_i)\}$, by solving

$$\min_{\boldsymbol{w},b} \sum_{i=1}^{n} \|\boldsymbol{w}^T\boldsymbol{x}_i + b - y_i\|^2$$

$$\equiv \left\|\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{w}} - \boldsymbol{y}\right\|^2$$



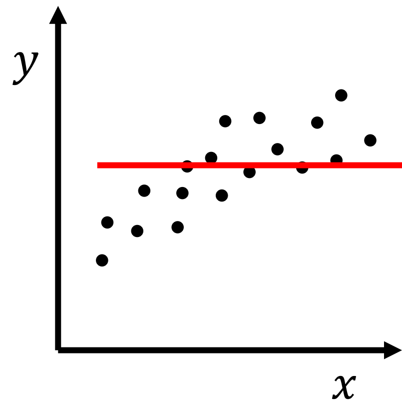
Where $\widetilde{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{x}_1^T, 1 \\ \vdots \\ \boldsymbol{x}_n^T, 1 \end{bmatrix}$, $\widetilde{\boldsymbol{w}} = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}$, $\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

- Revisit last lecture, $\widetilde{\boldsymbol{w}}^* = \widetilde{\boldsymbol{X}}^\dagger \boldsymbol{y}$

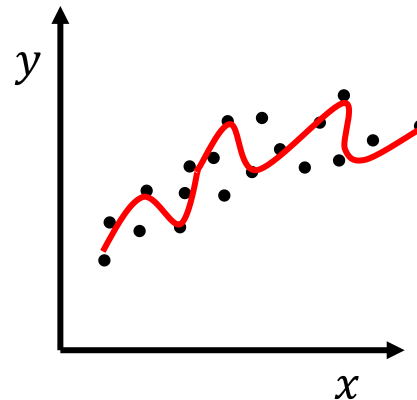
Degree of the polynomial

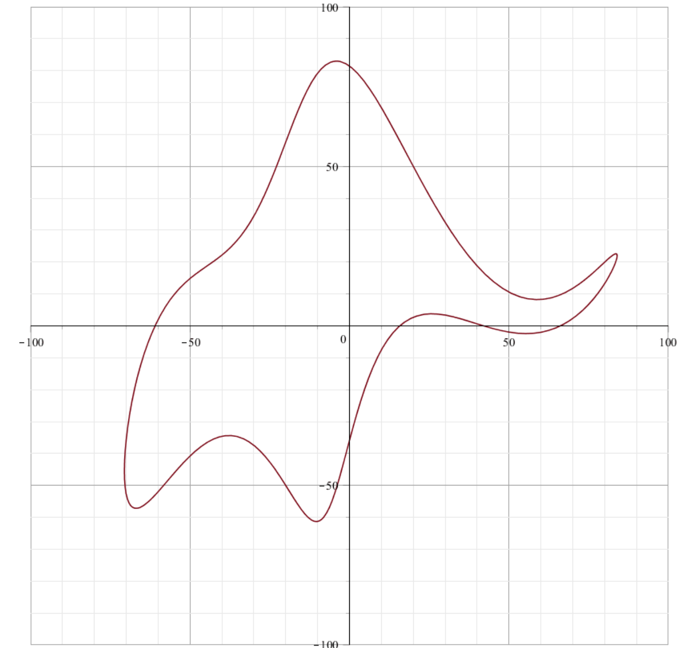
> “With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.”
> --- John von Neumann

- Lower degree
  - Pros: avoid overfitting
  - Cons: less powerful



- Higher degree
  - Pros: more powerful
  - Cons: overfits

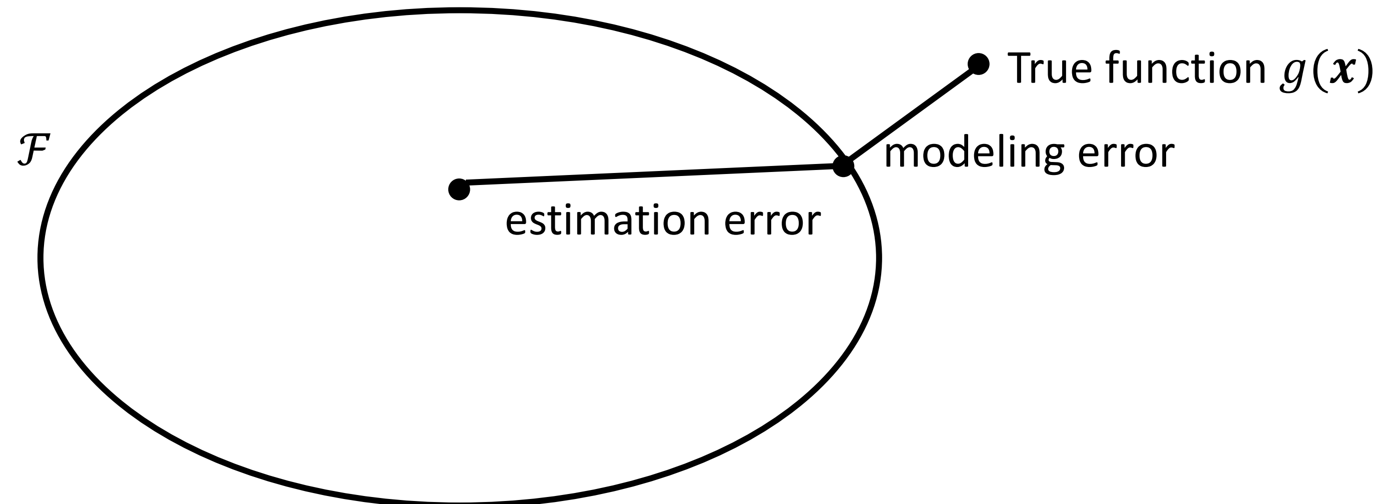




[Fermi-Neumann elephant]

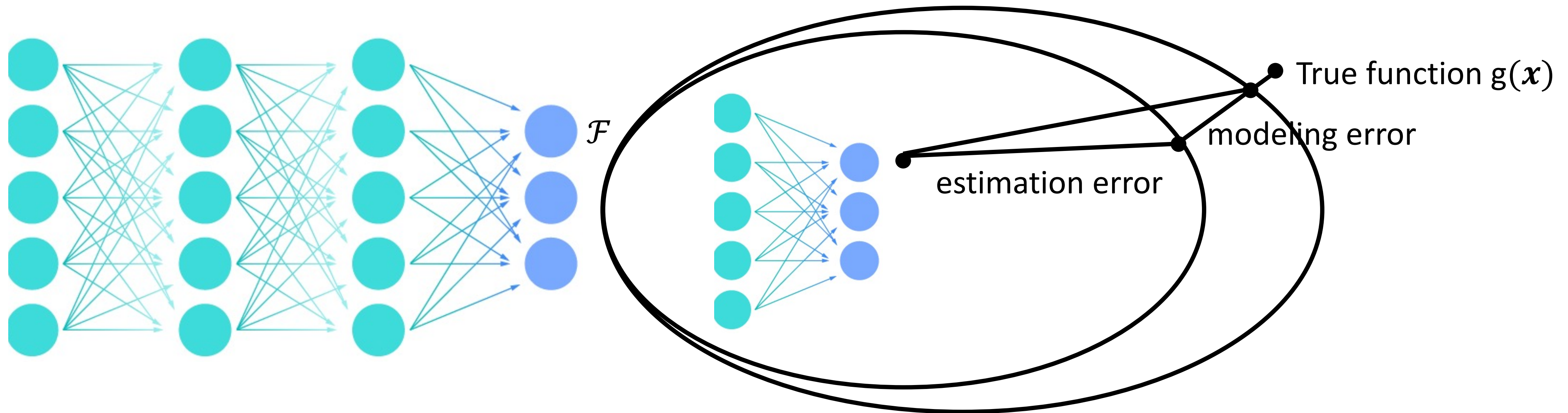
Generalization

- The ability to predict well on unseen samples
- Model family $\mathcal{F} = \{f_{\boldsymbol{\theta}}(\boldsymbol{x})\}$
- Generalization error: expected error/loss on a test input
- $\approx$ modeling error + estimation error



Cartoon adapted from [OMCS lecture slides]

Bias Variance trade-off View



- Modeling error is bias
- Estimation error is variance w.r.t. training data
- bigger $\mathcal{F}$: modeling error↓, but estimation error ↑

Cartoon adapted from OMCS lecture slides

Bias Variance trade-off View

- $y = g(\boldsymbol{x}) + \varepsilon$, ε noise with mean=0, std=σ
- Learn a predictor $\hat{y} = f(\boldsymbol{x}; \mathcal{D})$ by minimizing Mean Square Error (MSE) on training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$
- The generalization error is

$$\begin{aligned}\mathbb{E}_{\mathcal{D},\varepsilon}[(y-\hat{y})^2] &= \mathbb{E}_{\mathcal{D},\varepsilon}[(g(\boldsymbol{x}) - f(\boldsymbol{x};\mathcal{D}) + \varepsilon)^2] \\ &= \mathbb{E}_{\mathcal{D}}\left[[g(\boldsymbol{x}) - f(\boldsymbol{x};\mathcal{D})]^2\right] + \mathbb{E}_{\varepsilon}[\varepsilon^2] \\ &= \mathbb{E}_{\mathcal{D}}\left[[g(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f(\boldsymbol{x};\mathcal{D}) + \mathbb{E}_{\mathcal{D}} f(\boldsymbol{x};\mathcal{D}) - f(\boldsymbol{x};\mathcal{D})]^2\right] + \sigma^2 \\ &= \underbrace{[g(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}} f(\boldsymbol{x};\mathcal{D})]^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}} f(\boldsymbol{x};\mathcal{D})]^2}_{\text{variance}} + \underbrace{\sigma^2}_{\text{Irreducible error}}\end{aligned}$$

More Interpretations

- How to generalize well?
- Reduce bias: small training loss, rich $\mathcal{F}$
- Reduce variance: small gap between testing and training loss
  - Bigger training set
  - Regularization: constrain to a subset of $\mathcal{F}$

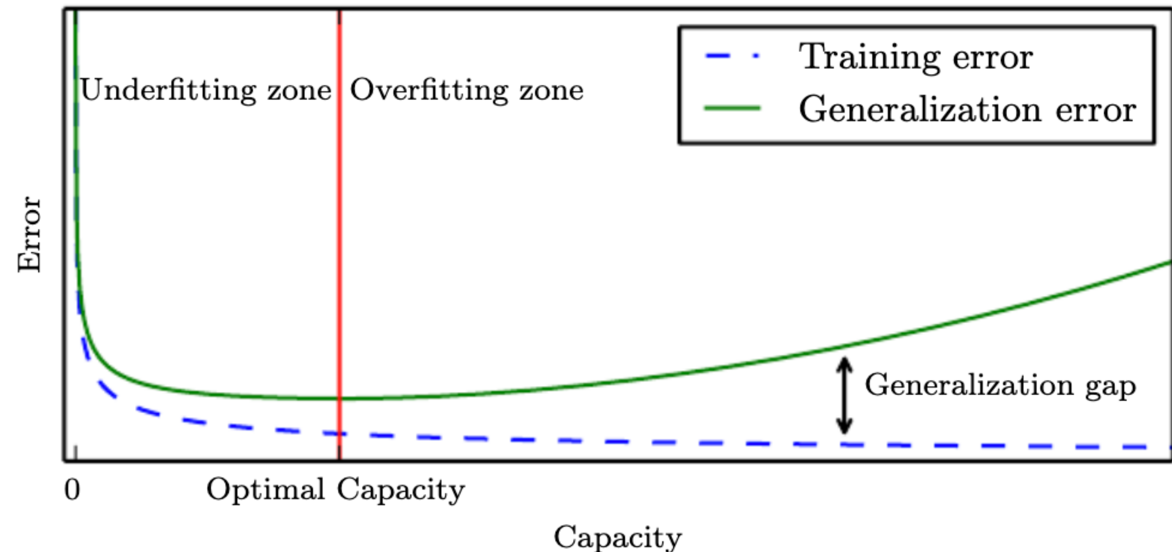


Illustration from textbook chapter 5

20 years of research in Learning Theory oversimplified

If you have:

Enough training data $\mathcal{D}$ and

$\mathcal{F}$ is not too complex

Then:

probably we can generalize to unseen test data

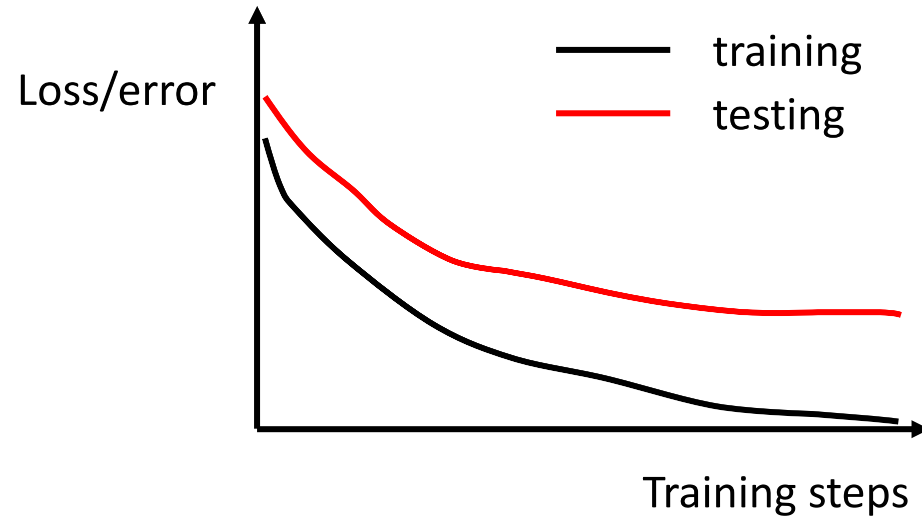
Caveats:

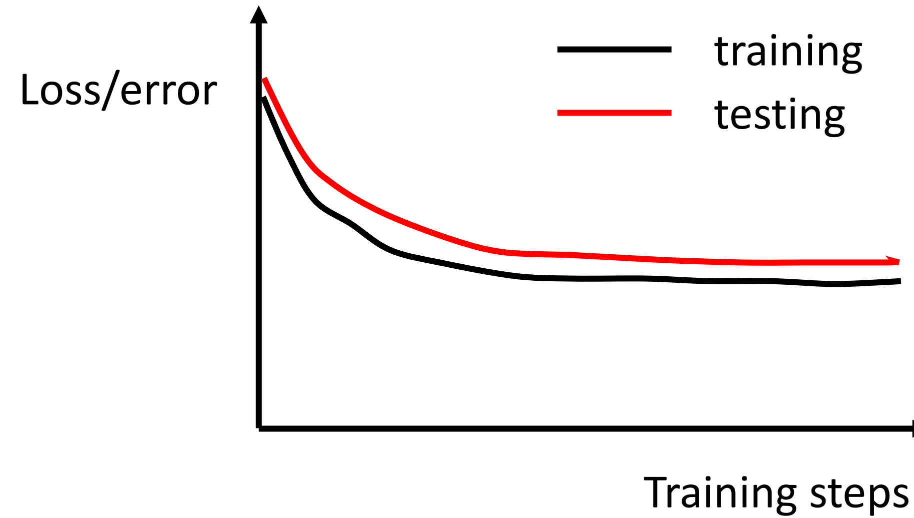
A number of recent empirical results (Zhang et. al) question our intuitions built from this clean separation.

Slides adapted from OMCS lecture slides (page 34)

Exercise

- Judge overfitting/underfitting from learning curves

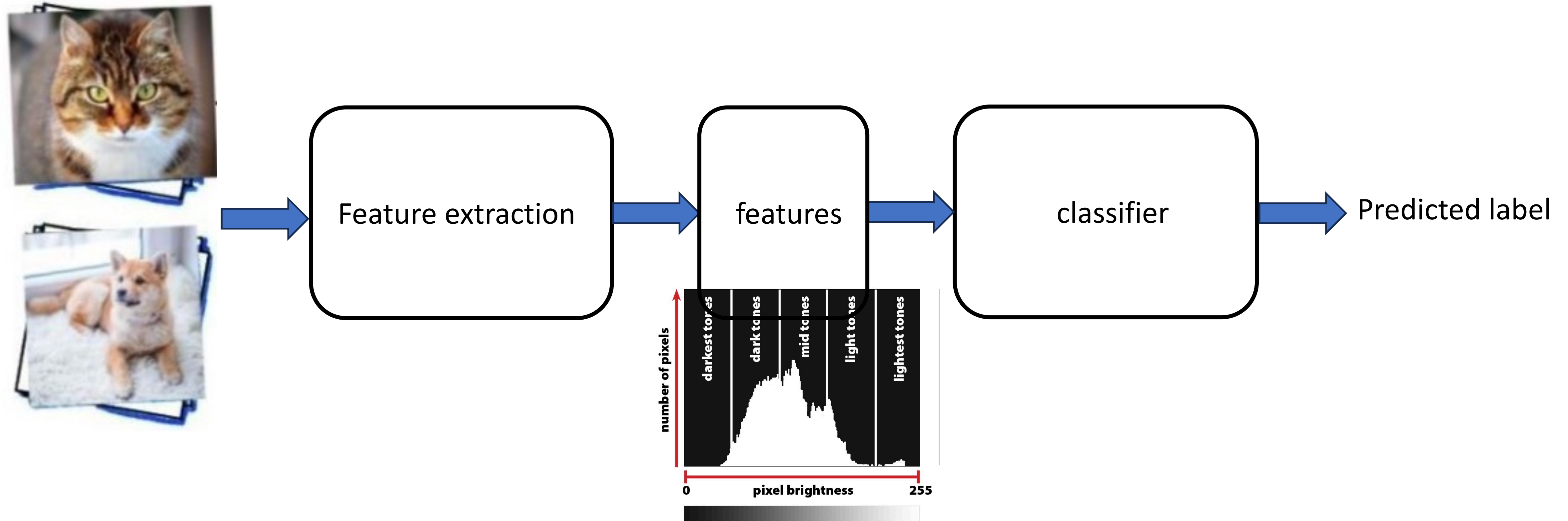




Agenda

- Machine Learning Paradigms
- Non-Parametric v.s. Parametric Models
- Linear Classifier
- Multi-layer Perceptron (MLP)

Classification problem

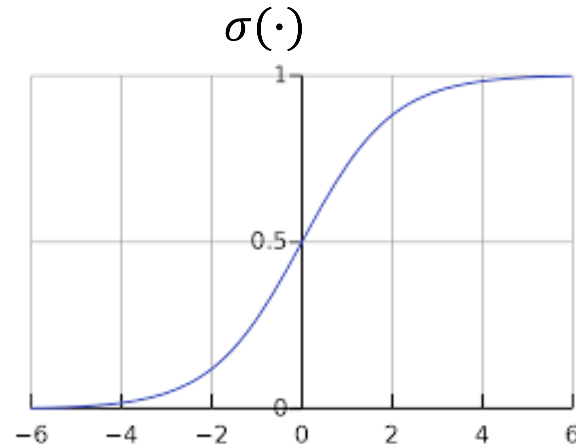


- Before deep learning, we hand-craft the features, e.g., histogram
- With deep learning, we learn the features jointly with classifier

Illustration of histogram from this page

Binary classification: Logistic Regression

- Use a hyperplane to separate the two classes
- $p(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$
- $\sigma(z) = \frac{1}{1+e^{-z}}$: sigmoid function
- $\boldsymbol{w}^T \boldsymbol{x} + b$: logit



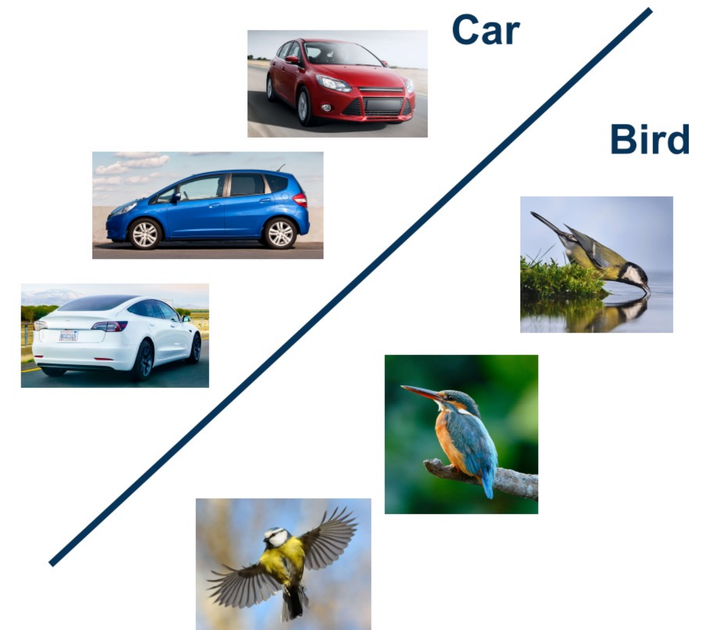


Illustration from [OMCS lecture 2 slides]

Training loss

- Recall Cross-entropy

$$\mathbb{E}_{y\sim p_{data}}[-\log p(y)]$$

- Discretize for each data sample, that's

$$\frac{1}{N}\sum_{i=1}^{N} -p_{data}(y_i = 1)\log p(y_i = 1|\boldsymbol{x}_i)$$
$$-p_{data}(y_i = 0)\log p(y_i = 0|\boldsymbol{x}_i)$$

- Compactly, that's

$$\frac{1}{N}\sum_{i} -y_i \log p(y_i = 1|\boldsymbol{x}_i) + (y_i - 1)\log[1 - p(y_i = 1|\boldsymbol{x}_i)]$$

Gradient

$$\frac{1}{N}\sum_i \{-y_i \log p(y_i = 1|\boldsymbol{x}_i) + (y_i - 1)\log[1 - p(y_i = 1|\boldsymbol{x}_i)] \equiv \ell_i\}$$

- Derive $\frac{\partial \ell_i}{\partial \boldsymbol{w}}$ and $\frac{\partial \ell_i}{\partial b}$ for $i \in \mathfrak{B}$ (some mini-batch)
- Then update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma \cdot \frac{1}{|\mathfrak{B}|}\sum_{i\in\mathfrak{B}} \frac{\partial \ell_i}{\partial \boldsymbol{w}}, \qquad b \leftarrow b - \gamma \cdot \frac{1}{|\mathfrak{B}|}\sum_{i\in\mathfrak{B}} \frac{\partial \ell_i}{\partial \boldsymbol{b}}$$

- Exercise: derive the gradients
  - Hint: denote $p_i = p(y_i = 1|\boldsymbol{x}_i)$, and invoke chain rule

Multi-class: softmax Classifier

- Multiple hyper-planes
- Defined by $\{(\boldsymbol{w}_c, b_c)\}, c = 1, \ldots, C$
- Logits: $\{\boldsymbol{w}_c^T\boldsymbol{x}+b_c\}$
- $p(y = c|\boldsymbol{x})$ =softmax$_c(\{\boldsymbol{w}_c^T\boldsymbol{x}+b_c\})$
- Softmax$_c(\{z_c\})= \frac{e^{z_c}}{\sum_{i=1}^{C} e^{z_i}}$

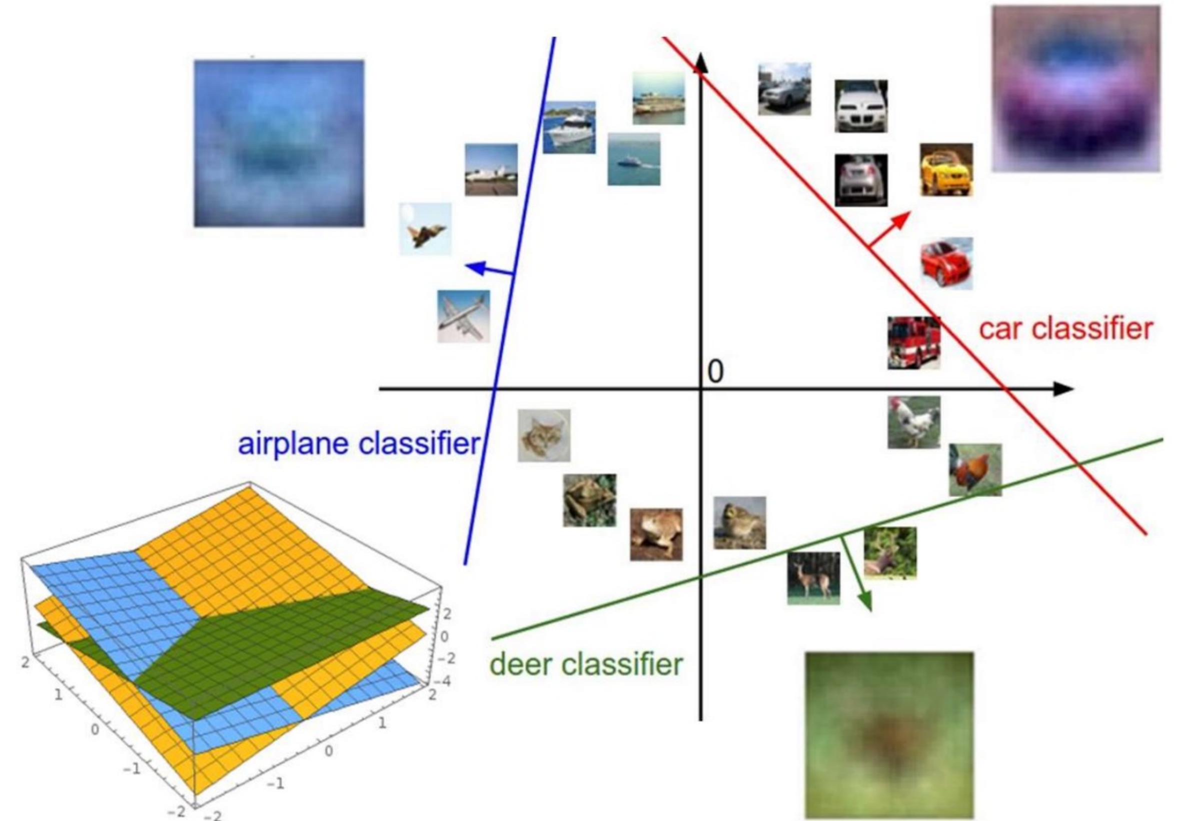


Illustration adapted from [OMCS lecture 2 slides]

Softmax

- E.g., C=3, logits {-5, 0, 5} corresponding to class ID: 0, 1, 2
  - P(y=0)=$\frac{e^{-5}}{e^{-5}+e^{0}+e^{5}} = 4.5094\times10^{-5}$
  - P(y=1)=$\frac{e^{0}}{e^{-5}+e^{0}+e^{5}} = 6.6925\times10^{-3}$
  - P(y=2)=$\frac{e^{5}}{e^{-5}+e^{0}+e^{5}} = 9.9326\times10^{-1}$
- May result in underflow
  - Do $\log p$ instead
  - In pytorch, that's torch.nn.LogSoftmax

Check manual

Training Softmax Classifier

- Cross Entropy

$$\mathbb{E}_{y\sim p_{data}}[-\log p(y)]$$

- Discretize for each data sample, that's

$$\frac{1}{N}\sum_{i=1}^{N} -\log p(y = y_i|\boldsymbol{x}_i)$$

- The minimum loss? 0
- The maximum loss? $+\infty$
- Loss at initialization? $\log C$

Derive the Gradient

- Denote logits $z_c = \boldsymbol{w}_c^T x + b$
- And $p_c = softmax_c(\{z_c\})$
- $\ell = -\log p(y|\boldsymbol{x}) = \log \sum_{c=1}^{C} e^{z_c} - z_y$
- $\frac{\partial \ell}{\partial \boldsymbol{w}_c} = \begin{cases} (p_y - 1) \cdot \boldsymbol{x}, & c = y \\ p_c \cdot \boldsymbol{x}, & c \neq y \end{cases}$
- Practically
  - moves $\boldsymbol{w}_c$ closer to $\boldsymbol{x}$ if it's label $y = c$
  - Otherwise, move away from $\boldsymbol{x}$

Discussion

- Does initialization matter?

  No, the function is convex
- What if the number of classes is very big?

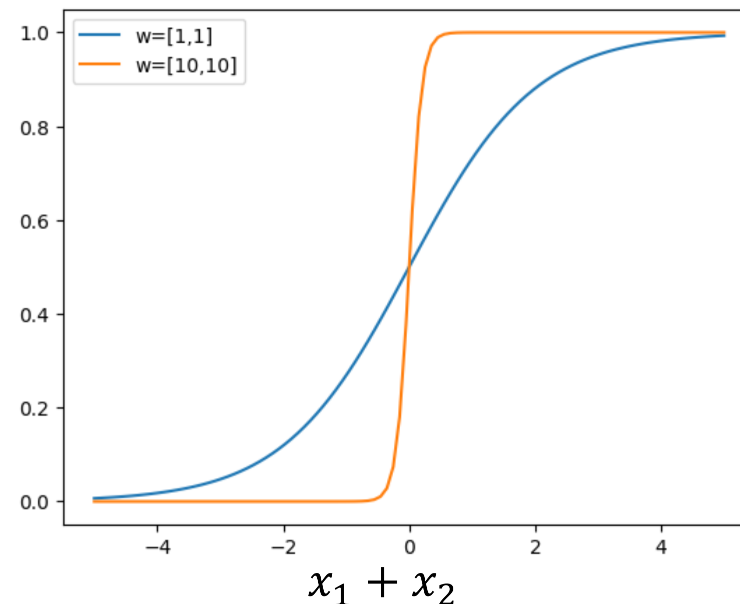
  Hierarchical softmax, sampled softmax

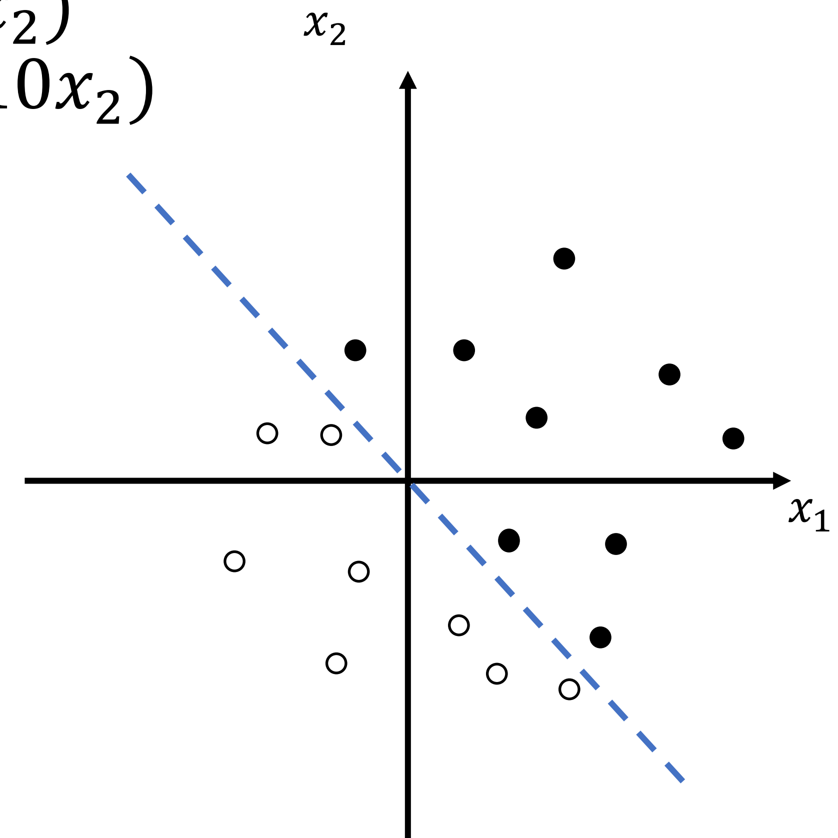
Reduce Overfitting in Linear classifiers

- Consider two logistic regression models

$$\mathcal{M}_1 : p(y = 1|\boldsymbol{x}) = \sigma(x_1 + x_2)$$
$$\mathcal{M}_2 : p(y = 1|\boldsymbol{x}) = \sigma(10x_1 + 10x_2)$$

- Training data is better distinguished by $\mathcal{M}_2$

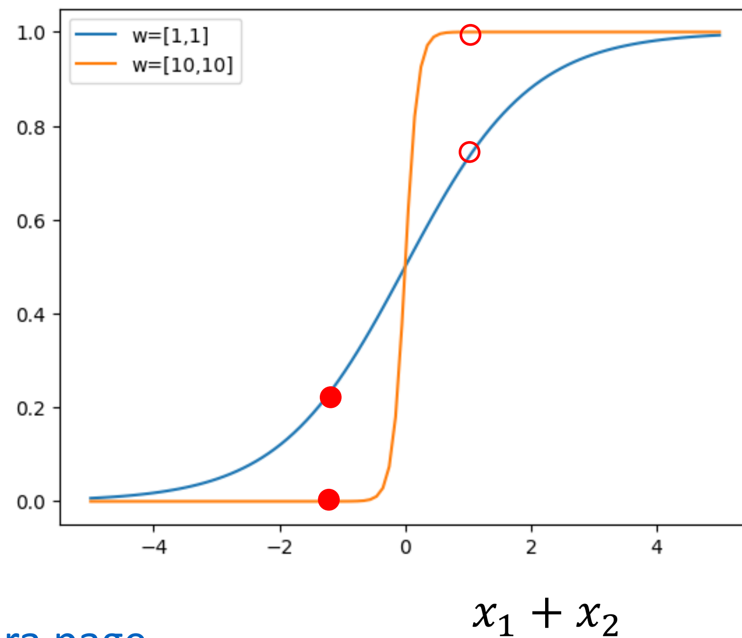


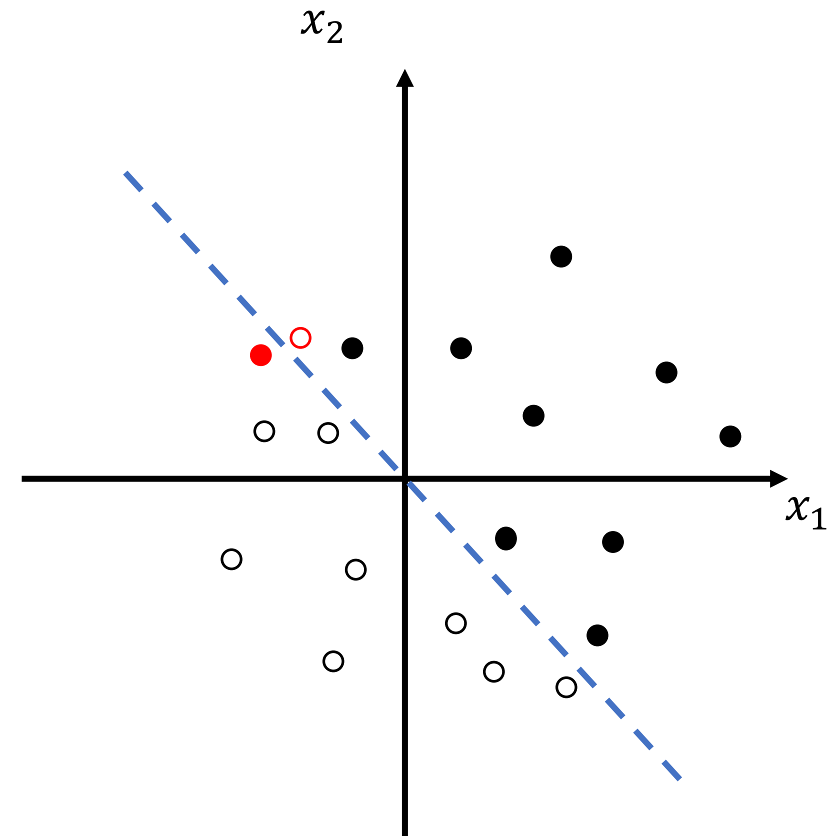


Example from Quora page

Reduce Overfitting in Linear classifiers

- But $\mathcal{M}_2$ is too certain
- $\mathcal{M}_2$ can be very wrong for some test samples





Example from [Quora page]

Reduce Overfitting in Linear classifiers

- Use smaller weights
- Weight decay

$$\text{Cross-entropy loss} + \lambda \|\boldsymbol{w}\|^2$$

- Sparsity

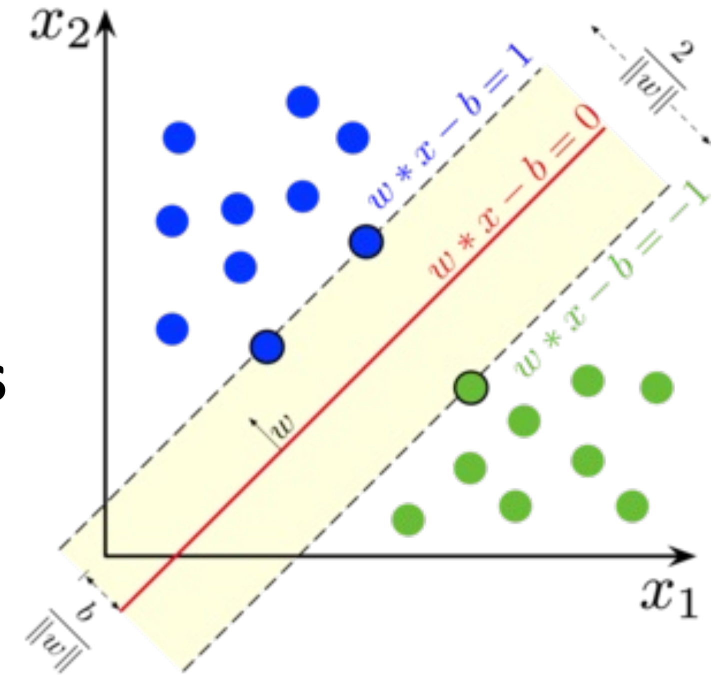
$$\text{Cross-entropy loss} + \lambda \|\boldsymbol{w}\|_1$$

Recap of SVM

- Instead of modeling $p(y|x)$,
- Learn a hyperplane to separate the classes with maximum margin
- Samples on the margin are called **support vectors**
- Learn $\boldsymbol{w}$ by solving:

$$\min \frac{1}{2} ||\boldsymbol{w}||^2$$
$$s.t. (2y_i - 1)(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1$$

- Predict by $\hat{y} = 1$ if $\boldsymbol{w}^T \boldsymbol{x} + b \geq 0$, 0 otherwise



Source: wikipedia

Connect SVM to Logistic Regression

- SVM only asks for good separation
- i.e., if data label $y = 1$, SVM ask

$$\frac{p(y = 1|x)}{p(y = 0|x)} \geq c$$

- Adopting $p(y = 1|x) = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$, that's

$$\boldsymbol{w}^T \boldsymbol{x} + b \geq \log c$$

- Non-unique solution, so we put a penalty $||\boldsymbol{w}||^2$

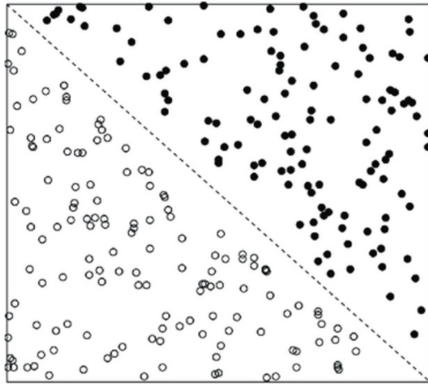
Adapted from CSC2515 slides

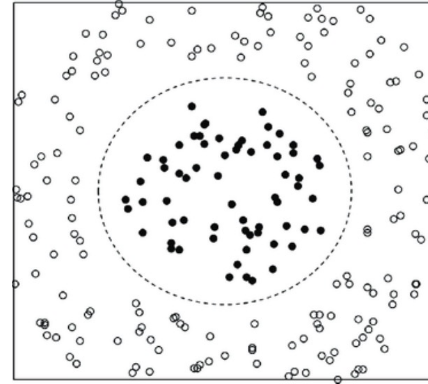
Agenda

- Machine Learning Paradigms
- Non-Parametric v.s. Parametric Models
- Linear Classifier
- Multi-layer Perceptron (MLP)

linearly Separable vs Non-separable

- linearly separable (left) and nonlinear separable (middle)

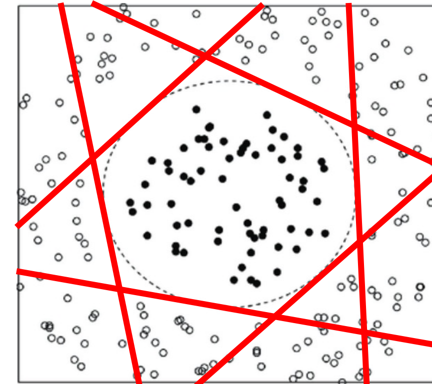




- Logistic regression fails for the right case
  - Huge modeling error (bias)

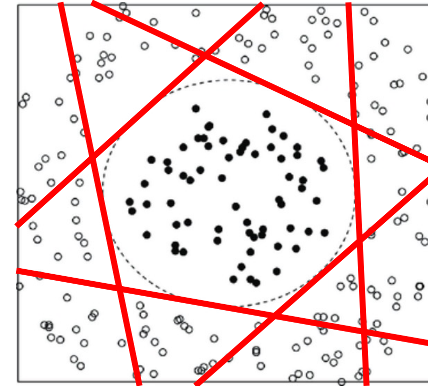
Multi-layer Perceptron (MLP)

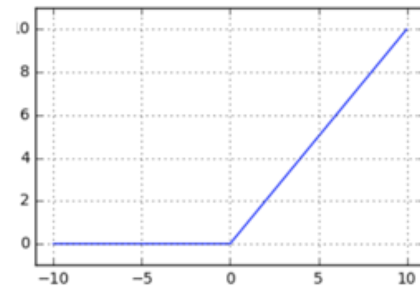
- We could use multiple lines to segregate the classes
- For each line,
  - assign 0 for the side with dots
  - assign 1 for the side with circle
- So the center region get an all-0 coding
- All other regions get at least one coding of 1
- Adding the 6 codings suffice to distinguish



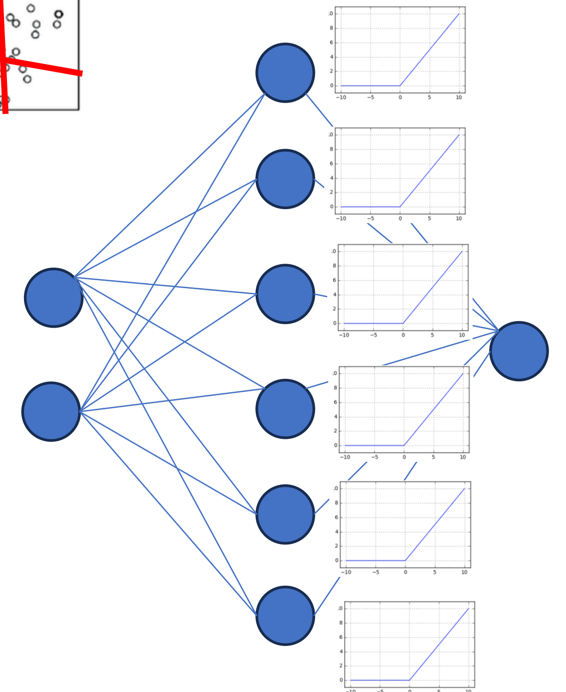
Multi-layer Perceptron (MLP)

- How do we assign the 1's and 0's?
- Need a (nonlinear) activation function
- Rectifier, i.e.,ReLU
- More terminologies:
  - The network is also called feedforward network
  - The linear layer is also called as fully connected layer



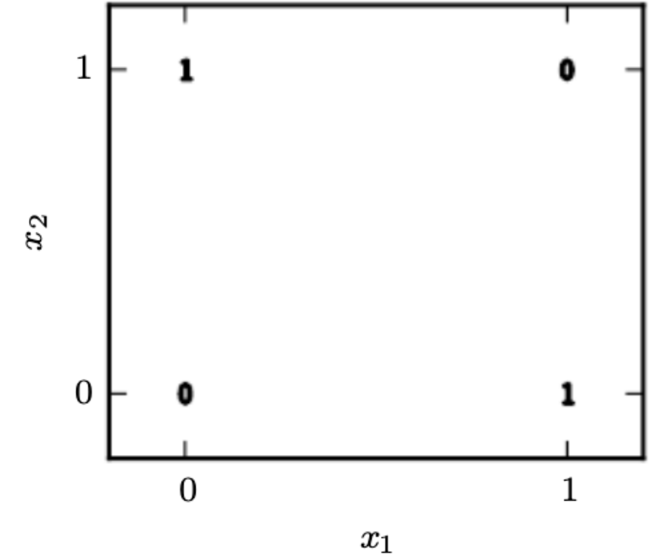


$ReLU(x) = \max(x, 0)$



Another example: modeling XOR

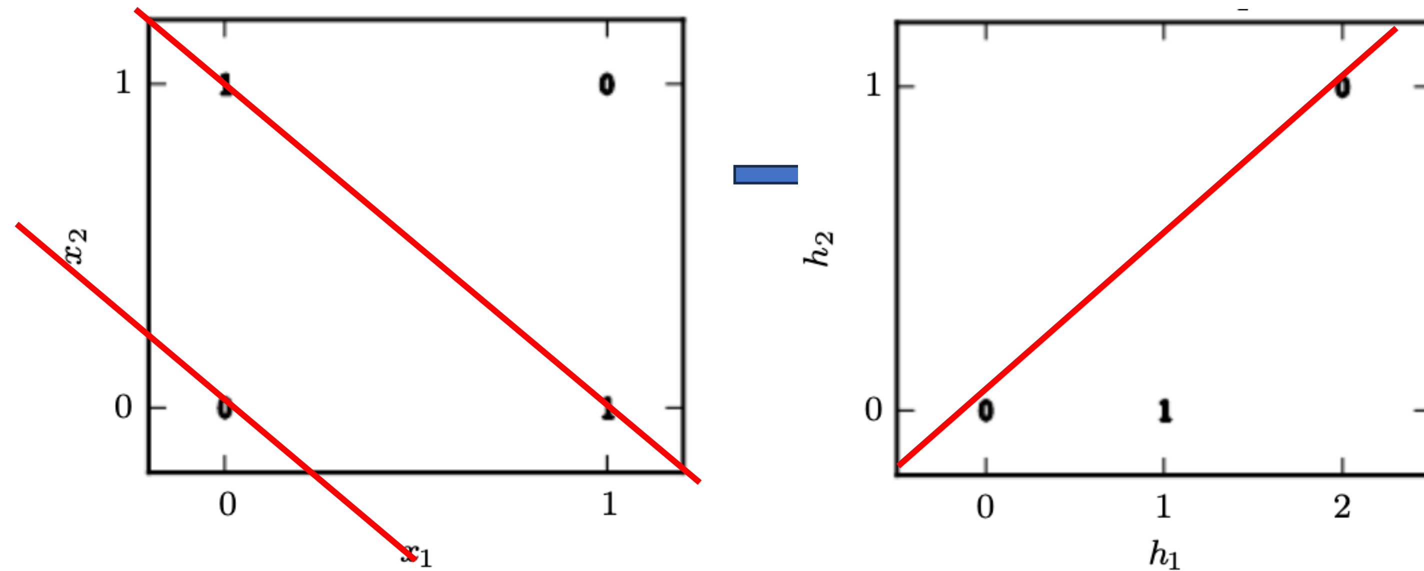
- Find a model space $\mathcal{F}$ that capture this setup
- i.e., no modeling error
- $\mathcal{F}$ cannot be a space of linear models
- As a single line cannot separate the two classes



Plot from deep learning textbook [chapter 5]

Modeling XOR

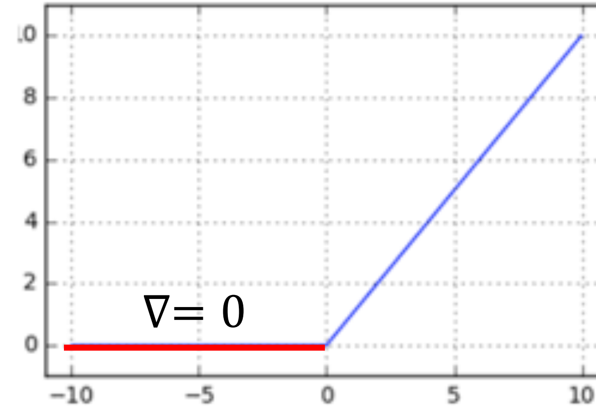
- $f(x) = \boldsymbol{w}^T ReLU(\boldsymbol{W}^T \boldsymbol{x} + c)$
- $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
- $c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$
- $\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

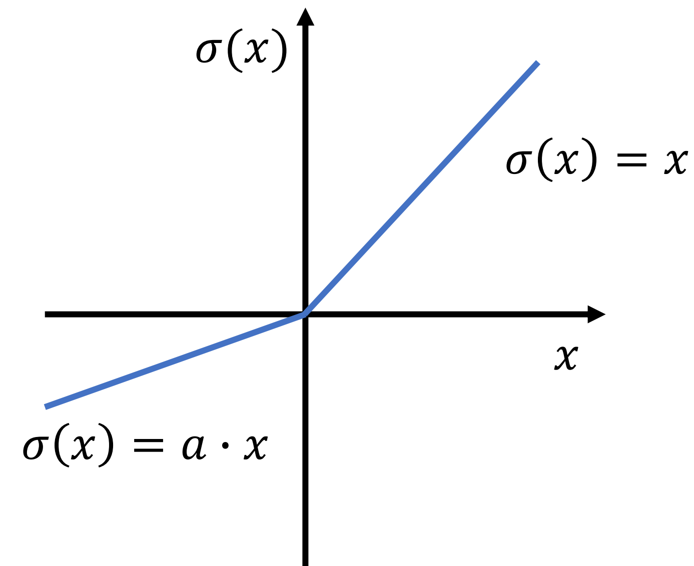


Plots from deep learning textbook [chapter 5]

Generalize ReLU

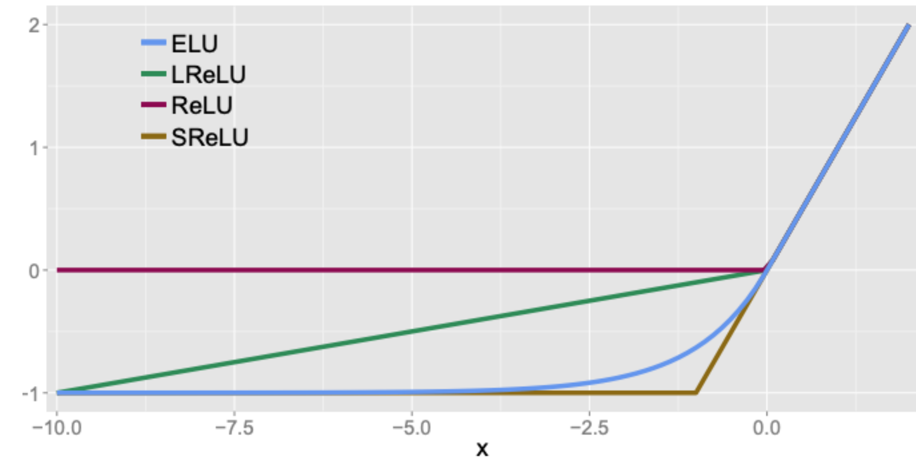
- Drawback of ReLU
- Zero gradient when not activated
  - No learning happens
- Leaky ReLU, fix a small $a > 0$
- Parametric ReLU (PReLU), learn a





Generalize ReLU

- Exponential Linear Units (**ELU**)
- $\sigma(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases} \quad (\alpha > 0)$
- push mean unit activations closer to zero
- Shown to speed up learning

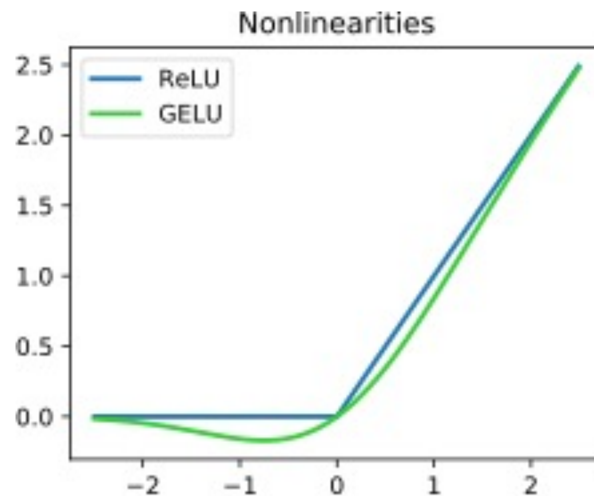


Clevert et. al 2016

Generalize ReLU

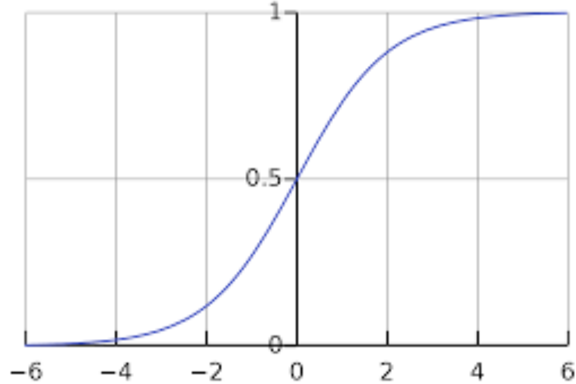
- But still it's non-differentiable at zero
- Gaussian-error Linear Unit (**GELU**): $\sigma(x) = x \cdot \Phi(x)$

  $\Phi(x)$ is the CDF of standard Gaussian distribution
- Used in BERT

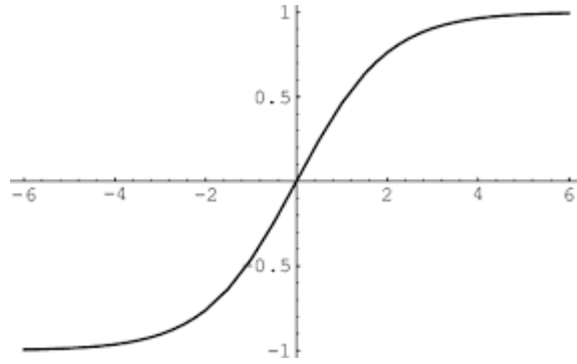


Plot from wikipage

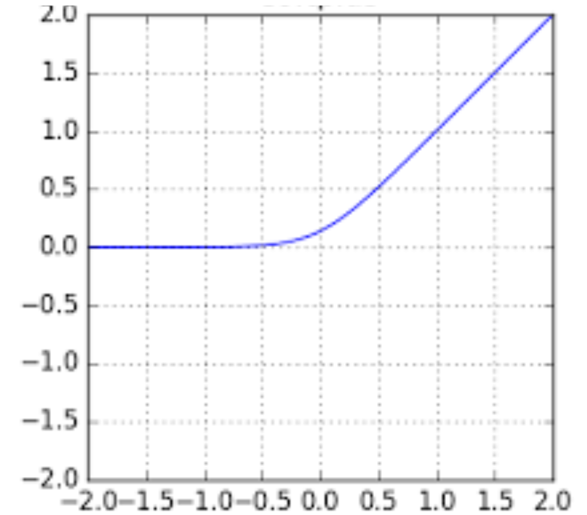
Other activation functions



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



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



$$softplus(x) = \ln(1 + e^{x})$$

Note: sigmoid and tanh are called **squashing function** as their output range is bounded

Recap end-to-end pipeline

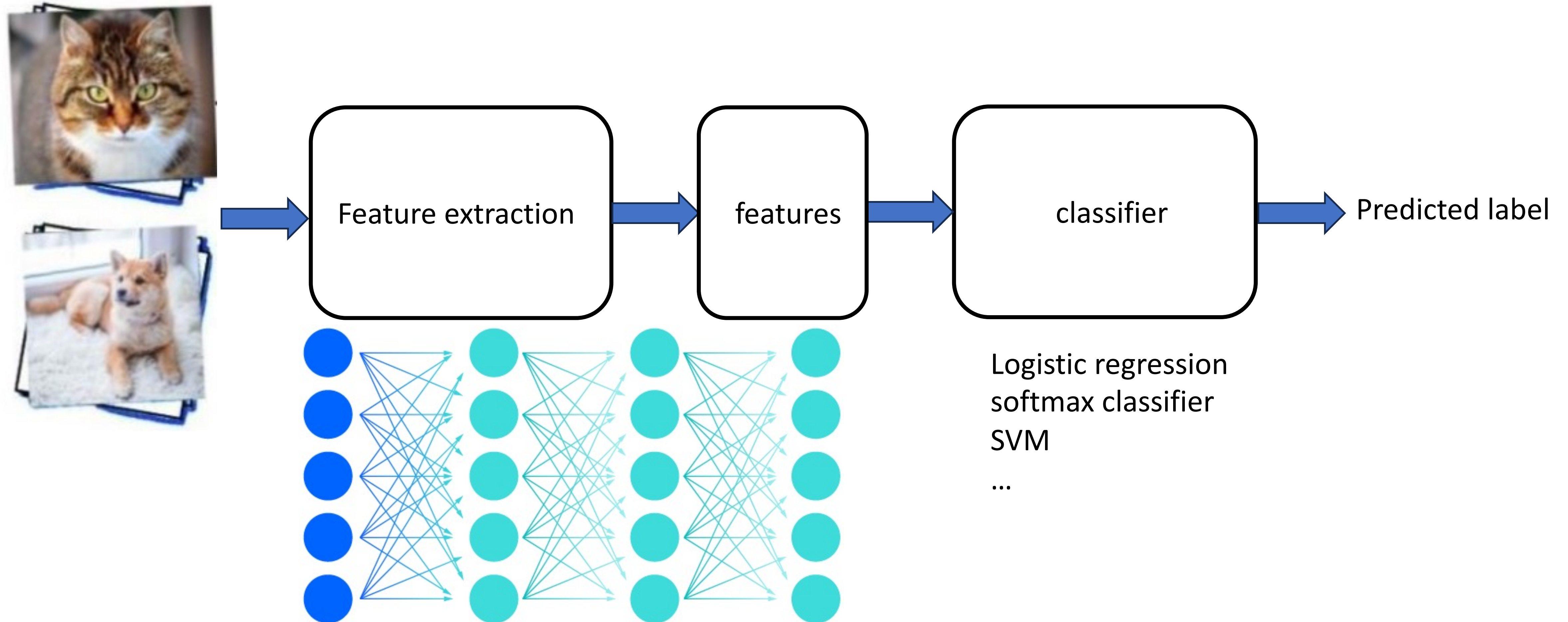
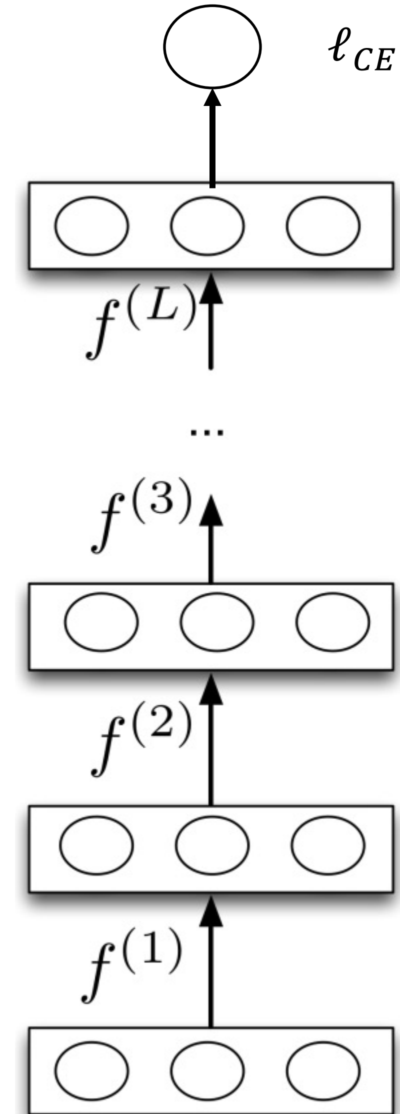


Illustration of histogram from this page

As composite function

- Each layer compute
$$\boldsymbol{x}^{(l)} = f^{(l)}(\boldsymbol{x}^{(l-1)}) = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)})$$
- Over all layers, a composite function
$$f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(\boldsymbol{x})$$
- Last layer is your favorite classifier with a loss
- Trained by back-propagation



Cartoon from [CSC lecture 5 slides]

Pytorch Implementation

- Define and instantiate the network

```
class Network(nn.Module):

    def __init__(self):
        super(Network, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.l3 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = self.l1(x)
        x = self.relu(x)
        x = self.l3(x)
        return F.log_softmax(x)
net = Network()
```

One hidden layer

The softmax classifier

Log softmax avoids underflow

original example here, colab notebook here

Pytorch Implementation

- Define (cross-entropy) loss, optimizer, and train by SGD

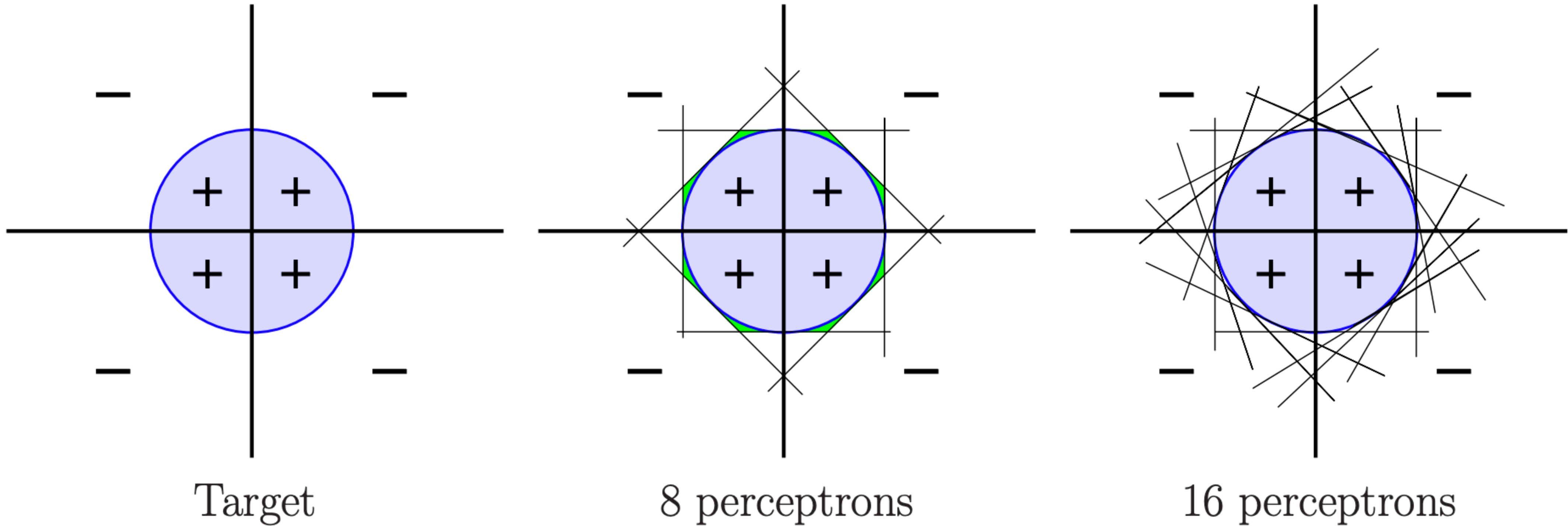
```
optimizer = optim.SGD(net.parameters(), lr=learning_rate, momentum=0.9)
loss_func = nn.CrossEntropyLoss()
loss_log = []

for e in range(epochs):
    for i in range(0, x.shape[0], batch_size):
        x_mini = x[i:i + batch_size]
        y_mini = y[i:i + batch_size]

        x_var = Variable(x_mini)
        y_var = Variable(y_mini)

        optimizer.zero_grad()
        net_out = net(x_var)

        loss = loss_func(net_out, y_var)
        loss.backward()
        optimizer.step()

        if i % 100 == 0:
            loss_log.append(loss.data[0])

    print('Epoch: {} - Loss: {:.6f}'.format(e, loss.data[0]))
```

Don't forget this!
Forward pass

Cross-entropy loss
Auto grad
Optimizer step

original example here, colab notebook here

Universal Approximation Theorem



Slide page from [here]

Universal Approximation Theorem (Simplified)

A feedforward network with

- a linear output layer, and
- at least one hidden layer, with enough hidden units

can approximate any function from one finite-dimensional space to another

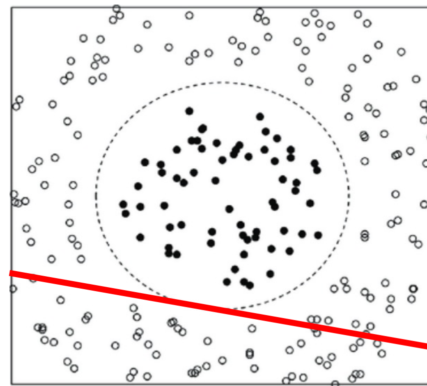
Caveats:

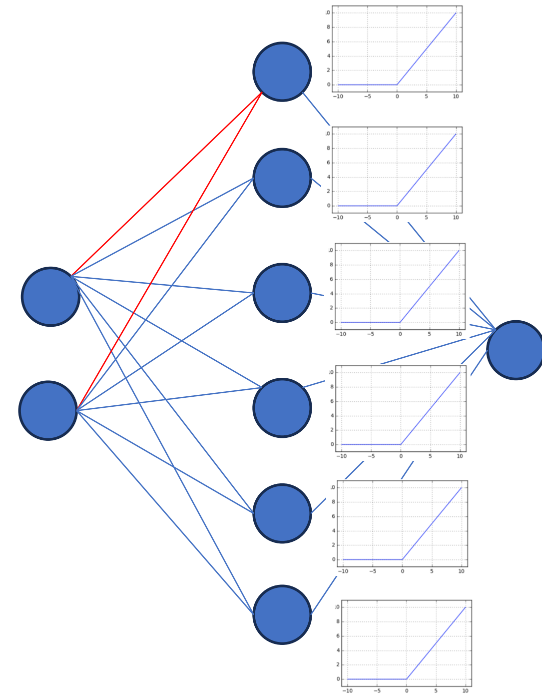
1. Although modeling error is 0, there may still be estimation error
2. Optimizer may fail to find the correct parameter values

Counting (linear) regions

Consider one layer:

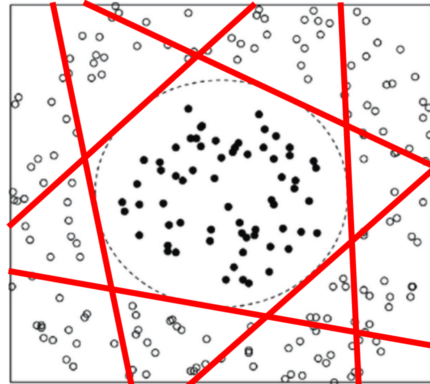
- $ReLU(\boldsymbol{w}^T\boldsymbol{x} + b)$ defines a hyper plane
- One side deactivated (all 0)
- The other side activated: computes $\boldsymbol{w}^T\boldsymbol{x} + b$





Counting (linear) regions

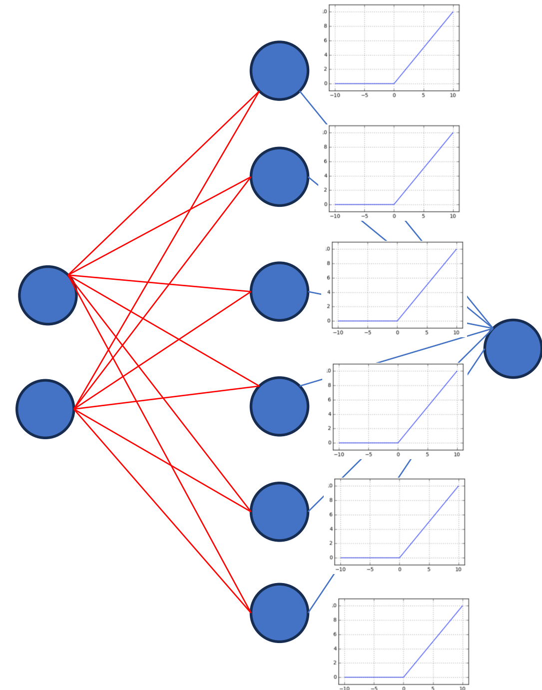
- n $\boldsymbol{w}$'s: separate the space into multiple regions



- More generally, $\boldsymbol{x} \in \mathbb{R}^d$, at most

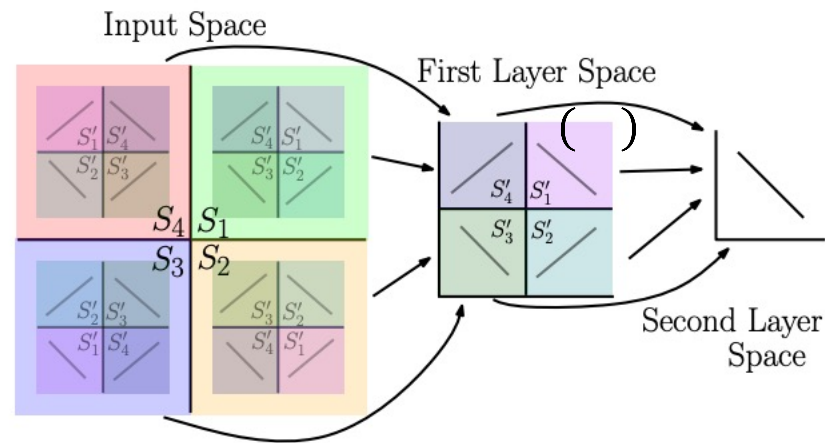
$$\sum_{i=1}^{d} \binom{n}{i}$$

regions



The effect of more layers

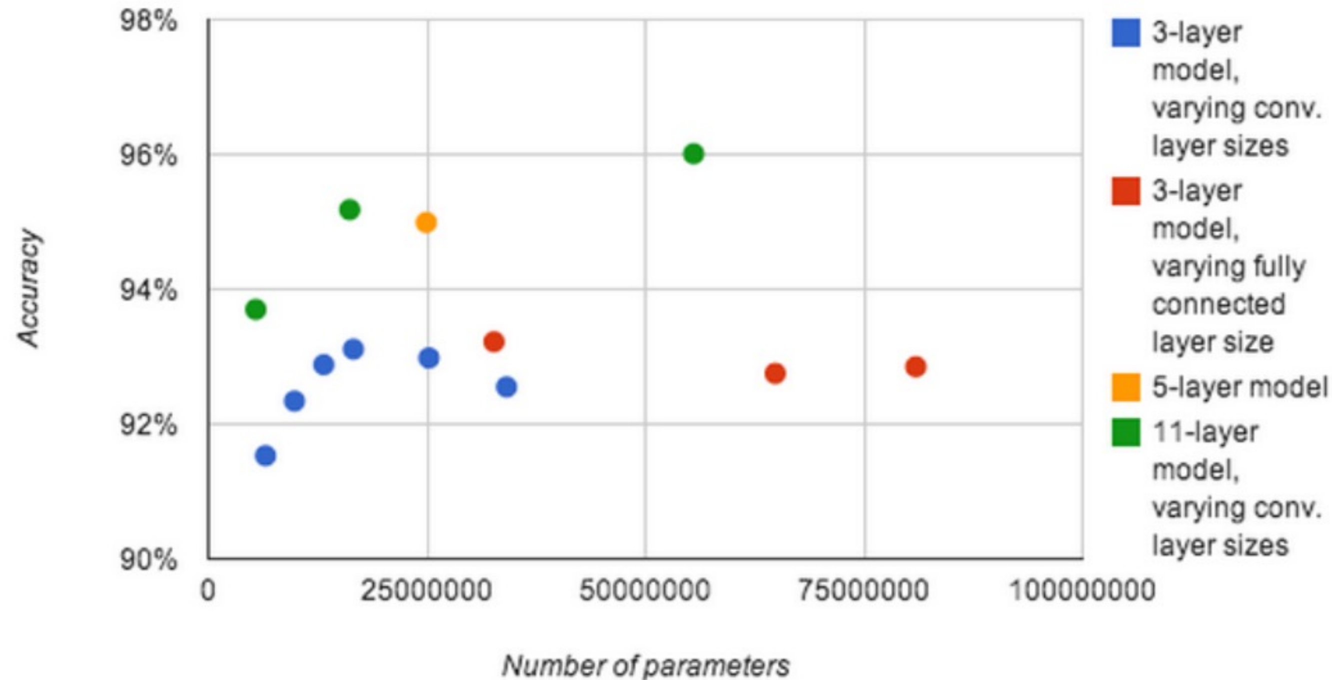
- The 2nd layer works on each of the regions defined by the first layer



- L layer, each n hidden units: Number of regions $O\left(\binom{n}{d}^{d(L-1)} n^d\right)$

Montufar et. al, 2014

Shallow vs Deep



Increasing the number of parameters in shallower models does not allow such models to reach the same level of performance as deep models, primarily due to overfitting.

- (linear) Regions grows exponentially w.r.t depth
- Deeper model use fewer parameter to achieve necessary number of (linear) regions

Plot from [Goodfellow et. al, 2014]