CSE 417T
Introduction to Machine Learning

Lecture 23

Instructor: Chien-Ju (CJ) Ho

- Homework 5: due **April 30** (Friday)
  - Again, please start it early

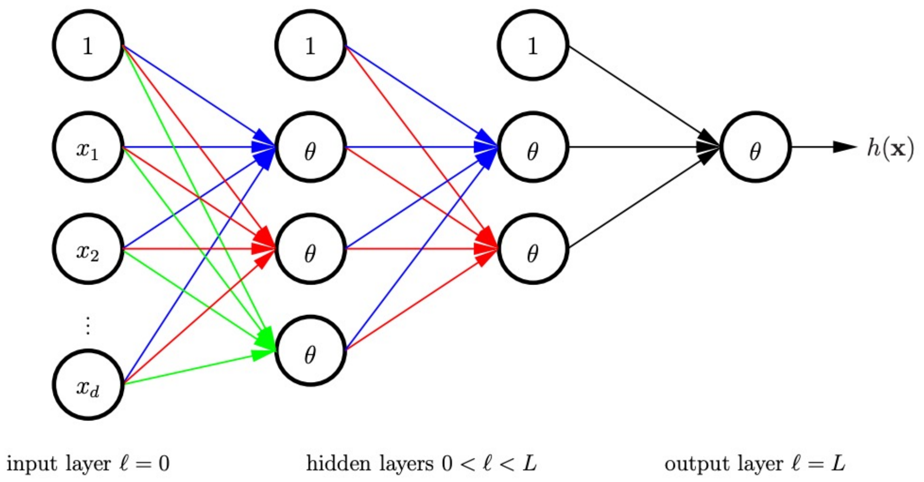
- Exam 2: (**May 4**, Tuesday)
  - Duration: 75+5 Minutes
  - Content: Focus on the content of 2nd half of the semester
    - Though knowledge is cumulative
  - Time: by default, everyone is expected to take it during lecture time
    - If you can't, let me know by this Friday through this google form
  - Review lecture: Apr 29
    - Practice questions will be posted next week
  - Other logistics are the same as Exam 1
    - Format: Gradescope online exam + Zoom (with camera on)
    - Information access during exam:
      - Allowed: Textbook, slides, hardcopy materials (e.g., your own notes)
      - Not allowed: search for information online during exam, talk to any other persons
    - **Follow Piazza announcements** for updates/information

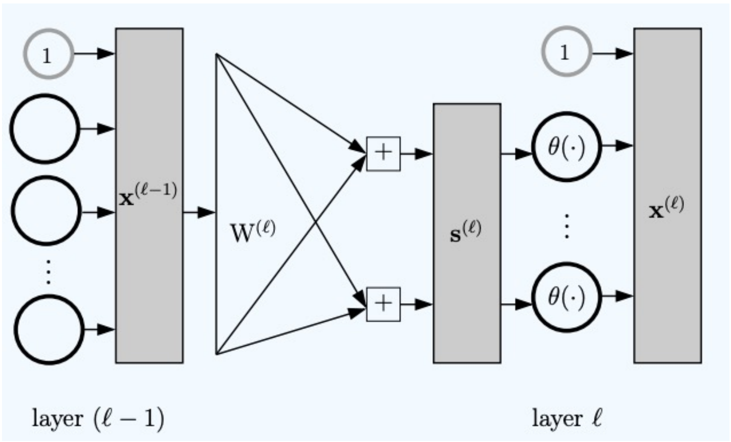
Recap

Neural Networks (NN)

- Notations:
  - $\ell = 0$ to L: layer
  - $d^{(\ell)}$: dimension of layer ℓ
  - $\vec{x}^{(\ell)}$: the nodes in layer ℓ
  - $w_{i,j}^{(\ell)}$: weights; characterize hypothesis in NN
  - $s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{i,j}^{(\ell)} x_i^{(\ell-1)}$: linear signals
  - θ: activation function
    - $x_j^{(\ell)} = \theta\left(s_j^{(\ell)}\right)$

Feed-forward network

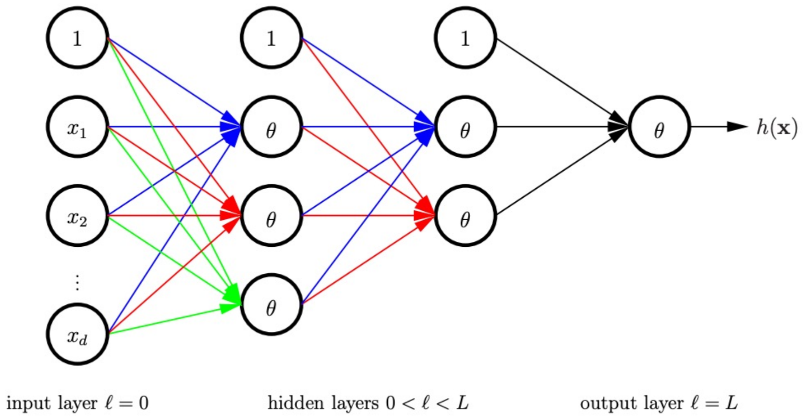




Forward Propagation and Backpropagation

- Evaluate $h(\vec{x})$ given h (characterized by $\left\{w_{i,j}^{(\ell)}\right\}$)
  - Forward propagation

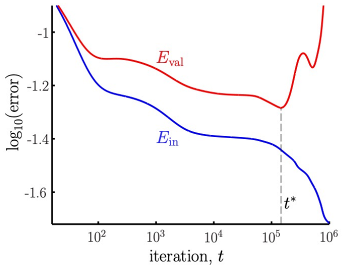
$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \cdots \xrightarrow{\mathrm{W}^{(L)}} \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x}).$$



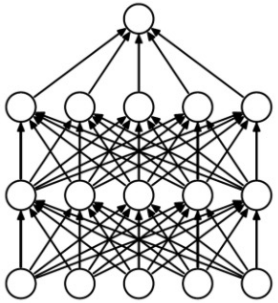
- Given D, learn the weights $W = \left\{w_{i,j}^{(\ell)}\right\}$
  - Backpropagation
  - Initialize $w_{i,j}^{(\ell)}$ randomly
  - For $t = 1$ to T
    - Randomly pick a point from D (for stochastic gradient descent)
    - Forward propagation: Calculate all $x_i^{(\ell)}$ and $s_i^{(\ell)}$
    - Backward propagation: Calculate all $\delta_j^{(\ell)}$
    - Update the weights $w_{i,j}^{(\ell)} \leftarrow w_{i,j}^{(\ell)} - \eta\delta_j^{(\ell)} x_i^{(\ell-1)}$
  - Return the weights

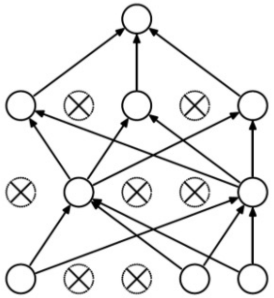
Regularizations in Neural Networks

- Weight-based regularization
- Early stopping

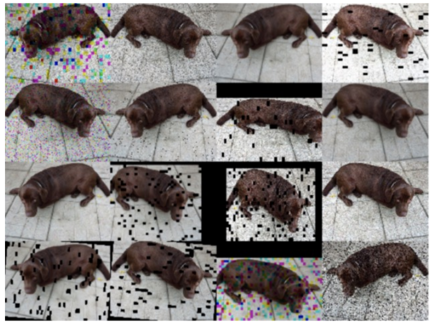


- Dropout



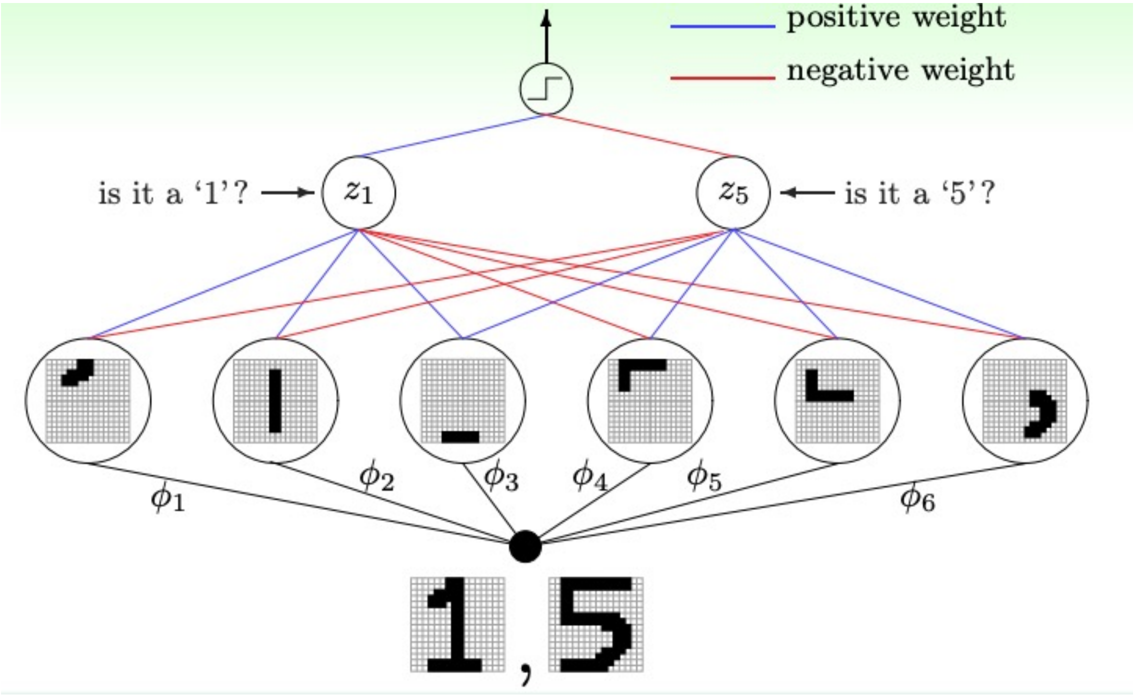


- Adding noises



Deep Learning (NN with many layers)

- “Shallow” neural network is powerful (universal approximation theorem holds with a single hidden layer). Why “deep” neural networks?



Each layer captures features of the previous layers.

We can use “raw data” (e.g., pixels of an image) as input. The hidden layer are extracting the features.

Design different network architectures to incorporate domain knowledge.

Some Techniques in Improving Deep Learning

- Regularization to mitigate overfitting
  - Weight-based, early stopping, dropout, etc
- Incorporating domain knowledges
  - Network architectures (e.g., Convolutional Neural Nets)
- Improving computation with huge amount of data
  - Hardware architecture to improve parallel computation
- Improving gradient-based optimization
  - Choosing better initialization points

Today’s Lecture

The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook.

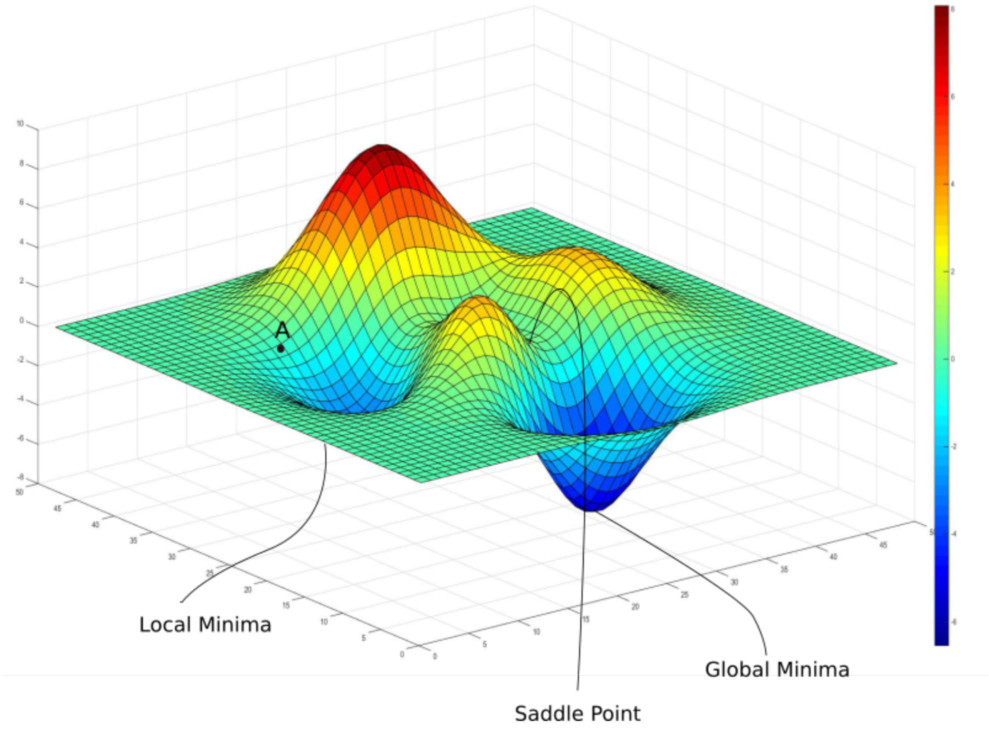
Let me know if you spot errors.

Initialization

Why initialization matters in deep learning

- Error is nonconvex in NN
- Vanishing/exploding gradient problem

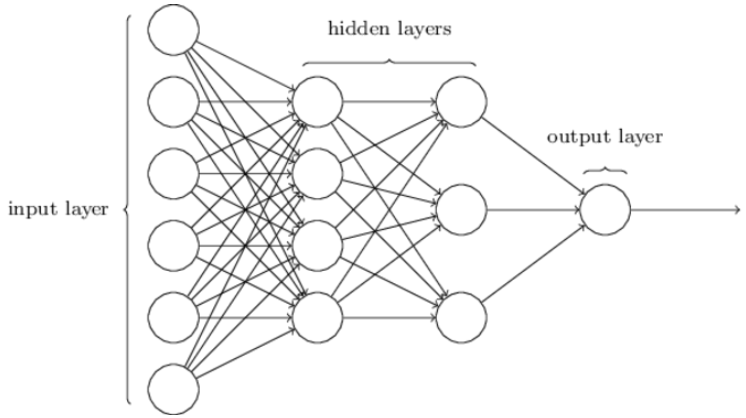
Error is Nonconvex in Neural Networks



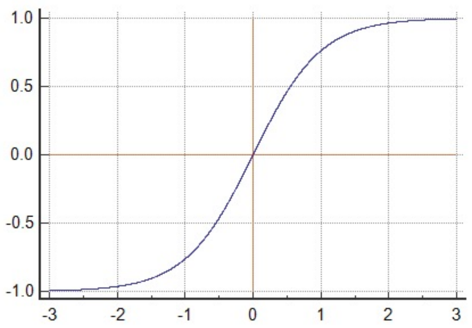
- We mostly adopt gradient-descent-style algorithms for optimization.
- No guarantee to converge to global optimal.
- Need to run it many times.
- Initialization matters!

Vanishing Gradient Problem

- Backpropagation
  - $\frac{\partial e_n(W)}{\partial w_{i,j}^{(\ell)}} = \delta_j^{(\ell)} x_i^{(\ell-1)}$
  - $\delta_j^{(\ell)} = \theta'\left(s_j^{(\ell)}\right) \sum_{k=1}^{d^{(\ell+1)}} \delta_k^{(\ell+1)} w_{j,k}^{(\ell+1)}$



- If we use activation function $\theta(s) = \tanh(s)$
  - $\theta'(s) = 1 - \theta(s)^2 < 1$



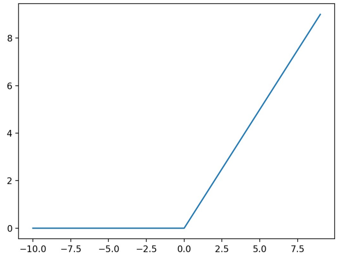
- In deep learning with a lot of layers,
  - the gradient might vanish
  - hard to update the early layers

Vanishing Gradient Problem

- $\delta_j^{(\ell)} = \theta'\left(s_j^{(\ell)}\right) \sum_{k=1}^{d^{(\ell+1)}} \delta_k^{(\ell+1)} w_{j,k}^{(\ell+1)}$

- There is also a corresponding "exploding gradient problem"

- What can we do
  - Choose more suitable activation functions
    - One common choice is Rectified Linear Unit (ReLU) and its variant
      - $\theta(s) = \max(0, s)$
  - Choose better initialization
    - Many approaches



Weight Initialization

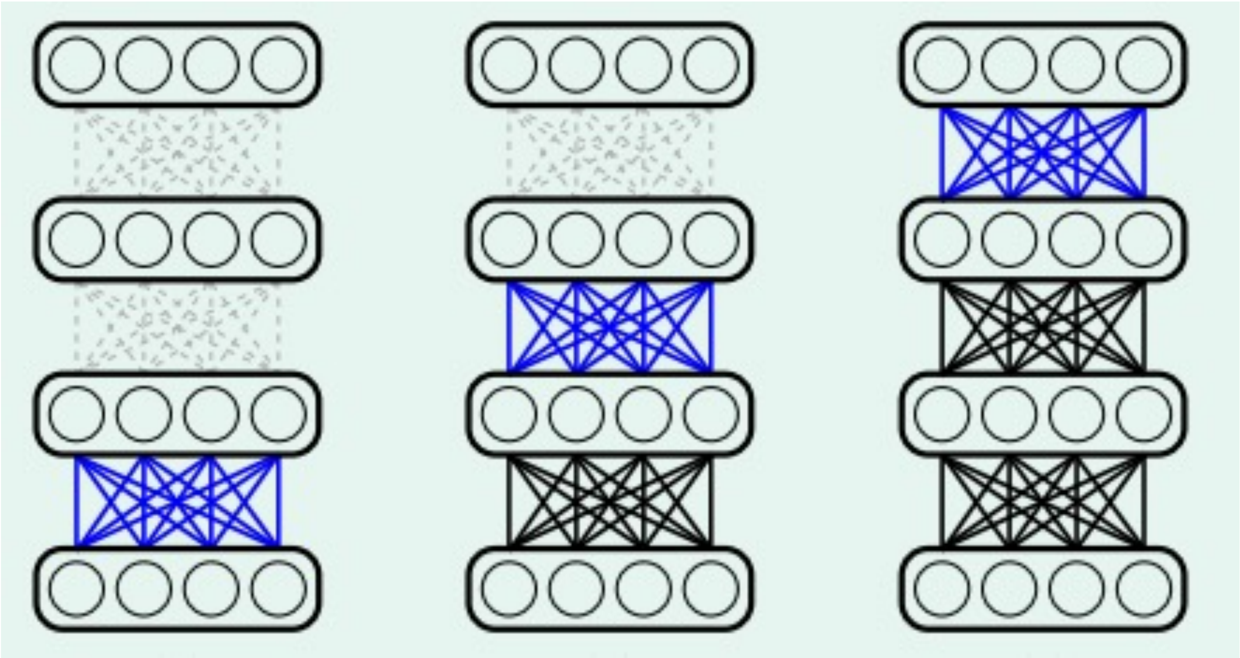
- Initializing weights to all 0 is a bad idea
  - Q5 of HW1
  - Hint: Look at the backpropagation formulation

- Randomly Initializing weights to regions so that vanishing/exploding gradients are less likely to happen
  - Activation-function dependent
    - e.g., Xavier initialization for tanh

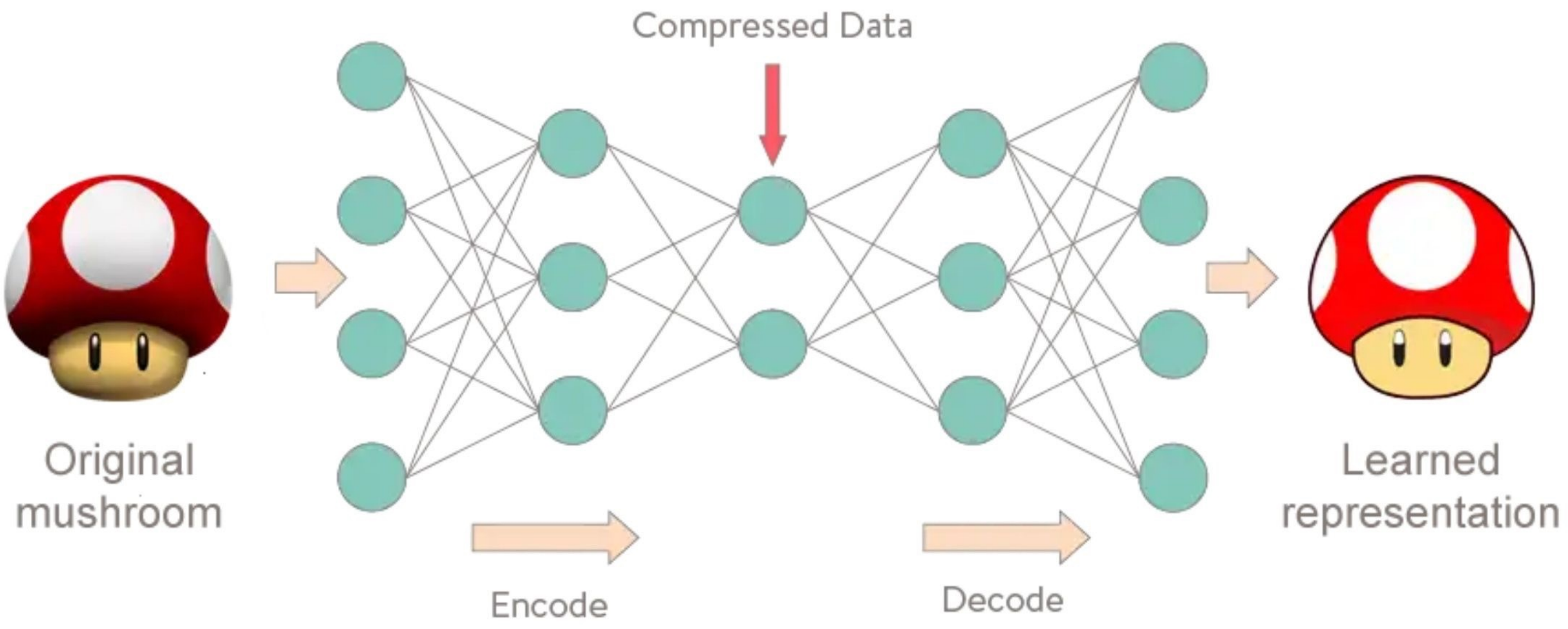
- Learning the initialization that might be closer to the optimal
  - E.g., using autoencoder

Initialization

- Hard to initialize the entire network well.
- Intuition: Initialize the weights layer by layer such that each layer preserves the properties of the previous layer.



Autoencoder



Unsupervised learning!

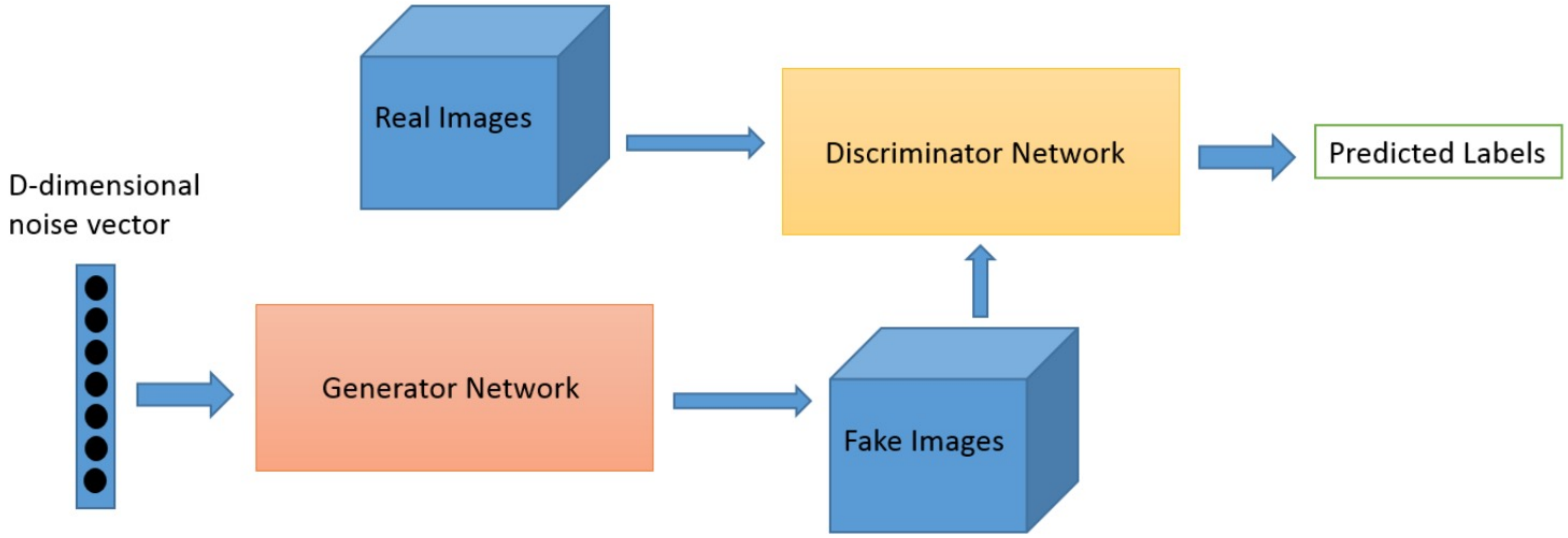
https://www.meetup.com/Le-Mans-School-of-AI/photos/29159837/477498782/

Cool Stuffs for Deep Learning

[Safe to Skip for the exam]

Generative Adversarial Nets (GAN)

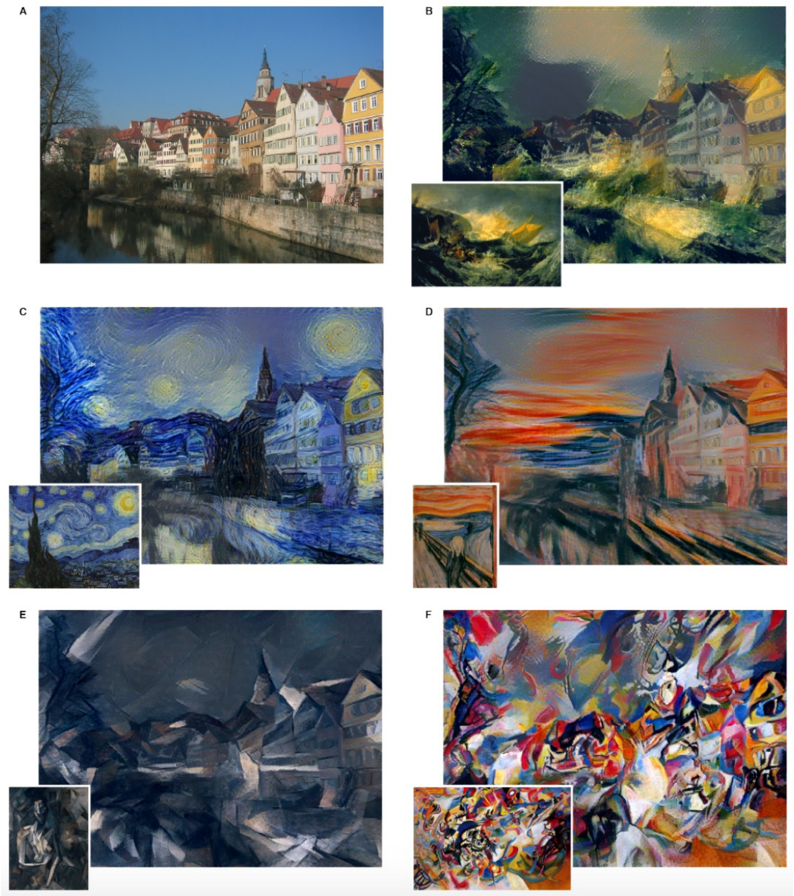
- A Competition: Generator vs Discriminator
  - Discriminator wants to correctly classify the images (true images or not)
  - Generator wants to generate images that discriminator can't classify



[Safe to Skip for the Exam]

https://thisPersonDoesNotExist.com/

Style Transfer



- Informal intuitions:
  - Recall that we can treat hidden layers as feature transforms
  - Deep learning is learning representation of data
  - How to achieve style transfer:
    - Learn a content representation for an image using hidden layers
    - Learn a style representation for an image using hidden layers
    - Compute an image that jointly minimizes the distance from the content image's content representation and the style image's style representation
    - https://arxiv.org/pdf/1508.06576.pdf

[Safe to Skip for the Exam]

Radial Basis Functions (RBF)

Instance-Based Learning

- Make predictions based on data instances

- k-nearest neighbor (K-NN)
  - $g(\vec{x}) = sign\left(\sum_{i=1}^{k} y_{[i]}(\vec{x})\right)$
  - Predict according to the nearest neighbors
- Kernel SVM
  - $g(\vec{x}) = sign\left(\sum_{\alpha_n^*>0} \alpha_n^* y_n K(\vec{x}_n, \vec{x}) + b^*)\right)$
  - Predict according to support vectors
- Radial Basis Functions (RBF)
  - Focus of today

Radial Basis Functions

- Think about k-nearest neighbor (K-NN) again
  - $g(\vec{x}) = sign\left(\sum_{i=1}^{k} y_{[i]}(\vec{x})\right)$

  - Make predictions based on k nearest data points
  - Each of the k data points has the same weight

- Natural questions:
  - Can we use more (or even all) data?
  - Weight them based on how close data points are to $\vec{x}$

Radial Basis Functions

- Given dataset $D = \{\vec{x}_1, \ldots, \vec{x}_N\}$
- Task: Make a prediction on $\vec{x}$

- Radial Basis Function:
  - $g(\vec{x}) = \frac{1}{Z(\vec{x})} \sum_{n=1}^{N} \phi\left(\frac{\|\vec{x}-\vec{x}_n\|}{r}\right) y_n$
  - $\phi(s)$: a monotonically decreasing function

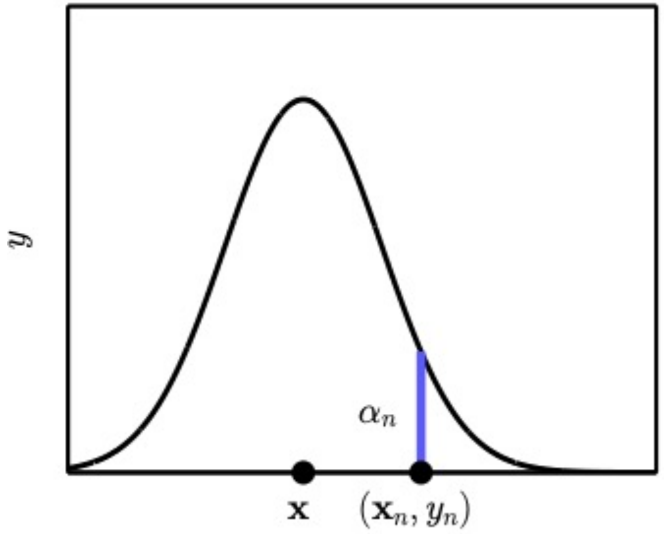
- This is for regression. We can take a sign and make it a classification.
- $Z(\vec{x}) = \sum_{m=1}^{N} \phi\left(\frac{\|\vec{x}-\vec{x}_m\|}{r}\right)$ is for normalization

- It's called radial basis function since it takes the distance to the points as the basis function

Radial Basis Functions

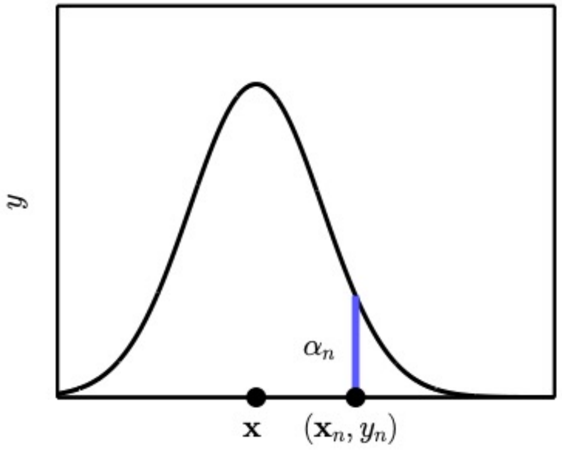
- Radial Basis Function:
  - $g(\vec{x}) = \sum_{n=1}^{N} \frac{1}{Z(\vec{x})} \phi\left(\frac{\|\vec{x}-\vec{x}_n\|}{r}\right) y_n$
  - Example of ϕ
    - Gaussian RBF (we have seen this in SVM): $\phi(s) = e^{-s}$

- Intuitions
  - The impact of $\vec{x}_n$ to $\vec{x}$ is higher if it's closer to $\vec{x}$
  - The role of r is similar to k in k-NN
    - $r = 0$: 1-NN
    - $r \to \infty$: N-NN (i.e., $k = N$)



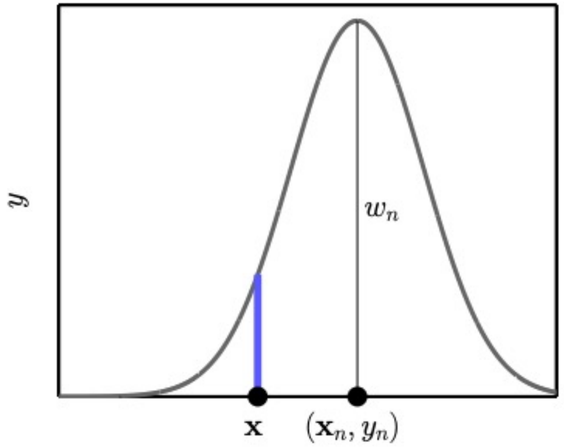
Changing the Viewpoints

- ϕ centered around $\vec{x}$



$$g(\vec{x}) = \sum_{n=1}^{N} \frac{y_n}{Z(\vec{x})} \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$$

- ϕ centered around $\vec{x}_n$



$$g(\vec{x}) = \sum_{n=1}^{N} w_n(\vec{x})\, \phi\left(\frac{\|\vec{x} - \vec{x}_n\|}{r}\right)$$

From Nonparametric to Parametric RBF

- Nonparametric RBF
  - $g(\vec{x}) = \sum_{n=1}^{N} \frac{y_n}{Z(\vec{x})} \phi\left(\frac{\|\vec{x}-\vec{x}_n\|}{r}\right)$
  - $g(\vec{x}) = \sum_{n=1}^{N} w_n(\vec{x})\, \phi\left(\frac{\|\vec{x}-\vec{x}_n\|}{r}\right)$
  - The hypothesis is defined by dataset

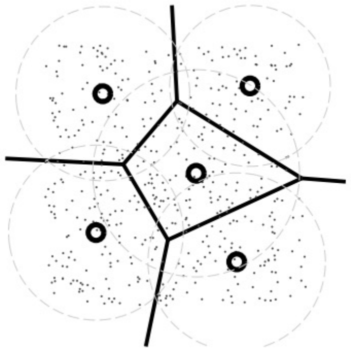
- Parametric RBF hypothesis set
  - $h(\vec{x}) = \sum_{n=1}^{N} w_n\, \phi\left(\frac{\|\vec{x}-\vec{x}_n\|}{r}\right)$
  - Learn w_n from data

Parametric RBF => Linear Models

- Parametric RBF is linear model with nonlinear transformation
  - $h(\vec{x}) = \sum_{n=1}^{N} w_n \, \phi\left(\frac{\|\vec{x}-\vec{x}_n\|}{r}\right)$
  - The projection $\Phi(\vec{x}) = \begin{bmatrix} \phi\left(\frac{\|\vec{x}-\vec{x}_1\|}{r}\right) \\ \phi\left(\frac{\|\vec{x}-\vec{x}_2\|}{r}\right) \\ \vdots \\ \phi\left(\frac{\|\vec{x}-\vec{x}_N\|}{r}\right) \end{bmatrix}$
- We can apply what we learned in linear models to learn w_n
- However, this seems to be overfitting (N parameters for N points)

From N points to K points

- Use only K points $(\vec{\mu}_1, \ldots, \vec{\mu}_K)$
  - $h(\vec{x}) = \sum_{k=1}^{K} w_k \, \phi\left(\frac{\|\vec{x}-\vec{\mu}_k\|}{r}\right)$
- Which K points?
  - We can find K representative points
  - Use clustering algorithms, e.g., Lloyd algorithm as introduced earlier
    1. Randomly pick K points as centers
    2. Create the Voronoi regions as clusters
    3. Update the centers (calculating the mean)
    4. Update the region
    5. Repeat 3 and 4



More Discussion on RBF

- $h(\vec{x}) = \sum_{k=1}^{K} w_k \, \phi\left(\frac{\|\vec{x} - \vec{\mu}_k\|}{r}\right)$

- Connection to linear models
  - Parametric RBF is essentially linear model with nonlinear transformation

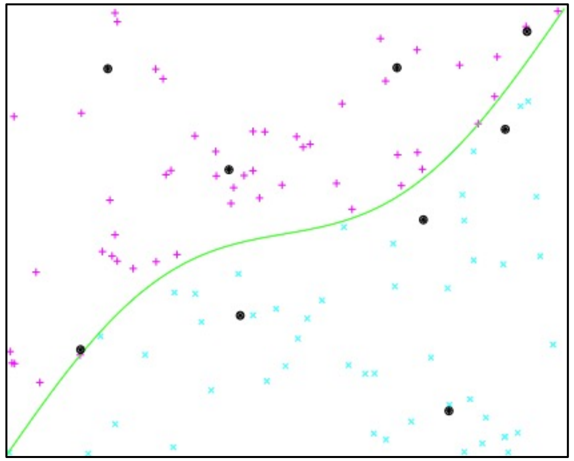
- Connection to nearest neighbor
  - Radial Basis Function is defined by “similarity
  - A prediction for a point is based on the “similarity” of the points to be predicted and other points

More Discussion on RBF

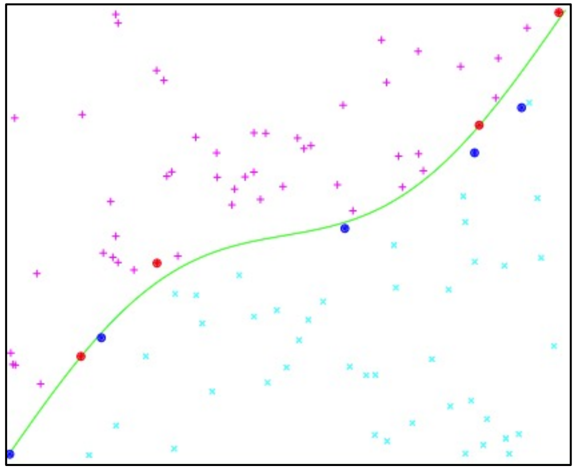
- $h(\vec{x}) = \sum_{k=1}^{K} w_k \, \phi\left(\frac{\|\vec{x}-\vec{\mu}_k\|}{r}\right)$

- Connection to SVM
  - Gaussian RBF Kernel: $g(\vec{x}) = sign\left(\sum_{\alpha_n^*>0} \alpha_n^* y_n K(\vec{x}_n, \vec{x}) + b^*)\right)$
  - Use cluster centers or support vectors to characterize a hypothesis

RBF



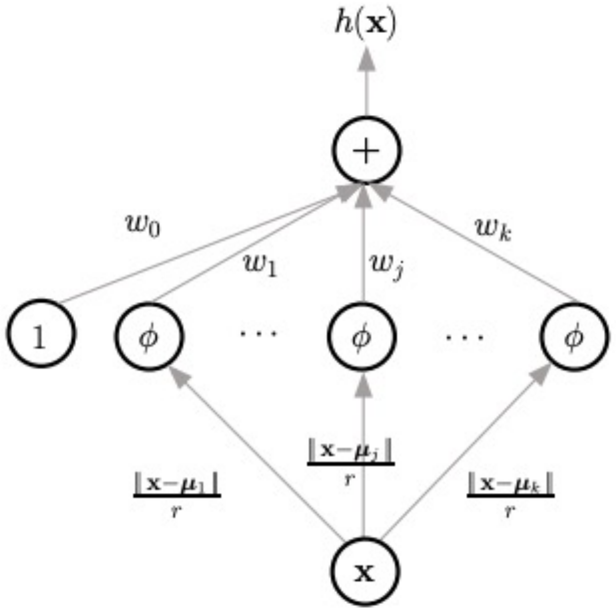
SVM



More Discussion on RBF

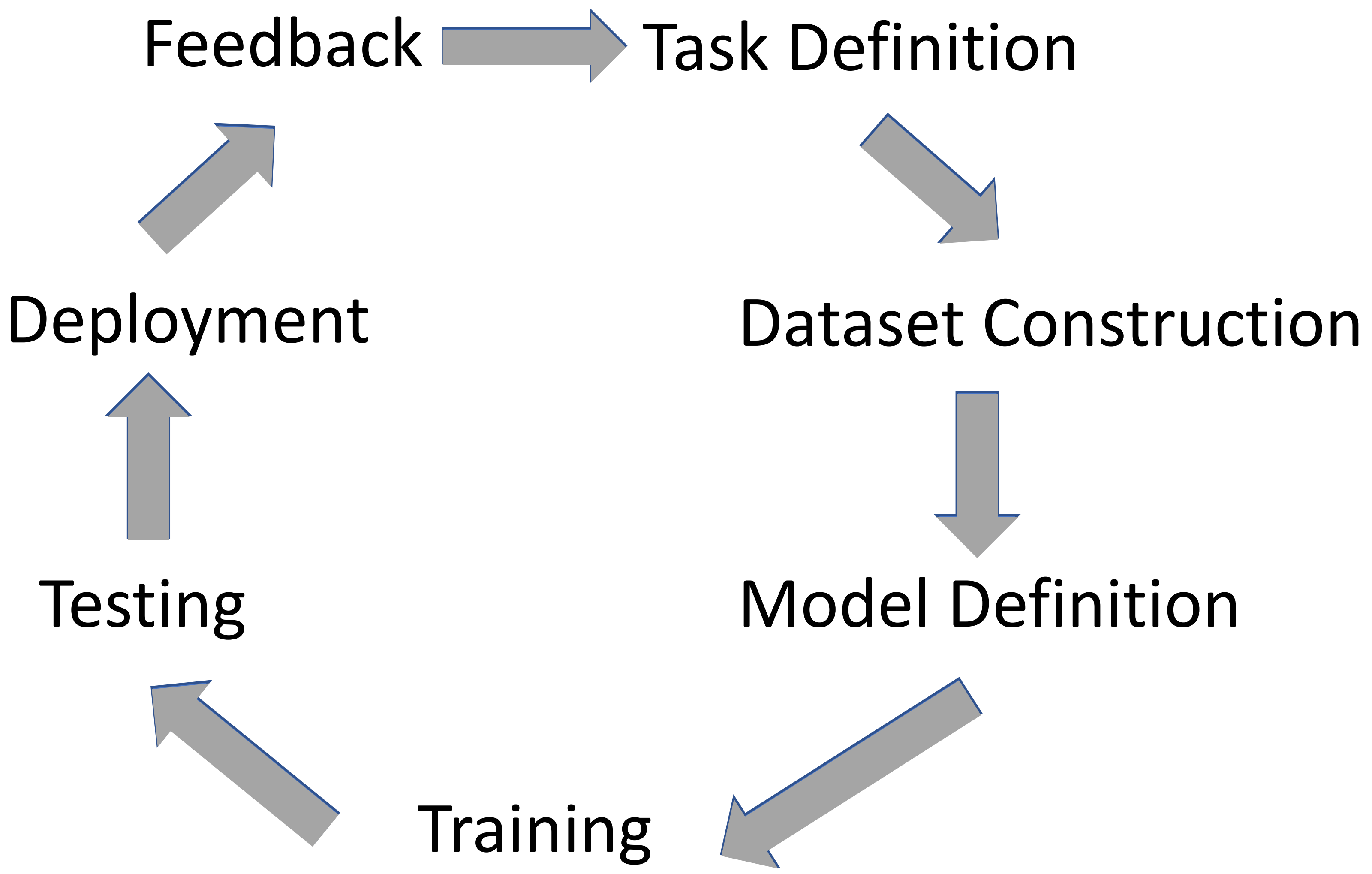
- $h(\vec{x}) = \sum_{k=1}^{K} w_k \, \phi\left(\frac{\|\vec{x}-\vec{\mu}_k\|}{r}\right)$

- Connection to Neural Network
  - RBF can be graphically presented:
  - (Similar to SVM, it's a single-hidden layer NN)

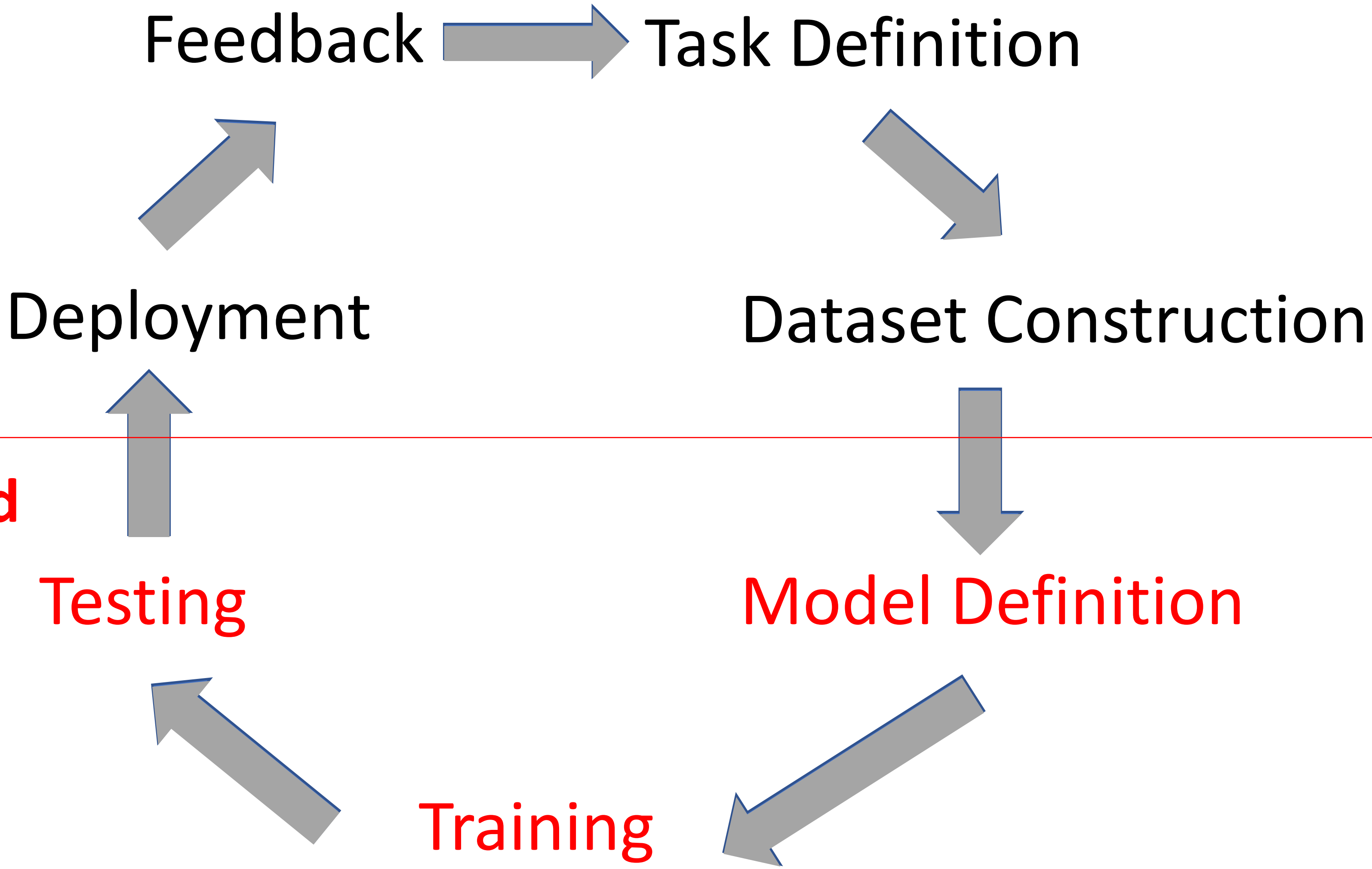


Discussion Beyond This Course

Machine Learning Lifecycle



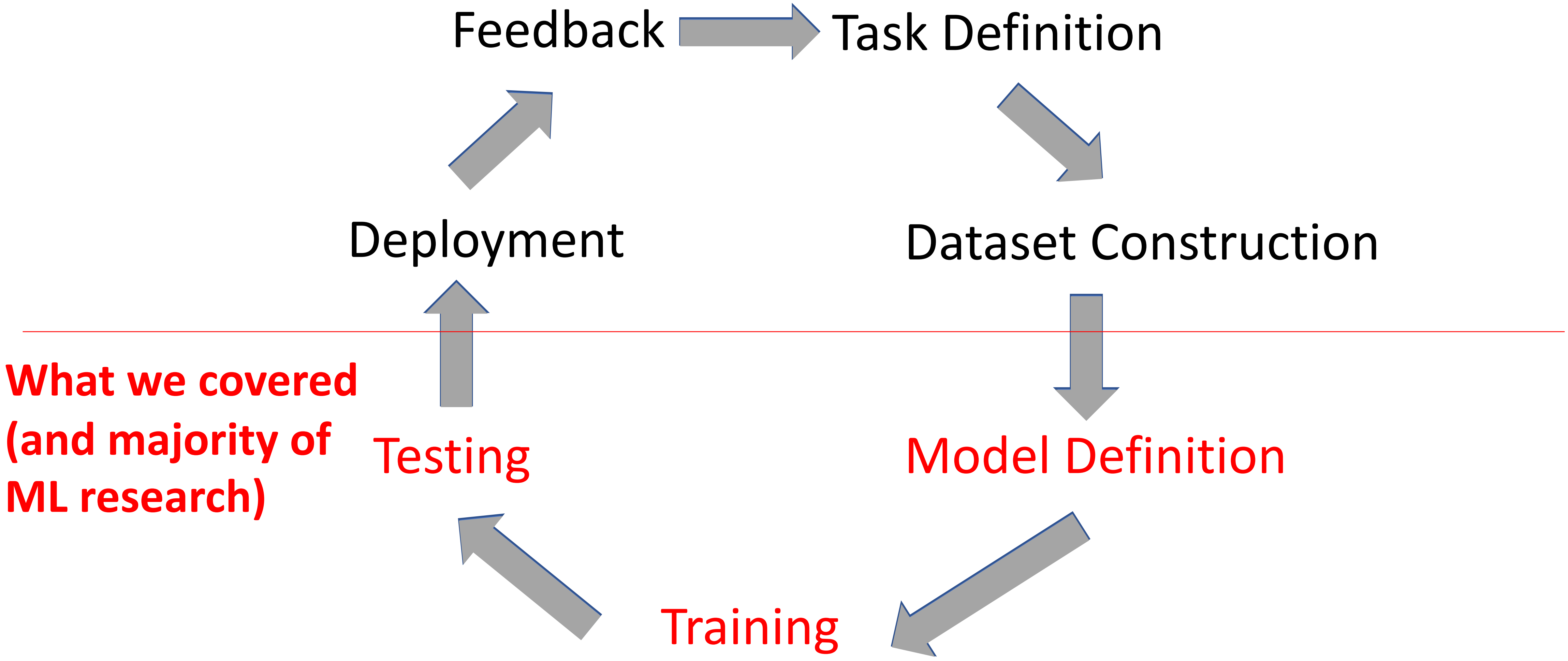
Machine Learning Lifecycle



What we covered (and majority of ML research)

Machine Learning Lifecycle

To have "positive" impacts, we need to be careful in every stage

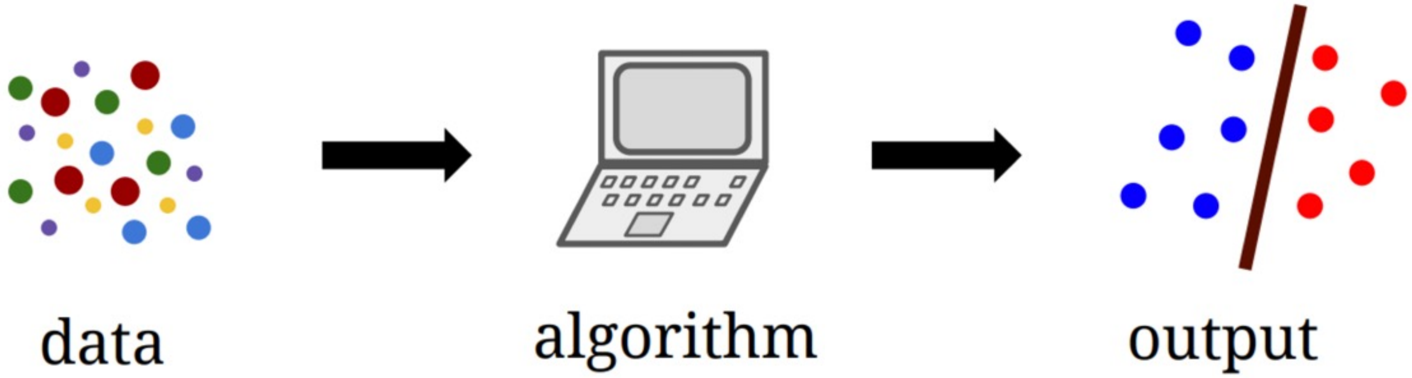


Strategic Classification

Or more recently, people start to call it “performative prediction”

Classification

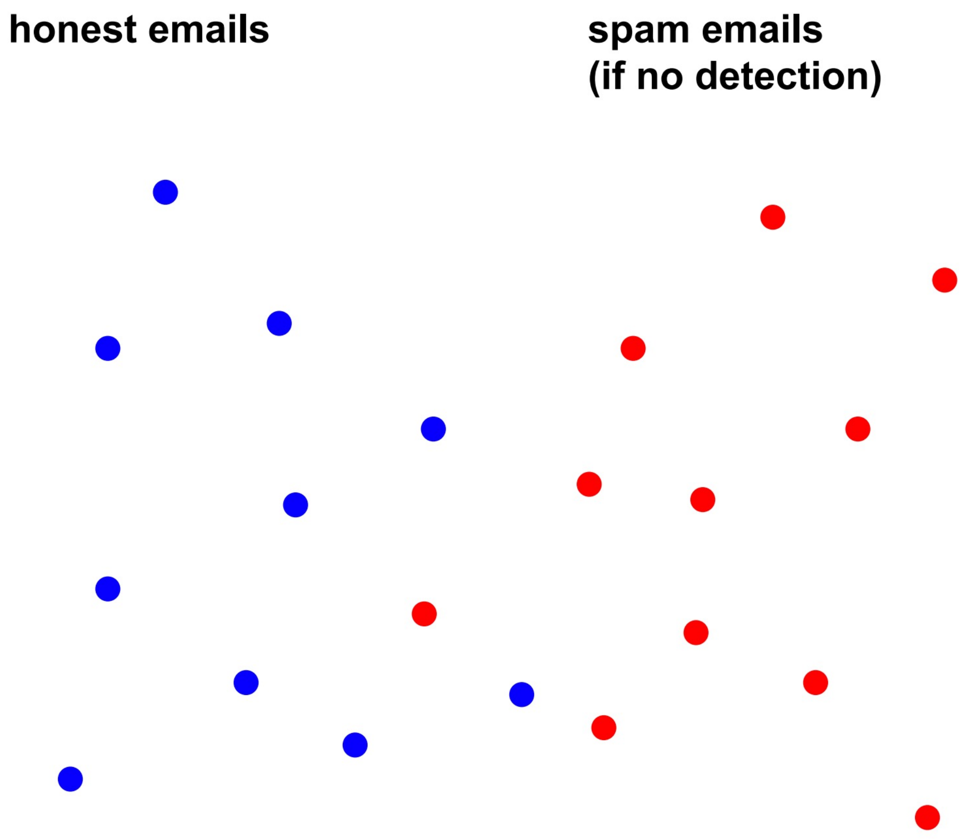
- Standard setup of (supervised) machine learning



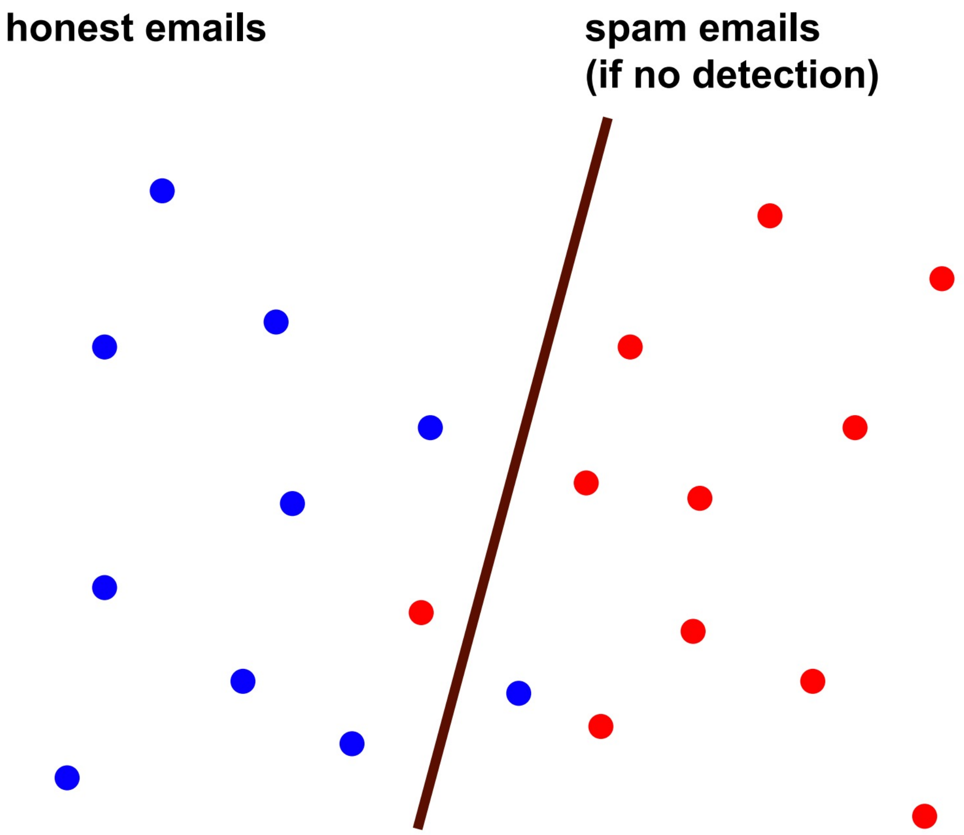
  - Finding patterns from the given training datasets
  - Use the pattern to make predictions on new testing data

- Fundamental assumption:
  - Training and testing data points are i.i.d. drawn from the same distribution

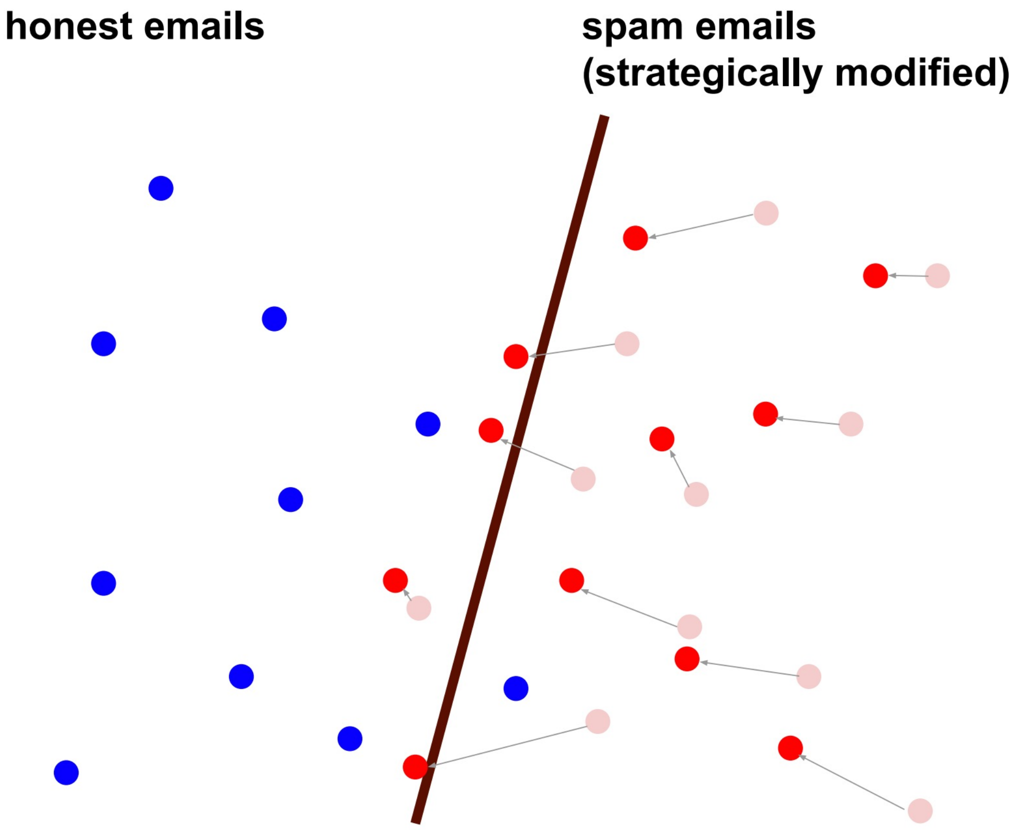
Example: Spam Filter



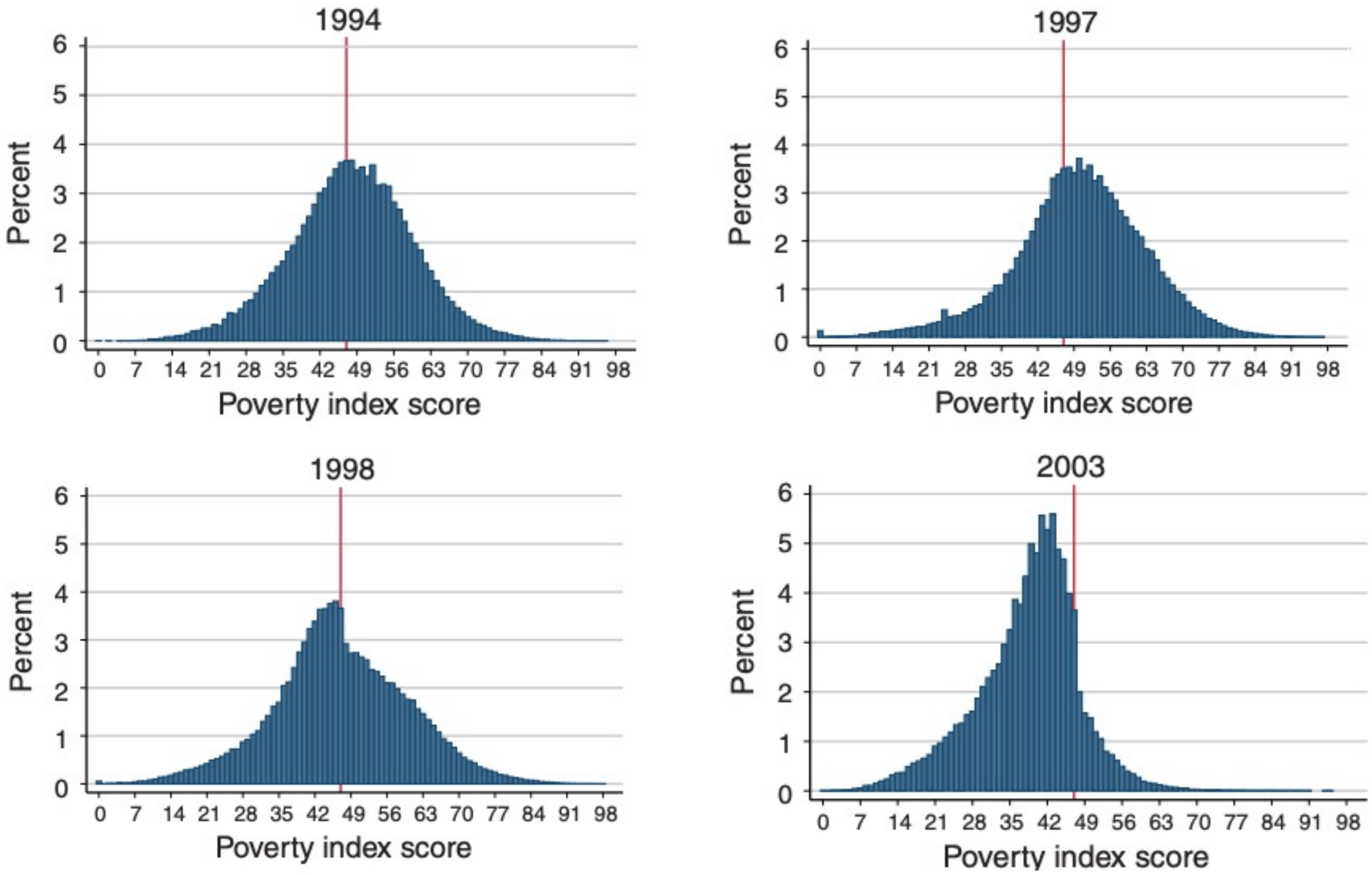
Example: Spam Filter



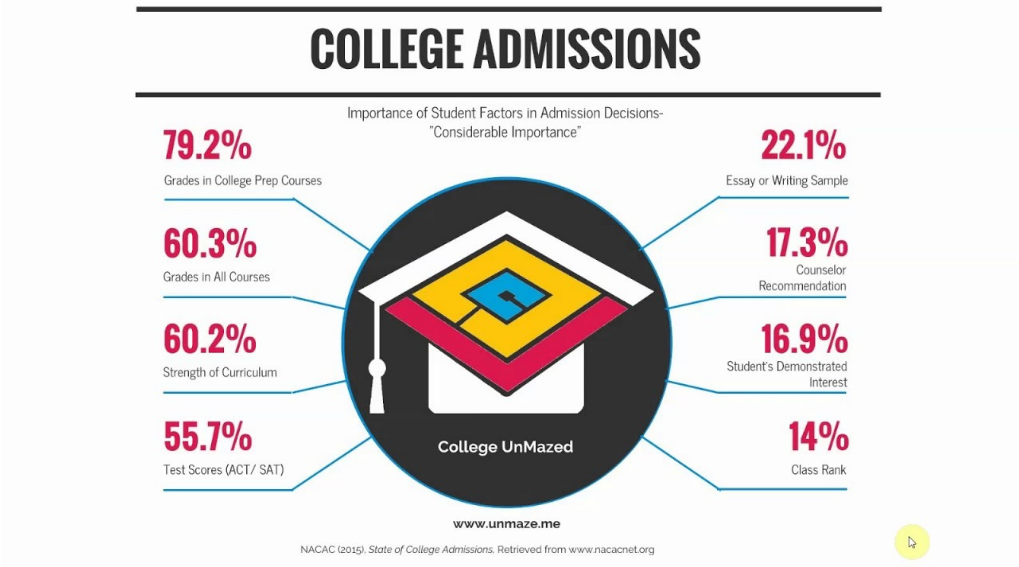
Example: Spam Filter

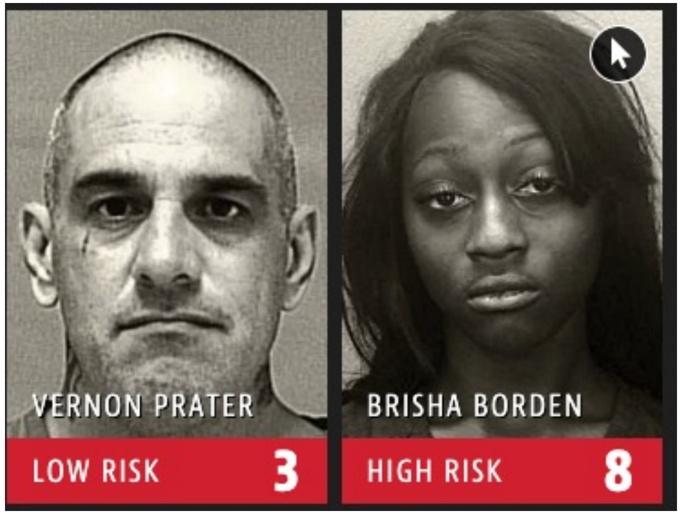


Social Program Eligibility [Camacho and Conover, 2012]

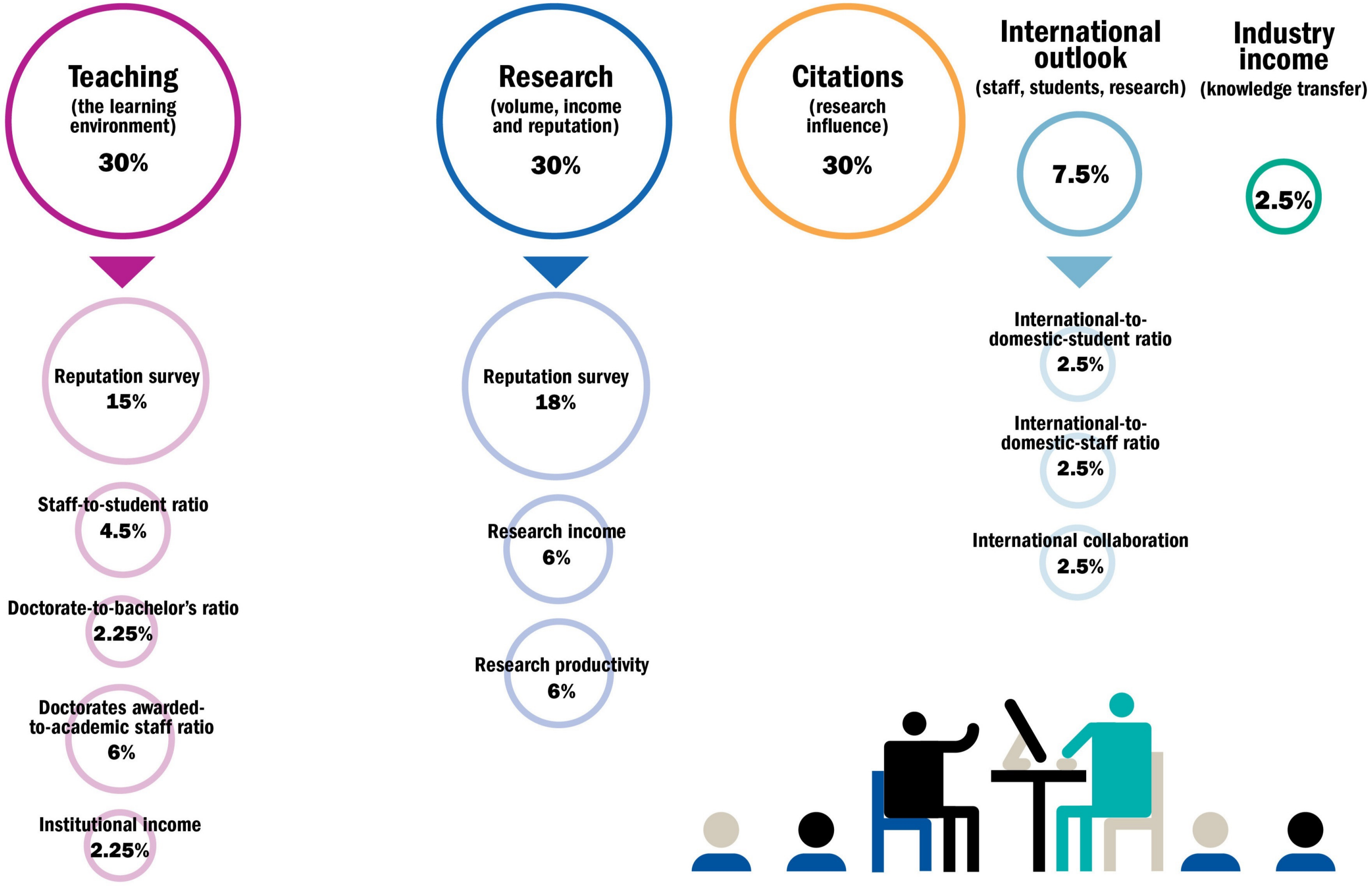


Goodhart’s law:
“If a measure becomes the public’s goal,
it is no longer a good measure.”

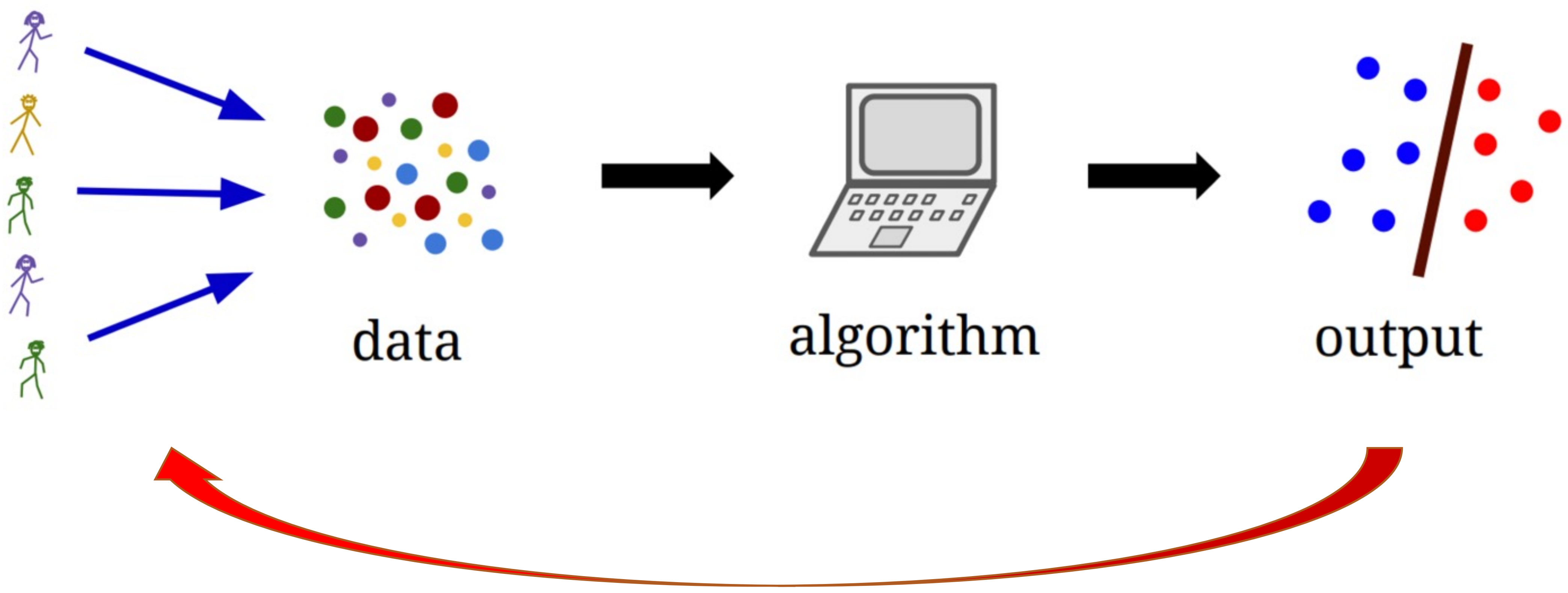




Methodology



Strategic Classification



How to take this interaction between ML algorithms and data-holders into account?

Game theoretical modeling

Game Theoretical Modeling

- Key elements:
  - Players, actions, payoffs

- Players: Jury (e.g., university) and Contestants (student applicants)
- Actions:
  - First, Jury decides on the machine learning model (binary classification)
  - Then, Contestant decides how to alter their features based on the model
- Payoffs
  - Jury wants to maximize the probability of **correct** predictions
  - Contestants want to be **selected** (being predicted as 1)