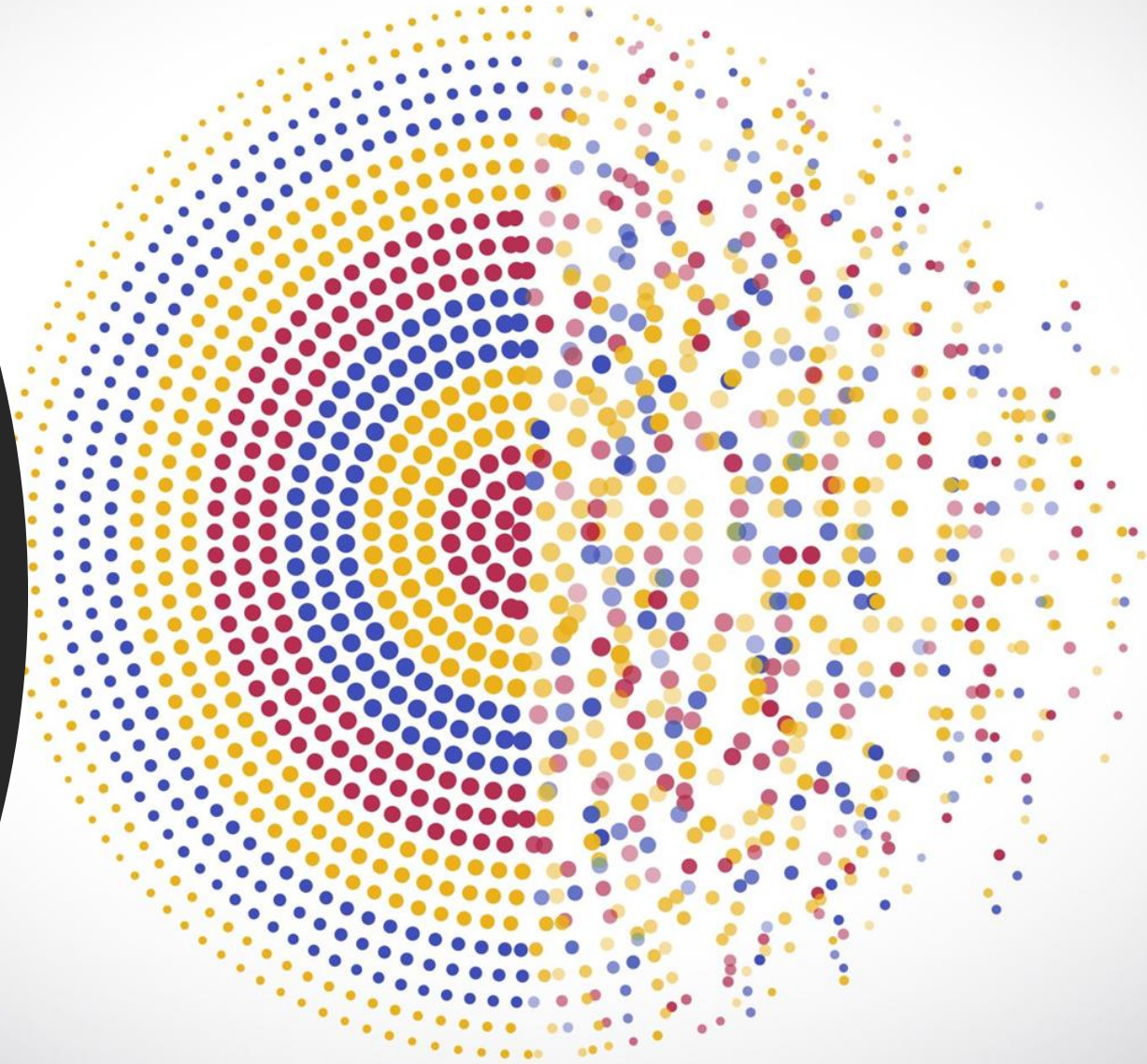
Supervised Learning

Dr. Mohamed AlHajri



Content

- Supervised Learning
- Linear Classification
    - Perceptron Algorithm
    - Linear Support Vector Machine (Linear SVM)
- Linear Regression
    - Least Square
    - Weighted Least Square
    - Ridge Regression
- Non-Linear Classification (Kernel)
    - Kernel Perceptron Algorithm
    - Kernel Support Vector Machine (Linear SVM)

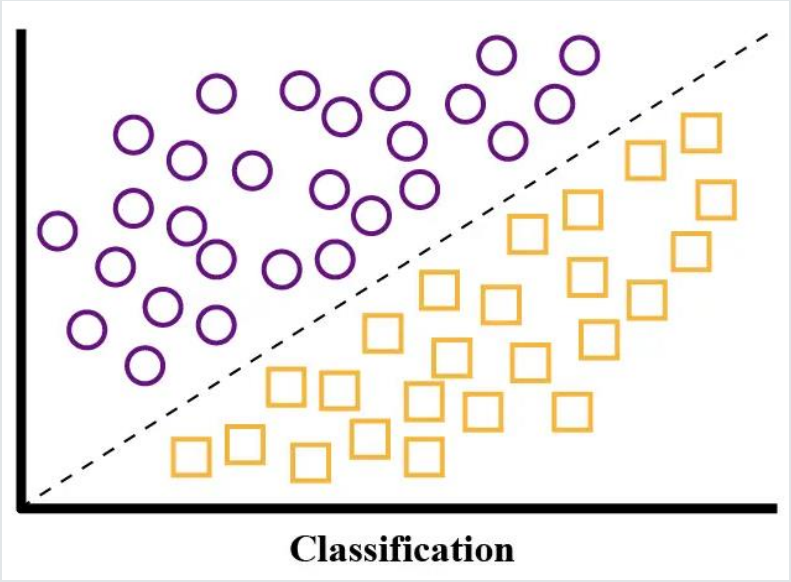
Content

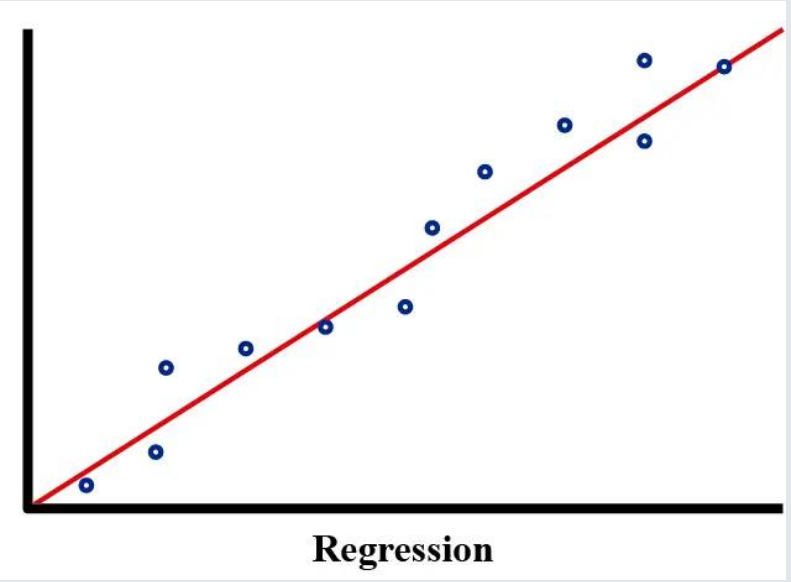
- Non-Linear Regression
    - Non-Linear Least Square
- K-nearest neighbor
- Decision Trees
- Feedforward Neural Network
- Convolutional Neural Network
- Recurrent Neural Network

Lecture 6 (Linear methods)

Supervised Learning

- Supervised learning involves learning a mapping function $f: X \rightarrow Y$ from input features $X \in \mathbb{R}^n$ to output labels Y, where:
  - Classification: Y is discrete (e.g., $Y \in \{1,2,\cdots,k\}$).
  - Regression: Y is continuous (e.g., $Y \in \mathbb{R}$).





https://panamahitek.com/en/what-is-the-difference-between-regression-and-classification-in-machine-learning/

Linear Classification

- **Linear separability** refers to the ability to separate two classes of data points in a feature space using a single straight hyperplane. Mathematically, a dataset is said to be linearly separable if there exists a hyperplane such that all data points belonging to one class are on one side of the hyperplane, while all data points belonging to the other class are on the opposite side.
- In $\mathbb{R}^n$, the hyperplane can be defined as:

$$w.x + b = 0$$

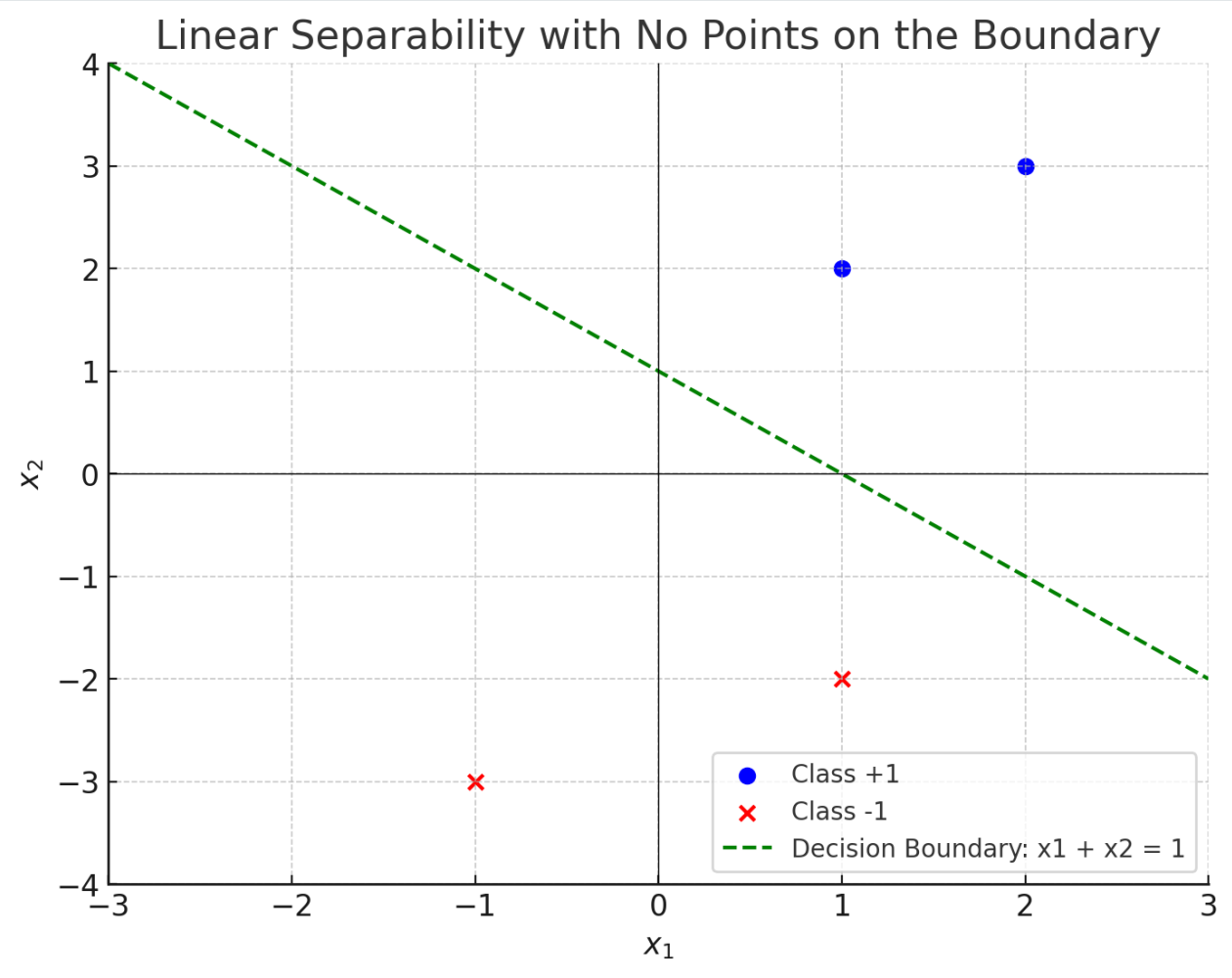
Where:

- w is the weight vector (defining the orientation of the hyperplane),
- x is the input feature vector, and
- b is the bias term (defining the position of the hyperplane).

For a dataset to be linearly separable, there must exist a weight vector w and a bias b such that:

$$y_i(w \cdot x_i + b) > 0 \; \forall i$$

Linear Classification



- Hyperplane

$$w_1 x_1 + w_2 x_2 + b = 0$$

$$x_1 + x_2 - 1 = 0$$

- Datapoints

$$(1,1)\ ;\ w.x + b = 2 > 0\ (+1)$$

$$(2,2)\ ;\ w.x + b = 4 > 0\ (+1)$$

$$(1,-2)\ ;\ w.x + b = -3 < 0\ (-1)$$

$$(-1,-3)\ ;\ w.x + b = -5 < 0\ (-1)$$

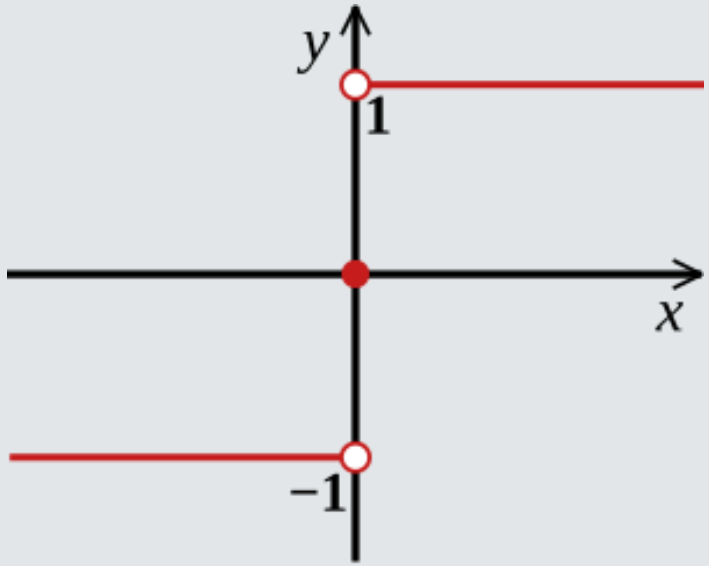
Linear Classification

- **Perceptron Algorithm:** foundational algorithm for binary classification, aiming to find a linear decision boundary that separates two classes. It updates its weights iteratively based on misclassifications.

$$f(x) = sign(w.x + b)$$

Where

- $w \in \mathbb{R}^n$ is the weight vector
- $b \in \mathbb{R}$ is the bias term



Linear Classification

- **Perceptron Algorithm Update Rule:** When a data point (x_i, y_i) is misclassified, update the weights and bias:

$$w \leftarrow w + \eta y_i x_i$$
$$b \leftarrow b + \eta y_i$$

where η is the learning rate

How did we derive this update?

Linear Classification

- **Gradient Descent Interpretation:** The perceptron update can be viewed as a form of gradient descent on the hinge loss:

$$L(w, b) = \max(0, -y_i(w \cdot x_i + b))$$

- **Partial Derivatives:**

$$\frac{\partial L}{\partial w} = \begin{cases} -y_i x_i \; ; if \; y_i(w.x_i + b) \leq 0 \\ 0 \; ; otherwise \end{cases}$$

$$\frac{\partial L}{\partial b} = \begin{cases} -y_i \; ; if \; y_i(w.x_i + b) \leq 0 \\ 0 \; ; otherwise \end{cases}$$

- **Weight Update as Gradient Step**:

$$w \leftarrow w \; - \eta \frac{\partial L}{\partial w} = w + \eta y_i x_i$$

$$b \leftarrow \; b \; - \eta \frac{\partial L}{\partial b} = b + \eta y_i$$

Linear Classification

- **Pseudo Code:**
- Inputs:

X: Training data matrix of shape (m, n) ; y: Labels vector of shape (m,), with values in {-1, +1}

η: Initial learning rate ; MaxIter: Maximum number of iterations ; LearningRateSchedule

- w = zeros(n); b = 0

For t = 1 to MaxIter:

For each (x_i, y_i) in (X, y):

if y_i * (w · x_i + b) <= 0:

w = w + η * y_i * x_i

b = b + η * y_i

η = LearningRateSchedule(η, t)

Output: weight vector w, bias b

Linear Classification

- **Numerical Example:** Consider 2D dataset

Sample	x_1	x_2	y
1	2	3	+1
2	1	1	-1
3	2	1	-1
4	3	3	+1
5	5	5	+1

$(2,3); w = [2,3], b = 1$
$(1,1); w = [1,2], b = 0$
$(2,1); w = [-1,1], b = -1$
$(3,3); w = [2,4], b = 0$
$(5,5)$; no update
$(2,3)$; no update
$(1,1); w = [1,3], b = -1$

$(2,1); w = [-1,2], b = -2$(Answer)

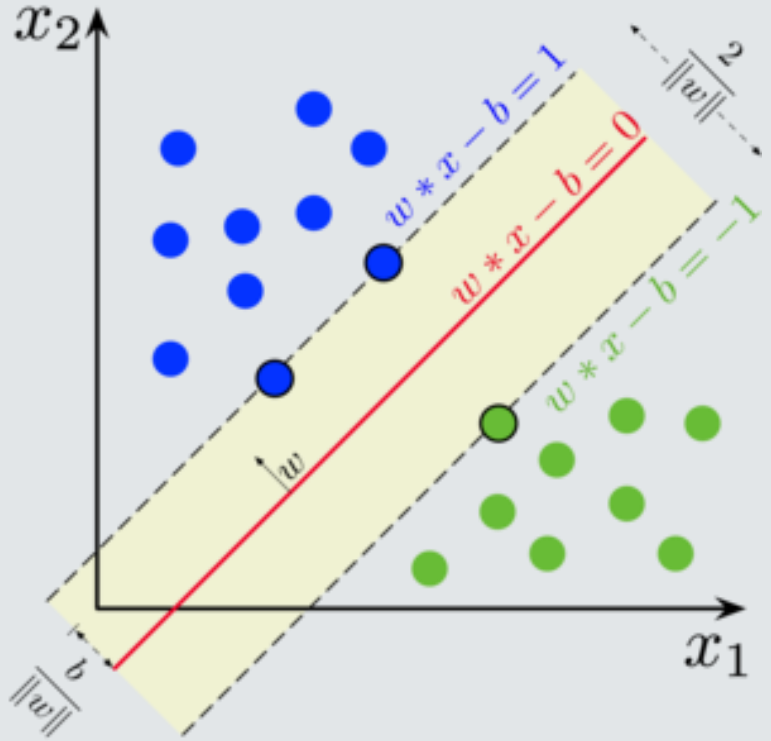
Is it unique?

Linear Classification

- **Limitations of the Perceptron:**
- Linearly Separable Data
  - Requirement: The perceptron algorithm converges only if the data is linearly separable.
  - Non-linear Separability: Fails on datasets like XOR where no linear hyperplane can separate the classes (Infinite Iterations)
- No Margin Optimization
  - No Maximum Margin: The perceptron does not seek the hyperplane with the largest margin, potentially leading to poor generalization.

Linear Classification (SVM)

- **Support Vector Machine (SVM):** aims to find the maximum margin hyperplane, which separates two classes with the largest possible margin. This ensures better generalization to unseen data.



https://en.wikipedia.org/wiki/Support_vector_machine

Linear Classification (SVM)

- For linear classification, the decision boundary is:

$$w \cdot x + b = 0$$

- The margin is defined as the distance between the decision boundary and the closest data points from either class. The goal is to maximize this distance.
- For correctly classified points, we have:

$$y_i(w \cdot x_i + b) \geq 1;\ \forall i$$

- This inequality ensures that points from class +1lie on one side of the hyperplane, and points from class −1lie on the other side, separated by a margin of at least 1 unit.
- The **margin** is given by $\frac{2}{\|w\|}$, where $\| w \|$ is the Euclidean norm of the weight vector w. To maximize the margin, we minimize $\| w \|$, or equivalently, minimize $\frac{\|w\|^2}{2}$.

Linear Classification (SVM)

- For linear classification, the decision boundary is:

$$w \cdot x + b = 0$$

- The margin is defined as the distance between the decision boundary and the closest data points from either class. The goal is to maximize this distance.
- For correctly classified points, we have:

$$y_i(w \cdot x_i + b) \geq 1;\ \forall i$$

- This inequality ensures that points from class +1lie on one side of the hyperplane, and points from class −1lie on the other side, separated by a margin of at least 1 unit.
- The **margin** is given by $\frac{2}{\|w\|}$, where $\| w \|$ is the Euclidean norm of the weight vector w. To maximize the margin, we minimize $\| w \|$, or equivalently, minimize $\frac{\|w\|^2}{2}$.

Linear Classification (SVM)

- The optimization problem is formulated as:

$$\min_{w,b} \frac{||w||^2}{2}$$

subject to the constraints

$$y_i(w.x_i + b) \geq 1 \; ; \forall i$$

- We solve this constrained optimization problem using Lagrange multiplers. We introduce a lagrange multipler $\alpha_i \geq 0$ for each constraint $y_i(w \cdot x_i + b) \geq 1$, and the Lagrangian is defined as:

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i (y_i(w \cdot x_i + b) - 1)$$

Linear Classification (SVM)

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i (y_i (w \cdot x_i + b) - 1)$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

If we replace the w with $\sum_{i=1}^{n} \alpha_i y_i x_i$

The **optimization problem** becomes

$$\max_{\alpha} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i x_j) + \sum_{i=1}^{n} \alpha_i$$

$$s.t. \sum_{i=1}^{n} \alpha_i \, y_i = 0, \alpha_i \geq 0 \; \forall i$$

Linear Classification (SVM) [Dual]

Extra Notes (Beyond the scope of the course):

To solve the following problem there are different optimization algorithms that can be used to solve it (Interior Point Methods, Sequential Minimal Optimization (SMO)). In addition

Why $\min||w||^2$?

Linear Classification (SVM)

- The margin is the **perpendicular distance** between the decision boundary (hyperplane) and the closest data points.
- The distance d from a point x_0 to the hyperplane can be calculated using the formula for the distance between a point and a plane:

$$d = \frac{|w.x_0 + b|}{||w||}$$

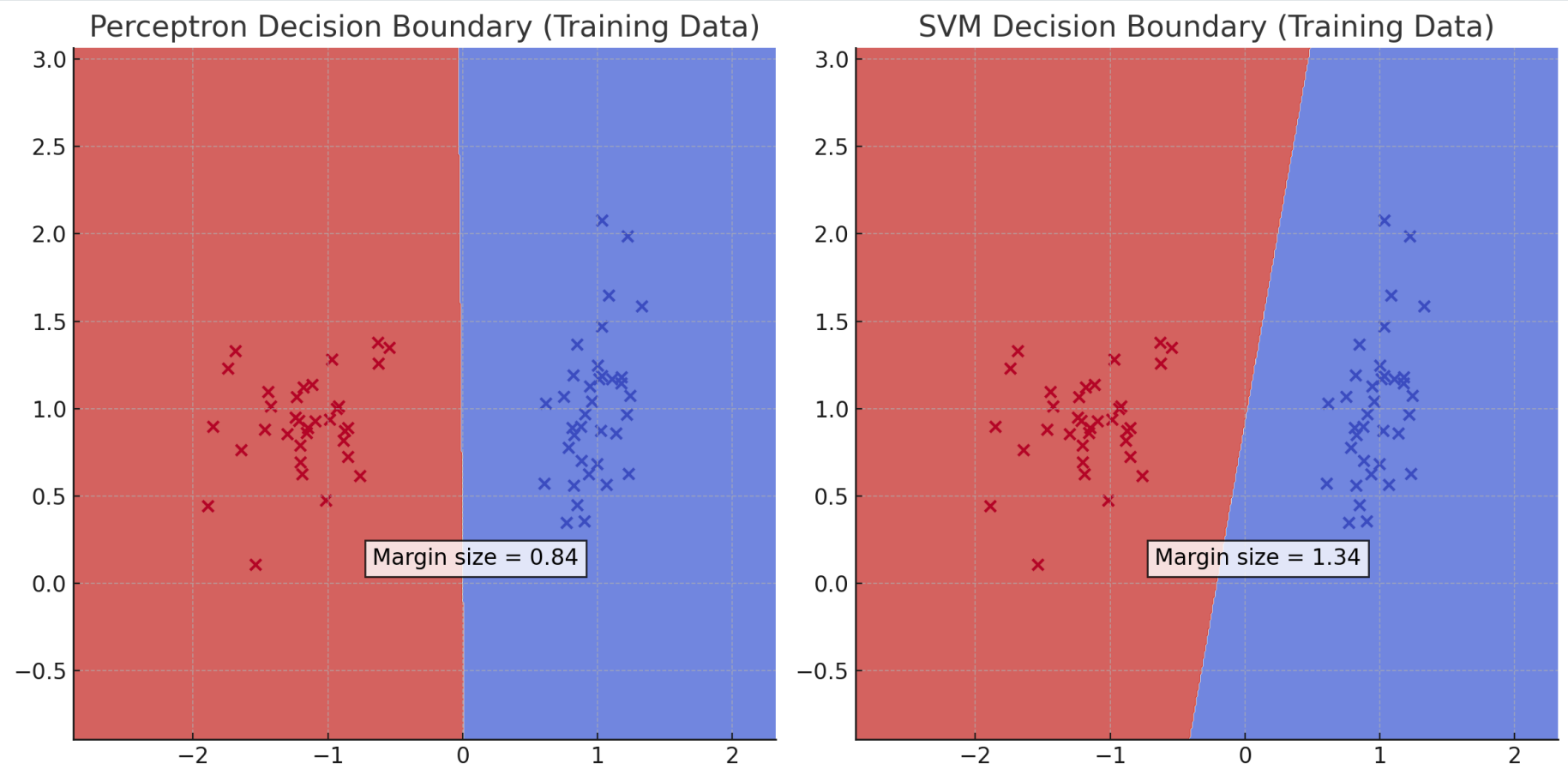
This formula gives the **perpendicular distance** from the point x_0 to the hyperplane.

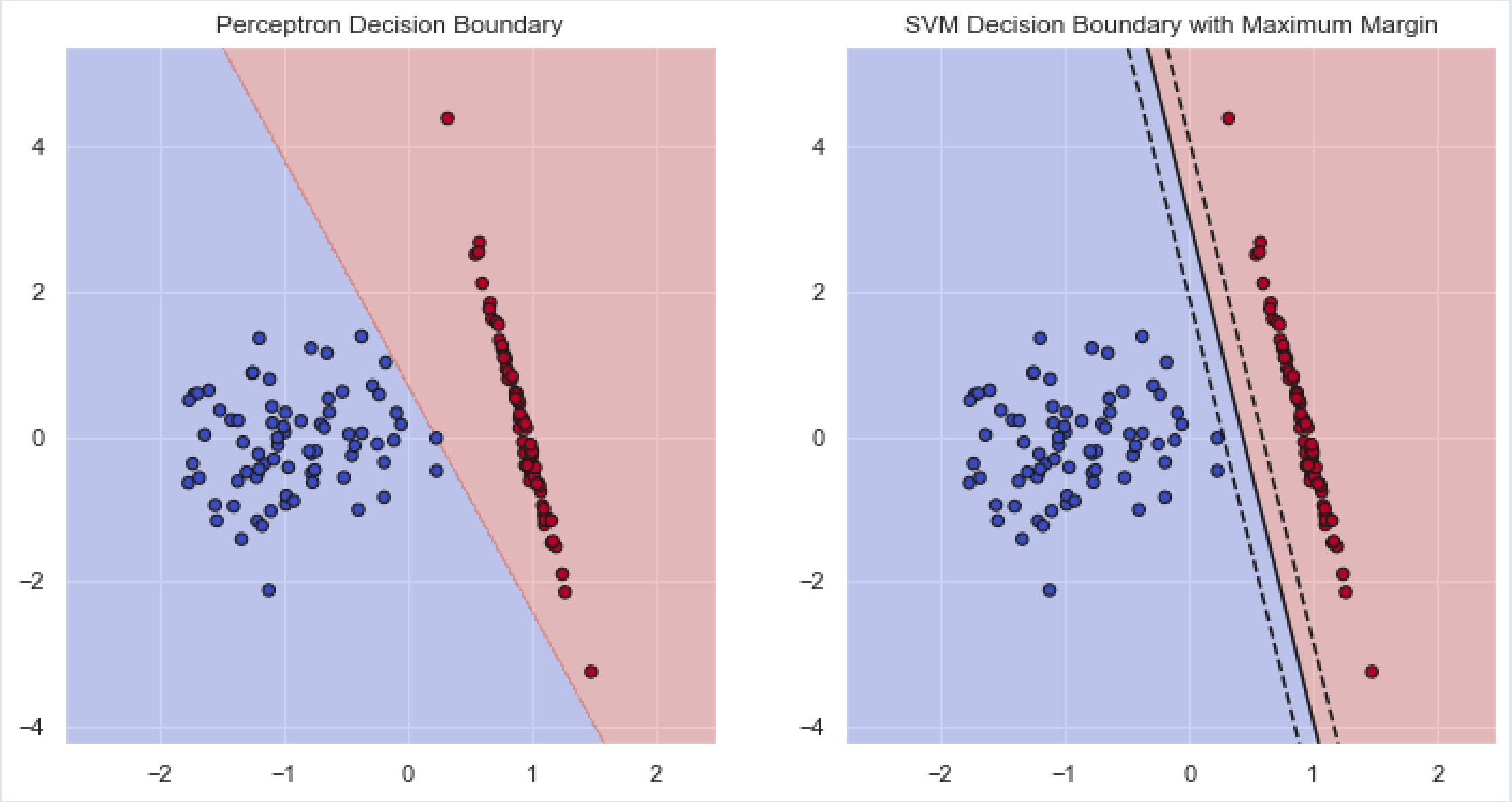
- The decision boundary $w.x + b = 0$
- The hyperplanes that pass through the support vectros, given by $w.x + b = \pm 1$
- The distance between these two hyperplanes is the margin. $\frac{2}{||w||}$

Thus, the margin is inversely proportional to $||w||$, meaning the larger $||w||$, the smaller the margin and vice versa. We square it since it would be easier to minimize since it is quadratic that is differentiable.

Linear Classification (SVM)

- **Limitations of SVM:**
- Computational Complexity


Perceptron Decision Boundary (Training Data)
Margin size = 0.84
SVM Decision Boundary (Training Data)
Margin size = 1.34
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−2
−1
0
1
2



Experiment 1

Linear Regression

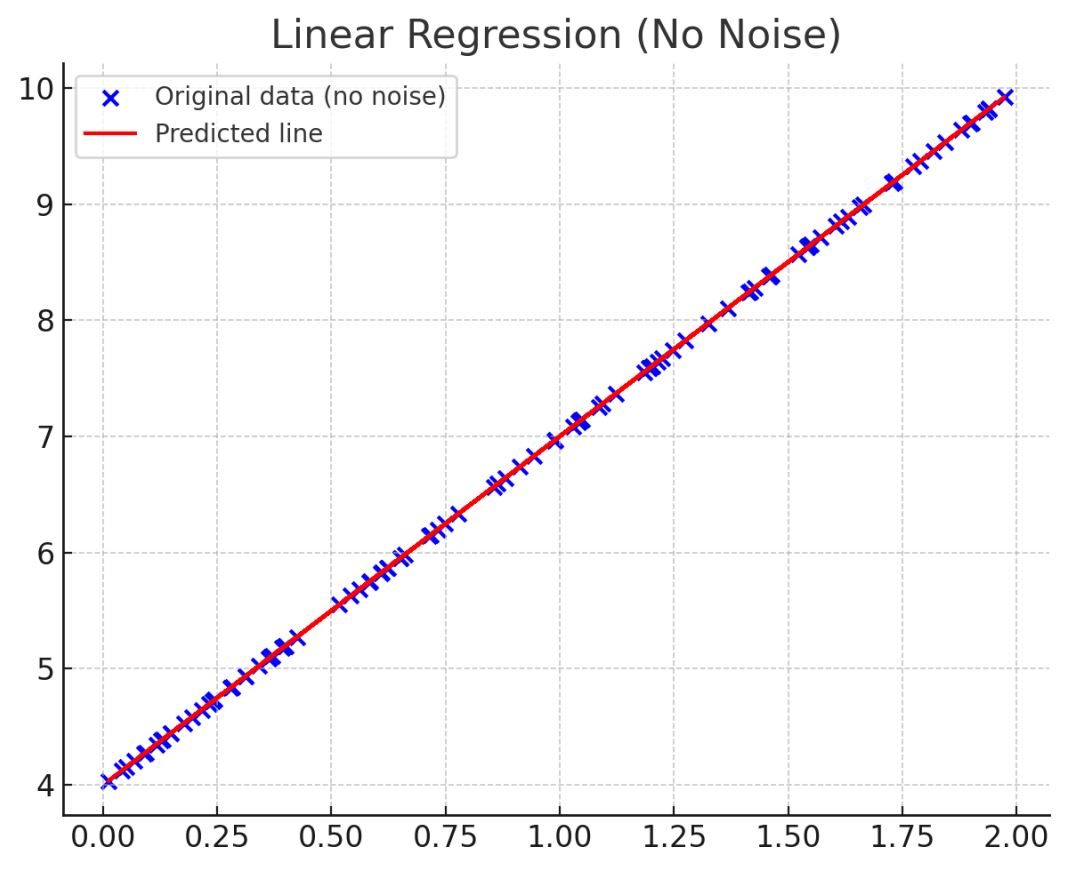
- Linear regression models the relationship between the input features $X \in R^{n \times p}$ and the target values $y \in \mathbb{R}^n$ as:

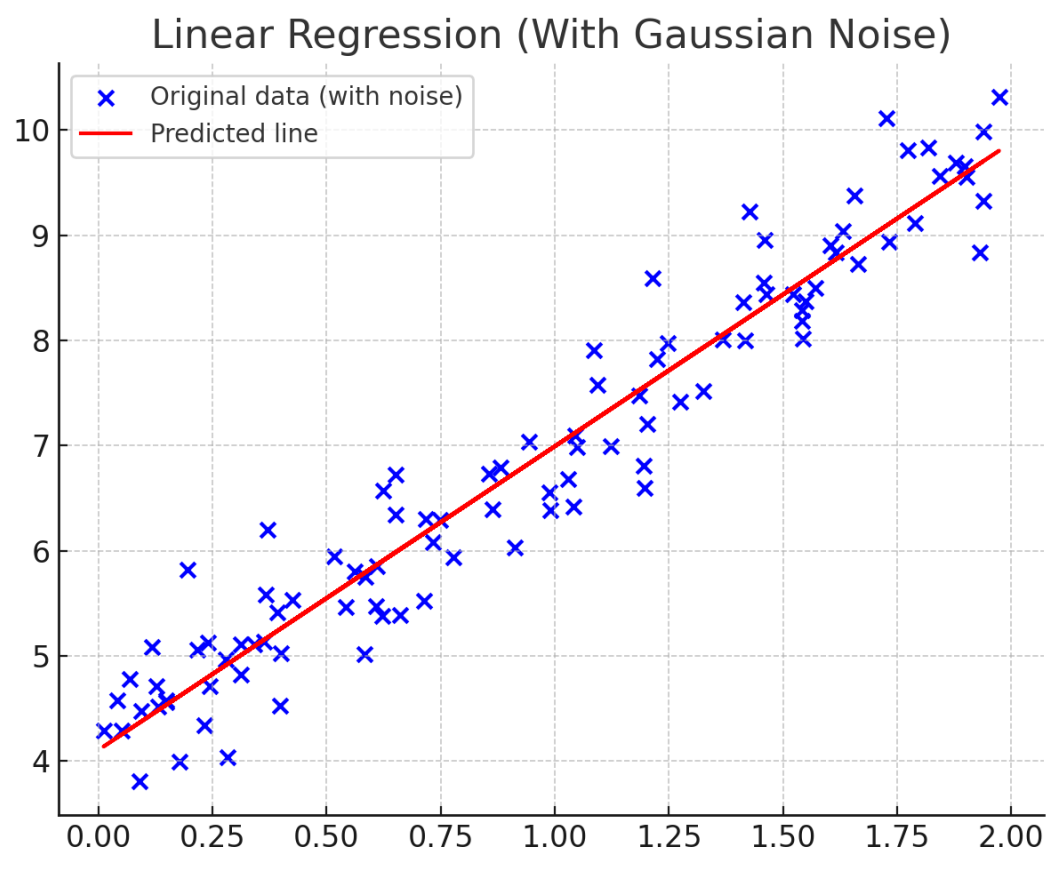
$$y = X\beta + \epsilon$$

- X is the design matrix of input features.
- β is the vector of coefficients.
- ϵ is the error term.
- To estimate β, we minimize the sum of squared residuals:

$$\min_{\beta} \left|\left|y - X\beta\right|\right|_2^2 = \min_{\beta} \sum_{i=1}^{n} (y_i - X_i\beta)^2$$

Linear Regression





Linear Regression

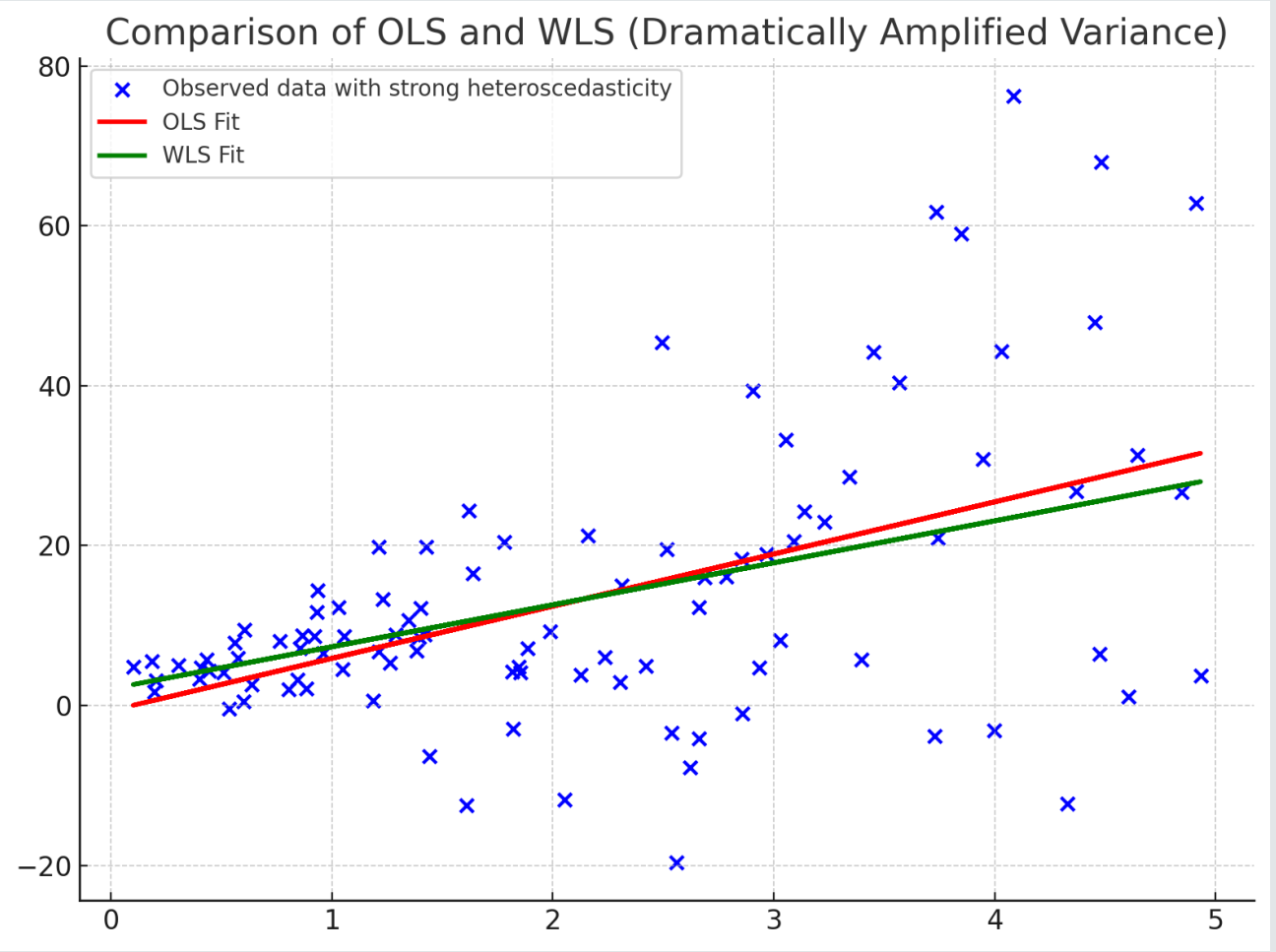
- Limitations:
- Linear relationship
- Constant Variance
- Singularity in $(X^T X)^{-1}$

Linear Regression

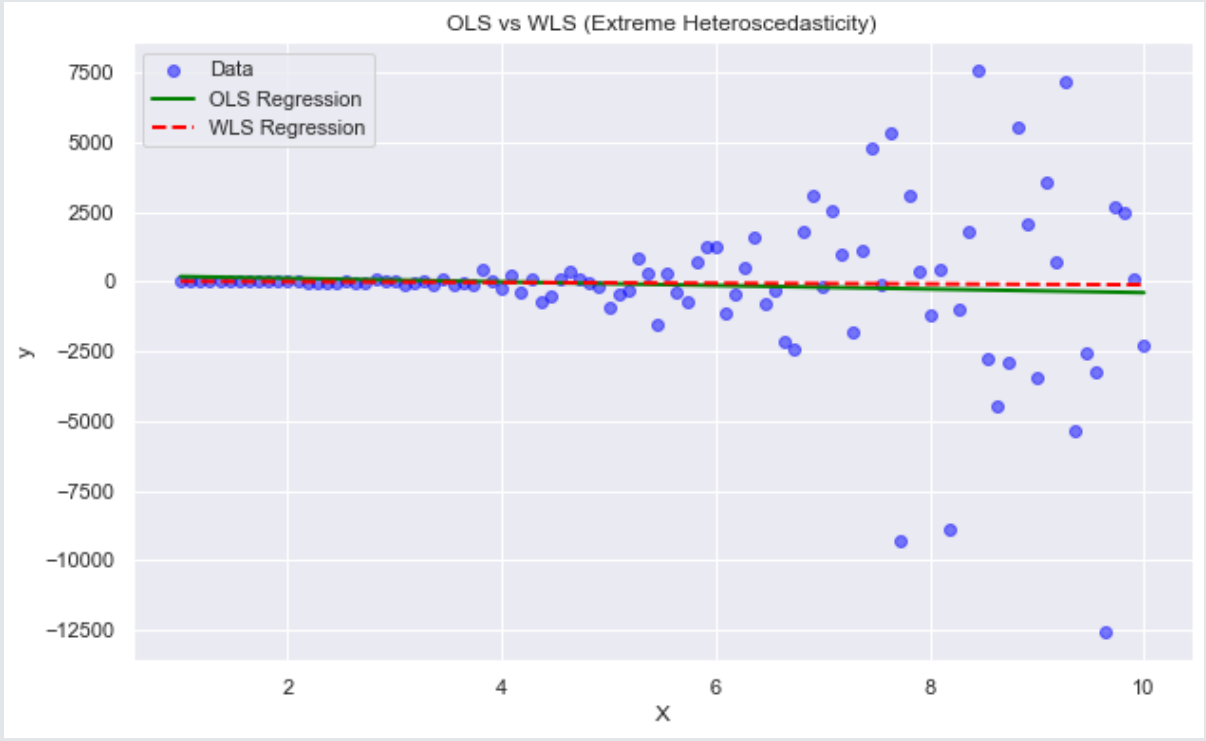
- **Weighted Least Square (WLS)**
- In WLS, we assume that the observations have **non-constant variance**, meaning the errors associated with different observations are not the same. The goal is to account for this heteroscedasticity by giving different weights to different observations.
- The idea is to minimize a weighted sum of the squared residuals, where observations with **higher variance** are given **less weight**, and observations with **lower variance** are given **more weight**.

Linear Regression

- **Weighted Least Square**



Linear Regression



Experiment 2

Linear Regression

- **Ridge Regression**
- In **Ordinary Least Squares (OLS)**, we aim to minimize the residual sum of squares:

$$\min_{\beta} \left|\left|y - X\beta\right|\right|_2^2$$

- However, when the matrix XX^T is **singular** or **nearly singular** (often caused by **multicollinearity**), the inverse $(X^TX)^{-1}$doesn't exist or becomes unstable. This instability makes OLS unreliable for certain datasets.
- To overcome this, **Ridge Regression** introduces a regularization term that penalizes large values of β, effectively **shrinking** the coefficients and stabilizing the solution. This regularization is key to handling multicollinearity and singularity issues.

$$\min_{\beta} \left|\left|y - X\beta\right|\right|_2^2 + \lambda \left|\left|\beta\right|\right|_2^2$$

- $\lambda \geq 0$ is the **regularization parameter**. It controls the amount of regularization applied:
  - $\boldsymbol{\lambda = 0}$ *gives the OLS solution,*
  - $\boldsymbol{\lambda > 0}$ *penalizes large coefficients, reducing overfitting and addressing multicollinearity.*

Linear Regression (Extra Reading)

- In **Lasso Regression,** introduces an L1 regularization term, which penalizes the absolute value of the coefficients. The objective function is:

$$\min_{\beta} \left|\left|y - X\beta\right|\right|_2^2 + \lambda \left|\left|\beta\right|\right|_1$$

Pros	Cons
Feature Selection: Lasso performs automatic feature selection by driving some coefficients to exactly zero, making the model sparse and more interpretable.	**Instability with Correlated Features**: Lasso tends to arbitrarily select one feature from a group of highly correlated variables, which may result in unstable solutions.
Good for High-Dimensional Data: Lasso works well when there are many features, especially when some are irrelevant, as it can exclude irrelevant features.	**Bias**: Because of the L1 penalty, Lasso may shrink important variables more than necessary, introducing bias into the estimates.
Improved Interpretability: By selecting a subset of features, Lasso creates a simpler, more interpretable model.	**No Closed-Form Solution**: Lasso requires iterative algorithms (like Coordinate Descent, or IRLS) to compute the solution, which can be computationally expensive for very large datasets.

Linear Regression (Extra Reading)

- **Elastic Net Regression,** is a hybrid of Ridge and Lasso that combines both L1 and L2 penalties. The objective function is:

$$\min_{\beta} \left|\left|y - X\beta\right|\right|_2^2 + \lambda_1 \left|\left|\beta\right|\right|_2^2 + \lambda_2 \left|\left|\beta\right|\right|_1$$

Pros	Cons
Combines the Strengths of Ridge and Lasso: Elastic Net combines the feature selection ability of Lasso with the stability of Ridge, making it useful when dealing with highly correlated features.	**More Complex Tuning**: Elastic Net requires tuning two hyperparameters (λ_1 and λ_2), which adds complexity to the model selection process.
Stability with Correlated Features: Unlike Lasso, Elastic Net doesn't arbitrarily pick one feature from a group of correlated features; instead, it tends to include all or none of the correlated features.	**Can Over-Select Features**: In some cases, Elastic Net may include too many features, especially when λ_2 (the L2 penalty) dominates.
Flexible Regularization: The model can be tuned to prioritize either the L1 or L2 penalty, offering flexibility based on the dataset.	**Computationally Intensive**: Similar to Lasso, Elastic Net does not have a closed-form solution and requires iterative algorithms, which can be expensive for large datasets.

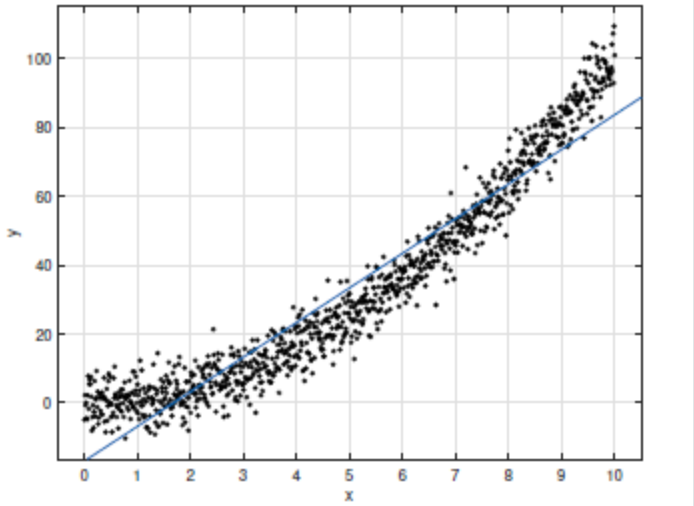
Linear Regression (Extra Reading)

	Ridge	Lasso	Elastic Net
Penalty	L2 (Sum of squared coefficients)	L1 (Sum of absolute coefficients)	L1 + L2 (Combination of both)
Feature Selection	No	Yes (Sparse solution)	Yes (Incorporates groups of correlated features)
Handling Multicollinearity	Reduces multicollinearity, but keeps all features	May arbitrarily select one feature from a correlated group	Handles correlated features better than Lasso
Bias-Variance Tradeoff	Lower bias, but all coefficients shrink	More bias, as L1 penalty shrinks important variables	Flexible; can balance bias-variance tradeoff
Interpretability	Lower	Higher (fewer features)	Moderate (depends on tuning)
Computational Complexity	Lower (closed-form solution)	Higher (iterative solution)	Higher (iterative solution with two penalties)

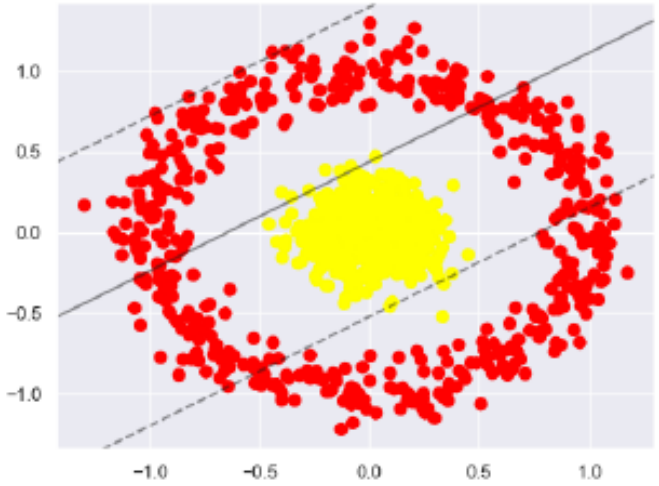
Lecture 7 (kernel methods)

Limitations of Linear Classification

- In the previous lecture, we have learnt of how to classify different datapoints by using a linear classifier (perceptron, linear svm). we have learnt of how to learn different relationships between variables using linear regression, or how to classify different datapoints by using a linear classifier (perceptron, linear svm).
- **What happens if the relationship between the variables is non-linear as shown in Fig. 1(a)?**
- **What happens if the different datapoins can not be separated by linear classifier as shown in Fig. 1(b)?**



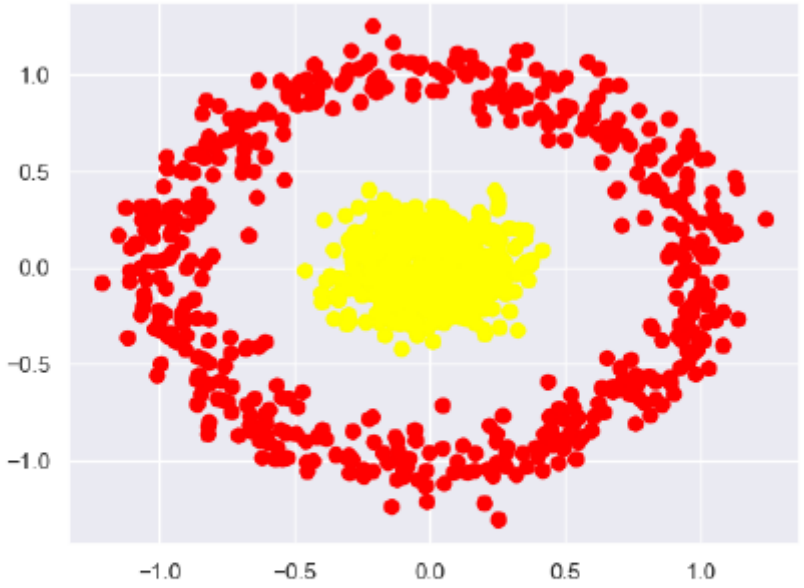
(a) Linear Regression on non-linear data



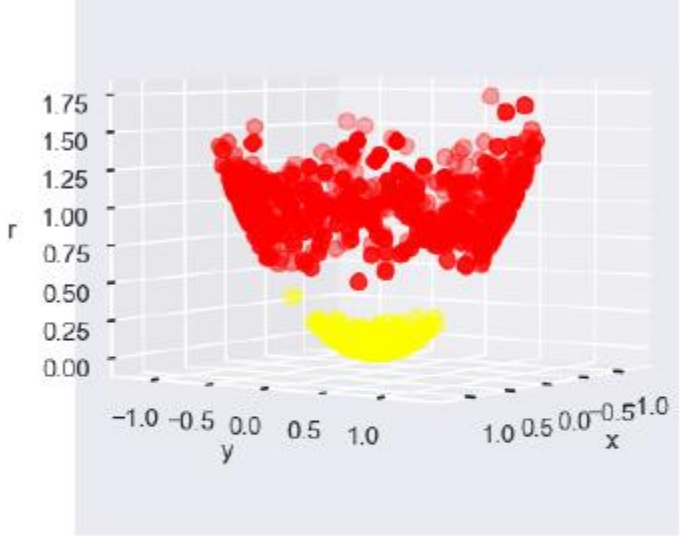
(b) Linear svm on non-linearly separable data

Non-Linear Classification

- In this lecture, we will explore the idea of non-linear classification. The idea that we will use is that we will transform the data $x \in \mathbb{R}^{d_1}$ to $\phi(x) : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$, where $d_2 > d_1$, and the transformed data will be linearly separable.



(a) Non-linearly separable data



(b) Linearly Separable Transformed Data

Non-Linear Classification

- In the previous example we have

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$$

The data is not linearly separable and so to overcome this issue we will do the following transformation:

$$\phi(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix} \in \mathbb{R}^3$$

After the new transformation, the data is linearly separable. The classifier will have the following form:

$$h(\phi(\boldsymbol{x}); \theta, \theta_0) = \text{sgn}(\theta^T \phi(\boldsymbol{x}) + \theta_0) = \text{sgn}(\theta_1 x_1 + \theta_2 x_2 + \theta_3 (x_1^2 + x_2^2) + \theta_0)$$

Non-Linear Classification

- The classifier that we will get will be a circle in 2D as shown in figure below

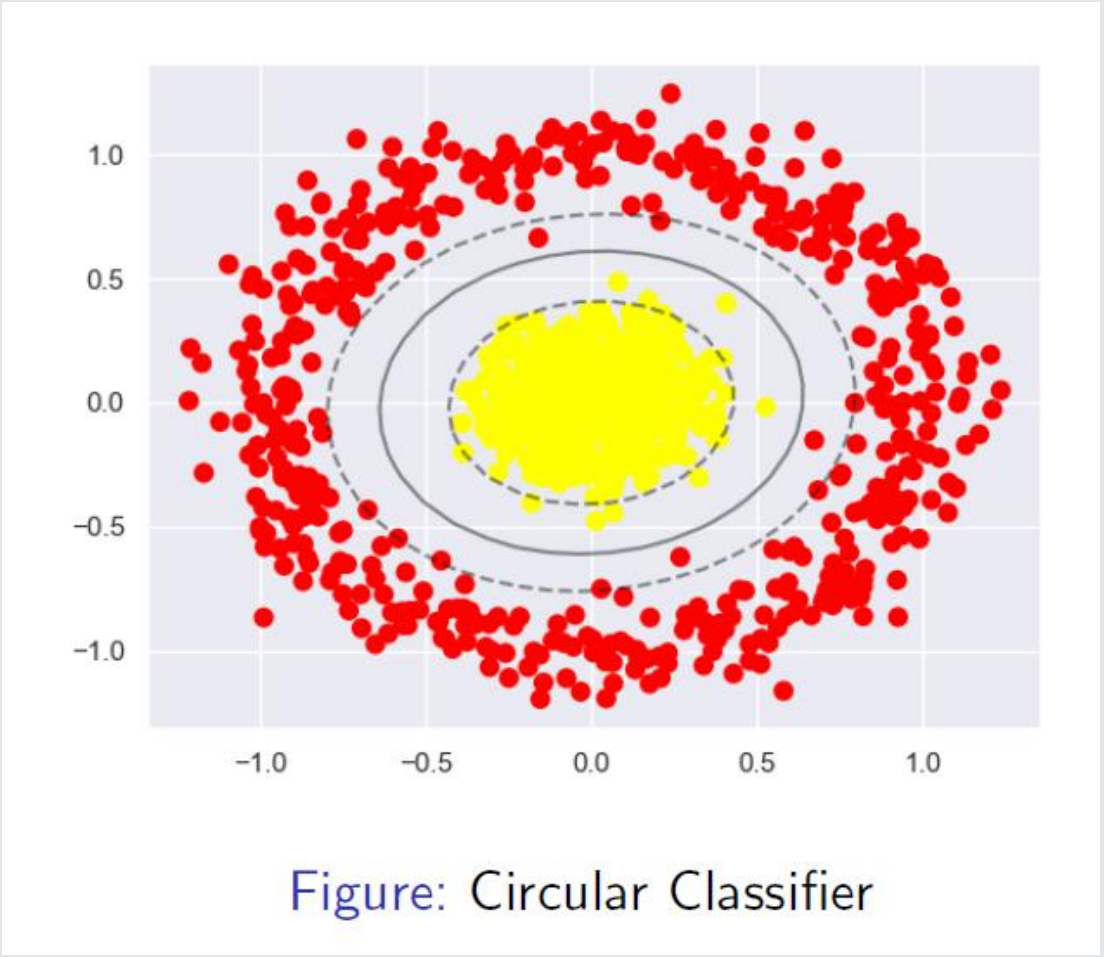


Figure: Circular Classifier

Kernel Functions

- As we have seen that these functions will map the data to higher dimensions and this results in larger vectors and so higher complexity. Therefore, we will discuss two specific functions we can compute efficiently.
- **Polynomial function**: Assume we have $\boldsymbol{x} \in \mathbb{R}^n$ and the polynomial of degree d:

$$\phi(\boldsymbol{x}) = \left[\frac{\sqrt{d!}}{\sqrt{j_1! j_2! \cdots j_{n+1}!}} x_1^{j_1} \cdots x_n^{j_n} 1^{j_{n+1}} \right]_{j_1 + j_2 + \cdots + j_{n+1} = d}$$

For example if we have $n = 2, d = 2$

$$\phi(\boldsymbol{x}) = \begin{bmatrix} 1\,(j_1 = 0, j_2 = 0, j_3 = 2) \\ \sqrt{2}x_1(j_1 = 1, j_2 = 0, j_3 = 1) \\ \sqrt{2}x_2(j_1 = 0, j_2 = 1, j_3 = 1) \\ \sqrt{2}x_1x_2(j_1 = 1, j_2 = 1, j_3 = 0) \\ x_1^2(j_1 = 2, j_2 = 0, j_3 = 0) \\ x_2^2(j_1 = 0, j_2 = 2, j_3 = 0) \end{bmatrix}$$

Kernel Functions

- The size of the vector $\phi(\boldsymbol{x})$ is

$$\binom{n+d}{d} = \frac{(n+d)!}{d!\,n!}$$

If we have a large n or d or both then $\phi(\boldsymbol{x})$ can get very large and become very expensive to deal with

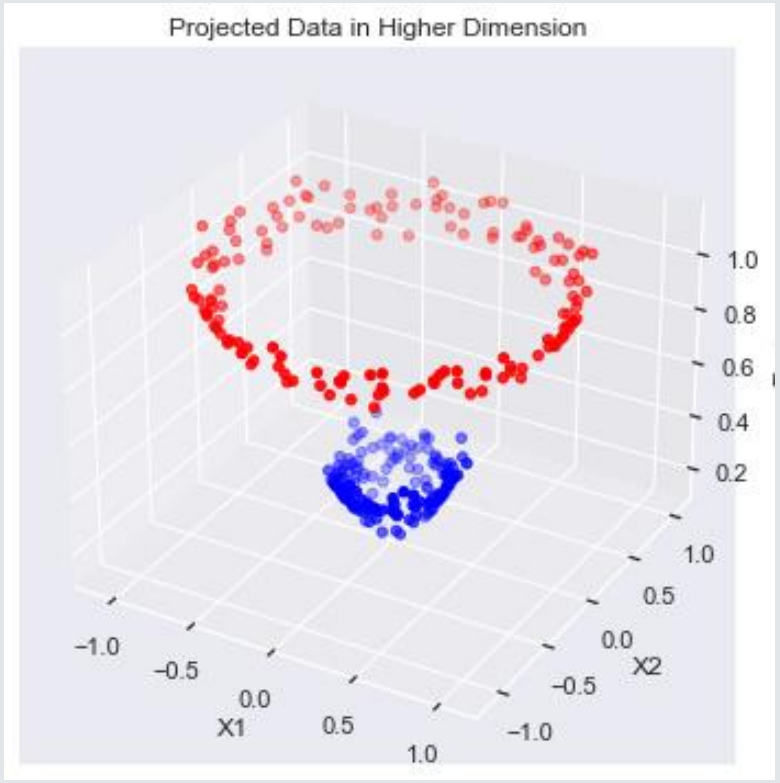
$$n = 20, d = 2 \Rightarrow \binom{22}{2} = 231$$

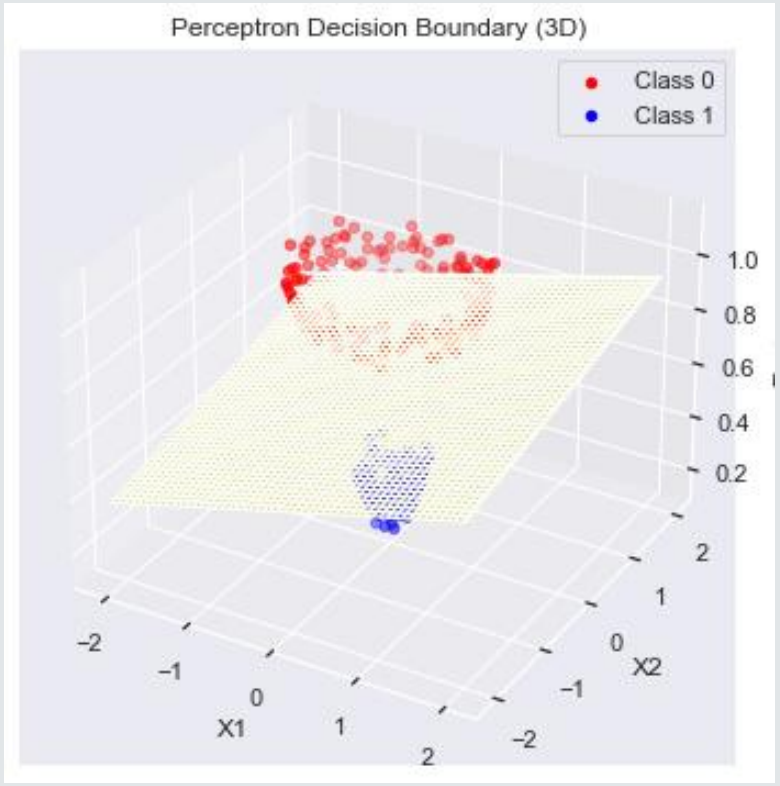
$$n = 100, d = 2 \Rightarrow \binom{102}{2} = 5151$$

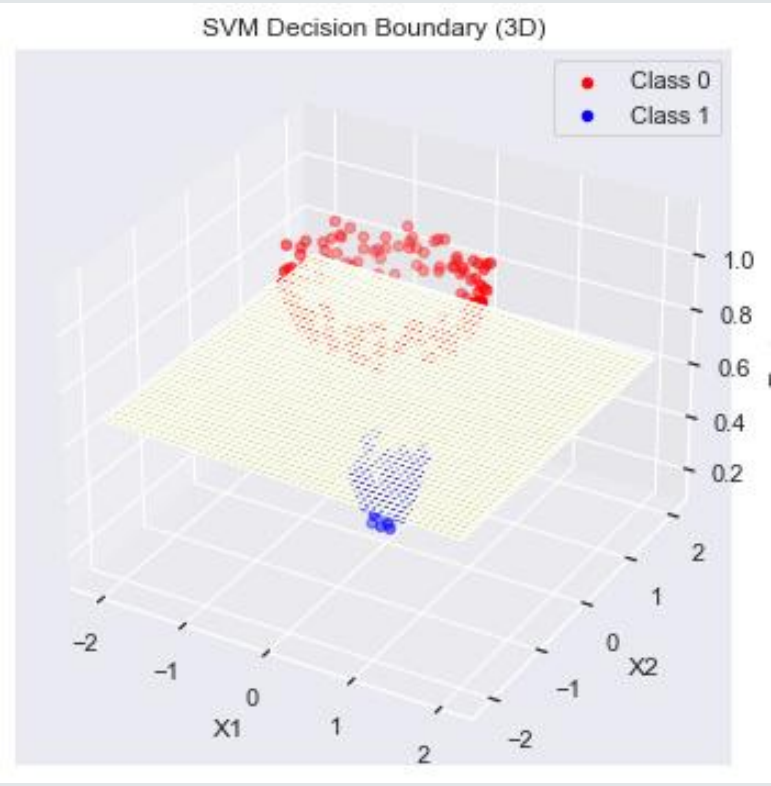
$$n = 100, d = 3 \Rightarrow \binom{103}{3} = 176851$$

Non-Linear Classification (Experiment 3)

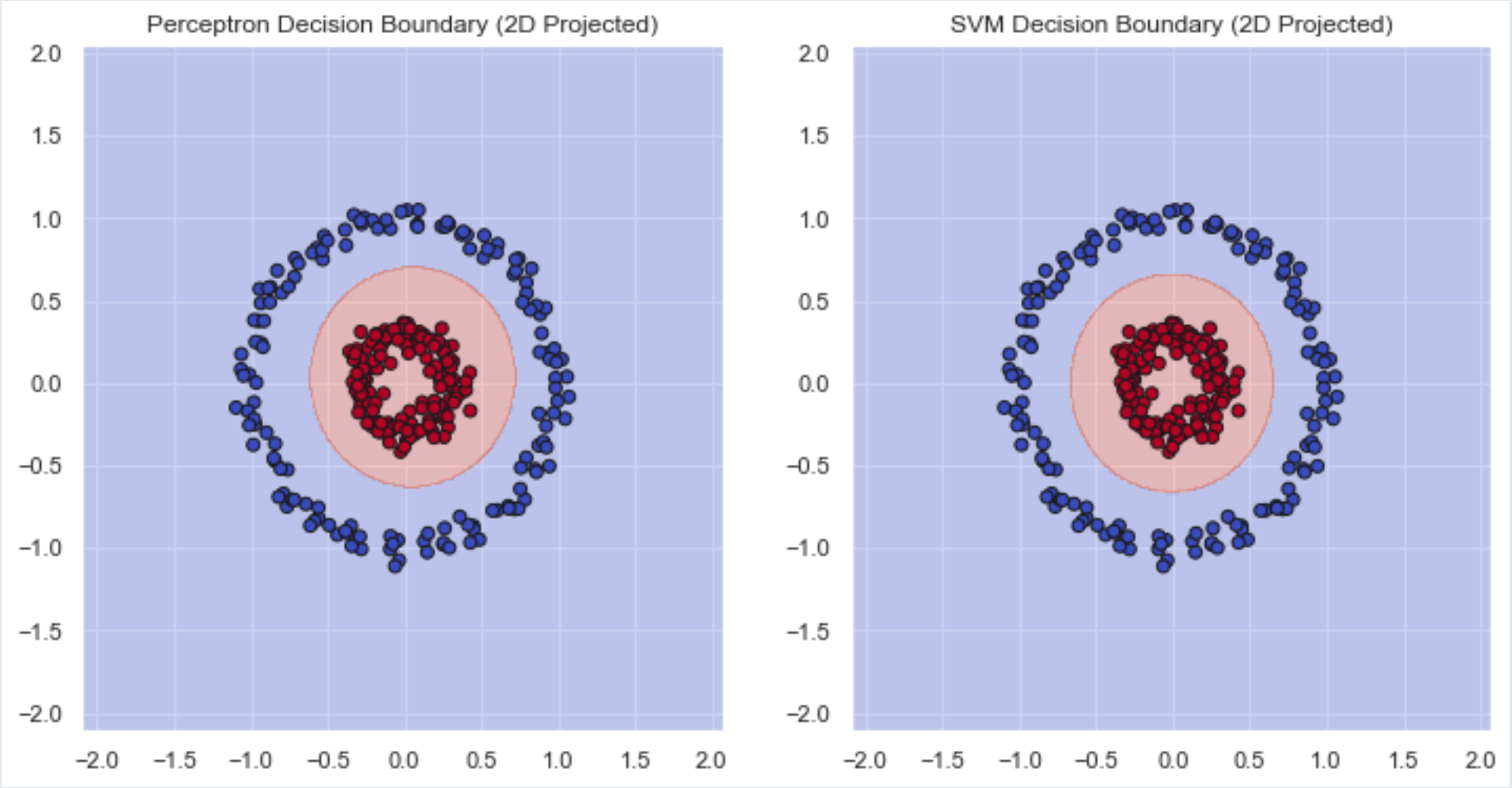
Non-Linear Classification (Experiment 3)







Non-Linear Classification (Experiment 3)



Kernel Functions

- **Additivity**: The sum of two valid kernels is also a valid kernel.
- **Scalar Multiplication**: The product of a valid kernel and a positive scalar is also a valid kernel.
- **Product of Kernels**: The product of two valid kernels is also a valid kernel.
- **Exponentiation**: Raising a valid kernel to a positive power yields another valid kernel.

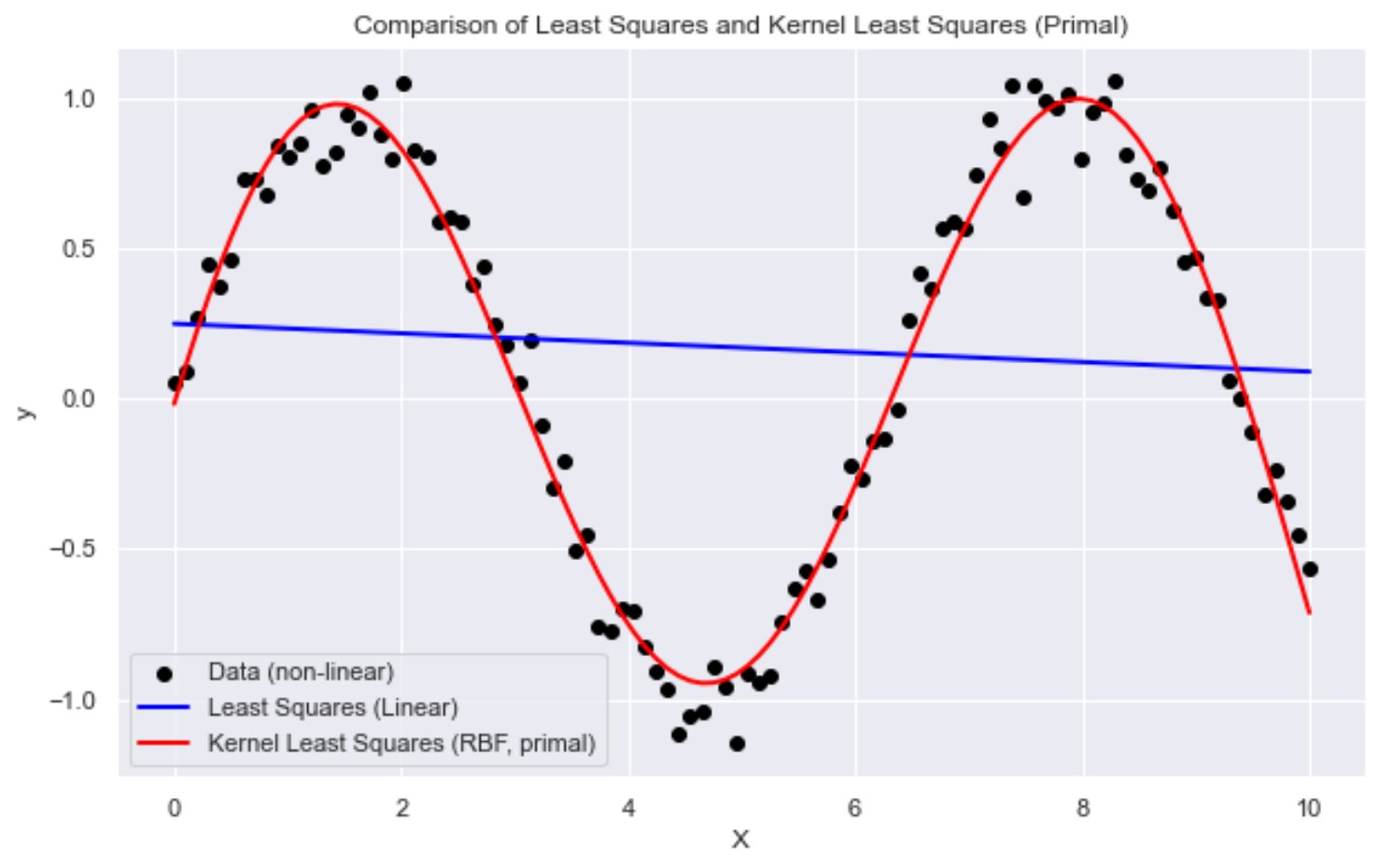
Kernel Least Square

- In linear least squares, the model is limited to linear relationships between input features and outputs. To handle non-linear relationships, we can map the input data into a higher-dimensional space using a feature map $\phi(x)$, which transforms $x \in \mathbb{R}^d$ to a higher-dimensional space.

$$\min_{\beta} \left|\left|y - \phi(X)\beta\right|\right|_2^2$$

$$\boldsymbol{\beta = \left(\phi(X)^T \phi(X)\right)^{-1} \phi(X)^T y}$$

Kernel Least Square (Experiment 4)



Limitations of Kernel Least Square

- **Complex Patterns:** When data is too complex, kernel methods might not capture the intricacies
- **Overfitting:** Flexible kernels can easily overfit, especially in noisy datasets.
- **Curse of Dimensionality:** Kernel methods may become ineffective in very high-dimensional spaces.
- **Kernel and Parameter Tuning:** Choosing the right kernel and tuning parameters is often non-trivial.

Lecture 8 (k-NN, Decision Tree)

k-nearest neighbor

- *k*-Nearest Neighbors (*k*-NN) is an intuitive, simple, yet powerful **non-parametric** and **instance-based** learning algorithm widely used for both **classification** and **regression** tasks.
- Unlike many supervised learning algorithms that require training a model, *k*-NN stores the entire training dataset and makes predictions for new data points by comparing them directly to the stored instances. It is a **lazy learning algorithm**, meaning no explicit training occurs, and the predictions are made based on the proximity of new instances to existing ones.

k-nearest neighbor

- Given a query point x_q, the *k*-NN algorithm identifies the **k nearest neighbors** in the training set based on a distance metric:

Euclidean Distance

$$d(\boldsymbol{x}_q, \boldsymbol{x}_i) = \sqrt{\sum_{j=1}^{n} (x_{qj} - x_{ij})^2}$$

Manhattan Distance

$$d(\boldsymbol{x}_q, \boldsymbol{x}_i) = \sum_{j=1}^{n} |x_{qj} - x_{ij}|$$

k-nearest neighbor - Classification

- In classification tasks, the algorithm assigns a class label to the query point x_q based on the most frequent label (the mode) among its *k*-nearest neighbors.
- Once the *k*-nearest neighbors are identified, the class labels of the neighbors are collected, and the query point is classified by majority vote:

$$\hat{y}_q = mode([y_1, y_2, \cdots, y_k])$$

where $y_1, y_2, \cdots, y_k$ are the class labels of the *k*-nearest neighbors.

- **Example 1:**
- For a query point with neighbors' class labels [0, 1, 1, 0, 1], the **mode** is 1, so the predicted class is 1.
- **Example 2:**
- For a query point with neighbors' class labels [0, 1, 1, 0], the **mode** is ?. There is a **tie**

k-nearest neighbor - Classification

- In cases where multiple classes have the same frequency among the neighbors, a **tie** may occur. Possible strategies to handle ties include:
- **Random selection**: Randomly select one of the tied classes.
- **Preference to closest neighbor**: Choose the class label of the nearest neighbor in case of a tie.
- **Weighted voting**: Apply weights based on the distance of each neighbor, giving closer neighbors more influence in the decision.

Approach	**Advantages**	**Disadvantages**
Random Selection	Simple, neutral bias	Inconsistent, lacks interpretability
Nearest Neighbor Preference	Consistent, clear logic, interpretable	Sensitive to noise and scaling issues
Weighted Voting	Reliable, less noisy, reduces impact of ties	Extra computational cost, sensitive to weight choice

k-nearest neighbor - Regression

- In regression tasks, *k*-NN predicts the output based on the average of the target values of the *k*-nearest neighbors.
- For a query point x_q, the predicted value is the mean of the target values y_i of the nearest neighbors:

$$\hat{y}_q = \frac{1}{k}\sum_{i=1}^{k} y_i$$

- To improve performance, especially when some neighbors are much closer than others, weighted *k*-NN can be used. Here, each neighbor's influence is weighted by its distance from the query point:

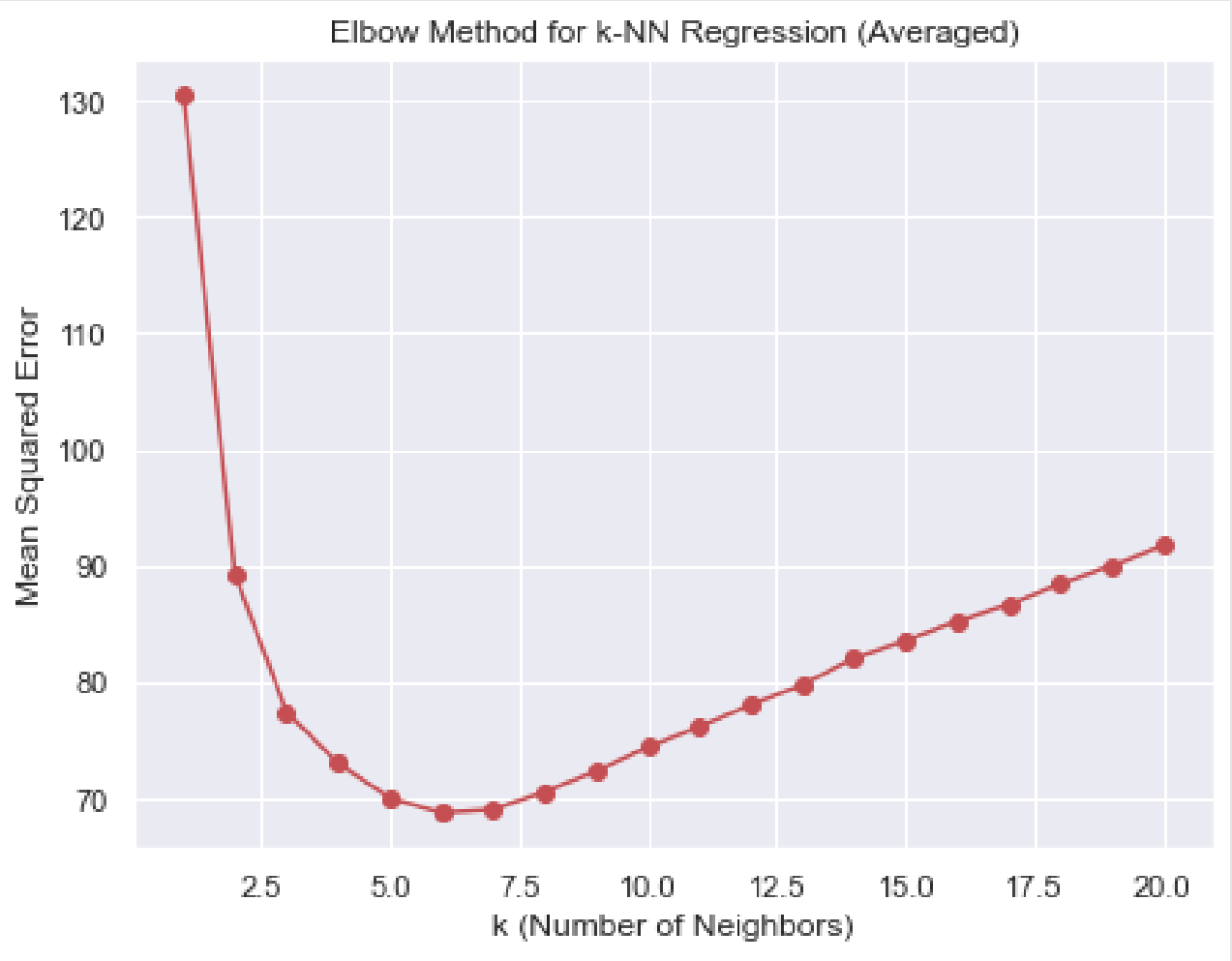
$$\hat{y}_q = \frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

where $w_i = \frac{1}{d(x_q, x_i)}$ is the weight assigned to neighbor *i* based on its distance to the query point.

k-nearest neighbor

- Selecting an appropriate value of *k* is crucial for *k*-NN's performance. Small values of *k* can lead to **overfitting**, while large values can cause **underfitting**.
- One common approach to choose *k* is the **elbow method**. This method involves plotting the error (classification error rate or mean squared error for regression) as a function of *k* and identifying the point where the error stops decreasing significantly (the "elbow" point).

k-nearest neighbor



Experiment 5

Limitations *k*-nearest neighbor

- **Curse of Dimensionality**: As the number of features increases, the distance between data points becomes less meaningful. In high-dimensional spaces, points tend to become equidistant from each other, reducing the effectiveness of distance-based methods like k-NN.

  ***Solution**: Dimensionality reduction techniques like **Principal Component Analysis (PCA)** or **Linear Discriminant Analysis (LDA)** can help reduce the number of features, improving the meaningfulness of distance computations.*

- **Computational Complexity**: For each query, *k*-NN requires computing the distance between the query point and all training points. This leads to a time complexity of $O(N \cdot n)$, where N is the number of training points and n is the number of features.

  ***Solution**: Data structures such as **KD-trees** and **ball trees** can reduce the computational burden, especially in low-dimensional data.*

- **Imbalance in Class Distribution**: When one class dominates the dataset, the nearest neighbors of any query point may frequently belong to the majority class, leading to biased predictions.

  ***Solution**: Use **distance-weighted voting**, where closer neighbors are given more importance in the decision-making process, reducing bias toward majority classes.*

Decision Tree

- A decision tree is a simple model for supervised classification. It is used for classifying a single discrete target feature.
- Each internal node performs a Boolean test on an input feature (in general, a test may have more than two options, but these can be converted to a series of Boolean tests). The edges are labeled with the values of that input feature.
- Each leaf node specifies a value for the target feature.

Content adapted from Alice Gao

Decision Tree – Example 1 (All examples belong to the same class)

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Content adapted from Alice Gao

Decision Tree – Example 2 (No features left)

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No
15	Sunny	Mild	High	Weak	No
16	Sunny	Mild	High	Weak	Yes
17	Sunny	Mild	High	Strong	Yes

Content adapted from Alice Gao

Decision Tree – Example 3 (No examples left)

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No
15	Sunny	Hot	High	Weak	No

Content adapted from Alice Gao

Decision Tree

Which feature we use at each step?

- Ideally, we would like to find the optimal order of testing features, which will minimize the size of our tree. Unfortunately, finding the optimal order is too expensive computationally. Instead, we will use a greedy approach.
- The greedy approach will make the best choice at each step without worrying about how our current choice could affect the potential choices in the future. More concretely, at each step, we will choose a feature that makes the biggest difference to the classification, or a feature that helps us make a decision as quickly as possible.

Content adapted from Alice Gao

Decision Tree

- Feature that reduces our uncertainty at much as possible will be chosen.
- To measure the reduction in uncertainty, we will calculate the uncertainty in the examples before testing the feature, and subtract the uncertainty in the examples after testing the feature. The difference measures the information content of the feature. Intuitively, testing the feature allows us to reduce our uncertainty and gain some useful information. We will select the feature that has the highest information content.
- How do we measure uncertainty?

Content adapted from Alice Gao

Decision Tree

$$I\big(P(c_1),\cdots,P(c_k)\big) = -\sum_{i=1}^{k} P(c_i)\log_2(P(c_i))$$

- What is the entropy of the distribution (0.5,0.5)?

$$-0.5\log_2(0.5) - 0.5\log_2(0.5) = 1$$

This distribution has 1 bit of uncertainty.

- What is the entropy of the distribution (0.01,0.99)?

$$-0.01\log_2(0.01) - 0.99\log_2(0.99) = 0.08$$

This distribution has 0.08 bit of uncertainty.

- The entropy is maximized in the case of a uniform distribution
- Information gain will be the metric to be used to determine the feature to be used (ID3).

$$\text{InfoGain} = \text{I}_{\text{before}} - \text{I}_{\text{after}} = \text{I}_{\text{before}} - \sum_{i=1}^{k} \frac{p_i + n_i}{p + n} * I(\frac{p_i}{p + n}, \frac{n_i}{p + n})$$

Content adapted from Alice Gao

Decision Tree – Example 4

There are 14 examples, 9 positive and 5 negative

What is the entropy of the examples before we select a feature for the root node of the tree?

$$I\left(\frac{9}{14},\frac{5}{14}\right) = -\left(\frac{9}{14}\log_2\frac{9}{14}+\frac{5}{14}\log_2\frac{5}{14}\right) \approx 0.94$$

What is the expected information gain if we select Outlook as the root node of the tree?

$$Outlook = \begin{cases} Sunny; 2\ Yes, 3\ No\ ; 5\ Total \\ Overcast; 4\ Yes, 0\ No\ ; 4\ Total \\ Rain; 4\ Yes, 2\ No\ ; 5\ Total \end{cases}$$

$$Gain(Outlook) = 0.94 - \left(\frac{5}{14}.I\left(\frac{2}{5},\frac{3}{5}\right)+\frac{4}{14}.I\left(\frac{4}{4},\frac{0}{4}\right)+\frac{5}{14}I\left(\frac{3}{5},\frac{2}{5}\right)\right)$$

$$= 0.94 - \left(\frac{5}{14}(0.971)+\frac{4}{14}(0)+\frac{5}{14}(0.971)\right) = 0.94 - 0.694 = 0.247$$

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Content adapted from Alice Gao

Decision Tree – Example 4

What is the expected information gain if we select Humidity as the root node of the tree?

$$Humidity = \begin{cases} Normal\ ; 6\ Yes\ 1\ No; 7\ Total \\ High; 3\ Yes\ 4\ No; 7\ Total \end{cases}$$

$$Gain(Humidity) = 0.94 - \left(\frac{7}{14}.I\left(\frac{6}{7},\frac{1}{7}\right) + \frac{7}{14}.I\left(\frac{3}{7},\frac{4}{7}\right)\right)$$

$$= 0.94 - \left(\frac{7}{14}(0.592) + \frac{7}{14}(0.985)\right) = 0.94 - 0.789 = 0.151$$

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Content adapted from Alice Gao

Decision Tree – Example 4

What is the expected information gain if we select Wind as the root node of the tree?

$$Wind = \begin{cases} Weak\ ; 6\ Yes\ 2\ No; 8\ Total \\ Strong; 3\ Yes\ 3\ No; 6\ Total \end{cases}$$

$$Gain(Wind) = 0.94 - \left(\frac{8}{14}.I\left(\frac{6}{8},\frac{2}{8}\right) + \frac{6}{14}.I\left(\frac{3}{6},\frac{3}{6}\right)\right)$$

$$= 0.94 - \left(\frac{8}{14}(0.81) + \frac{6}{14}(1)\right) = 0.94 - 0.891 = 0.0485$$

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Content adapted from Alice Gao

Decision Tree – Example 4

What is the expected information gain if we select Temperature as the root node of the tree?

$Gain(Temperature) = 0.029$

Day	Outlook	Temp	Humidity	Wind	Tennis?
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Content adapted from Alice Gao

Decision Tree

- Information gain is not the only splitting criterion to be used in decision tree.
- The next criterion is the gini index (CART)

$$Gini(t) = 1 - \sum_{i=1}^{C} p_i^2$$

where

C is the number of classes

p_i is the proportion of instances belonging to class *i* at particular node t.

- Gini index is computationally efficient
- Information gain will be useful in the case of imbalanced datasets

Decision Tree

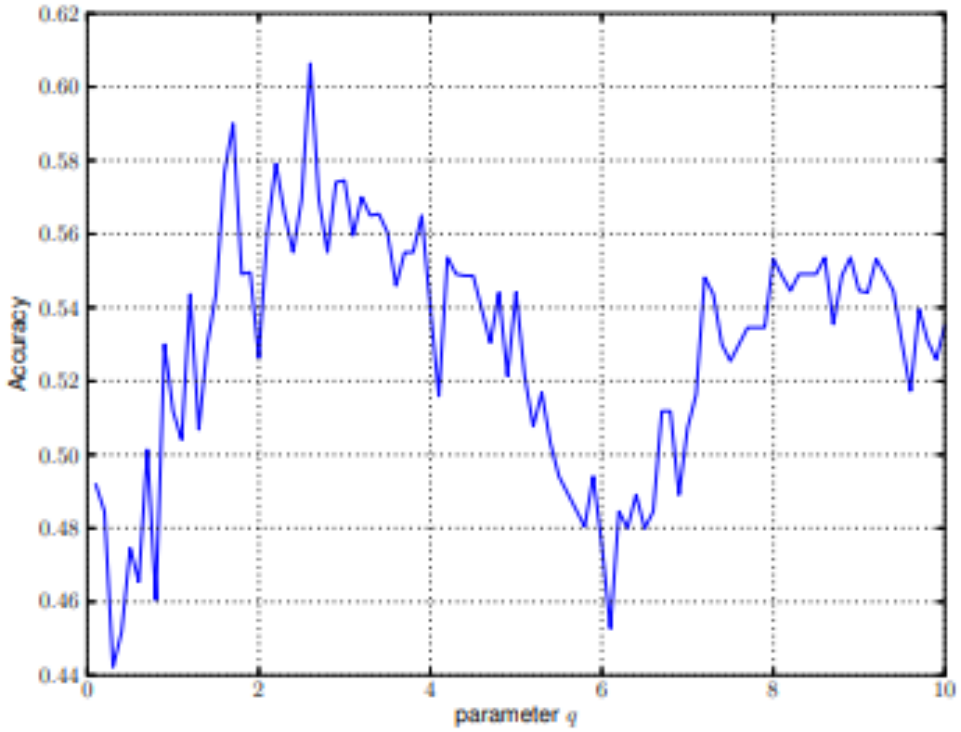
- The gini coefficient and the information gain (entropy) are different splitting criteria but are unified under what is known as the **Tsallis Entropy**.

$$S_q(X) = \frac{1}{1-q}\left(\sum_{i=1}^{n} p_i^q - 1\right), q \in \mathbb{R}$$

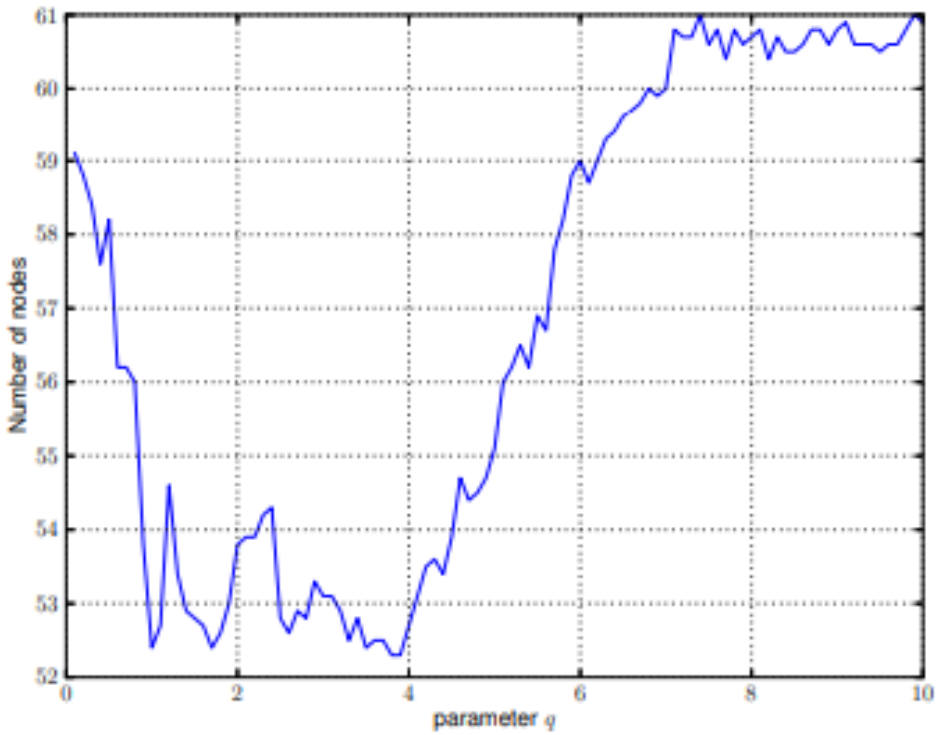
$$\lim_{q \to 1} S_q(X) = H(X)$$

$$S_2(X) = 1 - \sum_{i=1}^{n} p_i^2 = Gini\ Index$$

Decision Tree



(a) Accuracy with different values of q



(b) Tree complexity with different values of q

Wang, Yisen, Chaobing Song, and Shu-Tao Xia. "Unifying the Split Criteria of Decision Trees Using Tsallis Entropy." *arXiv preprint arXiv:1511.08136* (2016).

Decision Tree

Data Set	Shannon entropy (ID3)		Gini index (CART)		Tsallis entropy (TEC)		
	Accuracy (%)	No. of nodes	Accuracy (%)	No. of nodes	Accuracy (%)	No. of nodes	q
Yeast	52.8	199	51.8	196.6	**56.9**	**195.8**	1.4
Glass	51.2	52.4	52.6	53.8	**60.6**	52.6	2.6
Vehicle	71.7	103	70.2	**100**	**73.8**	111.0	0.6
Wine	92.9	12.0	90.0	12.0	**95.9**	9.6	3.1
Haberman	70.3	32.2	70.3	33.0	74.2	33.2	7.1
Car	98.2	106.4	98.1	106.8	98.3	**106.2**	0.8
Scale	75.9	97.6	76.1	97.2	78.2	93.1	3.1
Hayes	81.5	28.8	80.0	25.3	**82.3**	19.5	8.6
Monks	51.9	89.0	52.1	88.6	**57.3**	89.6	8.9
Abalone	25.4	89.2	25.0	85.8	**26.8**	86.2	0.8
Cmc	49.1	267.0	47.4	264.0	**52.0**	264.2	1.2

Wang, Yisen, Chaobing Song, and Shu-Tao Xia. "Unifying the Split Criteria of Decision Trees Using Tsallis Entropy." *arXiv preprint arXiv:1511.08136* (2016).

Choosing the optimal value of q is still an open question

Decision Tree

- It would be better to grow a smaller and shallower tree. The smaller and shallower tree may not predict all of the training data points perfectly but it may generalize to test data better.
- We have two options to prevent over-fitting when learning a decision tree
- Pre-pruning: stop growing the tree early
- Post-pruning: grow a full tree first and then trim it afterwards.

Content adapted from Alice Gao

Decision Tree

Pre-pruning

- If we decide not to split the examples at a node and stop growing the tree there, we may still have examples with different labels. At this point, we can decide to use the majority label as the decision for that leaf node. Here are some criteria we can use:
- **Maximum depth:** We can decide not to split the examples if the depth of that node has reached a maximum value that we decided beforehand.
- **Minimum number of examples at the leaf node:** We can decide not to split the examples if the number of examples remaining at that node is less than a predefined threshold value.
- **Minimum information gain:** We can decide not to split the examples if the benefit of splitting at that node is not large enough. We can measure the benefit by calculating the expected information gain. In other words, do not split examples if the expected information gain is less than the threshold.
- **Reduction in training error:** We can decide not to split the examples at a node if the reduction in training error is less than a predefined threshold value.

Content adapted from Alice Gao

Decision Tree

- Post-pruning is particularly useful when any individual feature is not informative, but multiple features working together is very informative.
- **Example:** Suppose we are considering post-pruning with the minimal information gain metric.

  First of all, we will restrict our attention to nodes that only have leaf nodes as its descendants. At a node like this, if the expected information gain is less than a predefined threshold value, we will delete this node's children which are all leaf nodes and then convert this node to a leaf node.

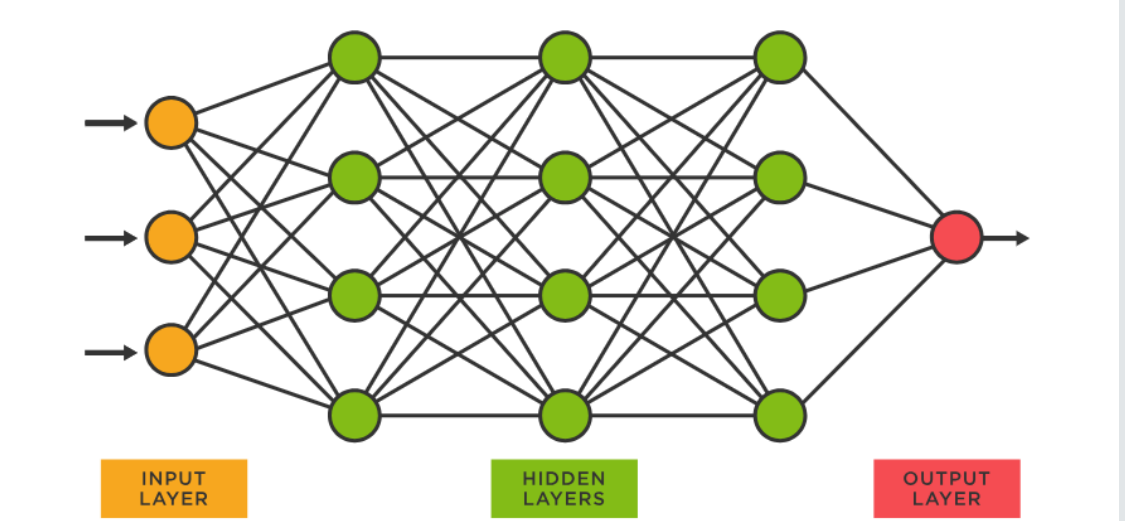
  There has to be examples with different labels at this node possibly both positive and negative examples. We can make a majority decision at this node.

Content adapted from Alice Gao

Lecture 9 (FNN)

Feedforward Neural Network

- Neural networks learns a mapping function $f: X \rightarrow Y$ from input features $X \in \mathbb{R}^n$ to output labels Y, where:
  - Classification: Y is discrete (e.g., $Y \in \{1,2,\cdots,k\}$).
  - Regression: Y is continuous (e.g., $Y \in \mathbb{R}$).
- Key components of a neural network:
  - Layers (Depth): define the number of layers. This include the input, hidden, and output layer.
  - Width (Number of neurons per layer): define the number of neurons per layers.
  - Activation function: This will add a layer of non-linearity to allow to model complex relationships
  - Loss function: Measures the difference between the predicted output and the actual target. (MSE [Regression], Cross-entropy [Classification])



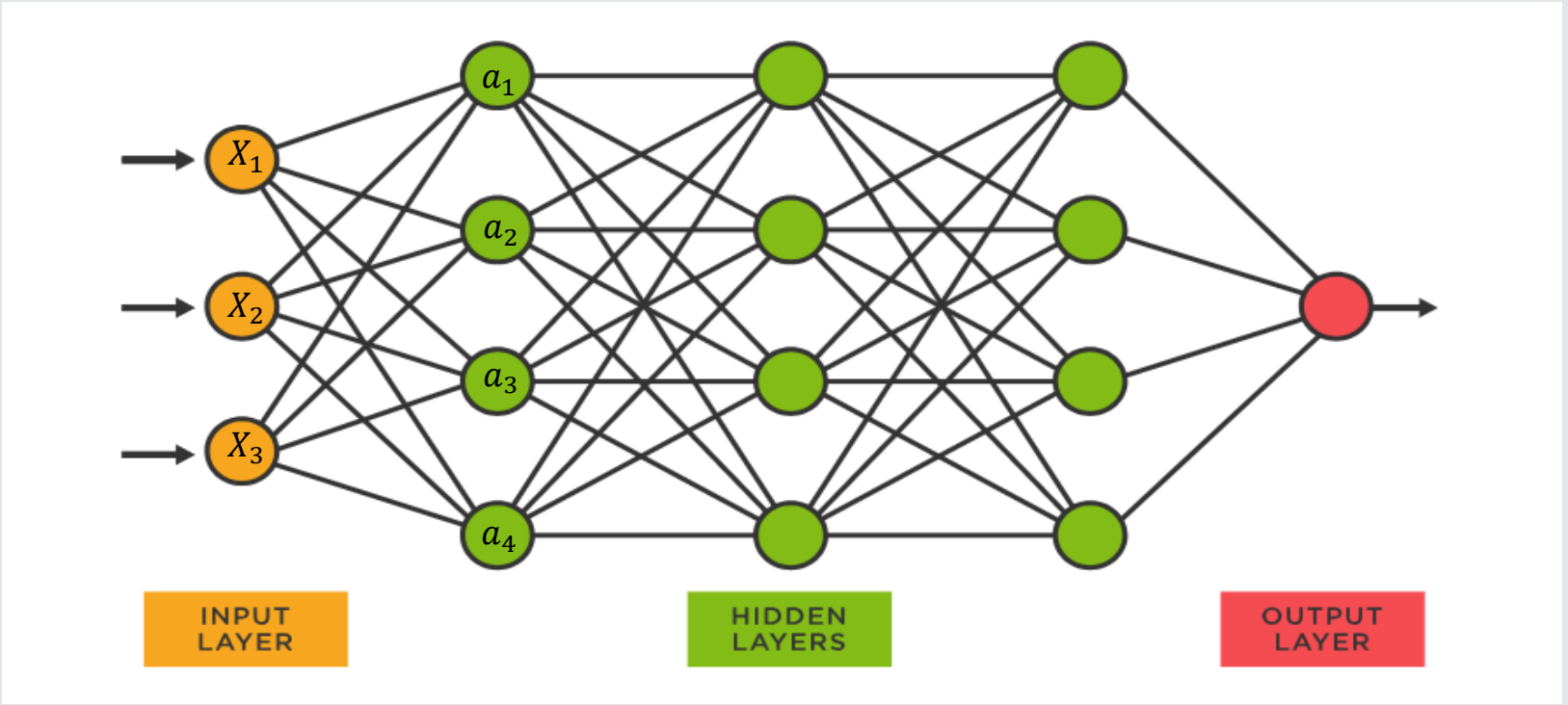
https://www.spotfire.com/glossary/what-is-a-neural-network

Feedforward Neural Network

- $a_1 = f(W_1 X + b_1)$

where $W_1 \in \mathbb{R}^{1\times 3}, X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}, b_1 \in \mathbb{R}$

f is the activation function which could be a linear or a non-linear activation function. The non-linearity will give us the flexibility of modelling non-linear problems.



https://www.spotfire.com/glossary/what-is-a-neural-network

Feedforward Neural Network – Forward Propagation (Classification)

- $a_1 = f\left(W_1^{(1)}X + b_1\right) = \left(W_1^{(1)}X + b_1^{(1)}\right)$; f is a linear activation function.

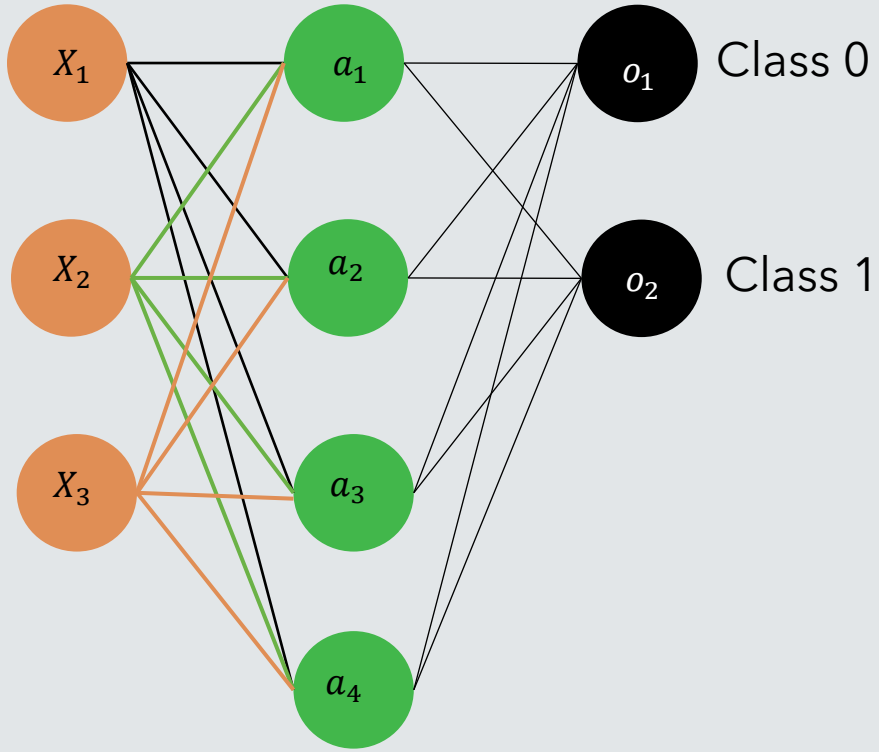
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1\times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f\left(W_1^{(2)}a + b_1^{(2)}\right) = f(\hat{o}_1) = \frac{e^{\hat{o}_1}}{e^{\hat{o}_1}+e^{\hat{o}_2}}$; f is a softmax function
- $o_2 = f\left(W_2^{(2)}a + b_2^{(2)}\right) = f(\hat{o}_2) = \frac{e^{\hat{o}_2}}{e^{\hat{o}_1}+e^{\hat{o}_2}}$; f is a softmax function
- $l = -(y\log(o_2) + (1-y)\log(o_1))$; where y is the true label and $\hat{y}$ is the predicated label



Feedforward Neural Network – Forward Propagation (Classification)

- $a_1 = f\left(W_1^{(1)}X + b_1\right) = \left(W_1^{(1)}X + b_1^{(1)}\right)$; f is a linear activation function.

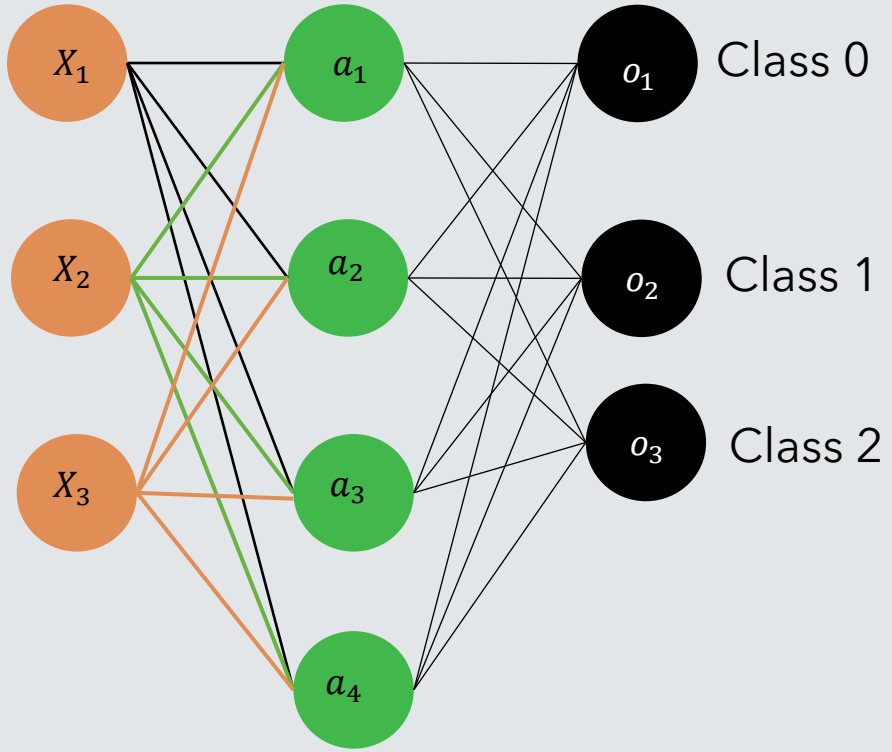
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1\times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f\left(W_1^{(2)}a + b_1^{(2)}\right) = f(\hat{o}_1) = \frac{e^{\hat{o}_1}}{e^{\hat{o}_1}+e^{\hat{o}_2}+e^{\hat{o}_3}}$; f is a softmax function
- $o_2 = f\left(W_2^{(2)}a + b_2^{(2)}\right) = f(\hat{o}_2) = \frac{e^{\hat{o}_2}}{e^{\hat{o}_1}+e^{\hat{o}_2}+e^{\hat{o}_3}}$; f is a softmax function
- $o_3 = f\left(W_2^{(2)}a + b_3^{(2)}\right) = f(\hat{o}_3) = \frac{e^{\hat{o}_3}}{e^{\hat{o}_1}+e^{\hat{o}_2}+e^{\hat{o}_3}}$; f is a softmax function
- $l = -\sum_{i=1}^{3} p_i \log(o_i)$; where p_i is the true distribution and o_i is the predicated distribution



Feedforward Neural Network – Forward Propagation (Regression)

- $a_1 = f\left(W_1^{(1)}X + b_1\right) = \left(W_1^{(1)}X + b_1^{(1)}\right)$; *f* is a linear activation function.

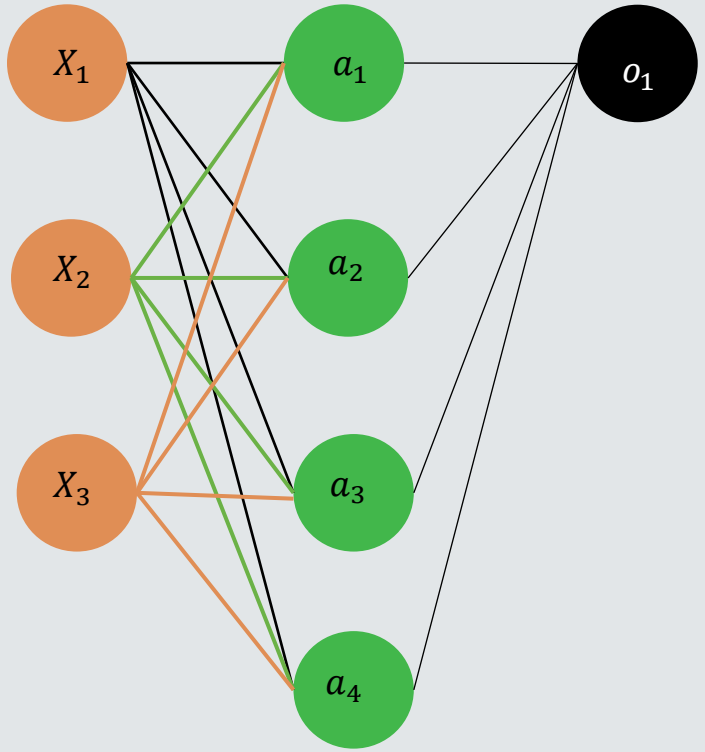
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1\times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f\left(W_1^{(2)}a + b_1^{(2)}\right) = W_1^{(2)}a + b_1^{(2)}$; *f* is a linear function
- $l = (y - o_1)^2$ (SE); where y is the true value and o_1 is the predicated value



Feedforward Neural Network – Forward Propagation (Regression)

- $a_1 = f\left(W_1^{(1)}X + b_1\right) = \left(W_1^{(1)}X + b_1^{(1)}\right)$; f is a linear activation function.

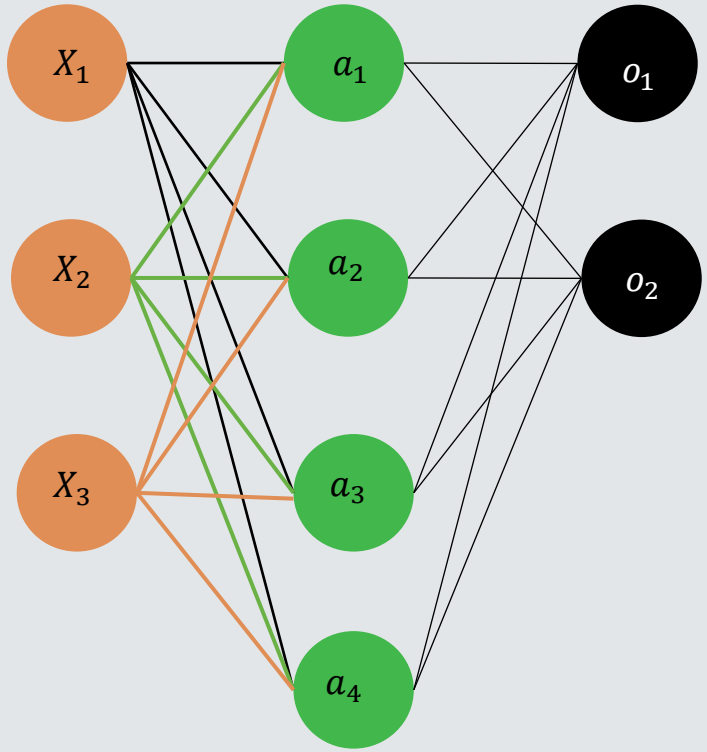
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1\times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

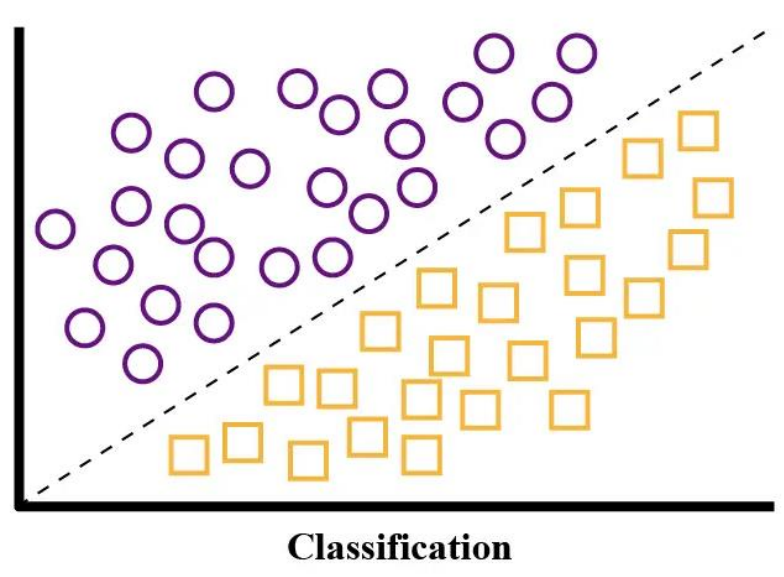
$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$

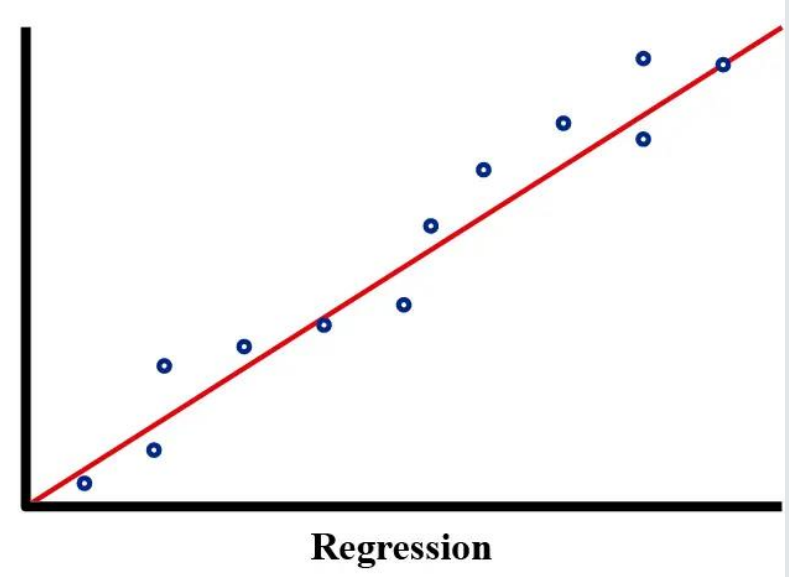
- $o_1 = f\left(W_1^{(2)}a + b_1^{(2)}\right) = W_1^{(2)}a + b_1^{(2)}$; f is a linear function
- $o_2 = f\left(W_2^{(2)}a + b_2^{(2)}\right) = W_2^{(2)}a + b_2^{(2)}$; f is a linear function
- $l = \frac{1}{2}[(y_1 - o_1)^2 + (y_2 - o_2)^2]$ (MSE); where y_1, y_2 is the true value and o_1, o_2 is the predicted value



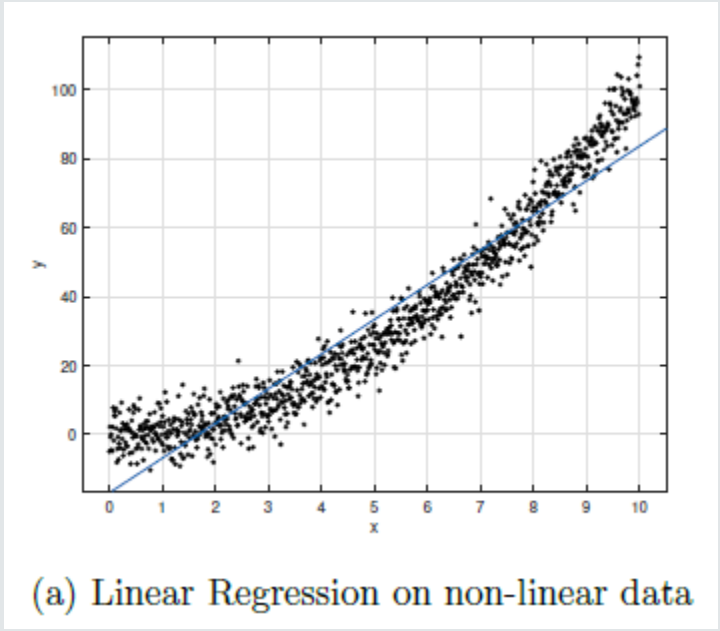
Feedforward Neural Network

- These linear activation functions will allow us to model and capture linear relationships as shown below but it fails in the case of non-linear relationships.

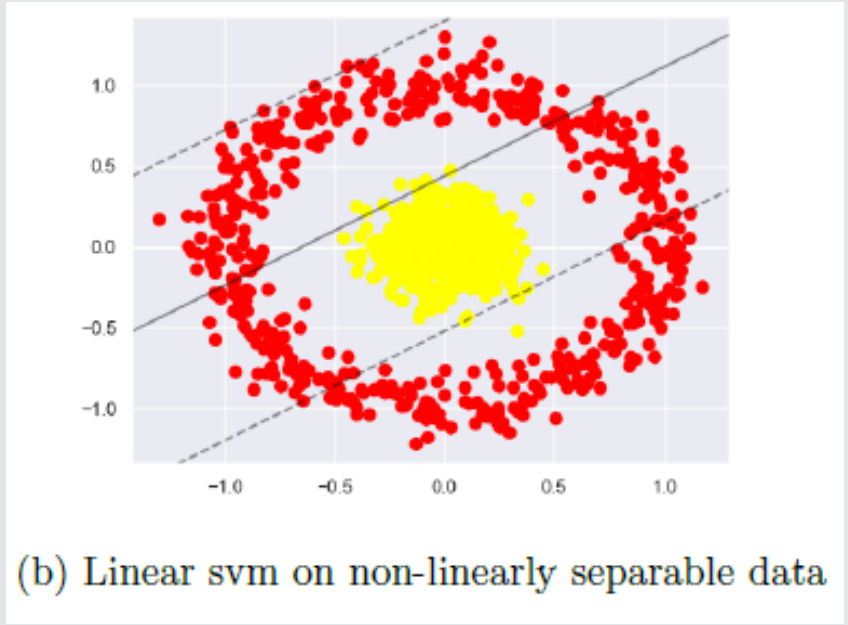




Feedforward Neural Network



(a) Linear Regression on non-linear data



(b) Linear svm on non-linearly separable data

Therefore, we need non-linear activation functions

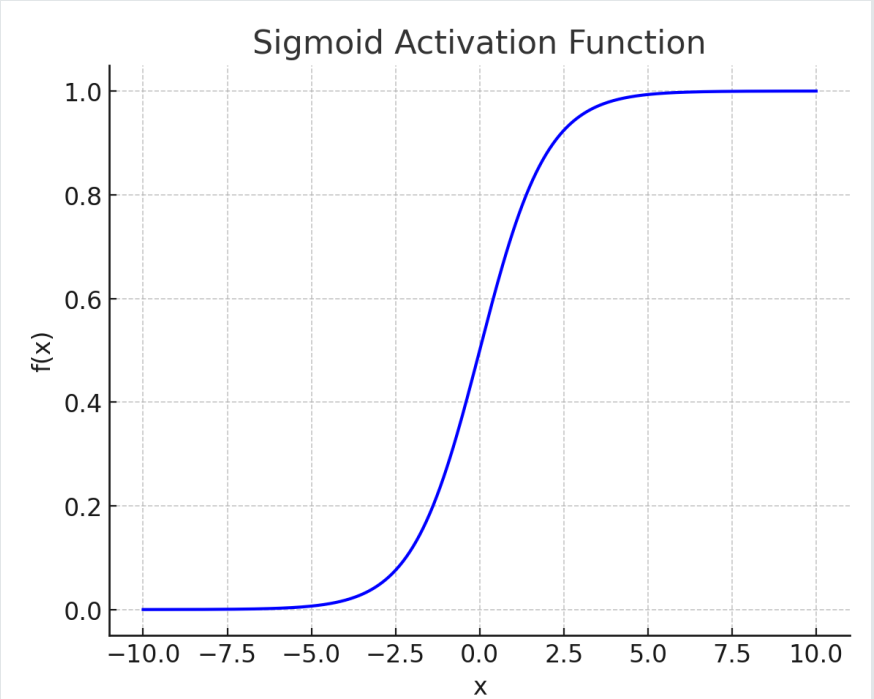
Feedforward Neural Network – Activation Functions

- Activation functions introduce non-linearity into neural networks, enabling them to approximate complex functions. Each function has unique characteristics that impact the network's performance, convergence, and generalization. Below are common activation functions used in neural networks.

Feedforward Neural Network – Activation Functions

- Sigmoid function

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



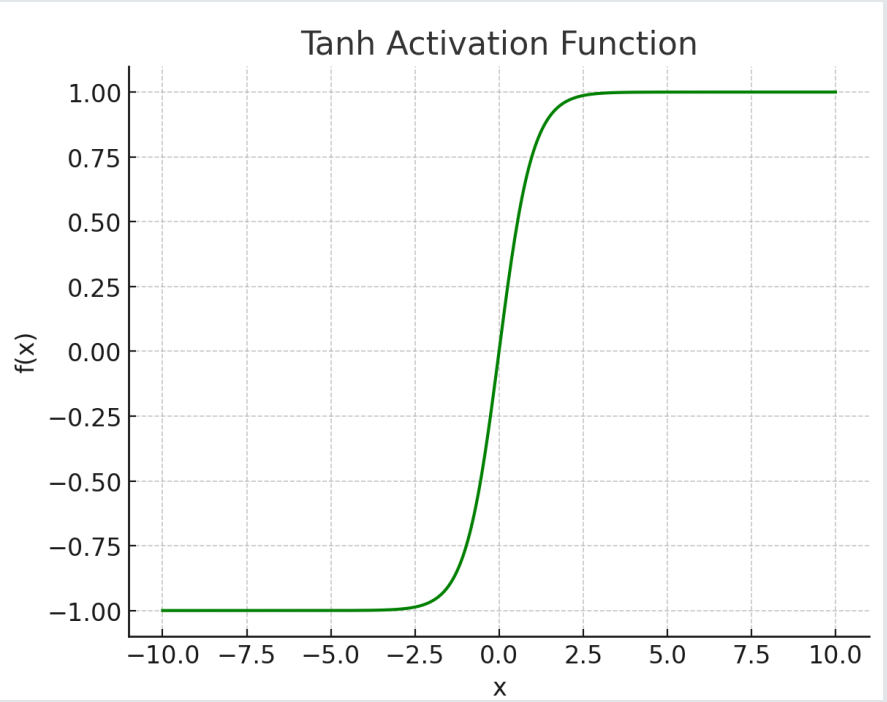
- **Advantages**:
  - *Smooth gradient, useful for binary classification.*
  - *Output values between 0 and 1, which can represent probabilities.*

- **Disadvantages:**
  - *Vanishing gradient problem for large or small inputs, slowing down training.*

Feedforward Neural Network – Activation Functions

- Tanh function

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



- **Advantages**:
  - *Zero-centered output, which helps faster convergence*
  - *Larger gradients compared to sigmoid, reducing vanishing gradient*

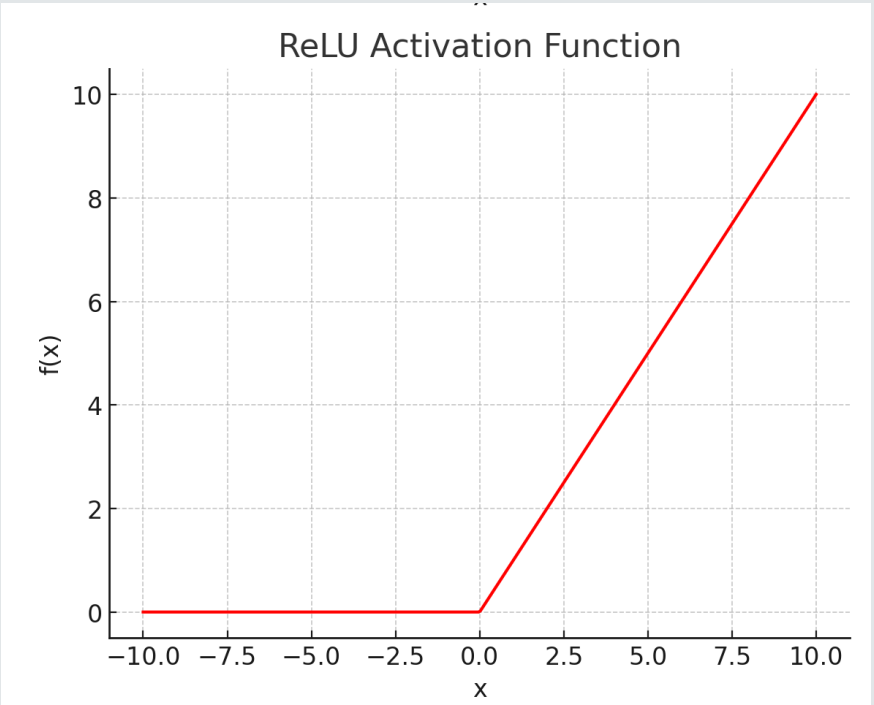
- **Disadvantages:**
  - *Still suffers from the vanishing gradient problem, especially in deep networks.*

Feedforward Neural Network – Activation Functions

- Relu function

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

- **Advantages**:
  - Solves vanishing gradient problem for positive values
- **Disadvantages:**
  - *Dead neurons issue, where certain neurons always output zero and stop learning.*
  - *Can cause gradient explosion in certain cases.*



Feedforward Neural Network – Activation Functions

- Add something about 0/1 function and the difficultity of optimization

Feedforward Neural Network – Forward Propagation (Classification)

- $a_1 = f\left(W_1^{(1)}X + b_1\right) = \begin{cases} \max\left(0, W_1^{(1)}X + b_1\right); f\ is\ a\ relu\ function \\ \frac{1}{1+e^{-W_1^{(1)}X+b_1}}; f\ is\ a\ sigmoid\ function \\ \frac{e^{W_1^{(1)}X+b_1}-e^{-W_1^{(1)}X+b_1}}{e^{W_1^{(1)}X+b_1}+e^{-W_1^{(1)}X+b_1}}; f\ is\ a\ \tanh function \end{cases}$

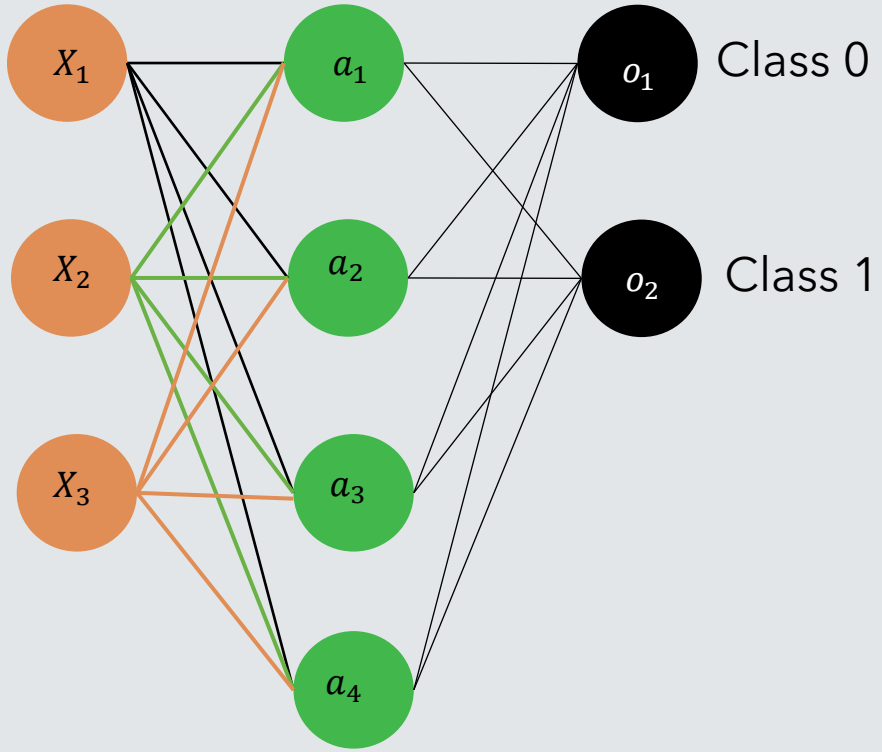
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1\times 3}$$

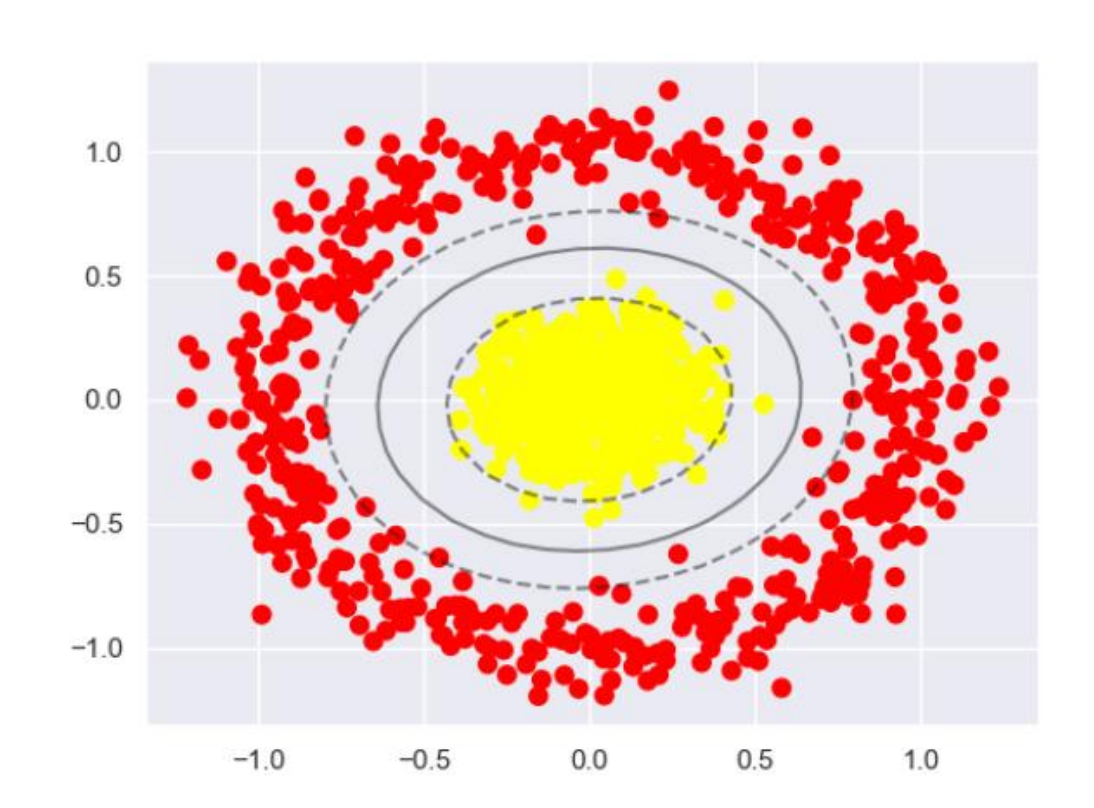
$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

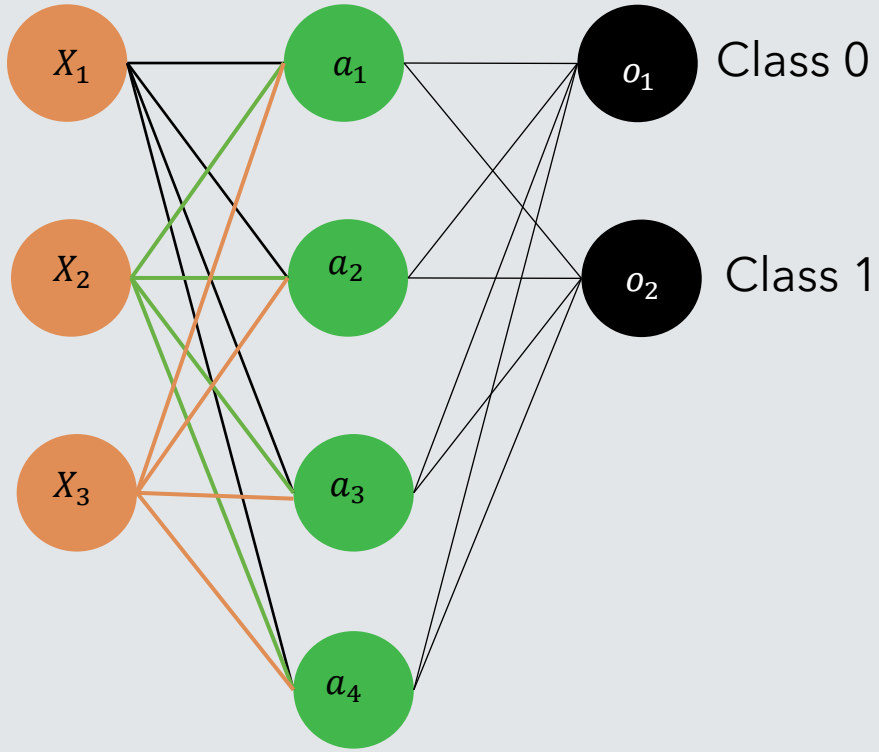
$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f\left(W_1^{(2)}a + b_1^{(2)}\right) = f(\hat{o}_1) = \frac{e^{\hat{o}_1}}{e^{\hat{o}_1}+e^{\hat{o}_2}}$; f is a softmax function
- $o_2 = f\left(W_2^{(2)}a + b_2^{(2)}\right) = f(\hat{o}_2) = \frac{e^{\hat{o}_2}}{e^{\hat{o}_1}+e^{\hat{o}_2}}$; f is a softmax function
- $l = -(y\log(o_2) + (1-y)\log(o_1))$; where y is the true label and $\hat{y}$ is the predicated label



Feedforward Neural Network – Forward Propagation (Classification)





Feedforward Neural Network

Wait!

How do we calculate all of the weights and biases of the neural network?

Feedforward Neural Network – Backpropagation (Regression)

- $a_1 = \left(W_1^{(1)} X + b_1^{(1)}\right)$

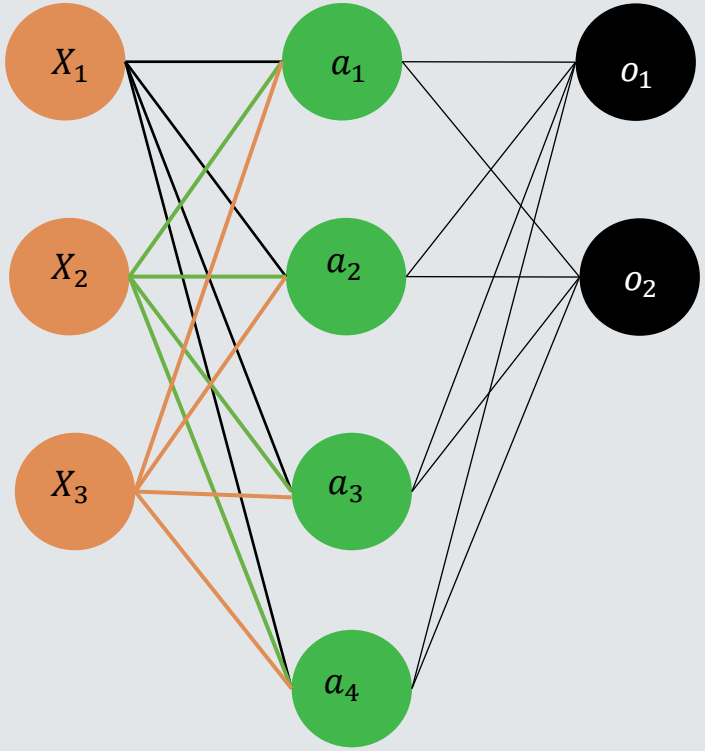
$$W_1^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{21}^{(1)} & W_{31}^{(1)} \end{bmatrix} \in \mathbb{R}^{1\times 3}$$

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$

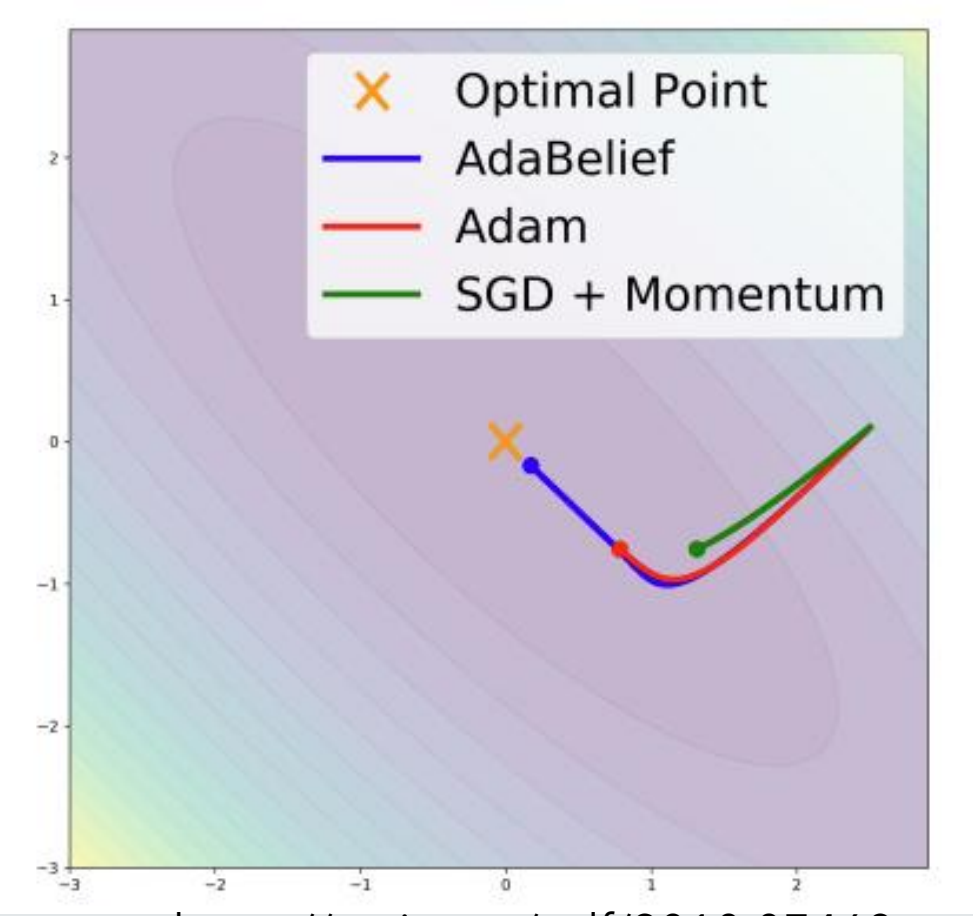
- $o_1 = W_1^{(2)} a + b_1^{(2)}; o_2 = W_2^{(2)} a + b_2^{(2)}$
- $l = \frac{1}{2}[(y_1 - o_1)^2 + (y_2 - o_2)^2]$
- How to update the weights and biases?

$$\min_{w_{11}^{(1)}, w_{21}^{(1)}, w_{31}^{(1)}, w_{12}^{(1)}, w_{22}^{(1)}, w_{32}^{(1)}, w_{13}^{(1)}, w_{23}^{(1)}, w_{33}^{(1)}, w_{11}^{(2)}, w_{21}^{(2)}, \ldots} l$$

- There are several optimizier that will be used:
- Stochastic Gradient Descent
- SGD with momentum
- Adam



Feedforward Neural Network – Backpropagation (Regression)



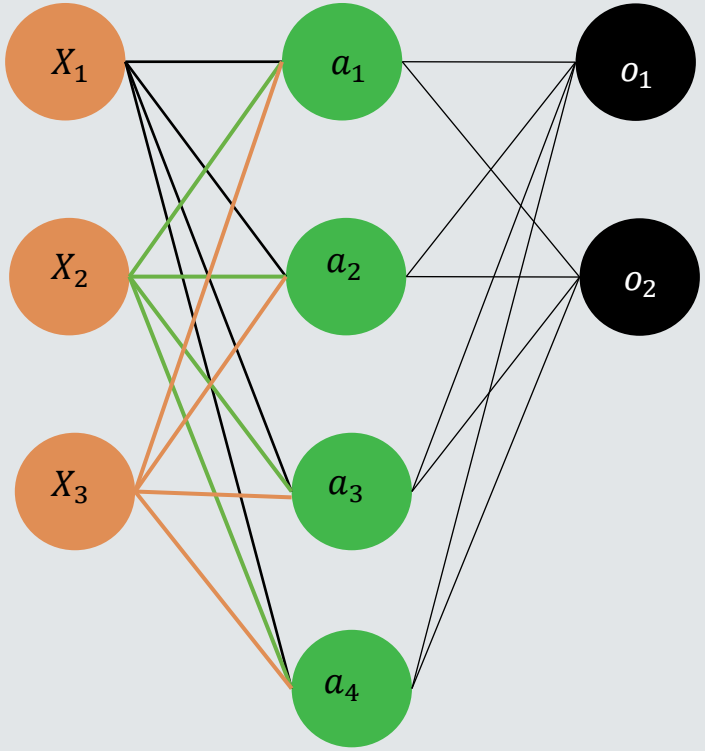
https://arxiv.org/pdf/2010.07468

Feedforward Neural Network – Backpropagation (Regression)

- $a_1 = \left(W_1^{(1)} X + b_1^{(1)}\right)$

$$W_1^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{21}^{(1)} & W_{31}^{(1)} \end{bmatrix} \in \mathbb{R}^{1\times 3}$$

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$



- $o_1 = W_1^{(2)} a + b_1^{(2)}; o_2 = W_2^{(2)} a + b_2^{(2)}$
- $l = \frac{1}{2}[(\mathrm{y}_1 - \mathrm{o}_1)^2 + (\mathrm{y}_2 - \mathrm{o}_2)^2]$
- $W_{11}^{(2)} = W_{11}^{(2)} - \eta \frac{\partial l}{\partial W_{11}^{(2)}} = W_{11}^{(2)} - \eta \frac{\partial o_1}{\partial W_{11}^{(2)}} \frac{\partial l}{\partial o_1}$

$$= W_{11}^{(2)} - \eta \frac{\partial o_1}{\partial W_{11}^{(2)}} \frac{\partial}{\partial o_1} \left(\frac{1}{2}[(\mathrm{y}_1 - \mathrm{o}_1)^2 + (\mathrm{y}_2 - \mathrm{o}_2)^2]\right)$$

$$= W_{11}^{(2)} - \eta \frac{\partial}{\partial W_{11}^{(2)}} \left[W_{11}^{(2)} a_1 + W_{21}^{(2)} a_2 + W_{31}^{(2)} a_3 + W_{41}^{(2)} a_4\right] (-(y_1 - o_1)) = W_{11}^{(2)} - \eta a_1$$

The same procedure will be done for all of the weights and biases in the neural network.

Feedforward Neural Network – Forward Propagation (Regression)

- $a_1 = f\left(W_1^{(1)}X + b_1\right) = \begin{cases} \max\left(0, W_1^{(1)}X + b_1\right); f\ is\ a\ relu\ function \\ \frac{1}{1+e^{-W_1^{(1)}X+b_1}}; f\ is\ a\ sigmoid\ function \\ \frac{e^{W_1^{(1)}X+b_1}-e^{-W_1^{(1)}X+b_1}}{e^{W_1^{(1)}X+b_1}+e^{-W_1^{(1)}X+b_1}}; f\ is\ a\ \tanh\ function \end{cases}$

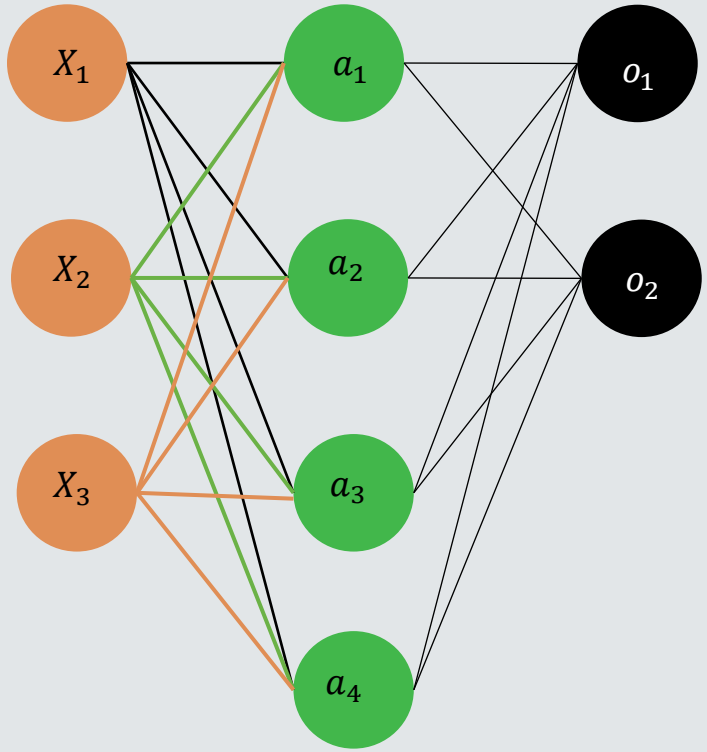
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1\times 3}$$

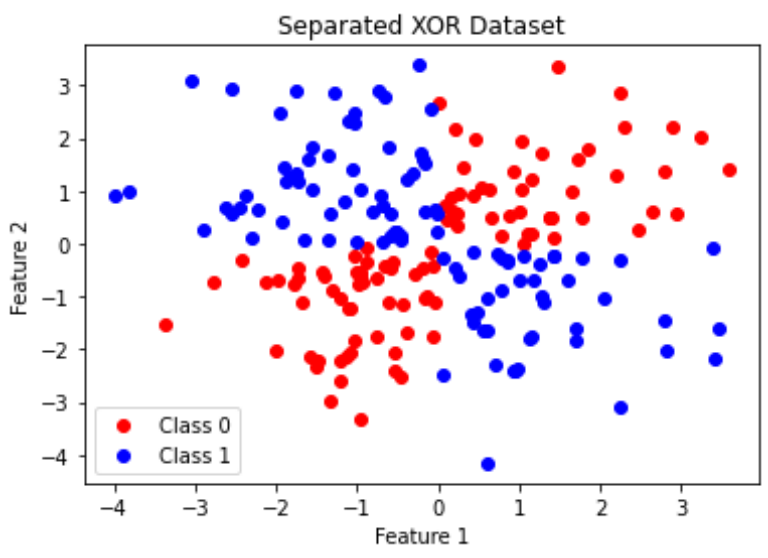
$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

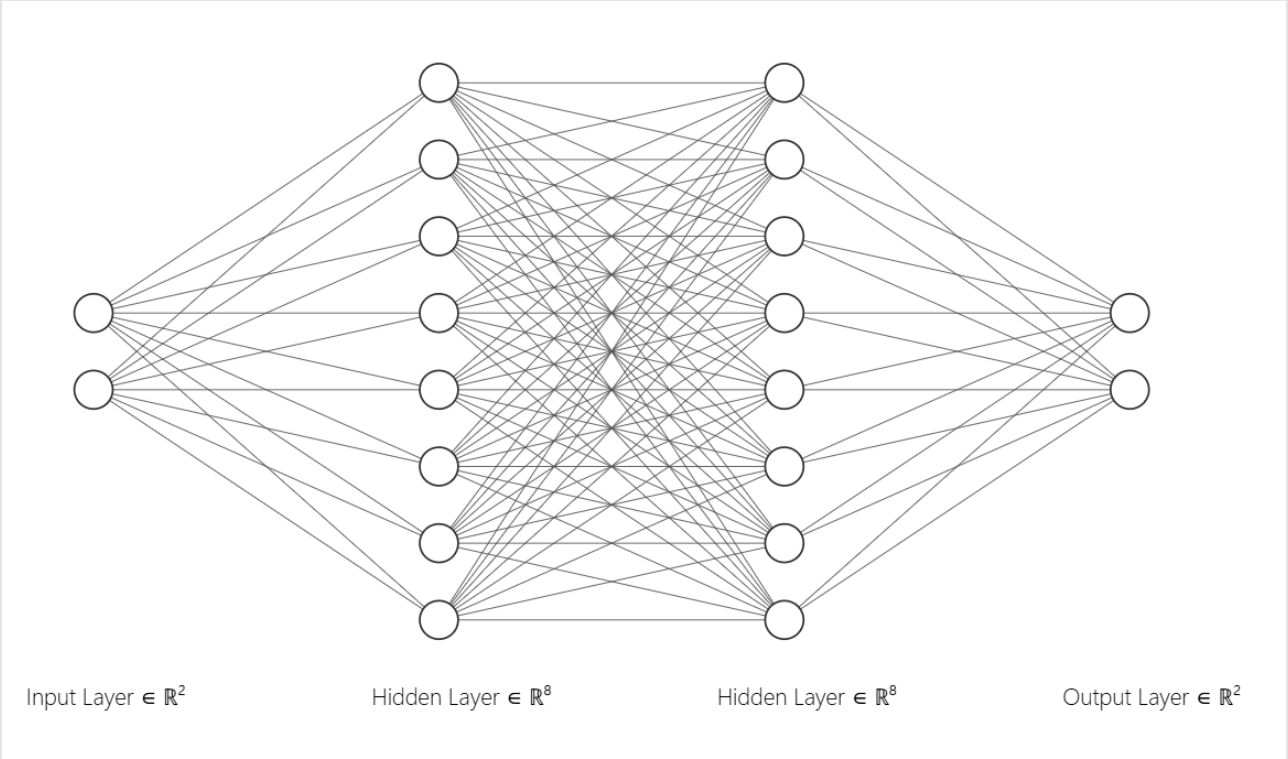
$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f\left(W_1^{(2)}a + b_1^{(2)}\right) = W_1^{(2)}a + b_1^{(2)}$; f is a linear function
- $o_2 = f\left(W_2^{(2)}a + b_2^{(2)}\right) = W_2^{(2)}a + b_2^{(2)}$; f is a linear function
- $l = \frac{1}{2}[(y_1 - o_1)^2 + (y_2 - o_2)^2]$ (MSE); where y_1, y_2 is the true value and o_1, o_2 is the predicted value

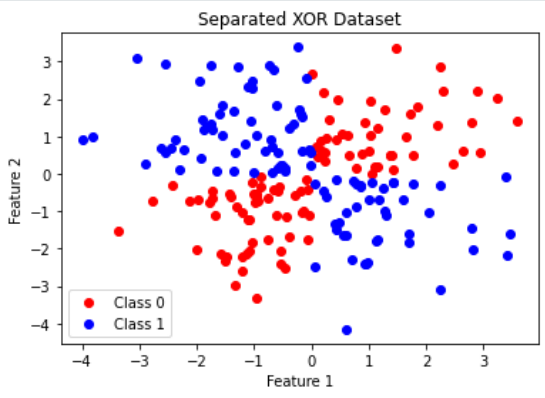


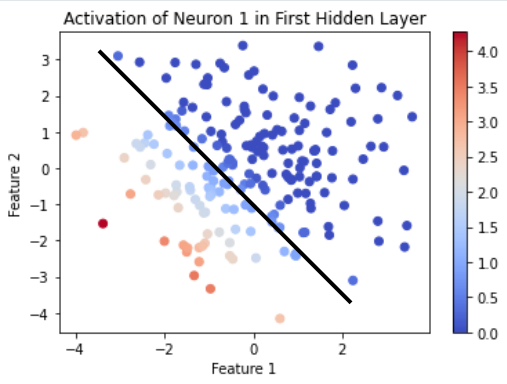
Feedforward Neural Network

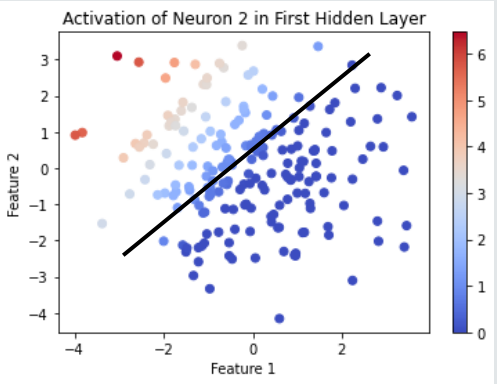


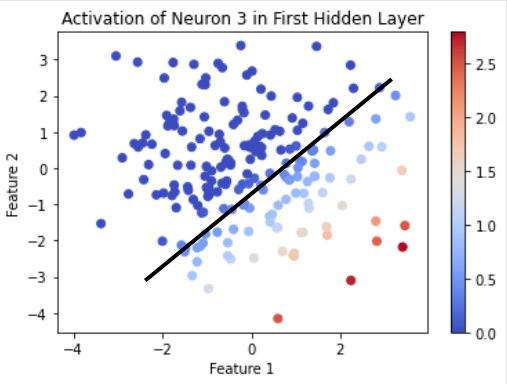


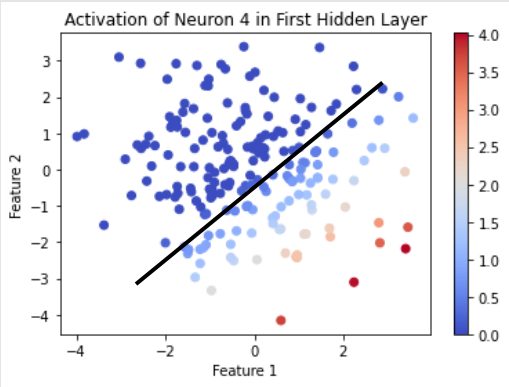
Feedforward Neural Network

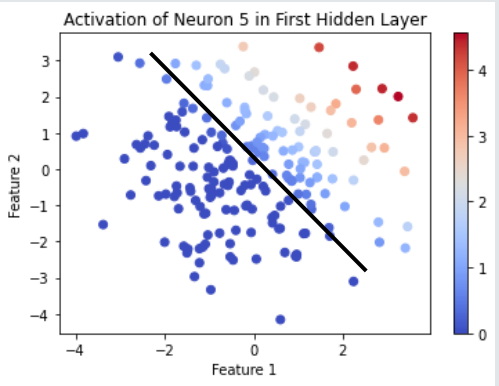


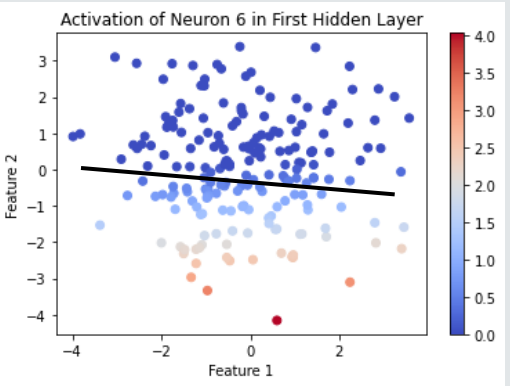


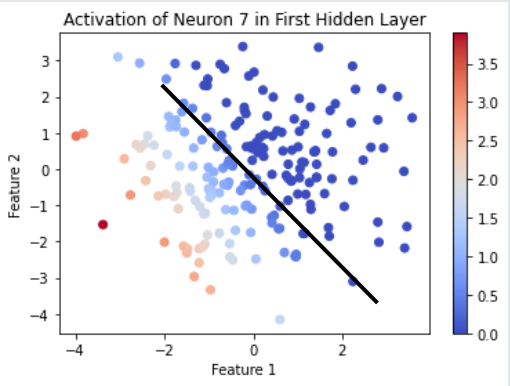


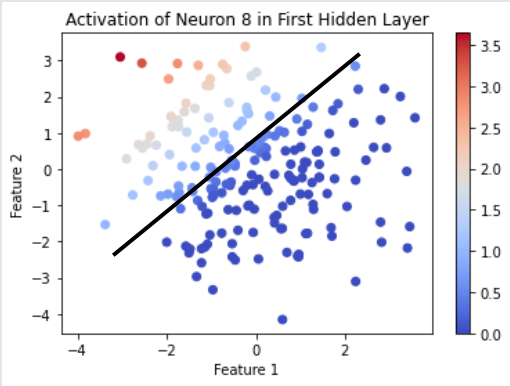


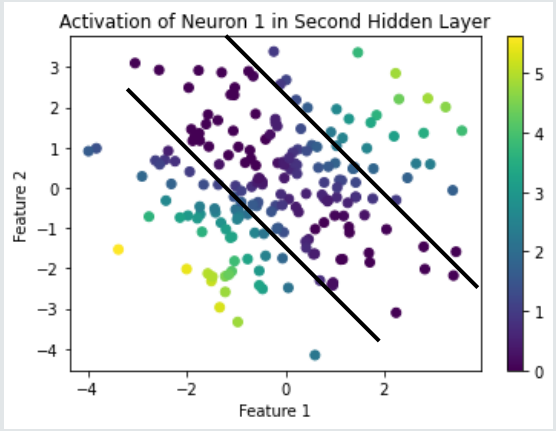


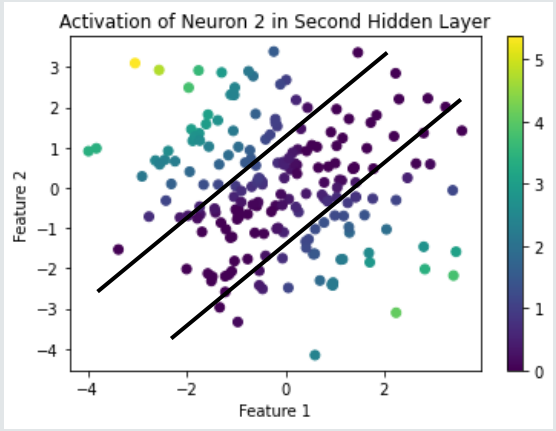


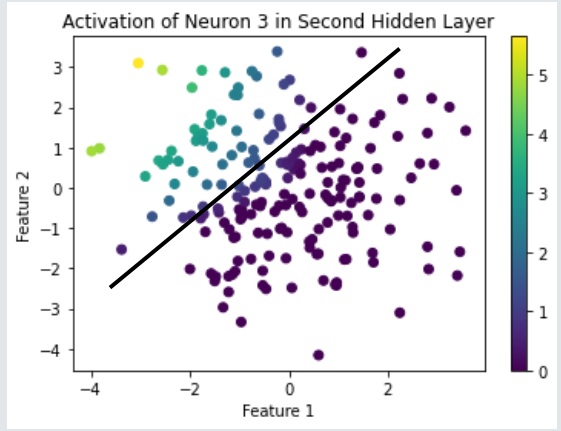


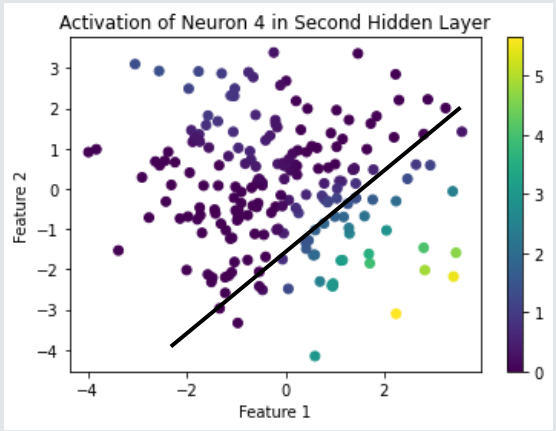


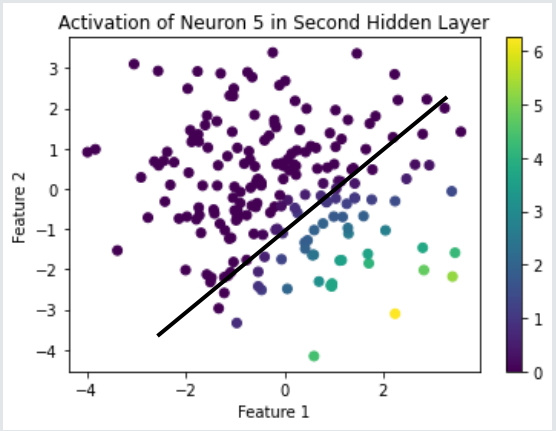


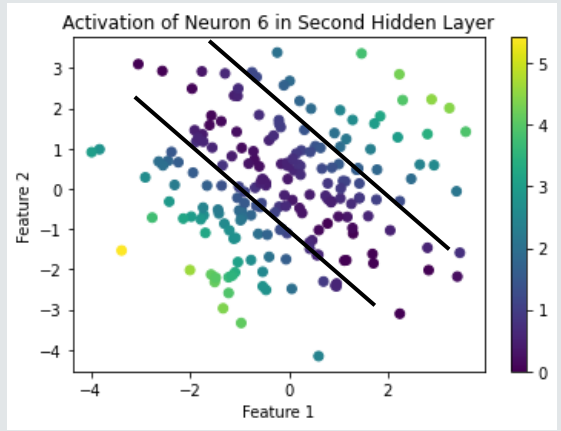

Activation of Neuron 1 in Second Hidden Layer
Feature 1
Feature 2

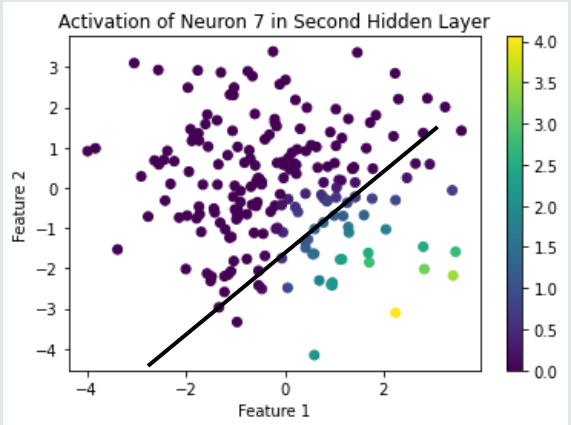

Activation of Neuron 2 in Second Hidden Layer
Feature 1
Feature 2

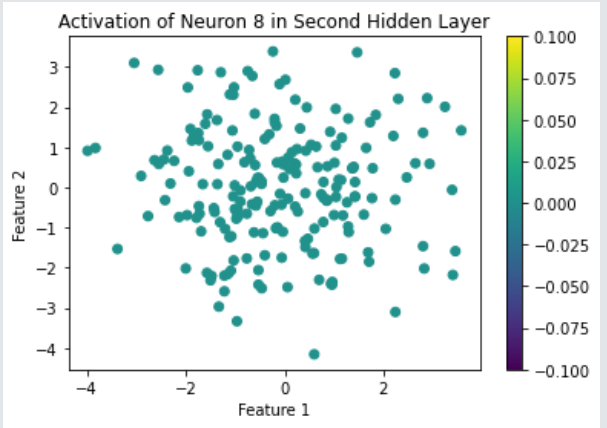

Activation of Neuron 3 in Second Hidden Layer
Feature 1
Feature 2


Activation of Neuron 4 in Second Hidden Layer
Feature 1
Feature 2

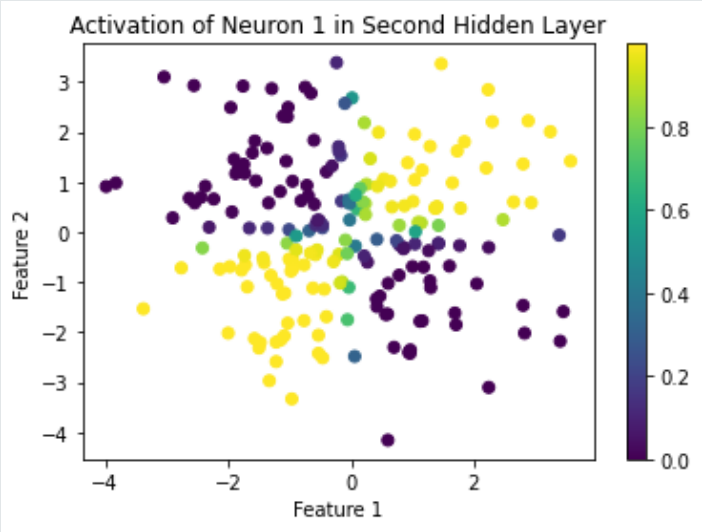

Activation of Neuron 5 in Second Hidden Layer
Feature 1
Feature 2

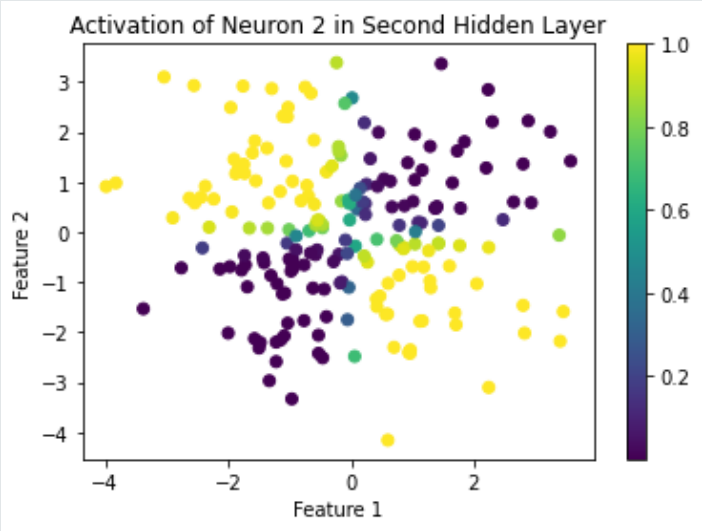

Activation of Neuron 6 in Second Hidden Layer
Feature 1
Feature 2

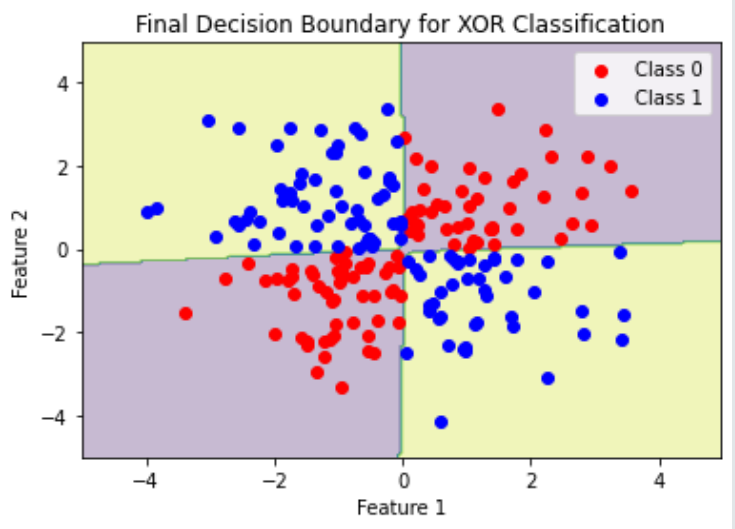

Activation of Neuron 7 in Second Hidden Layer
Feature 1
Feature 2


Activation of Neuron 8 in Second Hidden Layer
Feature 1
Feature 2

Feedforward Neural Network







Feedforward Neural Network – Weight Initialization

- Random Initialization: weights are random initialization, typically from a uniform or normal distribution. The issue with this initialization that it can leaf to large or small gradients that can cause slow convergence or divergence.

  Advantages: Allows neurons to learn different features

  Disadvantages: Without further scaling, it can lead to large or small gradients, causing small convergence or divergence.

- He Initialization: specifically designed for Relu activation

$$W \sim Normal\left(0, \sqrt{\frac{2}{n_{in}}}\right)$$

  Ensures that the variance of the weights does not shrink or explode.

- Xavier Initialization: or layer l, weights are initialized with a uniform or normal distribution scaled by

$$W \sim Uniform\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$$

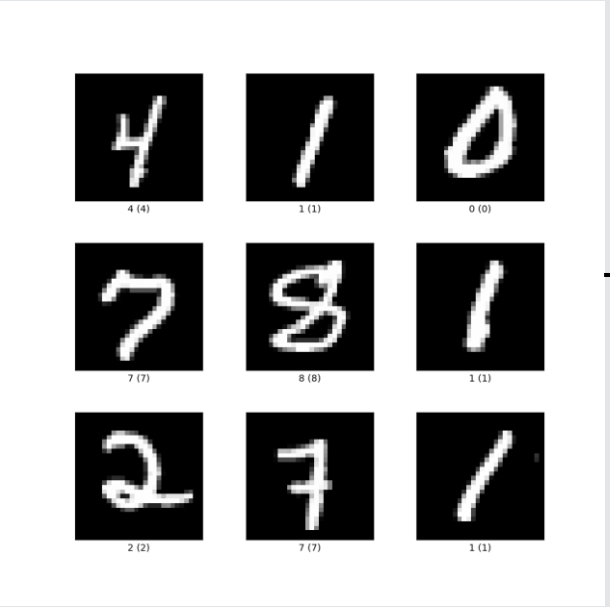
where n_{in} and n_{out} are the number of input and output units in the layer.

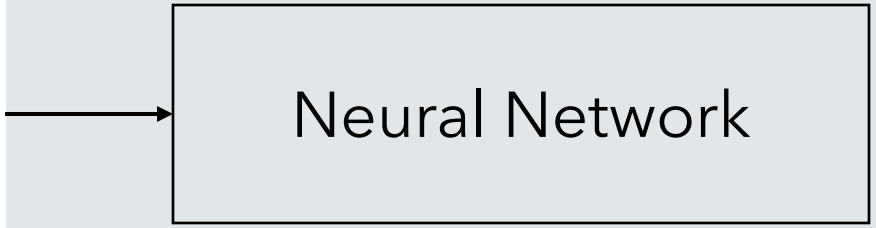
Balances the variance across layers, making it suitable for sigmoid and tanh activations.

Feedforward Neural Network – Wider vs Deeper

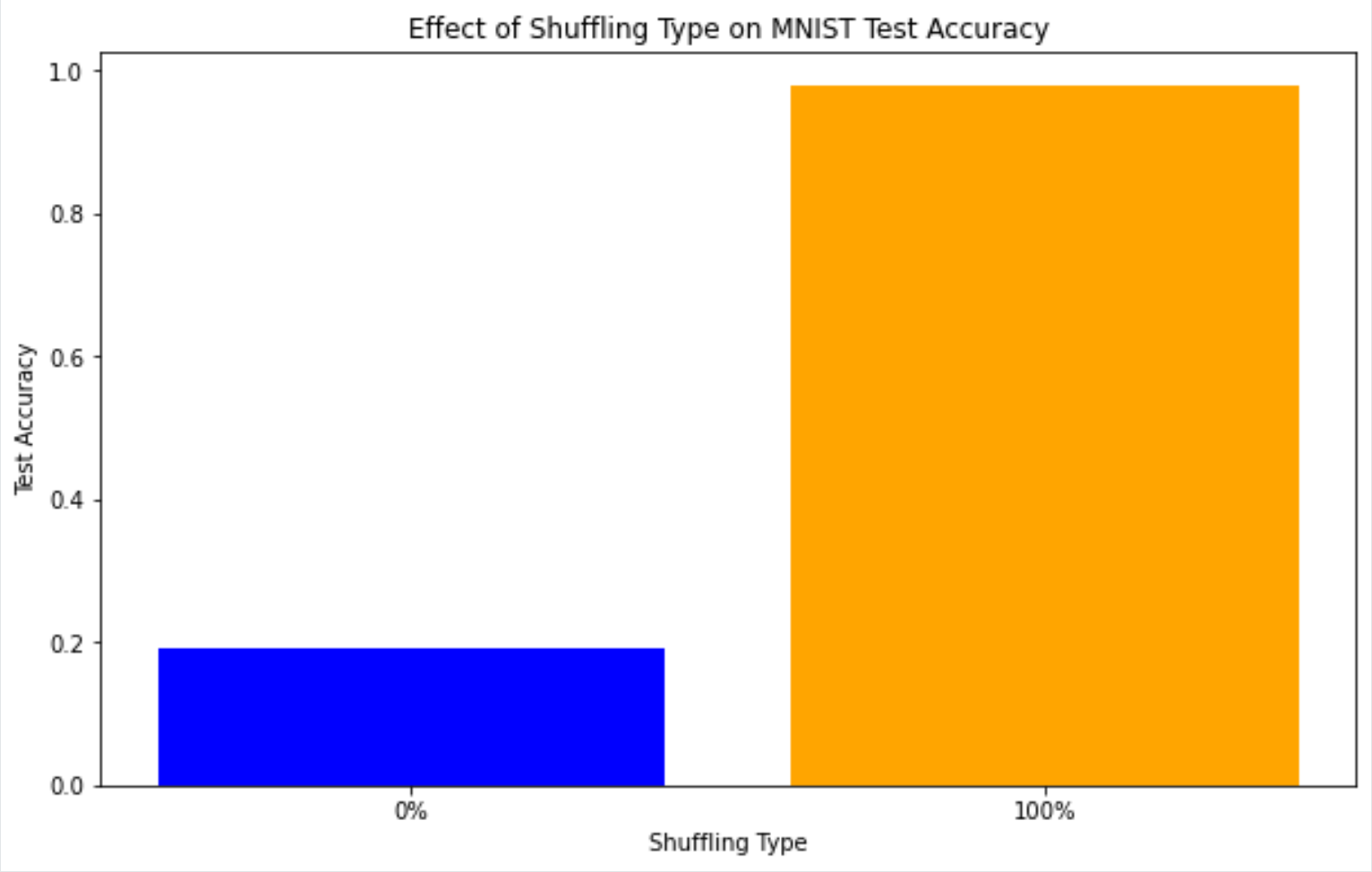
- **Are there functions that can be expressed by wide and shallow neural networks, that cannot be approximated by any narrow neural network, unless its depth is very large?**
- (Correct) On the other hand, if the answer is negative, then depth generally plays a more significant role than width for the expressive power of neural networks. [Proven in https://proceedings.mlr.press/v178/vardi22a/vardi22a.pdf]
- (Wrong) If the answer is positive, then width and depth, in principle, play an incomparable role in the expressive power of neural networks, as sometimes depth can be more significant, and sometimes width.

Feedforward Neural Network – MNIST

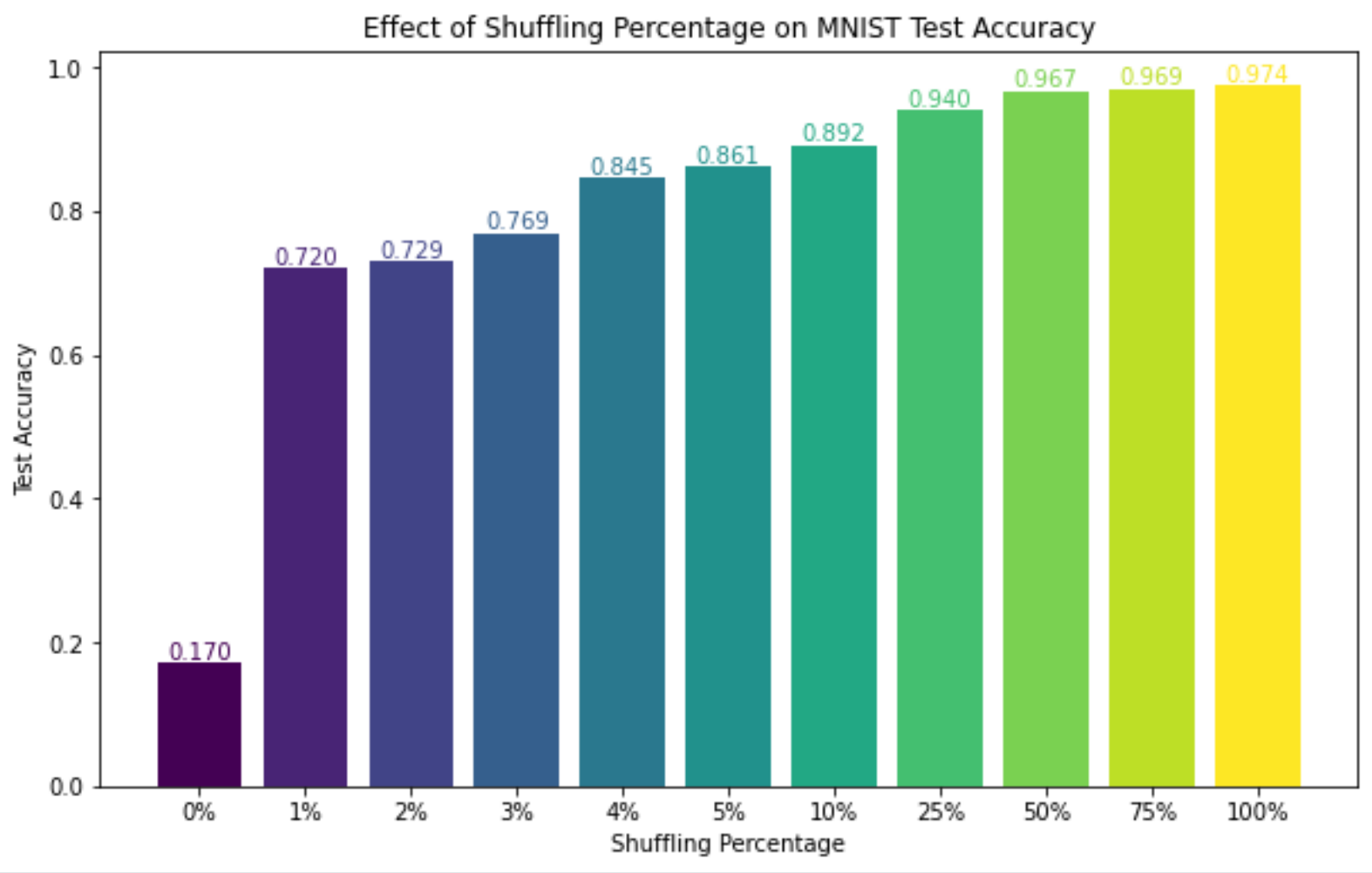




Feedforward Neural Network – MNIST



Feedforward Neural Network – MNIST

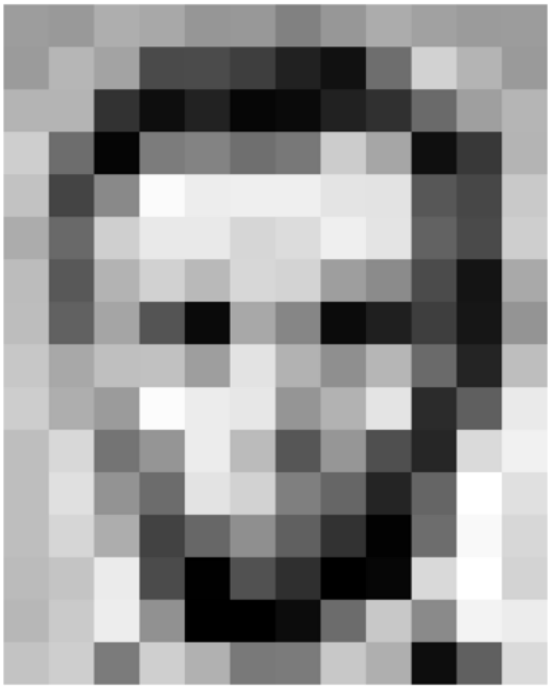


Lecture 10 (CNN,RNN)


CNN

What computers ‘see’: Images as Numbers

What you see



Input Image

What you both see

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	200	175	13	96	218

Input Image + values

What the computer "sees"

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	200	175	13	96	218

Pixel intensity values
(“pix-el”=picture-element)

Levin *Image Processing & Computer Vision*

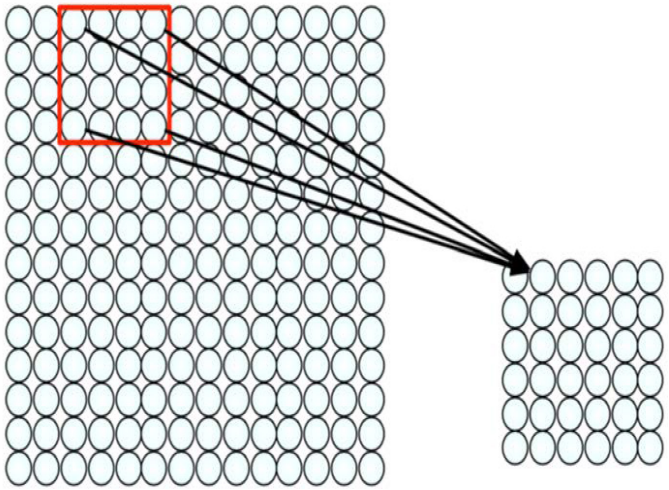
An image is just a matrix of numbers [0,255]. i.e., 1080x1080x3 for an RGB image.

Can I just do classification on the 1,166400-long image vector directly?

No. Instead: exploit image spatial structure. Learn patches. Build them up

This slide is taken from Manolis Kellis

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

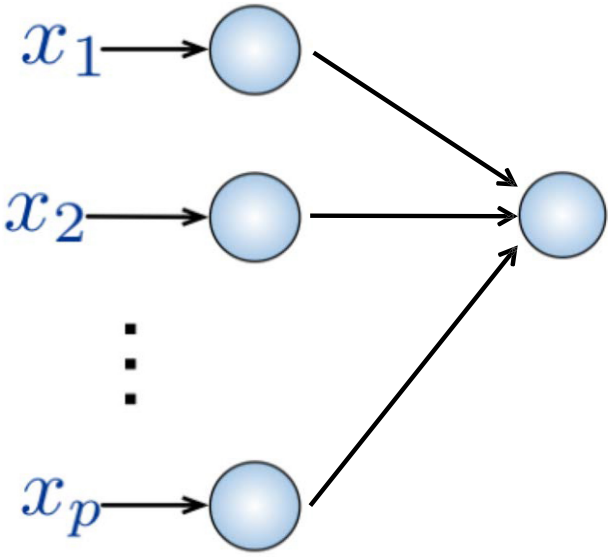
2) Use **multiple filters** to extract different features

3) Spatially share parameters of each filter

This slide is taken from Manolis Kellis

Fully Connected Neural Network

Input:
- 2D image
- Vector of pixel values

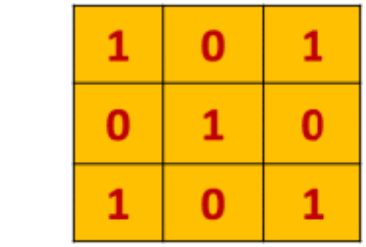


Fully Connected:
- Each neuron in hidden layer connected to all neurons in input layer
- No spatial information
- Many, many parameters

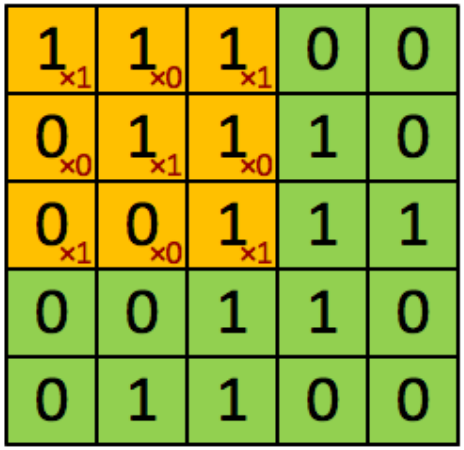
Key idea: Use **spatial structure** in input to inform architecture of the network

This slide is taken from Manolis Kellis

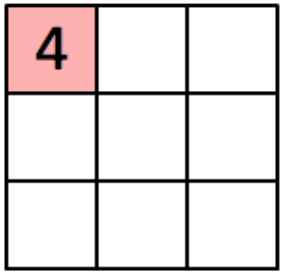
Convolution operation is element wise multiply and add



Filter / Kernel



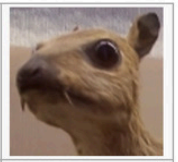
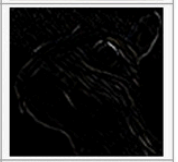
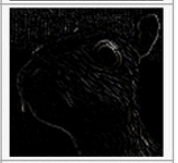
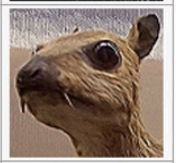
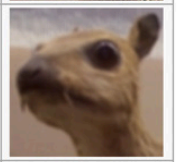
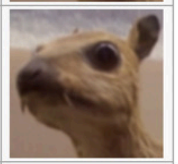
Image



Convolved Feature

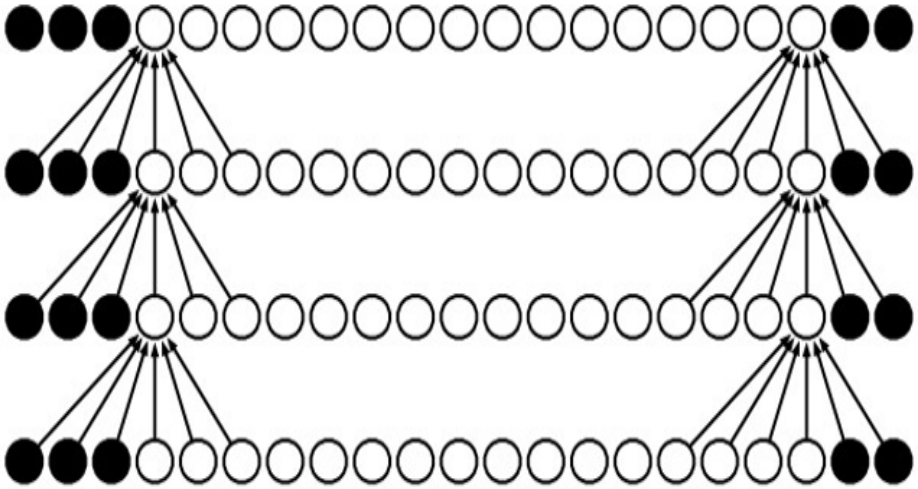
This slide is taken from Manolis Kellis

Simple Kernels / Filters

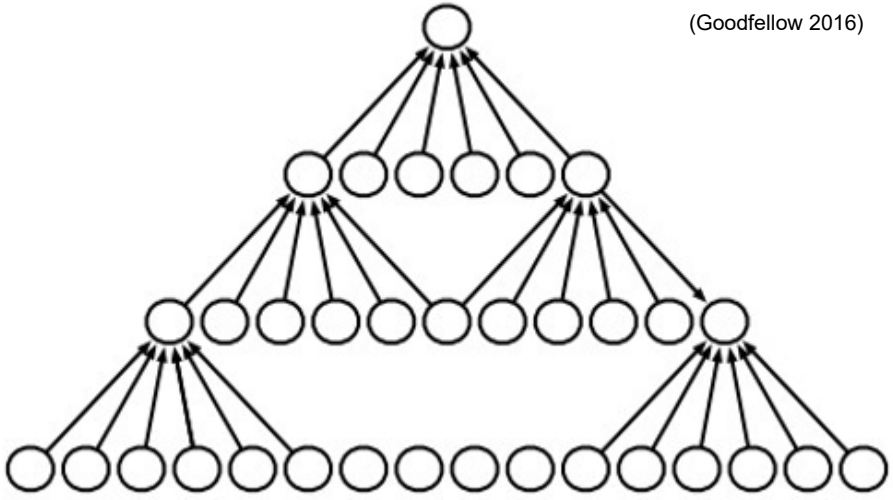
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

This slide is taken from Manolis Kellis

Zero Padding Controls Output Size



(Goodfellow 2016)



• **Same convolution**: zero pad input so output is same size as input dimensions

• **Valid-only convolution**: output only when entire kernel contained in input (shrinks output)

This slide is taken from Manolis Kellis

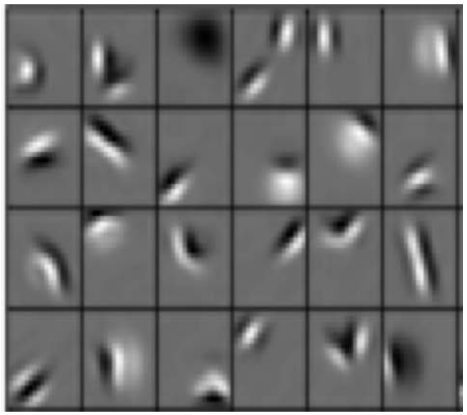
Key idea:

learn hierarchy of features

directly from the data

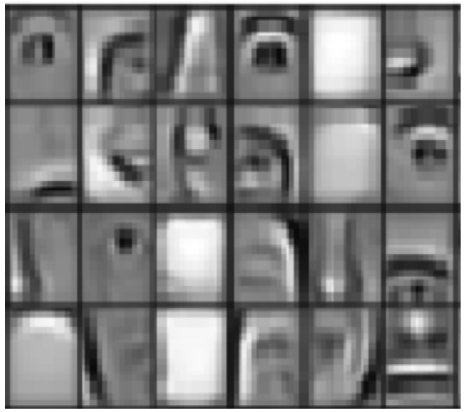
(rather than hand-engineering them)

Low level features



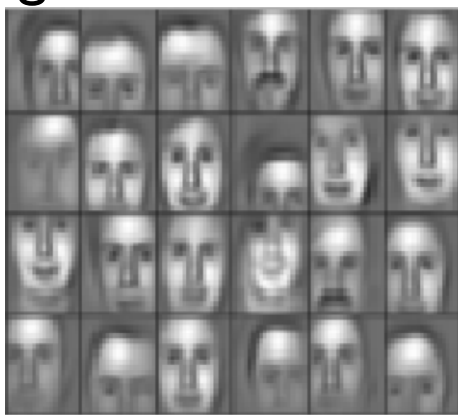
Edges, dark spots

Mid level features



Eyes, ears,nose

High level features



Facial structure

This slide is taken from Manolis Kellis

Lee+ *ICML* 2009

LeNet-5

- *Gradient Based Learning Applied To Document Recognition - Y. Lecun, L. Bottou, Y. Bengio, P. Haffner; 1998*
- Helped establish how we use CNNs today
- Replaced manual feature extraction

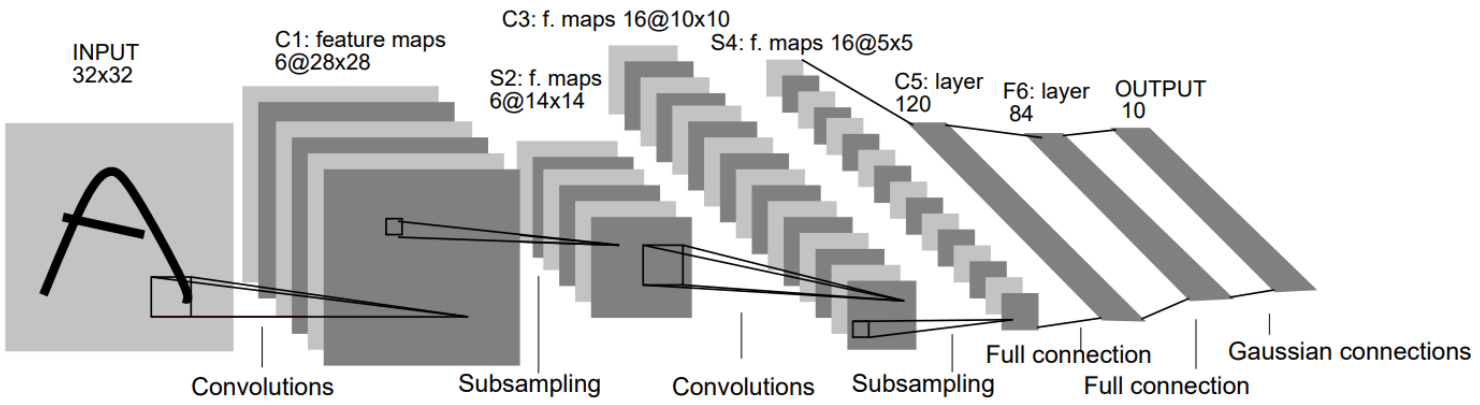
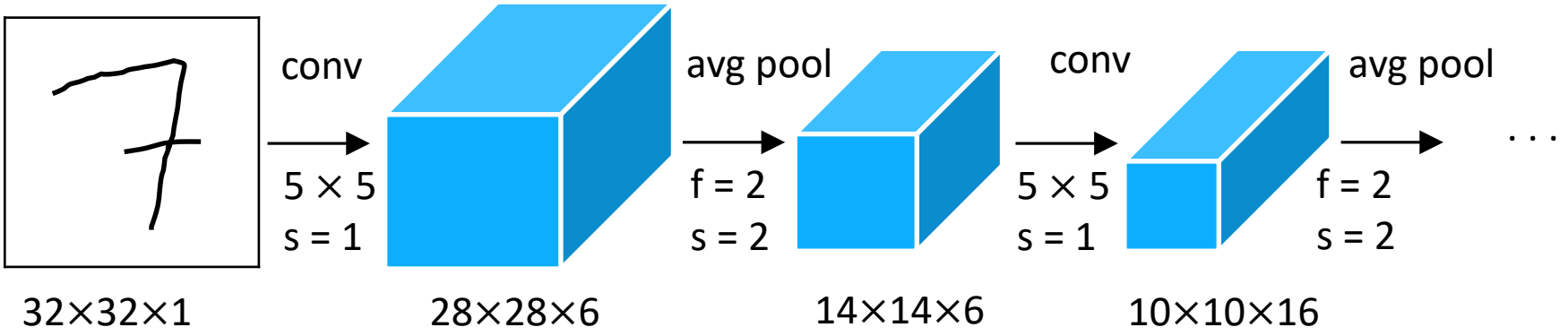


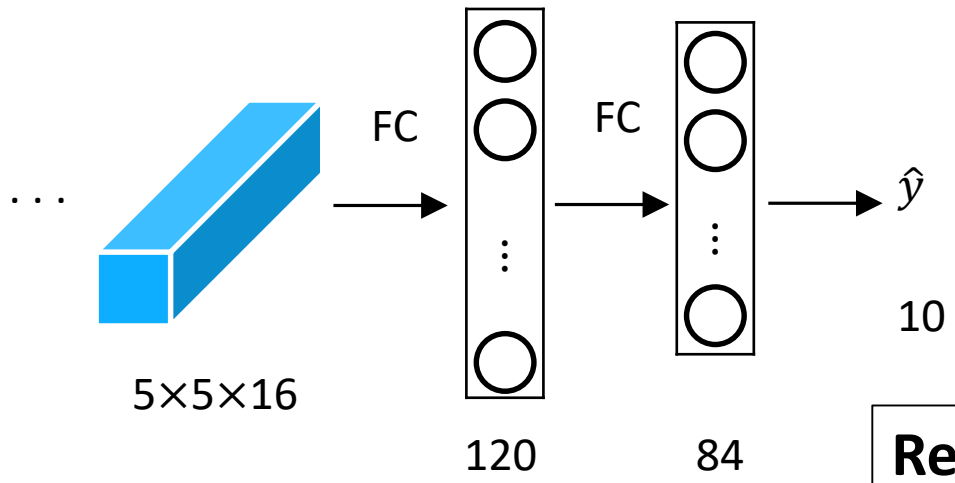
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

This slide is taken from Manolis Kellis

[LeCun et al., 1998]

LeNet-5



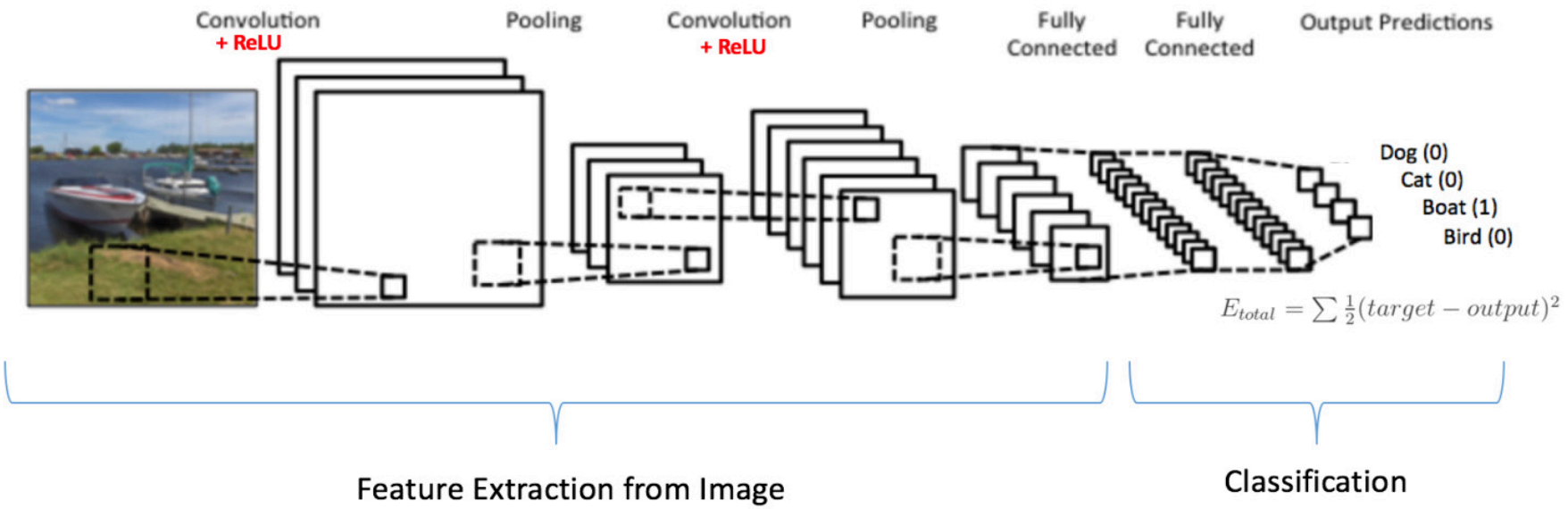


Reminder:
Output size = (N+2P-F)/stride + 1

This slide is taken from Andrew Ng

[LeCun et al., 1998]

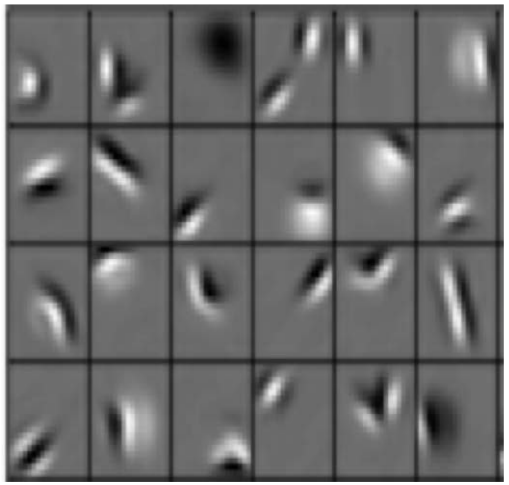
An image classification CNN



This slide is taken from Manolis Kellis

Representation Learning in Deep CNNs

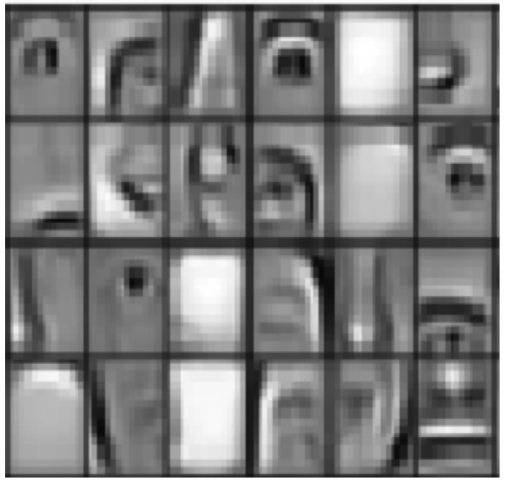
Low level features



Edges, dark spots

Conv Layer 1

Mid level features



Eyes, ears,nose

Conv Layer 2

High level features

Facial structure

Conv Layer 3

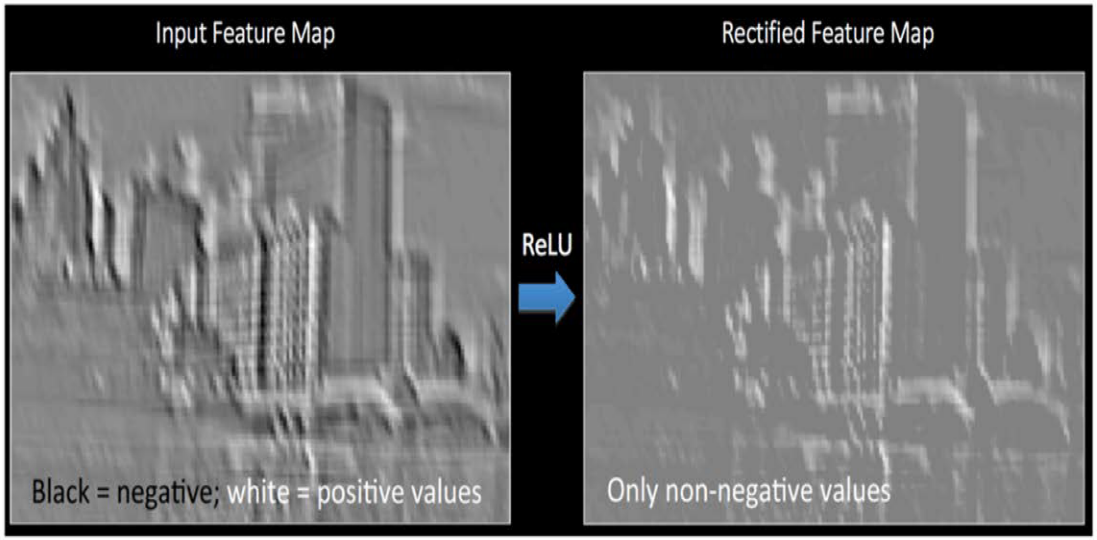
This slide is taken from Manolis Kellis

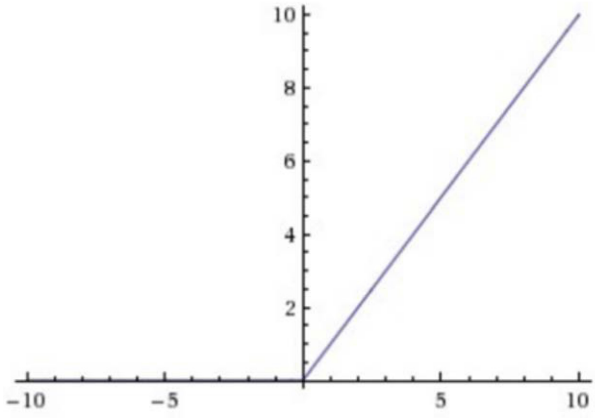
Lee+ *ICML* 2009

Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero.
- **Non-linear operation**

Rectified Linear Unit (ReLU)





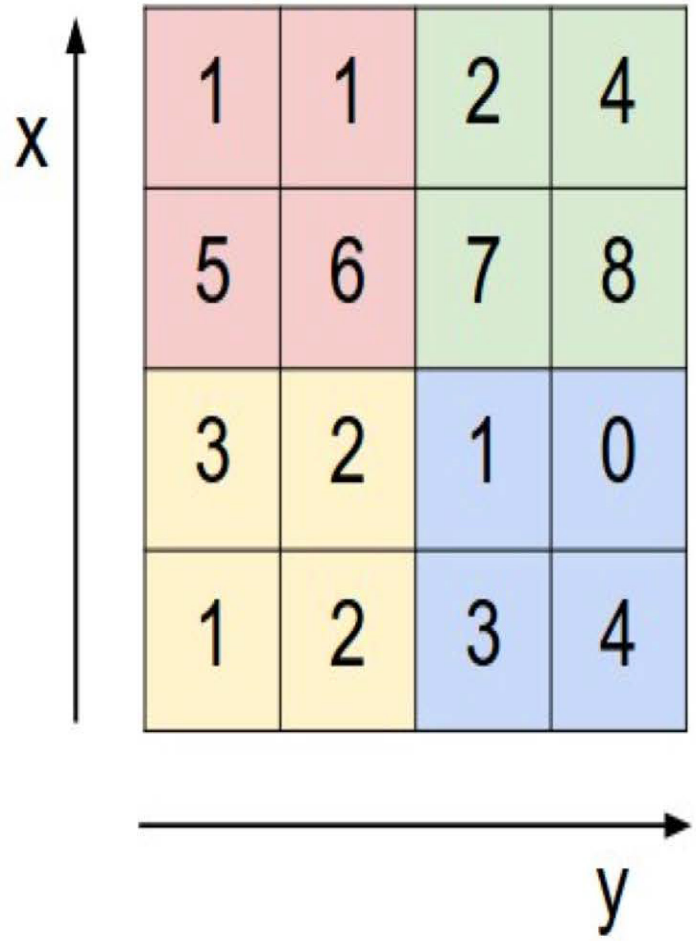
$$g(z) = \max(0, z)$$

`tf.keras.layers.ReLU`

This slide is taken from Manolis Kellis

Karn *Intuitive CNNs*

Pooling



max pool with 2x2 filters and stride 2

```
tf.keras.layers.MaxPool2D(
    pool_size=(2,2),
    strides=2
)
```

6	8
3	4

1) Reduced dimensionality
2) Spatial invariance

Max Pooling, average pooling

This slide is taken from Manolis Kellis

How can computers recognize objects?

Challenge:

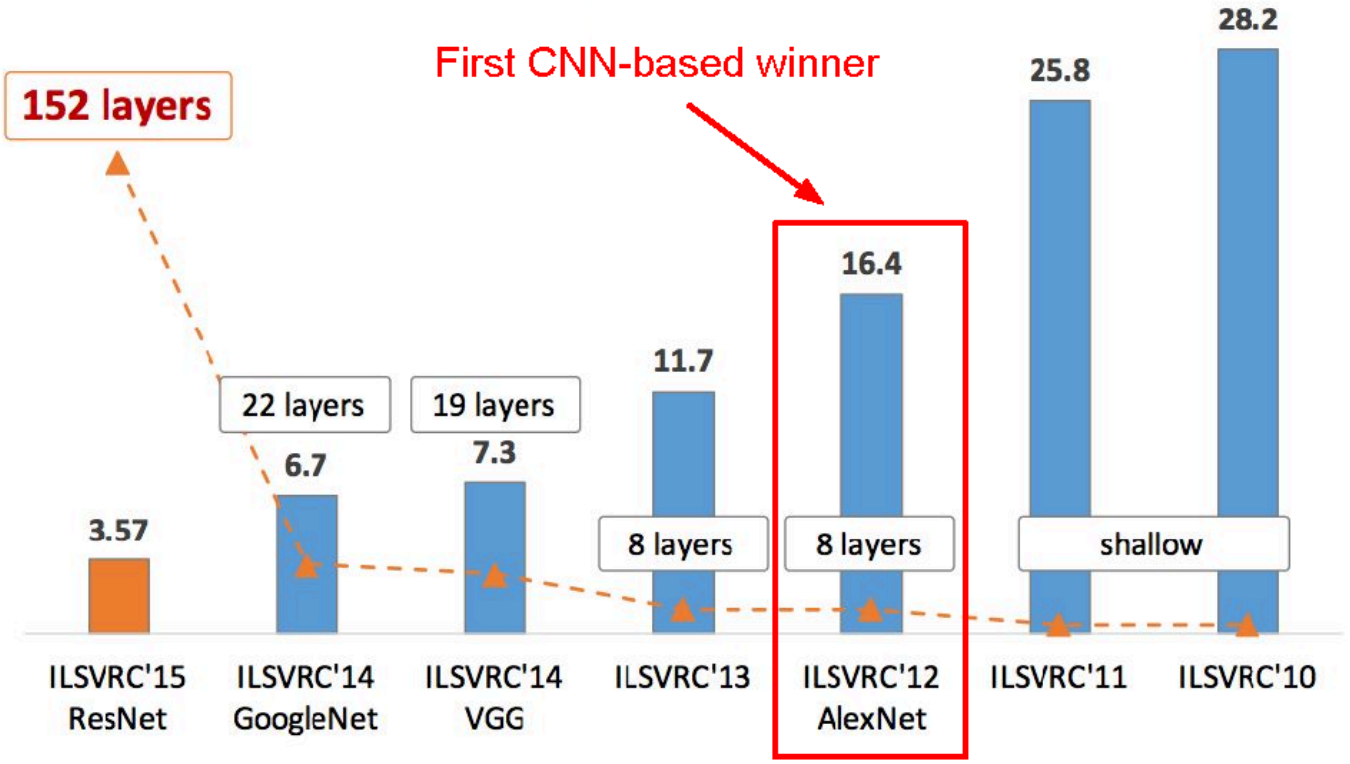
- Objects can be anywhere in the scene, in any orientation, rotation, color hue, etc.
- How can we overcome this challenge?

Answer:

- Learn a ton of features (millions) from the bottom up
- Learn the convolutional filters, rather than pre-computing them

This slide is taken from Manolis Kellis

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

AlexNet

- *ImageNet Classification with Deep Convolutional Neural Networks - Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; 2012*
- Facilitated by GPUs, highly optimized convolution implementation and large datasets (ImageNet)
- Large CNN
- Has 60 Million parameter compared to 60k parameter of LeNet-5

This slide is taken from Manolis Kellis

[**Krizhevsky et al.**, **2012**]

AlexNet

Architecture
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

- Input: 227x227x3 images (224x224 before padding)
- First layer: 96 11x11 filters applied at stride 4

- **Output volume size?**

  (N-F)/s+1 = (227-11)/4+1 = 55 -> [55x55x96]

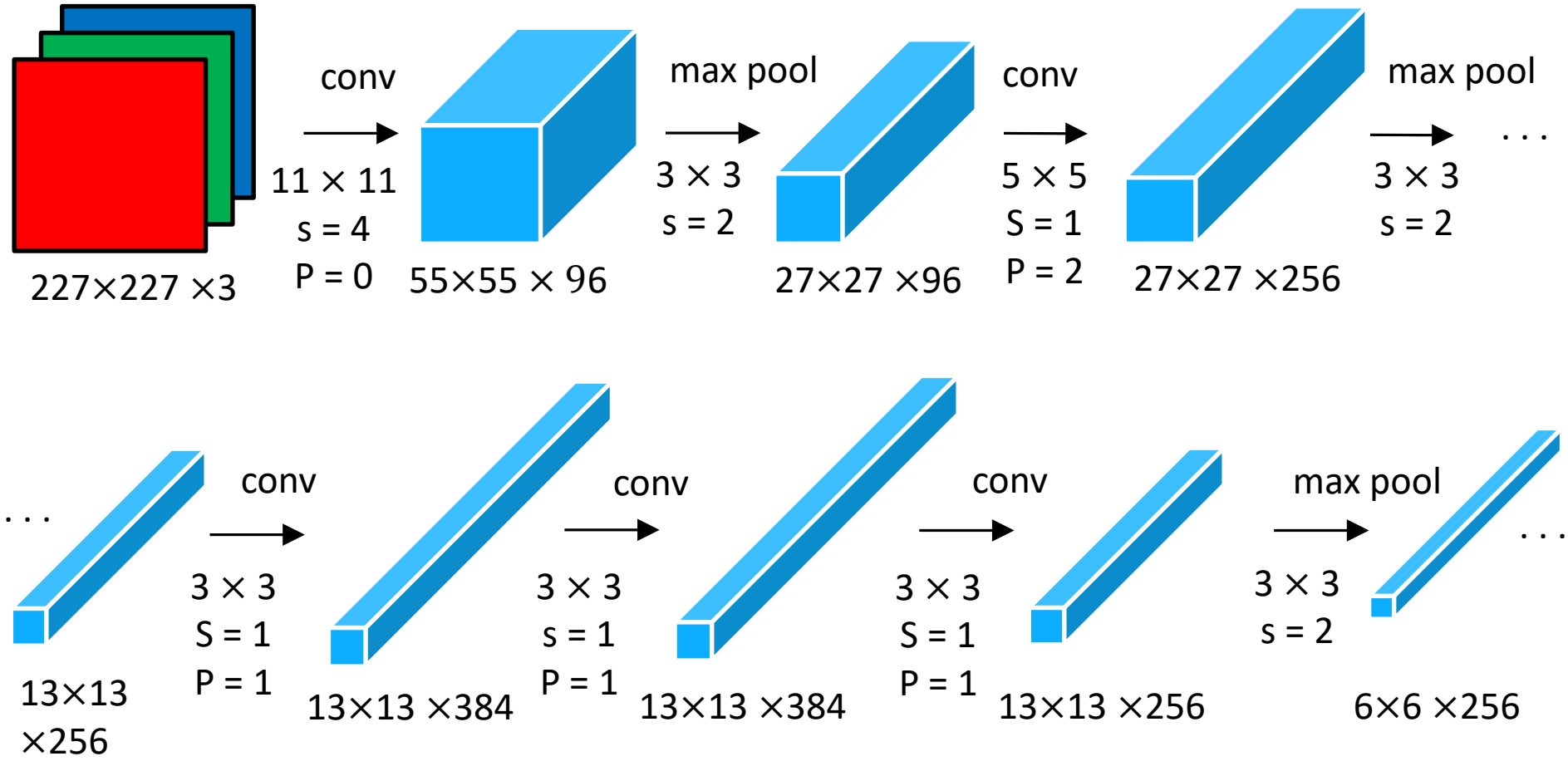
- **Number of parameters in this layer?**

  (11*11*3)*96 = 35K

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Krizhevsky et al., 2012]

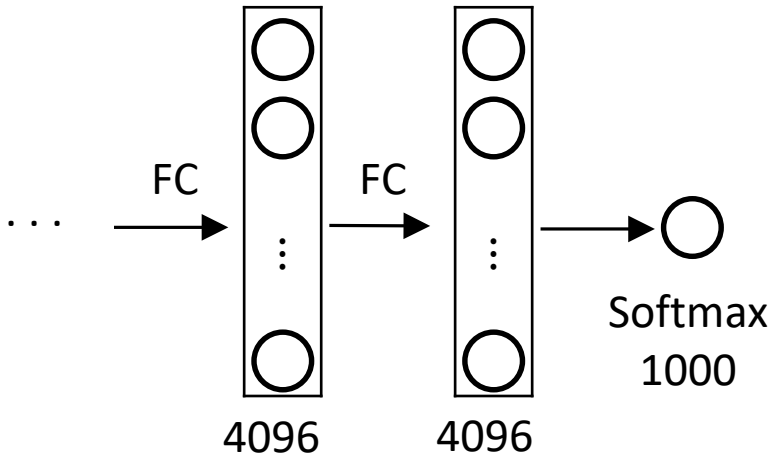
AlexNet



This slide is taken from Andrew Ng

[Krizhevsky et al., 2012]

AlexNet



This slide is taken from Andrew Ng

[**Krizhevsky et al.**, **2012**]

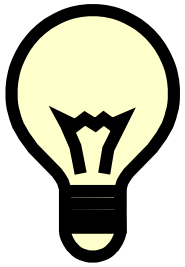
AlexNet

Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[**Krizhevsky et al., 2012**]



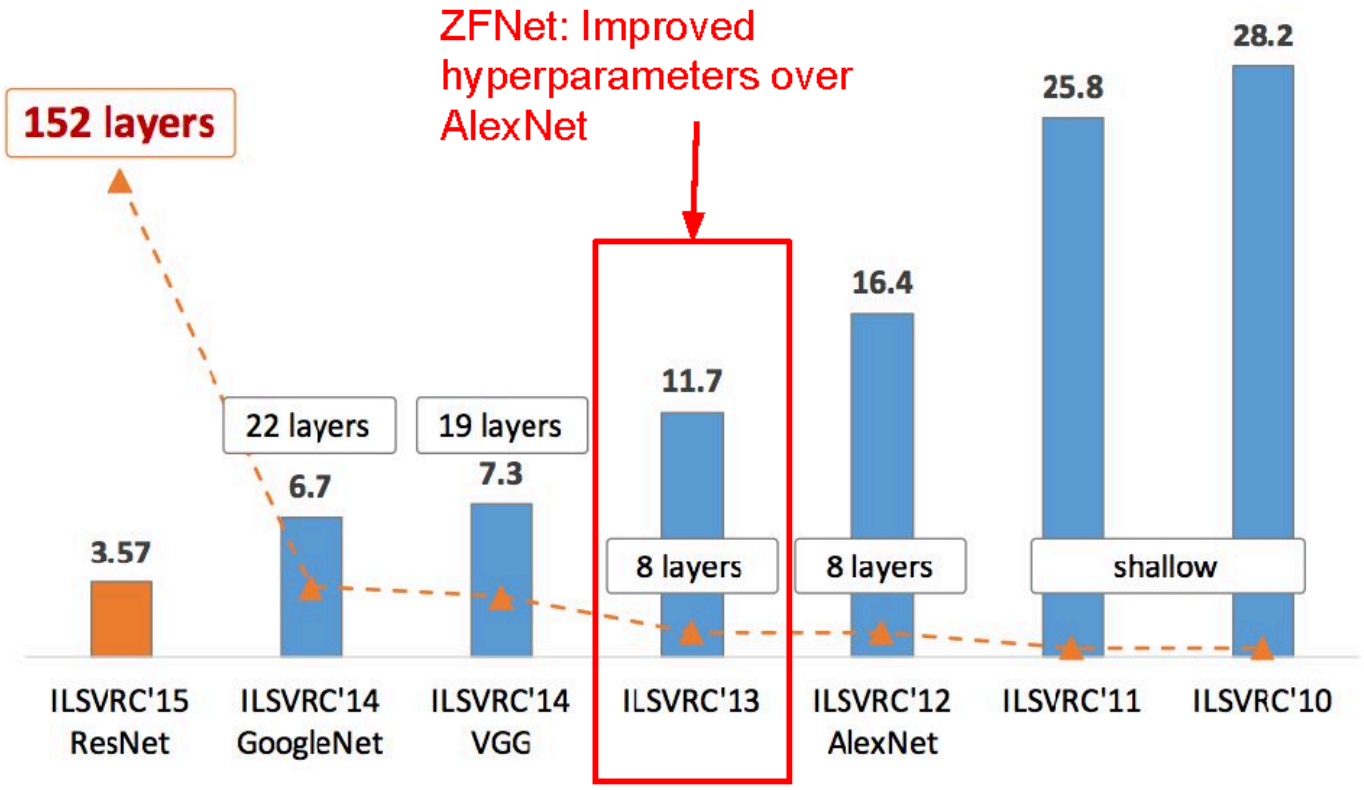
AlexNet

AlexNet was the coming out party for CNNs in the computer vision community. This was **the first time a model performed so well on a historically difficult ImageNet dataset**. This paper illustrated the benefits of CNNs and backed them up with record breaking performance in the competition.

This slide is taken from Manolis Kellis

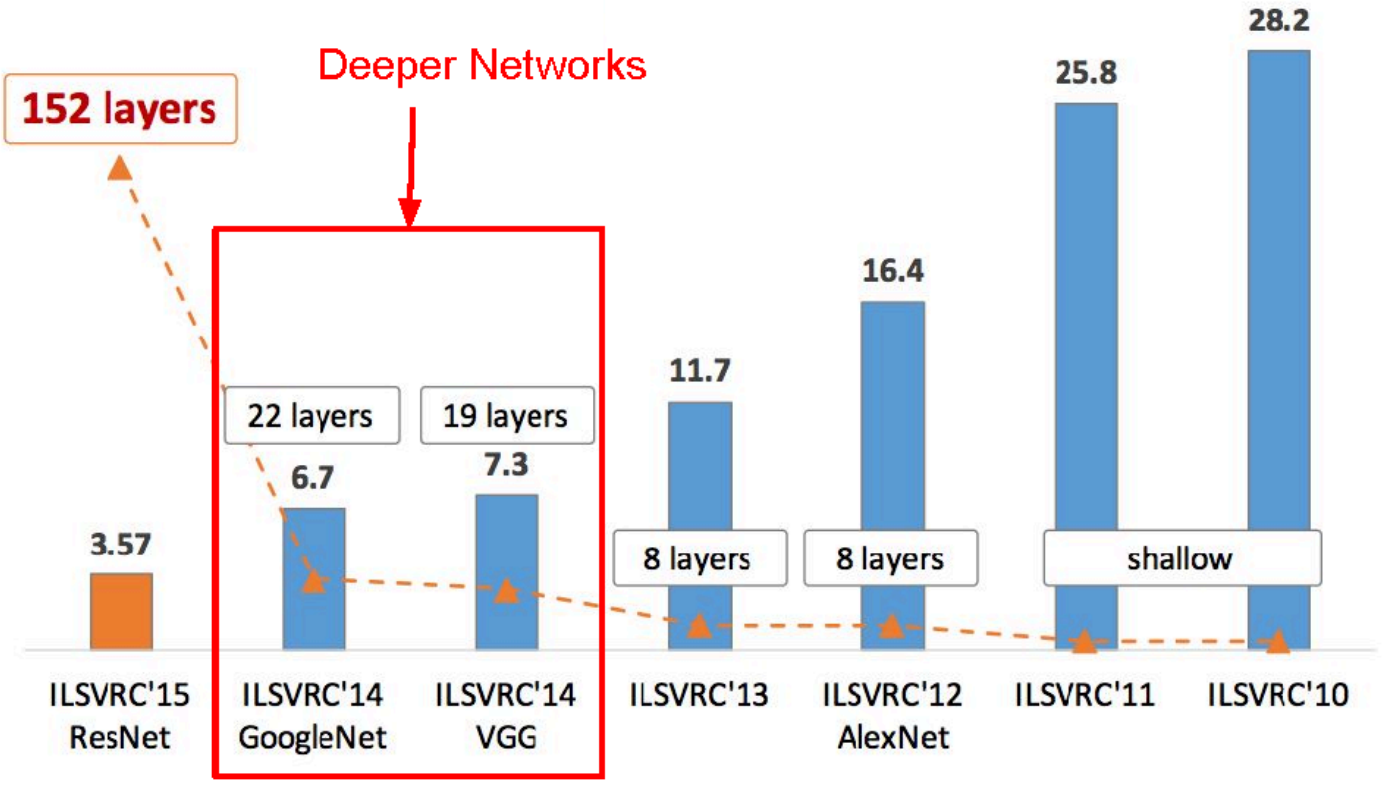
[**Krizhevsky et al., 2012**]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

Input
3x3 conv, 64
3x3 conv, 64
Pool 1/2
3x3 conv, 128
3x3 conv, 128
Pool 1/2
3x3 conv, 256
3x3 conv, 256
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
FC 4096
FC 4096
FC 1000
Softmax

VGGNet

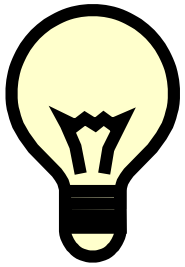
- **Smaller filters**
  Only 3x3 CONV filters, stride 1, pad 1
  and 2x2 MAX POOL , stride 2

- **Deeper network**
  AlexNet: 8 layers
  VGGNet: 16 - 19 layers

- ZFNet: 11.7% top 5 error in ILSVRC'13
- VGGNet: 7.3% top 5 error in ILSVRC'14

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Simonyan and Zisserman, 2014]



VGGNet

VGG Net reinforced the notion that **convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work**.

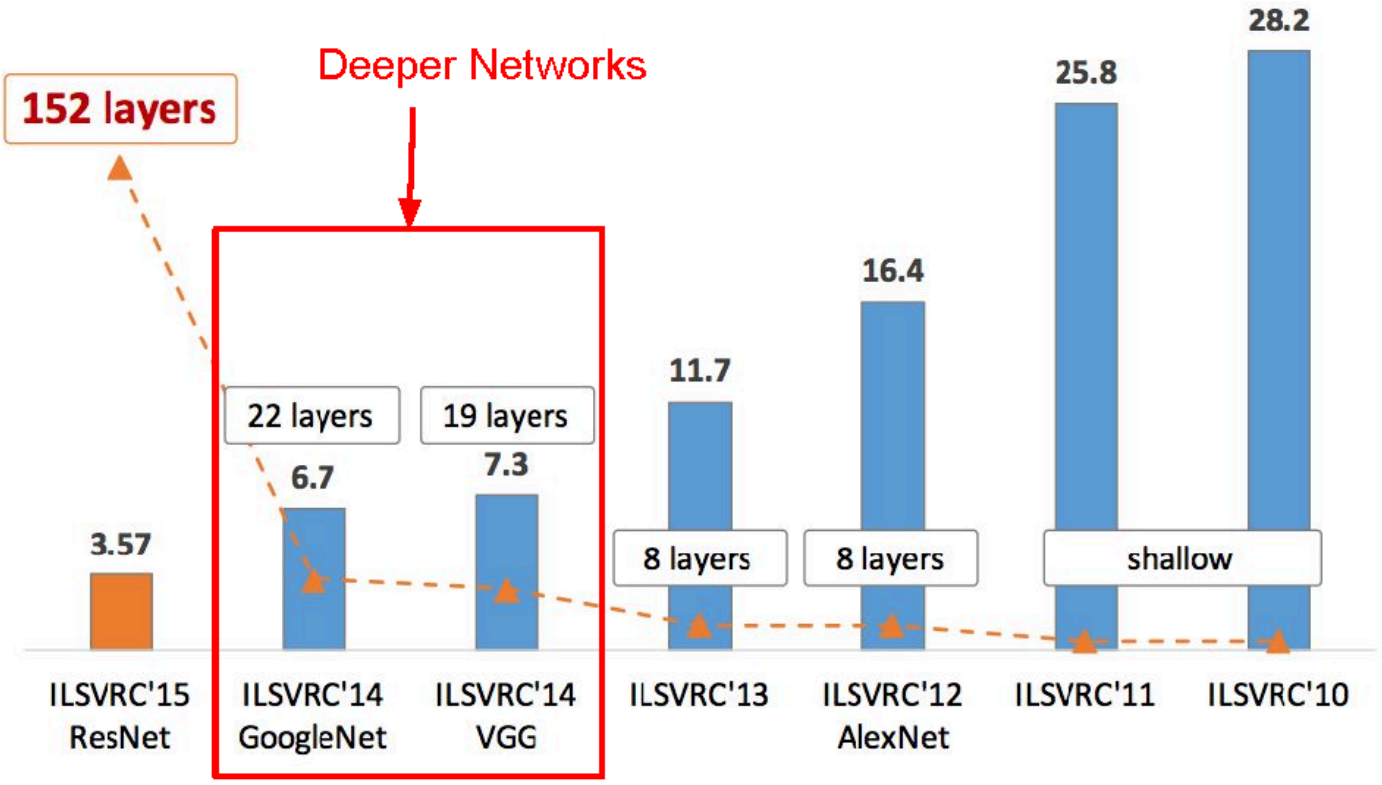
Keep it deep.

Keep it simple.

This slide is taken from Manolis Kellis

[Simonyan and Zisserman, 2014]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



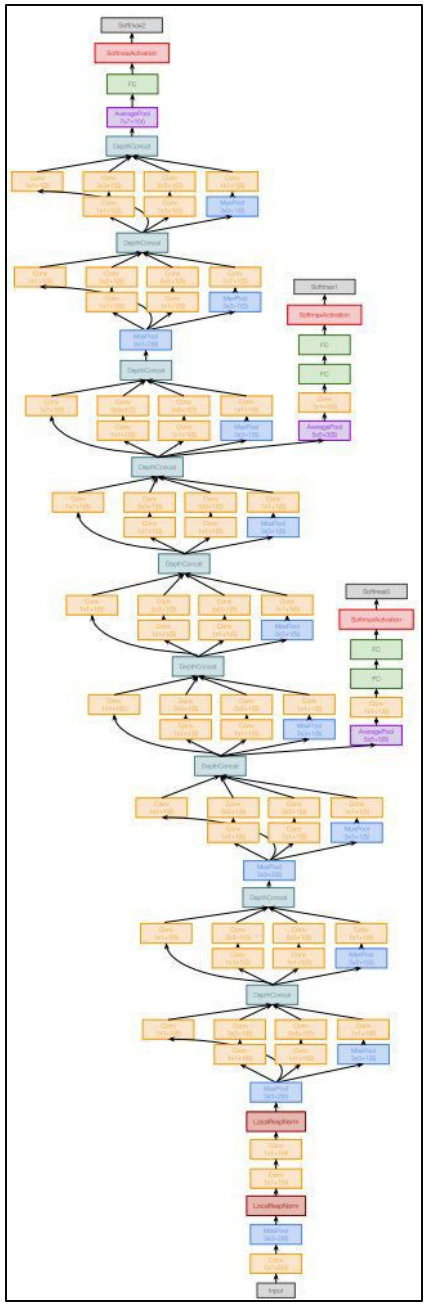
Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

GoogleNet

- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

This slide is taken from Manolis Kellis

[Szegedy et al., 2014]



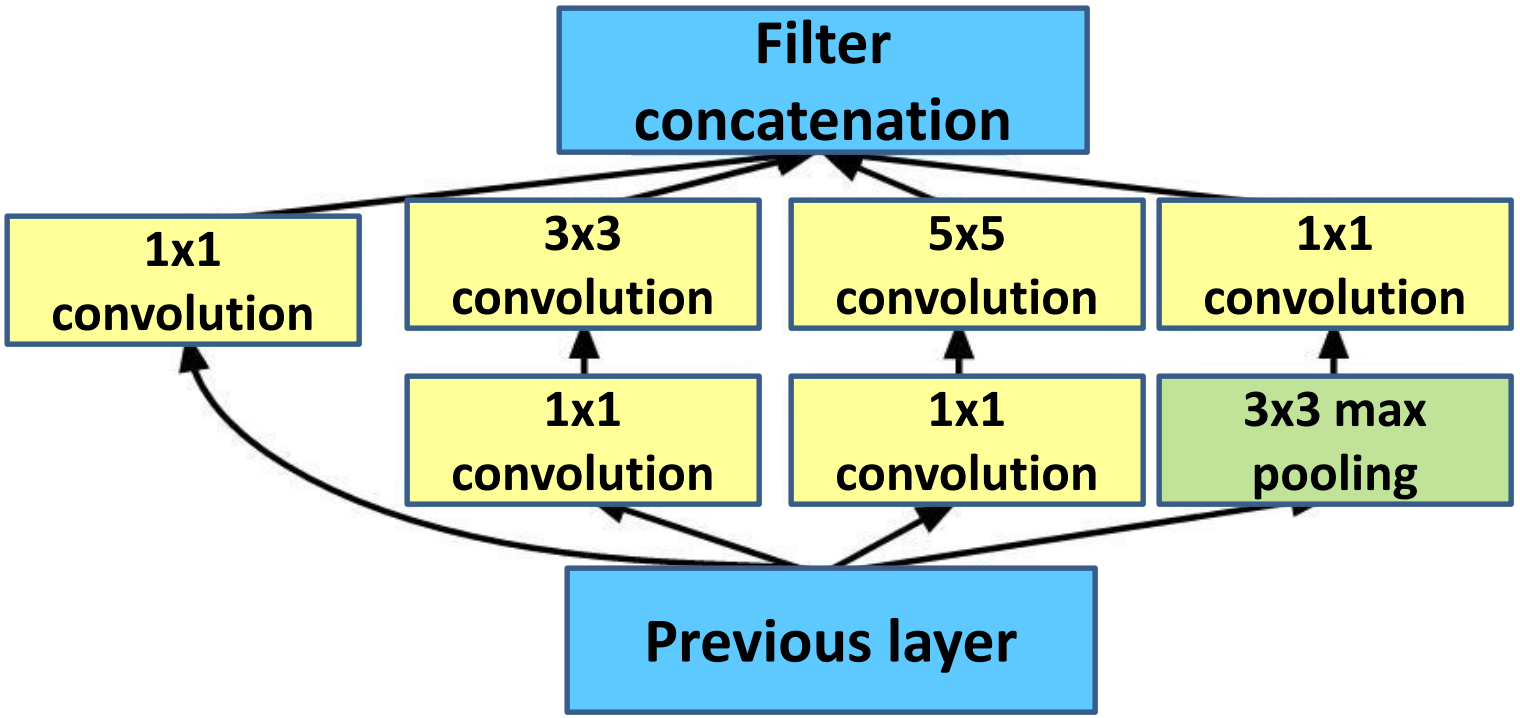
GoogleNet

- 22 layers
- Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)

[Szegedy et al., 2014]

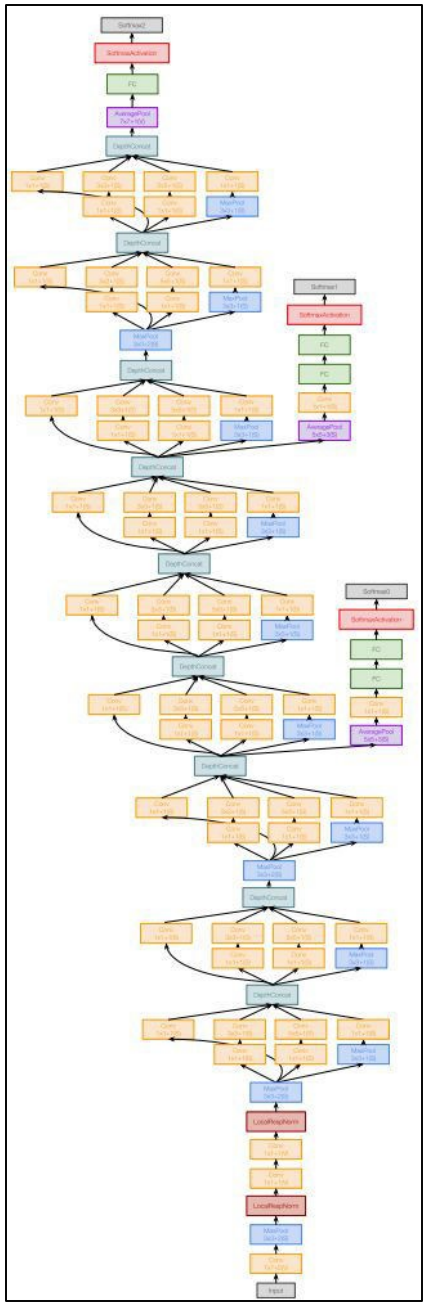
GoogleNet

"**Inception module":** design a good local network topology (network within a network) and then stack these modules on top of each other



Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Szegedy et al., 2014]



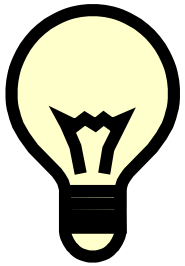
GoogleNet

Details/Retrospectives :

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[Szegedy et al., 2014]



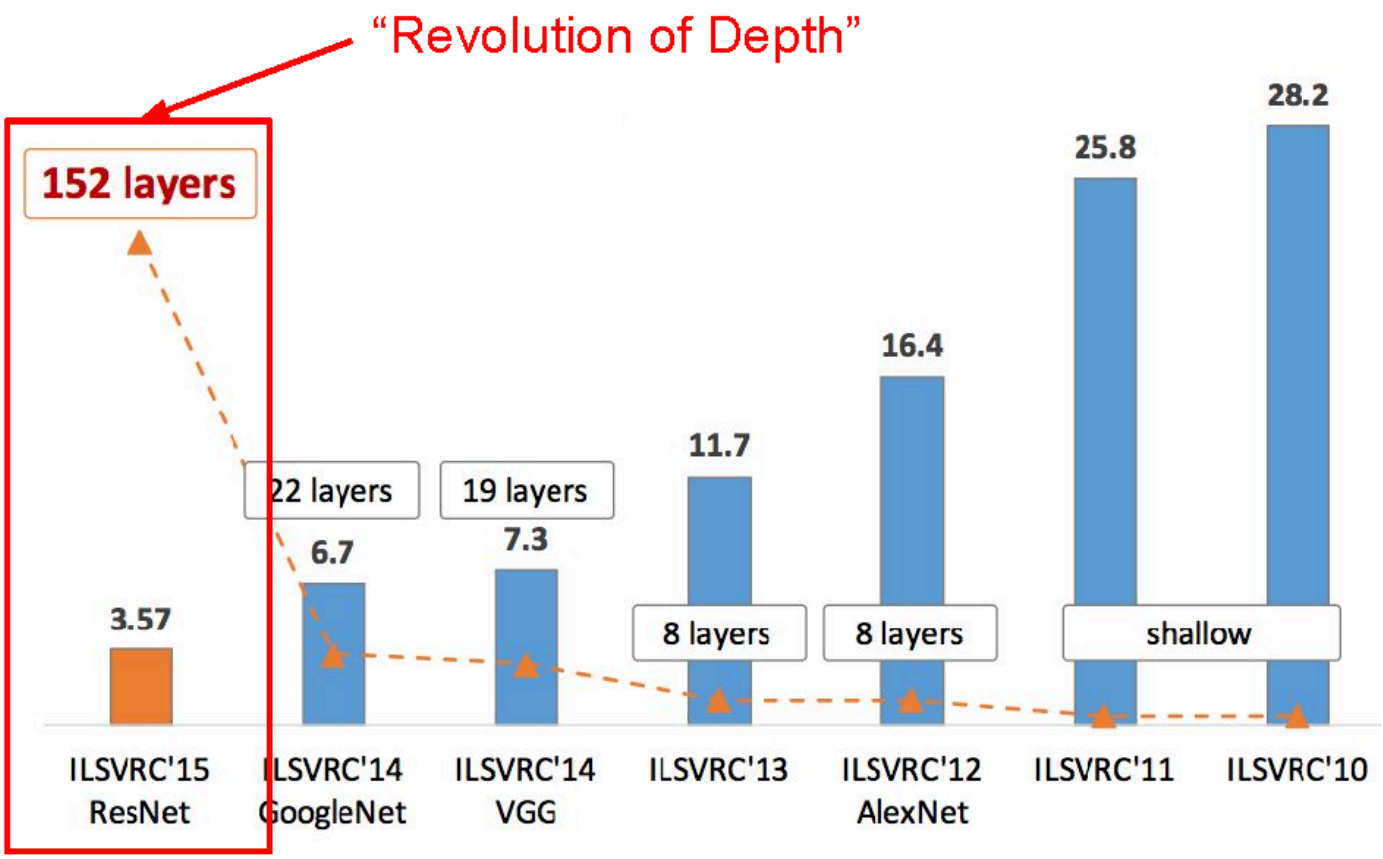
GoogleNet

Introduced the idea that CNN layers **didn't always have to be stacked up sequentially**. Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and **computationally efficiency**.

This slide is taken from Manolis Kellis

[Szegedy et al., 2014]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



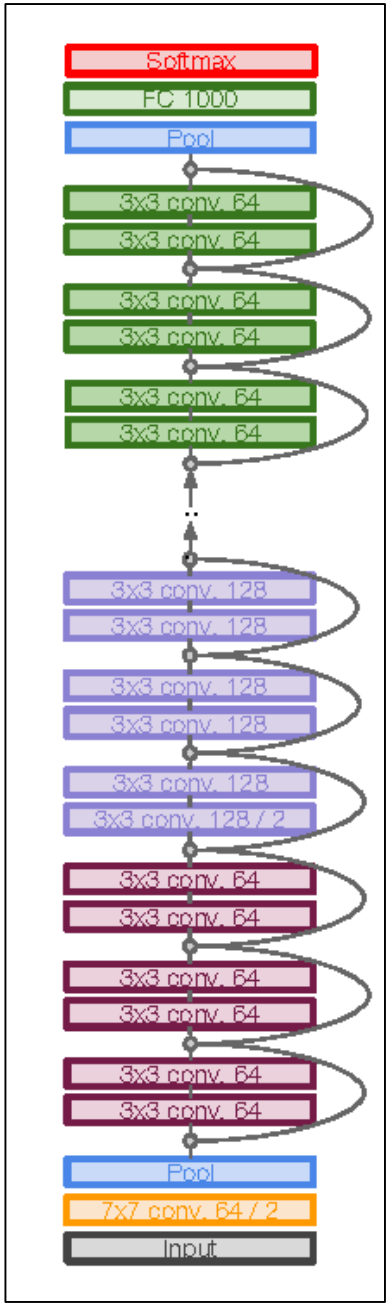
Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- Extremely deep network – 152 layers
- Deeper neural networks are more difficult to train.
- Deep networks suffer from vanishing and exploding gradients.
- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

This slide is taken from Manolis Kellis

[He et al., 2015]



ResNet

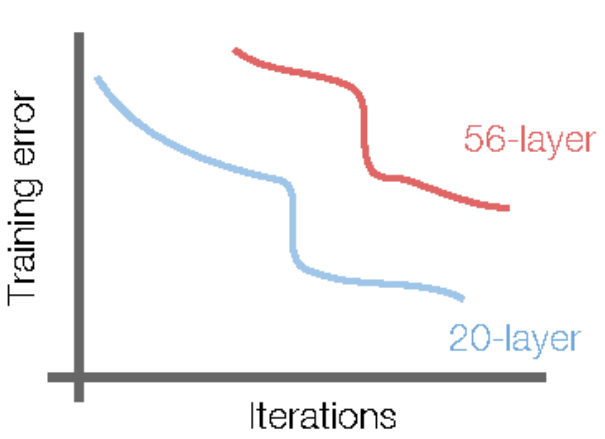
- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
  Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

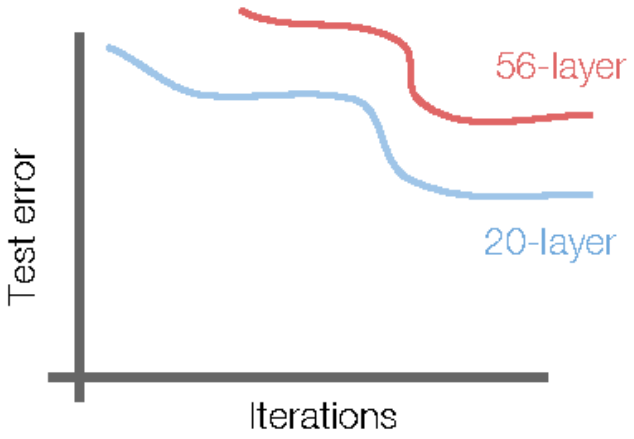
Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[He et al., 2015]

ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?





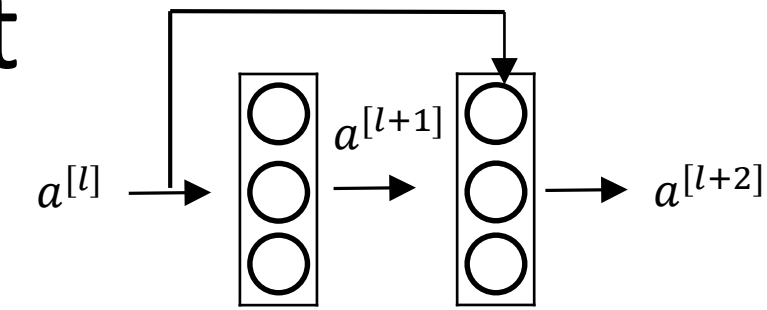
- 56-layer model performs worse on both training and test error

-> The deeper model performs worse (not caused by overfitting)!

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[He et al., 2015]

ResNet



Short cut/ skip connection

$a^{[l]}$ → **Linear** → **ReLU** → **Linear** → **ReLU** → $a^{[l+2]}$

$a^{[l+1]}$

$$\mathbf{z}^{[l+1]} = \mathbf{W}^{[l+1]}\,\mathbf{a}^{[l]} + \mathbf{b}^{[l+1]}$$

$$\mathbf{a}^{[l+1]} = \mathbf{g}(\mathbf{z}^{[l+1]})$$

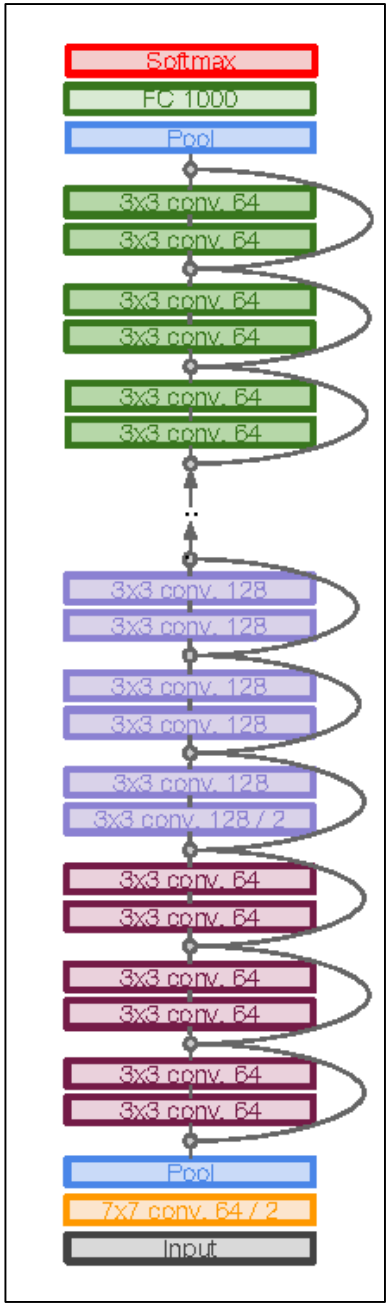
$$\mathbf{z}^{[l+2]} = \mathbf{W}^{[l+2]}\mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]}$$

$$\mathbf{a}^{[l+2]} = \mathbf{g}(\mathbf{z}^{[l+2]})$$

$$\mathbf{a}^{[l+2]} = \mathbf{g}\left(\mathbf{z}^{[l+2]} + \mathbf{a}^{[l]}\right) = \mathbf{g}(\mathbf{W}^{[l+2]}\mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]} + \mathbf{a}^{[l]})$$

This slide is taken from Manolis Kellis

[He et al., 2015]



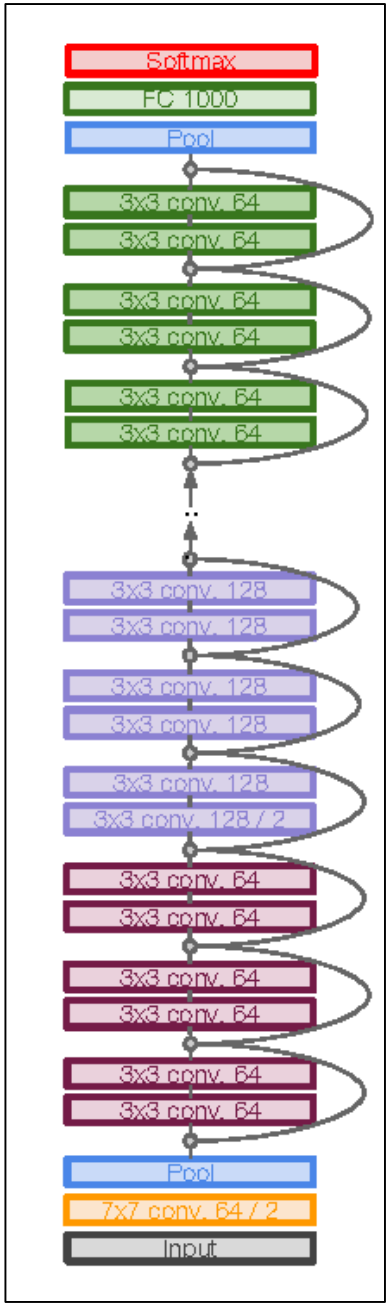
ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[He et al., 2015]



ResNet

- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[He et al., 2015]

ResNet

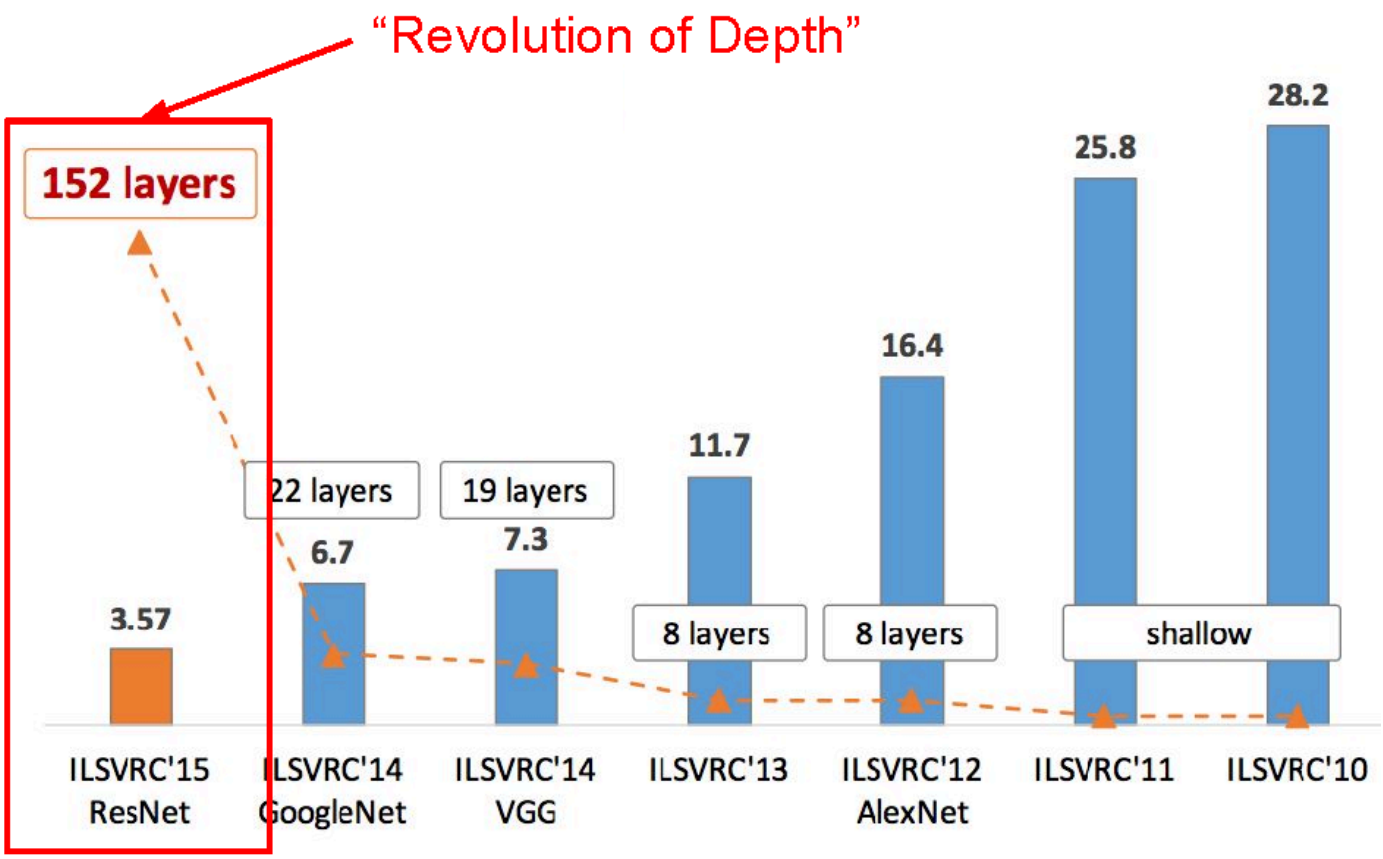
Experimental Results:

- Able to train very deep networks without degrading
- Deeper networks now achieve lower training errors as expected

Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

[He et al., 2015]

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

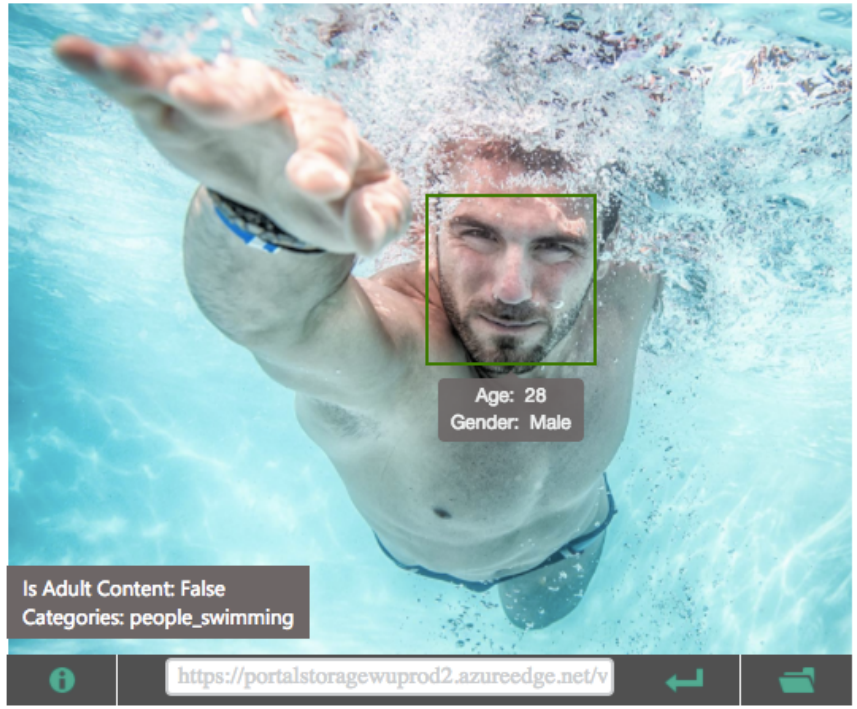


Slide taken from Fei-Fei & Justin Johnson & Serena Yeung. Lecture 9.

RNN

Sequence Applications: One-to-Many

- **Input**: fixed-size
- **Output**: sequence
- e.g., image captioning



Features:

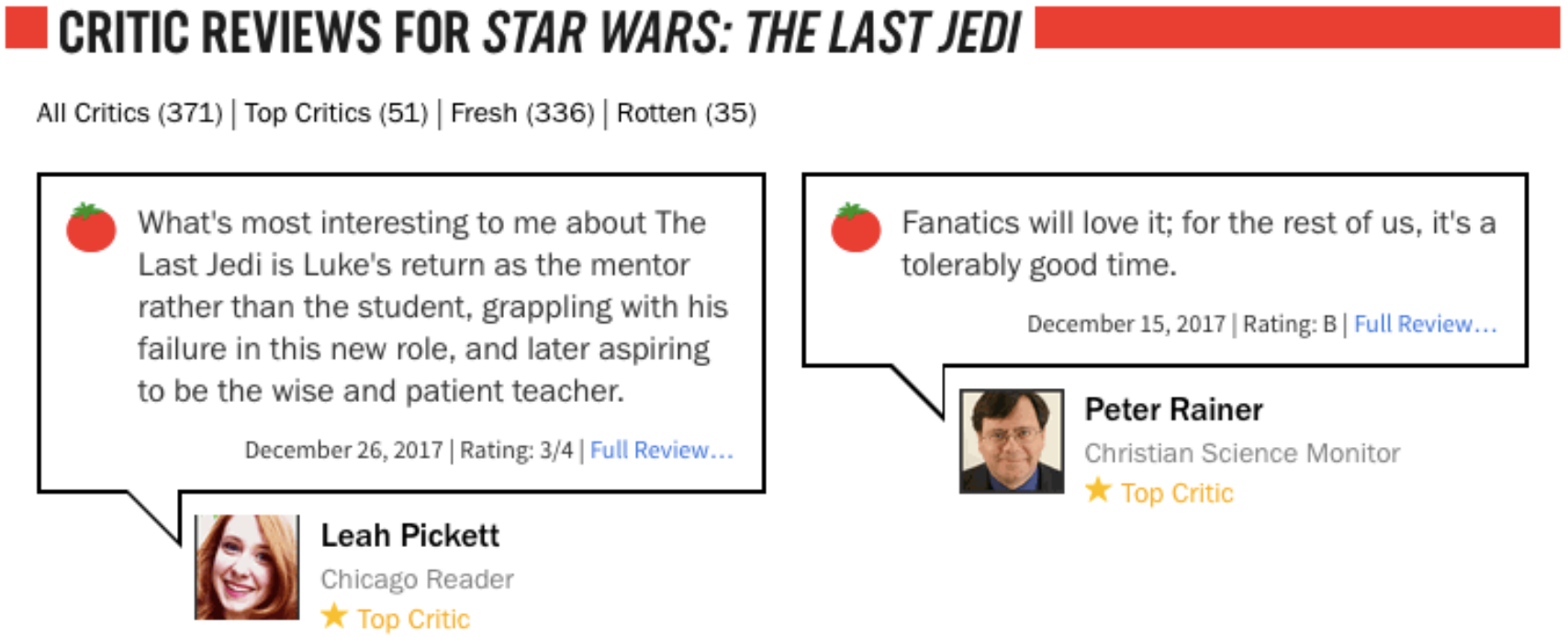
Feature Name	Value
Description	{ "type": 0, "captions": [{ "text": "a man swimming in a pool of water", "confidence": 0.7850108693093019 }] }
Tags	[{ "name": "water", "confidence": 0.9996442794799805 }, { "name": "sport", "confidence": 0.9504992365837097 }, { "name": "swimming", "confidence": 0.9062818288803101, "hint": "sport" }, { "name": "pool", "confidence": 0.8787588477134705 }, { "name": "water sport", "confidence": 0.631849467754364, "hint": "sport" }]
Image Format	jpeg
Image Dimensions	1500 x 1155
Clip Art Type	0 Non-clipart
Line Drawing Type	0 Non-LineDrawing
Black & White Image	False

Captions: https://www.microsoft.com/cognitive-services/en-us/computer-vision-api

This slide is taken from Danna Gurari

Sequence Applications: Many-to-One

- **Input**: sequence
- **Output**: fixed-size
- e.g., sentiment analysis (hate? love?, etc)



https://www.rottentomatoes.com/m/star_wars_the_last_jedi

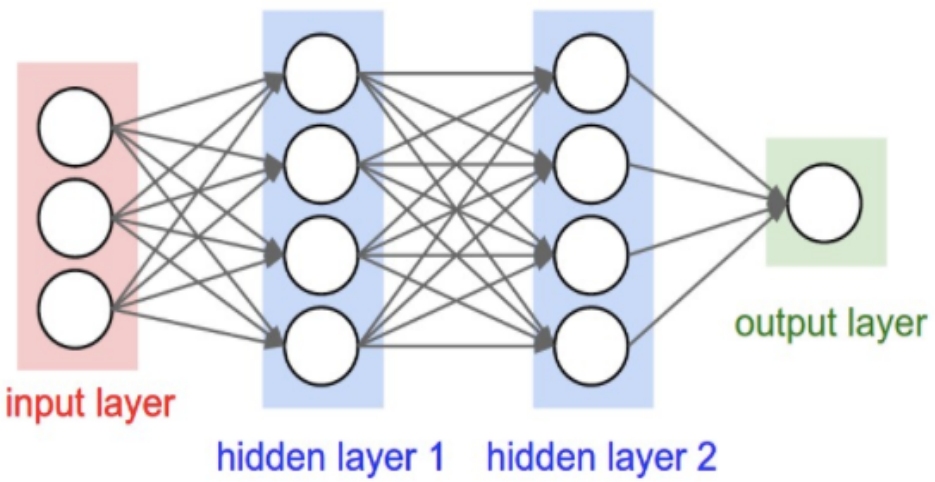
This slide is taken from Danna Gurari

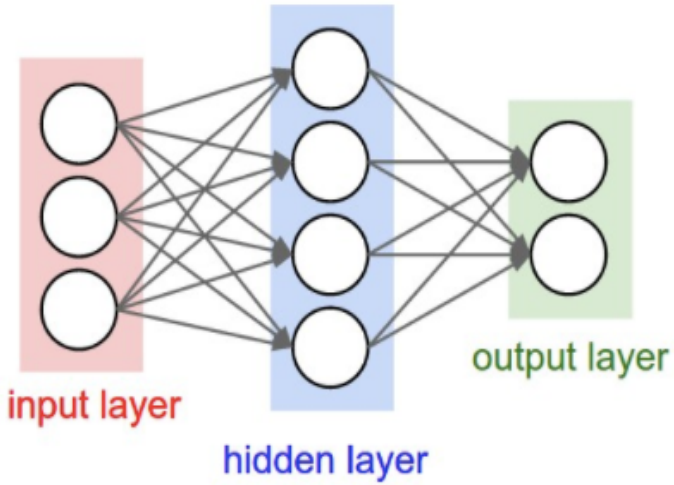
Sequence Applications: Many-to-Many

- **Input**: sequence
- **Output**: sequence
- e.g., language translation

This slide is taken from Danna Gurari

Recall: Feedforward Neural Networks





Problem: many model parameters!

Problem: no memory of past since weights learned independently

Each layer serves as input to the next layer with no loops

This slide is taken from Danna Gurari

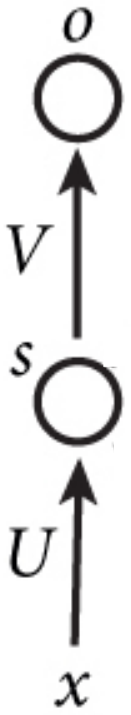
Figure Source: http://cs231n.github.io/neural-networks-1/

Recurrent Neural Networks (RNNs)

- Main idea: use hidden state to capture information about the past

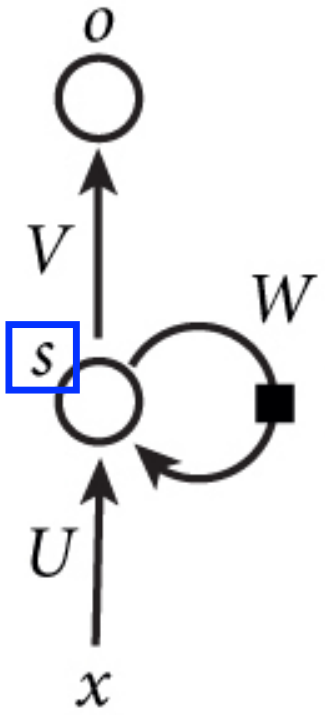
Feedforward Network

Each layer receives input from the previous layer with no loops



Recurrent Network

Each layer receives input from the previous layer and the output from the previous time step

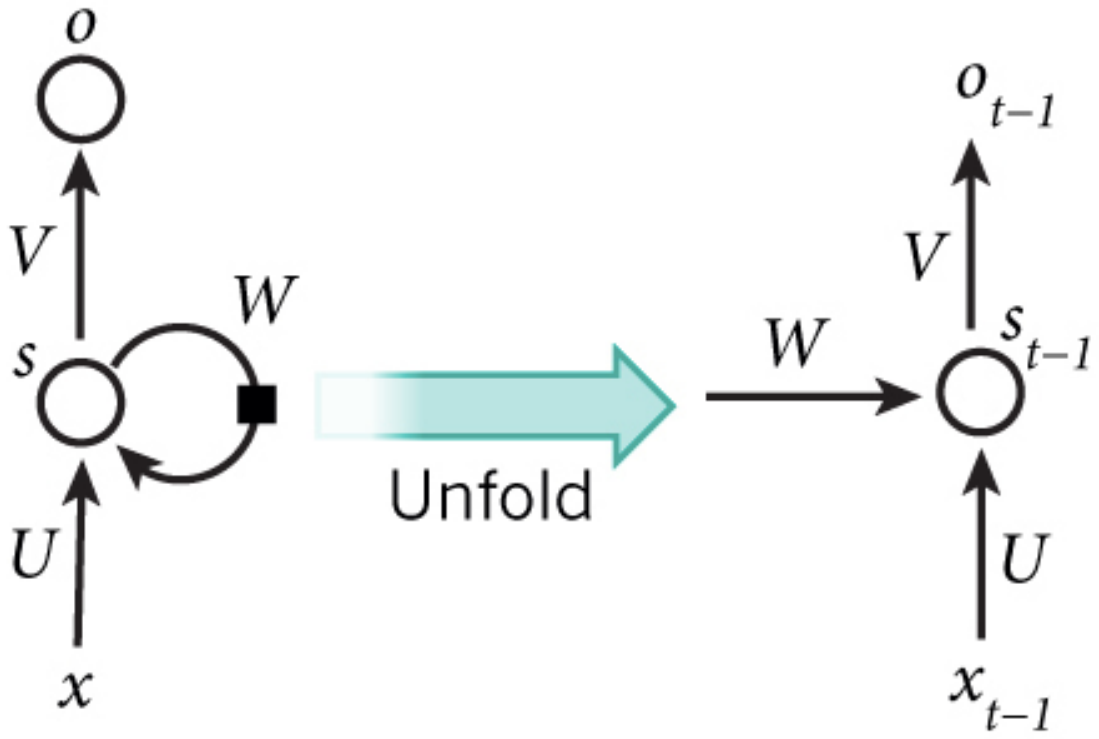


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: Time Step 1

- Main idea: use hidden state to capture information about the past

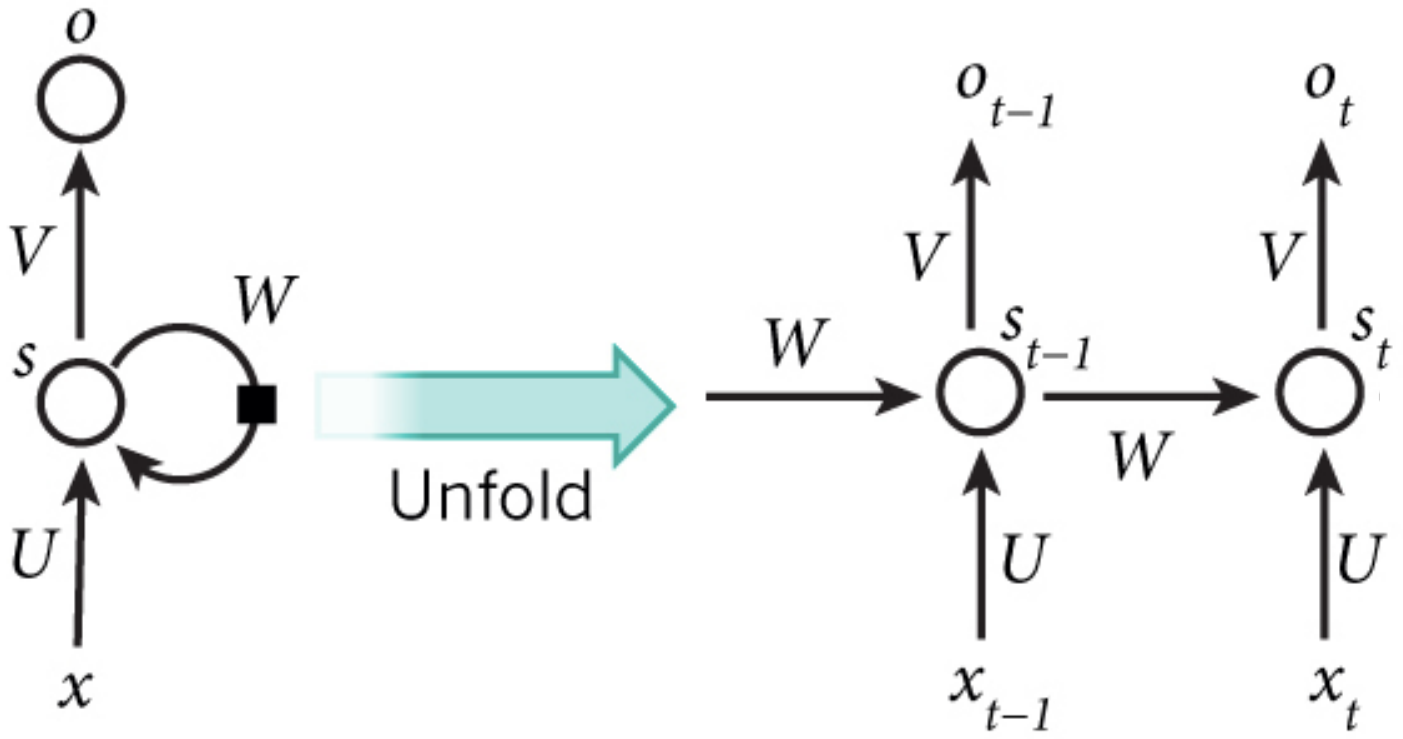


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: Time Step 2

- Main idea: use hidden state to capture information about the past

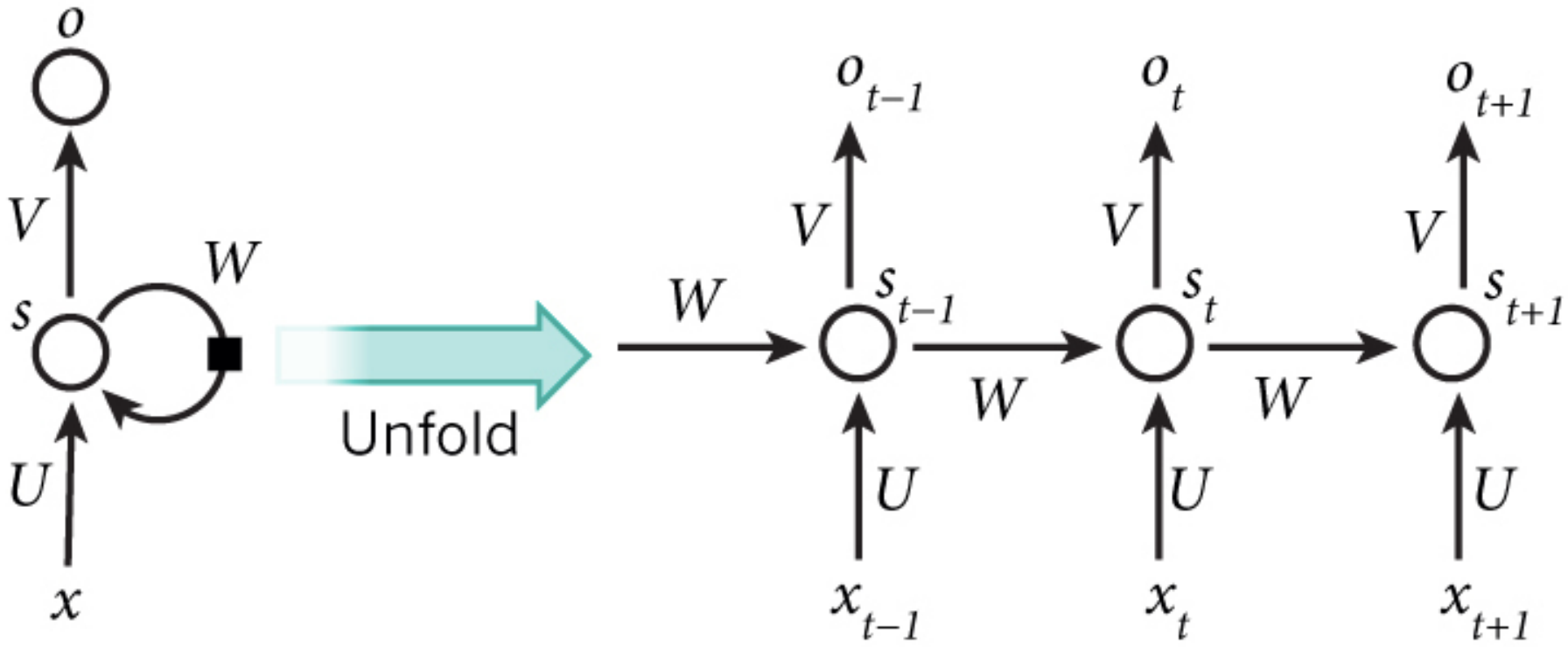


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: Time Step 3

- Main idea: use hidden state to capture information about the past

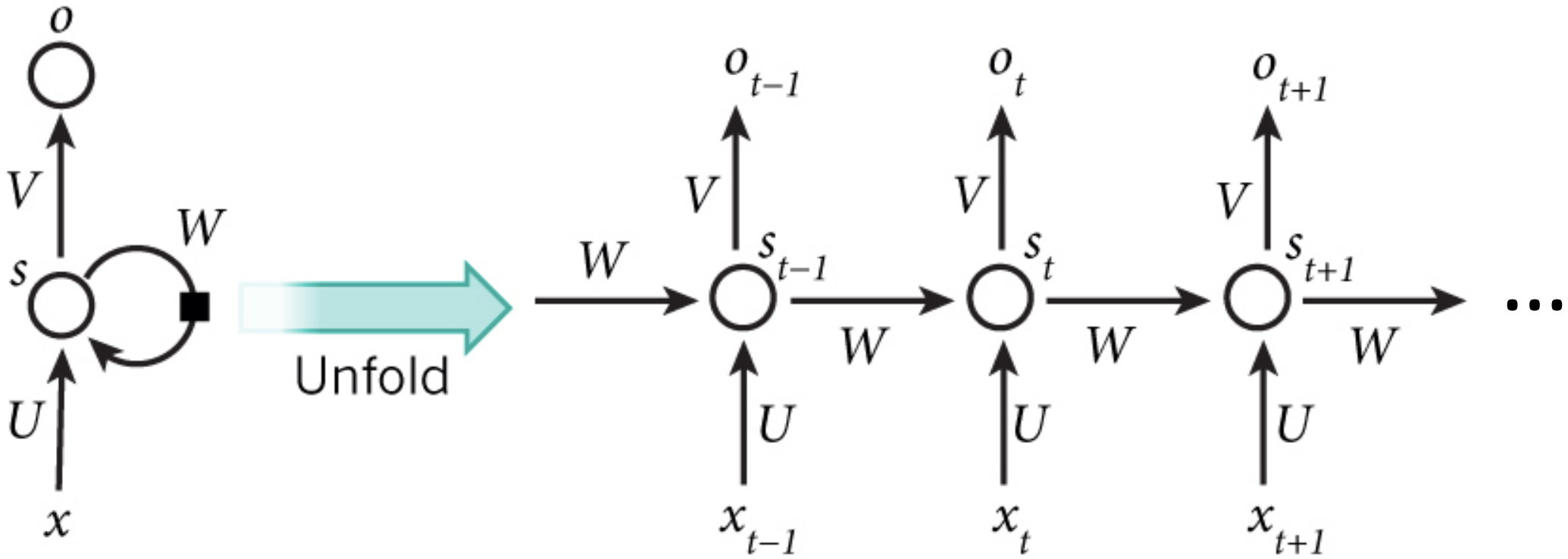


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: And So On…

- Main idea: use hidden state to capture information about the past

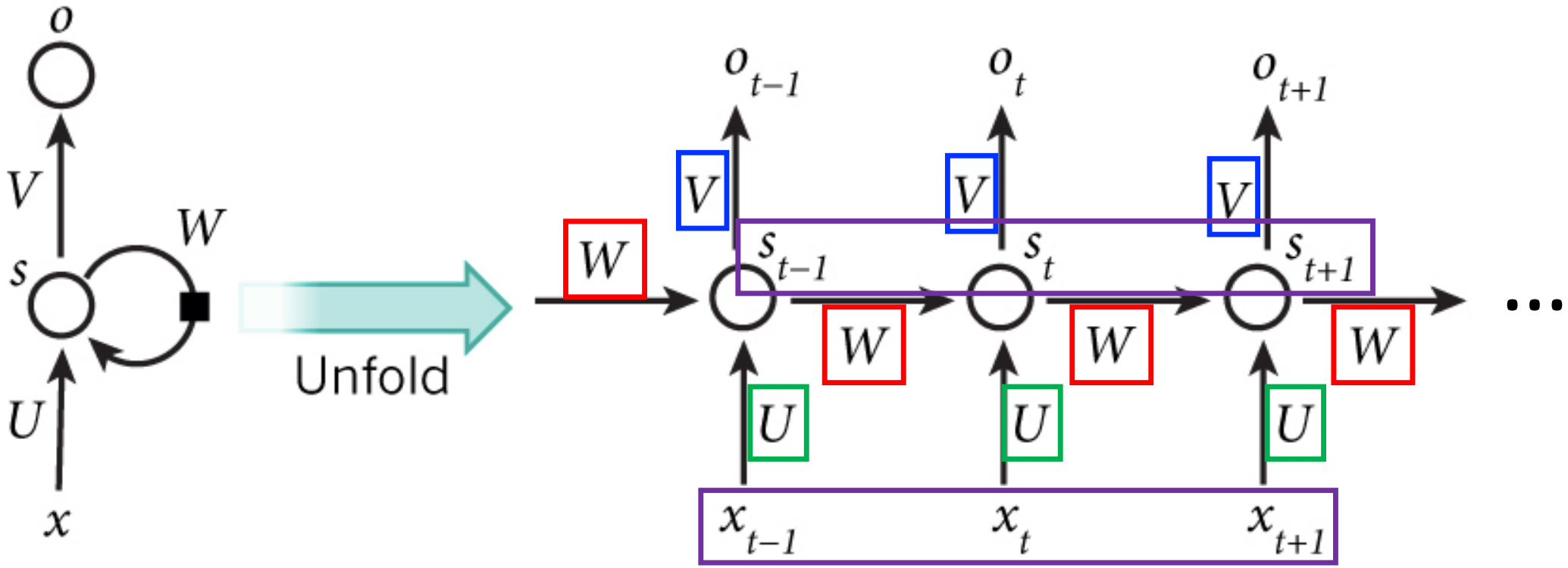


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: Model Parameters and Inputs

- All layers share the same model parameters (U, V, W)
  - What is different between the layers?

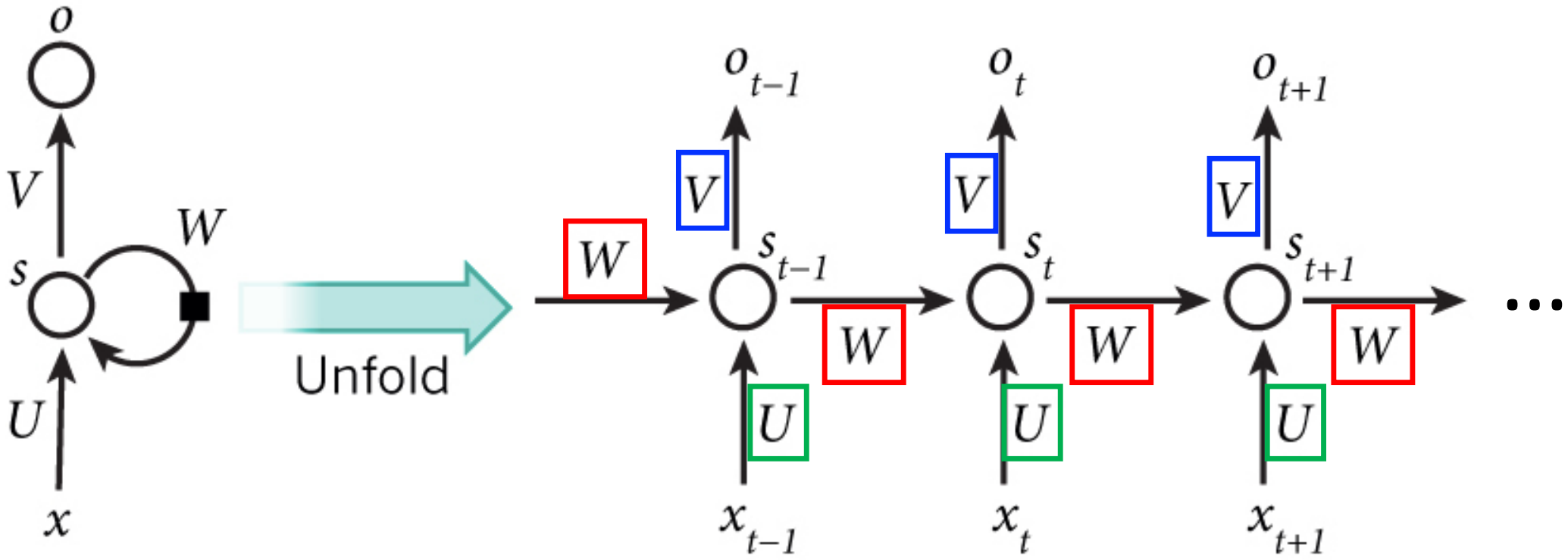


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: Model Parameters and Inputs

- When unfolded, a RNN is a deep feedforward network with shared weights!

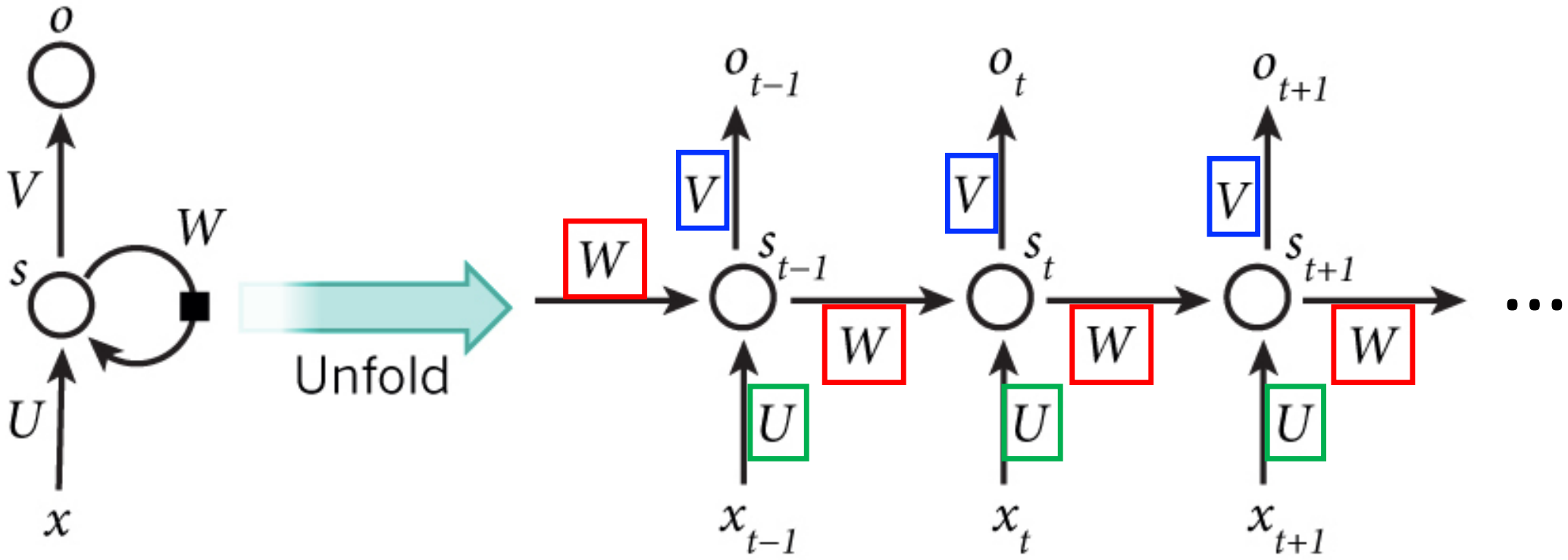


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: Advantages

- Overcomes problem that weights of each layer are learned **independently** by using previous hidden state
- Overcomes problem that model has many parameters since weights are shared across layers

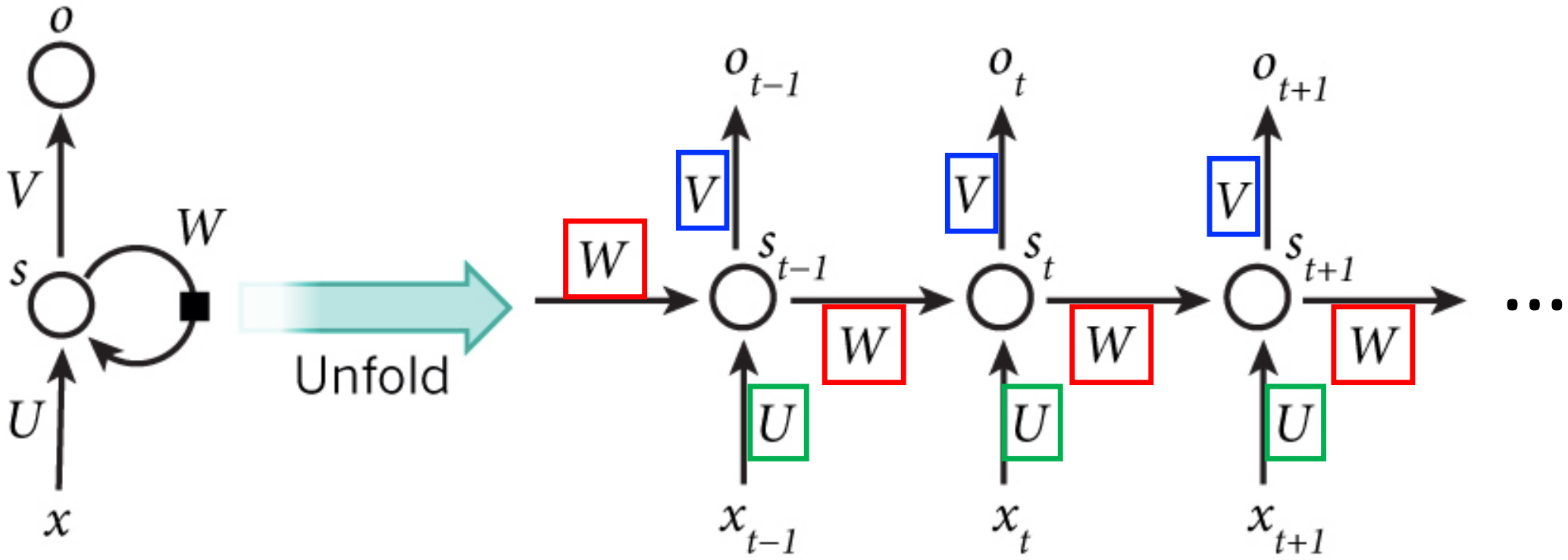


This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

RNN: Advantages

- Retains information about past inputs for an amount of time that depends on the model's weights and input data rather than a fixed duration selected a priori



This slide is taken from Danna Gurari

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

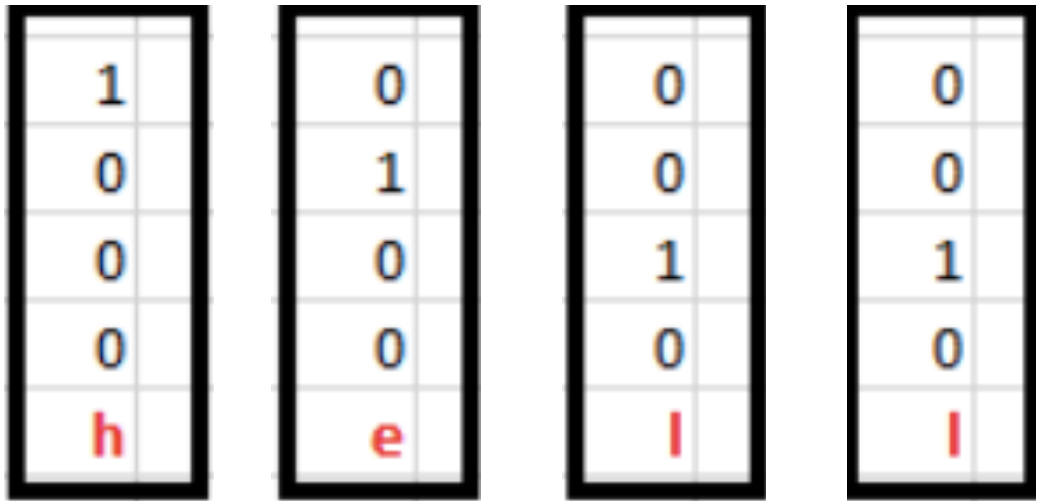
RNN Example: Predict Sequence of Characters

- Goal: predict next character in text
- Training Data: sequence of characters represented as one-hot vectors

This slide is taken from Danna Gurari

Example: Predict Sequence of Characters

- Goal: predict next character in text
- Prediction: feed training sequence of one-hot encoded characters; e.g., “hello”
  - For simplicity, assume the following vocabulary (i.e., character set): {h, e, l, o}
  - What is our input at time step 1?
  - What is our input at time step 2?
  - What is our input at time step 3?
  - What is our input at time step 4?
  - And so on…



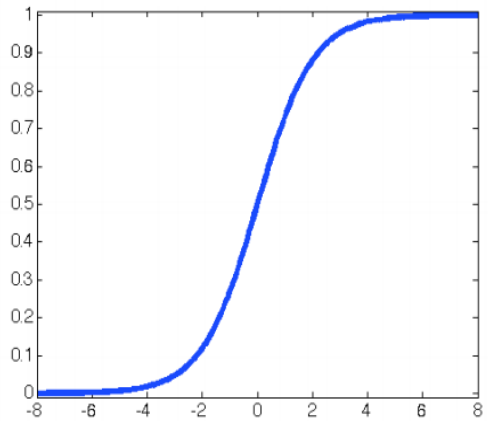
This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

Example: Predict Sequence of Characters

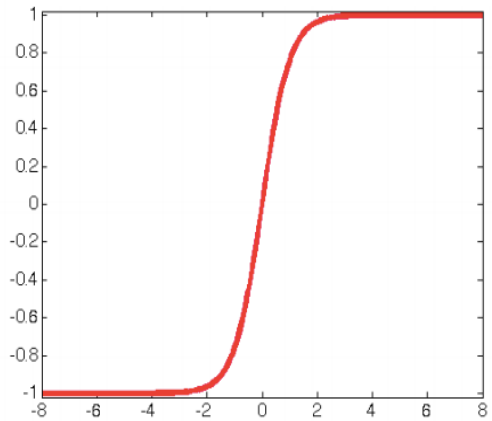
Recall activation functions: use tanh as activation function

Sigmoid



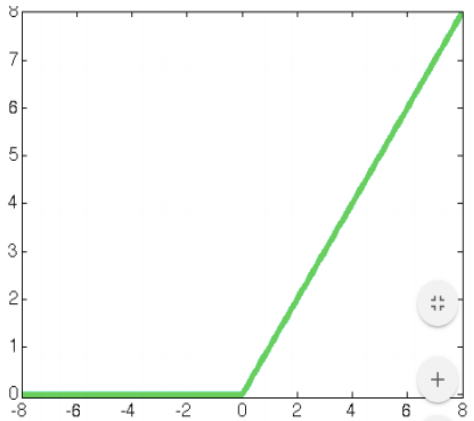
$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

Tanh



$$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$$

ReLU



$$\mathrm{ReLU}(z) = \max(0, z)$$

This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

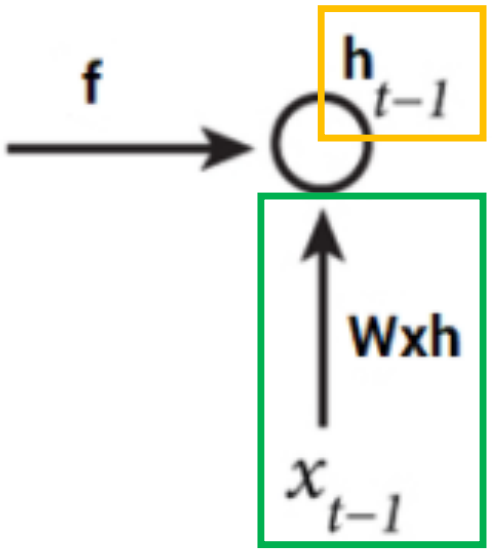
Example: Predict Sequence of Characters

Initialize to random value: 0.567001

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + \text{bias})$$

Initialize to 0

Input at next time step



wxh			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

× [1, 0, 0, 0, h] = [0.287027, 0.902874, 0.537524]

H_t = TANH { [0.287027359, 0.902874425, 0.537523791] + [0.567001, 0.567001, 0.567001] }

= [0.693168, 0.899554, 0.802118]

This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

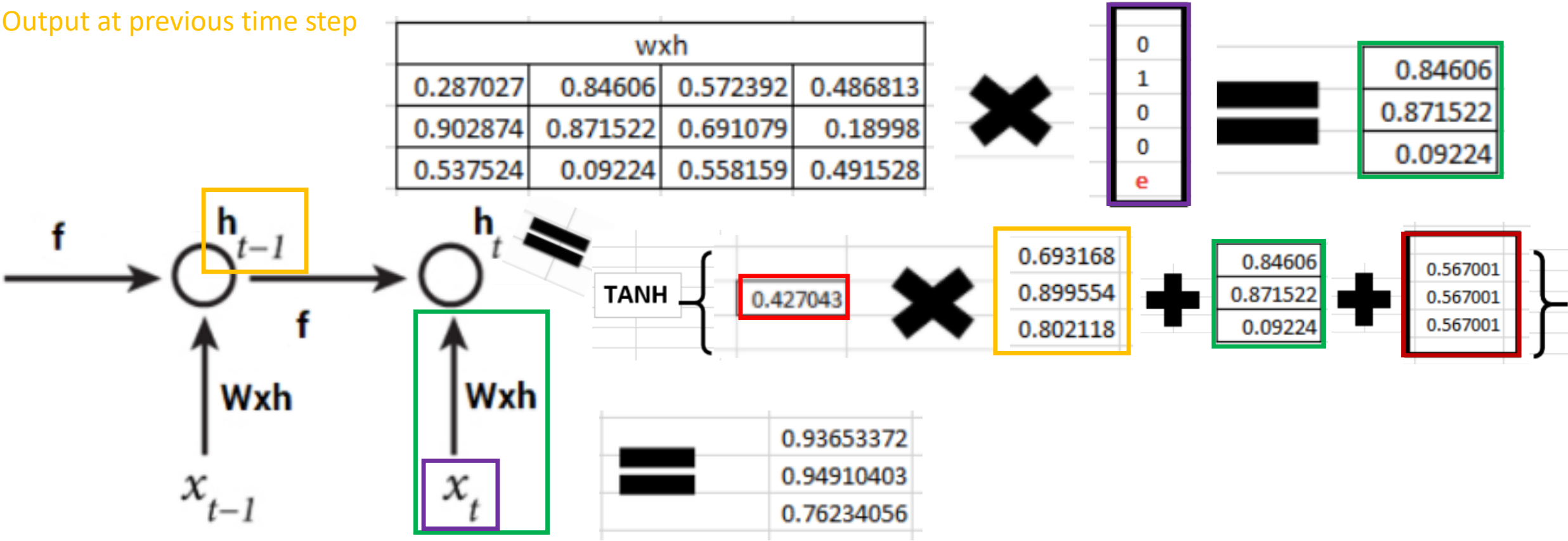
Example: Predict Sequence of Characters

Initialize to random value: 0.427043

Initialized to random value: 0.567001

$$h_t = \tanh\left(W_{hh} h_{t-1} + W_{xh} x_t + \text{bias}\right)$$

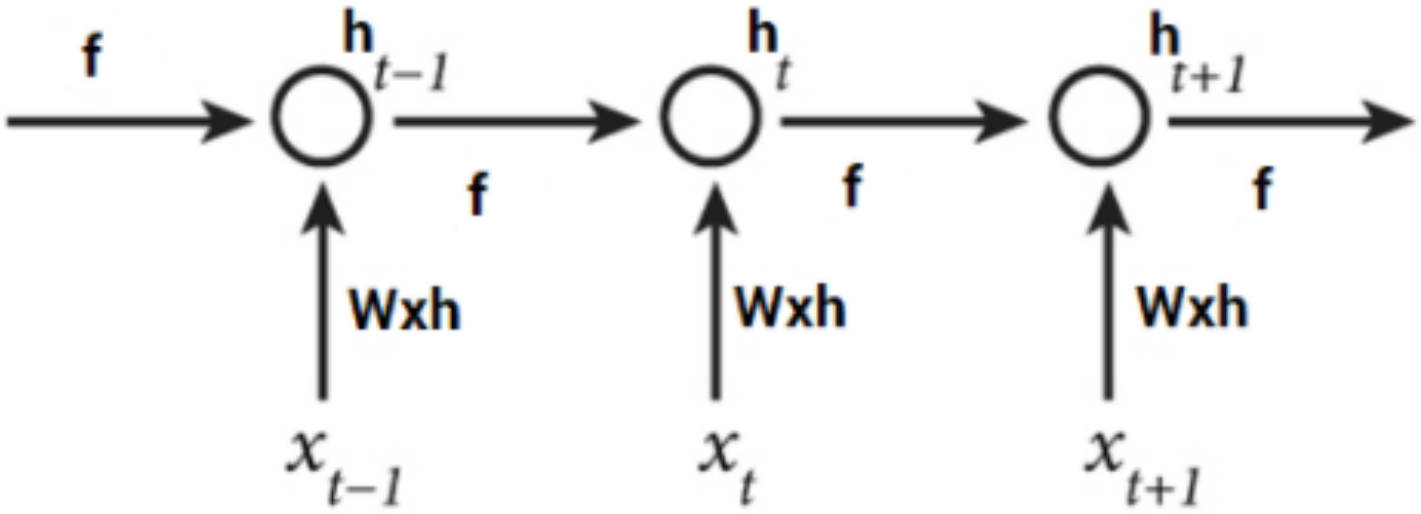
Output at previous time step



This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

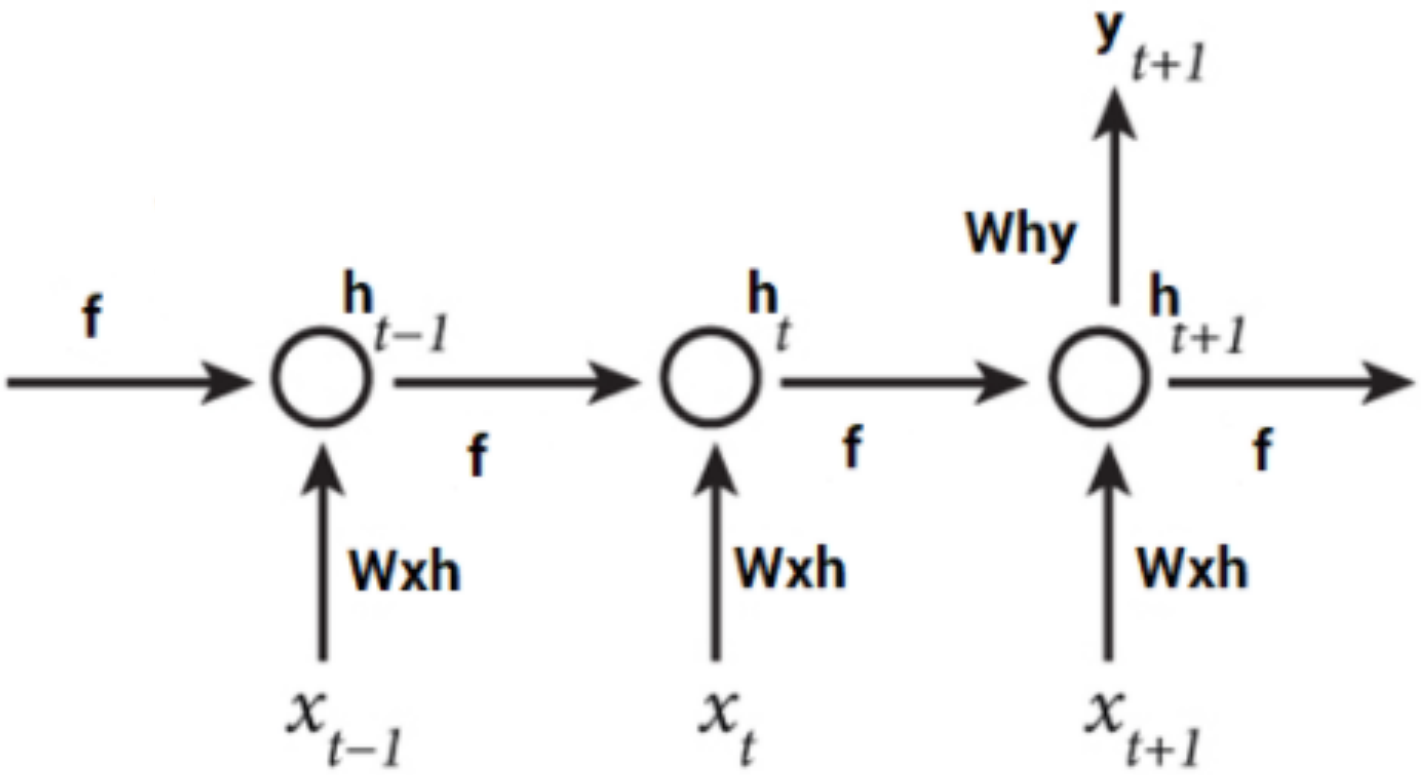
Example: Predict Sequence of Characters



This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

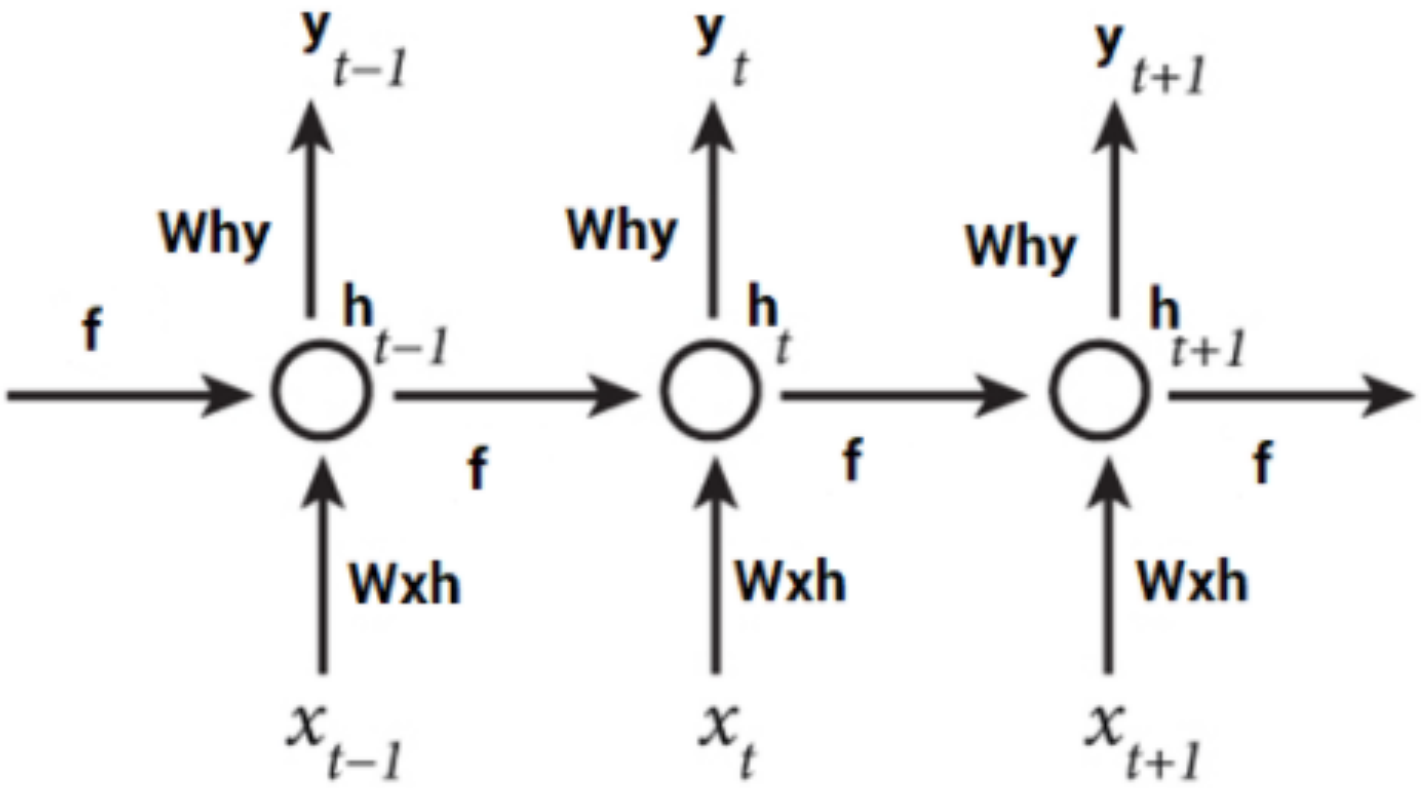
Example: Prediction (Many-To-One)



This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

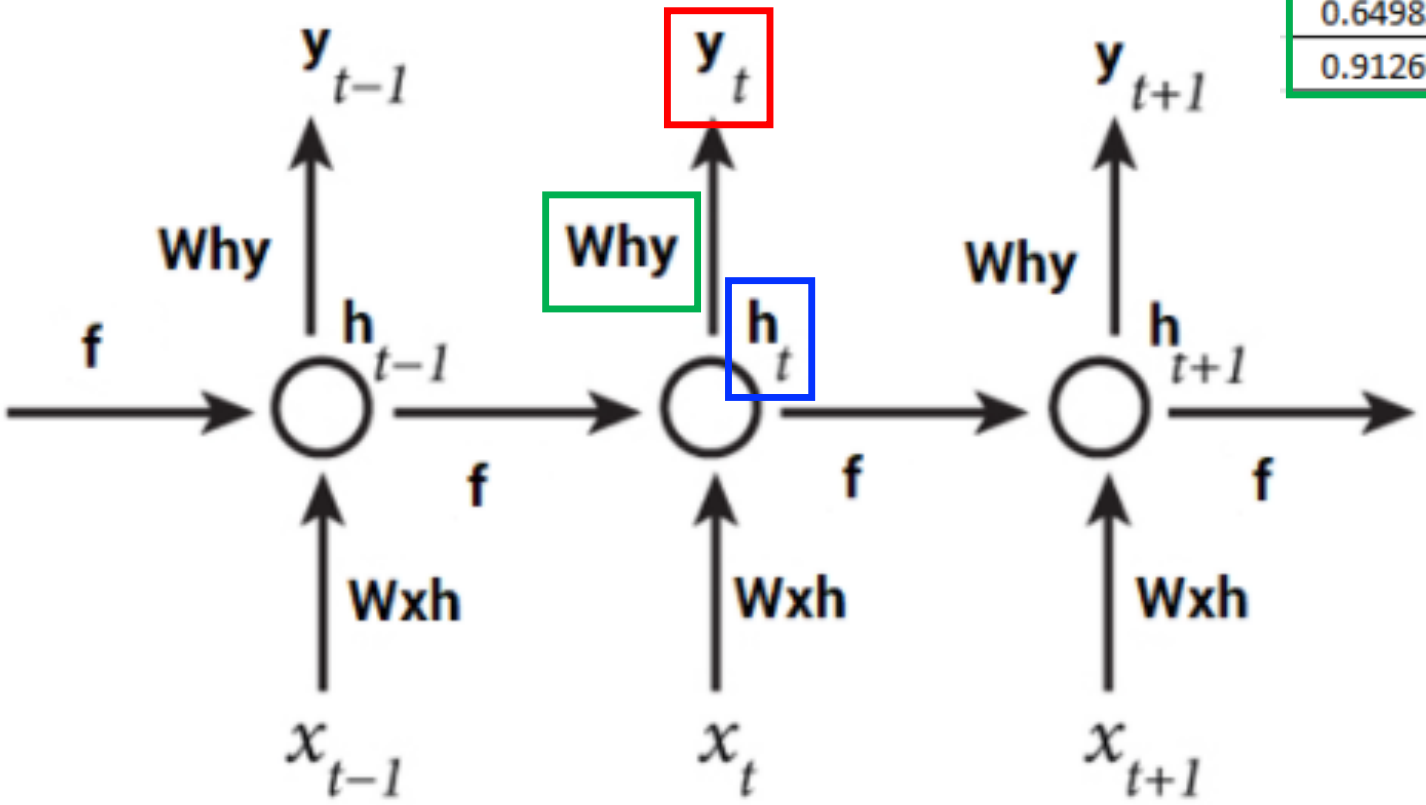
Example: Prediction (Many-To-Many)



This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

Example: Prediction for Time Step 2



why		
0.37168	0.974829459	0.830034886
0.39141	0.282585823	0.659835709
0.64985	0.09821557	0.334287084
0.91266	0.32581642	0.144630018

×

Ht
0.936534
0.949104
0.762341

=

yt
1.90607732
1.13779113
0.95666016
1.27422602

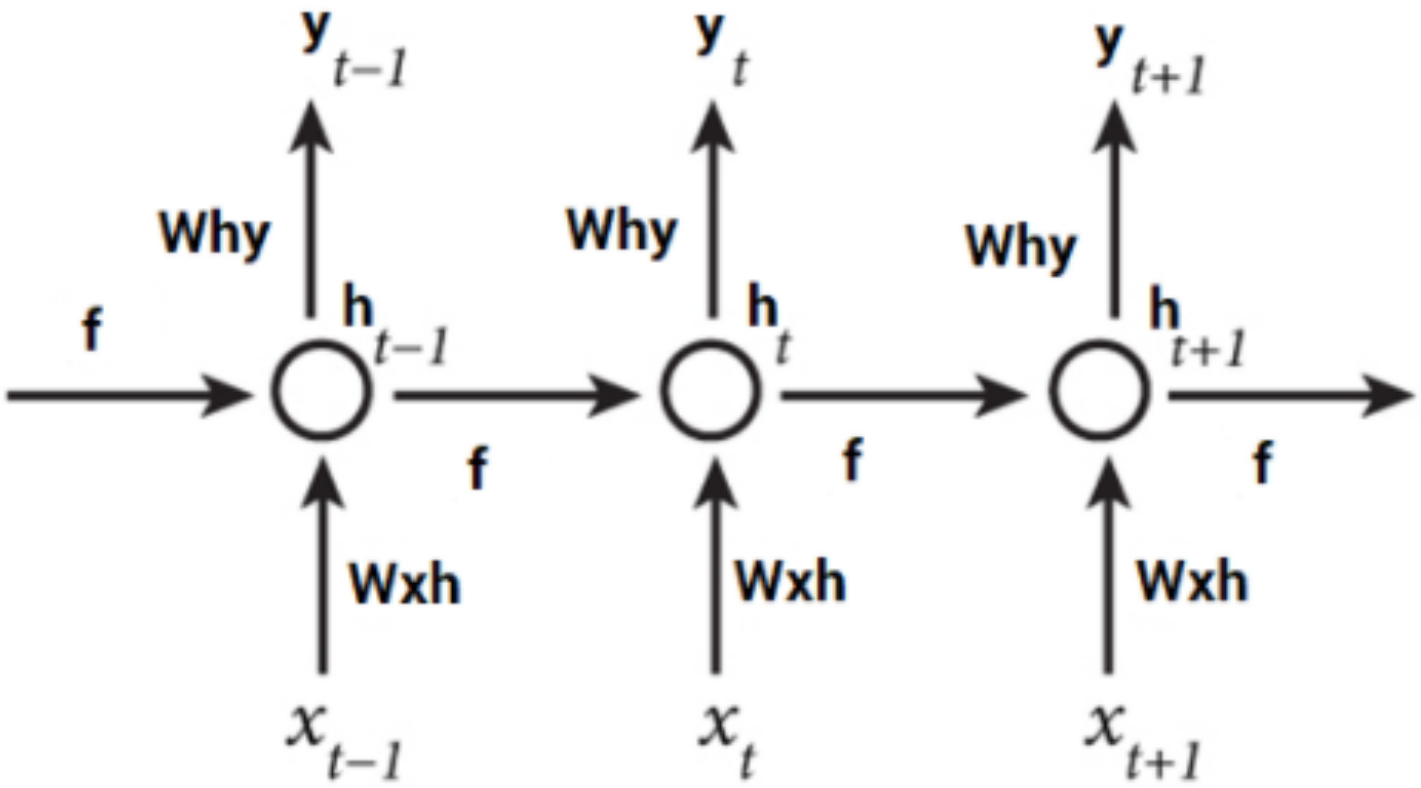
Applying softmax, to compute letter probabilities:

0.419748
0.194682
0.162429
0.223141

This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

Example: Prediction for Time Step 2



Given our vocabulary is {h, e, l, o}, what letter is predicted?

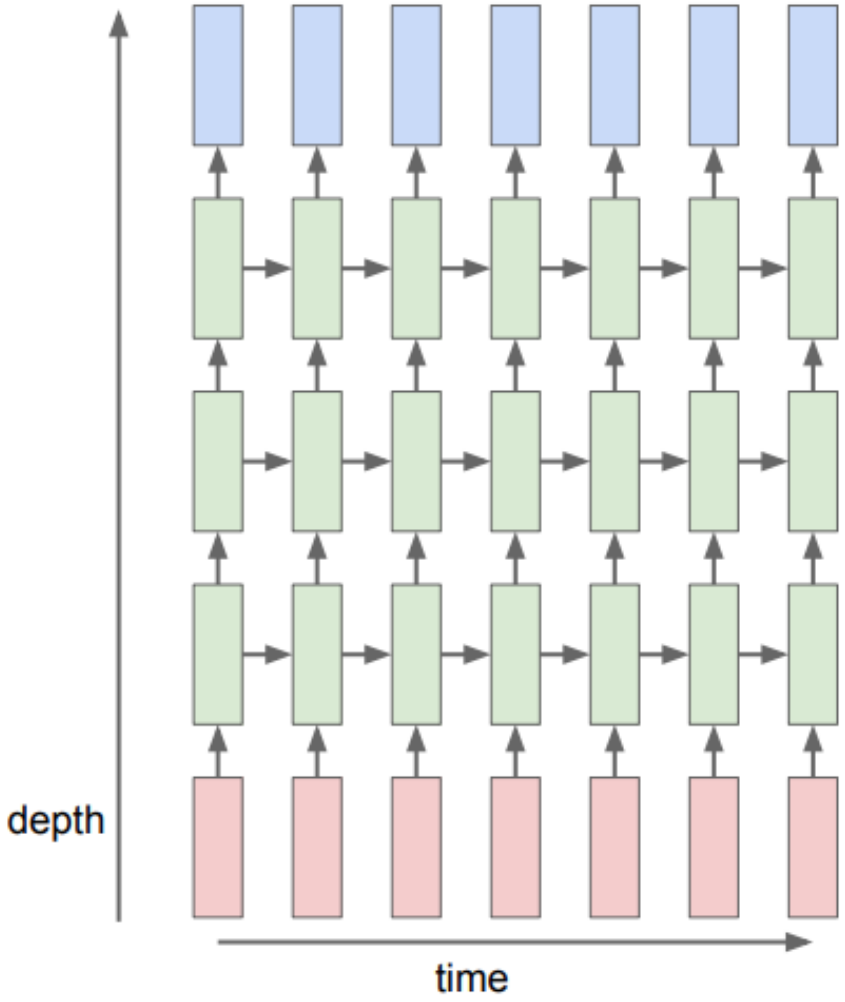
Applying softmax, to compute letter probabilities:

0.419748
0.194682
0.162429
0.223141

This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

RNN Variants: Different Number of Hidden Layers



Experimental evidence suggests deeper models can perform better:

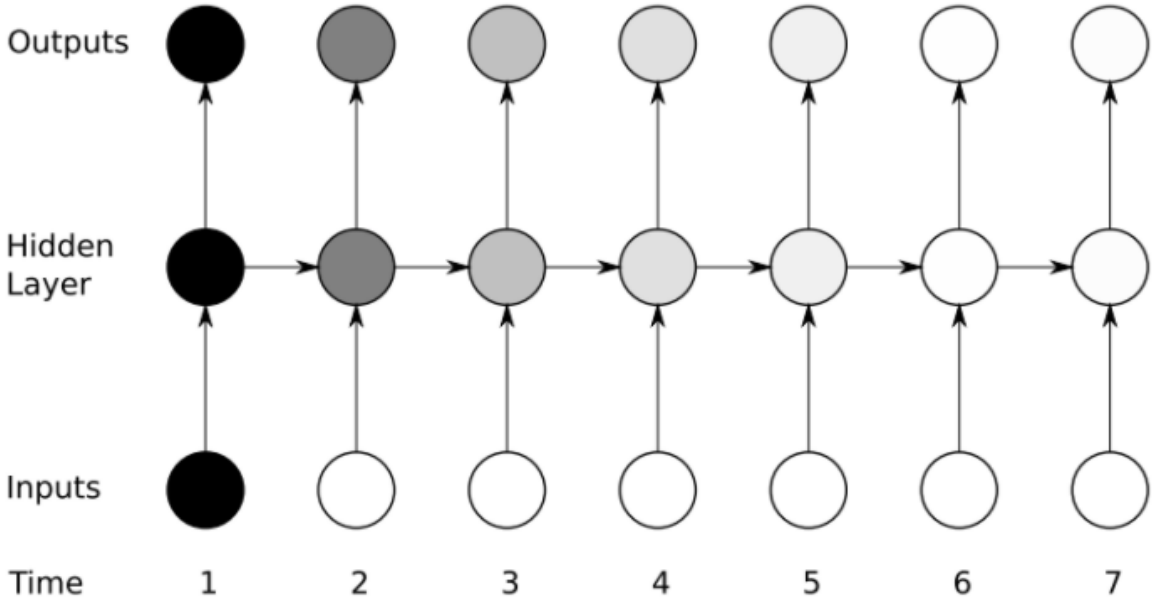
- Graves et al.; Speech Recognition with Deep Recurrent Neural Networks; 2013.
- Pascanu et al.; How to Construct Deep Recurrent Neural Networks; 2014.

This slide is taken from Danna Gurari

http://cs231n.stanford.edu/slides/2016/winter1516_lecture10.pdf

RNN: Vanishing Gradient Problem

- Problem: training to learn long-term dependencies
  - e.g., language: “In 2004, I started college” vs “I started college in 2004”



  - e.g., $\partial E/\partial W = \partial E/\partial y_3 * \partial y_3/\partial h_3 * \partial h_3/\partial y_2 * \partial y_2/\partial h_1$
- Vanishing gradient: a product of numbers less than 1 shrinks to zero
- Exploding gradient: a product of numbers greater than 1 explodes to infinity

This slide is taken from Danna Gurari

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/