THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

# CS 4364/6364

# Machine Learning

Fall Semester 9/26/2023
Lecture 10.
Feedforward Networks

John Sipple
jsipple@gwu.edu

# Announcements

Homework 2 Grades and Feedback

Homework 6 (vote for your favorite by **10/5/23!**)

- Unsupervised: Document Clustering
- Model Explainability with LIME, SHAP, and Integrated Gradients
- Autoencoders and Generative Adversarial Networks
- Optimal Control with Reinforcement Learning

Midterm Exam (**10/10/23**)

- Format - Design & Concepts
- Through Lecture 13 Optimization
- Practice Exams Posted
- Coordinate with me by **9/28/23** for special scheduling

# Final Project

Identify your team and submit a proposal by 10/17

- Teams of 3 - 5 (discuss exceptions with me)

Three options:

- Add to your existing research project (MS, PhD)
- Apply and perform a Kaggle, AIcrowd, or other competition
    - Deadline > 12/1/2023
- Uncanny ML Project

What you'll hand in:

- Technical paper (using the template provided): ⅓
- Working colab: ⅓
- Project presentation: ⅓

# Uncanny 1: CityLearn Challenge 2023


Energy Optimization Challenge:

Forecast Track

> design regression models to predict the 48-hour-ahead end-use load profiles for each building in a synthetic single-family neighborhood as well as the neighborhood-level 48-hour-ahead solar generation and carbon intensity profiles.

Control Track

> develop single-agent or multi-agent reinforcement learning control (RLC) policy and optional custom reward function or a model predictive control (MPC) policy for electrical (battery) and domestic hot water storage systems, and heat pump control in the buildings with the goal of maintaining thermal comfort, reducing carbon emissions, increasing energy efficiency and providing resiliency in the event of power outages

https://www.aicrowd.com/challenges/neurips-2023-citylearn-challenge

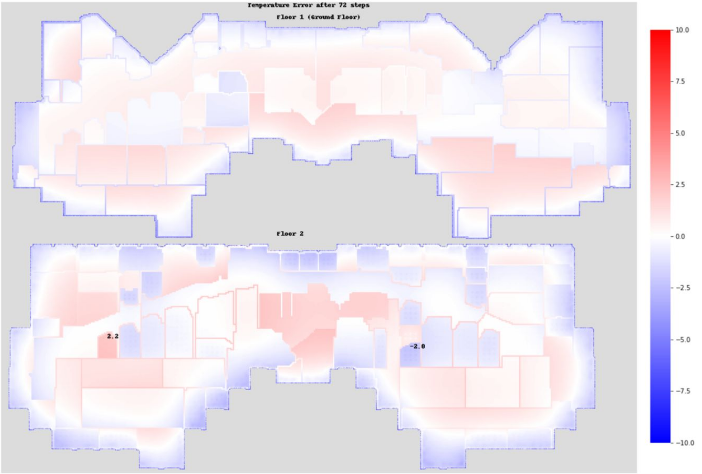
# Uncanny 2: Smart Buildings Optimization

Interactive Simulation of a real Google building in Mountain View, CA

To be published at BuildSys/RLEM in Nov

Research Reinforcement Learning for reducing energy consumption and carbon emission

Prototype at least one algorithm and compare against another

# Uncanny 3: Intelligent Diagnostics

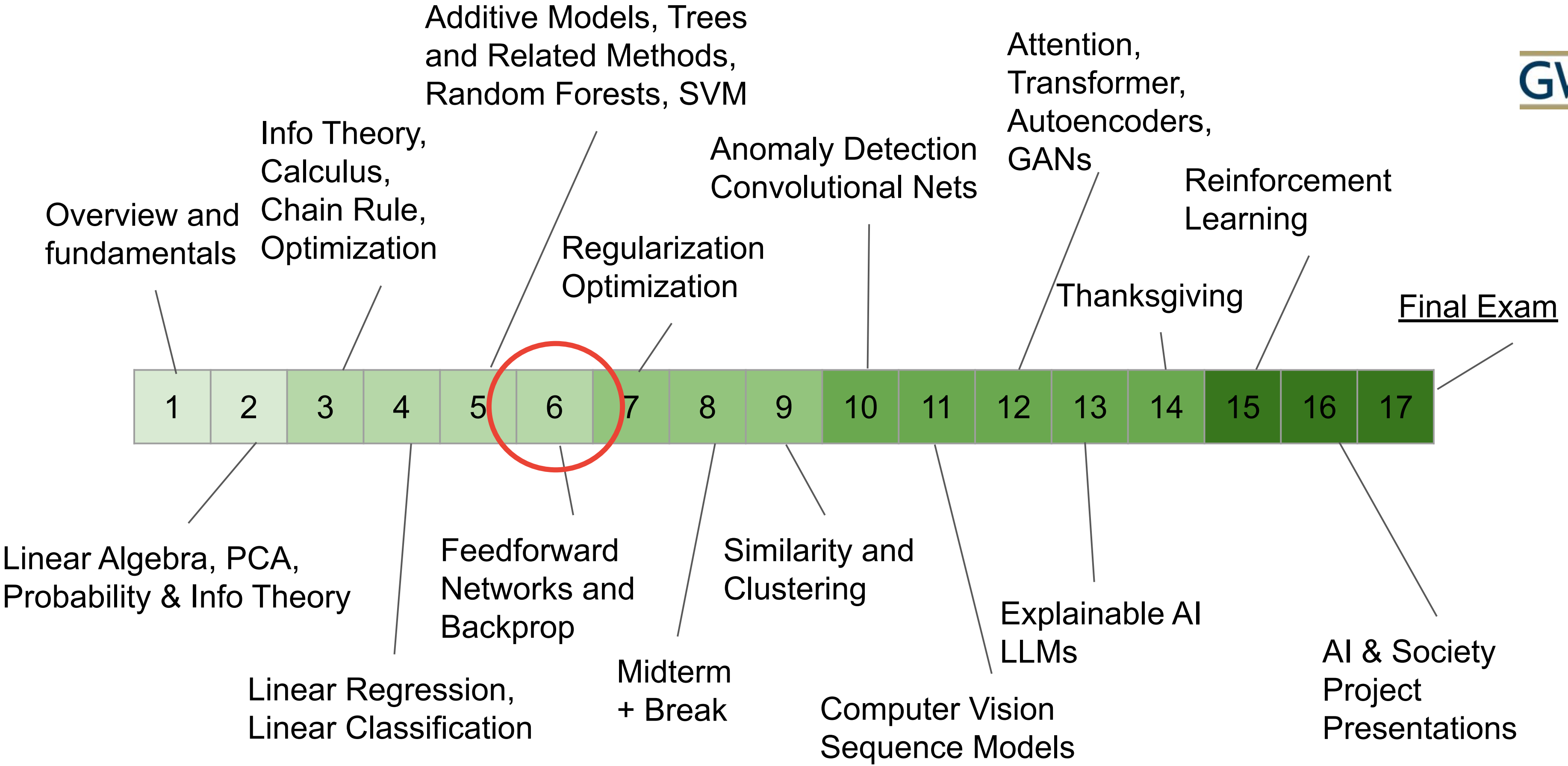
Detect and explain defects in Aircraft telemetry

Dataset: Historical Flights and Detailed Analysis of problematic flights

Extend MADI (github.com/google/madi) with

(a) Autoencoder/GAN based Anomaly Detectors
(b) Additional Explainability Methods

Evaluate both accuracy and explainability


Overview and fundamentals
Info Theory, Calculus, Chain Rule, Optimization
Additive Models, Trees and Related Methods, Random Forests, SVM
Regularization Optimization
Anomaly Detection Convolutional Nets
Attention, Transformer, Autoencoders, GANs
Thanksgiving
Reinforcement Learning
Final Exam
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
Linear Algebra, PCA, Probability & Info Theory
Linear Regression, Linear Classification
Feedforward Networks and Backprop
Midterm + Break
Similarity and Clustering
Computer Vision Sequence Models
Explainable AI LLMs
AI & Society Project Presentations

# Roadmap

Example: Learning XOR

Gradient-based Learning

Output Units

Hidden Units

Architecture Design

# Some basic terminology

Deep Feedforward Networks

- **Deep**: Many interacting layers
- **Feedforward**: information flows to the output prediction with no feedback or recurrence
- **Networks**: Composing many functions in a structured manner

Given three functions $f^{(1)}, f^{(2)}, f^{(3)}$, we can chain them together:

$$f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$$

- Input, Hidden, and Output Layers

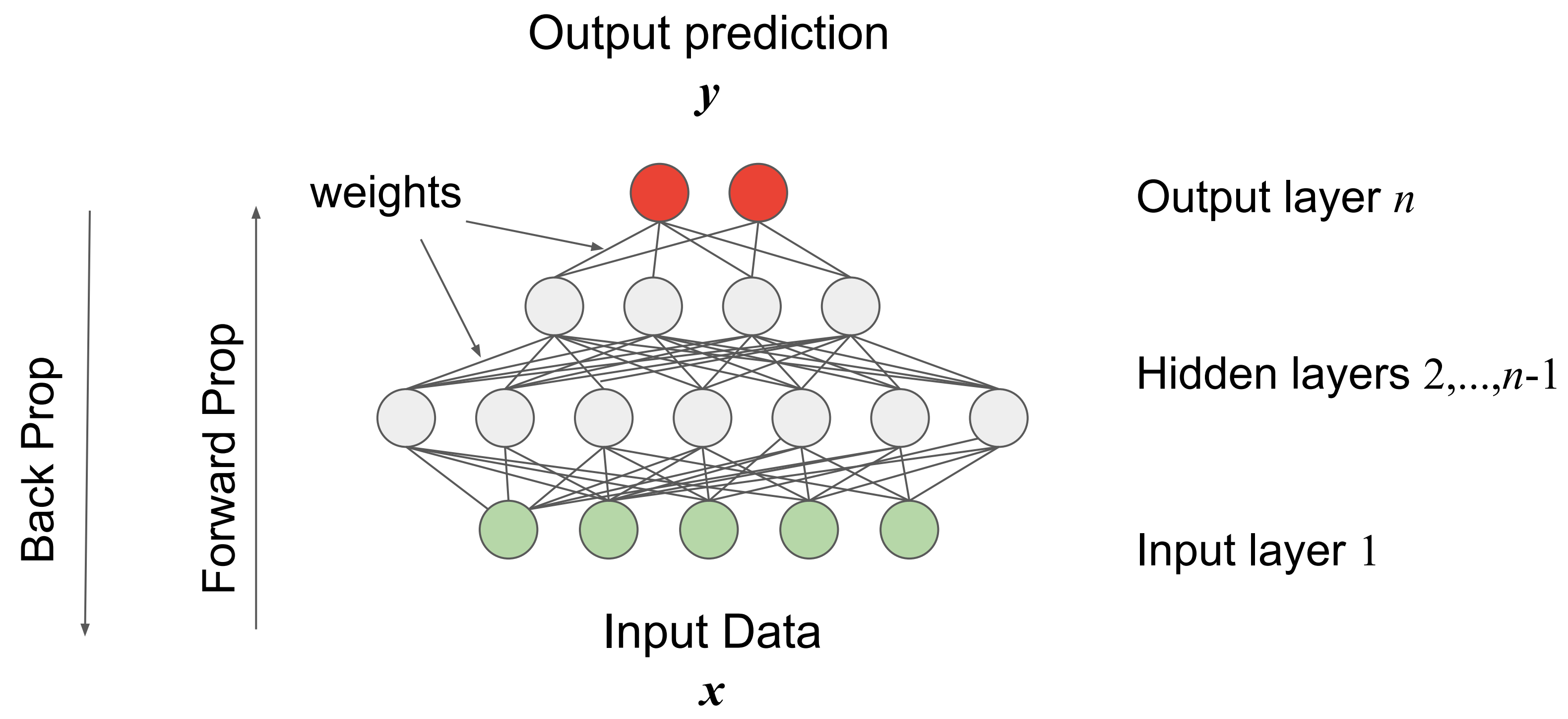
# Three strategies for nonlinear methods

Apply a nonlinear transformation $\boldsymbol{\phi}(\boldsymbol{x})$:

1. Apply a very generic transformation $\boldsymbol{\phi}$ such as the RBF in support vector machines
2. Manually engineer $\boldsymbol{\phi}$ to fit the specific problem
3. Learn $\boldsymbol{\phi}$ by adapting parameters $\boldsymbol{\theta}$:

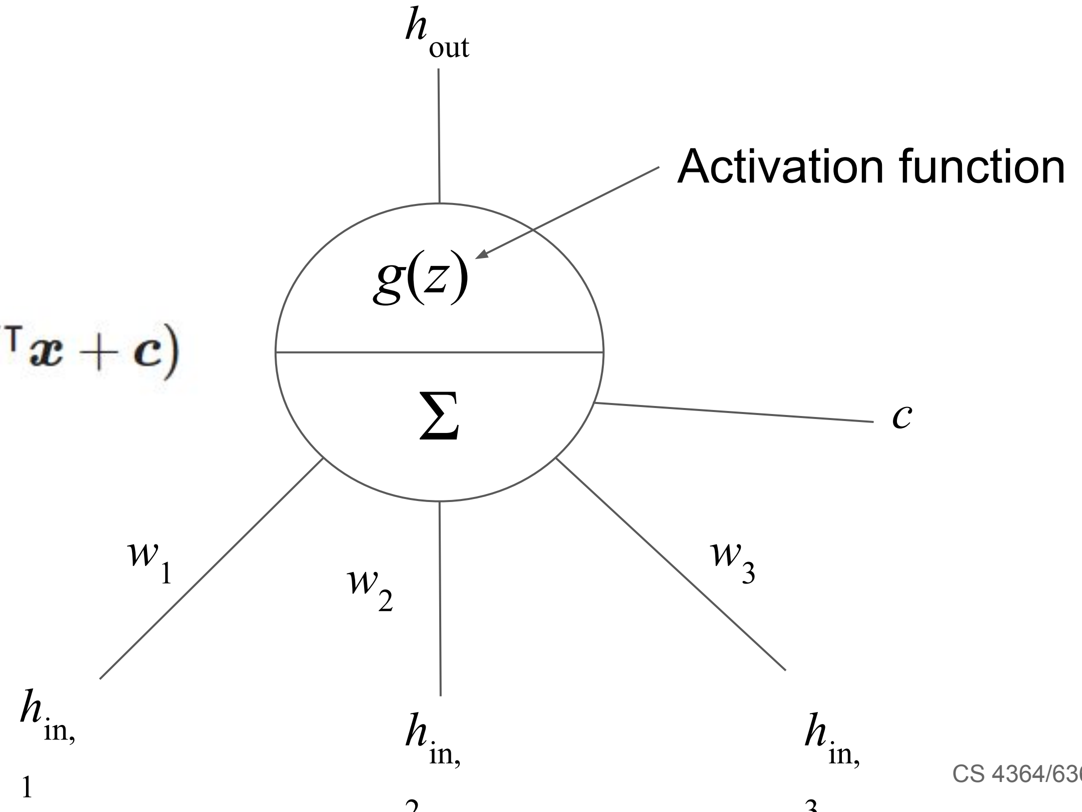
$$y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^\top \boldsymbol{w}$$

# General Architecture of Feedforward Nets

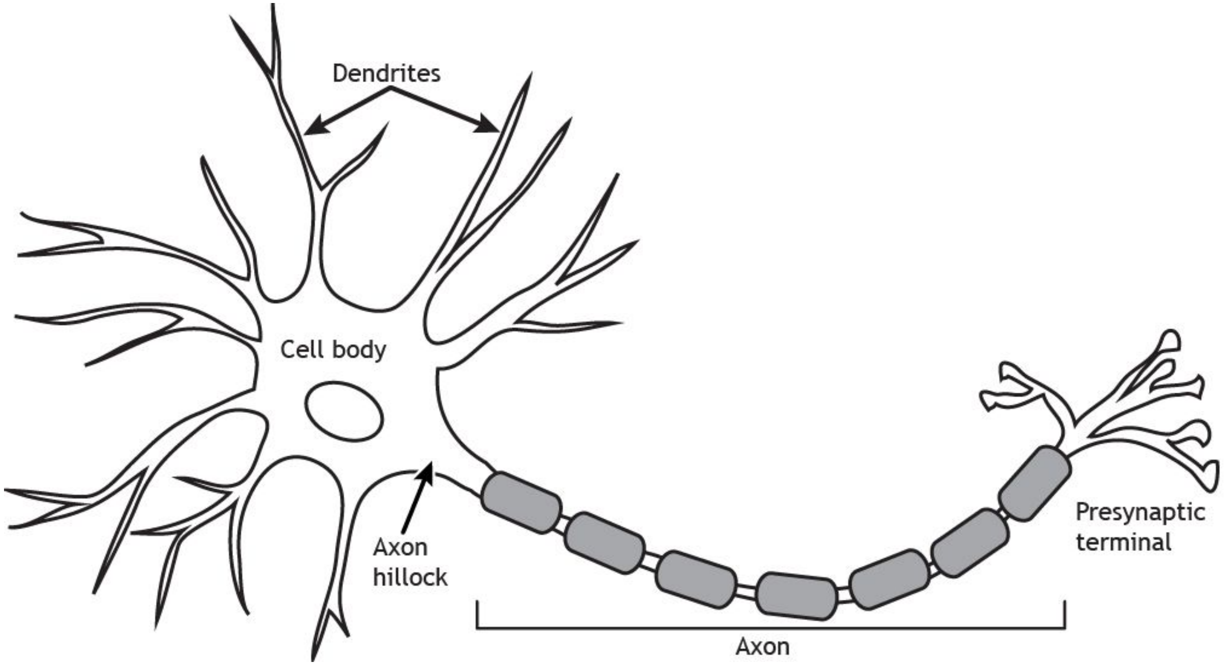


# The basic neural network unit

$$\boldsymbol{h} = g(\boldsymbol{W}^{\top}\boldsymbol{x} + \boldsymbol{c})$$



# The Biological Neuron



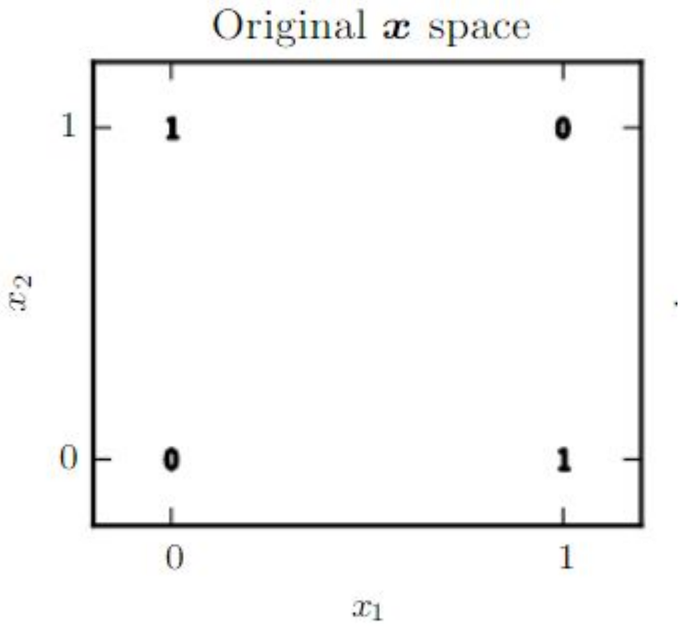

Source: Foundations of Neuroscience
https://openbooks.lib.msu.edu/neuroscience/chapter/the-neuron/

# Example: XOR

# Example: XOR

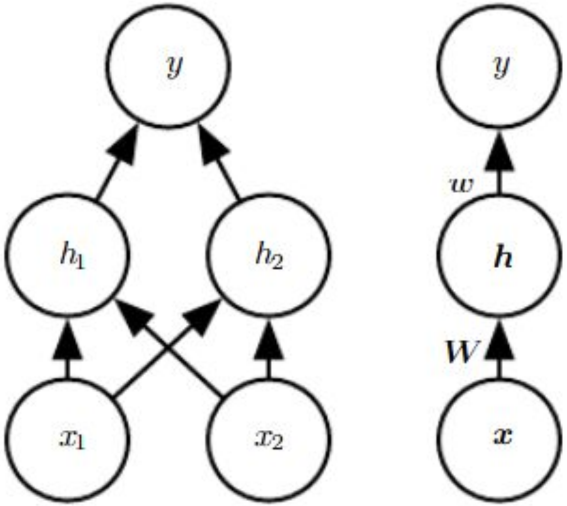
The XOR function is not separable separable by a linear regression model.

$$\mathrm{XOR}(1,0) = 1$$
$$\mathrm{XOR}(0,1) = 1$$
$$\mathrm{XOR}(1,1) = 0$$
$$\mathrm{XOR}(0,0) = 0$$



# Example: XOR

- Introduce a feedforward network with one hidden layer $\boldsymbol{h}$ and 2 hidden units computed by a function $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$.
- Second layer is the output of the network with input $\boldsymbol{h}$, $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$.
- The combined model: $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$

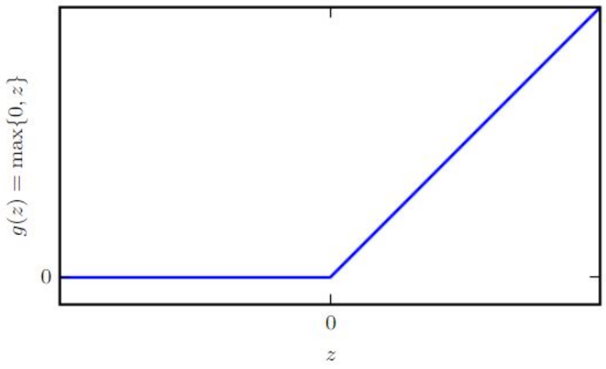


# Example: XOR

- Next we need to add in a nonlinear **activation function** $g$,

$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c})$$

- Choose the **Rectified Linear Unit** ReLU as a simple activation function: $g(z) = \max\{0, z\}$.




- The full network equation is:

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b$$

# Computing the XOR

1. Specify weights and bias:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

# Computing the XOR

2. Write out the XOR design matrix:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

3. Multiply by first layer weights:

$$\mathbf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

# Computing the XOR

4. Add the bias of the first layer:

$$\mathbf{XW} + \mathbf{c} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

5. Apply the nonlinear activation ReLU:

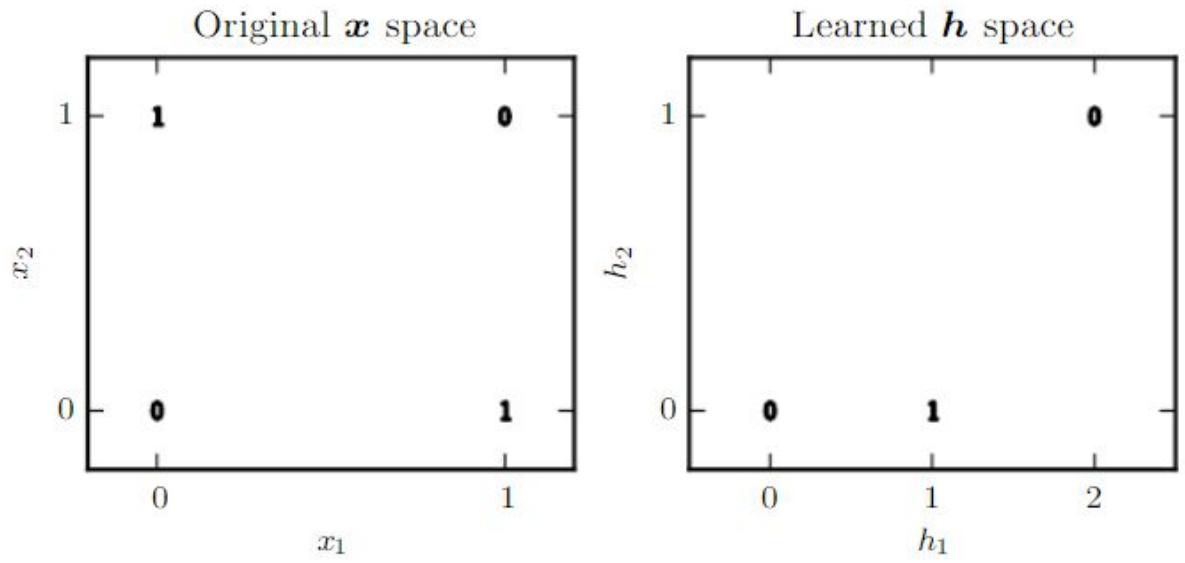
$$\max(0, \mathbf{XW} + \mathbf{c}) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

# Computing the XOR

6. Multiply by the output layer weights:

$$\mathbf{w}^\top \max(0, \mathbf{XW} + \mathbf{c}) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

# Example: XOR

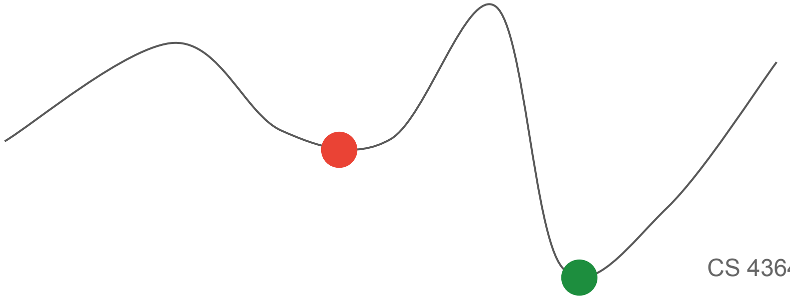



Nonlinear $\boldsymbol{h}$ projects $(0,1)$ and $(1,0)$ to $(1, 0)$ making it linearly separable.

# Gradient-based Learning

# Question

What is the biggest difference between linear models, like logistic regression, and neural networks (besides being nonlinear)?



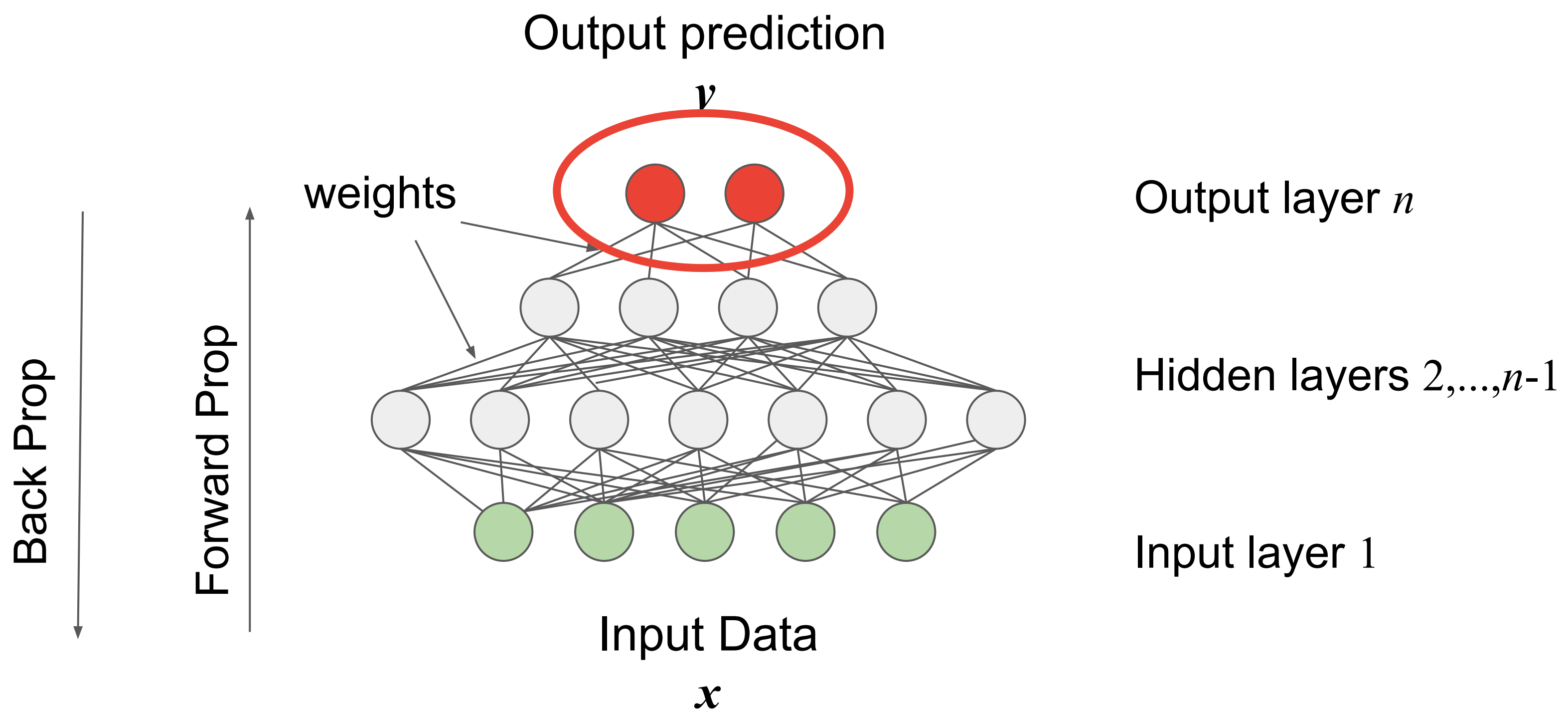
# Cost function considerations

- In most cases, our model defines a distribution $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$
- $\boldsymbol{\theta}$ are the parameters - here weights of the edges connecting the nodes.
- We will also leverage the use of Regularization (like Lasso and Ridge regression.)
- We generally apply the log-likelihood cost function:

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathrm{x,y}\sim\hat{p}_{data}} \log p_{model}(\boldsymbol{y}|\boldsymbol{x})$$

# Output Units

# General Architecture of Feedforward Nets



# Linear Output Units

- Simple **affine** transformation with no non-linearity:

$$\hat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$$

- Commonly used to predict the mean of a Gaussian distribution:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

- Very stable under gradient optimization

# Linear Units

How many linear units are required to create a nonlinear decision boundary?

# Sigmoid Output Units

- Binary classification with Bernoulli distribution
- Same technique as logistic regression to constrain the output between $0$ and $1$
- Linear Output Unit + Sigmoid transformation:

$$\hat{y} = \sigma(z) = \sigma(\boldsymbol{w}^\top + b)$$

$$\log \tilde{P}(y) = yz \text{ where } y \in \{0, 1\}$$

$$\tilde{P}(y) = \exp(yz)$$

$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)}$$

$$P(y) = \sigma((2y - 1)z)$$

# Softmax Output Unit

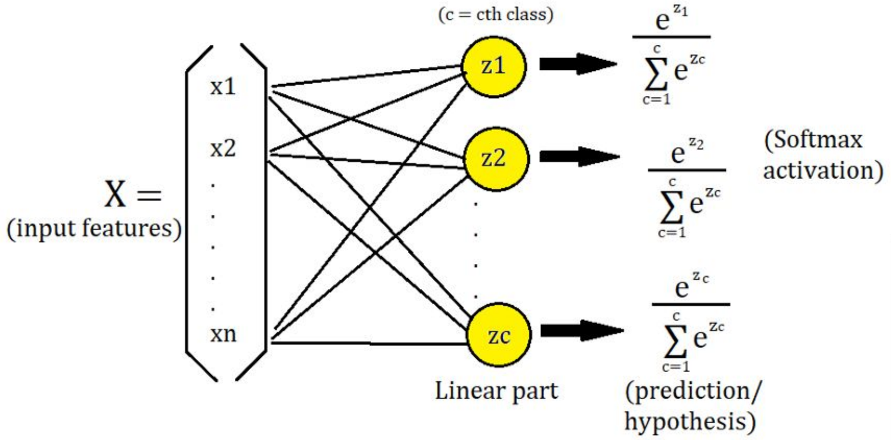
- Generalization of the sigmoid for $k > 2$ classes
- With $k - 1$ output units:

$$\boldsymbol{z} = \boldsymbol{W}^{\top}\boldsymbol{h} + \boldsymbol{b}$$

$$\text{where } z_i = \log \tilde{P}(y = i \mid \boldsymbol{x})$$

- Then the softmax function is:

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j^k \exp(z_j)}$$

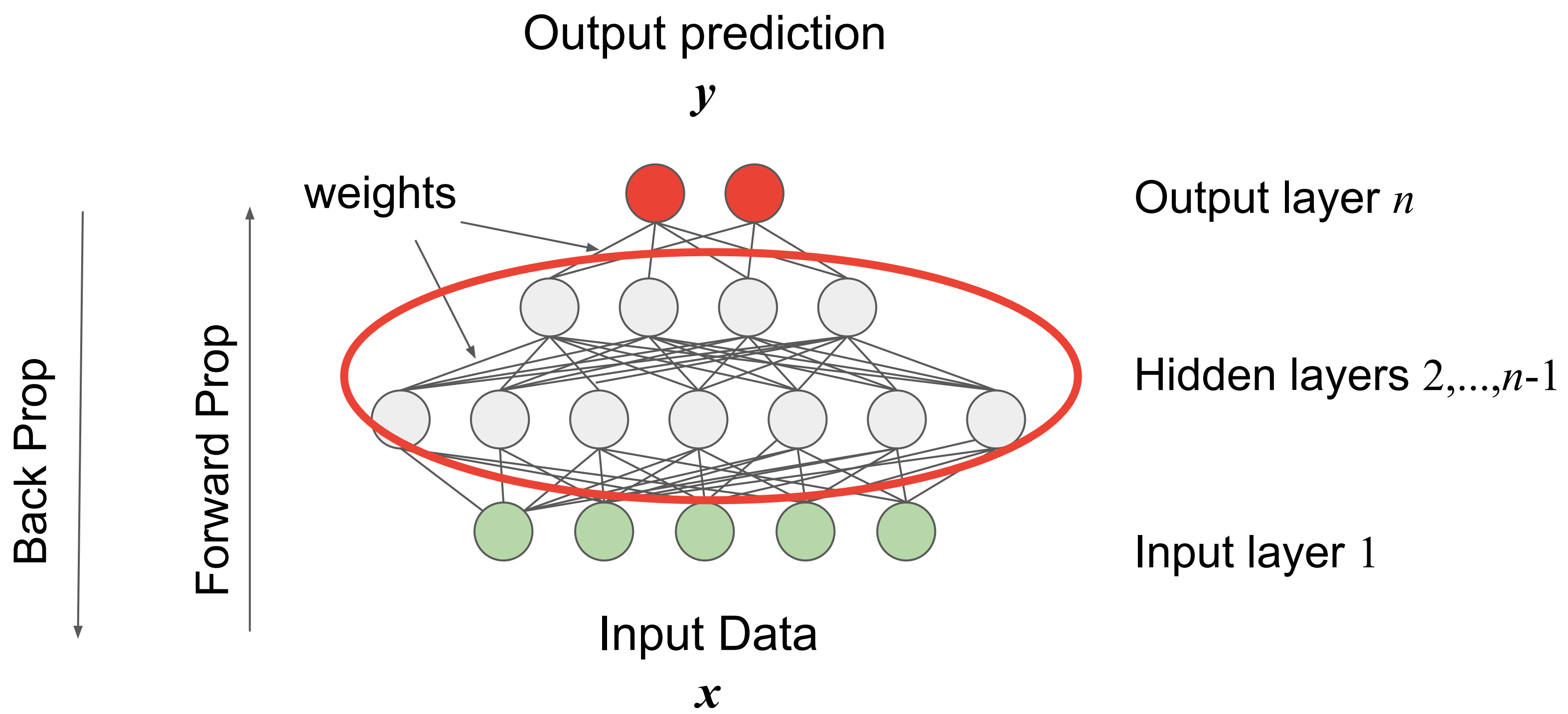


# Matching the output unit and loss function

| Output Type | Output Distribution | Output Layer | Cost Function |
|---|---|---|---|
| Binary | Bernoulli | Sigmoid | Binary cross-entropy |
| Discrete | Multinoulli | Softmax | Discrete cross-entropy |
| Continuous | Gaussian | Linear | Gaussian cross-entropy (MSE) |
| Continuous | Mixture of Gaussian | Mixture Density | Cross-entropy |
| Continuous | Arbitrary | See part III: GAN, VAE, FVBN | Various |

# Hidden Units

# General Architecture of Feedforward Nets



# Rectified Linear Unit

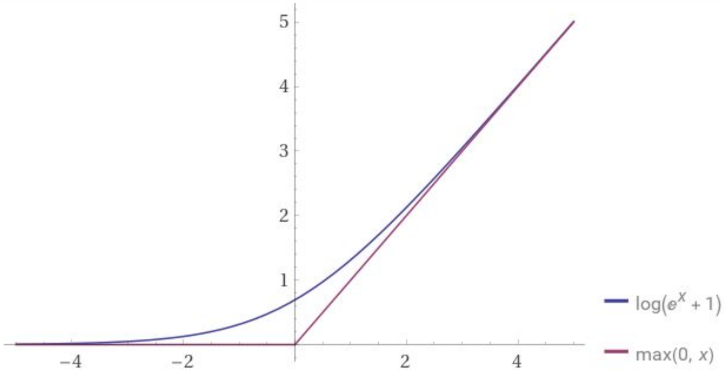
- ReLU activation: $g(z) = \max\{0, z\}$
- Not differentiable, but works well in practice:
$\frac{dg(0)}{dz}|_- \neq \frac{dg(0)}{dz}|_+$
- Used on top of an affine transformation:

$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$$

- Three variations on ReLU:

$$h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max\{0, z_i\} + \alpha_i \min\{0, z_i\}$$

1. Absolute Value: $\alpha_i = -1$
2. Leaky ReLU: $\alpha_i = \text{small const}$
3. Parametric Relu: $\alpha$ is learned



# Sigmoid and Hyperbolic Tangent

**Saturation**: Zero Gradient, no optimization

- Sigmoid $g(z) = \sigma(z)$ also be used as a hidden unit!
- Pre-dates the ReLU
- Saturate across most of the domain, which makes training difficult
- An alternative to use is hyperbolic tangent $g(z) = \tanh(z)$

# Maxout Unit

Divide the domain of $z$ in to $k$ values (windows), and return the max response.

Approximates any convex function.

$$h_i(x) = \max_{j\in[1,k]} z_{ij}$$

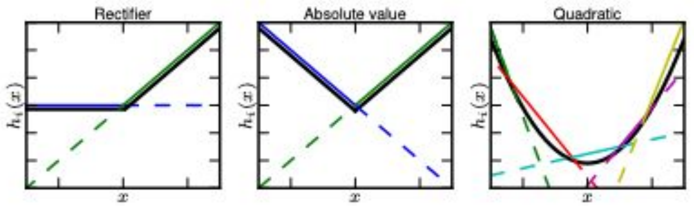



*Figure 1.* Graphical depiction of how the maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. This diagram is 2D and only shows how maxout behaves with a 1D input, but in multiple dimensions a maxout unit can approximate arbitrary convex functions.

Maxout Networks (2013), Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio
https://arxiv.org/abs/1302.4389

# Other Hidden Units

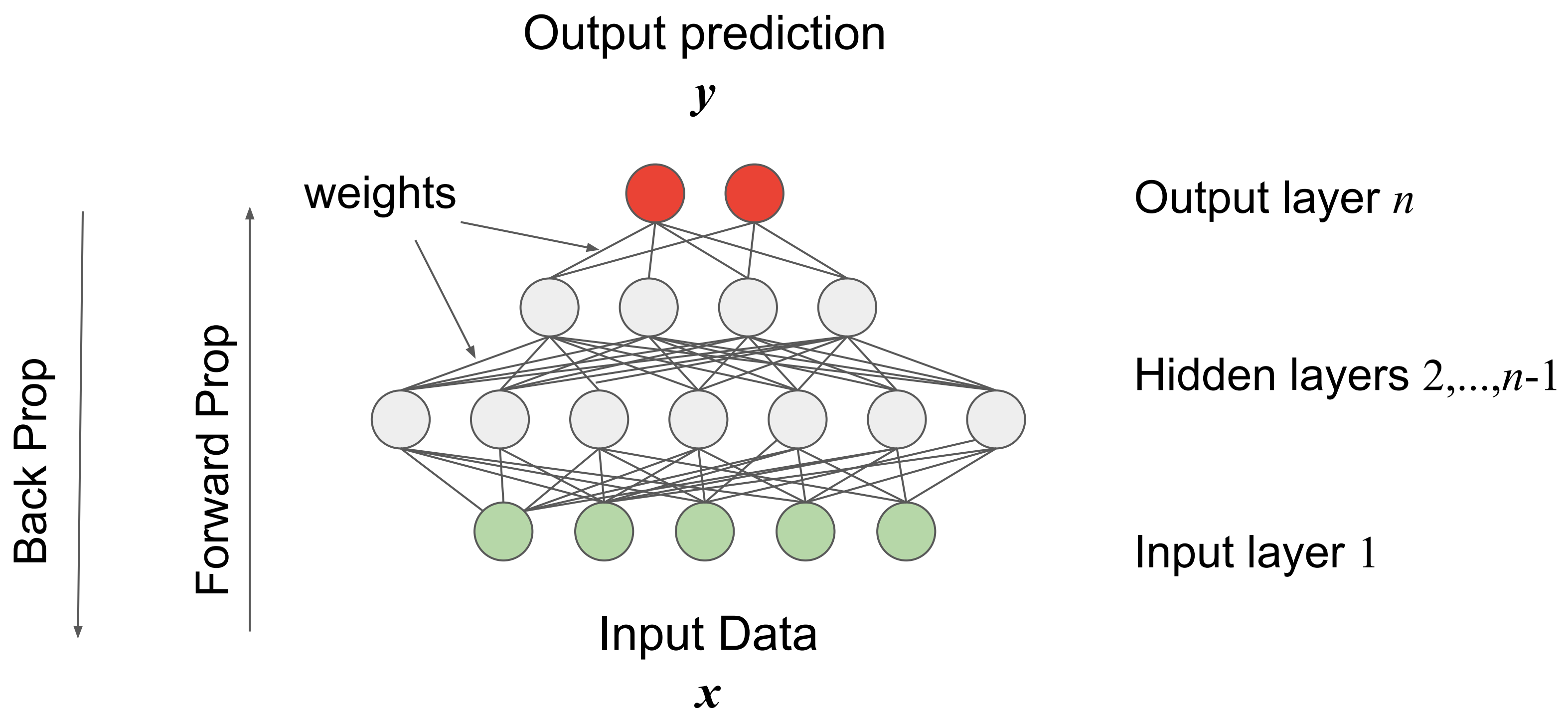
- There are many types of hidden units, but generaly don't perform significantly better that ReLU
- **Radial Basis Function** (RBF):

$$h_i = \exp\left(-\frac{1}{\sigma_i^2 ||\boldsymbol{W}_{:,i} - \boldsymbol{x}||^2}\right)$$

- **Softplus**: $g(z) = \log(1 + \exp z)$
- **Hard tanh**: $g(z) = \max(-1, min(1, a))$

# Architectural Design

# General Architecture of Feedforward Nets



# Architecture

**Architecture**: how many units and how are they connected?

Connected via **layers**:

- Input Layer: $\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$
- Second Layer: $\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{x} + \boldsymbol{b}^{(2)}\right)$
- $k$-th Layer: $\boldsymbol{h}^{(k)} = g^{(k)}\left(\boldsymbol{W}^{(k)\top}\boldsymbol{x} + \boldsymbol{b}^{(k)}\right)$

# Readings

- Goodfellow - Chapter 6