# Lecture I:
# Introduction to Machine Learning

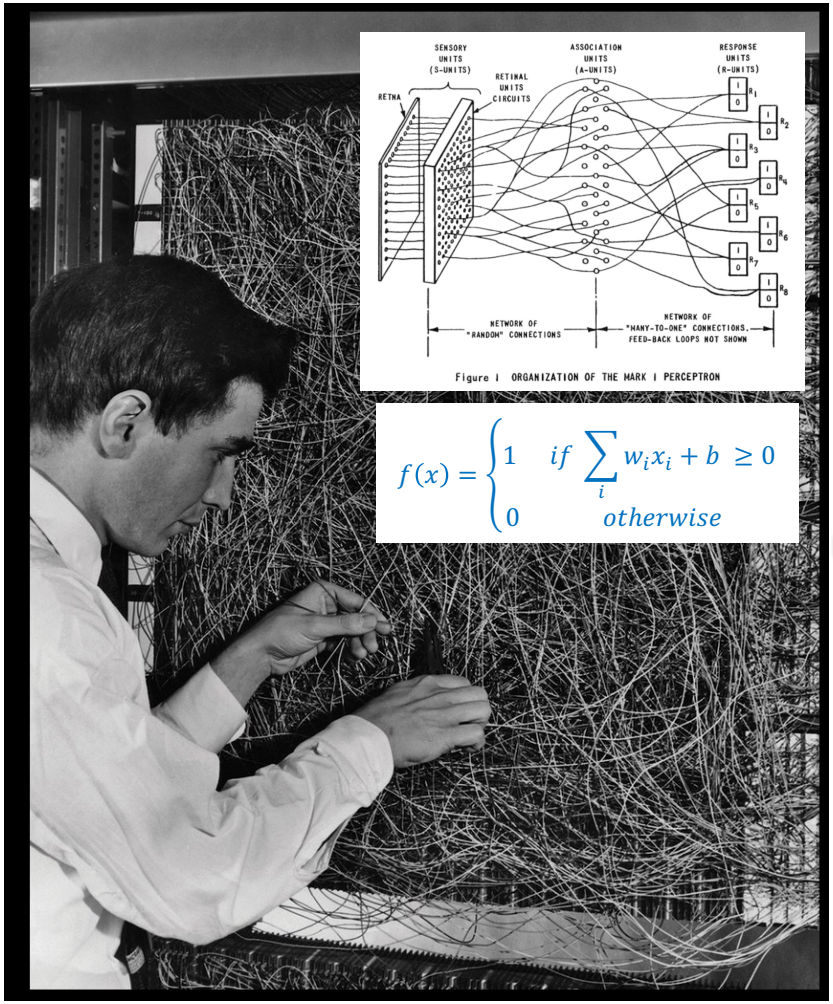
Michael Kagan

SLAC

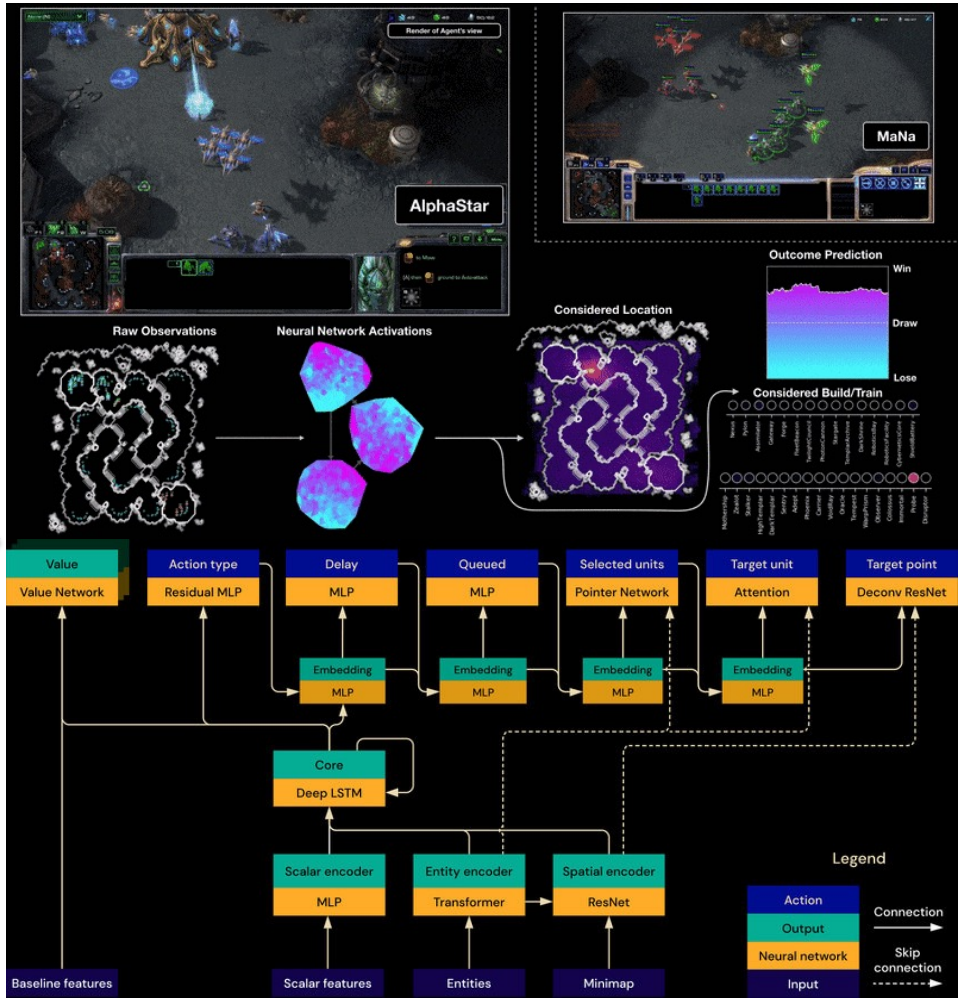
CERN Openlab Summer Student Lectures
July 13, 2023

# Long History of Machine Learning






Perceptron




AlphaStar

Rosenblatt 1958, 1960

Vinyals et. al. 2019

# The Power of ML is Changing Quickly



ChatGPT

Stable Diffusion

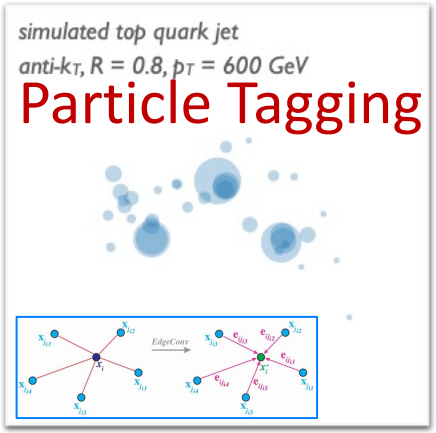
GitHub Copilot

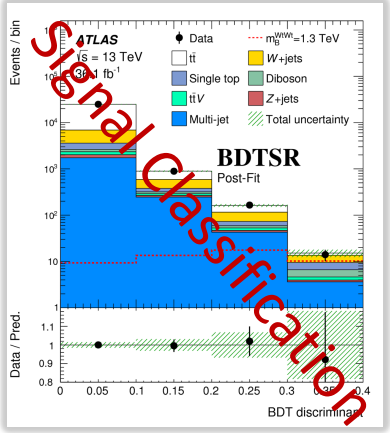
**Prompt:**

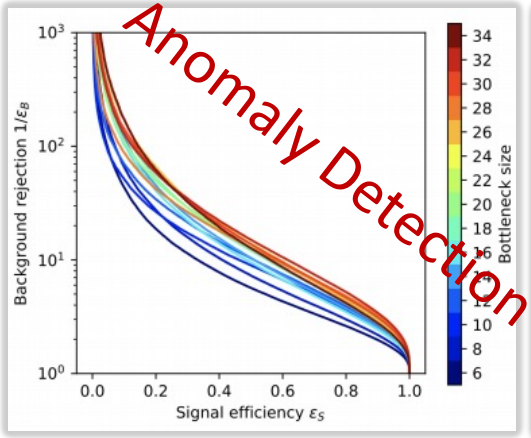
***street style photo of a woman selling pho at a Vietnamese street market, sunset, shot on fujifilm***

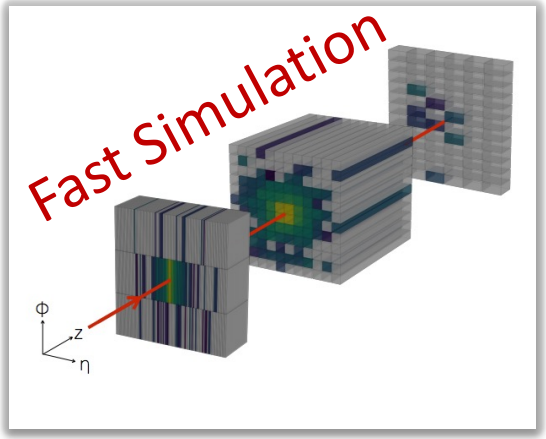
# Machine Learning in HEP

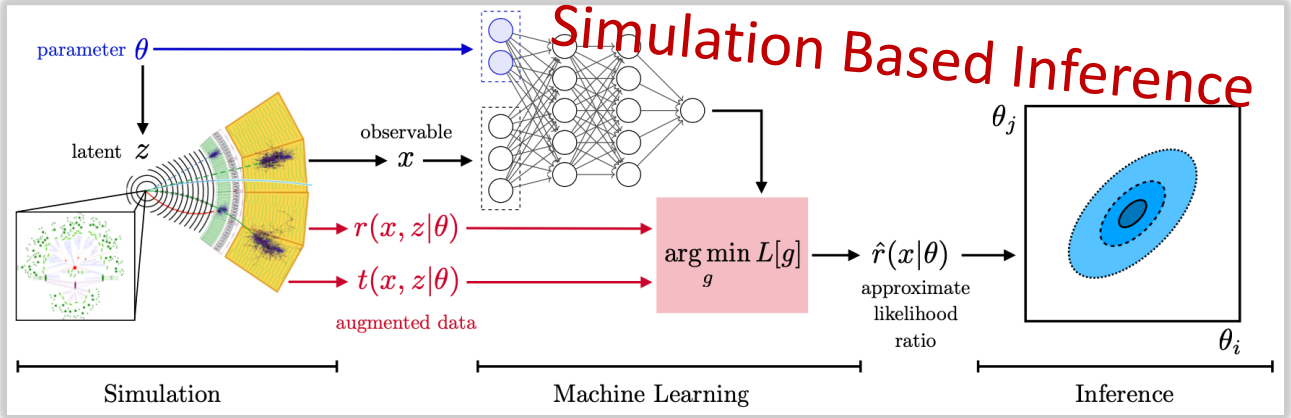


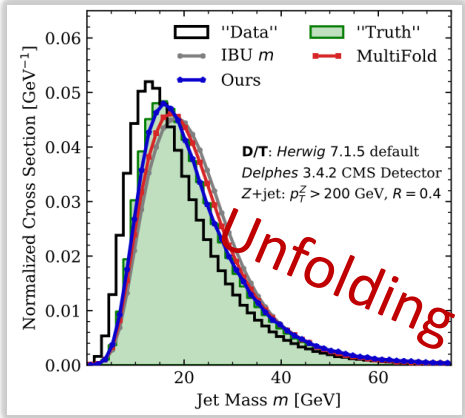


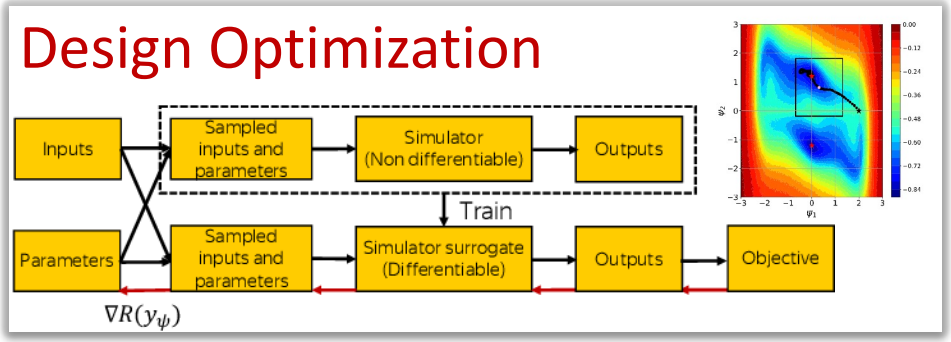



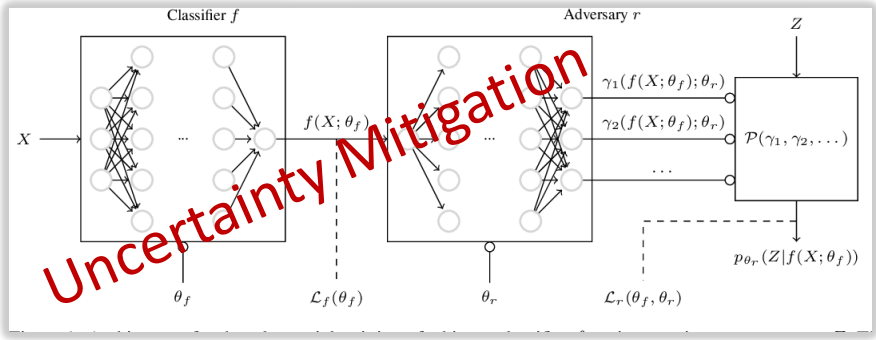




**+ More! Check out** [The Living Review of ML in HEP]

# What is Machine Learning?



- Giving computers the ability to learn without explicitly programming them (Arthur Samuel, 1959)
- Statistics + Algorithms
- Computer Science + Probability + Optimization
- **Fitting data with complex functions**
- **Mathematical models** learnt from data that characterize the patterns, regularities, and relationships amongst variables in the system

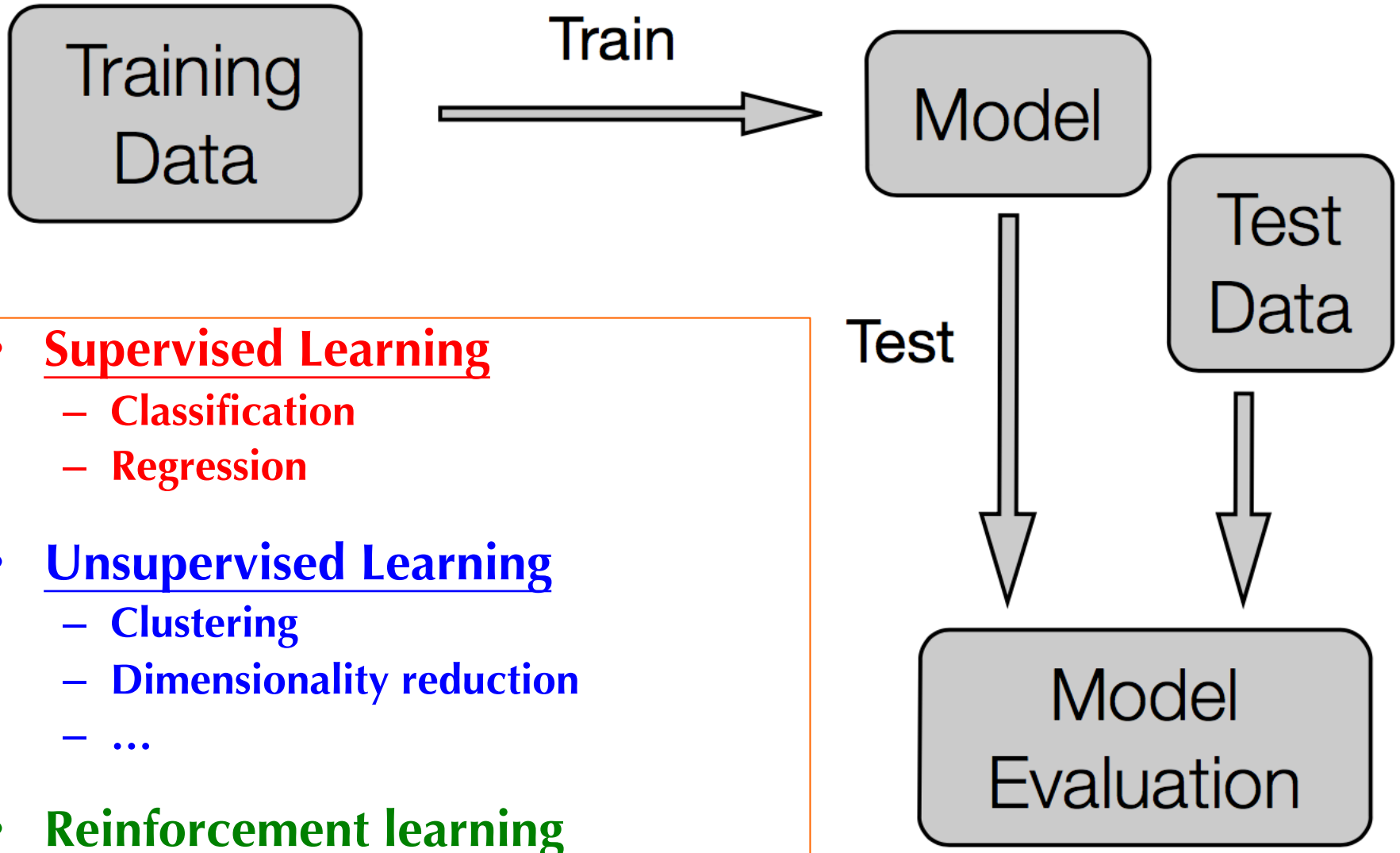
# Machine Learning: Models



- Key element is a **mathematical model**
  - A mathematical characterization of system(s) of interest, typically via random variables
  - Chosen model depends on the task / available data
- **Learning**: estimate statistical model from data
  - Supervised learning
  - Unsupervised Learning
  - Reinforcement Learning
  - …
- **Prediction and Inference:** use statistical model to make predictions on new data points & infer properties of system

# Learning






- **Supervised Learning**
  - Classification
  - Regression
- **Unsupervised Learning**
  - Clustering
  - Dimensionality reduction
  - …
- **Reinforcement learning**

[Ravikumar]

# Probability Review



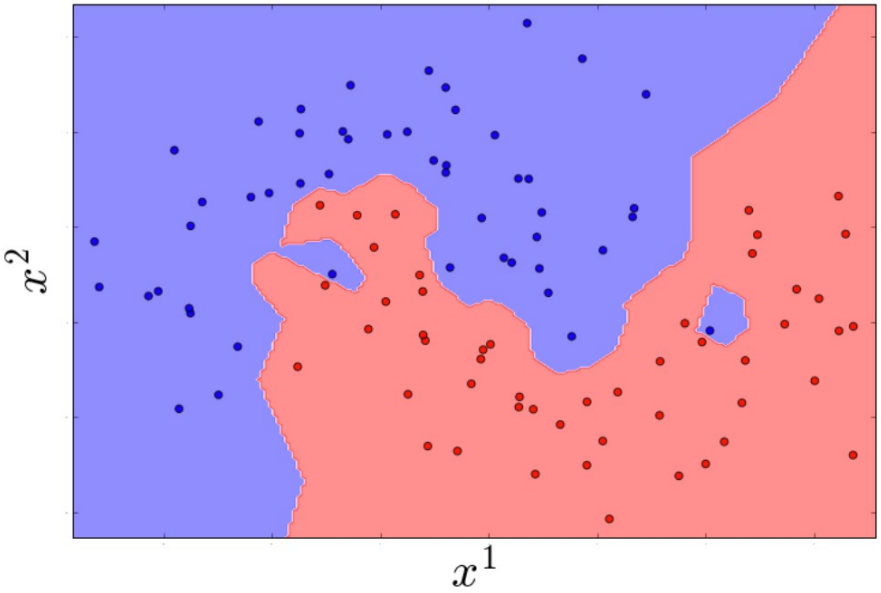
- Joint distribution of two variables: $p(x, y)$
- Marginal distribution: $p(x) = \int p(x, y)dy$
- Conditional distribution: $p(y|x) = \frac{p(x, y)}{p(x)}$
- Bayes theorem: $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
- Expected value: $\mathbf{E}[f(x)] = \int f(x)p(x)\mathrm{d}x$
- Normal distribution:
  - $x \sim N(\mu, \sigma) \quad \rightarrow \quad p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$

# Supervised Learning

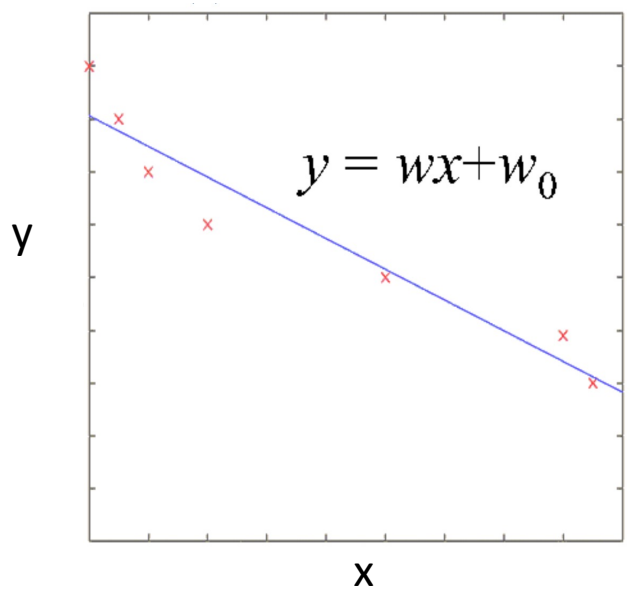


- Given N examples with observable features $\{x_i \in \mathcal{X}\}$ and prediction **targets** $\{y_i \in \mathcal{Y}\}$, learn function mapping **h(x)=y**

**Classification**:
$\mathcal{Y}$ is a finite set of **labels** (i.e. classes) denoted with integers




**Regression**:
$\mathcal{Y}$ is a real number



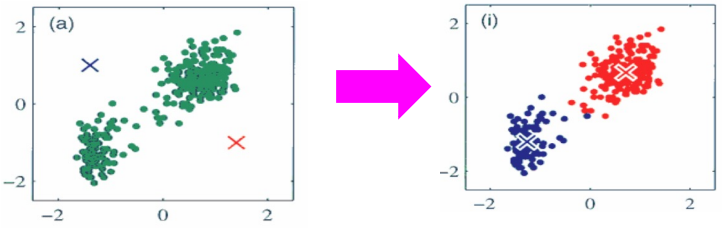
# Unsupervised Learning



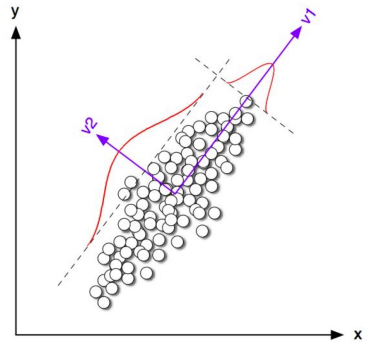
Given data D=$\{x_i\}$, but no labels, find structure in data

[Bishop]

**Clustering**: partition the data into groups $D = \{D_1 \cup D_2 \cup D_3 \ \ldots \cup D_k\}$




**Dimensionality reduction**: find a low dimensional (less complex) representation of the data with a mapping $Z = h(X)$




**Density estimation and sampling**: estimate the PDF $p(x)$, and/or learn to draw plausible new samples of x

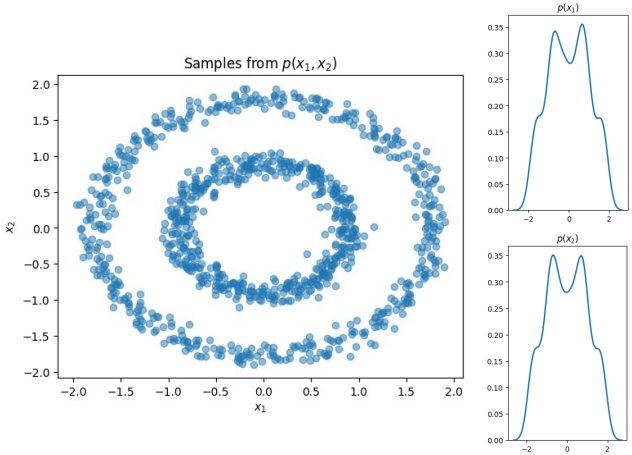


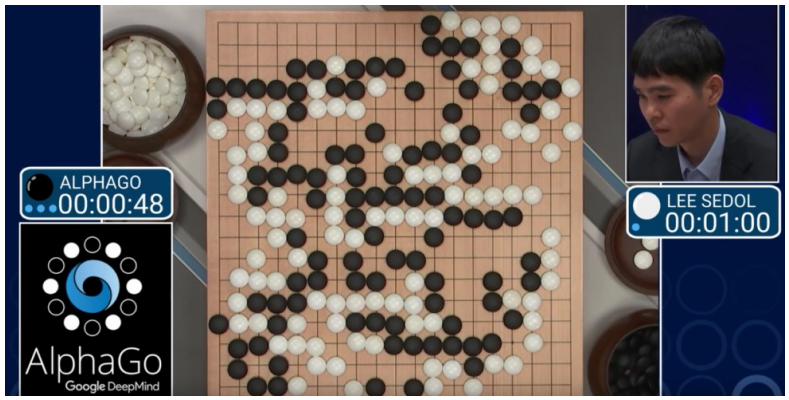

Image Credit - Link

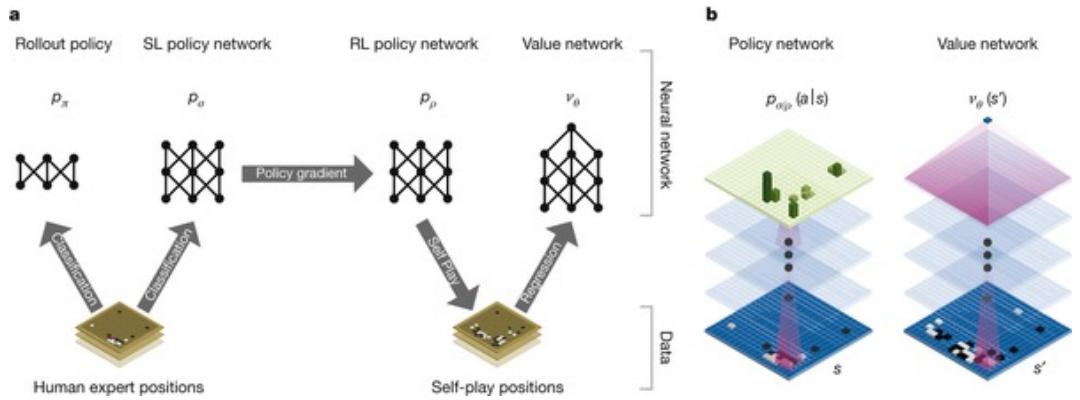
# Reinforcement Learning




[Ravikumar]

- Model agents that take actions depending on state
  - Actions incur rewards, and affect future states ("feedback")
- Learn to make the best sequence of decisions to achieve a given goal when feedback is often delayed until you reach the goal



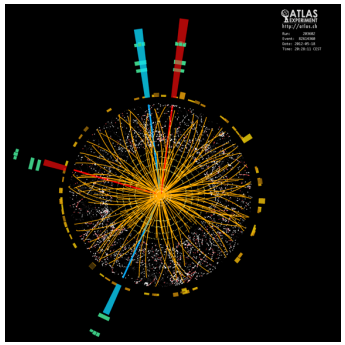




Nature 529, 484–489 (28 January 2016)

# Supervised Learning: How does it work?

# Supervised Learning: How does it work?





$h(\boldsymbol{x};\ \boldsymbol{w})$
Function with adjustable parameters

Loss Function

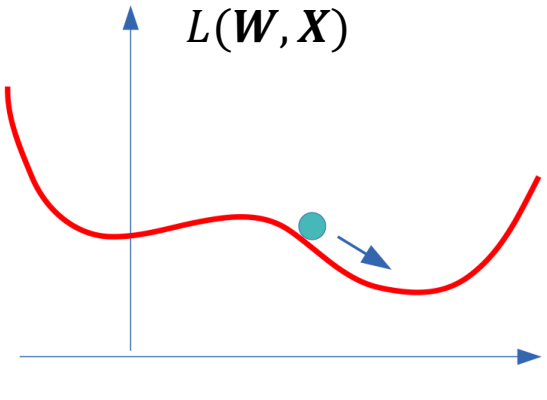
Compare prediction with true label

Loss

True labels:
Higgs = 1
Bkg = 0

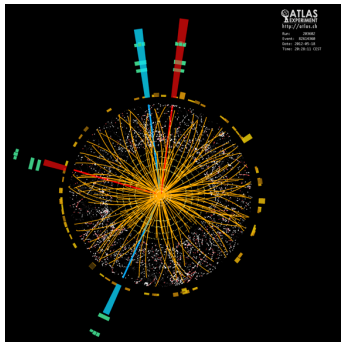
- Design function with adjustable parameters
- Design a Loss function
- Find best parameters which minimize loss

Y. Le Cun



# Supervised Learning: How does it work?





$h(\boldsymbol{x};\ \boldsymbol{w})$
Function with adjustable parameters

Loss Function

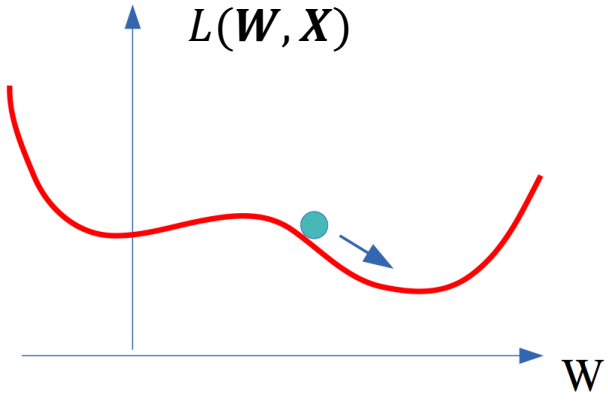
Compare prediction with true label

Loss

True labels:
Higgs = 1
Bkg = 0

- Design function with adjustable parameters
- Design a Loss function
- Find best parameters which minimize loss
  - Use a labeled *training-set* to compute loss
  - Adjust parameters to reduce loss function
  - Repeat until parameters stabilize

Y. Le Cun



# Reminder: Empirical Risk Minimization



$$\arg\min_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{N} L(h(\mathbf{x}_i;\mathbf{w}), y_i) + \lambda\Omega(\mathbf{w})$$

Average expected loss

Model regularization

- Framework to design learning algorithms
  - $L$ is a loss function comparing prediction $h(\cdot)$ w/ target $y$
  - $\Omega(\boldsymbol{w})$ is a regularizer, penalizing certain values of $\mathbf{w}$
    - $\lambda$ controls how much penalty: a hyperparameter we have to tune
- Learning is cast as an optimization problem

# Example Loss Functions



- Square Error Loss:
  - Often used in regression

$$L(h(\mathbf{x};\mathbf{w}),y) = \big(h(\mathbf{x};\mathbf{w}) - y\big)^2$$

- Cross entropy:
  - With $y \in \{0,1\}$
  - Often used in classification

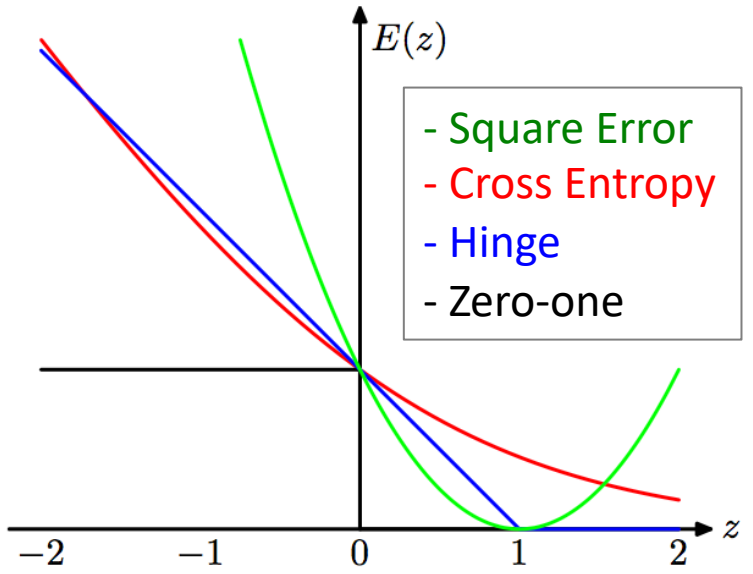
$$L(h(\mathbf{x};\mathbf{w}),y) = -\,y \log h(\mathbf{x};\mathbf{w}) - (1-y)\log(1-h(\mathbf{x};\mathbf{w}))$$

- Hinge Loss:
  - With $y \in \{-1,1\}$

$$L(h(\mathbf{x};\mathbf{w}),y) = \max(0, 1 - yh(\mathbf{x};\mathbf{w}))$$

- Zero-One loss
  - $h(\boldsymbol{x};\ \boldsymbol{w})$ predicting label

$$L(h(\mathbf{x};\mathbf{w}),y) = 1_{y \neq h(\mathbf{x};\mathbf{w})}$$




[Bishop]

# Maximum Likelihood



- Describe a process behind the data
- Write down the likelihood of the observed data

$$\mathcal{L}(\mathbf{w}) = p(\mathbf{y}|\mathbf{X};\mathbf{w}) = \prod_i p(y_i|\mathbf{x}_i;\mathbf{w})$$

  - Where second equality holds if data is independent and identically distributed

  - Often minimize negative-log-likelihood for numerical stability
    - Same as maximizing likelihood since log is monotonic and differentiable away from zero

# Maximum Likelihood



- Describe a process behind the data
- Write down the likelihood of the observed data

$$\mathcal{L}(\mathbf{w}) = p(\mathbf{y}|\mathbf{X};\mathbf{w}) = \prod_i p(y_i|\mathbf{x}_i;\mathbf{w})$$

- Select parameters that make data most likely
  - General strategy for parameter estimation

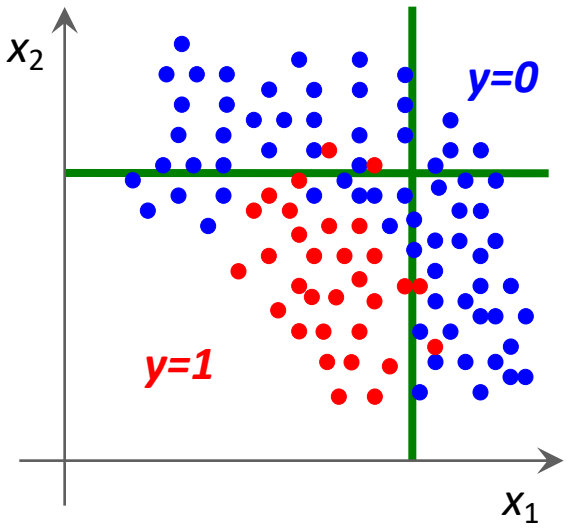
$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \arg\min_{\mathbf{w}} -\ln \mathcal{L}(\mathbf{w}) = \arg\min_{\mathbf{w}} -\sum_i \ln p(y_i|\mathbf{x}_i;\mathbf{w})$$

# Linear Classification

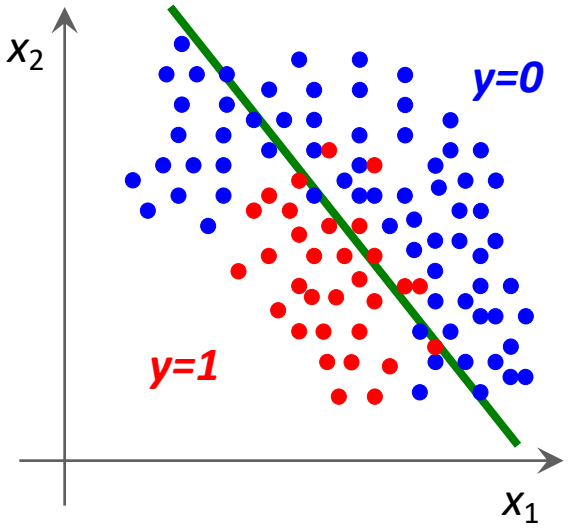
# Classification



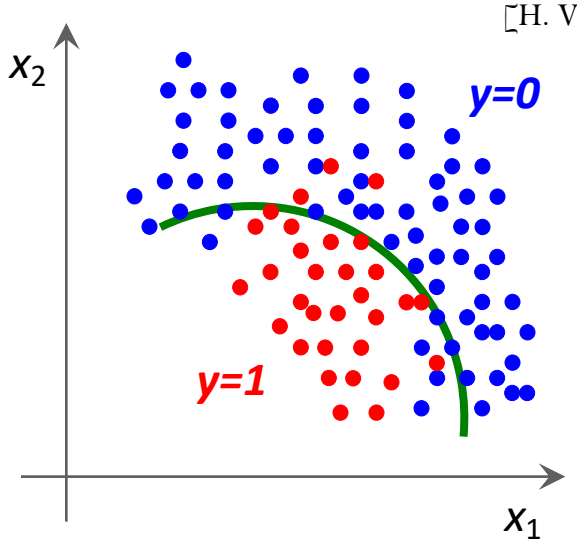
[H. Voss]




Rectangular cuts

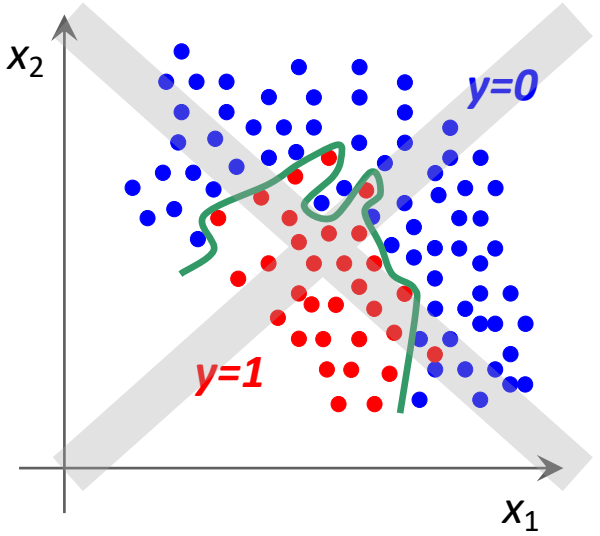



Linear discriminant




Nonlinear discriminant

- Learn a function to separate different classes of data
- Avoid over-fitting:
  - Learning too fined details about your training sample that will not generalize to unseen data



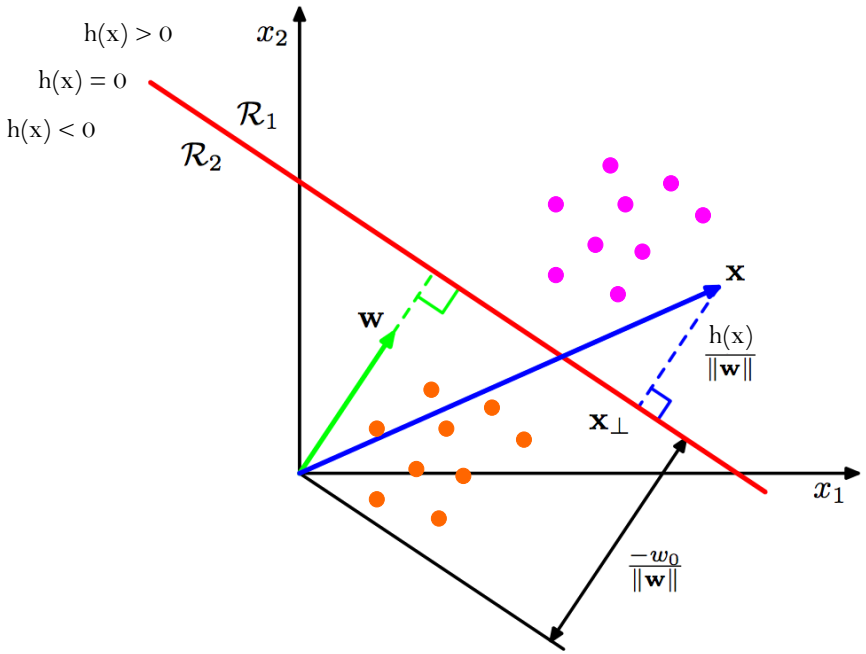
# Linear Decision Boundaries



- Separate two classes:
  - $\boldsymbol{x}_i \in \mathbb{R}^m$
  - $y_i \in \{-1,1\}$

- Linear discriminant model

$$h(\boldsymbol{x};\, \boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} + b$$




[Bishop]

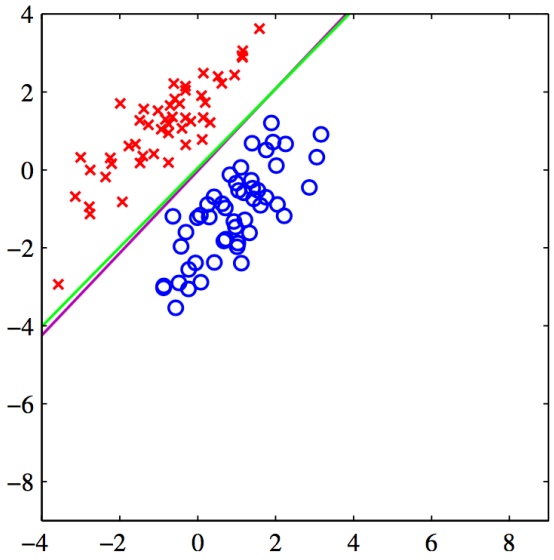
- **Decision boundary** defined by hyperplane

$$h(\boldsymbol{x};\, \boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} + b = 0$$

- Class predictions: Predict class 0 if $h(\boldsymbol{x}_i\,;\, \boldsymbol{w}) < 0$, else class 1

# Linear Classifier with Least Squares?





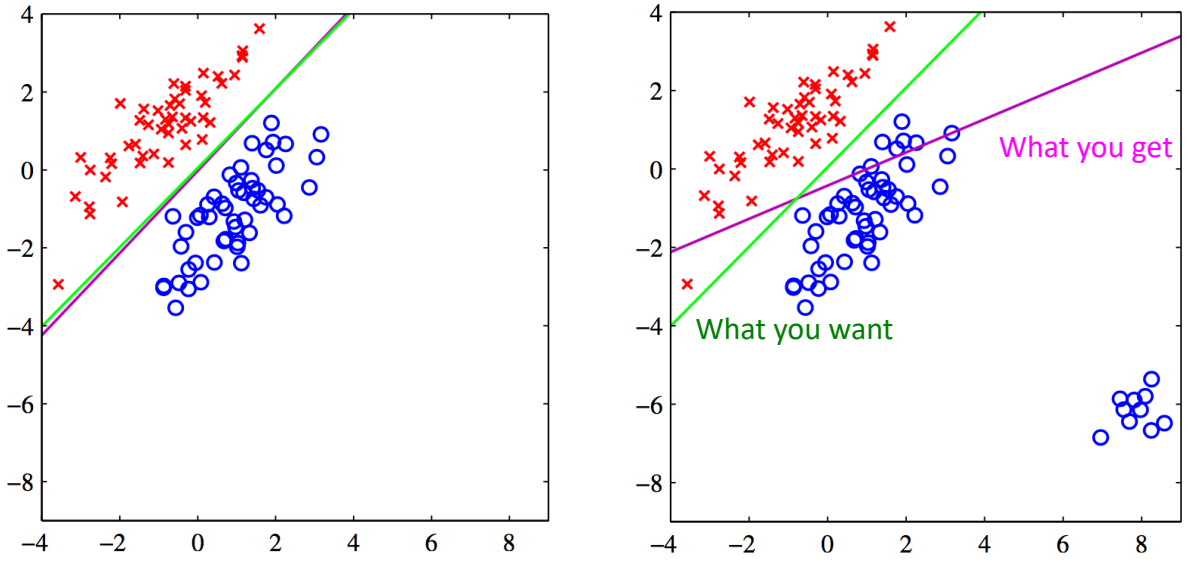

$$L(\mathbf{w}) = \frac{1}{2}\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

[Bishop]

- Why not use least squares loss with binary targets?

# Linear Classifier with Least Squares?






$$L(\mathbf{w}) = \frac{1}{2}\sum_i (y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

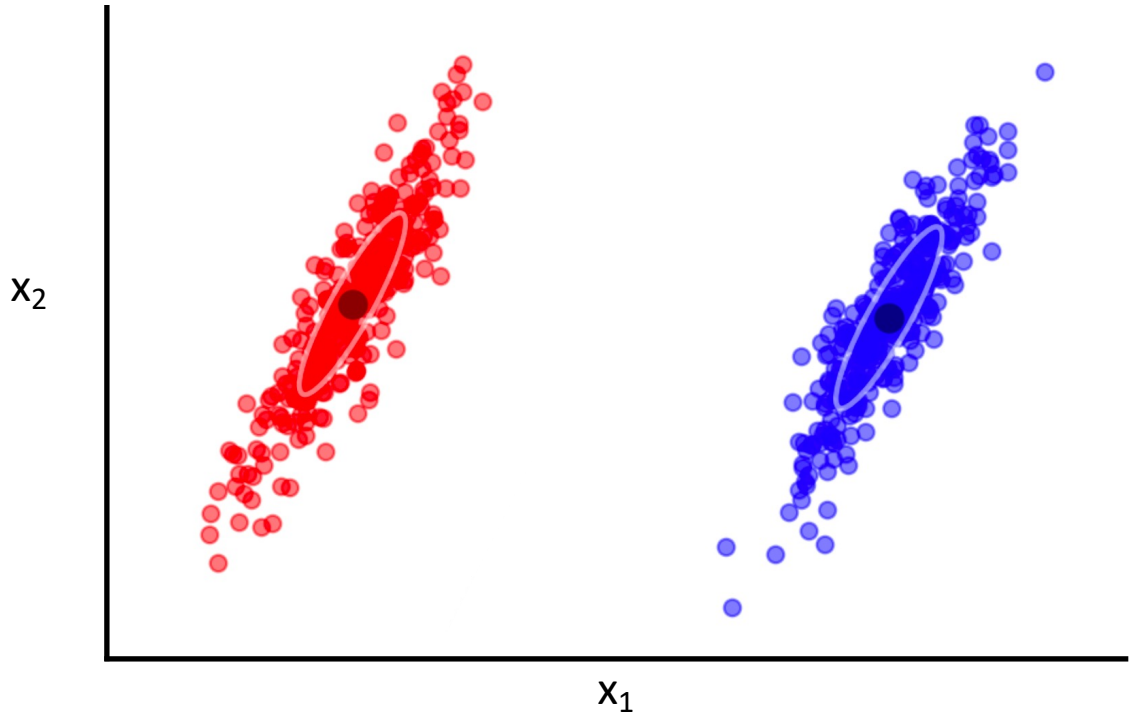
[Bishop]

- Why not use least squares loss with binary targets?
  - Penalized even when predict class correctly
  - Least squares is very sensitive to outliers

# Linear Discriminant Analysis



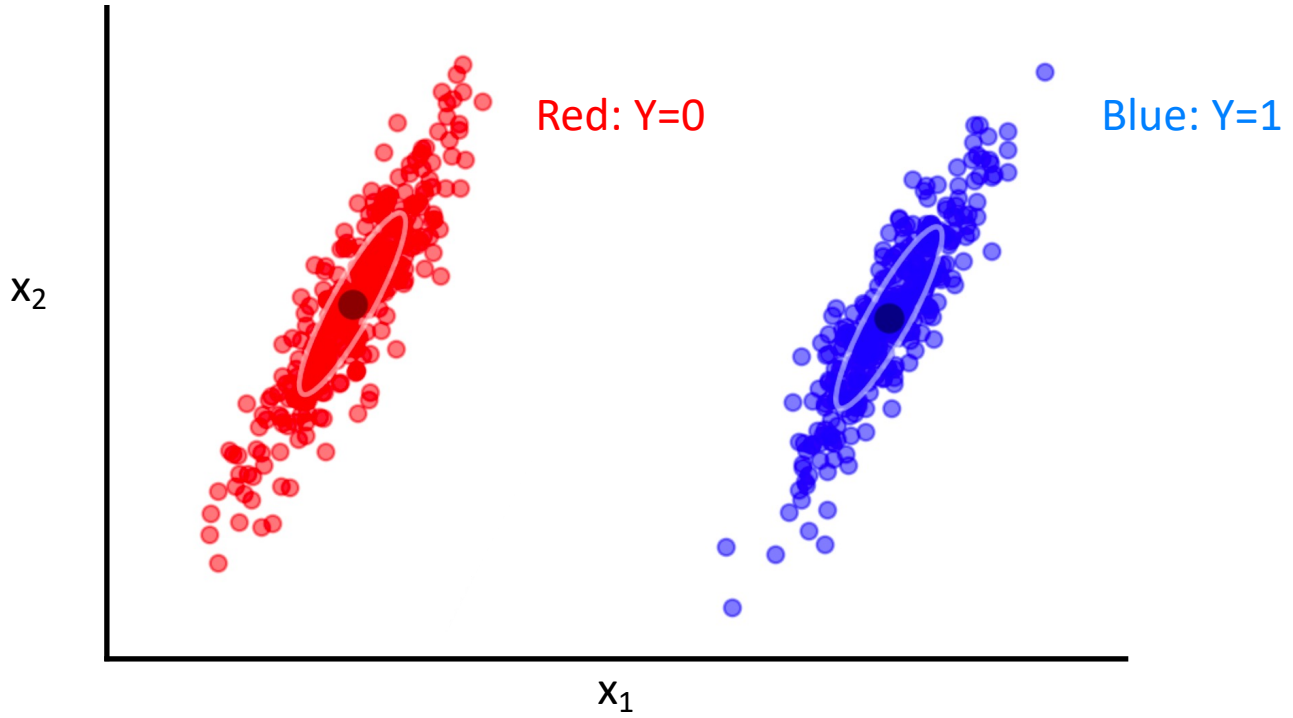
- Goal: Separate data from two classes / populations



# Linear Discriminant Analysis



- Goal: Separate data from two classes / populations
- Data from joint distribution $(\boldsymbol{x}, y) \sim p(\boldsymbol{X}, Y)$
  - Features: $\boldsymbol{x} \in \mathbb{R}^m$
  - Labels: $y \in \{0,1\}$



# Linear Discriminant Analysis



- Goal: Separate data from two classes / populations
- Data from joint distribution $(\boldsymbol{x}, y) \sim p(\boldsymbol{X}, Y)$
  - Features: $\boldsymbol{x} \in \mathbb{R}^m$
  - Labels: $y \in \{0,1\}$
- Breakdown the joint distribution:

$$p(x, y) = p(x|y)p(y)$$

Likelihood:
Distribution of features
for a given class

Prior:
Probability of each class

# Linear Discriminant Analysis



- Goal: Separate data from two classes / populations
- Data from joint distribution $(\boldsymbol{x}, y) \sim p(\boldsymbol{X}, Y)$
  - Features: $\boldsymbol{x} \in \mathbb{R}^m$
  - Labels: $y \in \{0,1\}$
- Breakdown the joint distribution:

$$p(x, y) = p(x|y)p(y)$$

- Assume likelihoods are Gaussian

$$p(x|y) = \frac{1}{\sqrt{(2\pi)^m |\Sigma|}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_y)^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}_y)\right)$$

# Predicting the Class



- Separating classes → Predict the class of a point **x**

$p(y = 1|\mathbf{x})$

- Want to build a classifier to predict the label y given and input **x**

# Predicting the Class



- Separating classes → Predict the class of a point **x**

$$p(y=1|\mathbf{x}) = \frac{p(\mathbf{x}|y=1)p(y=1)}{p(\mathbf{x})}$$

Bayes Rule

# Predicting the Class



- Separating classes → Predict the class of a point **x**

$$p(y=1|\mathbf{x}) = \frac{p(\mathbf{x}|y=1)p(y=1)}{p(\mathbf{x})}$$

Bayes Rule

$$= \frac{p(\mathbf{x}|y=1)p(y=1)}{p(\mathbf{x}|y=0)p(y=0) + p(\mathbf{x}|y=1)p(y=1)}$$

Marginal definition

# Predicting the Class



- Separating classes → Predict the class of a point **x**

$$p(y=1|\mathbf{x}) = \frac{p(\mathbf{x}|y=1)p(y=1)}{p(\mathbf{x})}$$

Bayes Rule

$$= \frac{p(\mathbf{x}|y=1)p(y=1)}{p(\mathbf{x}|y=0)p(y=0) + p(\mathbf{x}|y=1)p(y=1)}$$

Marginal definition

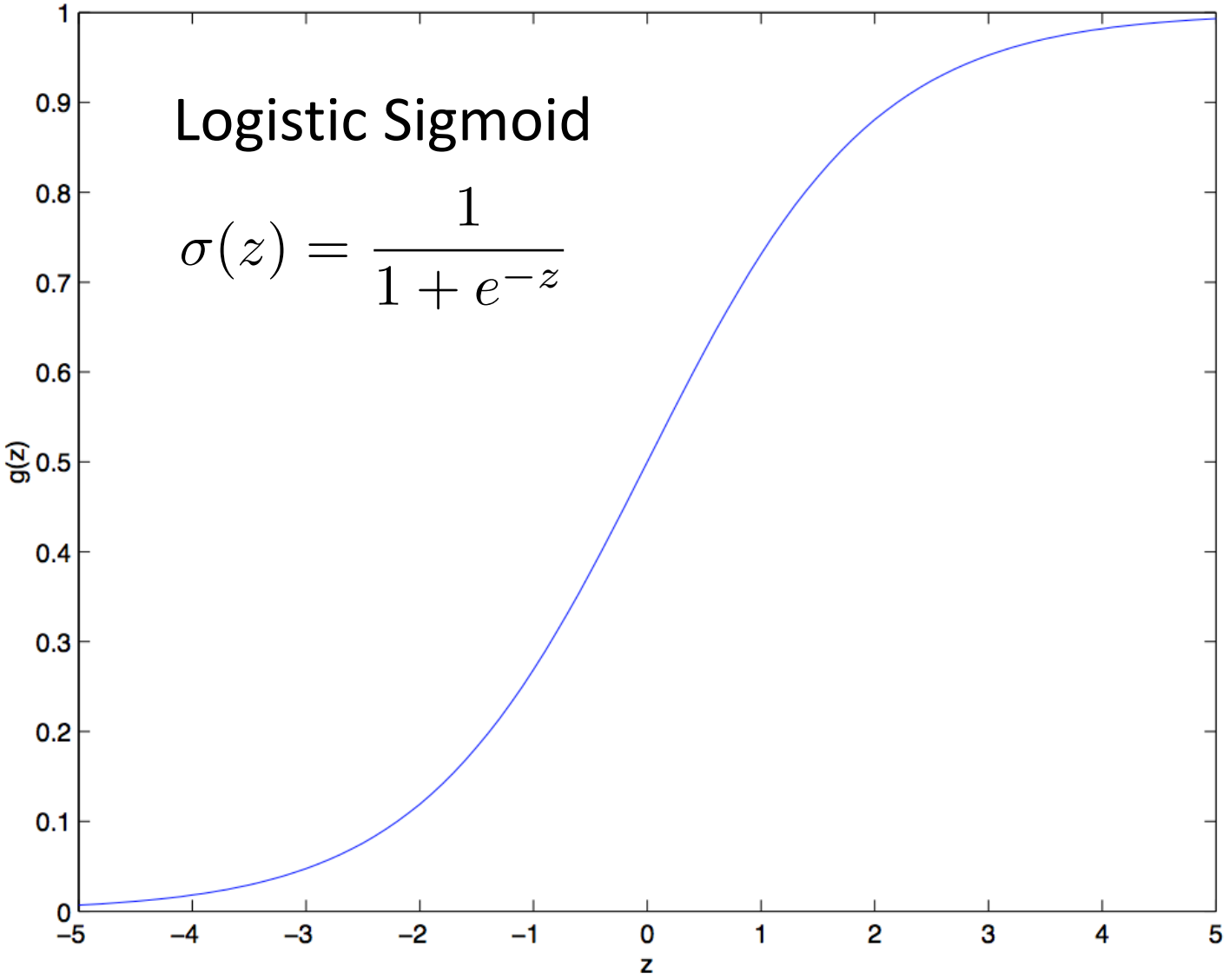
$$= \frac{1}{1 + \frac{p(\mathbf{x}|y=0)p(y=0)}{p(\mathbf{x}|y=1)p(y=1)}}$$

$$= \frac{1}{1 + \exp\left(\log \frac{p(\mathbf{x}|y=0)p(y=0)}{p(\mathbf{x}|y=1)p(y=1)}\right)}$$

Why?

# Logistic Sigmoid Function





# Predicting Classes with Gaussian Likelihoods



$$p(y=1|\mathbf{x}) = \sigma\Big(\log \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)} + \log \frac{p(y=1)}{p(y=0)}\Big)$$

Log-likelihood ratio

Constant w.r.t. **x**

# Predicting Classes with Gaussian Likelihoods



$$p(y=1|\mathbf{x}) = \sigma\Big( \log \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)} + \log \frac{p(y=1)}{p(y=0)} \Big)$$

- For our Gaussian data:

$$= \sigma\Big( \log p(\mathbf{x}|y=1) - \log p(\mathbf{x}|y=0) + const. \Big)$$

$$= \sigma\Big( -\frac{1}{2}(\mathbf{x}-\mu_1)^T \Sigma^{-1}(\mathbf{x}-\mu_1) + \frac{1}{2}(\mathbf{x}-\mu_0)^T \Sigma^{-1}(\mathbf{x}-\mu_0) + const. \Big)$$

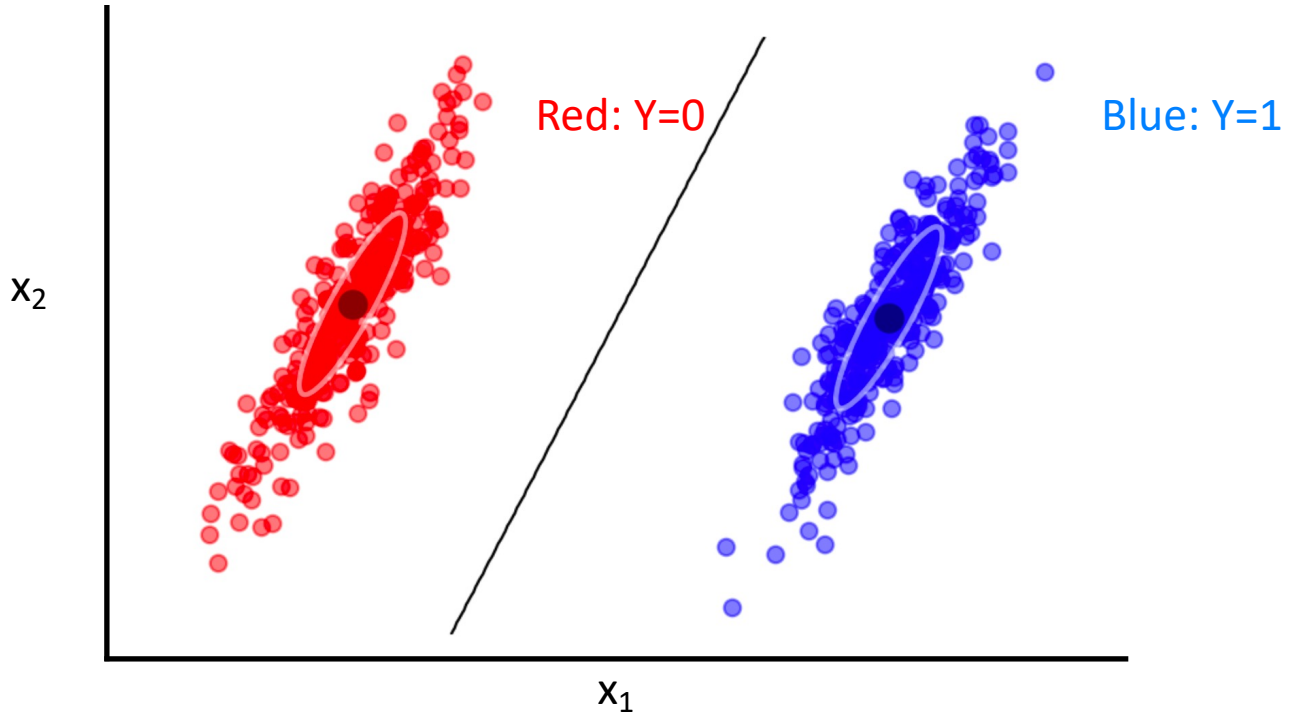
$$= \sigma\Big( \mathbf{w}^T \mathbf{x} + b \Big)$$

Collect terms

# What did we learn?



- For this data, the log-likelihood ratio is linear!
  - Line defines boundary to separate the classes
  - Sigmoid turns distance from boundary to probability



# Logistic Regression



- What if we ignore Gaussian assumption on data?

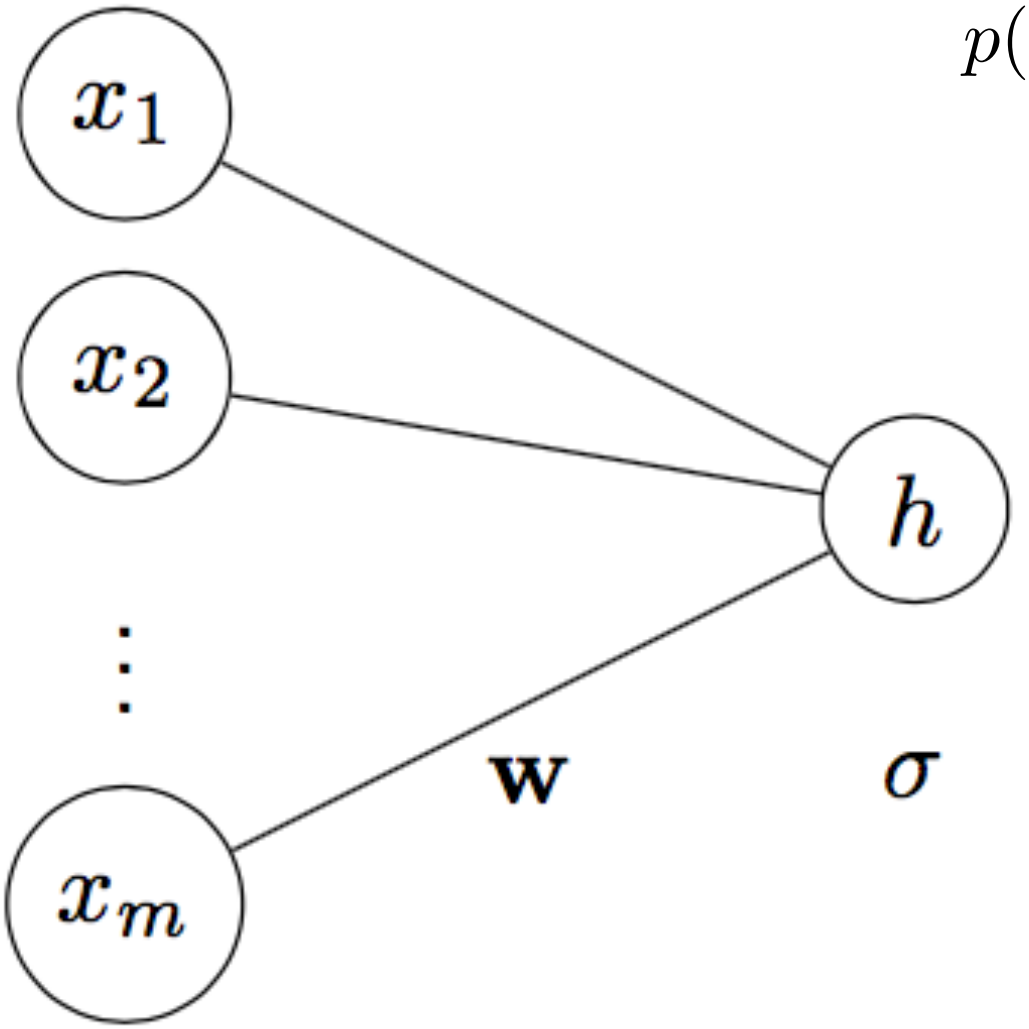
$$\text{Model:} \quad p(y=1|\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right) \equiv h(\mathbf{x};\mathbf{w})$$

- Farther from boundary $\boldsymbol{w}^T\boldsymbol{x} + b = 0$, more certain about class

- Sigmoid converts distance to class probability

# Logistic Regression






$$p(y = 1|\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$$
$$= \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x} - \mathbf{b}}}$$

This unit is the main building block of Neural Networks!

# Logistic Regression



- What if we ignore Gaussian assumption on data?

Model: $$p(y=1|\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x}+b\right) \equiv h(\mathbf{x};\mathbf{w})$$

- With $p_i \equiv p(y_i = y|\boldsymbol{x}_i)$

$$P(y_i = y|x_i) = \text{Bernoulli}(p_i) = (p_i)^{y_i}(1-p_i)^{1-y_i} = \begin{cases} p_i & \text{if } y_i=1 \\ 1\text{-}p_i & \text{if } y_i=0 \end{cases}$$

- **Goal**:
  - Given i.i.d. dataset of pairs $(\boldsymbol{x}_i, y_i)$ find $\mathbf{w}$ and b that maximize likelihood of data

# Logistic Regression



- Negative log-likelihood

$$-\ln \mathcal{L} = -\ln \prod_i (p_i)^{y_i} (1-p_i)^{1-y_i}$$

# Logistic Regression

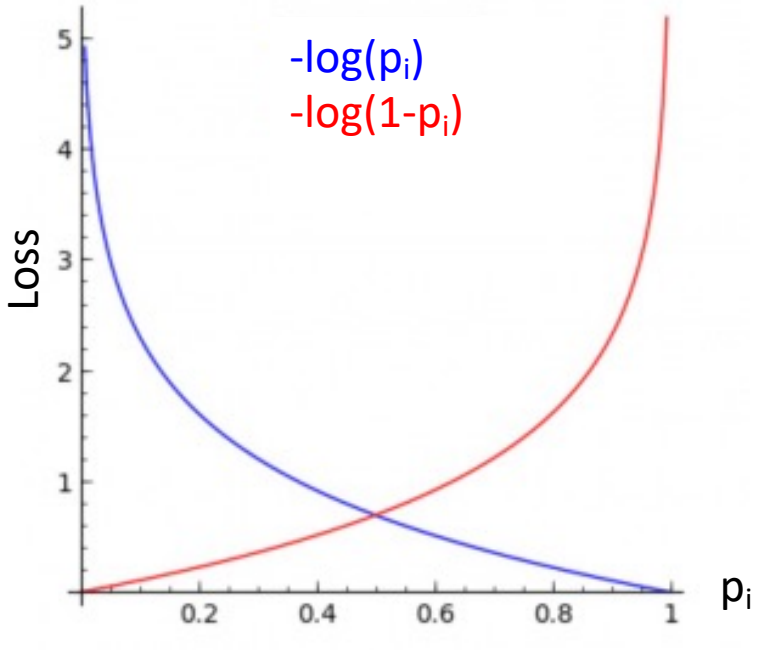


- Negative log-likelihood

$$-\ln \mathcal{L} = -\ln \prod_i (p_i)^{y_i} (1-p_i)^{1-y_i}$$

$$= -\sum_i y_i \ln(p_i) + (1-y_i) \ln(1-p_i)$$

binary cross entropy loss function!



# Logistic Regression



- Negative log-likelihood

$$-\ln \mathcal{L} = -\ln \prod_i (p_i)^{y_i} (1-p_i)^{1-y_i}$$

$$= -\sum_i y_i \ln(p_i) + (1-y_i) \ln(1-p_i)$$

$$= \sum_i y_i \ln(1+e^{-\mathbf{w}^T \mathbf{x}}) + (1-y_i) \ln(1+e^{\mathbf{w}^T \mathbf{x}})$$

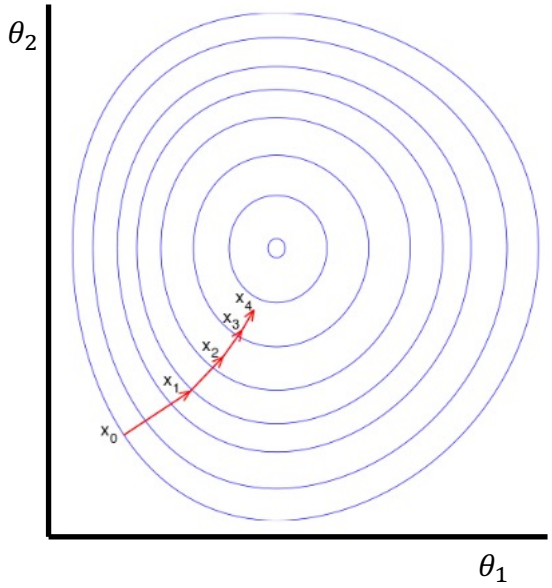
binary cross entropy loss function!

- No closed form solution to $w^* = \arg\min_w -\ln \mathcal{L}(w)$
- How to solve for $\mathbf{w}$?

# Gradient Descent



- Minimize loss by repeated gradient steps
  - Compute gradient w.r.t. current parameters: $\nabla_{\theta_i}\mathcal{L}(\theta_i)$
  - Update parameters: $\theta_{i+1} \leftarrow \theta_i - \eta\nabla_{\theta_i}\mathcal{L}(\theta_i)$
  - $\eta$ is the *learning rate*, controls how big of a step to take



# Stochastic Gradient Descent

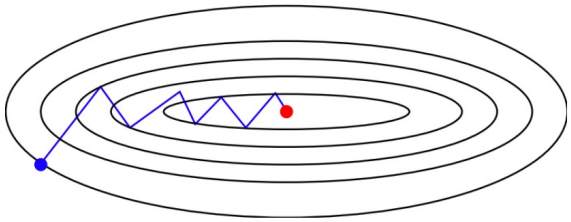


- Loss is composed of a sum over samples:

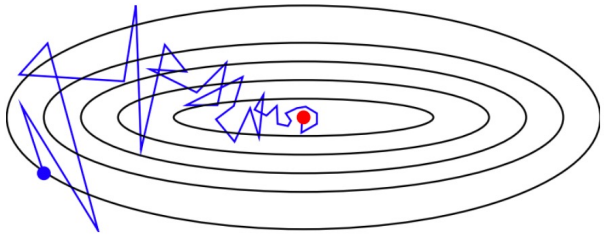
$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \mathcal{L}\big(y_i, h(x_i;\theta)\big)$$

  - Computing gradient grows linearly with N!

- **(Mini-Batch) Stochastic Gradient Descent**
  - Compute gradient update using 1 random sample (small size batch)
  - Gradient is unbiased → on average it moves in correct direction
  - Tends to be much faster the full gradient descent



*Batch gradient descent*



*Stochastic gradient descent*

# Stochastic Gradient Descent



- Loss is composed of a sum over samples:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \mathcal{L}\big(y_i, h(x_i;\theta)\big)$$

  - Computing gradient grows linearly with N!

- **(Mini-Batch) Stochastic Gradient Descent**
  - Compute gradient update using 1 random sample (small size batch)
  - Gradient is unbiased → on average it moves in correct direction
  - Tends to be much faster the full gradient descent

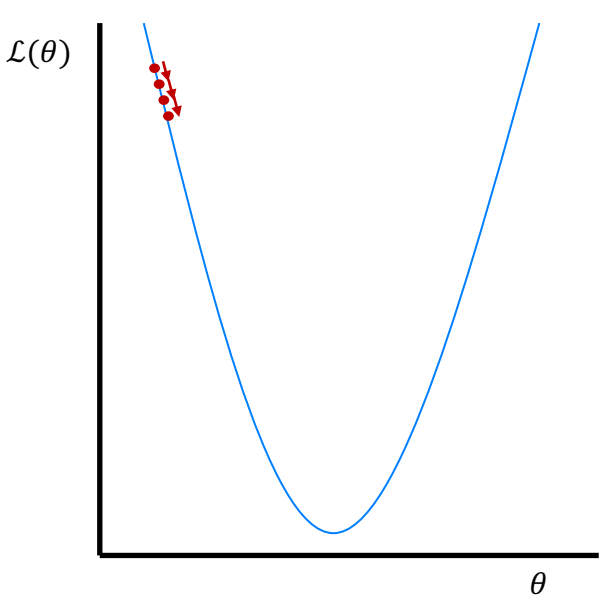
- Several updates to SGD, like momentum, ADAM, RMSprop to
  - Help to speed up optimization in flat regions of loss
  - Have adaptive learning rate
  - Learning rate adapted for each parameter
  - …

# Step Sizes

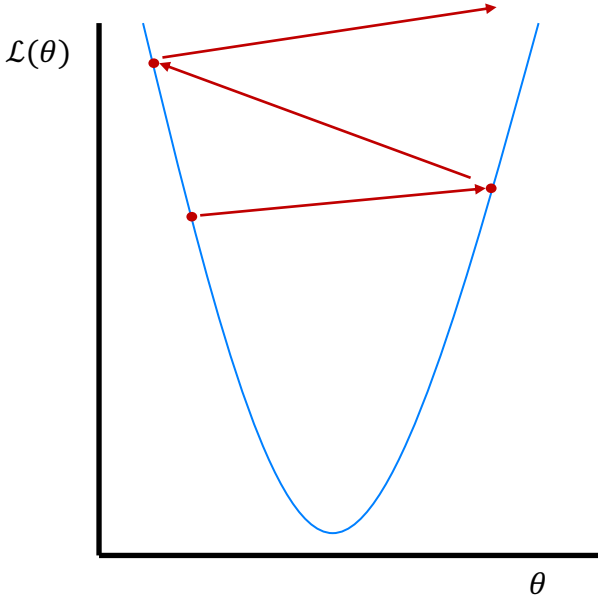


- Too small a learning rate, convergence very slow
- Too large a learning rate, algorithm diverges

Small Learning rate

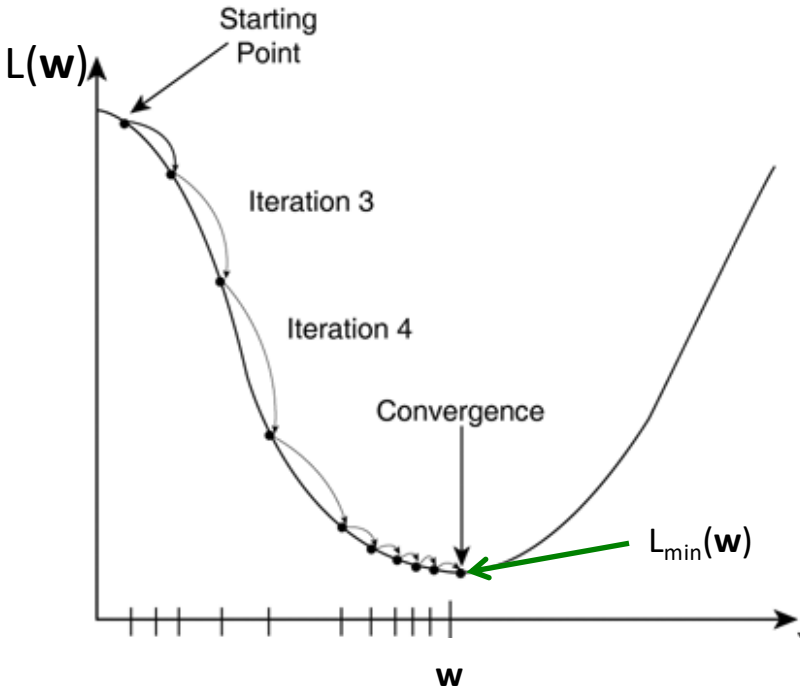


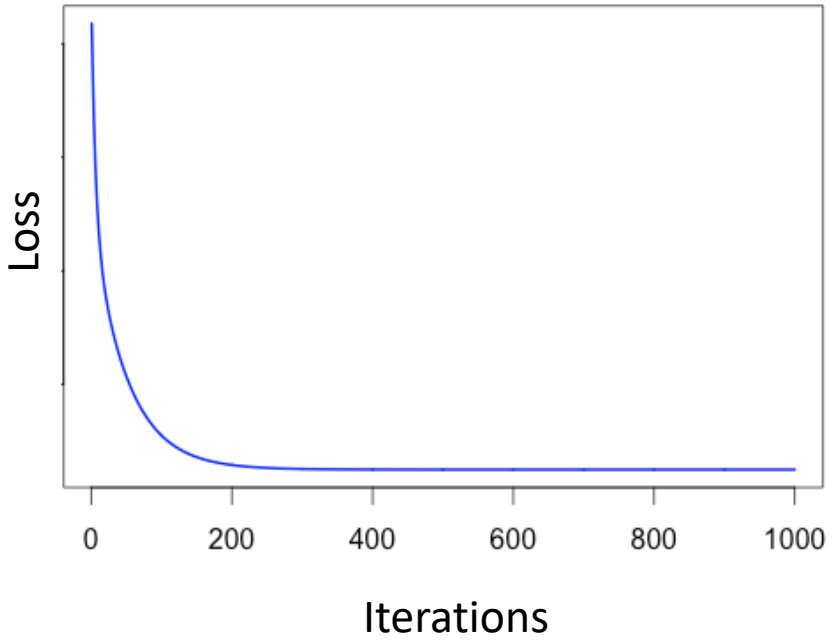

Large Learning rate



# Gradient Descent



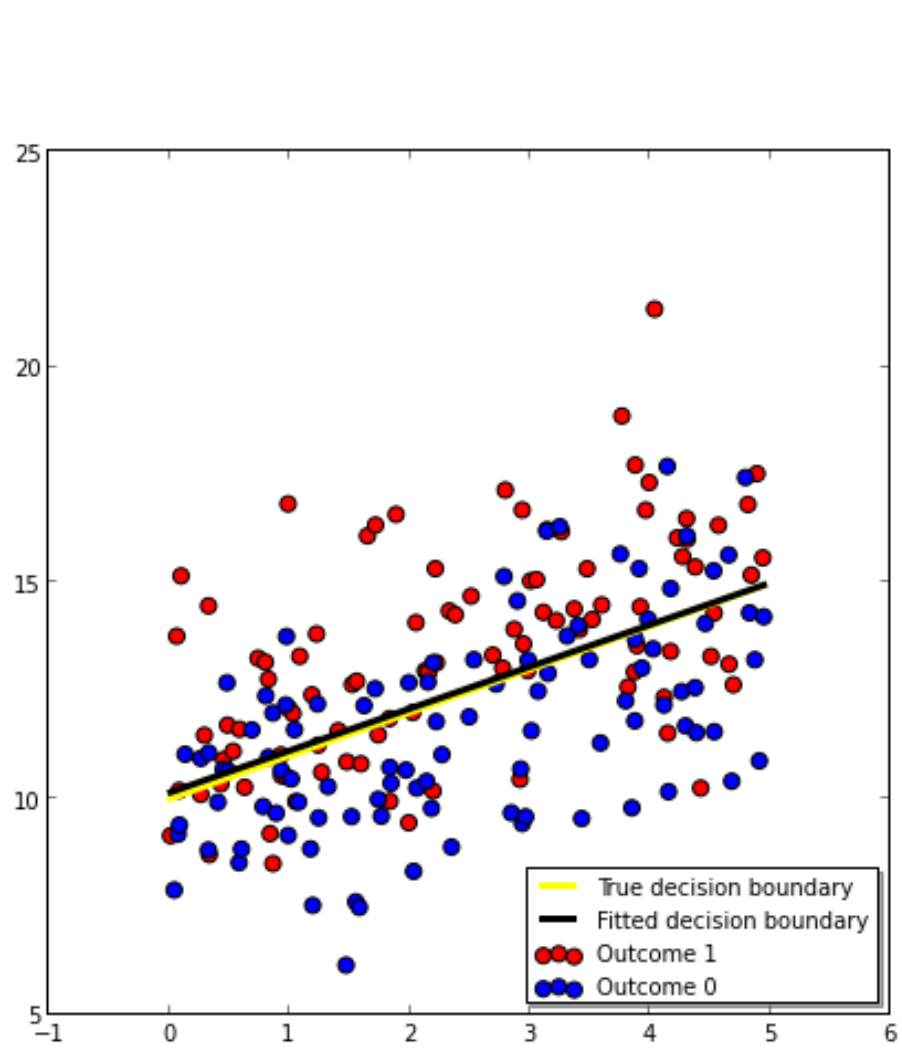



- Logistic Regression Loss is convex
  - Single global minimum
- Iterations lower loss and move toward minimum

# Logistic Regression Example





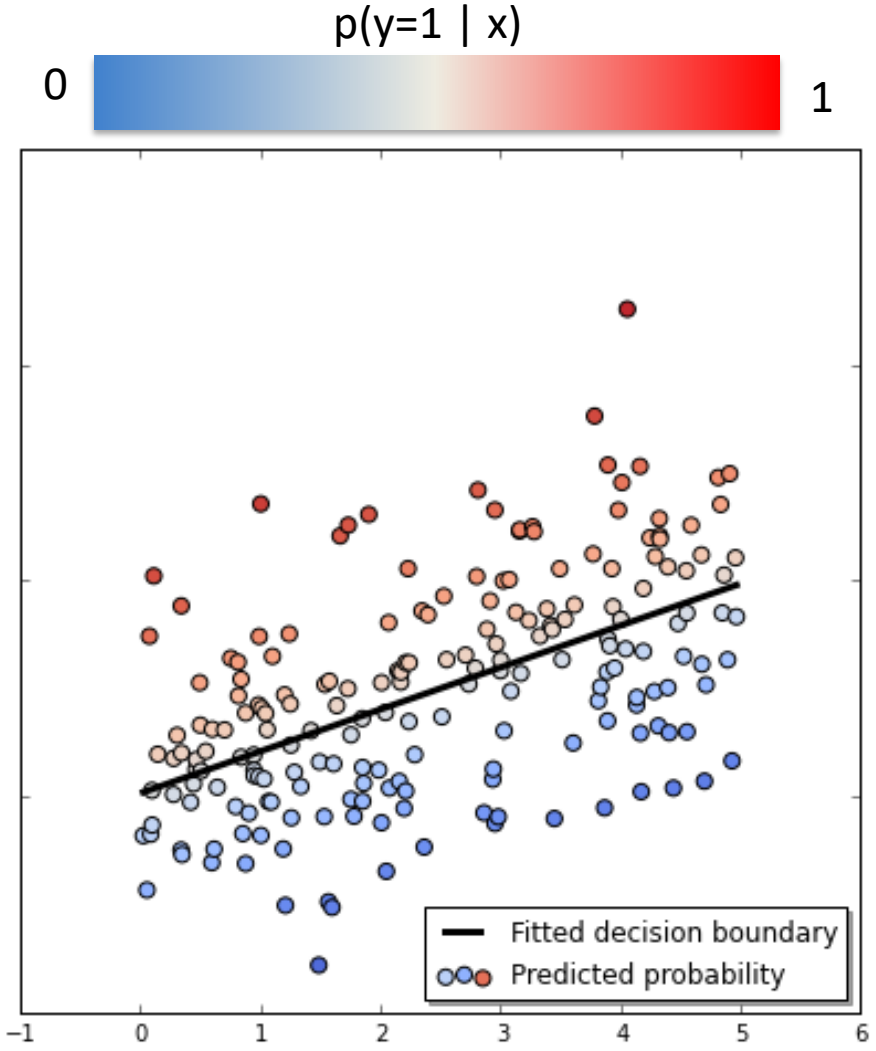
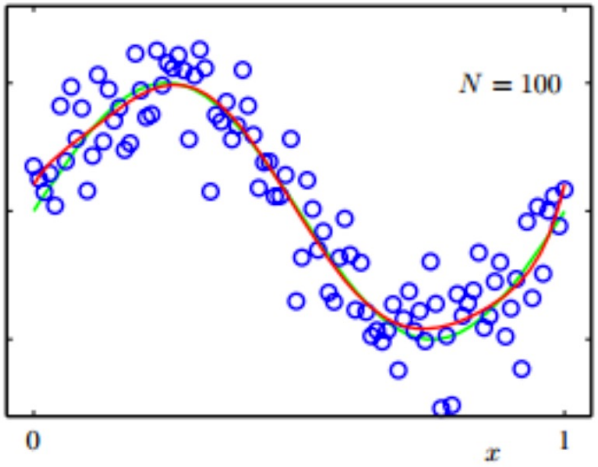




Image source

# Basis Functions

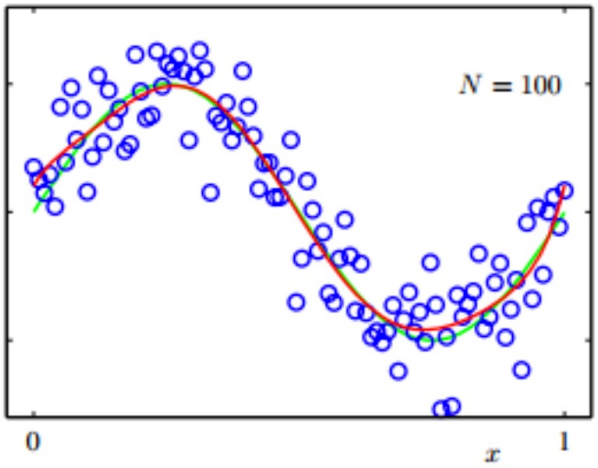




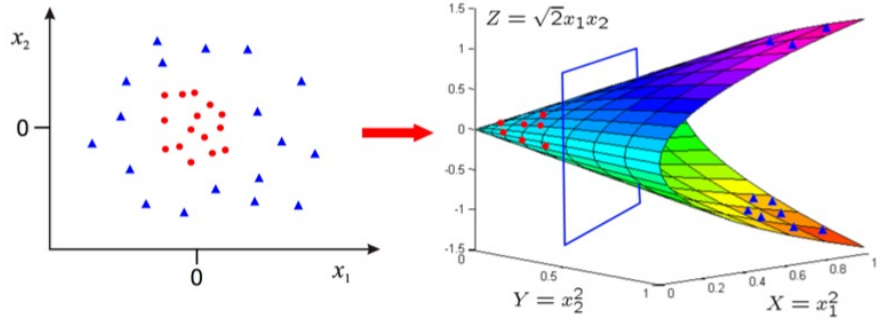

- What if non-linear relationship between **y** and **x**?

# Basis Functions






$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



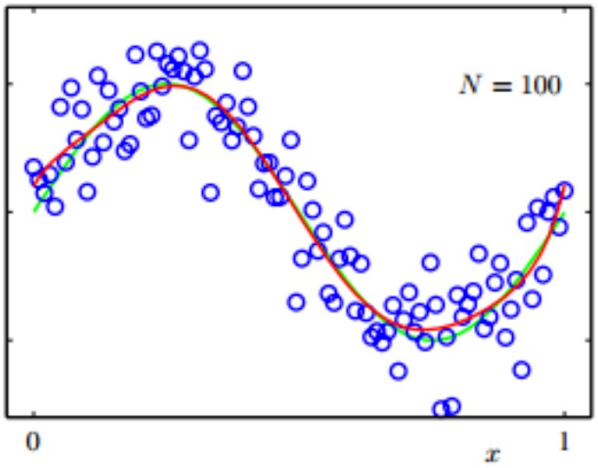

- What if non-linear relationship between **y** and **x**?
- Can choose basis functions $\phi(x)$ to form new features

$$h(x; w) = \sigma\left(w^T \phi(x)\right)$$

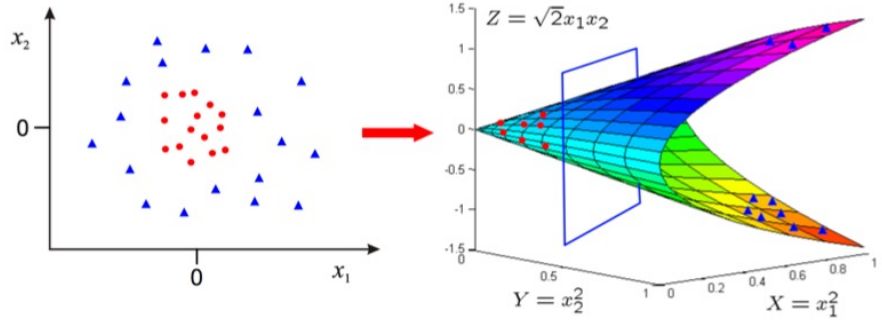
  - Polynomial basis $\phi(x) \sim \{1, x, x^2, x^3, \ldots\}$, Gaussian basis, …
  - Logistic regression on new features $\phi(x)$

# Basis Functions






$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



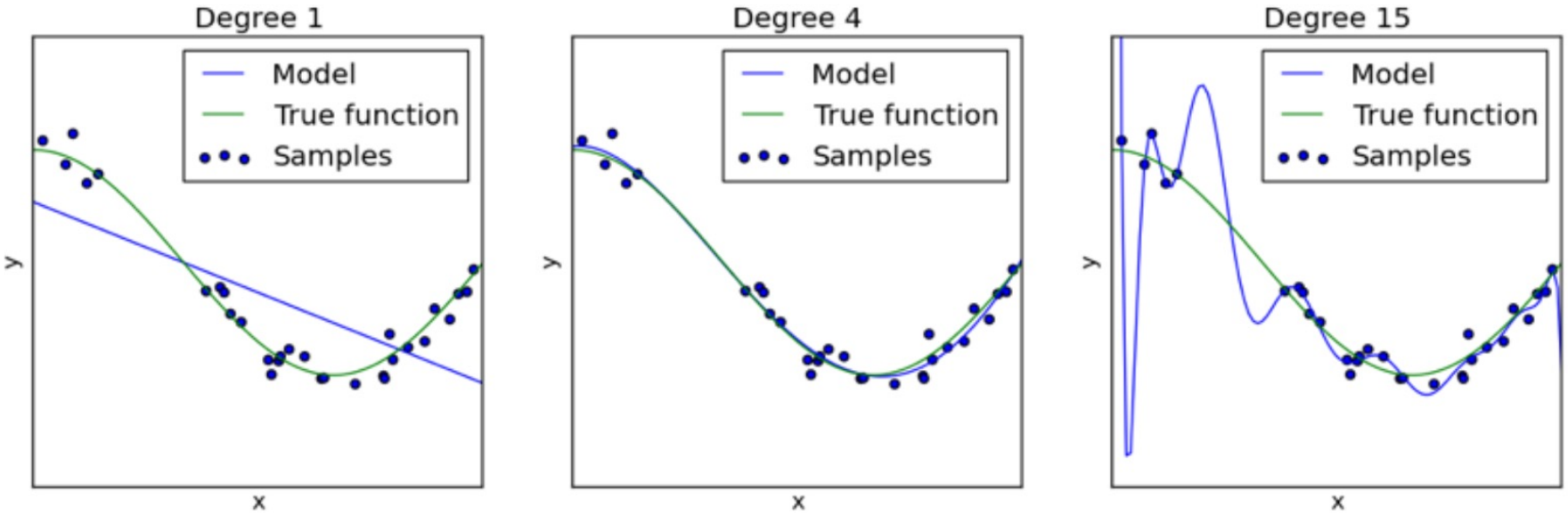

- What if non-linear relationship between **y** and **x**?
- Can choose basis functions ϕ(x) to form new features

$$h(x; w) = \sigma\left(w^T \phi(x)\right)$$

  - Polynomial basis ϕ(x) ~ {1, x, $x^2$, $x^3$, …}, Gaussian basis, …
  - Logistic regression on new features ϕ(x)
- What basis functions to choose? *Overfit* with too much flexibility?

# What is Overfitting






Underfitting

Overfitting

http://scikit-learn.org/

- Models allow us to **generalize** from data
- Different models generalize in different ways

# Bias Variance Tradeoff



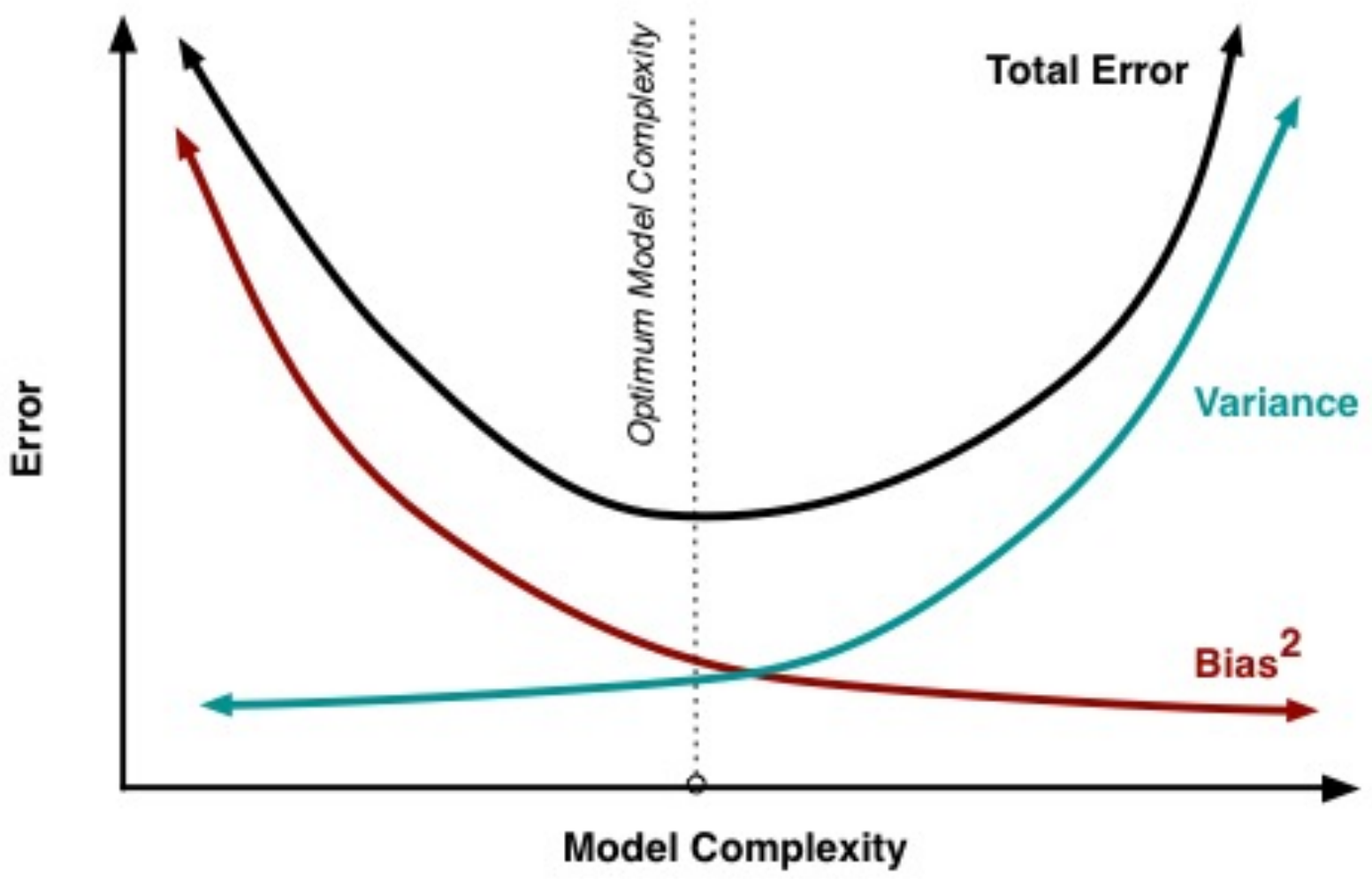
- generalization error = systematic error + sensitivity of prediction
  (bias) (variance)

- Simple models under-fit: will deviate from data (high bias) but will not be influenced by peculiarities of data (low variance).

- Complex models over-fit: will not deviate systematically from data (low bias) but will be very sensitive to data (high variance).
  - **As dataset size grows, can reduce variance!**
  - **Can use more complex model**

# Bias Variance Tradeoff



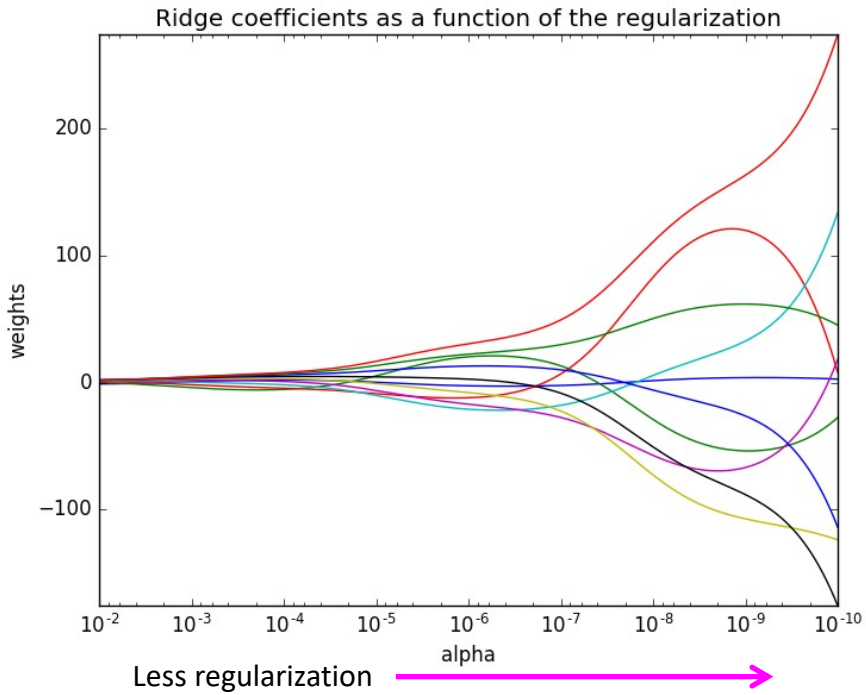


# Regularization – Control Complexity



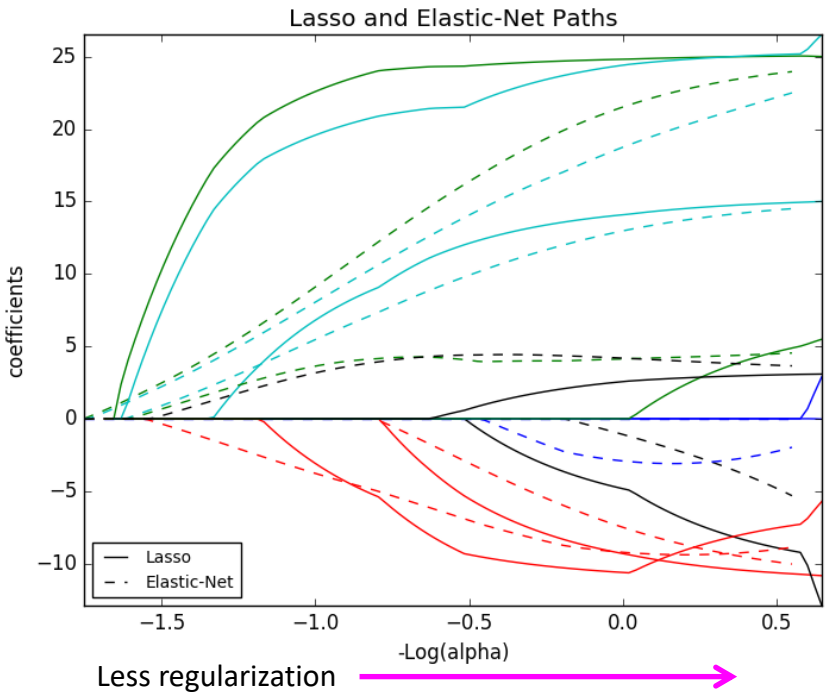
$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^2 + \alpha\Omega(\mathbf{w})$$

$$L2: \quad \Omega(\mathbf{w}) = ||\mathbf{w}||^2$$




Less regularization →

$$L1: \quad \Omega(\mathbf{w}) = ||\mathbf{w}||$$




Less regularization →

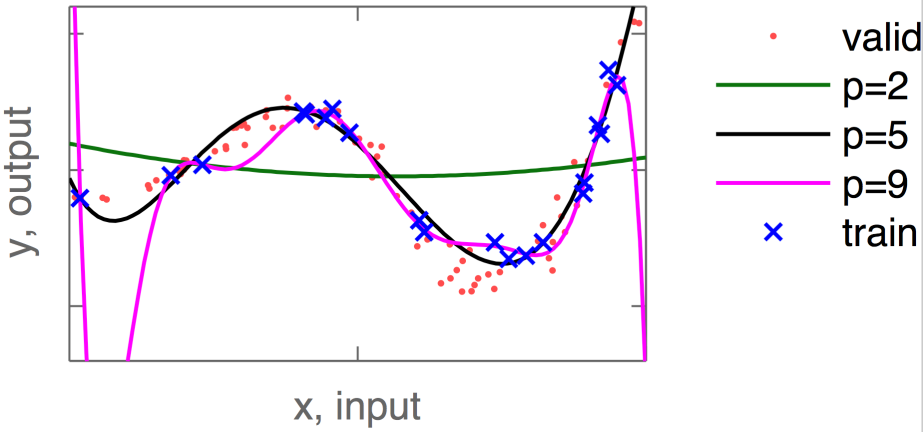
- L2 keeps weights small, L1 keeps weights sparse!
- But how to choose hyperparameter α?

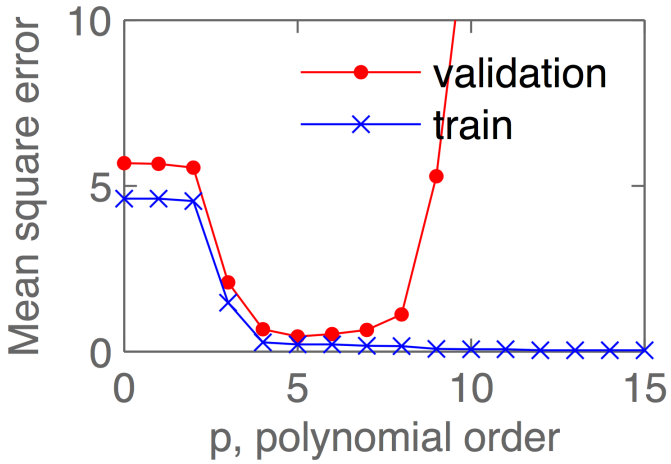
http://scikit-learn.org/

# How to Measure Generalization Error?




- Split dataset into multiple parts
- **Training set**
  - Used to fit model parameters
- **Validation set**
  - Used to check performance on independent data and tune hyper parameters
- **Test set**
  - final evaluation of performance after all hyper-parameters fixed
  - Needed since we tune, or "peek", performance with validation set

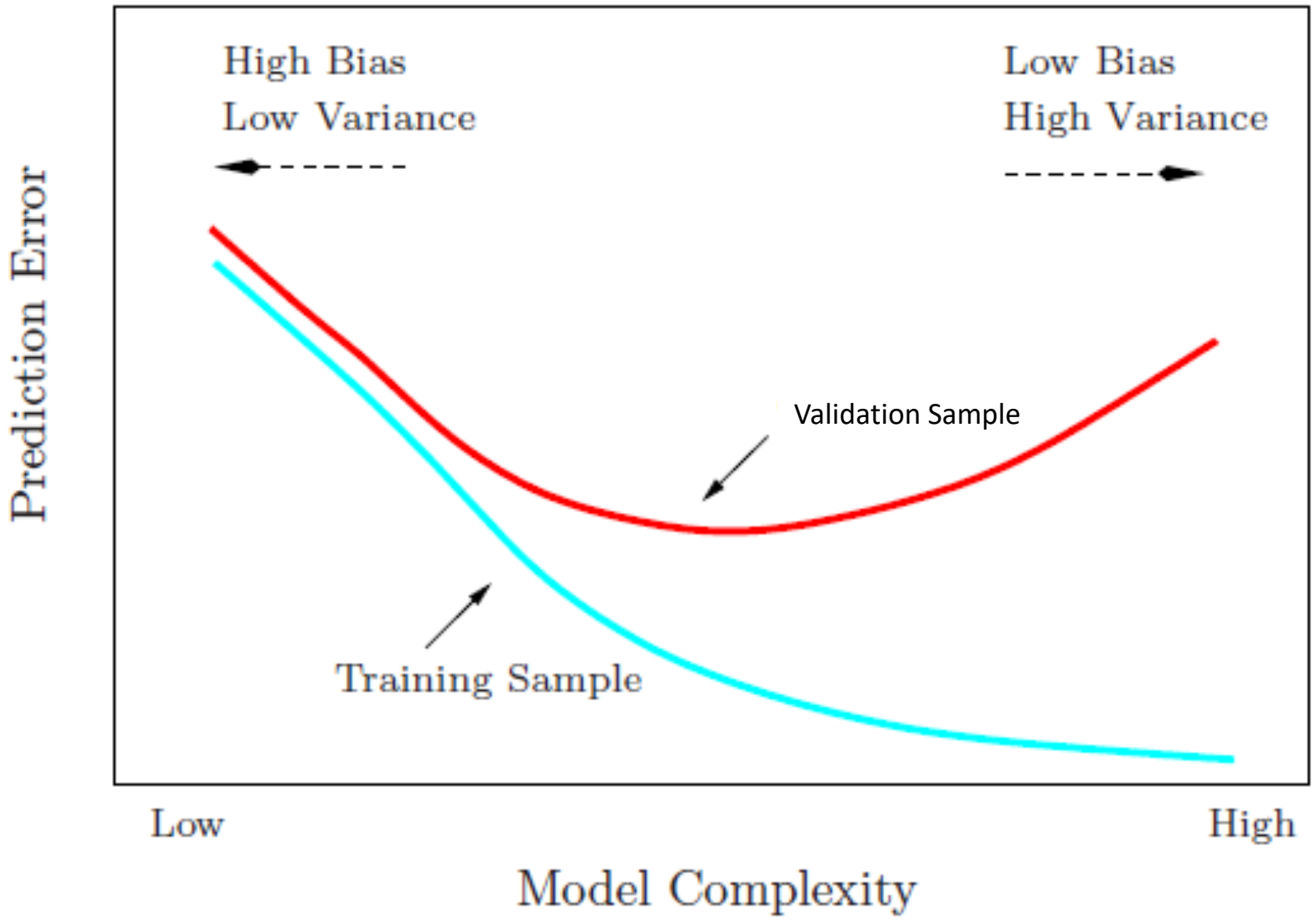



[Murray]

# How to Measure Generalization Error?





# Neural Networks

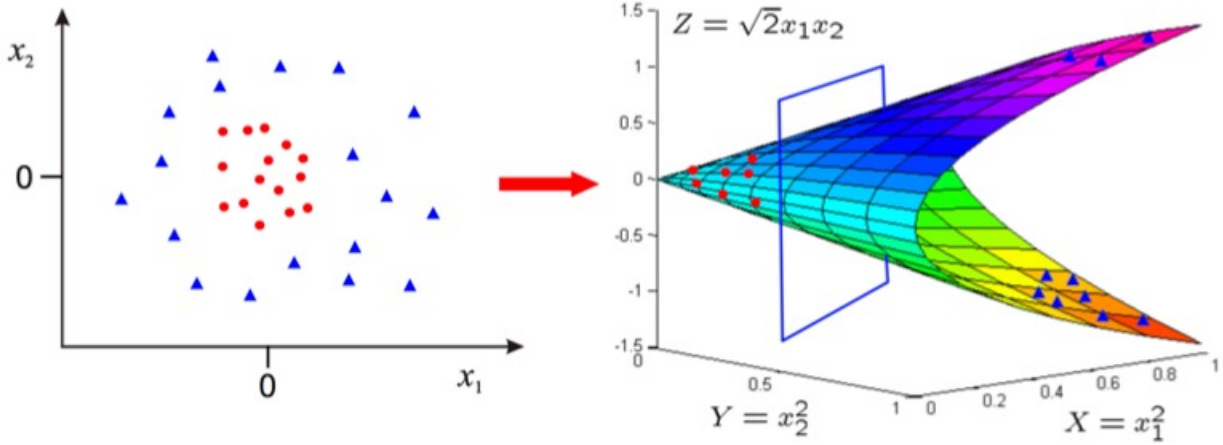
# Adding non-linearity



- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



# Adding non-linearity



- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?

# Adding non-linearity



- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?
- Learn the basis functions directly from data

$$\boldsymbol{\phi}(\boldsymbol{x};\, \boldsymbol{u}) \qquad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where **u** is a set of parameters for the transformation

# Adding non-linearity



- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?
- Learn the basis functions directly from data

$$\boldsymbol{\phi}(\boldsymbol{x};\, \boldsymbol{u}) \qquad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where **u** is a set of parameters for the transformation

  - Combines basis selection & learning→*Representation Learning*
  - Several different approaches, focus here on neural networks
  - Complicates the optimization

# Neural Networks



- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\boldsymbol{x};\, \boldsymbol{u}) = \sigma(\boldsymbol{u}_j{}^T \boldsymbol{x})$$

# Neural Networks



- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\boldsymbol{x};\, \boldsymbol{u}) = \sigma(\boldsymbol{u}_j{}^T \boldsymbol{x})$$

- Put all $\boldsymbol{u}_j \in \mathbb{R}^{1\times m}$ vectors into matrix $\boldsymbol{U}$

$$\phi(\boldsymbol{x};\, \boldsymbol{U}) = \sigma(\boldsymbol{U}\boldsymbol{x}) = \begin{bmatrix} \sigma(u_1^T x) \\ \sigma(u_2^T x) \\ \vdots \\ \sigma(u_d^T x) \end{bmatrix} \in \mathbb{R}^d$$

  - σ is a point-wise non-linearity acting on each vector element

# Neural Networks



- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\boldsymbol{x};\, \boldsymbol{u}) = \sigma(\boldsymbol{u}_j{}^T \boldsymbol{x})$$

- Put all $\boldsymbol{u}_j \in \mathbb{R}^{1\times m}$ vectors into matrix $\boldsymbol{U}$

$$\phi(\boldsymbol{x};\, \boldsymbol{U}) = \sigma(\boldsymbol{U}\boldsymbol{x}) = \begin{bmatrix} \sigma(u_1^T x) \\ \sigma(u_2^T x) \\ \vdots \\ \sigma(u_d^T x) \end{bmatrix} \in \mathbb{R}^d$$

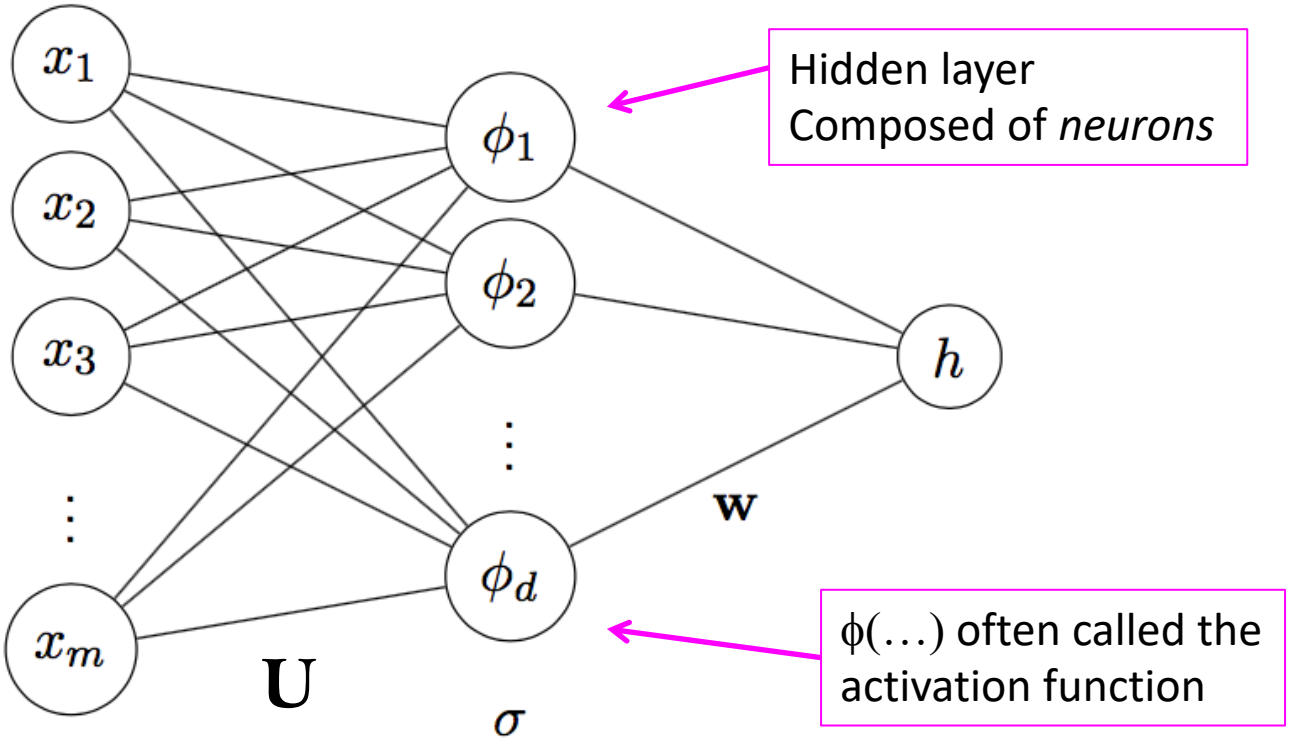
  - σ is a point-wise non-linearity acting on each vector element

- Full model becomes

$$h(\boldsymbol{x};\, \boldsymbol{w}, \boldsymbol{U}) = w^T \phi(\boldsymbol{x};\, \boldsymbol{U})$$

# Feed Forward Neural Network



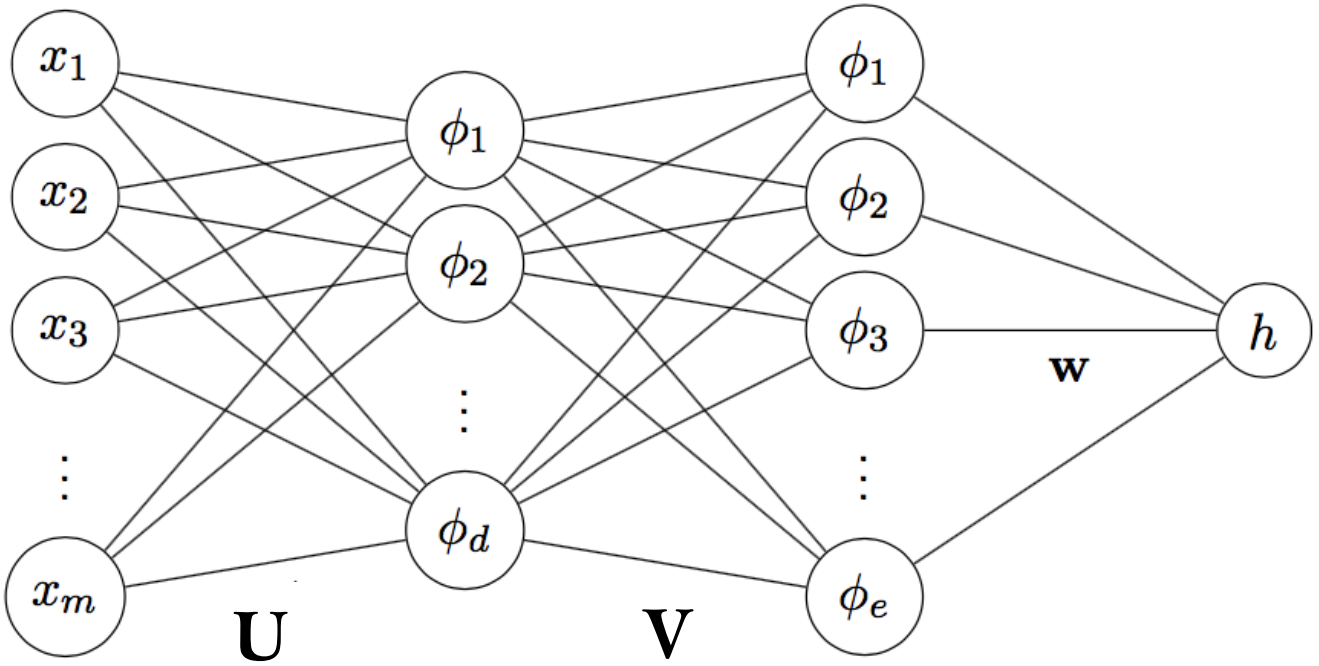



$$\phi(\mathbf{x}) = \sigma(\mathbf{U}\mathbf{x})$$

$$h(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

# Multi-layer Neural Network






- Multilayer NN
  - Each layer adapts basis functions based on previous layer

# Neural Network Optimization Problem



- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

# Neural Network Optimization Problem



- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

- **Regression**: Square error loss function

$$L(\mathbf{w}, \mathbf{U}) = \frac{1}{2} \sum_i (y_i - h(\mathbf{x}_i))^2$$

# Neural Network Optimization Problem



- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

- **Regression**: Square error loss function

$$L(\mathbf{w}, \mathbf{U}) = \frac{1}{2} \sum_i (y_i - h(\mathbf{x}_i))^2$$

- Minimize loss with respect to weights **w**, **U**

# Minimizing loss with gradient descent:



- Parameter update:

$$w \leftarrow w - \eta \frac{\partial L(w, U)}{\partial w}$$

$$U \leftarrow U - \eta \frac{\partial L(w, U)}{\partial U}$$

- How to compute gradients?

# Automatic Differentiation



- Exact derivatives for gradient-based optimization come from running **differentiable code** via **automatic differentiation**

$$f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$$

**automatic differentiation**

$$\nabla f(\mathbf{x}) = \left( \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

```
f(x) {…};
```

```
df(x) {…};
```

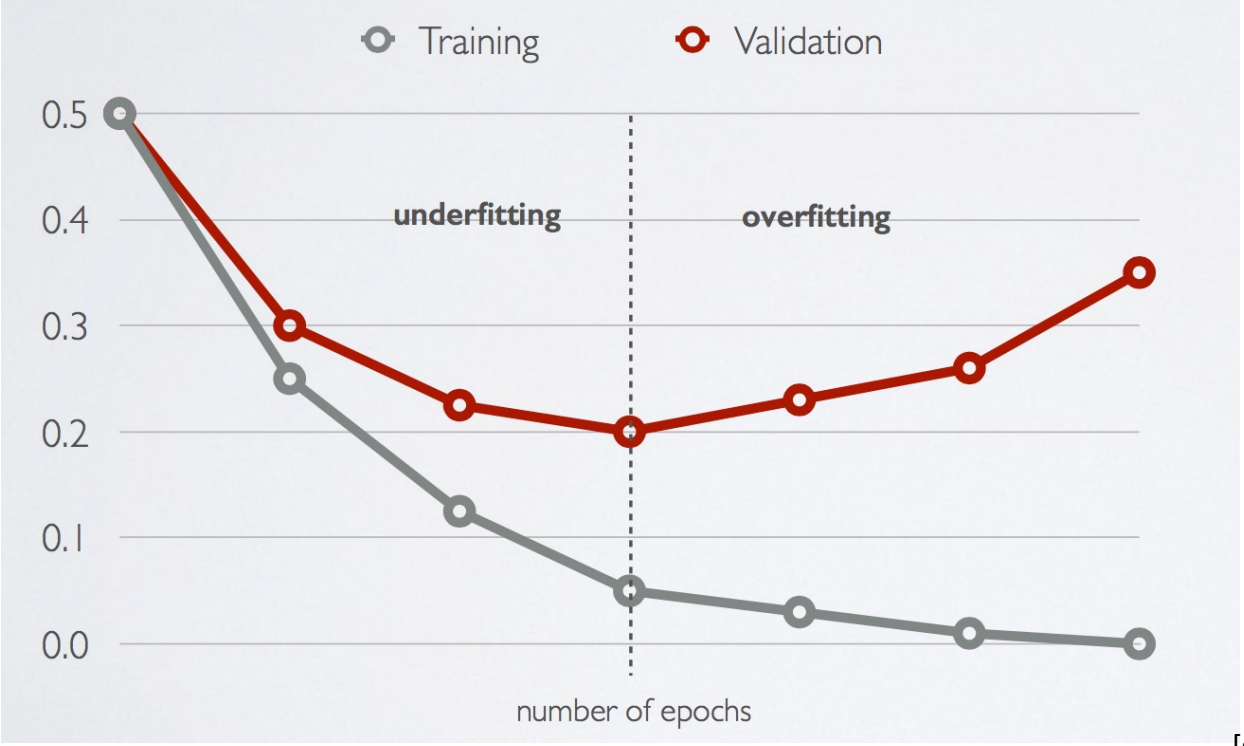
- Can compute derivatives not just of mathematical functions, but **derivatives of general purpose code** with control flow, loops, recursions, etc.

# Training



- Repeat gradient update of weights to reduce loss
  - Each iteration through dataset is called an epoch

- Use validation set to examine for overtraining, and determine when to stop training

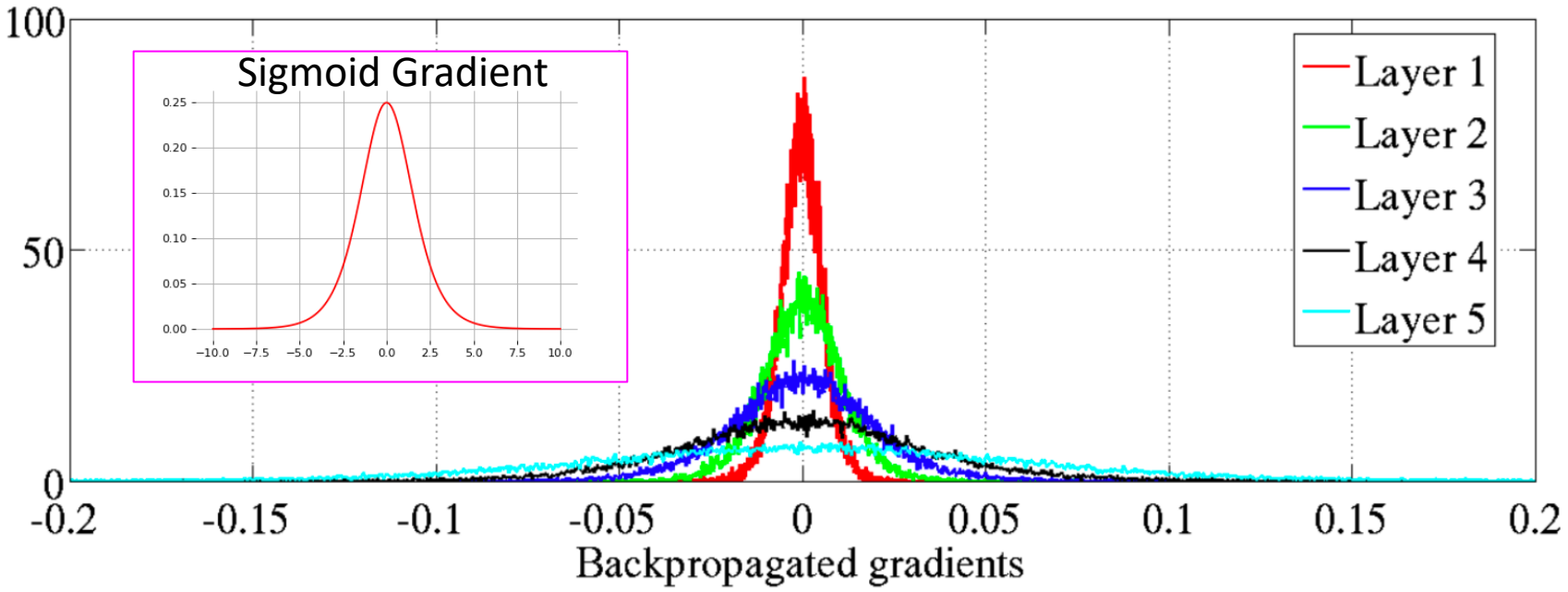



[graphic from H. Larochelle]

# Vanishing Gradients



- Major challenge in DL: Vanishing Gradients
- Small gradients slow down / block, stochastic gradient descent → Limits ability to learn!

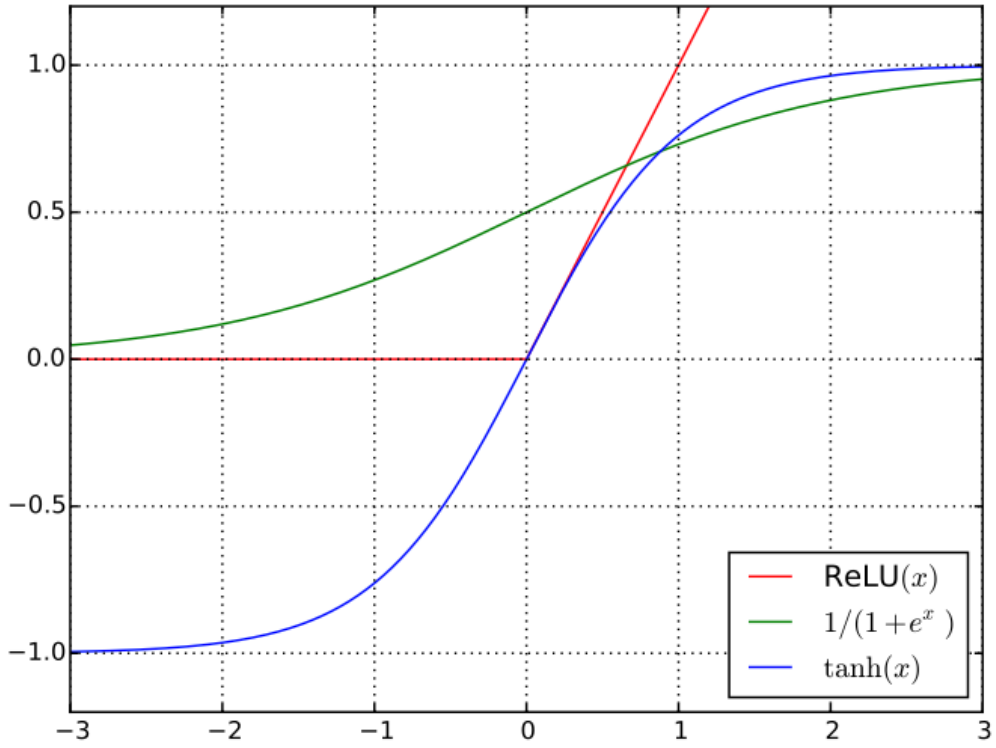



Backpropagated gradients normalized histograms (Glorot and Bengio, 2010). Gradients for layers far from the output vanish to zero.

Slide credit: G. Louppe

# Activation Functions






- **Vanishing gradient problem**
  - Derivative of sigmoid:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$

  - Nearly 0 when x is far from 0!
  - Can make gradient descent hard!

- **Rectified Linear Unit (ReLU)**
  - ReLU(x) = max{0, x}
  - Derivative is constant!

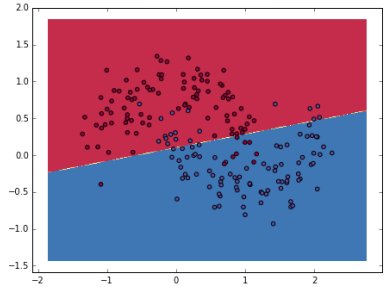
$$\frac{\partial \mathrm{Re}LU(x)}{\partial x} = \begin{cases} 1 & when\ x > 0 \\ 0 & otherwise \end{cases}$$

  - ReLU gradient doesn't vanish

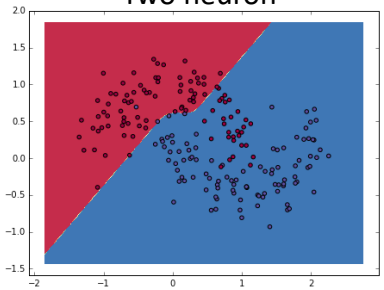
# Neural Network Decision Boundaries



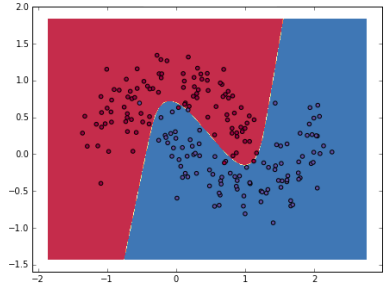
One neuron



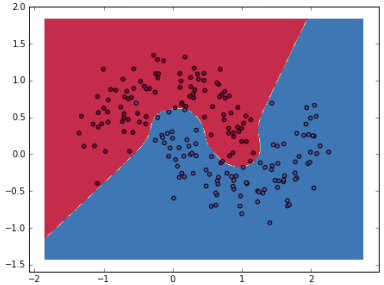
Two neuron



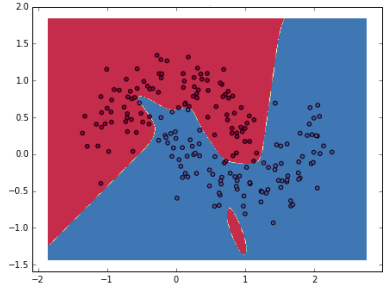
Three neurons



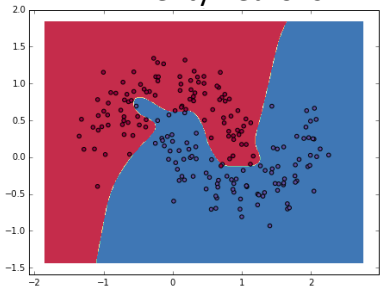
Four neurons



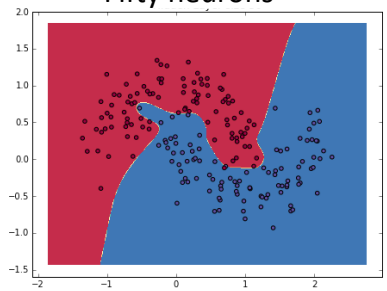
Five neurons



Twenty neurons



Fifty neurons



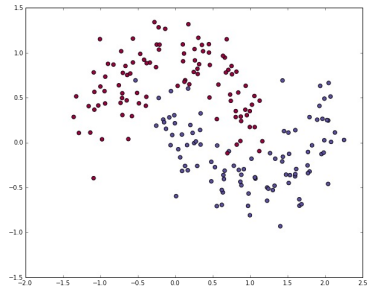
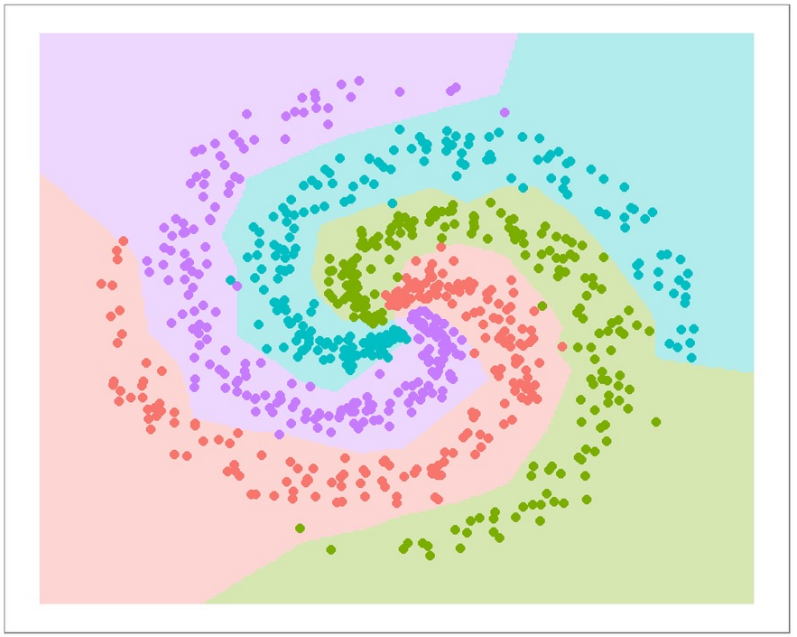


Image source

4-class classification
2-hidden layer NN
ReLU activations
L2 norm regularization

$x_2$



$x_1$

2-class classification
1-hidden layer NN
L2 norm regularization

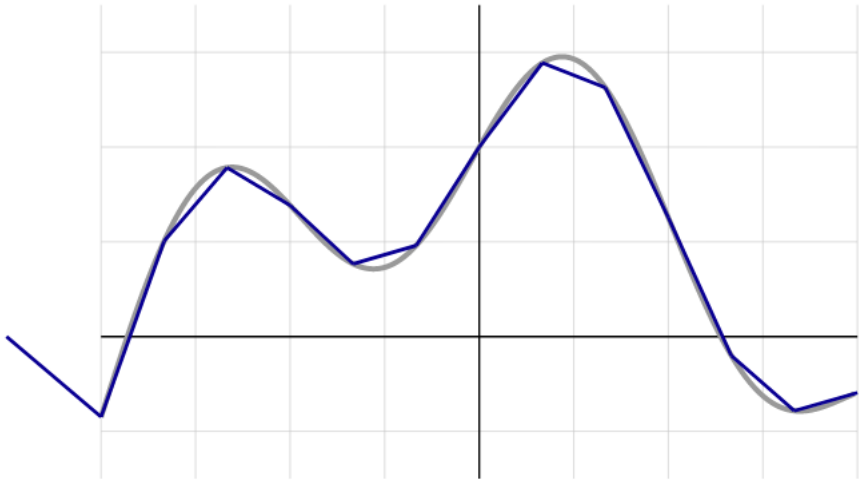
Image source

# Universal approximation theorem



- Feed-forward neural network with a single hidden layer containing a finite number of non-linear neurons (ReLU, Sigmoid, and others) can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \dots$$



Fleuret, Deep Learning Course

# Universal approximation theorem



- Feed-forward neural network with a single hidden layer containing a finite number of non-linear neurons (ReLU, Sigmoid, and others) can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

- NOTE!
  - A better approximation requires a larger hidden layer and this theorem says nothing about the relation between the two.
  - We can make training error as low as we want by using a larger hidden layer. Result states nothing about test error
  - Doesn't say how to find the parameters for this approximation

# Summary



- Machine learning uses mathematical and statistical models learned from data to characterize patterns and relations between inputs, and use this for inference / prediction

- ML comes in many forms, much of which has probabilistic and statistical foundations and interpretations (i.e. *Statistical Machine Learning*)

- ML provides a powerful toolkit to analyze data
  - Linear methods can help greatly in understanding data
  - Neural Networks allow us to learn nonlinear basis functions that help us solve our learning problem
  - Choosing a model for a given problem is difficult,
  - Keep in mind bias-variance tradeoff

# References




- http://scikit-learn.org/
- [Bishop] Pattern Recognition and Machine Learning, Bishop (2006)
- [ESL] Elements of Statistical Learning (2nd Ed.) Hastie, Tibshirani & Friedman 2009
- [Murray] Introduction to machine learning, Murray
  - http://videolectures.net/bootcamp2010_murray_iml/
- [Ravikumar] What is Machine Learning, Ravikumar and Stone
  - http://www.cs.utexas.edu/sites/default/files/legacy_files/research/documents/MLSS-Intro.pdf
- [Parkes] CS181, Parkes and Rush, Harvard University
  - http://cs181.fas.harvard.edu
- [Ng] CS229, Ng, Stanford University
  - http://cs229.stanford.edu/
- [Rogozhnikov] Machine learning in high energy physics, Alex Rogozhnikov
  - https://indico.cern.ch/event/497368/
- [Fleuret] Francois Fleuret, EE559 Deep Learning, EPFL, 2018
  - https://documents.epfl.ch/users/f/fl/fleuret/www/dlc/

# Backup

# Notation



- $\mathbf{X} \in \mathbb{R}^{mxn}$ Matrices in bold upper case:
- $\mathbf{x} \in \mathbb{R}^{n(x1)}$ Vectors in bold lower case
- $x \in \mathbb{R}$ Scalars in lower case, non-bold
- $\mathcal{X}$ Sets are script
- $\{\mathbf{x}_i\}_1^m$ Sequence of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$
- $y \in \mathbb{I}^{(k)} / \mathbb{R}^{(k)}$ Labels represented as
  - Integer for classes, often {0,1}. E.g. {Higgs, Z}
  - Real number. E.g electron energy

- Variables = features = inputs
- Data point $\mathbf{x} = \{x_1, \ldots, x_n\}$ has n-features

- Typically use affine coordinates:

$$y = \mathbf{w}^T\mathbf{x} + w_0 \rightarrow \mathbf{w}^T\mathbf{x}$$
$$\rightarrow \mathbf{w} = \{w_0, w_1, \ldots, w_n\}$$
$$\rightarrow \mathbf{x} = \{1, x_1, \ldots, x_n\}$$

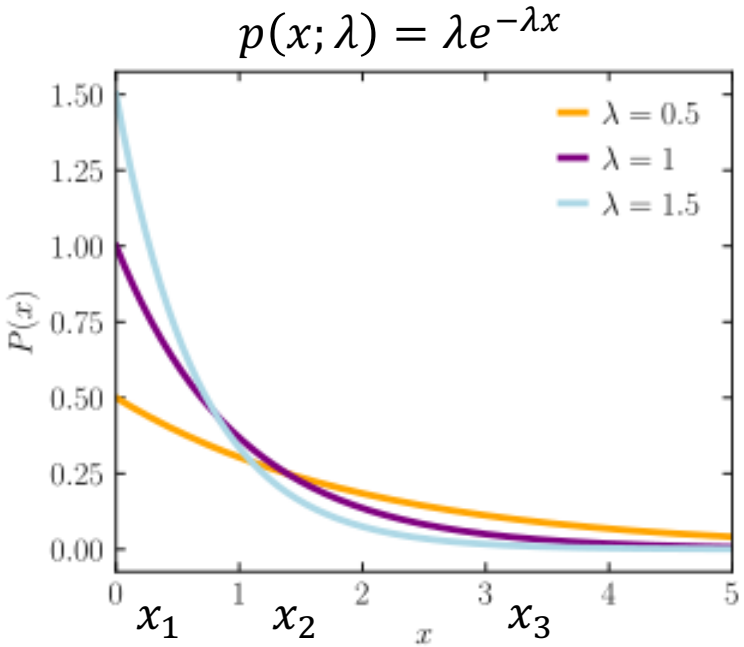
# Maximum Likelihood



- Describe a process behind the data
- Write down the likelihood of the observed data

$$\mathcal{L}(\mathbf{w}) = p(\mathbf{y}|\mathbf{X};\mathbf{w}) = \prod_i p(y_i|\mathbf{x}_i;\mathbf{w})$$

- Example: have samples $x_{1:n}$
  - Assume data comes from exponential distribution
    - $p(x_i;\lambda) = \lambda e^{-\lambda x_i}$



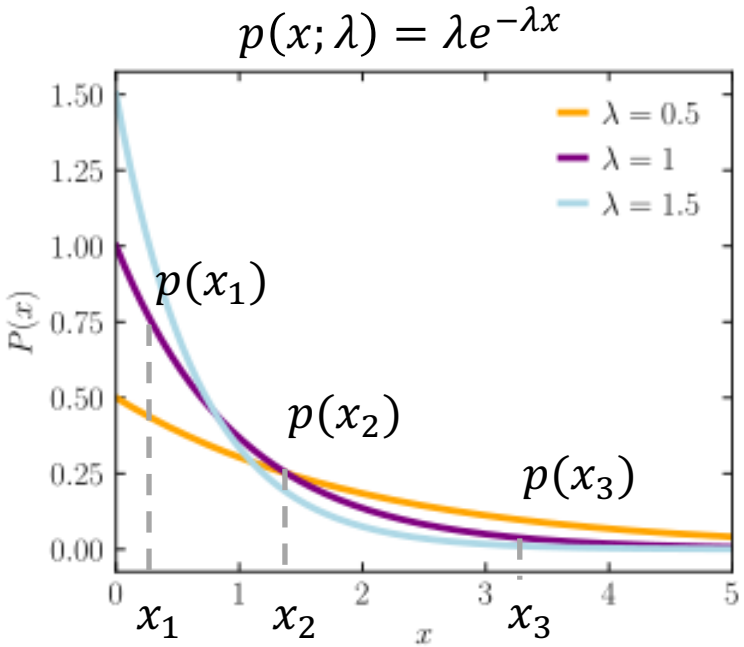
# Maximum Likelihood



- Describe a process behind the data
- Write down the likelihood of the observed data

$$\mathcal{L}(\mathbf{w}) = p(\mathbf{y}|\mathbf{X};\mathbf{w}) = \prod_i p(y_i|\mathbf{x}_i;\mathbf{w})$$

- Example: have samples $x_{1:n}$
  - Assume data comes from exponential distribution
    - $p(x_i;\lambda) = \lambda e^{-\lambda x_i}$
  - Evaluate $p(x_i;\lambda)$ for each $x_i$



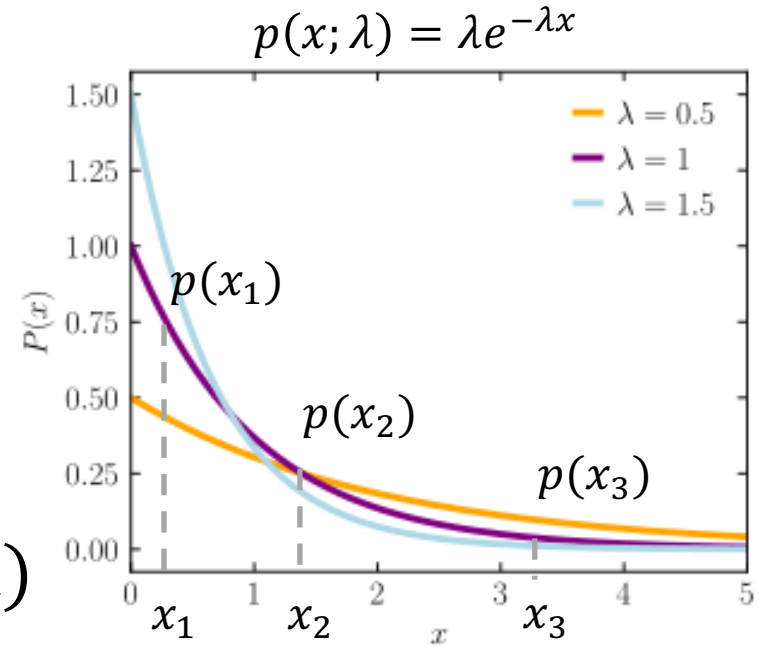
# Maximum Likelihood



- Describe a process behind the data
- Write down the likelihood of the observed data

$$\mathcal{L}(\mathbf{w}) = p(\mathbf{y}|\mathbf{X};\mathbf{w}) = \prod_i p(y_i|\mathbf{x}_i;\mathbf{w})$$

- Example: have samples $x_{1:n}$
  - Assume data comes from exponential distribution
    - $p(x_i;\lambda) = \lambda e^{-\lambda x_i}$
  - Evaluate $p(x_i;\lambda)$ for each $x_i$
  - Find $\lambda$ to maximize $\prod_i p(x_i;\lambda)$



# Bias Variance Tradeoff



- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$
$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$E[(y - h(x))^2] = E[(y - \bar{y})^2] + (\bar{y} - \bar{h}(x))^2 + E[(h(x) - \bar{h}(x))^2]$$
$$= \text{noise} + (\text{bias})^2 + \text{variance}$$

# Bias Variance Tradeoff



- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$
$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$E[(y - h(x))^2] = E[(y - \bar{y})^2] + (\bar{y} - \bar{h}(x))^2 + E[(h(x) - \bar{h}(x))^2]$$
$$= \text{noise} + (\text{bias})^2 + \text{variance}$$

Intrinsic noise in system or measurements
Can not be avoided or improved with modeling
Lower bound on possible noise

# Bias Variance Tradeoff



- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$
$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$E[(y - h(x))^2] = \boxed{E[(y - \bar{y})^2]} + \boxed{(\bar{y} - \bar{h}(x))^2} + E[(h(x) - \bar{h}(x))^2]$$
$$= \boxed{\text{noise}} + \boxed{(\text{bias})^2} + \text{variance}$$

- The **more complex the model** h(x) is, the more data points it will capture, and **the lower the bias** will be.

# Bias Variance Tradeoff



- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$
$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$E[(y - h(x))^2] = \boxed{E[(y - \bar{y})^2]} + \boxed{(\bar{y} - \bar{h}(x))^2} + \boxed{E[(h(x) - \bar{h}(x))^2]}$$
$$= \boxed{\text{noise}} + \boxed{(\text{bias})^2} + \boxed{\text{variance}}$$

- The **more complex the model** h(x) is, the more data points it will capture, and **the lower the bias** will be.

- **More Complexity** will make the model "move" more to capture the data points, and hence its **variance will be larger**.

# Bias Variance Tradeoff



- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$
$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$E[(y - h(x))^2] = \boxed{E[(y - \bar{y})^2]} + \boxed{(\bar{y} - \bar{h}(x))^2} + \boxed{E[(h(x) - \bar{h}(x))^2]}$$
$$= \boxed{\text{noise}} + \boxed{(\text{bias})^2} + \boxed{\text{variance}}$$

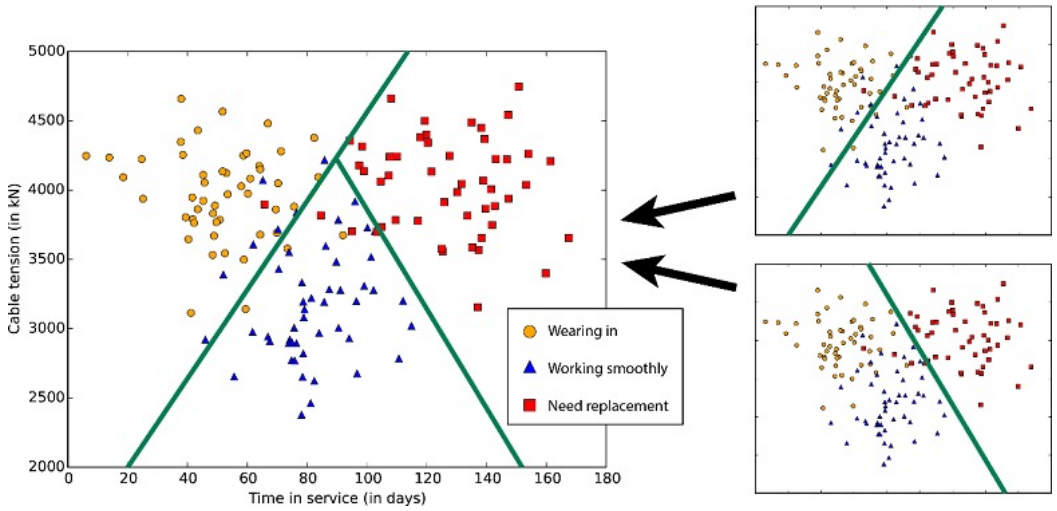
- The **more complex the model** h(x) is, the more data points it will capture, and **the lower the bias** will be.

- **More Complexity** will make the model "move" more to capture the data points, and hence its **variance will be larger**.
  - **As dataset size grows, can reduce variance! Can use more complex model**

# Multiclass Classification?



- What if there is more than two classes?




- Softmax → multi-class generalization of logistic loss
  - Have N classes $\{c_1, \ldots, c_N\}$
  - Model target $\mathbf{y}_k = (0, \ldots, 1, \ldots 0)$

$k^{th}$ element in vector

$$p(c_k|x) = \frac{\exp(\mathbf{w}_k x)}{\sum_j \exp(\mathbf{w}_j x)}$$

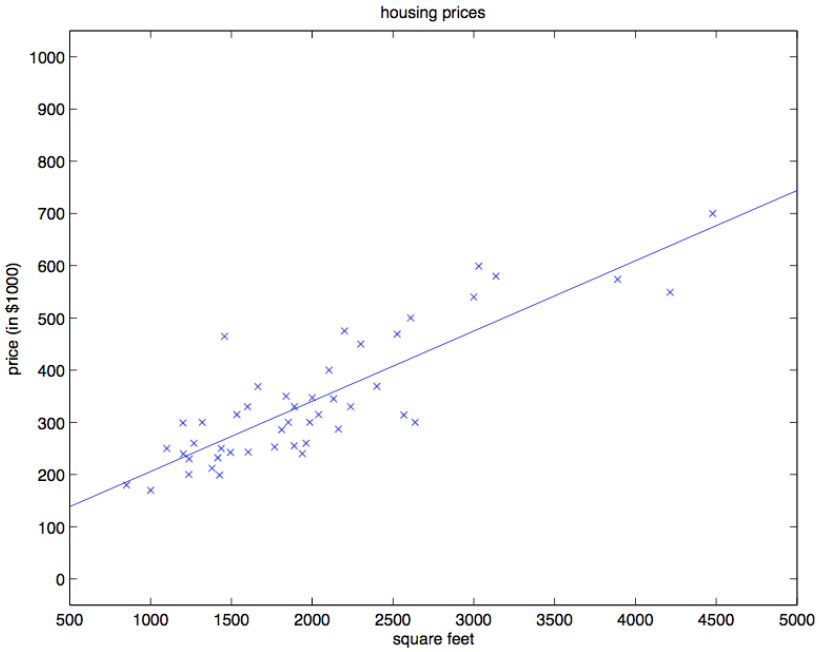
  - Gradient descent for each of the weights $\mathbf{w}_k$

# Least Squares Linear Regression

# Least Squares Linear Regression



- Set of input / output pairs $D = \{\mathbf{x}_i , y_i\}_{i=1\ldots n}$
  - $\mathbf{x}_i \in \mathbb{R}^m$
  - $y_i \in \mathbb{R}$




- Assume a linear model
  $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T\mathbf{x}$

- Squared Loss function:

$$L(\mathbf{w}) = \frac{1}{2}\sum_i \left(y_i - h(\mathbf{x}_i; \mathbf{w})\right)^2$$

- Find $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$

# Least Squares Linear Regression: Matrix Form



- Set of input / output pairs $D = \{\mathbf{x}_i , y_i\}_{i=1\ldots n}$
  - Design matrix $\mathbf{X} \in \mathbb{R}^{nxm}$
  - Target vector $\mathbf{y} \in \mathbb{R}^{n}$

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

# Least Squares Linear Regression: Matrix Form



- Set of input / output pairs $D = \{\mathbf{x}_i , y_i\}_{i=1\ldots n}$
  - Design matrix $\mathbf{X} \in \mathbb{R}^{nxm}$
  - Target vector $\mathbf{y} \in \mathbb{R}^n$

- Rewrite loss:

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

- Minimize w.r.t. $\mathbf{w}$:

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \arg\min_{\mathbf{w}} L(\mathbf{w})$$

# Linear Regression – Probabilistic Interpretation



- Assume $y_i = mx_i + e_i$
- Random error: $e_i \sim \mathcal{N}(0, \sigma) \rightarrow p(e_i) \propto \exp\left(\frac{1}{2}\frac{e_i^2}{\sigma^2}\right)$
  - Noisy measurements, unmeasured variables, …

# Linear Regression – Probabilistic Interpretation



- Assume $y_i = mx_i + e_i$
- Random error: $e_i \sim \mathcal{N}(0, \sigma) \rightarrow p(e_i) \propto \exp\left(\frac{1}{2}\frac{e_i^2}{\sigma^2}\right)$
  - Noisy measurements, unmeasured variables, …
- Then $y_i \sim \mathcal{N}(mx_i, \sigma) \rightarrow p(y_i|x_i; m) \propto \exp\left(\frac{1}{2}\frac{(y_i - mx_i)^2}{\sigma^2}\right)$

# Linear Regression – Probabilistic Interpretation



- Assume $y_i = mx_i + e_i$
- Random error: $e_i \sim \mathcal{N}(0, \sigma) \rightarrow p(e_i) \propto \exp\left(\frac{1}{2}\frac{e_i^2}{\sigma^2}\right)$
  - Noisy measurements, unmeasured variables, …
- Then $y_i \sim \mathcal{N}(mx_i, \sigma) \rightarrow p(y_i|x_i; m) \propto \exp\left(\frac{1}{2}\frac{(y_i - mx_i)^2}{\sigma^2}\right)$
- Likelihood function:

$$L(m) = p(\mathbf{y}|\mathbf{X}; m) = \prod_i p(y_i|x_i; m)$$

$$\rightarrow -\log L(m) \sim \sum_i (y_i - mx_i)^2$$

# Linear Regression – Probabilistic Interpretation



- Assume $y_i = mx_i + e_i$
- Random error: $e_i \sim \mathcal{N}(0, \sigma) \rightarrow p(e_i) \propto \exp\left(\frac{1}{2}\frac{e_i^2}{\sigma^2}\right)$
  - Noisy measurements, unmeasured variables, …
- Then $y_i \sim \mathcal{N}(mx_i, \sigma) \rightarrow p(y_i|x_i; m) \propto \exp\left(\frac{1}{2}\frac{(y_i - mx_i)^2}{\sigma^2}\right)$
- Likelihood function:

$$L(m) = p(\mathbf{y}|\mathbf{X}; m) = \prod_i p(y_i|x_i; m)$$

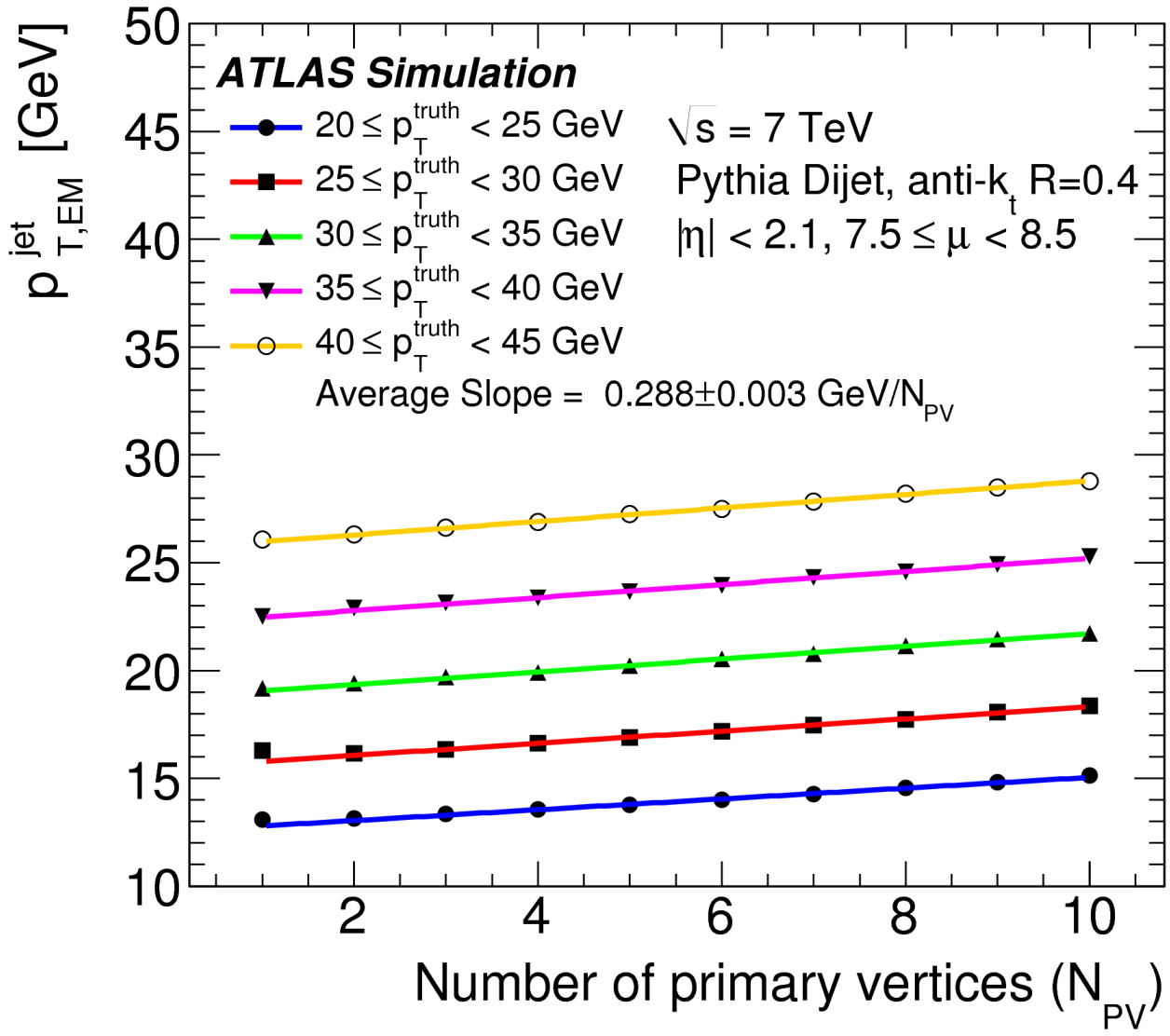
$$\rightarrow -\log L(m) \sim \sum_i (y_i - mx_i)^2$$

Squared loss function!

# Linear Regression Example



Eur. Phys. J. C (2015) 75:17



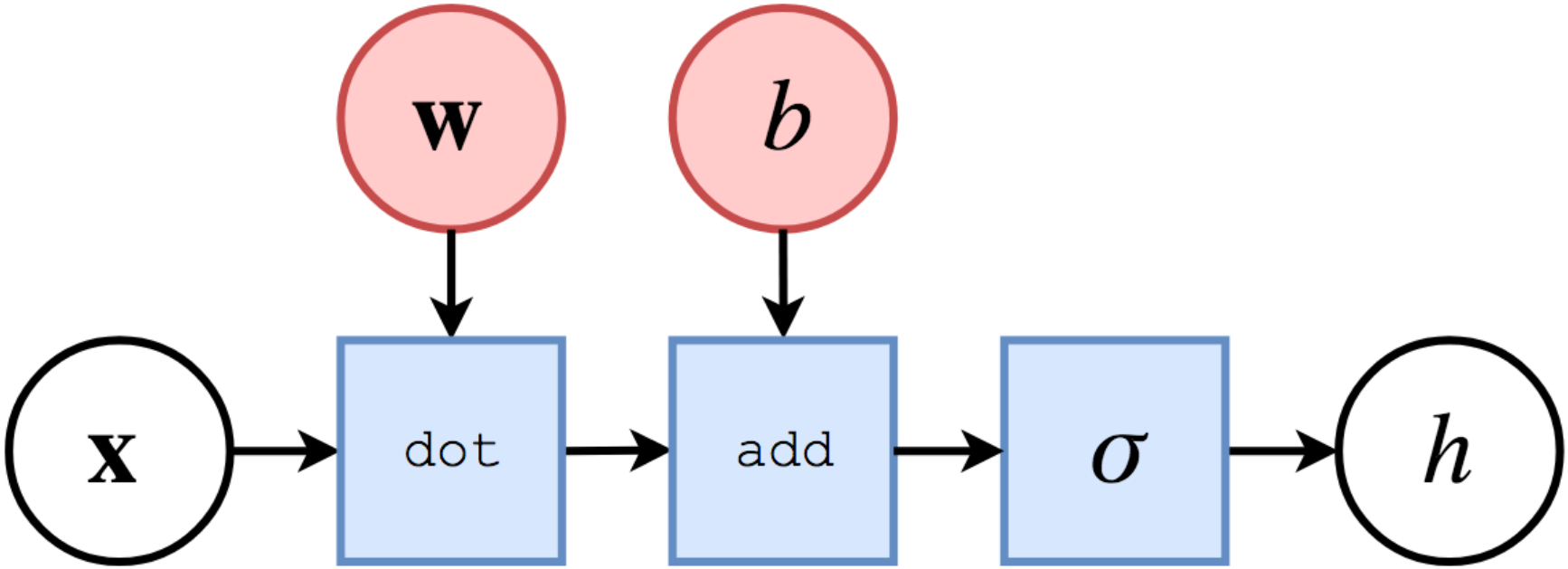

- Reconstructed Jet energy vs. Number of primary vertices

# Logistic Regression



- Computational Graph of function
  - White node = input
  - Red node = model parameter
  - Blue node = intermediate operations

Slide credit: G. Louppe




This unit is the main building block of Neural Networks!

# How to Minimize Loss $\mathcal{L}(\theta)$? Gradient Descent



- **Gradient Descent**:

  Make a step $\theta \leftarrow \theta - \eta v$ in ***direction*** $v$ with ***step size*** $\eta$ to reduce loss

- How does loss change in different directions?

  Let $\lambda$ be a perturbation along direction $v$

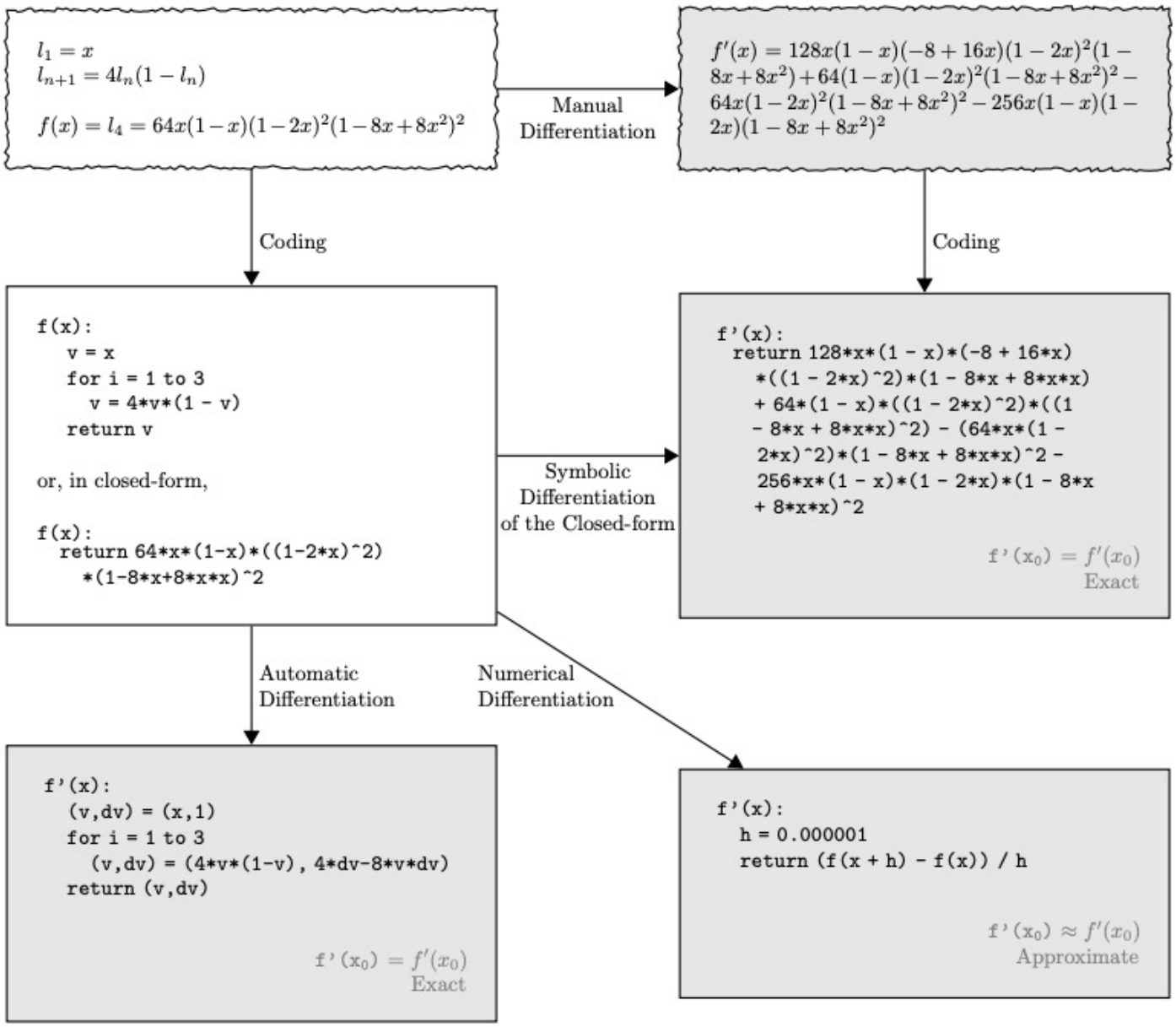
$$\frac{d}{d\lambda}\mathcal{L}(\theta + \lambda v)\Big|_{\lambda=0} = v \cdot \nabla_\theta \mathcal{L}(\theta)$$

- Then Steepest Descent direction is: $v = -\nabla_\theta \mathcal{L}(\theta)$

# Automatic Differentiation

# Differentiation in Code






Baydin, Pearlmutter, Radul, Siskind. 2018. "Automatic Differentiation in Machine Learning: a Survey." Journal of Machine Learning Research (**JMLR**)

# Chain Rule – Symbolic Differentiation Painful!



$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(\sigma(h(\mathbf{x}_i))) + (1 - y_i) \ln(1 - \sigma(h(\mathbf{x}_i)))$$

- Derivative of sigmoid: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$
- Chain rule to compute gradient w.r.t. **w**

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \mathbf{w}} = \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))\sigma(\mathbf{U}\mathbf{x}) + (1 - y_i)\sigma(h(\mathbf{x}))\sigma(\mathbf{U}\mathbf{x}_i)$$

- Chain rule to compute gradient w.r.t. $\mathbf{u}_j$

$$\frac{\partial L}{\partial \mathbf{u}_j} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \sigma}\frac{\partial \sigma}{\partial \mathbf{u}_j} =$$

$$= \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))w_j\sigma(\mathbf{u}_j\mathbf{x}_i)(1 - \sigma(\mathbf{u}_j\mathbf{x}_i))\mathbf{x}_i$$

$$+ (1 - y_i)\sigma(h(\mathbf{x}_i))w_j\sigma(\mathbf{u}_j\mathbf{x}_i)(1 - \sigma(\mathbf{u}_j\mathbf{x}_i))\mathbf{x}_i$$

# Forward and Reverse Mode



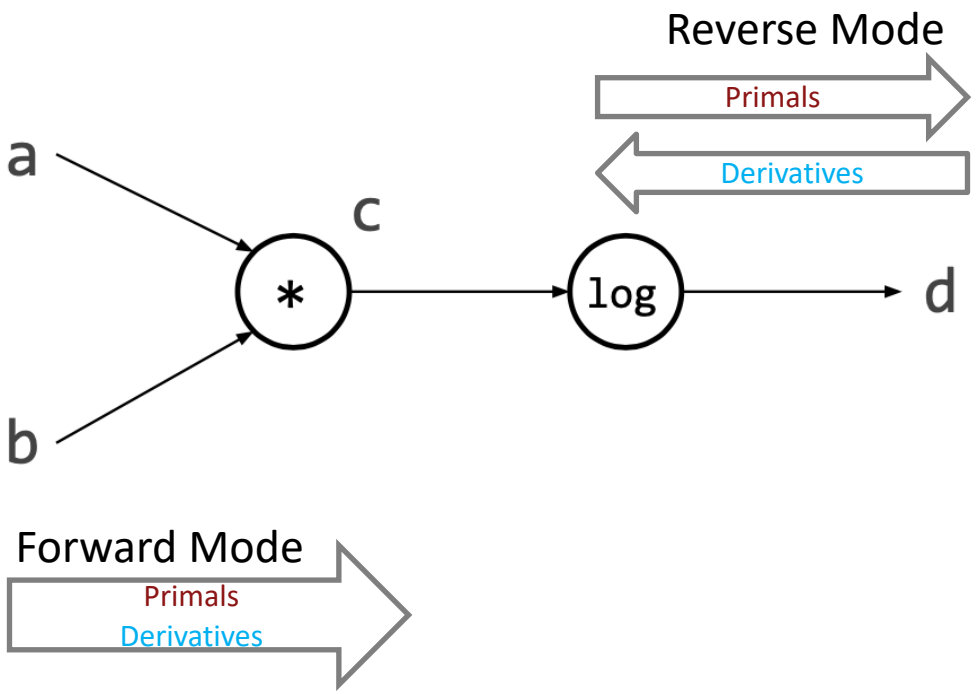
- Derivatives can be computed in **Forward Mode** and **Reverse Mode**

Forward Mode Single Evaluation: $\boldsymbol{f}(\boldsymbol{x})$: $\mathbb{R}^N \rightarrow \mathbb{R}^M$

$$\frac{d\boldsymbol{f}(\boldsymbol{x})}{d\boldsymbol{x}} = \begin{pmatrix} \frac{df_1}{dx_1} & \cdots & \frac{df_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{df_1}{dx_N} & \cdots & \frac{df_M}{dx_N} \end{pmatrix}$$

Reverse Mode Single Evaluation: $\boldsymbol{f}(\boldsymbol{x})$: $\mathbb{R}^N \rightarrow \mathbb{R}^M$

$$\frac{d\boldsymbol{f}(\boldsymbol{x})}{d\boldsymbol{x}} = \begin{pmatrix} \frac{df_1}{dx_1} & \cdots & \frac{df_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{df_1}{dx_N} & \cdots & \frac{df_M}{dx_N} \end{pmatrix}$$




Chain Rule: $\frac{\partial d}{\partial a} = \frac{\partial d}{\partial c}\frac{\partial c}{\partial a}$

# Automatic Differentiation Example



- All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives
  - Represent as a **computational graph** showing dependencies

$$f(a, b) = \log(ab)$$

$$\nabla f(a, b) = \left(\frac{1}{a}, \frac{1}{b}\right)$$

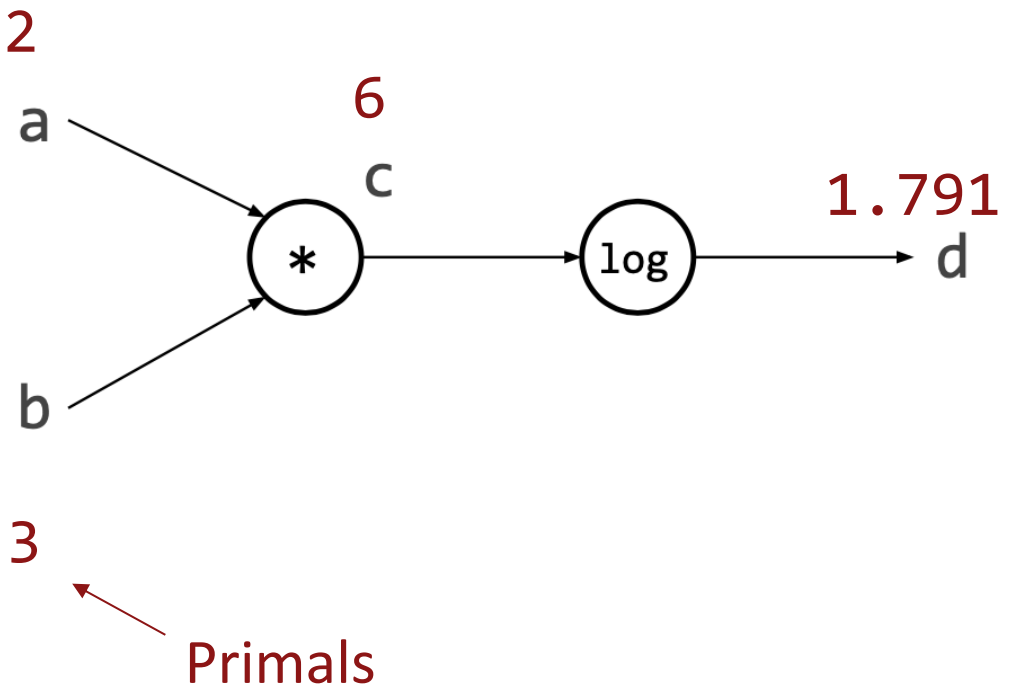
# Automatic Differentiation Example



- All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives
  - Represent as a **computational graph** showing dependencies

```
f(a, b):
    c = a * b
    d = log(c)
    return d
```

f(2, 3) = 1.791

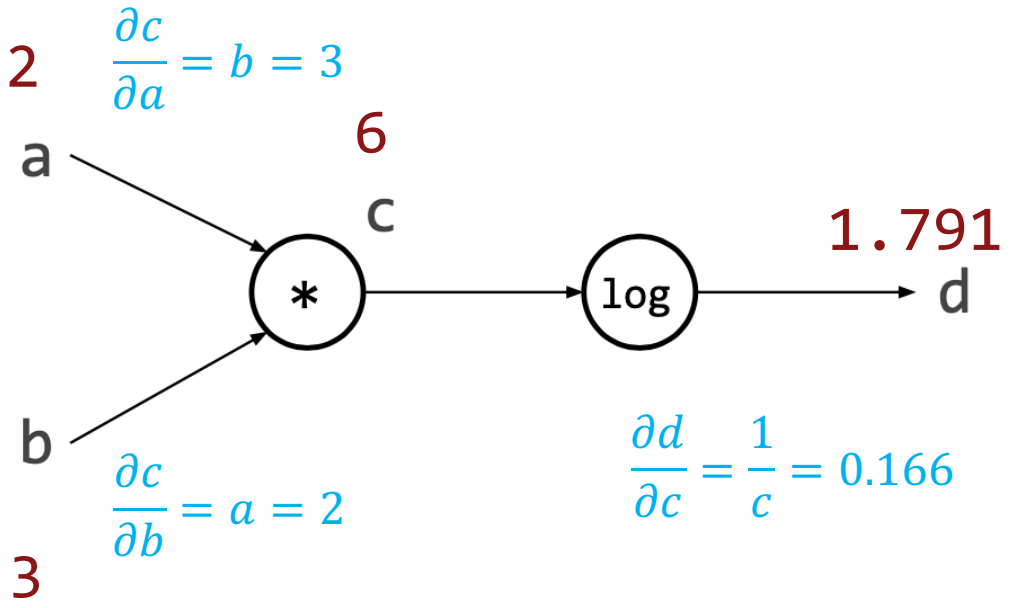


# Automatic Differentiation Example



- All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives
  - Represent as a **computational graph** showing dependencies

```
f(a, b):
    c = a * b
    d = log(c)
    return d
```




f(2, 3) = 1.791
df(2,3) = [0.5, 0.333]

Chain Rule: $\frac{\partial d}{\partial a} = \frac{\partial d}{\partial c}\frac{\partial c}{\partial a} = 0.166 * 3 = 0.5$

# Automatic Differentiation



**Problem**: Compute gradients of $z$ with respect to inputs $\{x_1, x_2\}$

$$z = \sin(x_1) + x_1 x_2$$

# Automatic Differentiation

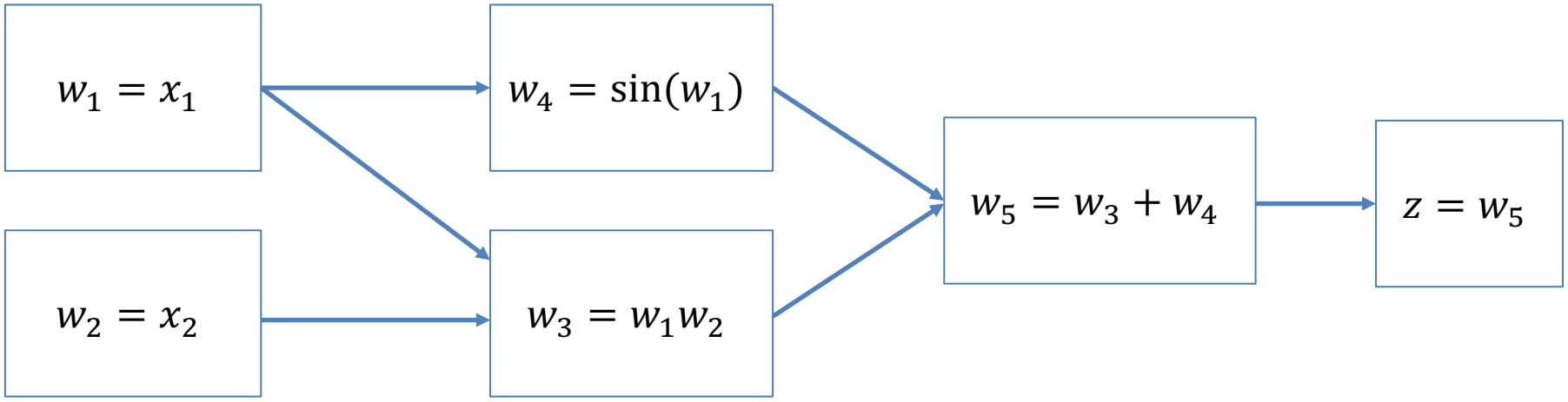


$w_1 = x_1$
$w_2 = x_2$
$w_3 = w_1 w_2$
$w_4 = \sin(w_1)$
$w_5 = w_3 + w_4$
$z = w_5$

**Problem**: Compute gradients of $z$ with respect to inputs $\{x_1, x_2\}$

$$z = \sin(x_1) + x_1 x_2$$

Organize as a computational Graph



# Automatic Differentiation



$w_1 = x_1$
$w_2 = x_2$
$w_3 = w_1 w_2$
$w_4 = \sin(w_1)$
$w_5 = w_3 + w_4$
$z = w_5$

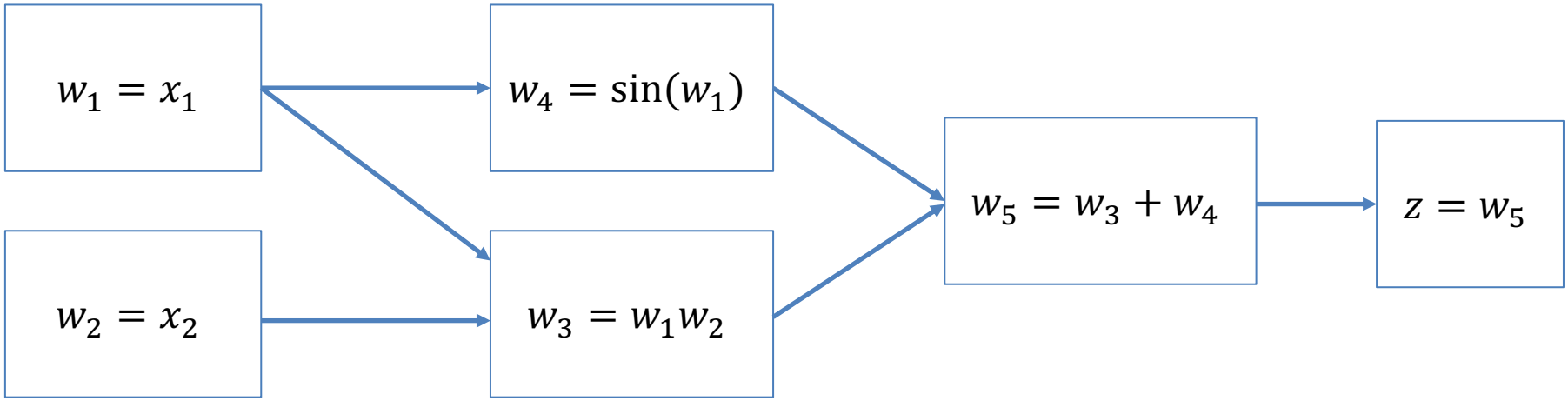
**Problem**: Compute gradients of $z$ with respect to inputs $\{x_1, x_2\}$

We know the gradients of simple functions: $\sin(x), x * y, x + y \ldots$

Chain rule:

$$\frac{dz}{dw_1} = \sum_{p \in parents} \frac{dz}{dw_p} \frac{dw_p}{dw_i}$$

$\frac{dw_1}{dx_1} = 1$
$\frac{dw_2}{dx_2} = 1$
$\frac{dw_3}{dw_1} = w_2$ $\frac{dw_3}{dw_2} = w_1$
$\frac{dw_4}{dw_1} = \cos(w_1)$
$\frac{dw_5}{dw_3} = 1$ $\frac{dw_5}{dw_4} = 1$



# Automatic Differentiation

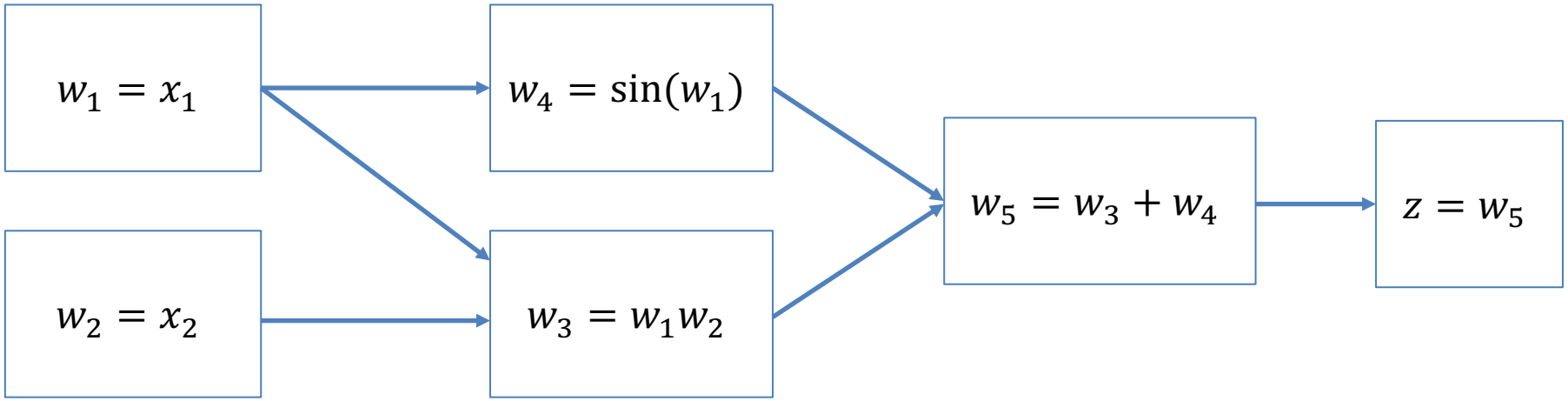


$w_1 = x_1$
$w_2 = x_2$
$w_3 = w_1 w_2$
$w_4 = \sin(w_1)$
$w_5 = w_3 + w_4$
$z = w_5$

**Problem**: Compute gradients of $z$ with respect to inputs $\{x_1, x_2\}$

NOT going to find analytic derivative

WILL find a way to compute value of gradient for a given input point

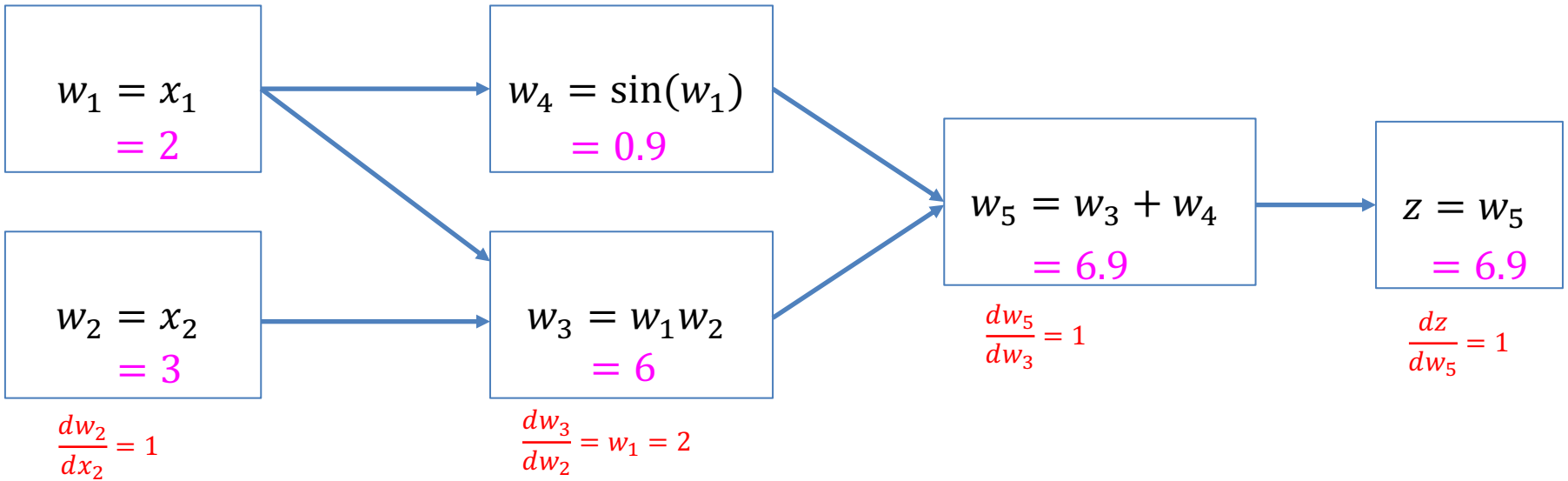


# Forward Mode Automatic Differentiation



$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

For each input, from input to output sequentially, evaluate graph and gradients and store values



# Forward Mode Automatic Differentiation

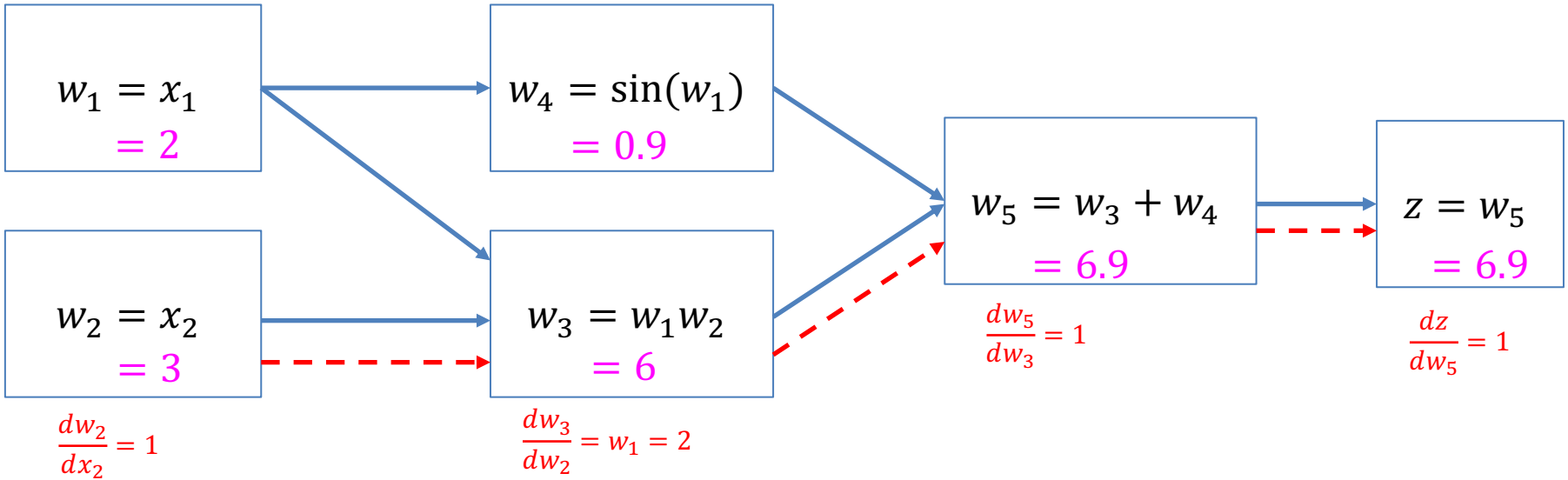


$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

For each input, from input to output sequentially, evaluate graph and gradients and store values

Apply chain rule with multiplication

$$\frac{dz}{dx_2} = \frac{dw_2}{dx_2}\frac{dw_3}{dw_2}\frac{dw_5}{dw_3}\frac{dz}{dw_5} = 1 * 2 * 1 * 1 = 2$$



# Forward Mode Automatic Differentiation

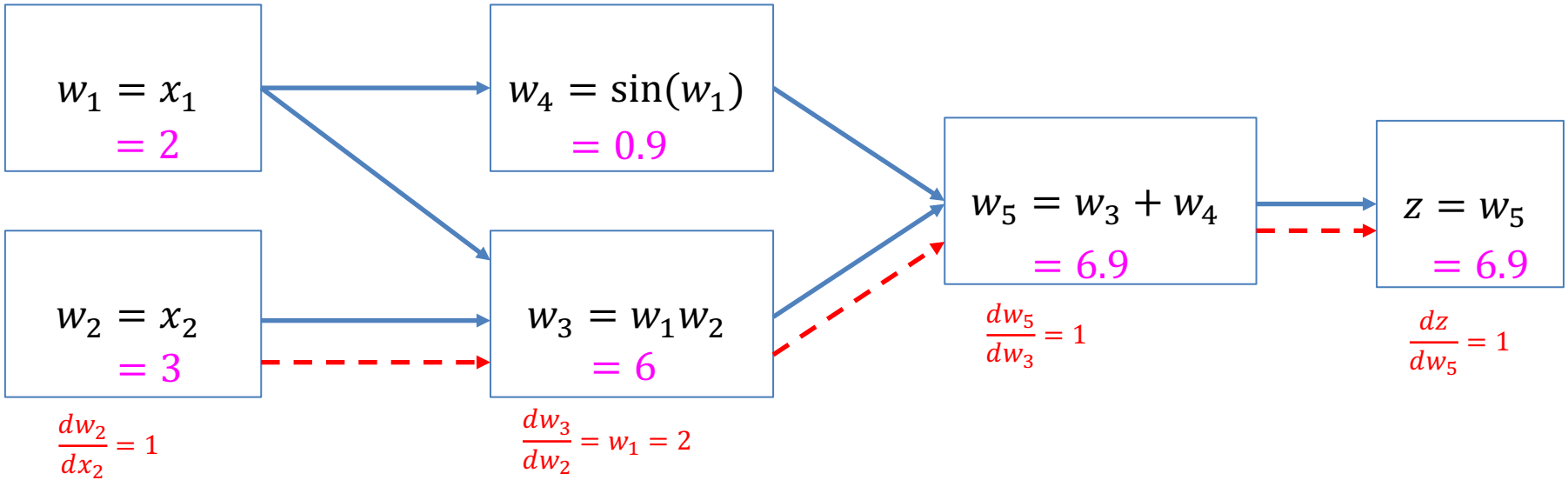


$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Forward Mode allows us to compute the gradient of one input with respect to all the output

$$\text{Jacobian } \frac{dz}{dx} = \begin{pmatrix} \frac{dz_1}{dx_1} & \cdots & \frac{dz_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{dz_1}{dx_N} & \cdots & \frac{dz_M}{dx_N} \end{pmatrix}$$

If we have 1 output (Loss) and many inputs → SLOW!

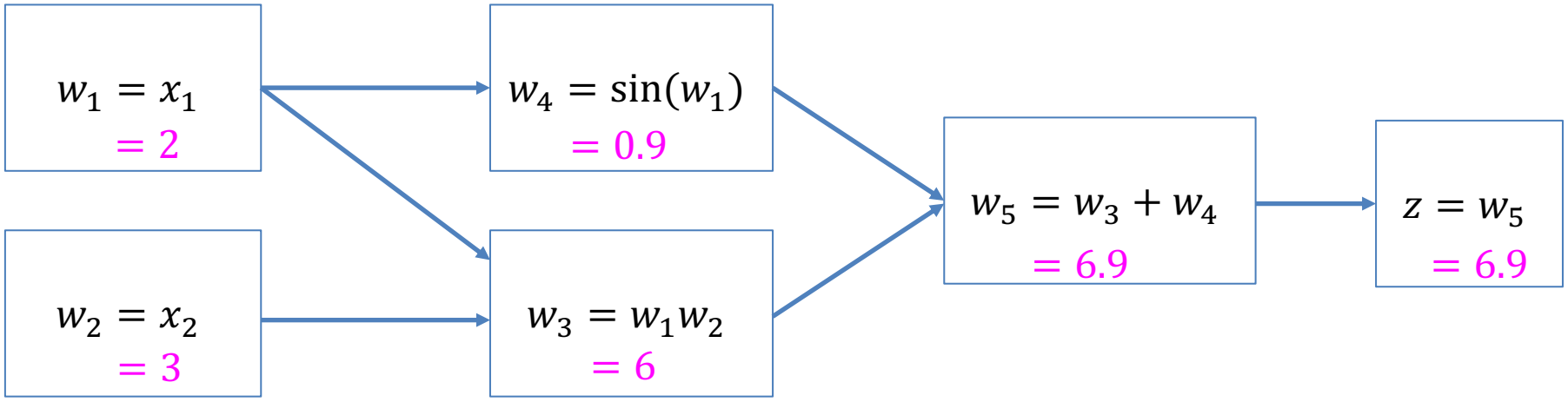


# Reverse Mode Automatic Differentiation



$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Evaluate graph and store values



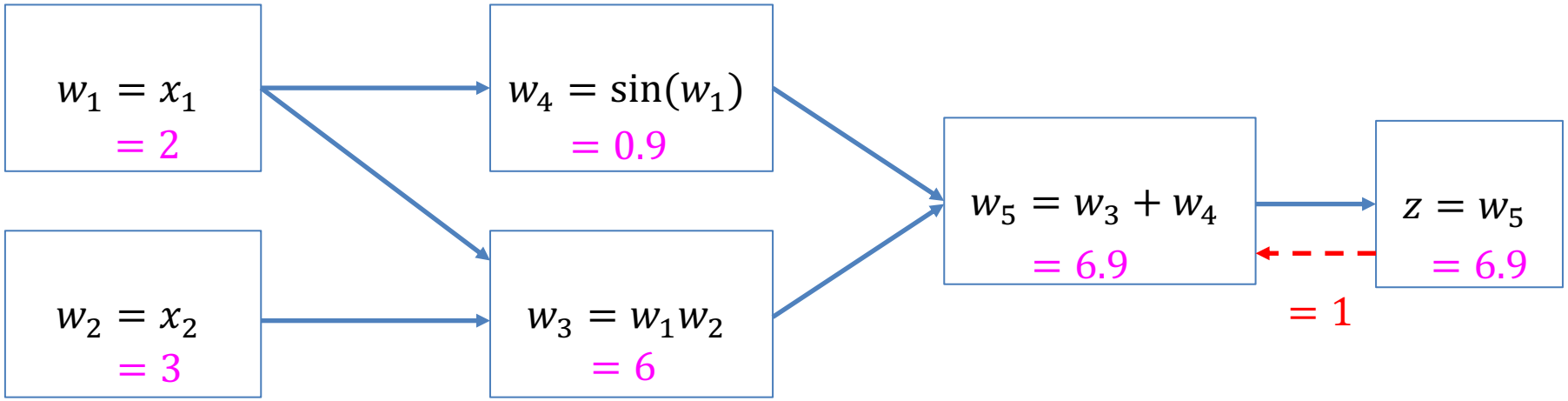
# Reverse Mode Automatic Differentiation



$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$



# Reverse Mode Automatic Differentiation

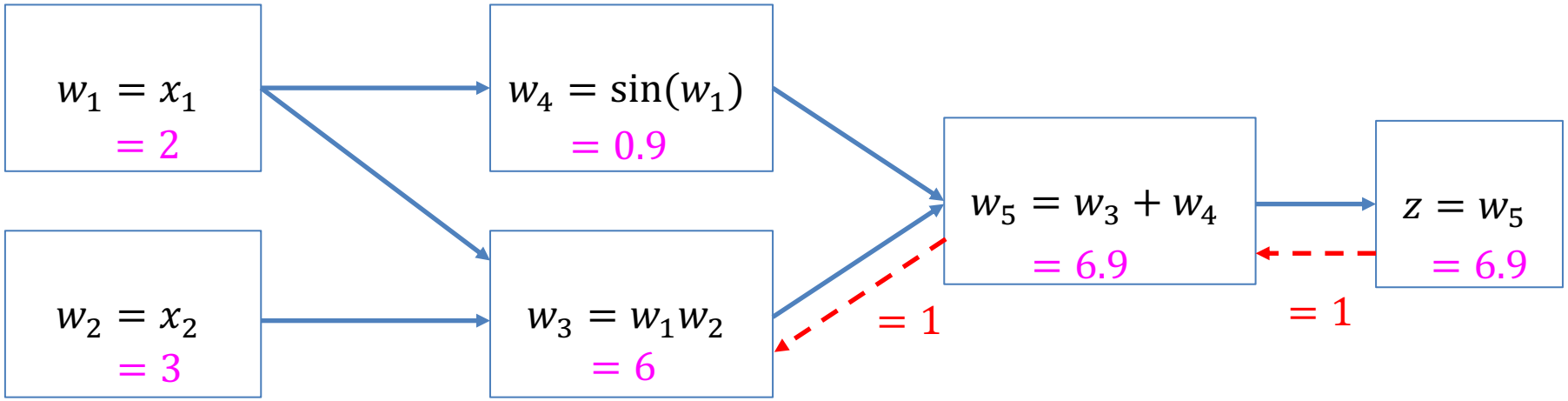


$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$

$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$



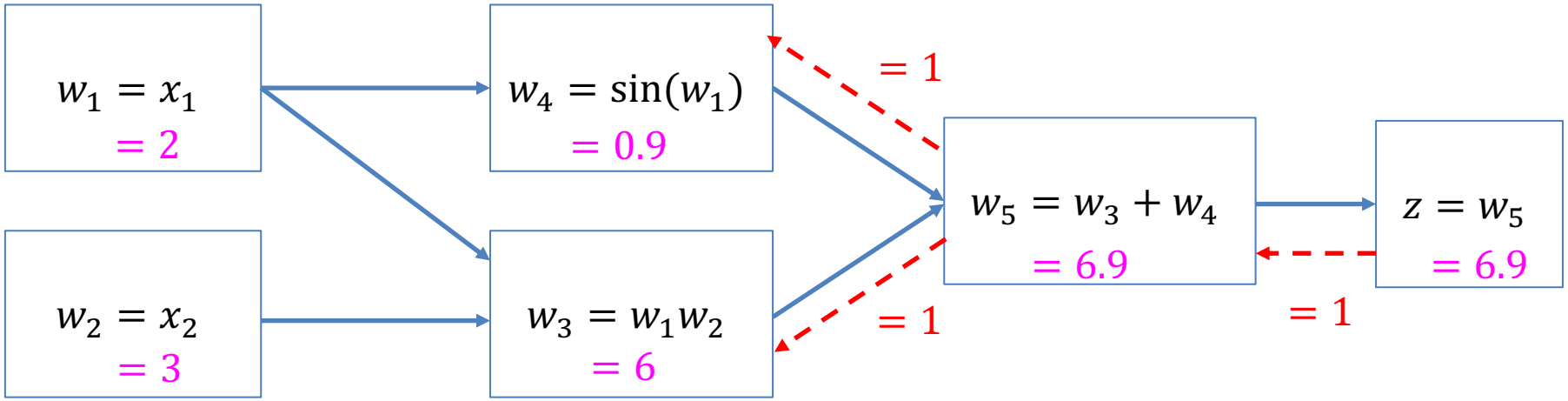
# Reverse Mode Automatic Differentiation



$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$
$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$
$$\frac{dz}{dw_4} = \frac{dz}{dw_5}\frac{dw_5}{dw_4} = 1 \times 1 = 1$$



# Reverse Mode Automatic Differentiation



$$w_1 = x_1 = 2$$
$$w_2 = x_2 = 3$$
$$w_3 = w_1 w_2 = 6$$
$$w_4 = \sin(w_1) = 0.9$$
$$w_5 = w_3 + w_4 = 6.9$$
$$z = w_5$$

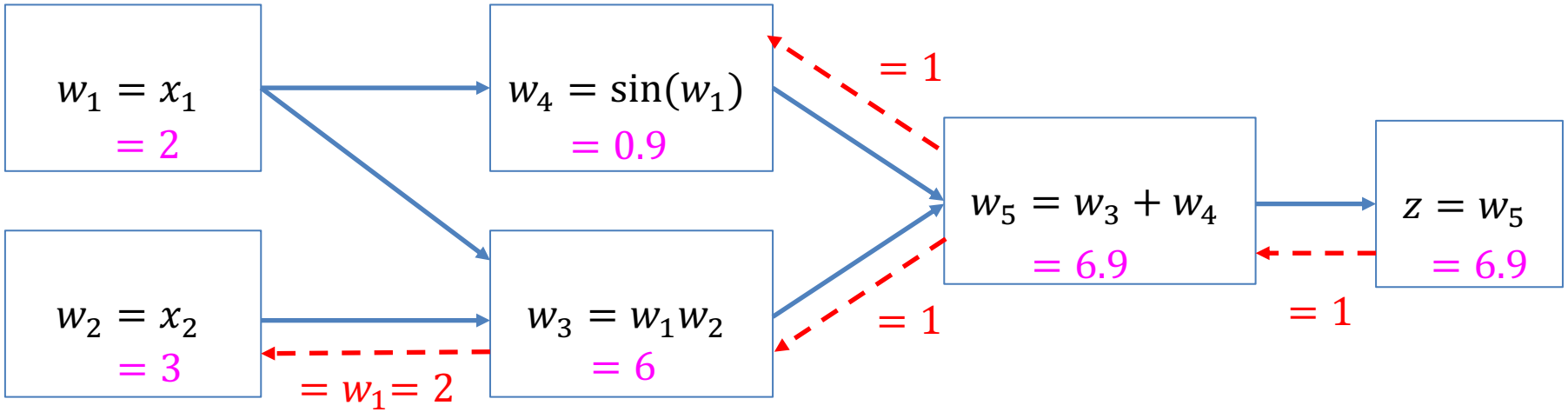
Compute derivatives with chain rule from end to beginning:

$$\frac{dz}{dw_5} = 1$$

$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$

$$\frac{dz}{dw_4} = \frac{dz}{dw_5}\frac{dw_5}{dw_4} = 1 \times 1 = 1$$

$$\frac{dz}{dw_2} = \frac{dz}{dw_3}\frac{dw_3}{dw_2} = 1 \times w_1 = w_1 = 2$$



# Reverse Mode Automatic Differentiation



$$w_1 = x_1 = 2$$
$$w_2 = x_2 = 3$$
$$w_3 = w_1 w_2 = 6$$
$$w_4 = \sin(w_1) = 0.9$$
$$w_5 = w_3 + w_4 = 6.9$$
$$z = w_5$$

Compute derivatives with chain rule from end to beginning:

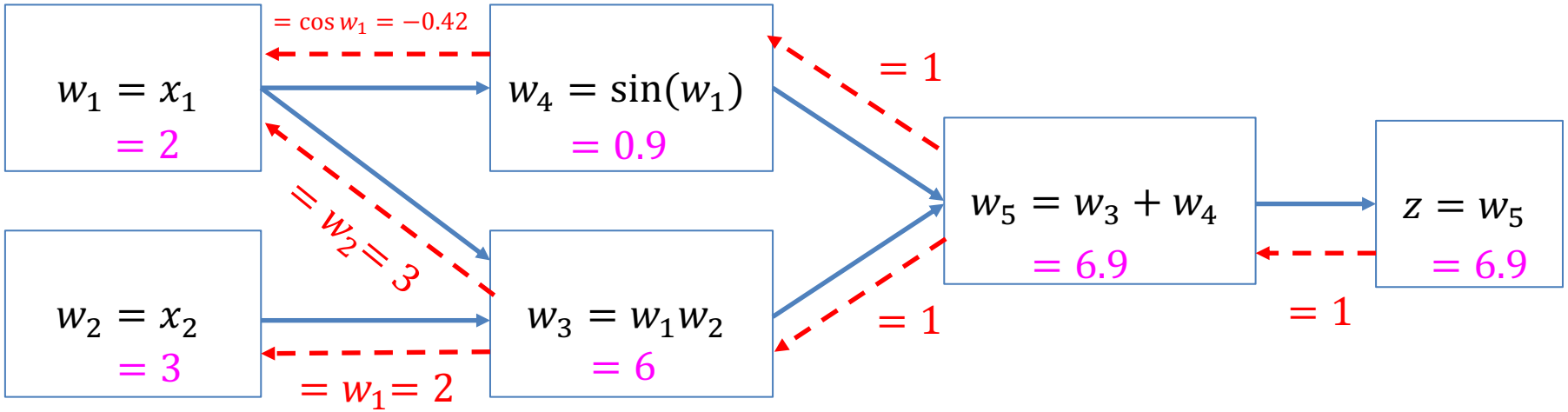
$$\frac{dz}{dw_5} = 1$$

$$\frac{dz}{dw_3} = \frac{dz}{dw_5}\frac{dw_5}{dw_3} = 1 \times 1 = 1$$

$$\frac{dz}{dw_4} = \frac{dz}{dw_5}\frac{dw_5}{dw_4} = 1 \times 1 = 1$$

$$\frac{dz}{dw_2} = \frac{dz}{dw_3}\frac{dw_3}{dw_2} = 1 \times w_1 = w_1 = 2$$

$$\frac{dz}{dw_1} = \frac{dz}{dw_4}\frac{dw_4}{dw_1} + \frac{dz}{dw_3}\frac{dw_3}{dw_1}$$
$$= \cos(w_1) + w_2 = \cos(2) + 3$$
$$= 2.58$$



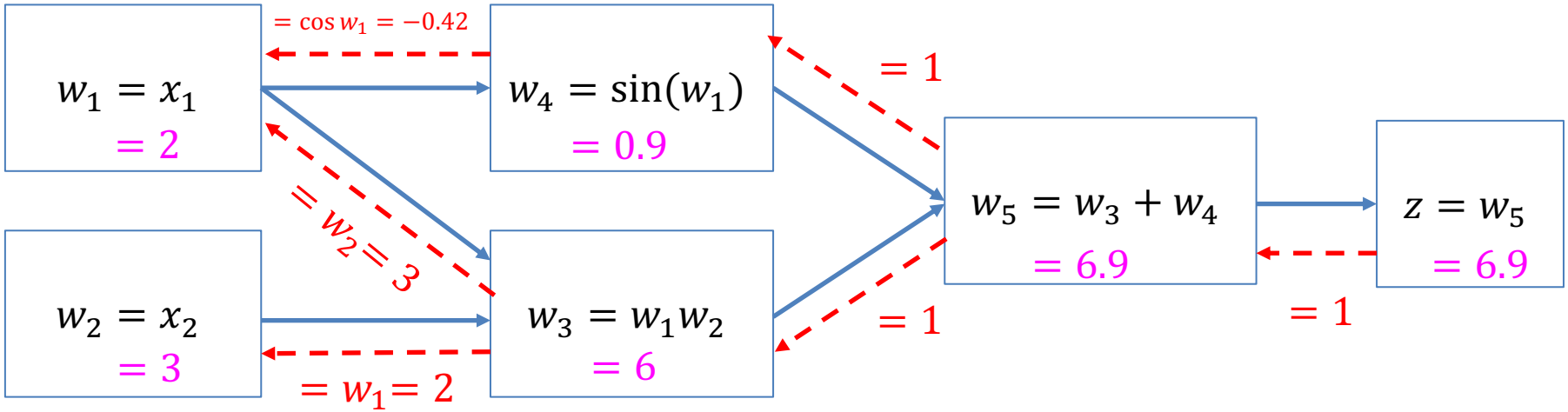
# Reverse Mode Automatic Differentiation



$w_1 = x_1 = 2$
$w_2 = x_2 = 3$
$w_3 = w_1 w_2 = 6$
$w_4 = \sin(w_1) = 0.9$
$w_5 = w_3 + w_4 = 6.9$
$z = w_5$

For each output, can compute the gradient w.r.t. all inputs in one pass!

$$\text{Jacobian } \frac{dz}{dx} = \begin{pmatrix} \frac{dz_1}{dx_1} & \cdots & \frac{dz_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{dz_1}{dx_N} & \cdots & \frac{dz_M}{dx_N} \end{pmatrix}$$



# Backpropagation



- Loss function composed of layers of nonlinearity

$$L\left(\phi^N\left(\ldots\phi^1(x)\right)\right)$$

- Forward step (f-prop)
  - Compute and save intermediate computations

$$\phi^N\left(\ldots\phi^1(x)\right)$$

- Backward step (b-prop)

$$\frac{\partial L}{\partial \phi^a} = \sum_j \frac{\partial \phi_j^{(a+1)}}{\partial \phi_j^a} \frac{\partial L}{\partial \phi_j^{(a+1)}}$$

- Compute parameter gradients

$$\frac{\partial L}{\partial \mathbf{w}^a} = \sum_j \frac{\partial \phi_j^a}{\partial \mathbf{w}^a} \frac{\partial L}{\partial \phi_j^a}$$