Introduction to Machine Learning (Part 1)

Melissa Quinnan (UC San Diego)

Machine Learning for Fundamental Physics 2026

June 1, 2026

Hello!

- ***Today:***
  - Big picture overview of what ML is
  - Intuitive understanding of how it works

Slides inspired by..

- Sascha's introduction from last year
- Dennis's Part II slides
- Javier Duarte's Machine Learning in Physics Course

Who am I?

- The lady who sent you all those emails (sorry!)
- Postdoc @UCSD with the CMS experiment
- Former CMS machine learning group knowledge subgroup convener
- Current offline software convener for the L1 Trigger

UC San Diego

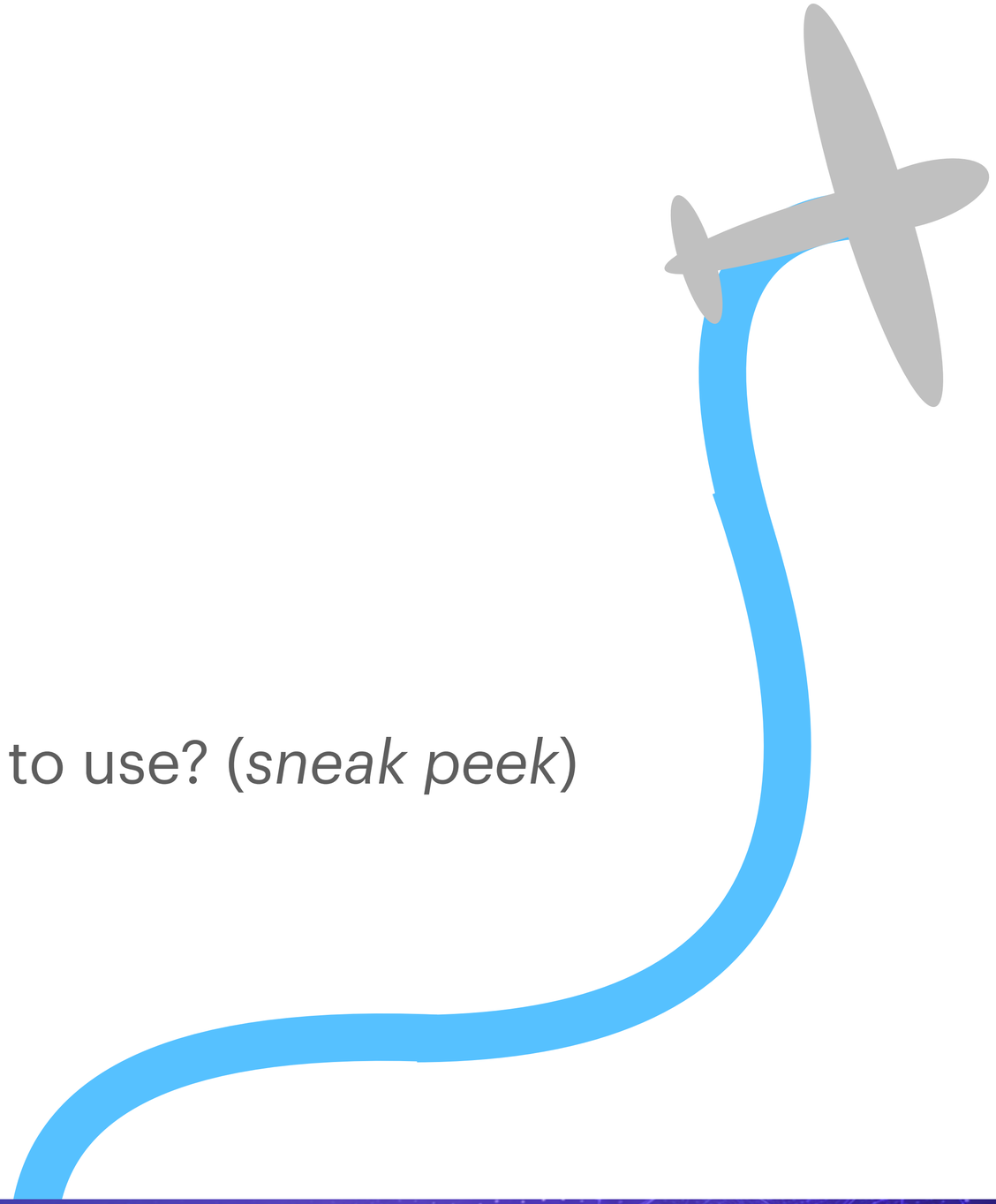
CMS

melissa.quinnan@cern.ch

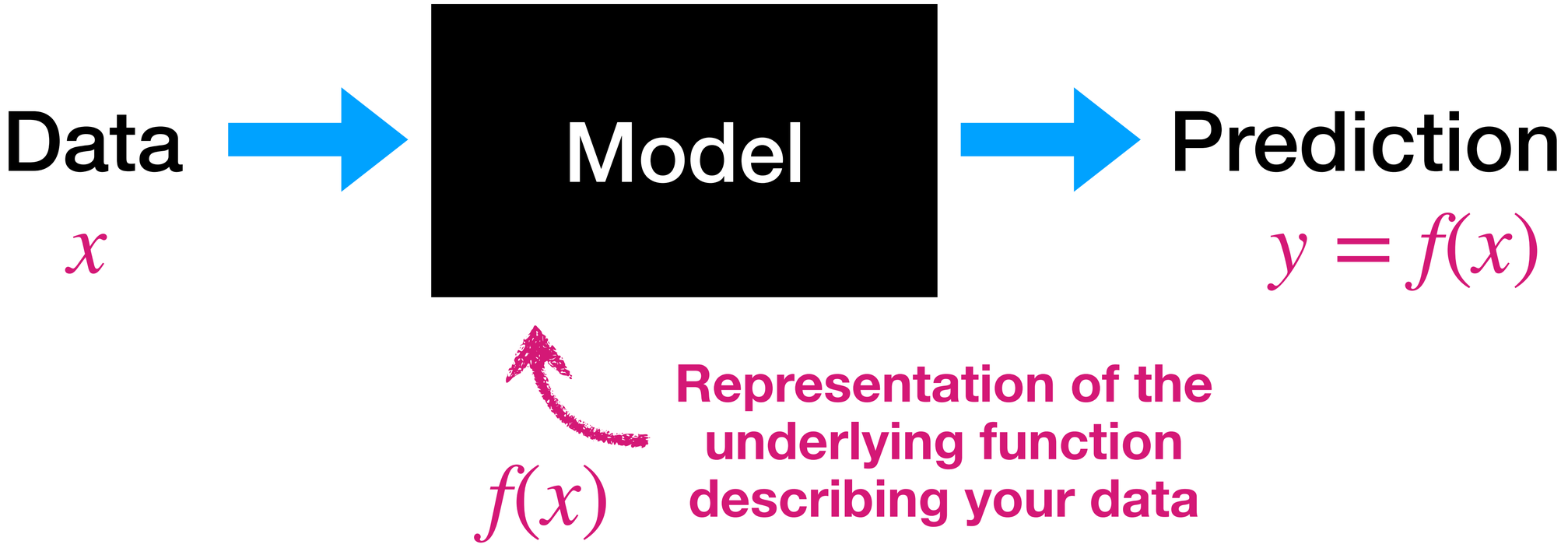
Data Collection

Today's Itinerary

1. What is Machine Learning (ML)?
2. What is a Neural Network (NN)?
3. How do NNs work?
4. How are NNs trained?
5. How do you choose which model to use? (*sneak peek*)



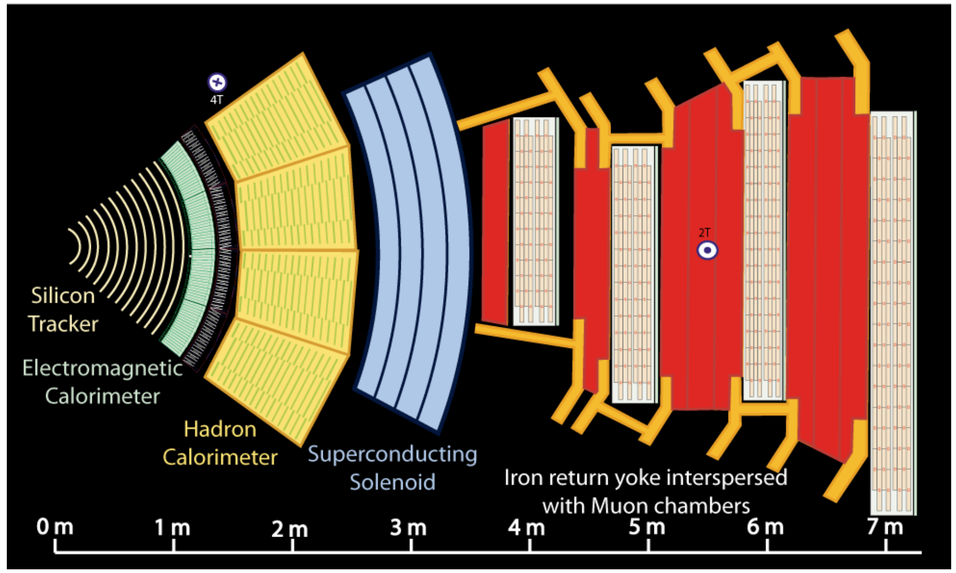
Modeling Data



- Common question in science:
  - What is the “model” that describes that data + allows you to *extrapolate*

Modeling Data

Detector hits → **Model** → Estimate particle momentum, charge, type…

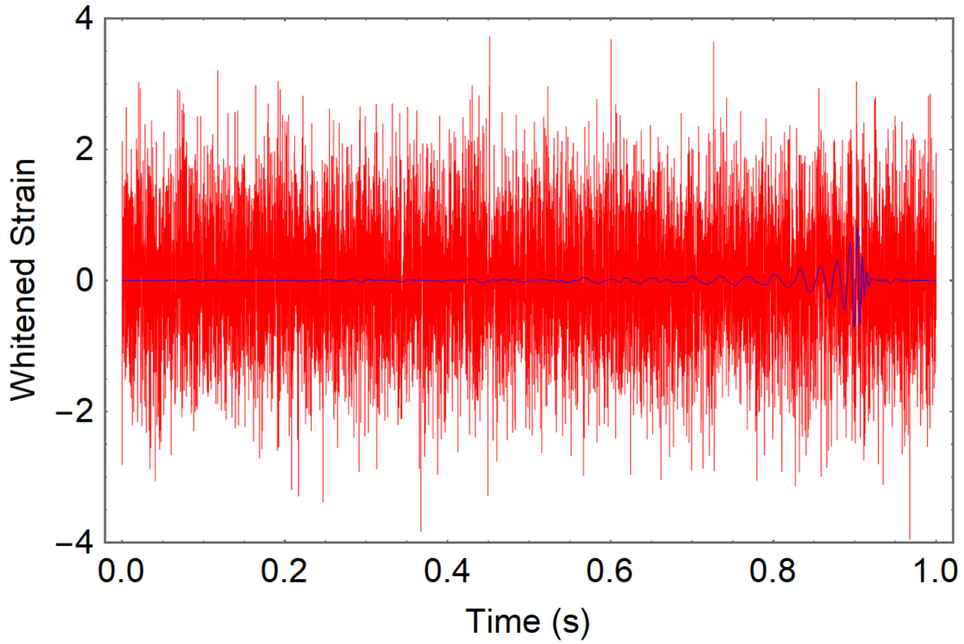


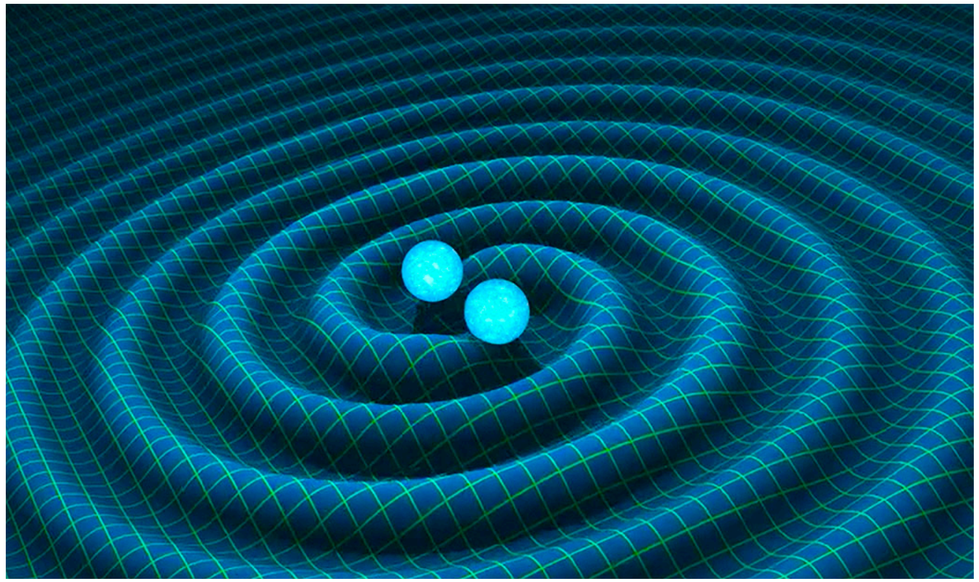
$$\rightarrow \begin{pmatrix} E \\ p_x \\ p_y \\ p_z \end{pmatrix}, q, \text{type}, p_{\text{pileup}}, \cdots$$

arXiv:2101.08578

Modeling Data

Time series data → **Model** → Reduce noise + identify gravitational waves

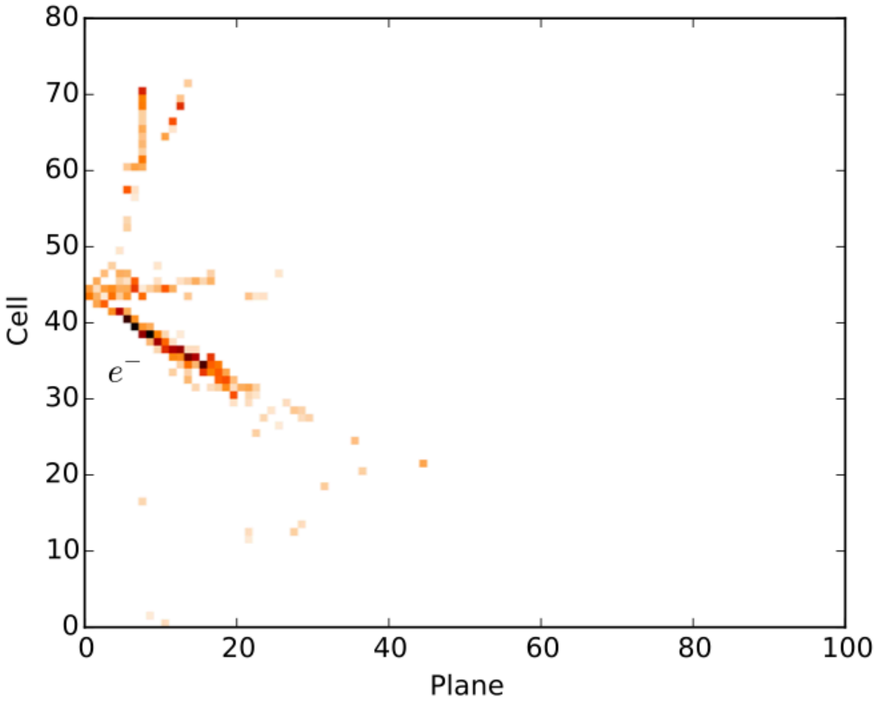


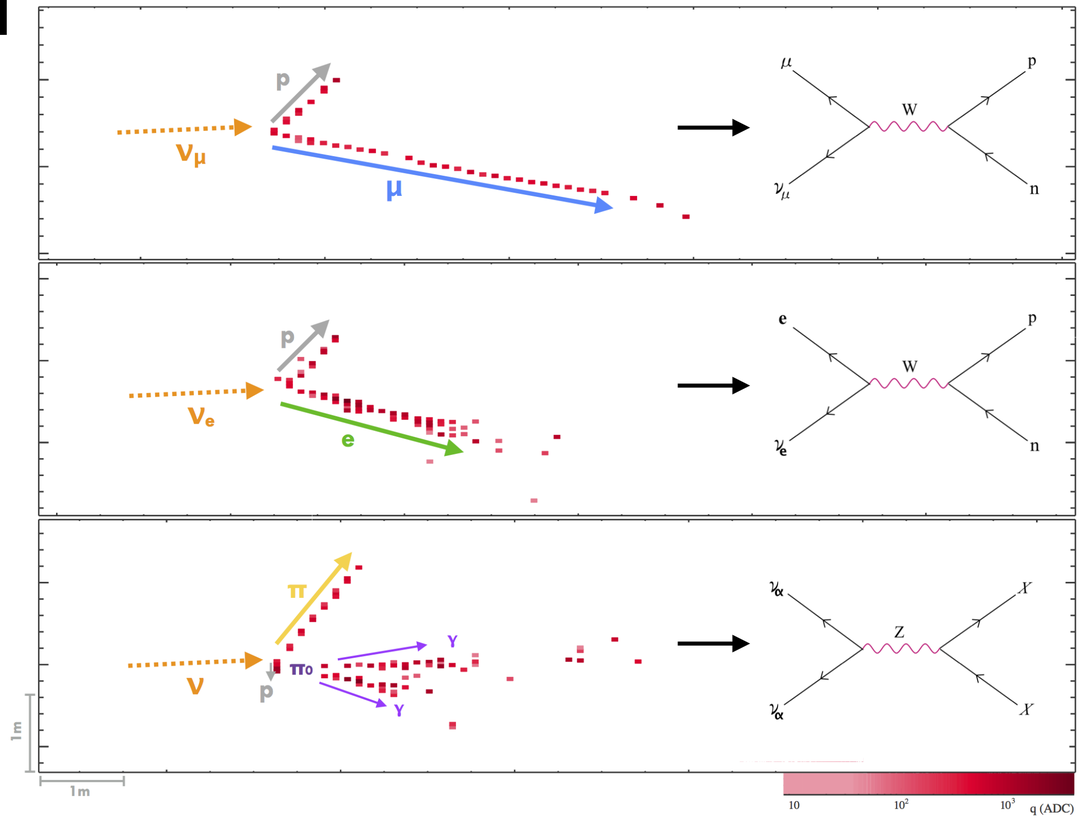


arXiv:1711.03121

Modeling Data

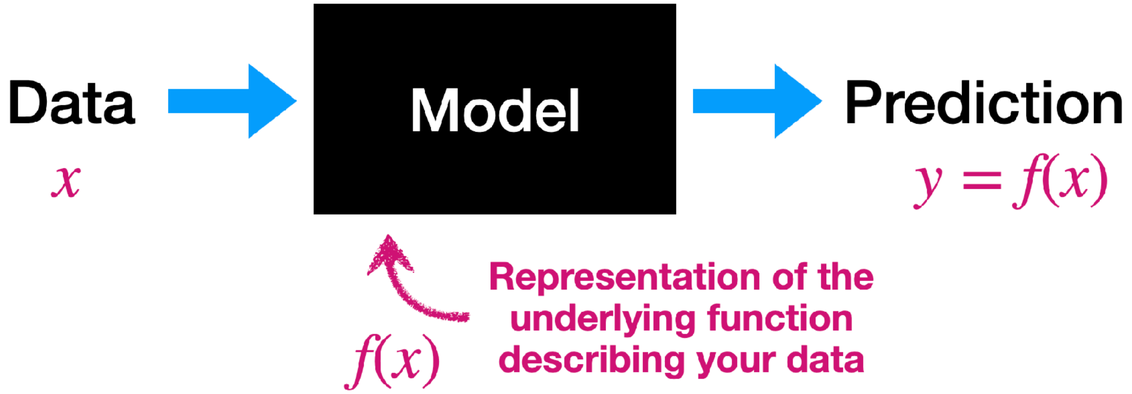
Image data → **Model** → Classify neutrino interactions



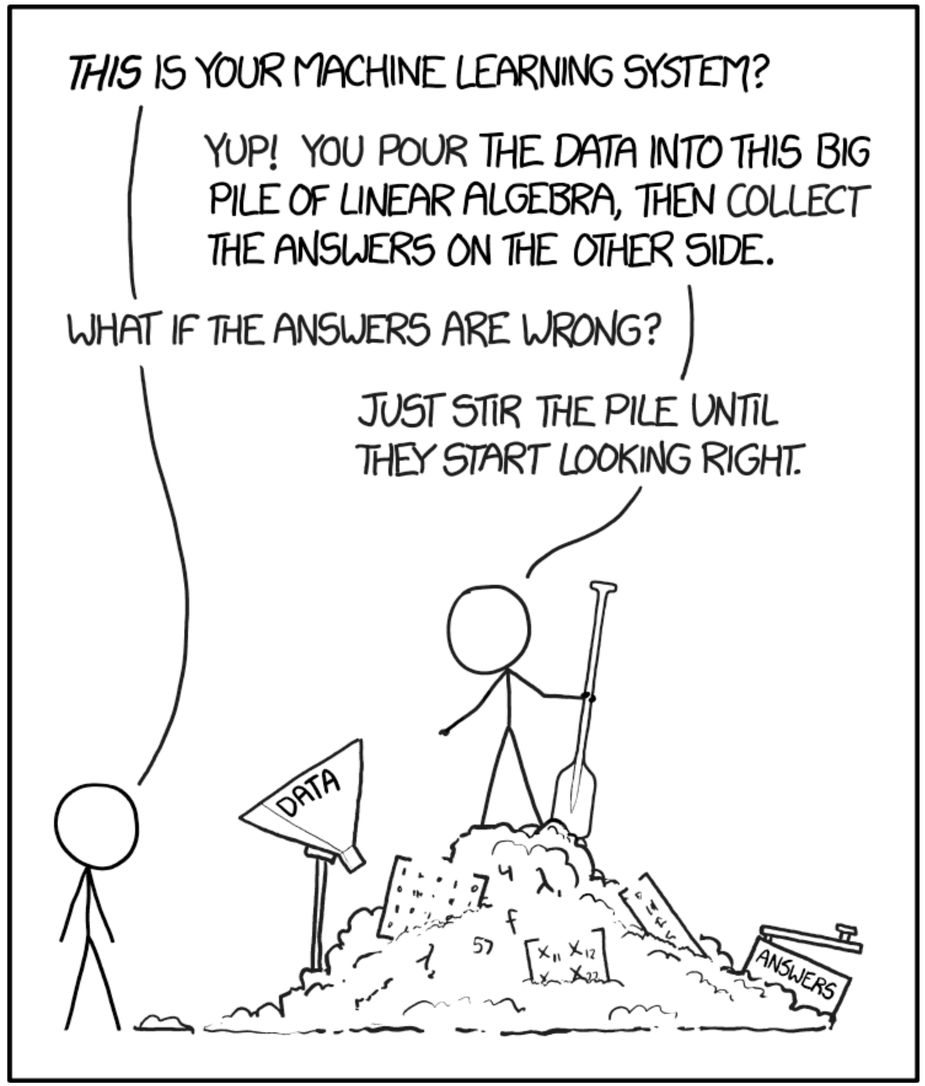


arXiv:1604.01444

What is Machine Learning?



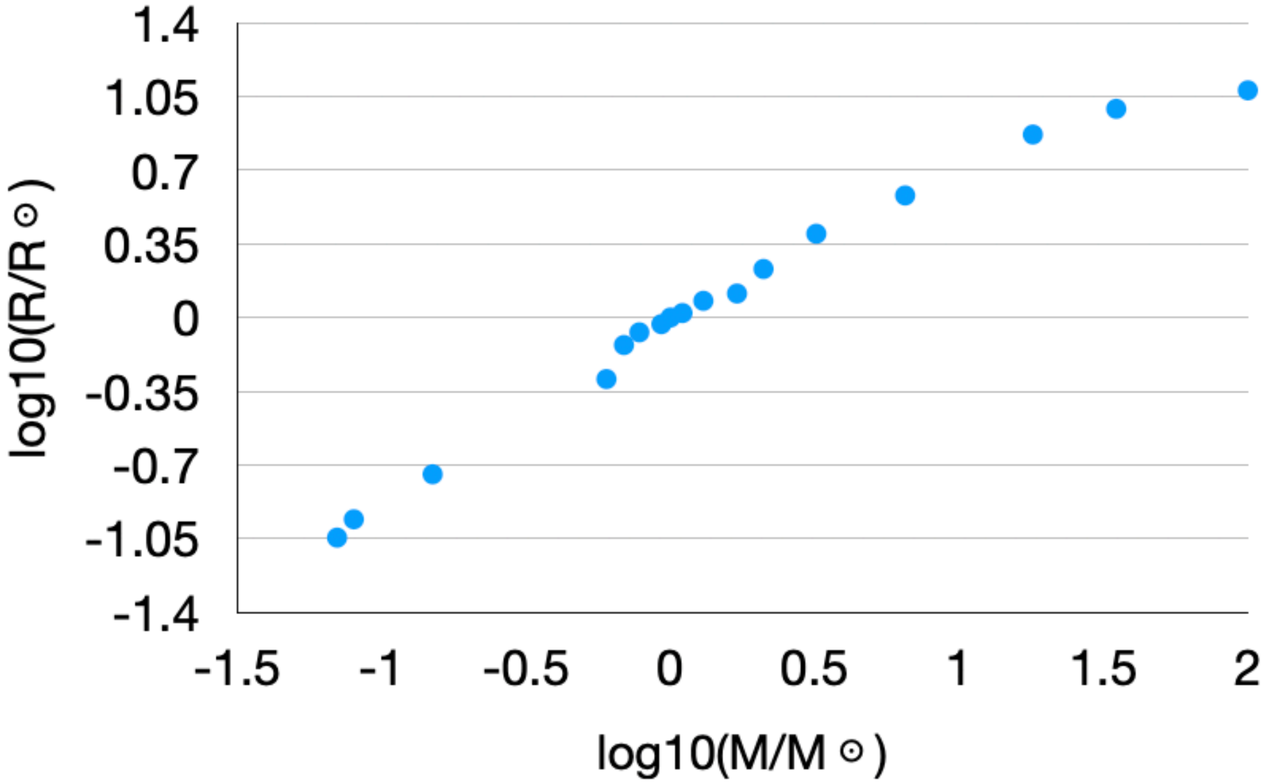
- Computer algorithms (models) that automatically learn patterns and make predictions from data
  - → *underlying function approximators*
  - Linear regression, BDTs, PCA, kNN, NNs...



→*Neural networks are universal function approximators*

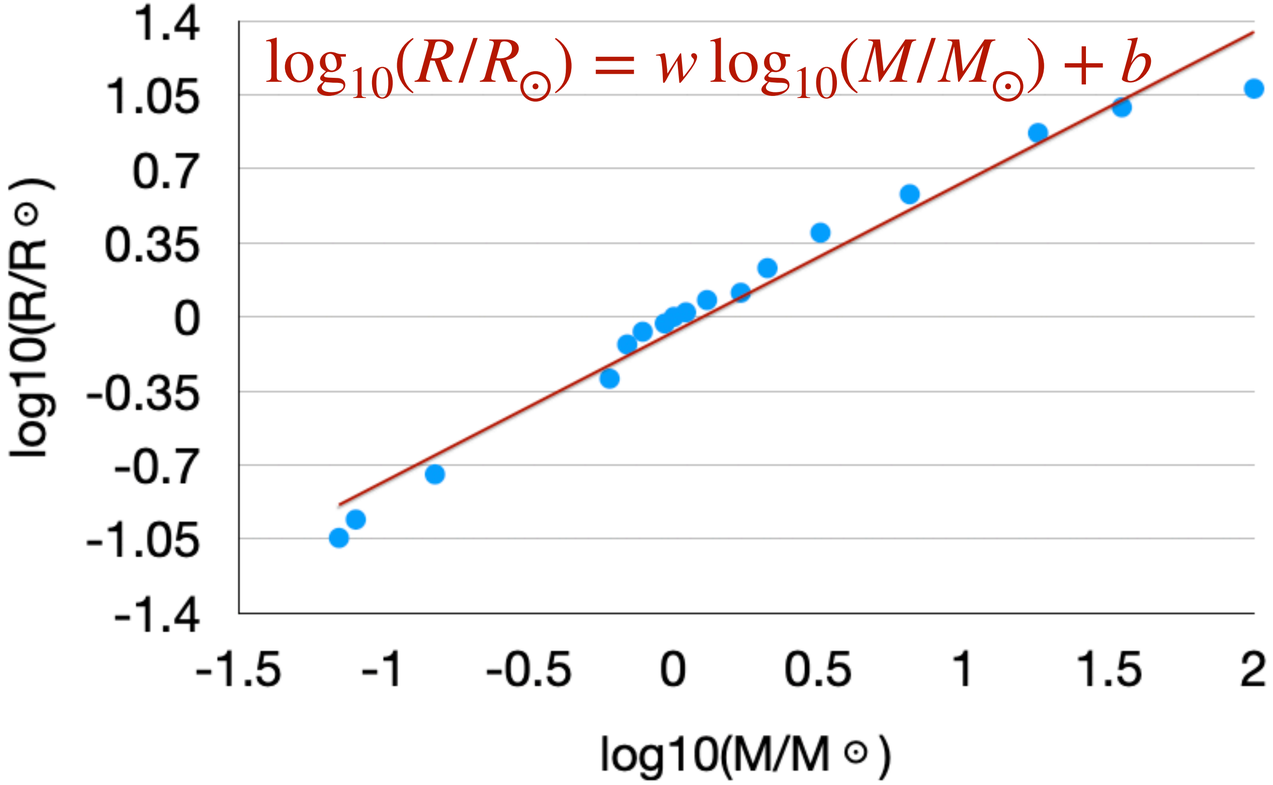
Machine Learning Example: Linear Regression

- Predict stellar radius given stellar mass



Machine Learning Example: Linear Regression

- Predict stellar radius given stellar mass

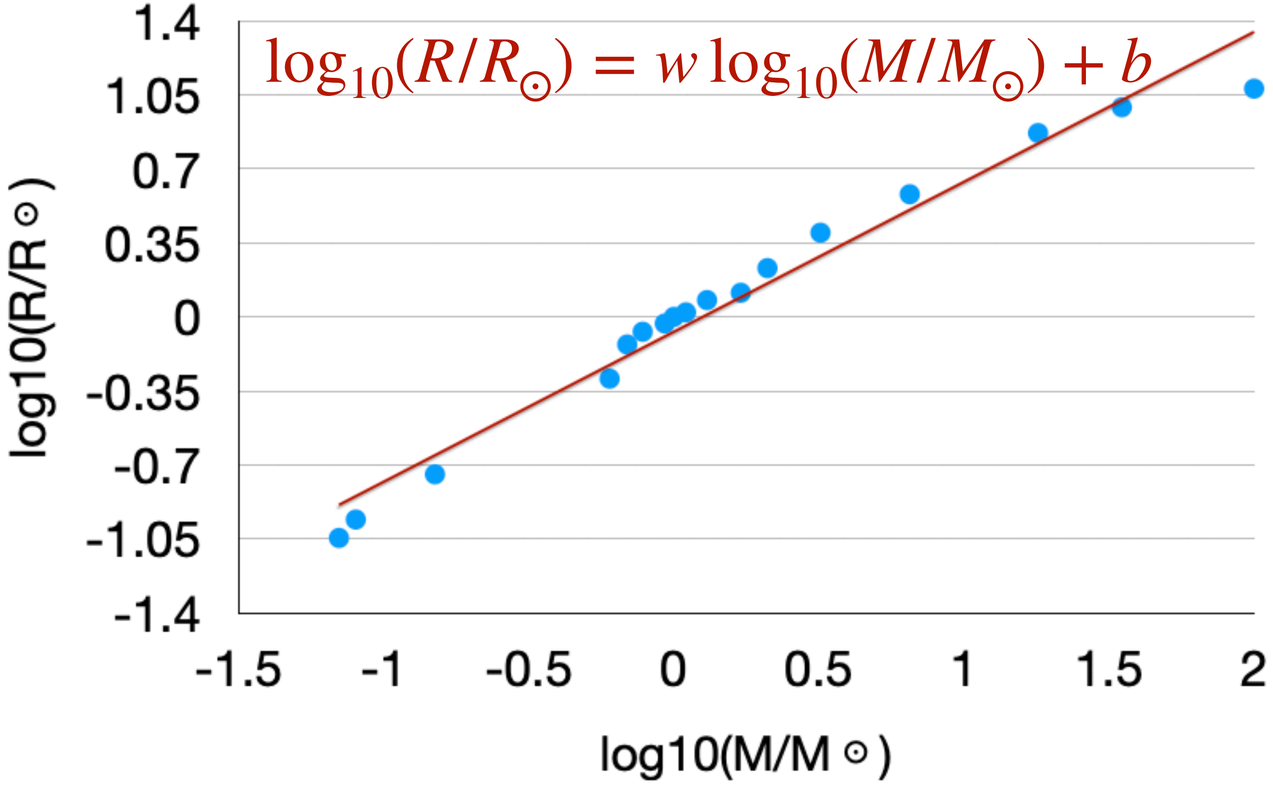


- Let's try to fit a straight line:

$$f(x) = wx + b$$ (linear model)

Machine Learning Example: Linear Regression

- Predict stellar radius given stellar mass



- Let's try to fit a straight line:

$$f(x) = wx + b \text{ (linear model)}$$

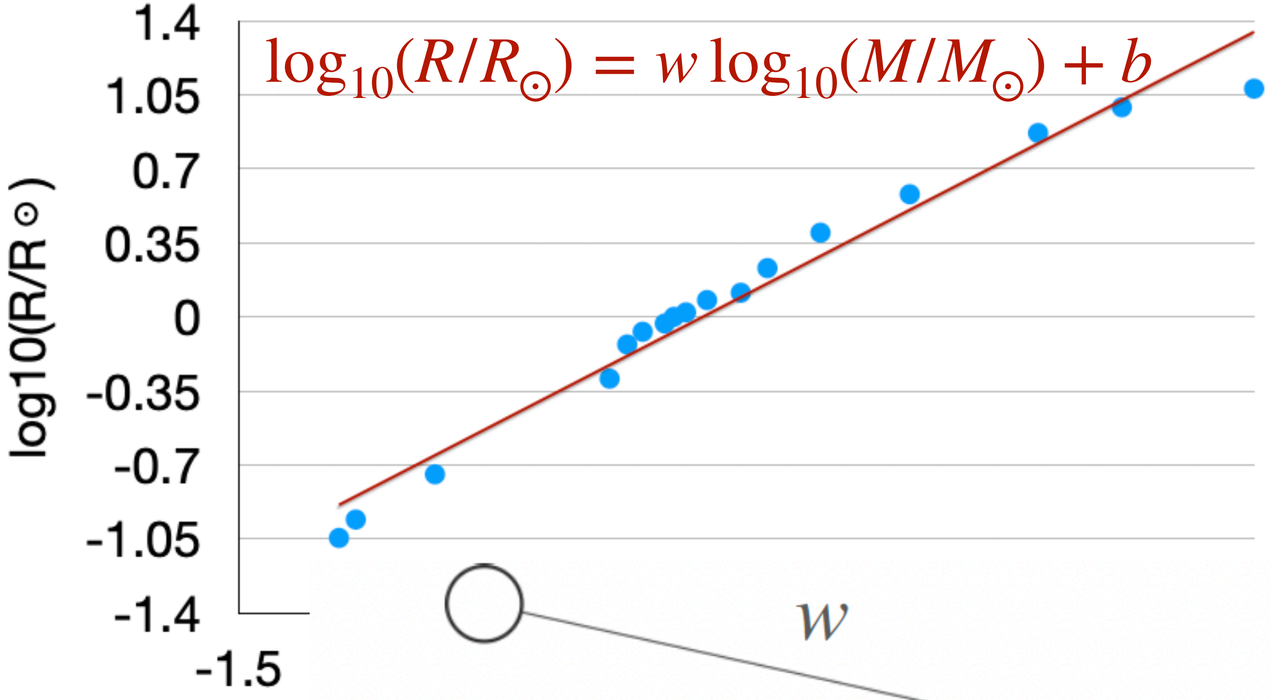
- If x_1 = mass and x_2 = luminosity:

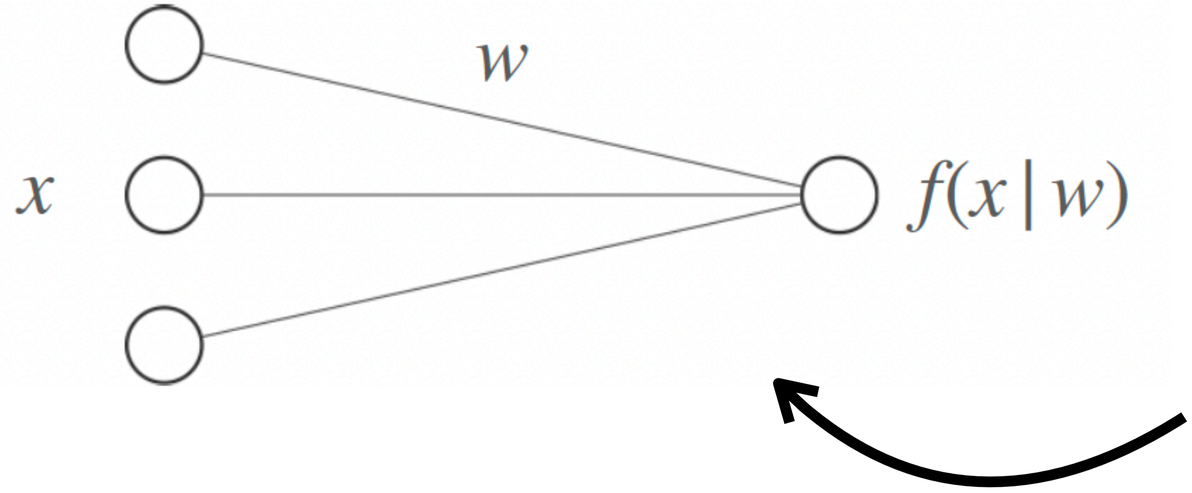
$$= w_1 x_1 + w_2 x_2 + b$$

$$= \Sigma_i w_i \cdot x_i + b$$

Machine Learning Example: Linear Regression

- Predict stellar radius given stellar mass





- Let's try to fit a straight line:

$$f(x) = wx + b \text{ (linear model)}$$

- If x_1 = mass and x_2 = luminosity:

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

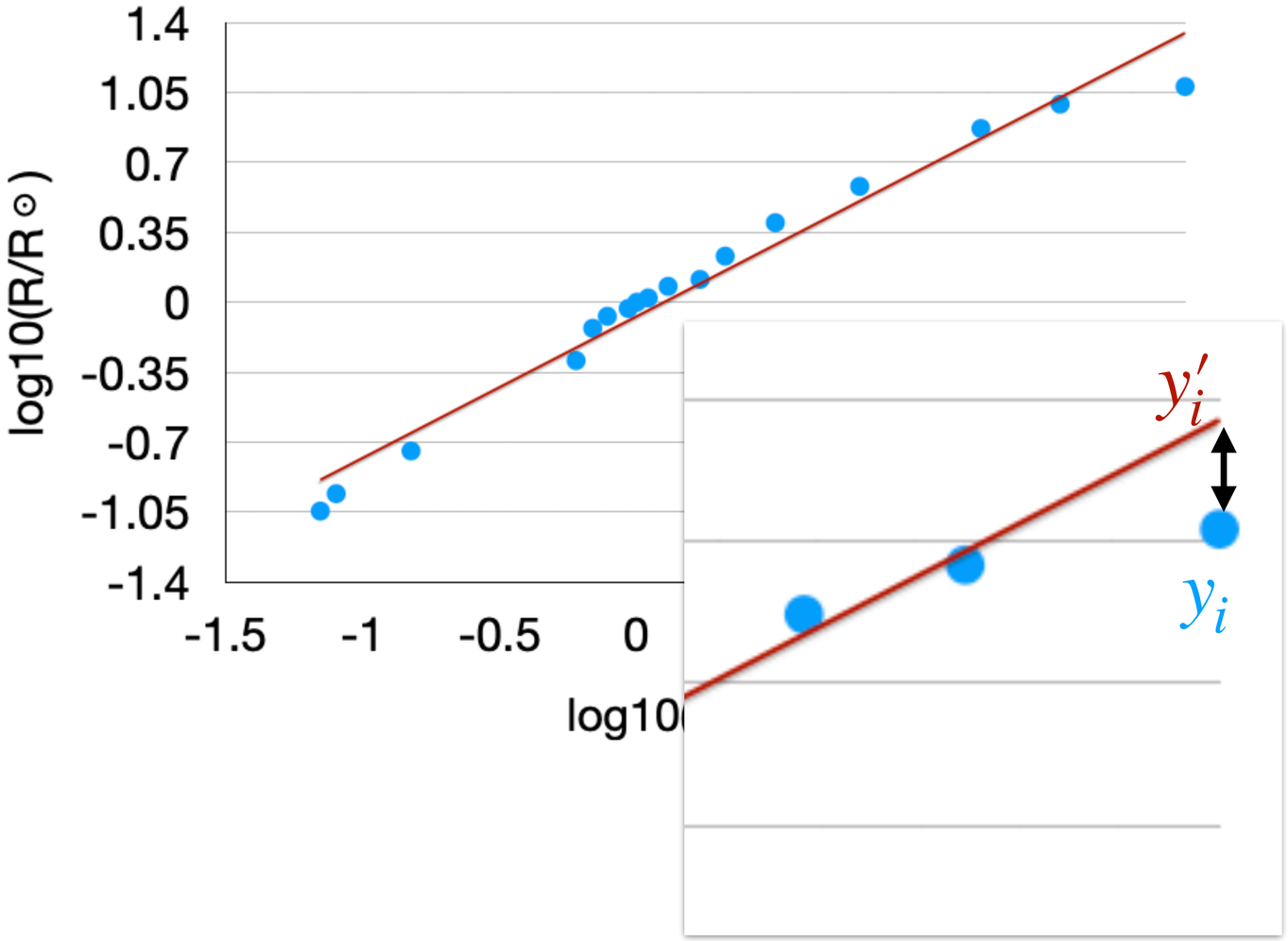
$$= \Sigma_i w_i \cdot x_i + \underbrace{b}_{\sim w_0}$$

Linear model:

$$f(x) = w^T x$$

Machine Learning Example: Linear Regression

- How do we select best fit parameters (weights) w?

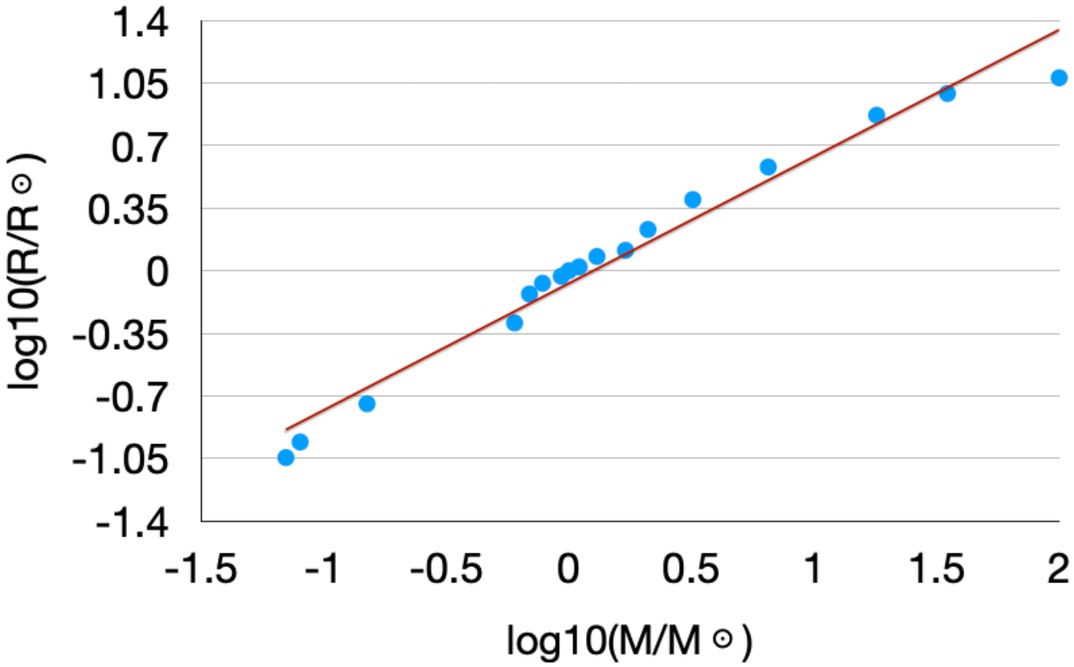


- We want $y_i \approx f(x_i)$
- Squared loss: $L(y, y') = (y - y')^2$

→ minimize distance between predicted y' and true y to find best fit parameters w

→set the derivative/gradient equal to 0 and solve

Building from Linear Models

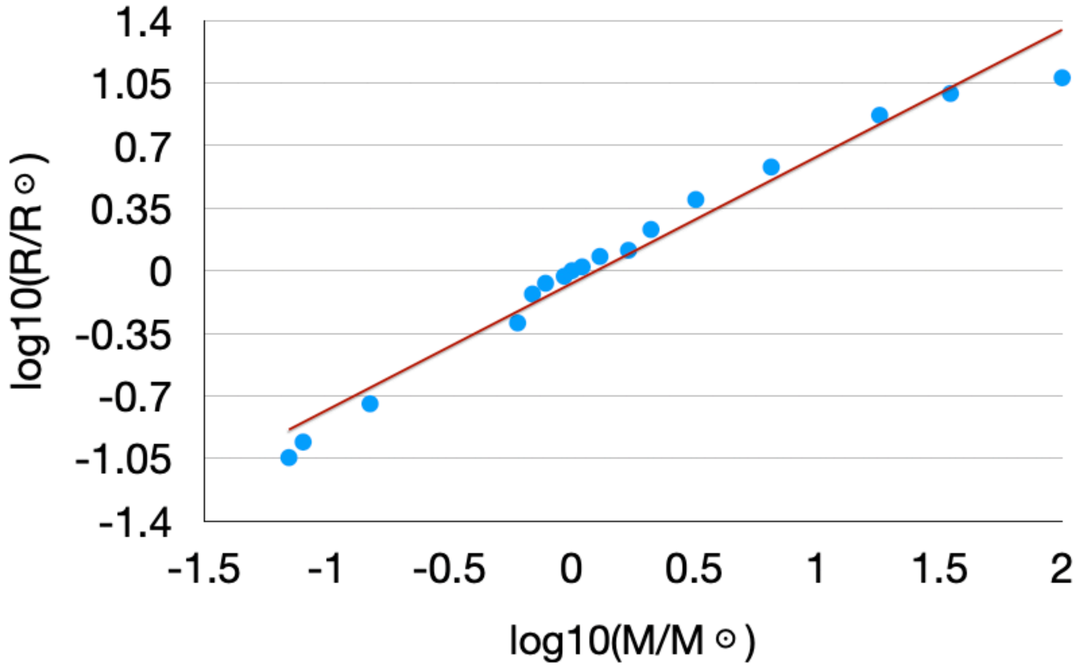


Linear model:

$$f(x) = w^T x$$

- What if we replaced our input vector x with some **features/ embedding** of x: $\phi(x)$?

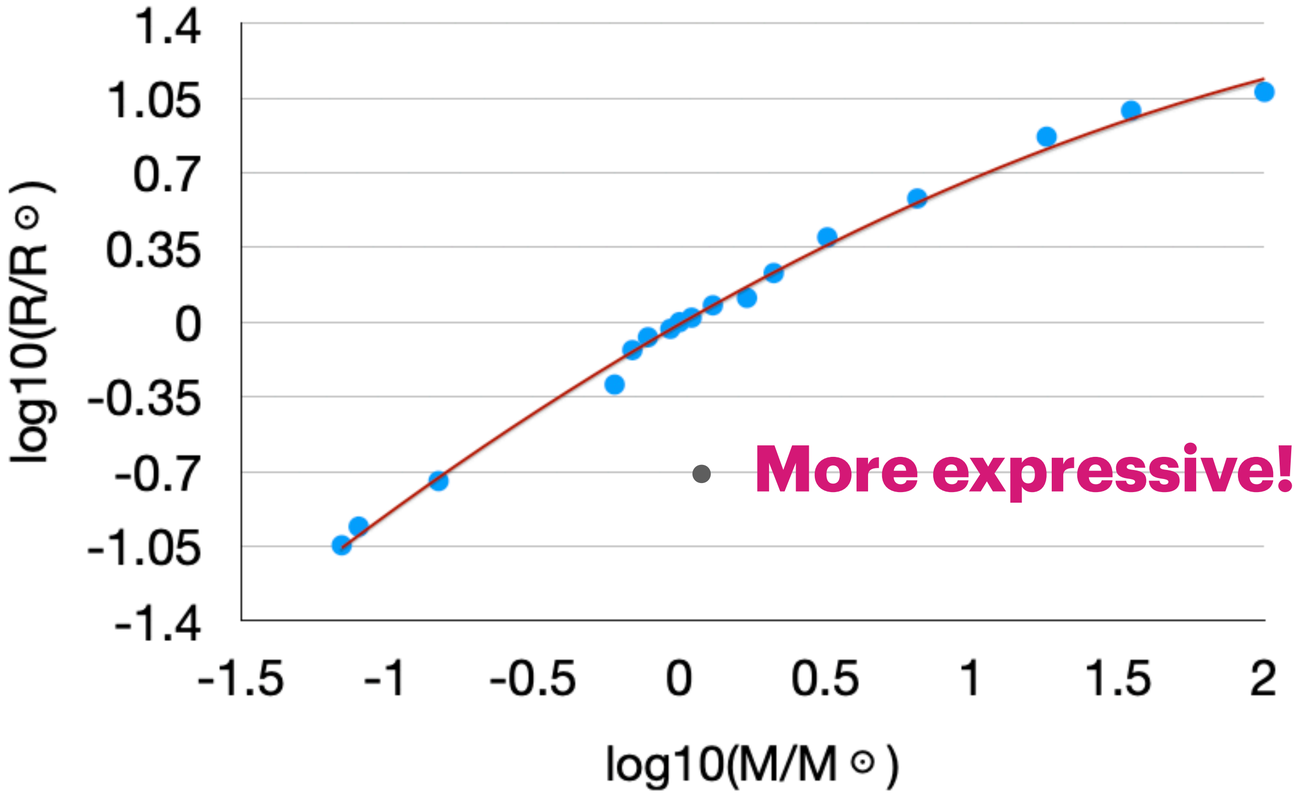
Building from Linear Models



Linear model:

$$f(x) = w^T x$$

- What if we replaced our input vector x with some **features/ embedding** of x: $\phi(x)$?

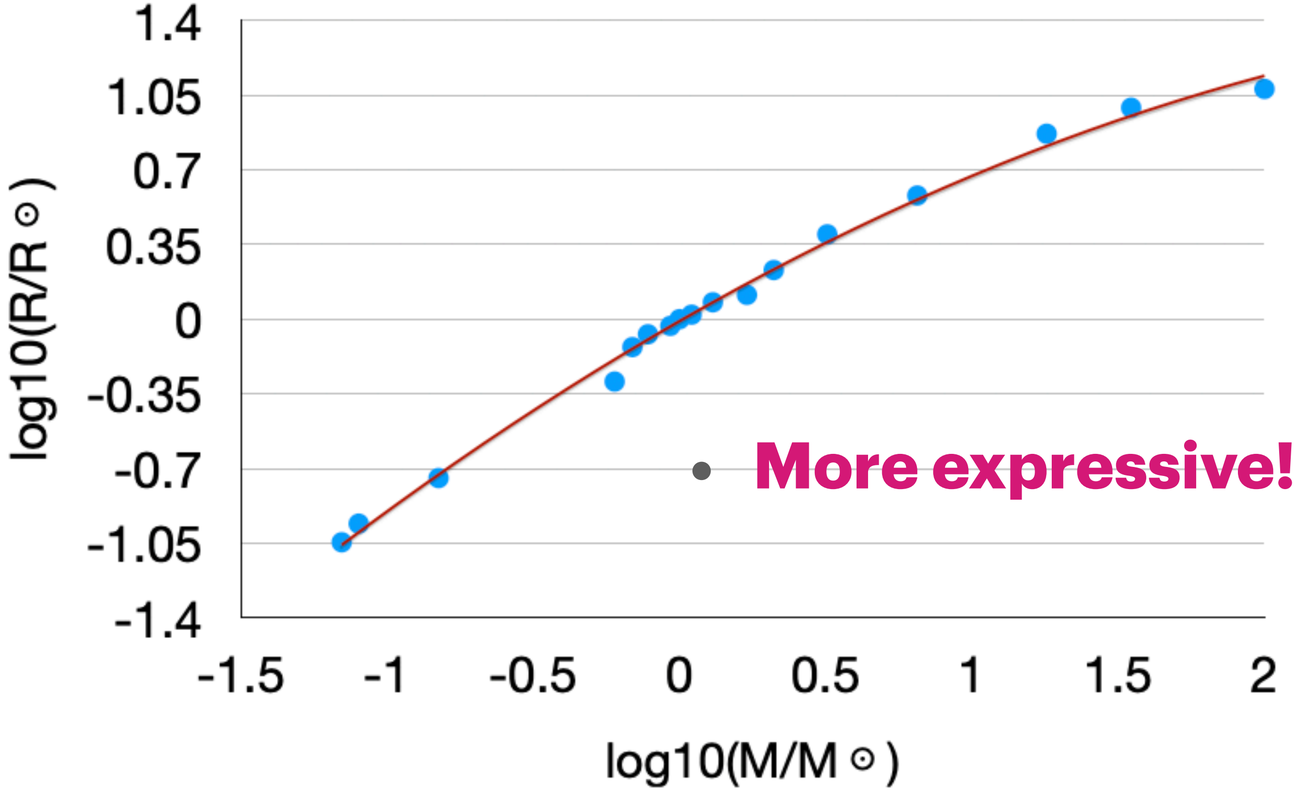


- Ex: choose embedding $\phi(x) = (x^2, x, 1)$
  - $\rightarrow f(x) = w_1 x^2 + w_2 x + b$

$$f(x) = w^T \phi(x)$$

Building from Linear Models

- Nonlinear transformation of x:
  - $x \rightarrow \phi(x)$
  - Still linear in w
- $\rightarrow$Linear models on top of good features can yield excellent results
- Neural networks = automatic featurizers with linear models as their basic building blocks



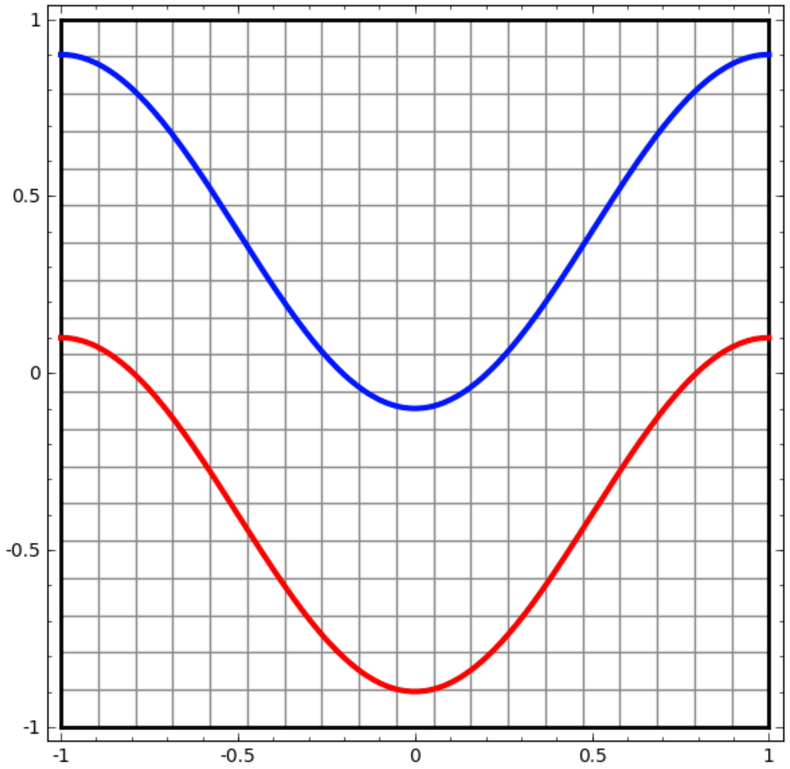
- Ex: choose embedding $\phi(x) = (x^2, x, 1)$
  - $\rightarrow f(x) = w_1 x^2 + w_2 x + b$

$$f(x) = w^T \phi(x)$$

Visualizing Neural Networks

colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Data

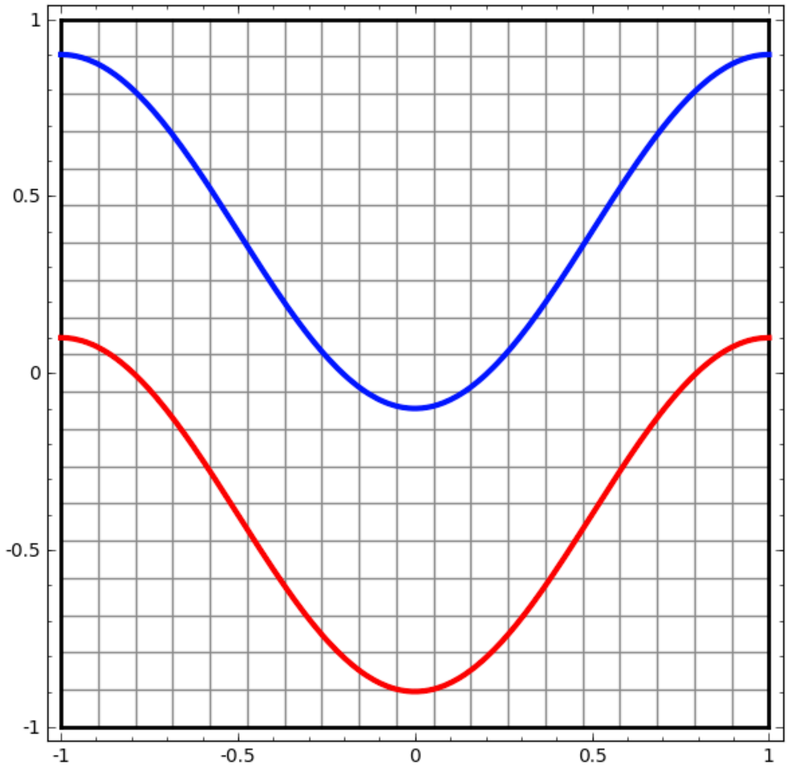


Goal: Separate red and blue classes

Visualizing Neural Networks

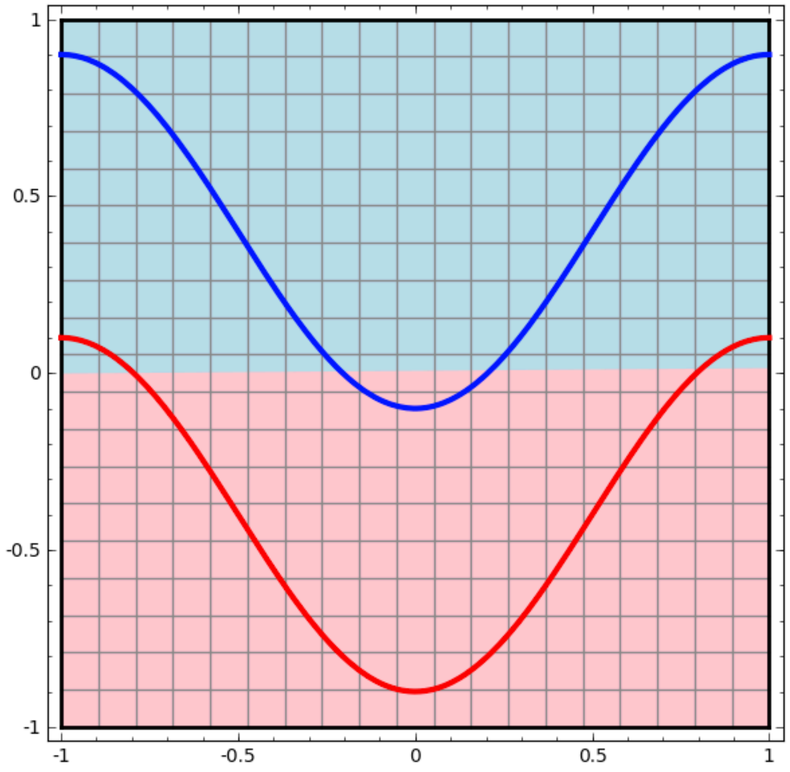
colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Data



Goal: Separate red and blue classes

Linear Classifier

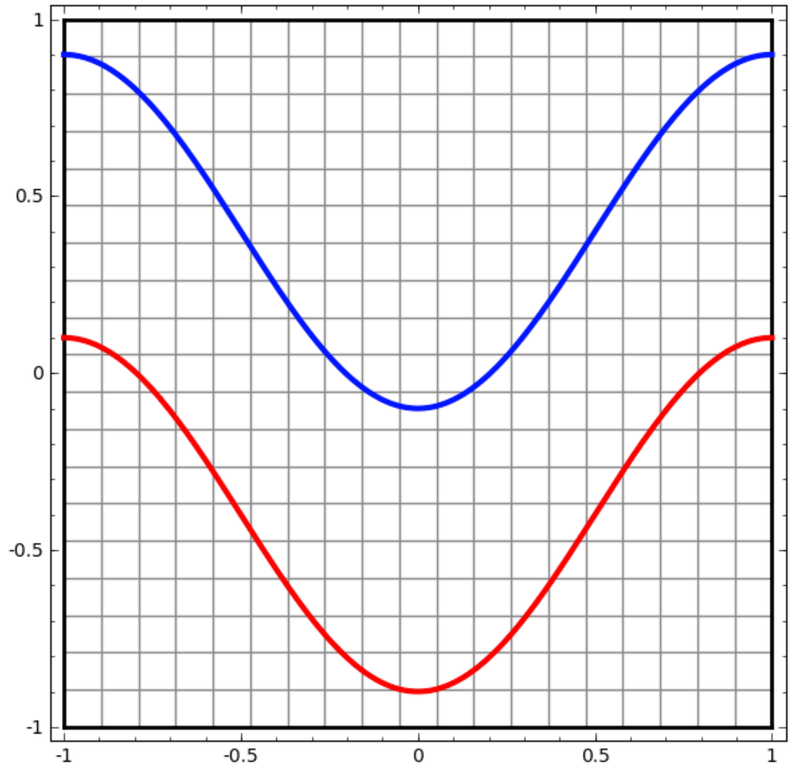


$$f(x) = w^\intercal x$$

Visualizing Neural Networks

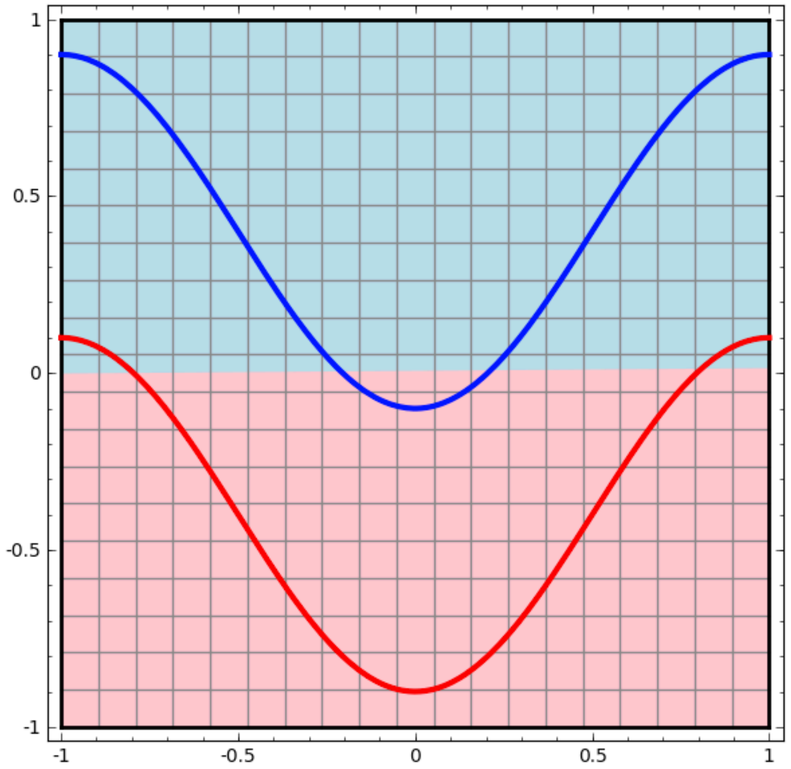
colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Data



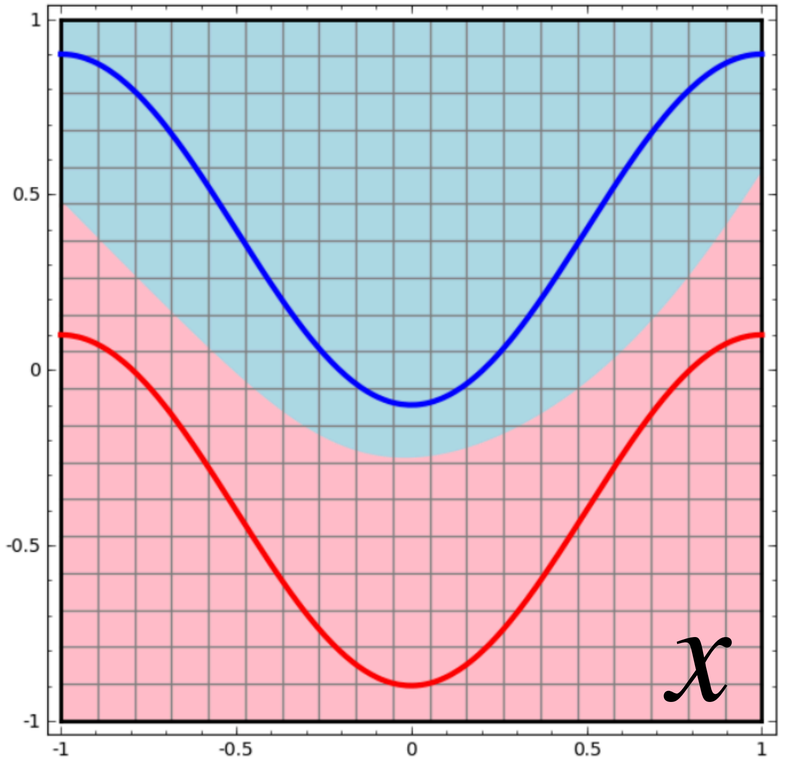
Goal: Separate red and blue classes

Linear Classifier



$$f(x) = w^\intercal x$$

Simple Neural Network



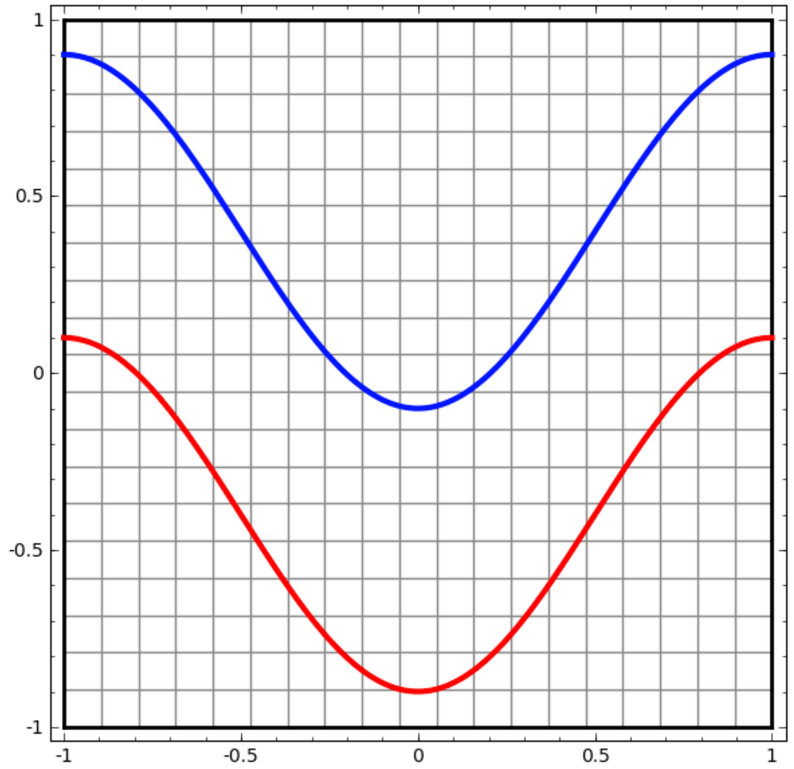
$$f(x) = w_2^\intercal \phi(w_1^\intercal x)$$

Raw inputs

Visualizing Neural Networks

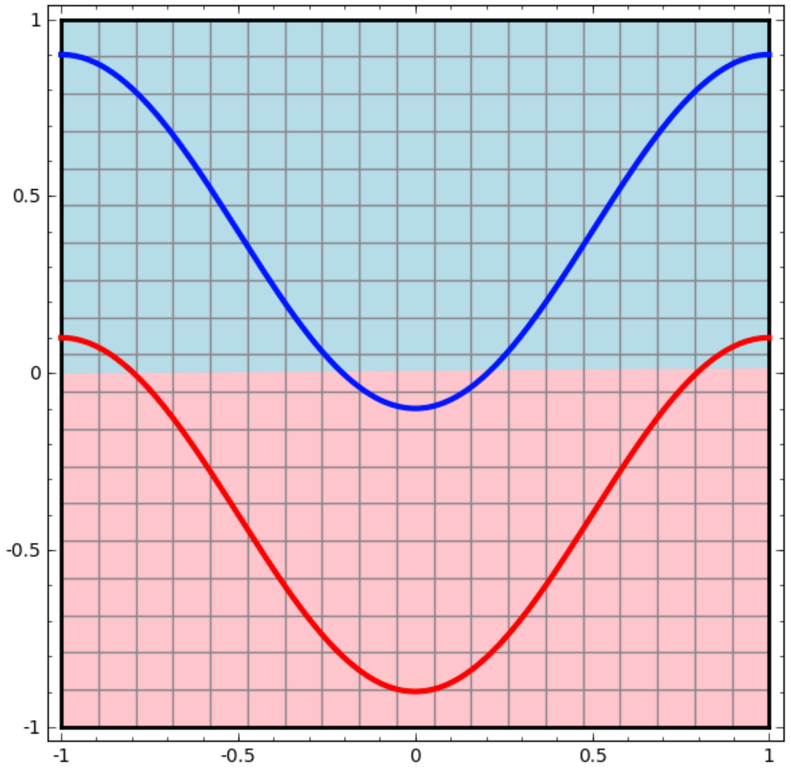
colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Data



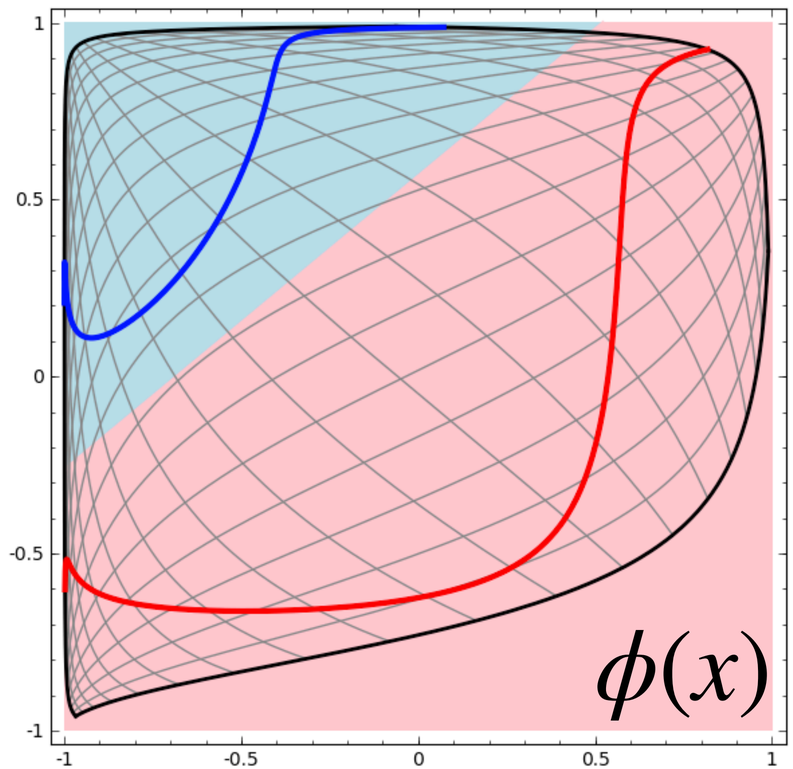
Goal: Separate red and blue classes

Linear Classifier



$$f(x) = w^\intercal x$$

Simple Neural Network



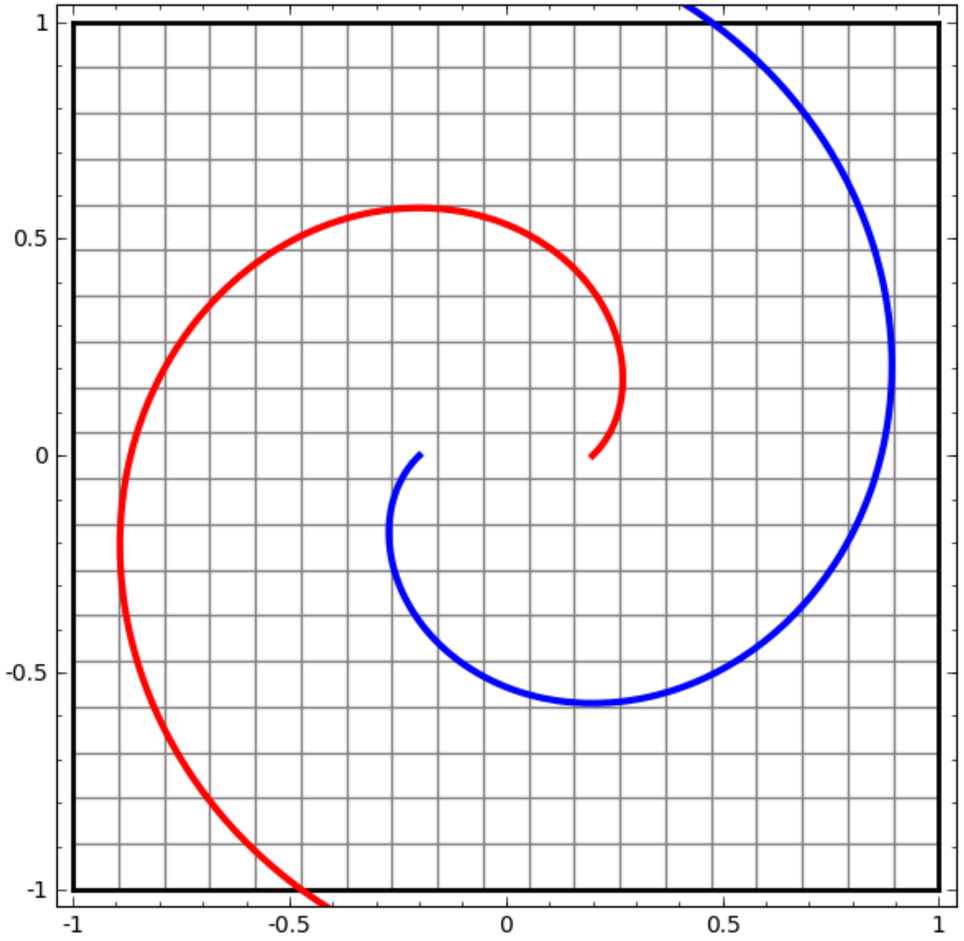
$$f(x) = w_2^\intercal \phi(w_1^\intercal x)$$

***Transformed* inputs**

Visualizing Neural Networks

colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Transformations learned during training:



→Learns linear separation of transformed inputs

Understanding Activation Functions

- →Linear models on top of good features can yield excellent results!
- Neural networks = automatic featurizers with linear models as their basic building blocks
- ***What does this really mean?***

$$f(x) = w^T \phi(x)$$

Understanding Activation Functions

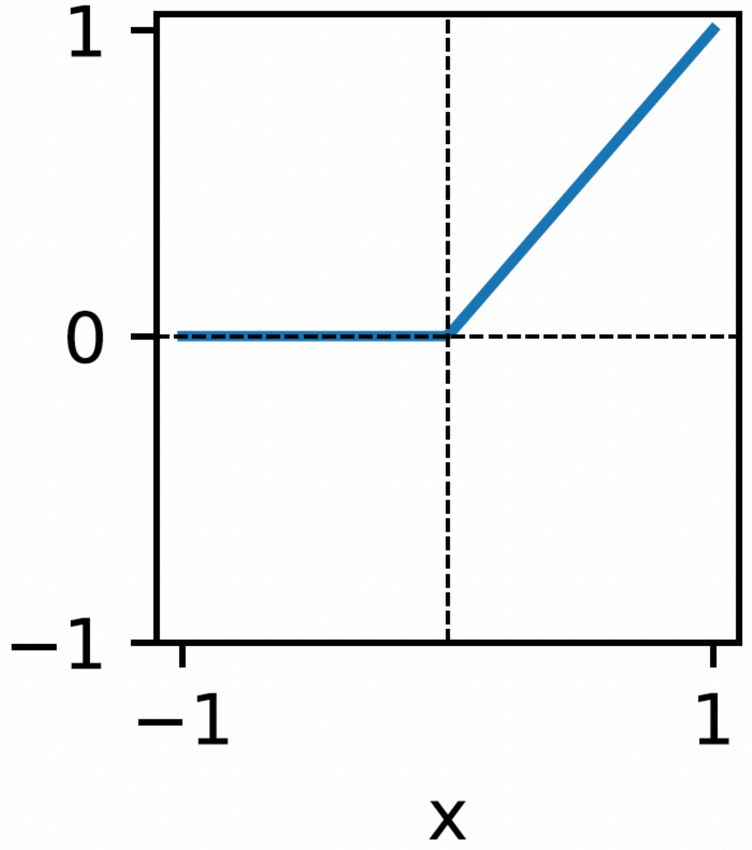
- →Linear models on top of good features can yield excellent results!
- Neural networks = automatic featurizers with linear models as their basic building blocks
- ***What does this really mean?***

$$f(x) = w^T \phi(x)$$

- Common choice: ReLU ("Rectified Linear Unit")
  - Nonlinear
  - Easy to compute
  - Simple derivative

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

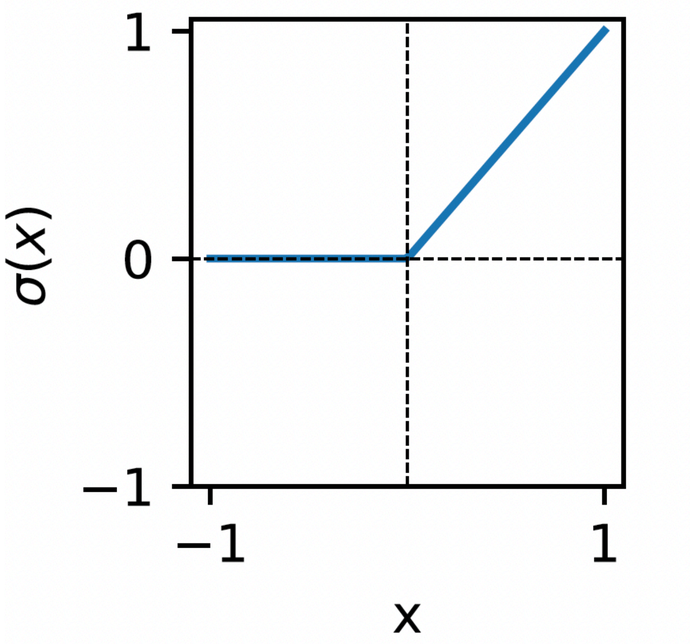
$$\frac{\partial}{\partial x}\sigma(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



More on this in next lecture!

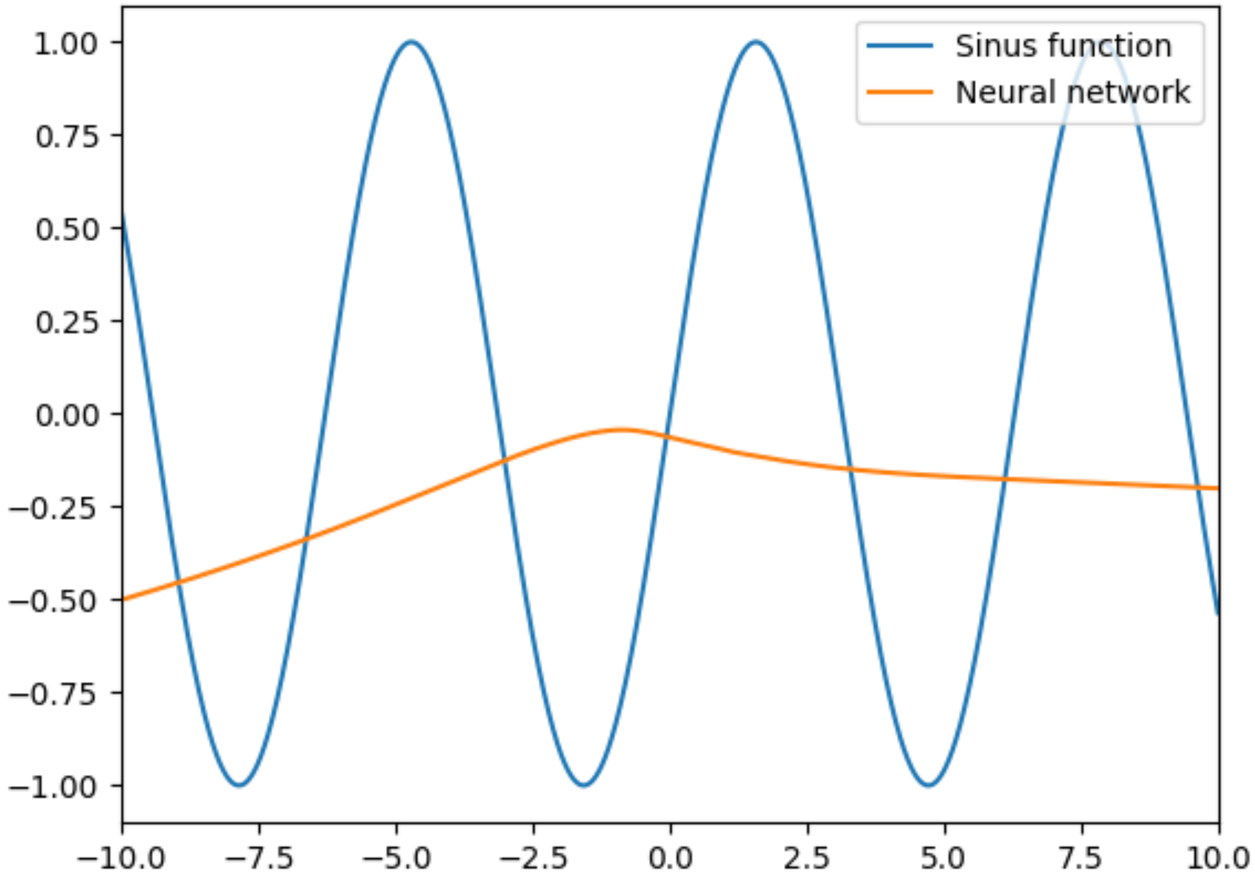
Understanding Activation Functions

- Can combine weighted ReLUs to approximate target function!



Understanding Activation Functions

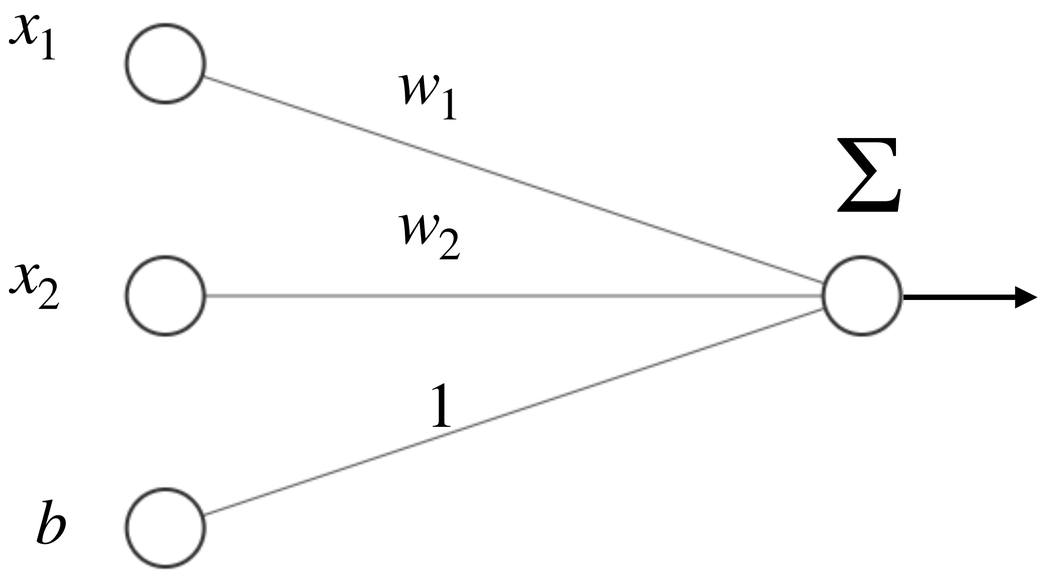
- NN ~ a stacking and combining of weighted linear combinations of inputs passed through nonlinear activation functions
- During training you learn the optimal weights



Source

Building Neural Networks

- Starting from the linear model…

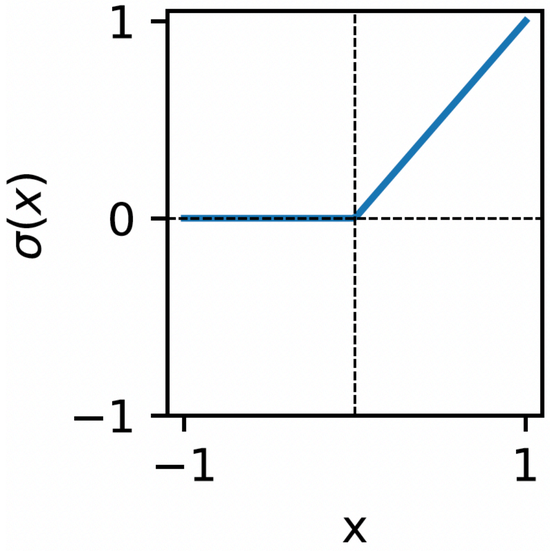


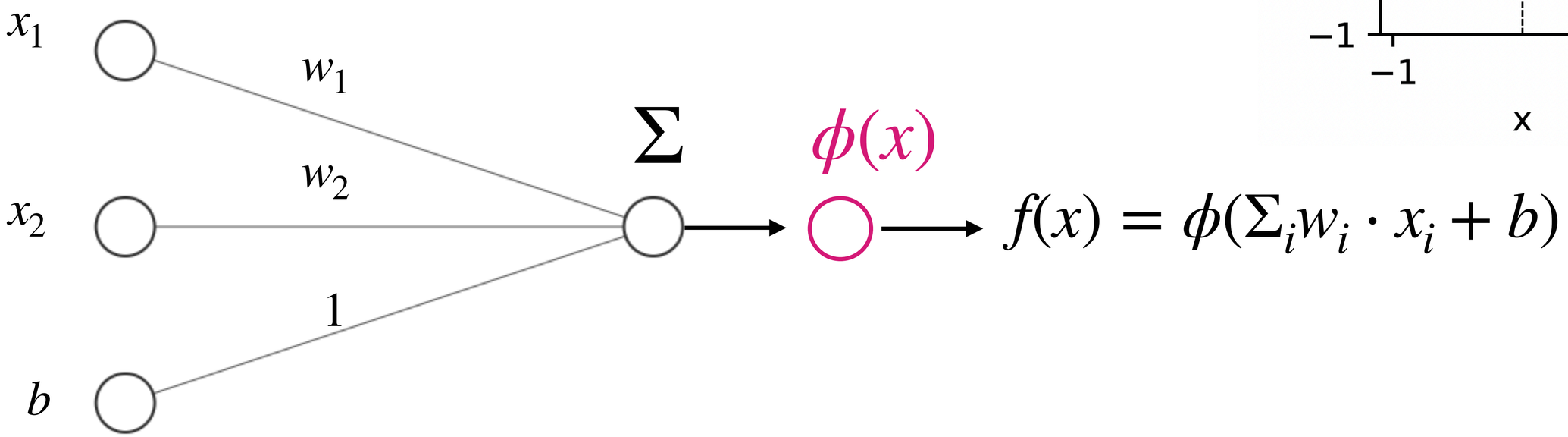
$$f(x) = \Sigma_i w_i \cdot x_i + b$$
$$= w_1 x_1 + w_2 x_2 + \underbrace{b}_{\sim w_0}$$

$$\boxed{f(x) = w^T x}$$

Building Neural Networks

- Add a **nonlinear transformation** $\phi(x)$



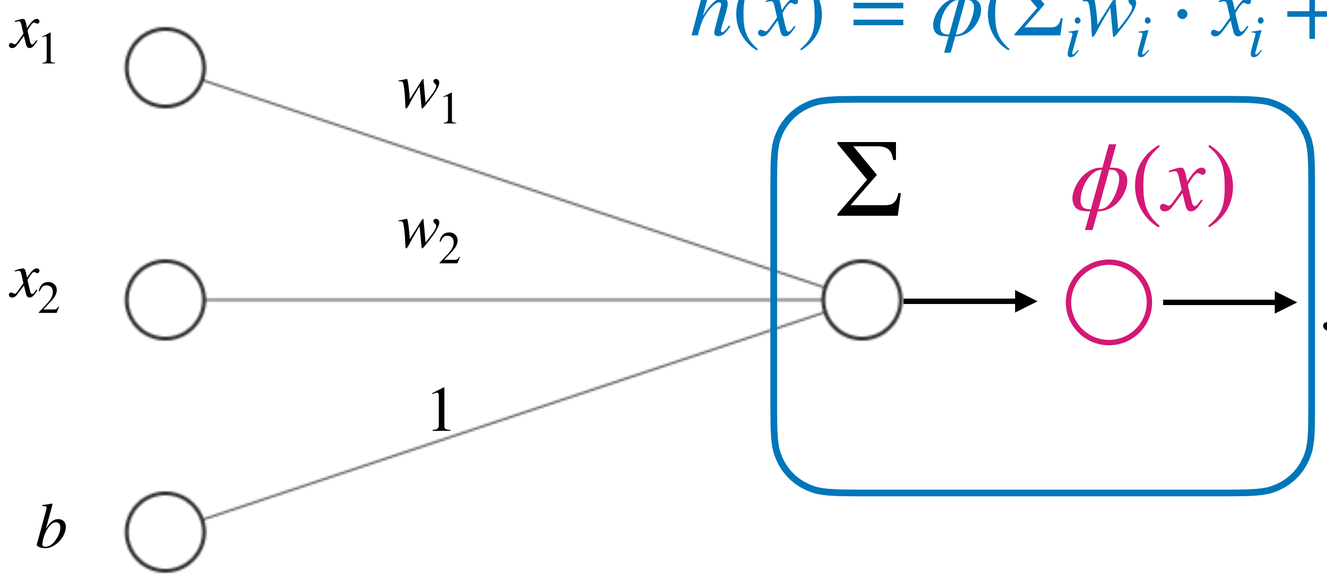


$$f(x) = w^T \phi(x)$$

Building Neural Networks

- Define a **node** $h(x)$

$$h(x) = \phi(\Sigma_i w_i \cdot x_i + b)$$

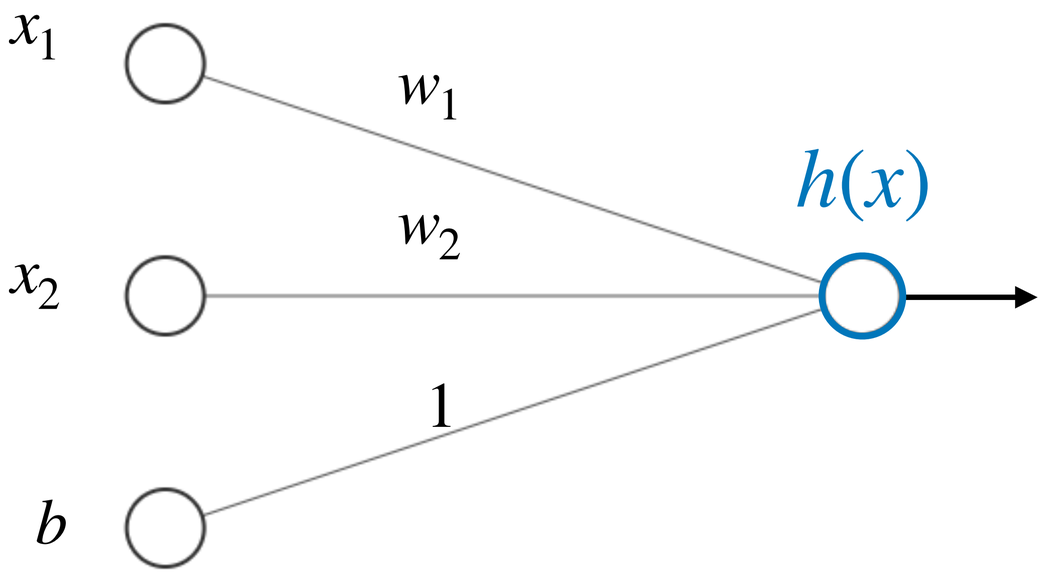


$$f(x) = \phi(\Sigma_i w_i \cdot x_i + b)$$
$$= h(x)$$

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

Building Neural Networks

- Define a **node** $h(x)$



$$h(x) = \phi(\Sigma_i w_i \cdot x_i + b)$$

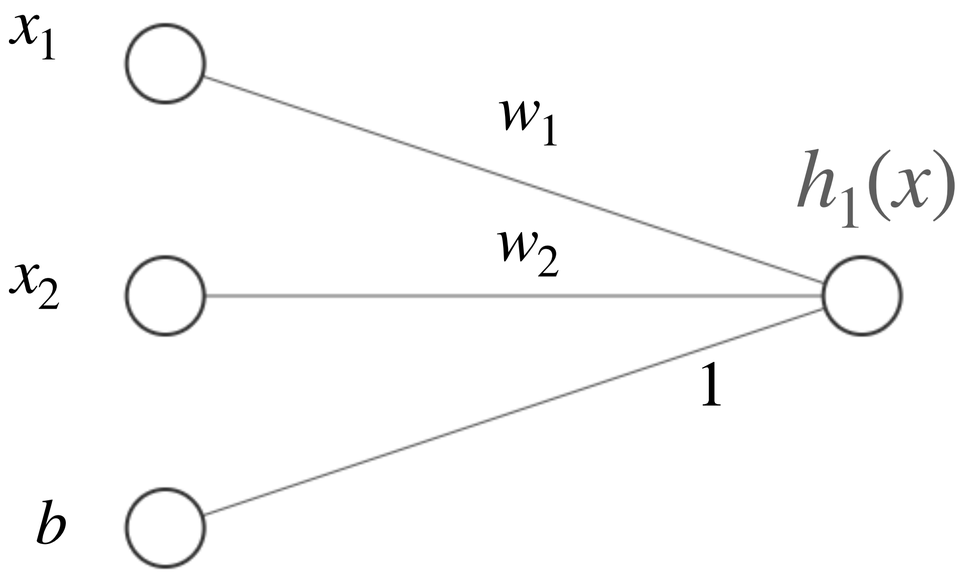
$$f(x) = h(x)$$

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

Building Neural Networks

- Let's add complexity: additional transformations for even more expressiveness!

$$h_1(x) = \phi(\Sigma_i w_i \cdot x_i + b)$$



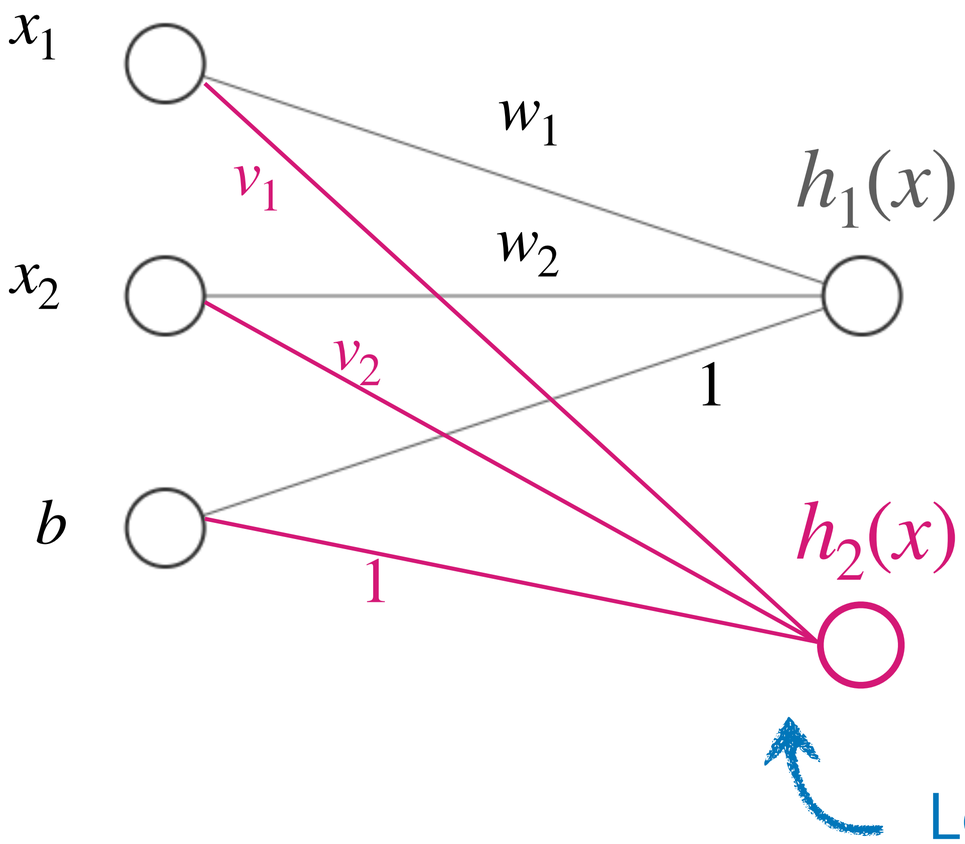
Let's add another transformation...

Building Neural Networks

- Let's add complexity: additional transformations for even more expressiveness!

$$h_1(x) = \phi(\Sigma_i w_i \cdot x_i + b)$$

$$h_2(x) = \phi(\Sigma_i v_i \cdot x_i + b)$$



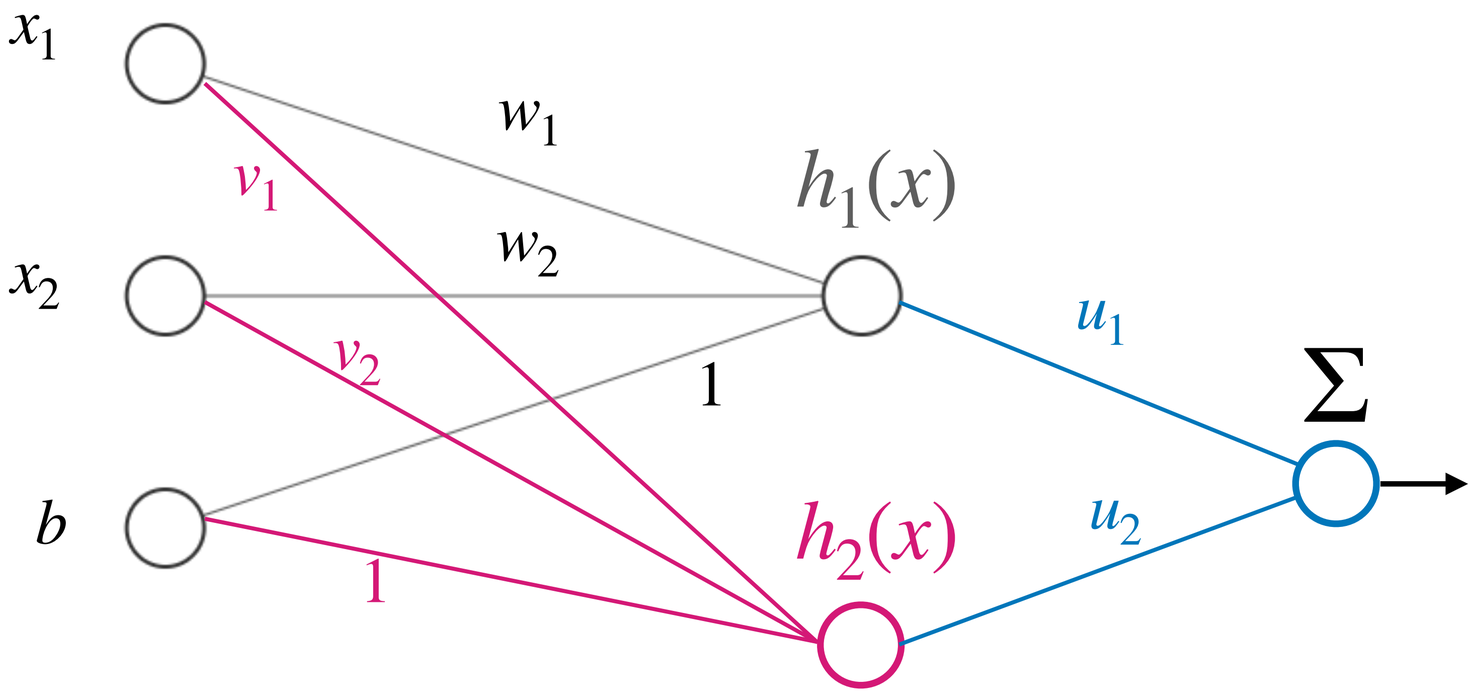
Let's combine these into a single output...

Building Neural Networks

- Let's add complexity: additional transformations for even more expressiveness!

$$h_1(x) = \phi(\Sigma_i w_i \cdot x_i + b)$$

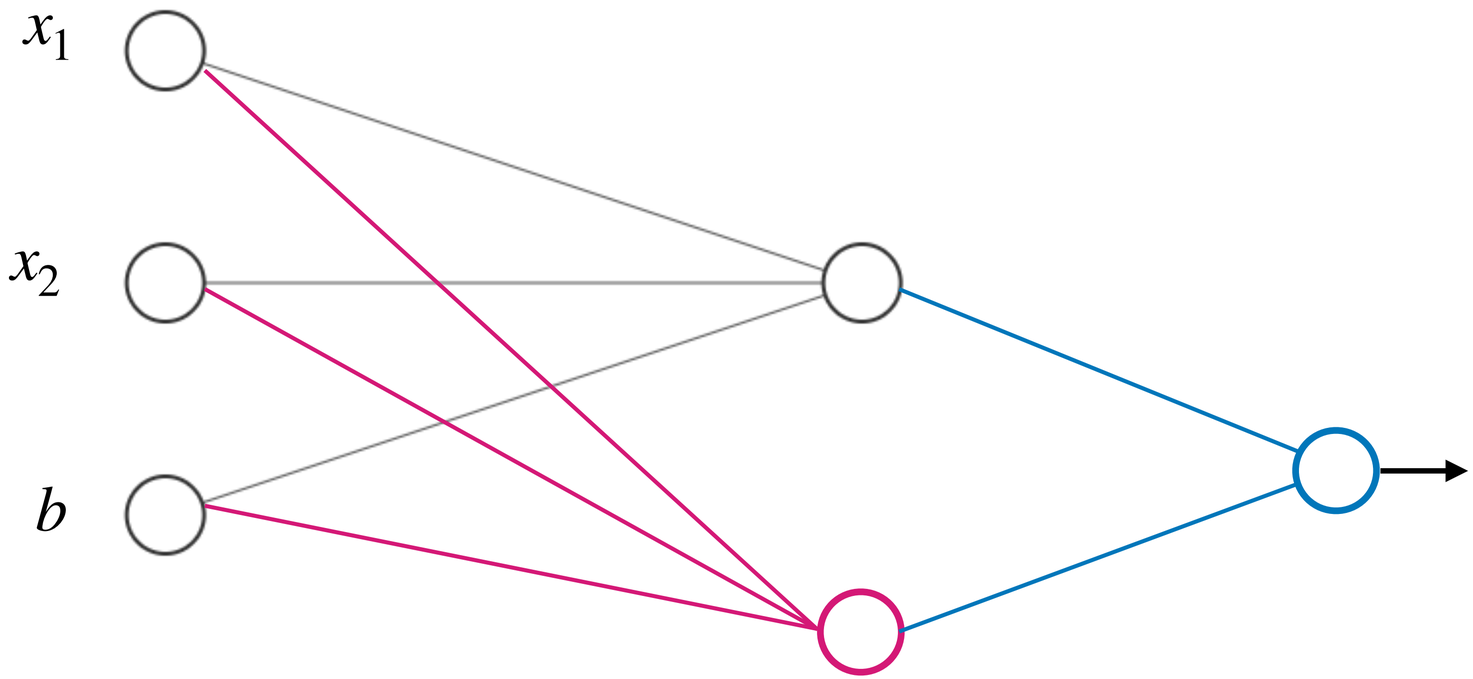
$$h_2(x) = \phi(\Sigma_i v_i \cdot x_i + b)$$



$$f(x) = u_1^T h_1(x) + u_2^T h_2(x)$$

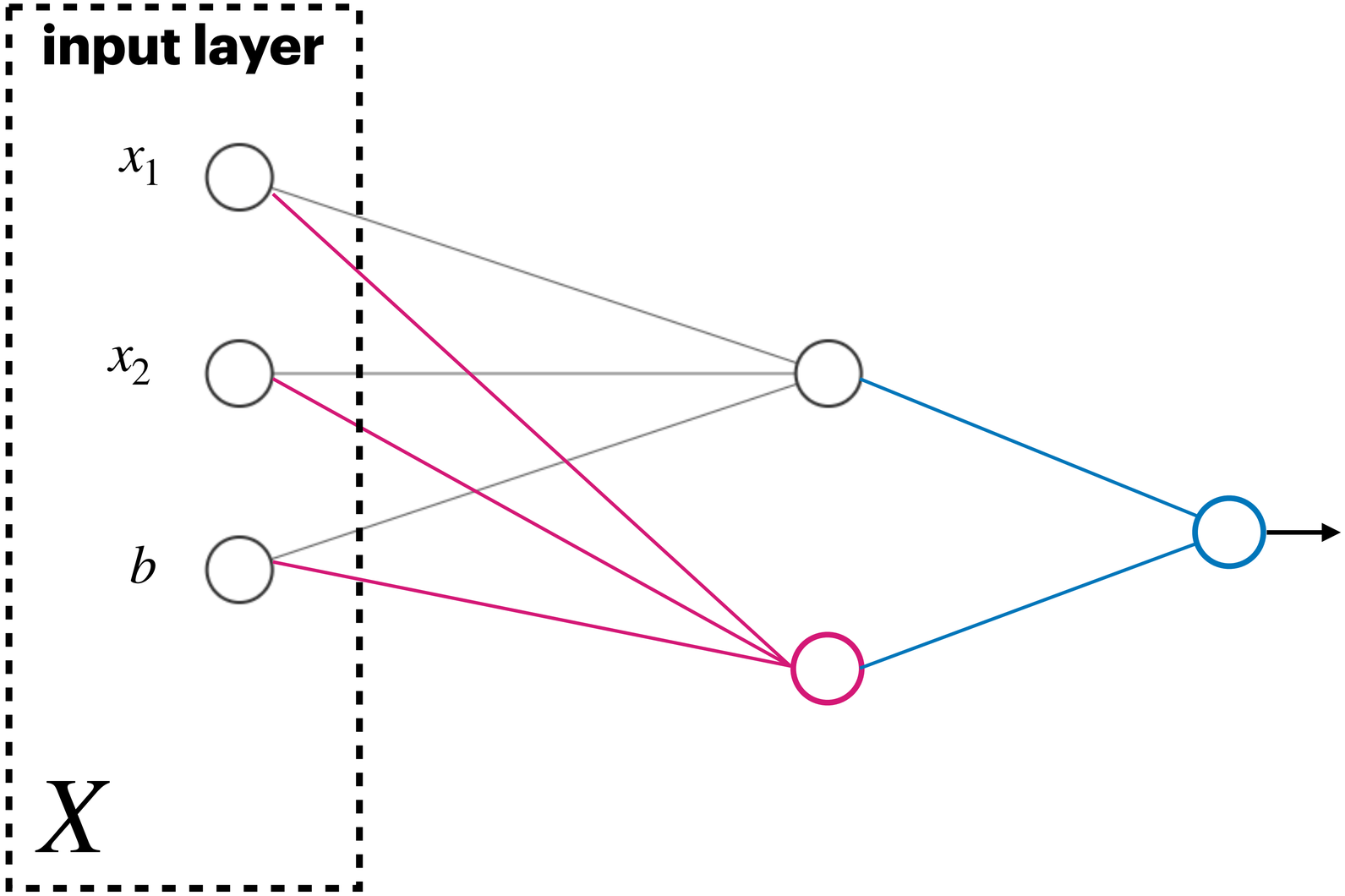
Building Neural Networks

- Linear algebra makes things simpler: Replace everything with matrices!



Building Neural Networks

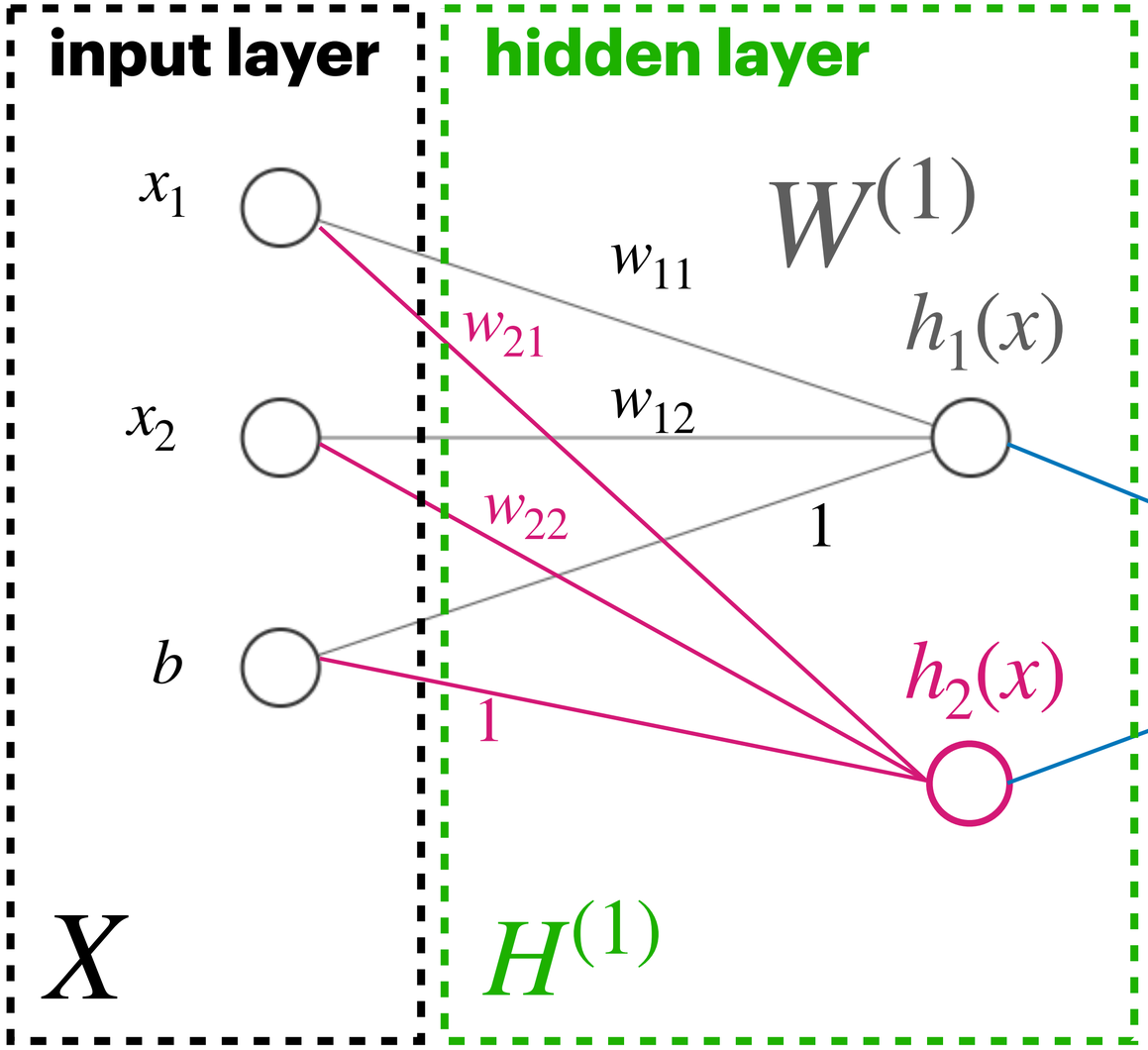
- Linear algebra makes things simpler: Replace everything with matrices!



$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$$

Building Neural Networks

- Linear algebra makes things simpler: Replace everything with matrices!



$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1D} \\ w_{21} & w_{22} & \cdots & w_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1} & w_{K2} & \cdots & w_{KD} \end{pmatrix}$$

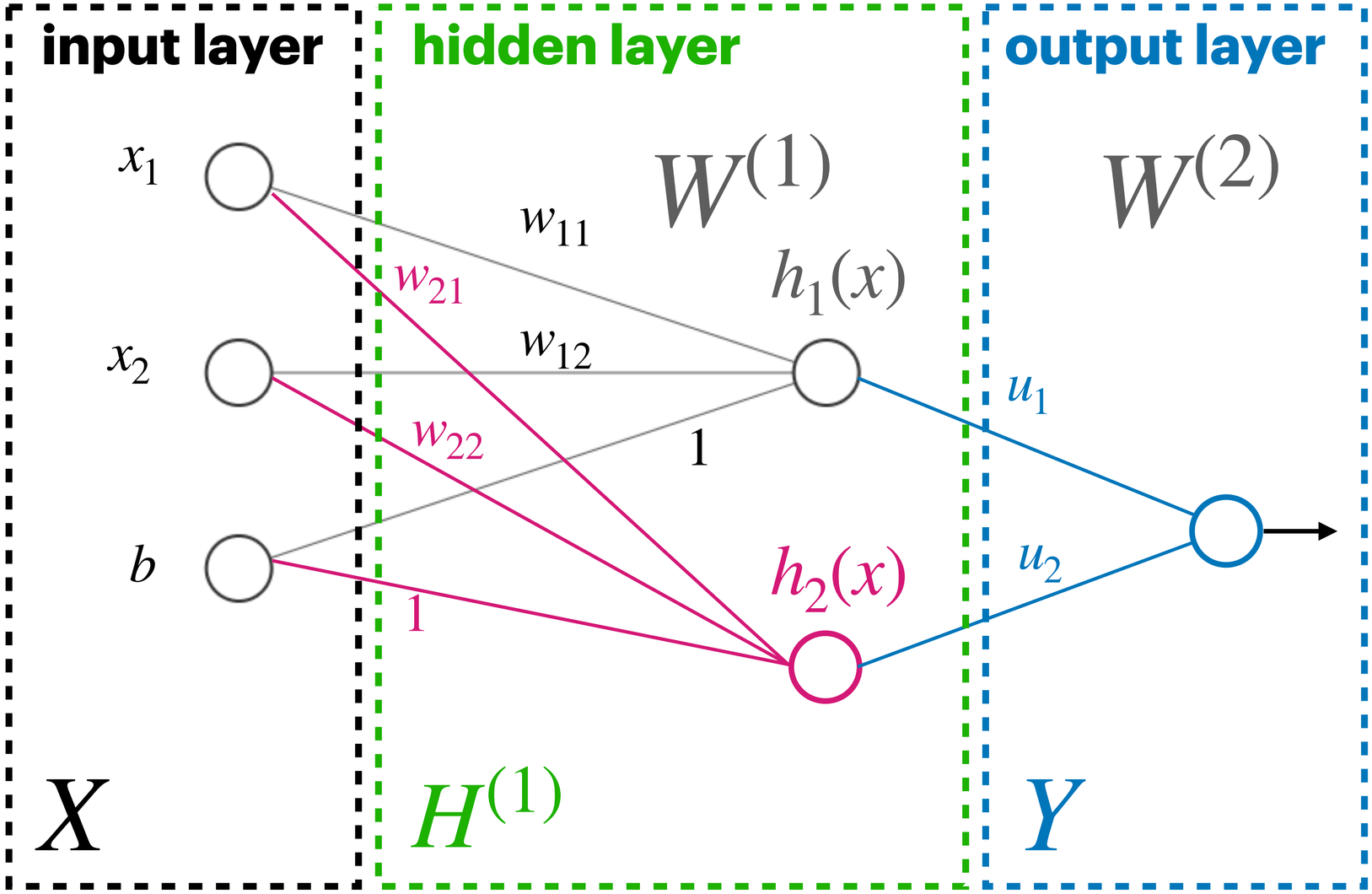
$$H^{(1)} = \phi(W^{(1)}X + b^{(1)})$$

or with L hidden layers:

$$H^{(L)} = \phi(W^{(L)}H^{(L-1)} + b^{(L)})$$

Building Neural Networks

- Linear algebra makes things simpler: Replace everything with matrices!



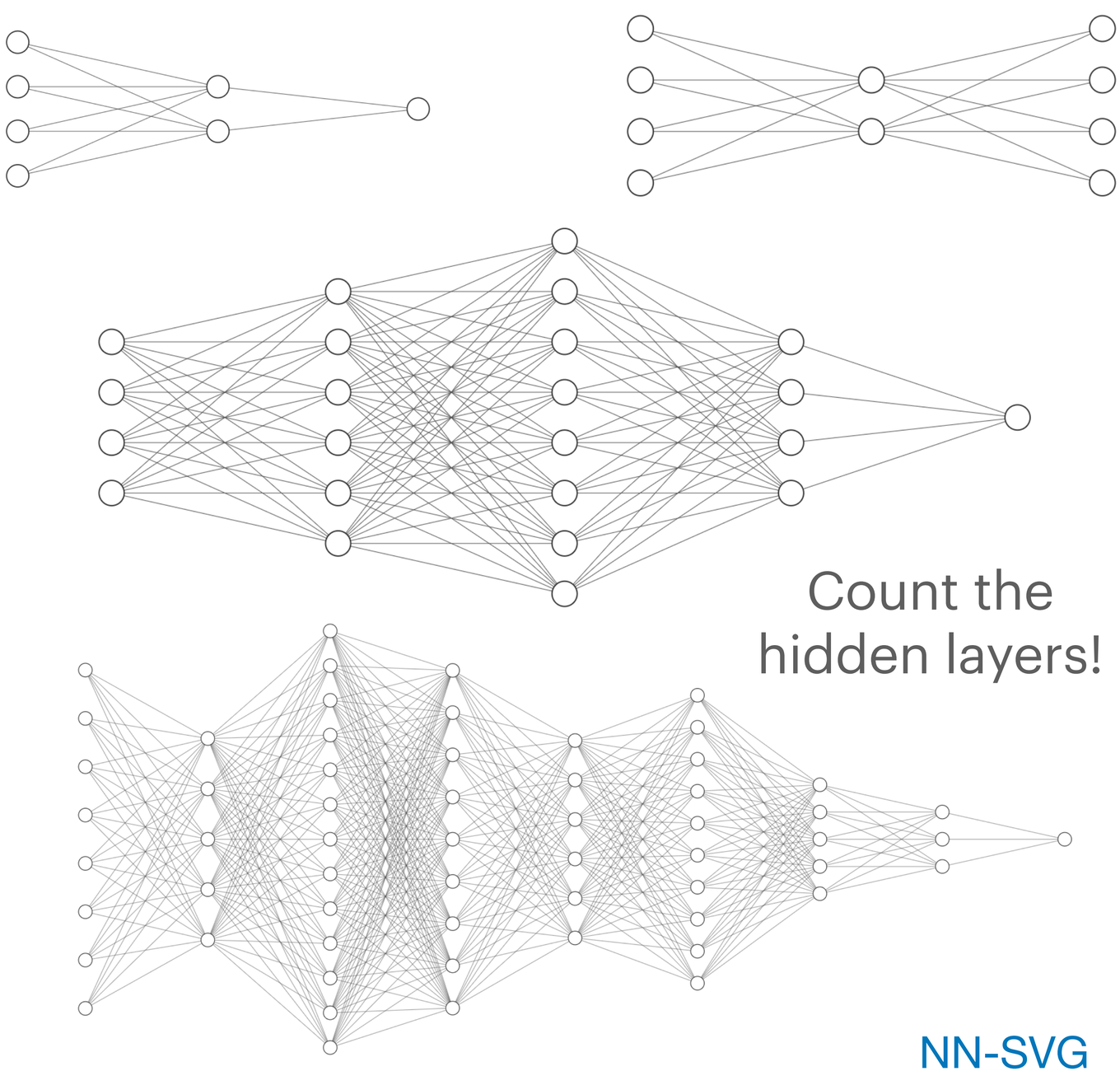
$$Y = W^{(2)}H^{(1)} + b^{(2)}$$

or with L hidden layers:

$$Y = W^{(L)}H^{(L-1)} + b^{(L)}$$

NN Architecture

- Neural networks: linear models after inputs are mapped to learned features through nonlinear transformations
- Architecture defines nature of nonlinear transformation
- Do not explicitly design nonlinear features!
  - Optimize how to weight and combine them during training



Count the hidden layers!

NN-SVG

How to train your neural network

- What we have so far:
  - **Training dataset:** input and target* data $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - **Model** that is a weighted series of **nonlinear transformations** of the inputs

$$H^{(1)} = \phi(W^{(1)}X + b^{(1)}) \quad H^{(L)} = \phi(W^{(L)}H^{(L-1)} + b^{(L)}) \quad Y = W^{(L)}H^{(L-1)} + b^{(L)}$$

  - **Learning objective: minimize loss** function $L(Y, Y')$ to find **optimal weights**
    - (eg. minimize distance from modeled output to target data)

How to train your neural network

- What we have so far:
  - **Training dataset:** input and target* data $S = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - **Model** that is a weighted series of **nonlinear transformations** of the inputs

$$H^{(1)} = \phi(W^{(1)}X + b^{(1)}) \quad H^{(L)} = \phi(W^{(L)}H^{(L-1)} + b^{(L)}) \quad Y = W^{(L)}H^{(L-1)} + b^{(L)}$$

  - **Learning objective: minimize loss** function $L(Y, Y')$ to find **optimal weights**
    - (eg. minimize distance from modeled output to target data)

What optimization algorithm do we use?

Gradient Descent

- We want to minimize the loss $L(Y, Y')$ to find the optimal weights w:

- At iteration t:

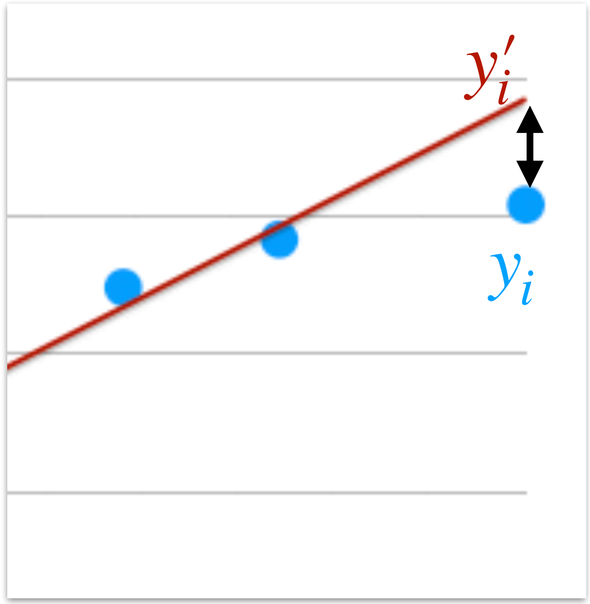
  - Compute the gradient $\nabla_w L(w(t))$: direction of steepest increase of $L(w)$ at $w(t)$

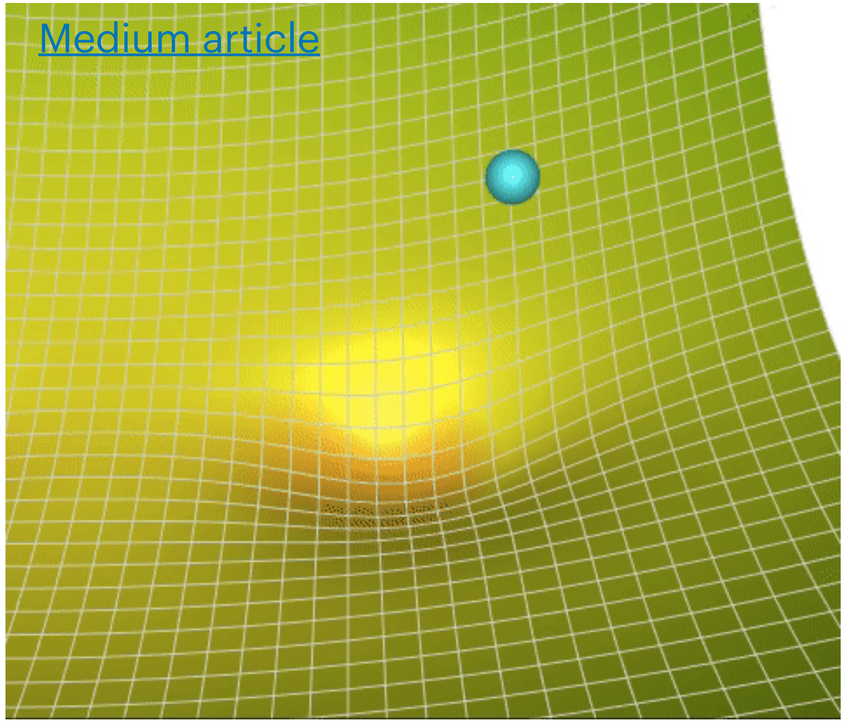
  - Take a small step in the opposite direction:

$$w(t+1) = w(t) - \eta \nabla_w L(w(t))$$

Step size / learning rate

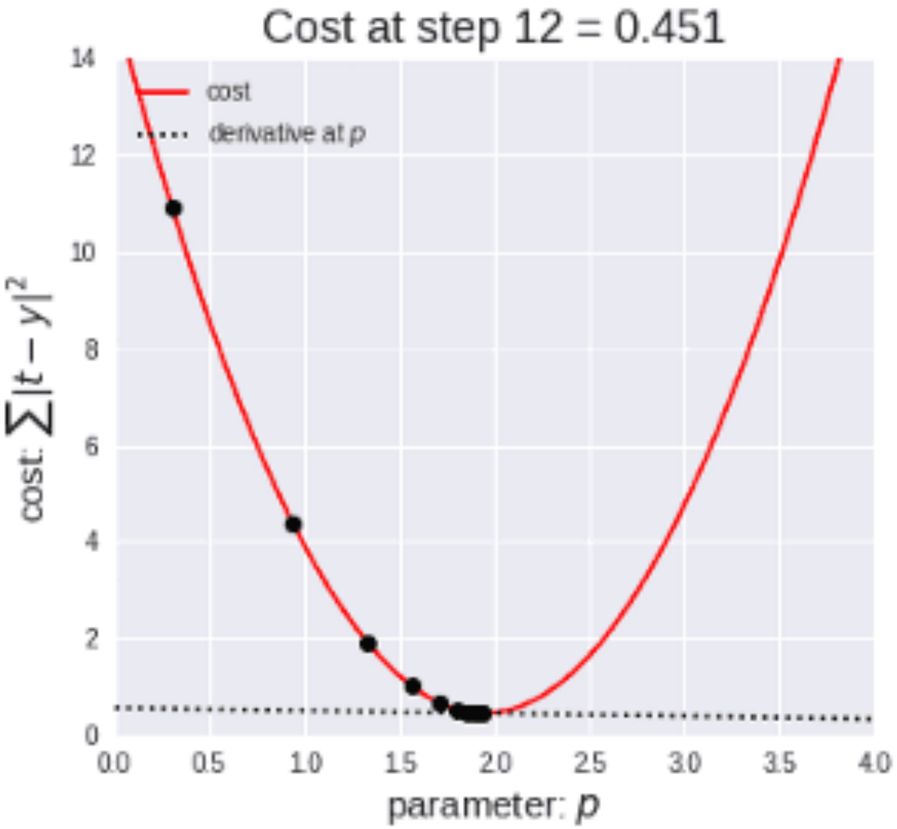
  - Continue until max t or stopping condition

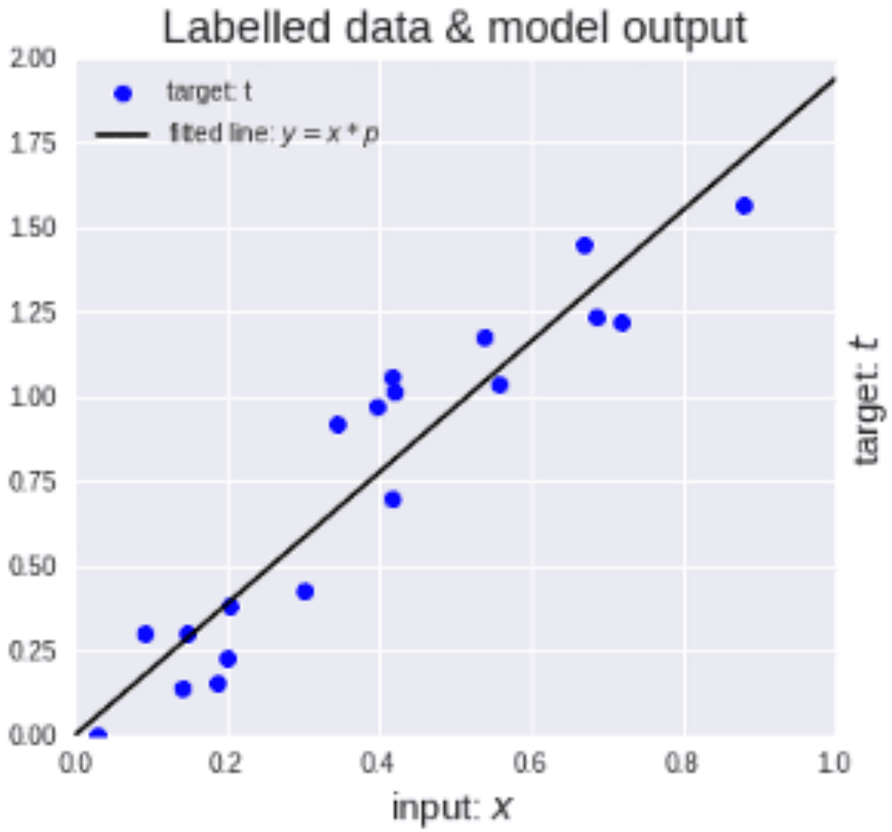




Gradient Descent

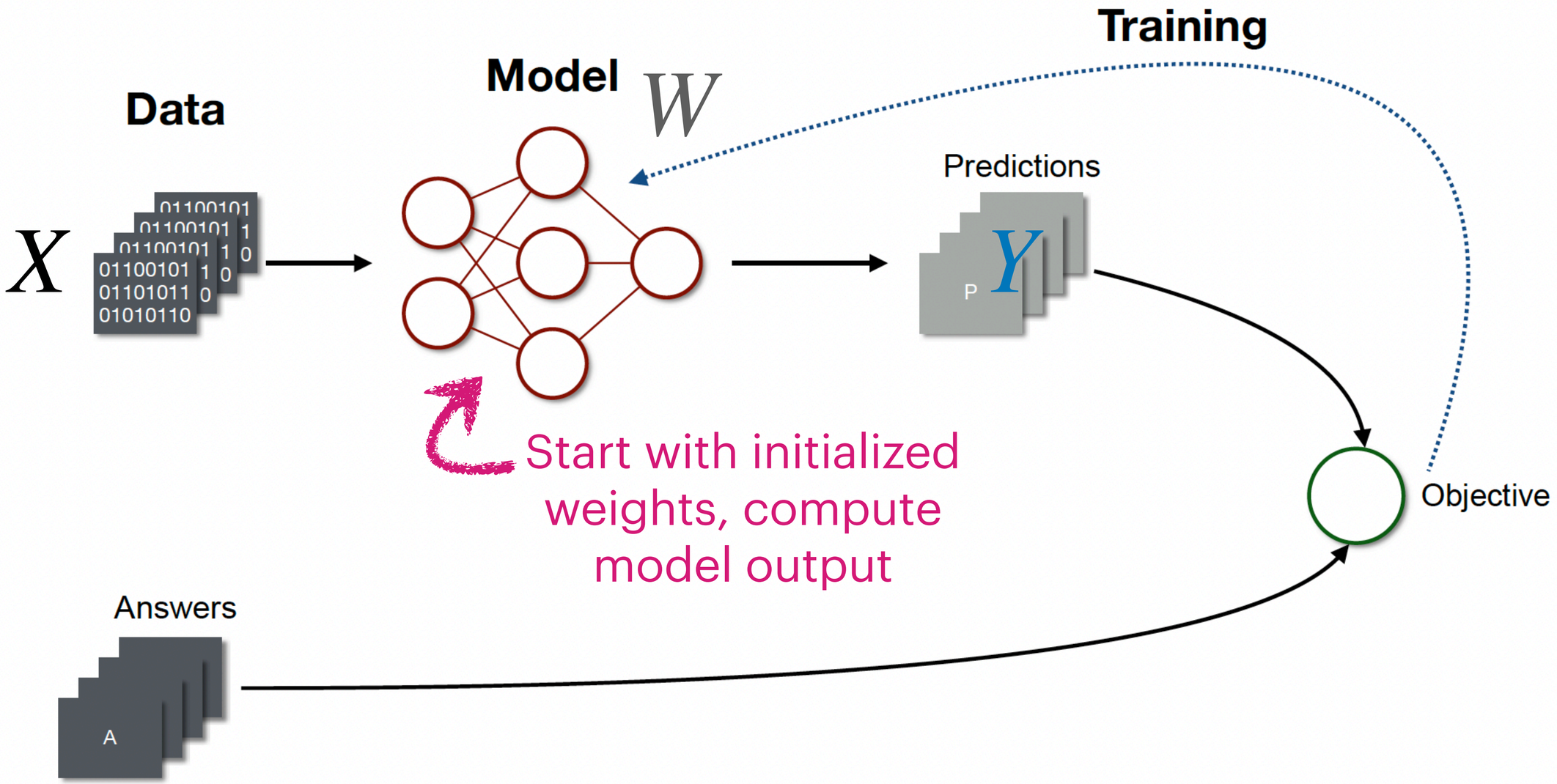
- Linear model example:



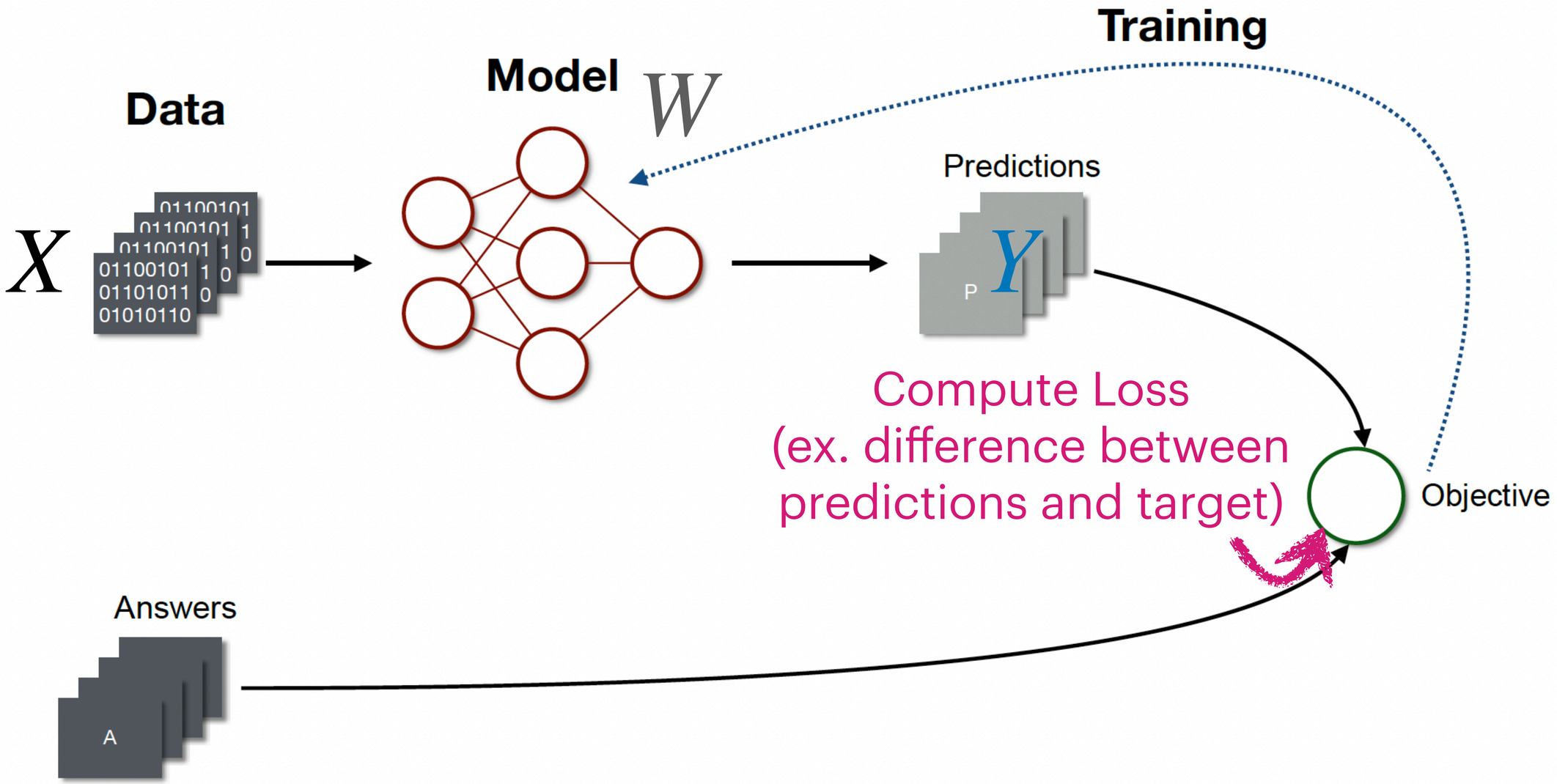


tutorial example

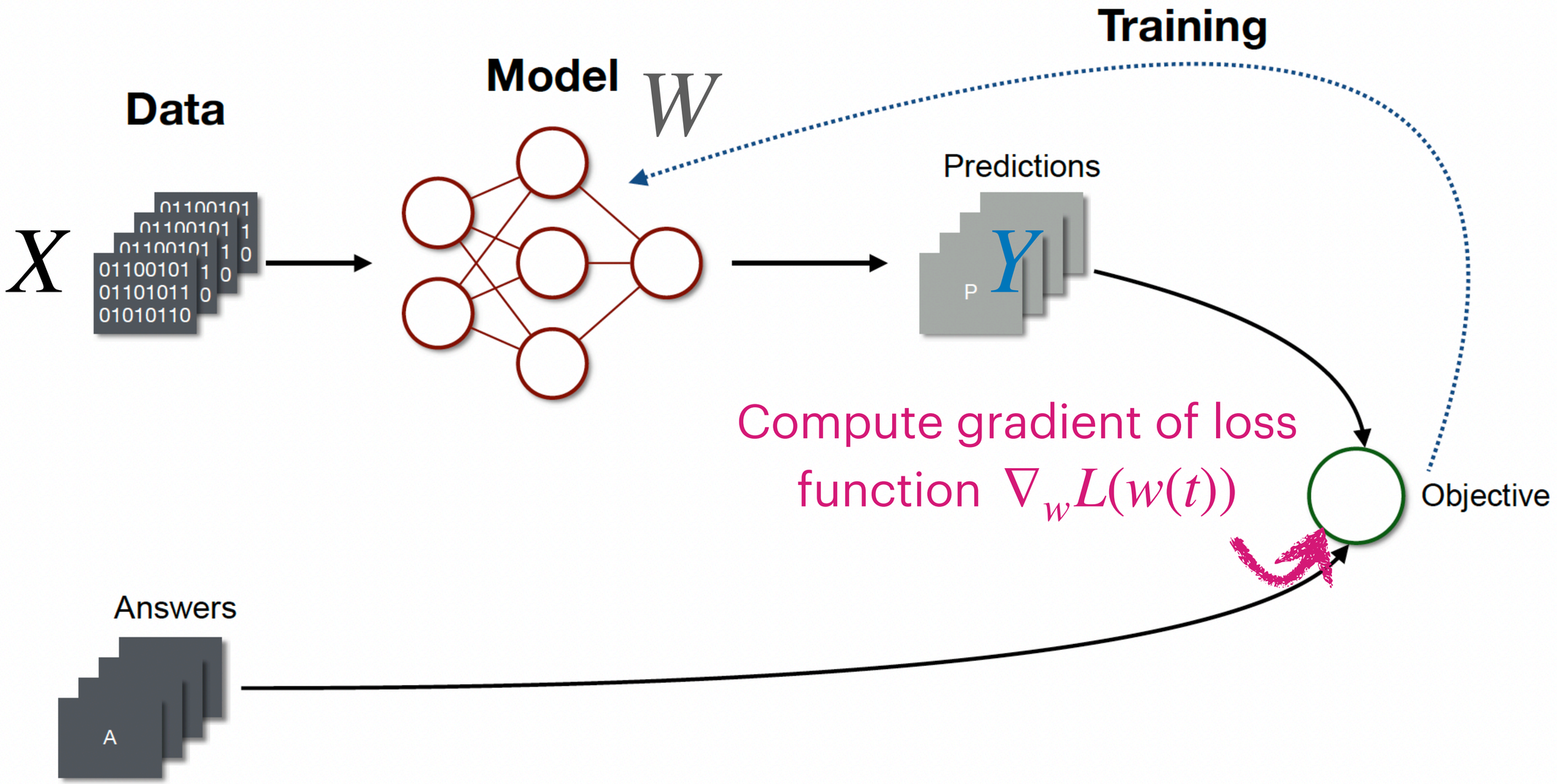
How to train your neural network



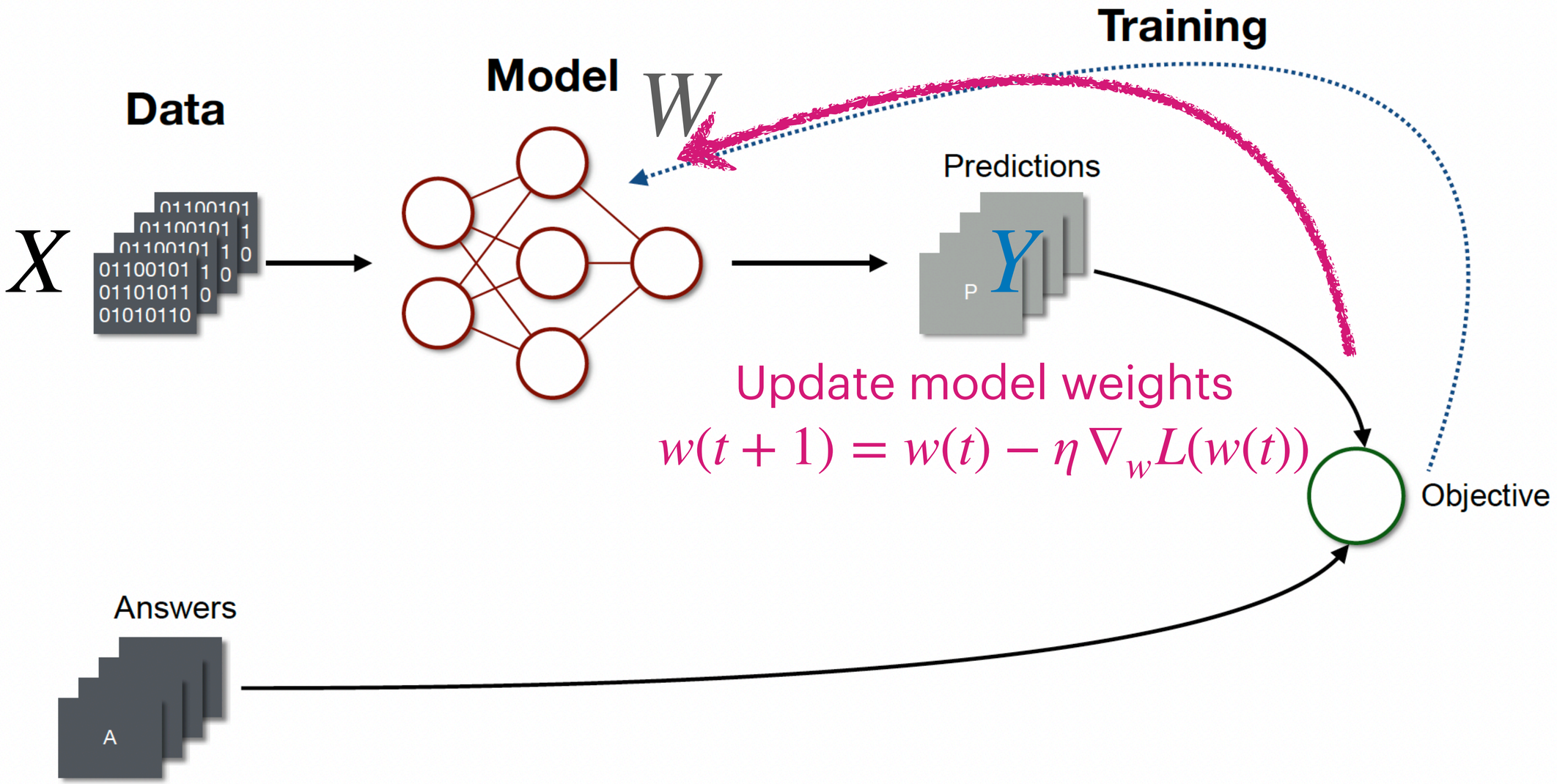
How to train your neural network



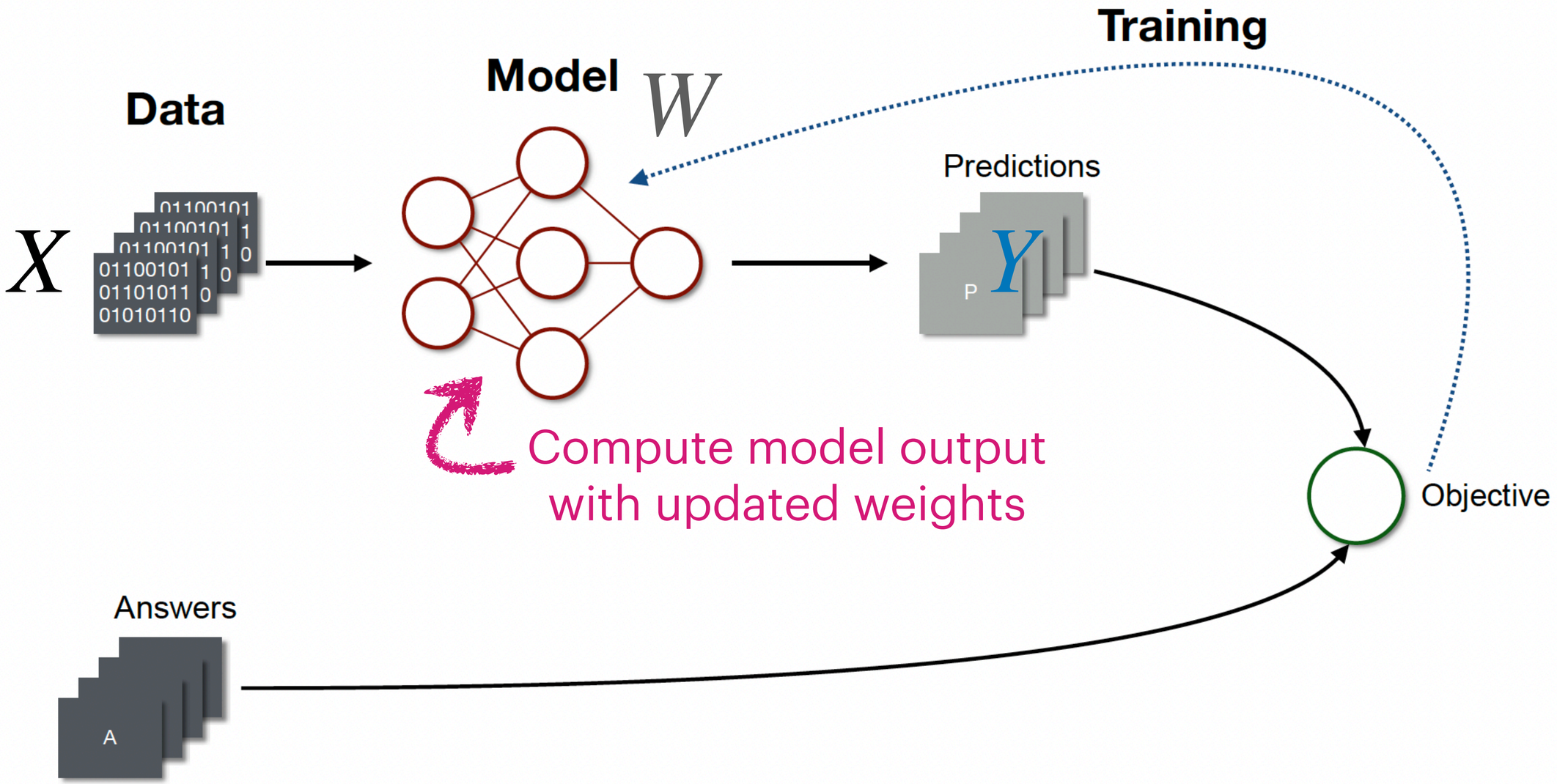
How to train your neural network



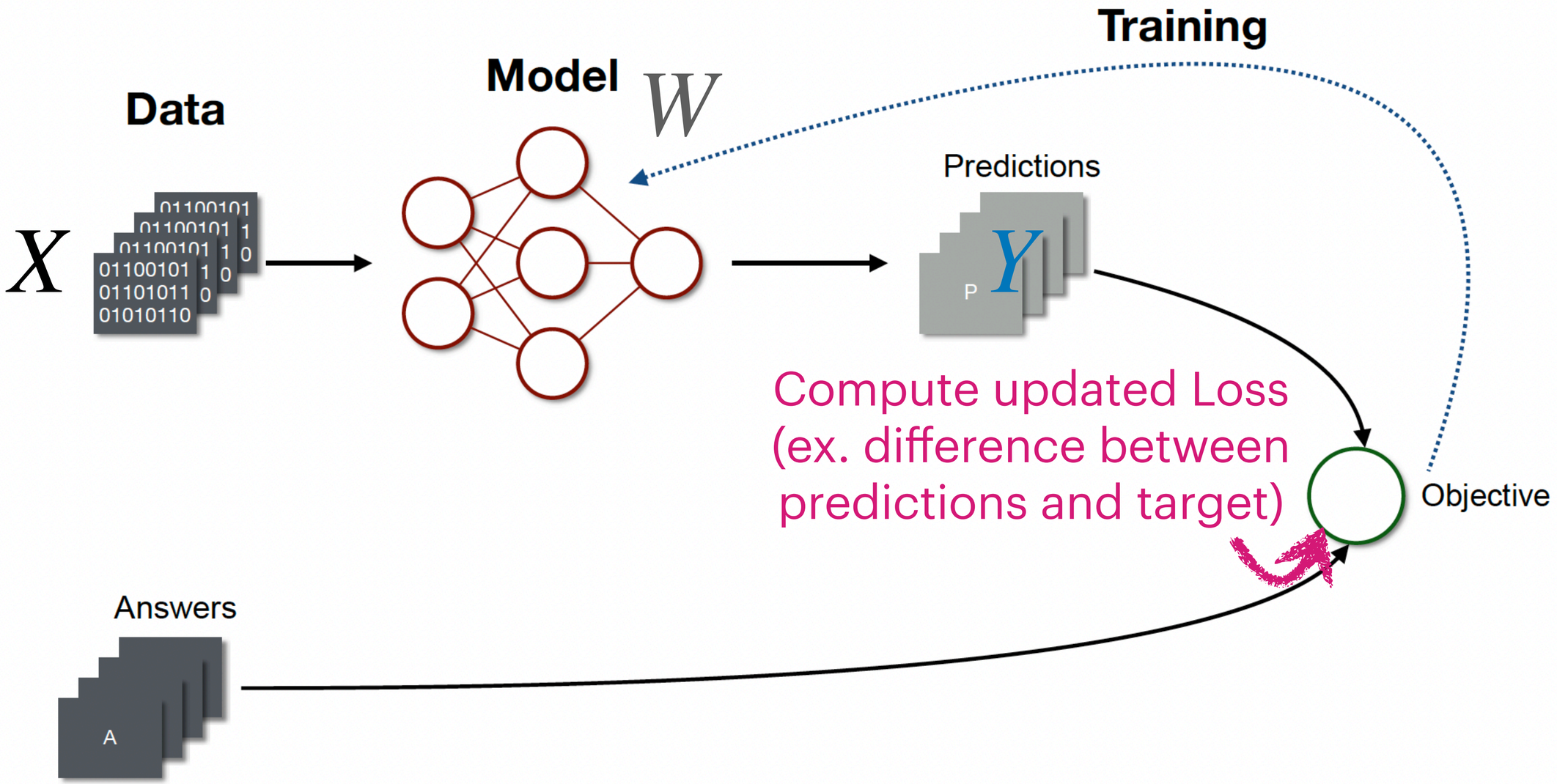
How to train your neural network



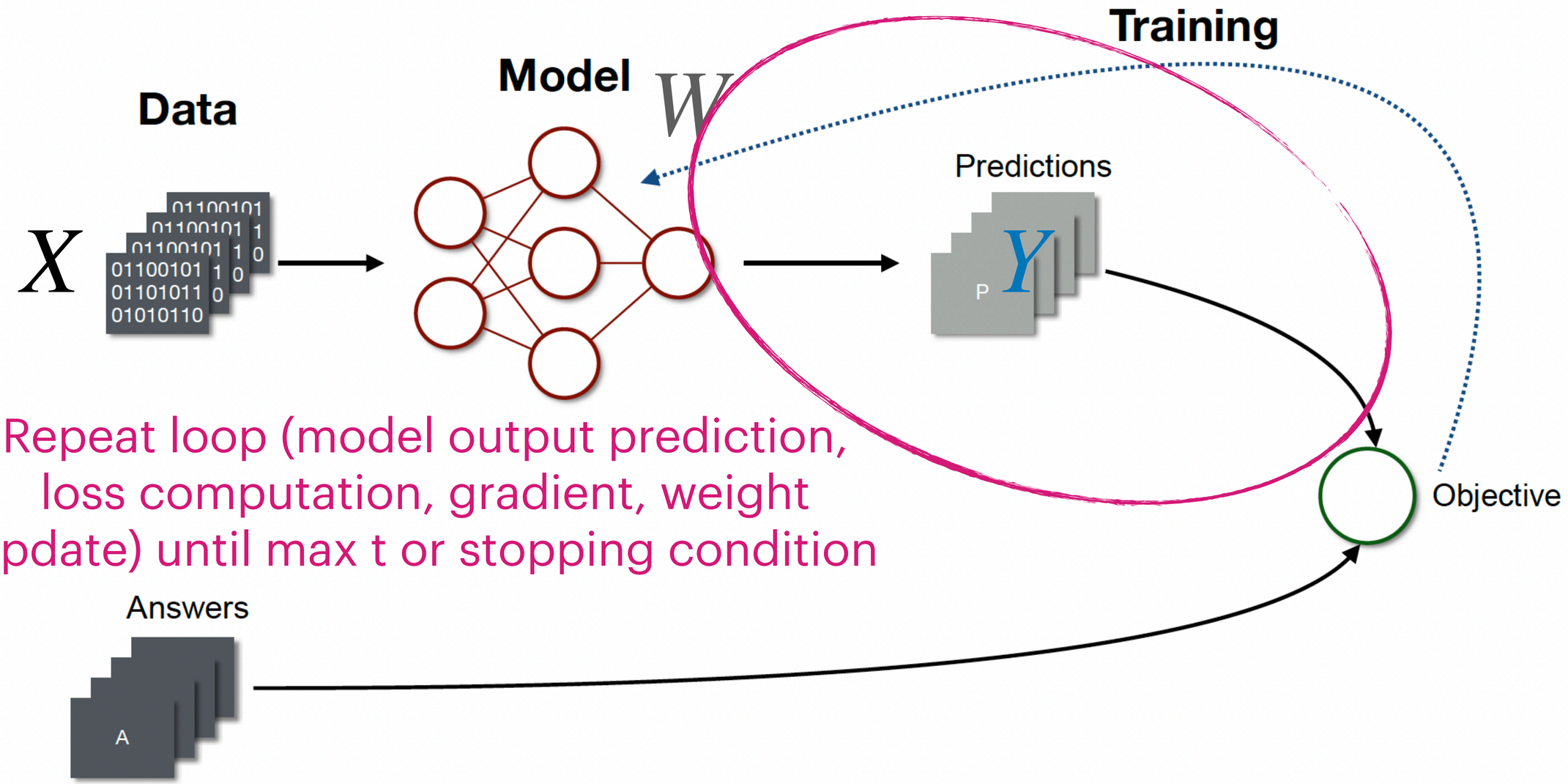
How to train your neural network



How to train your neural network

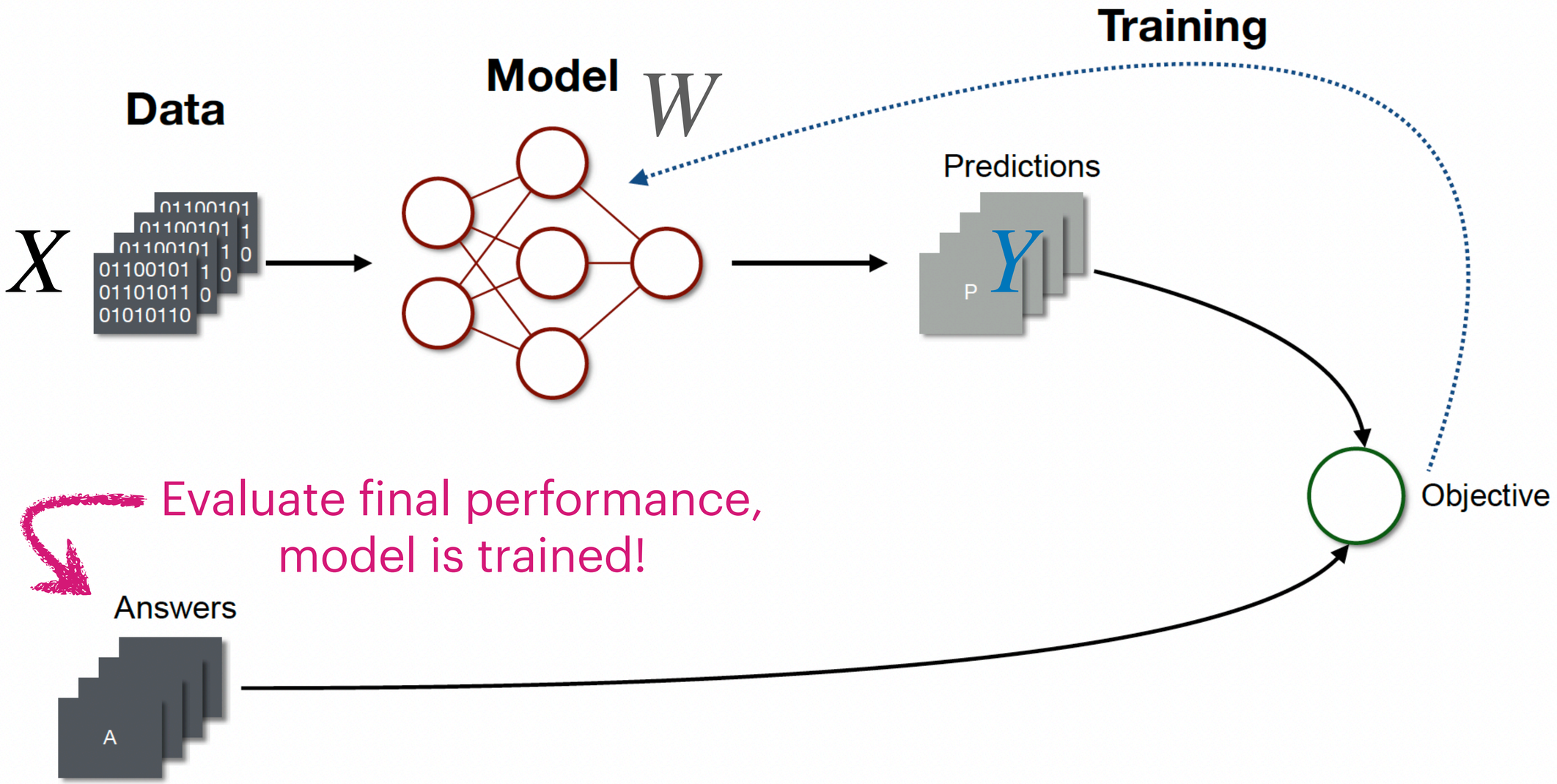


How to train your neural network

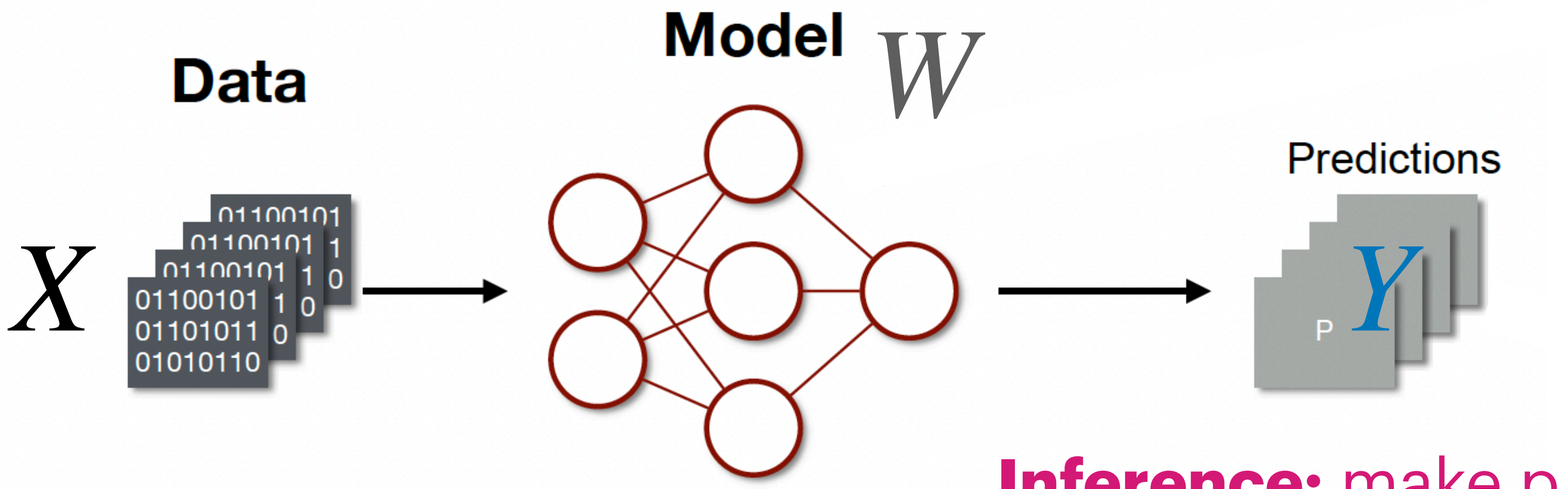


Repeat loop (model output prediction, loss computation, gradient, weight update) until max t or stopping condition

How to train your neural network

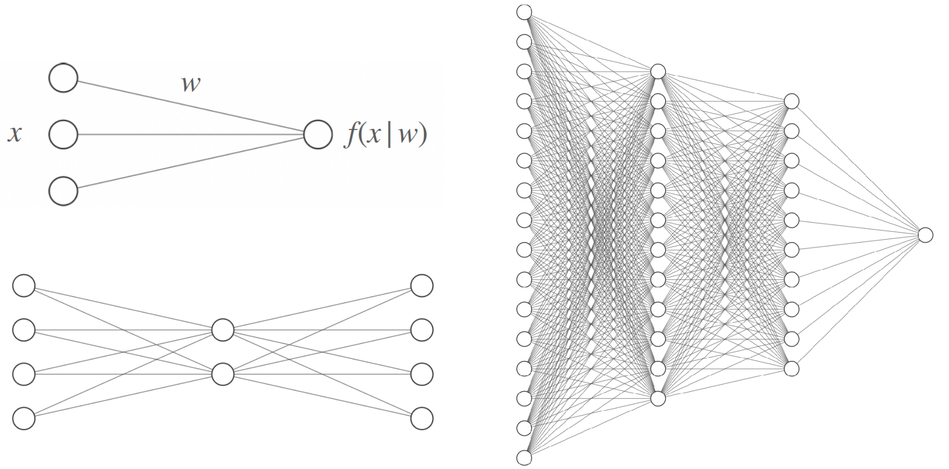


How to train your neural network

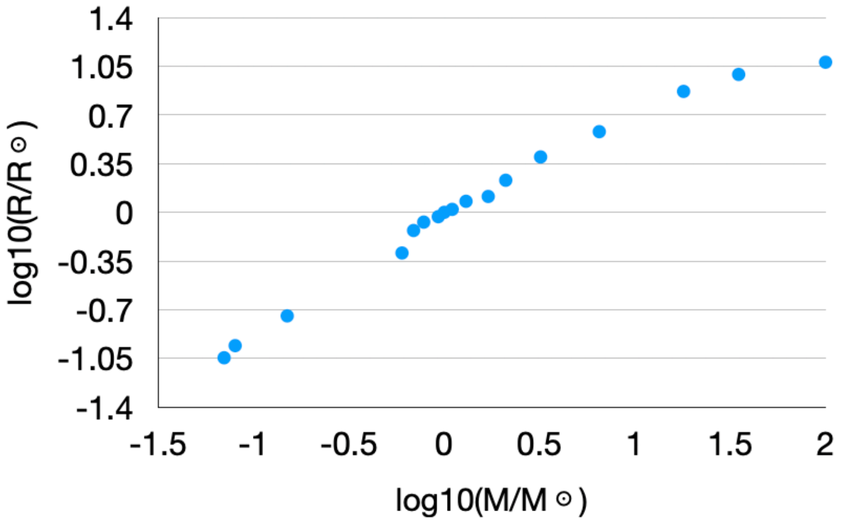


Inference: make predictions with trained neural network. Weights are finalized.

Model Choices →more in the rest of the school



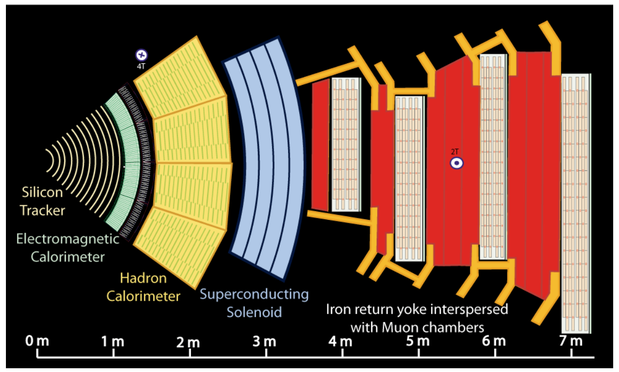
- Architecture: depth/complexity
- Activation functions: ReLU, ...? (*next lecture*)
- Loss function: pick to suit learning objective!
- Types of data: images, tabular, multi-modal...
  - Data management (preprocessing, batching etc)
- Types of machine learning?
  - Supervised vs. unsupervised (*next slide*)
  - Classification vs. regression (*next slide*)
  - Should you even use a NN? →***Pick the model that best suits your task***



Classification vs. Regression

Predict a continuous value
→Regression

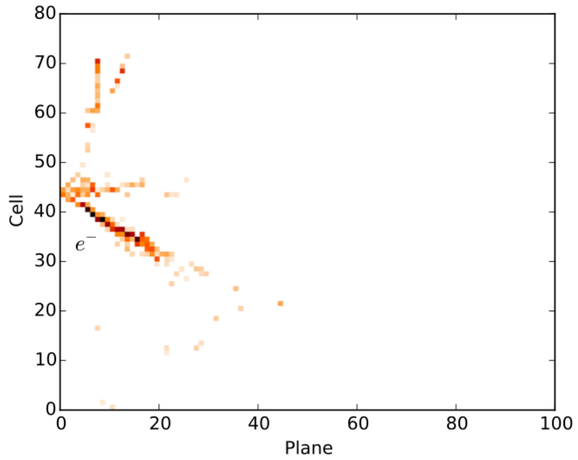
- Ex: Estimate particle momentum from detector hits

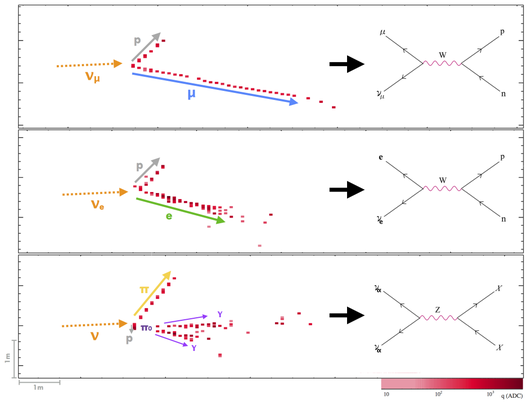


$$\rightarrow \begin{pmatrix} E \\ p_x \\ p_y \\ p_z \end{pmatrix}, q, \text{type}, p_{\text{pileup}}, \cdots$$

Predict a discrete class
→Classification

- Ex: Classify neutrino interactions



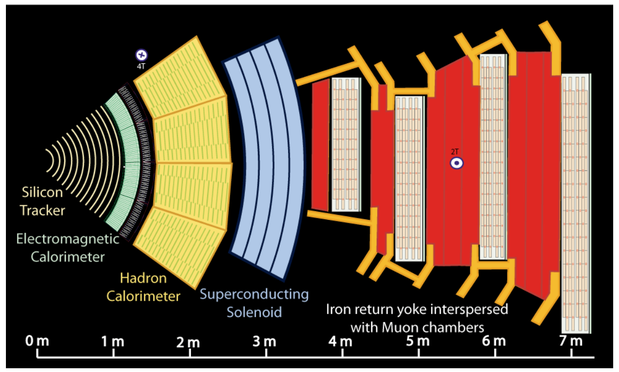


Classification vs. Regression

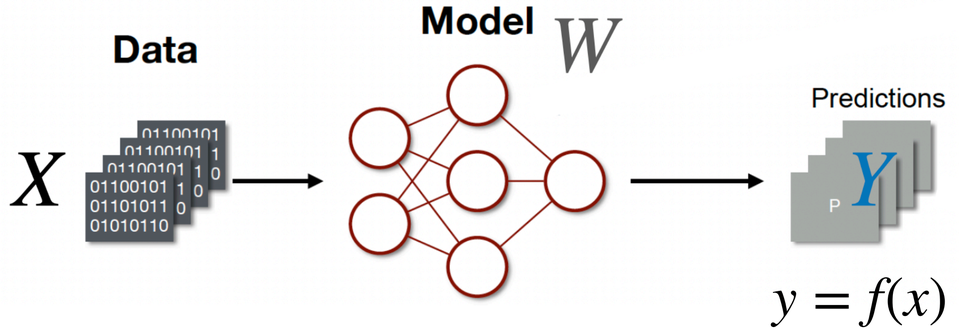
Predict a continuous value

→Regression

- Output layer $y = f(x)$
- Loss ex. Mean Squared Error
- Ex: Estimate particle momentum from detector hits



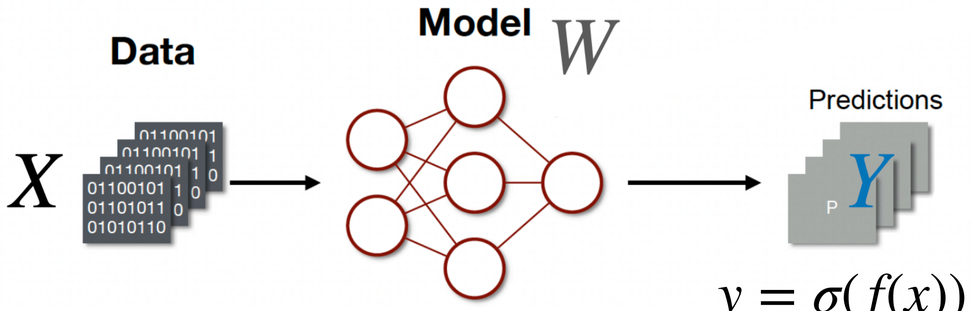
$$\rightarrow \begin{pmatrix} E \\ p_x \\ p_y \\ p_z \end{pmatrix}, q, \text{type}, p_{\text{pileup}}, \cdots$$



$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i^{\text{true}} - y_i^{\text{pred}})^2$$

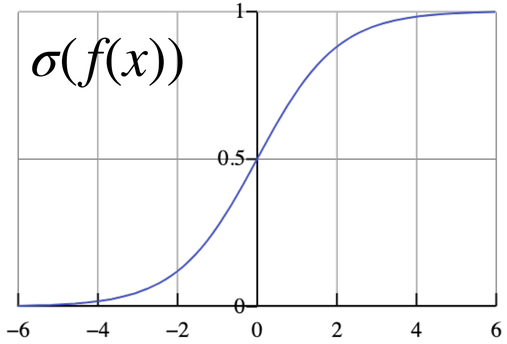
- MSE ~average squared distance between your predictions and the true values

Classification vs. Regression



$y = \sigma(f(x))$

$y = \text{softmax}(f(x))$



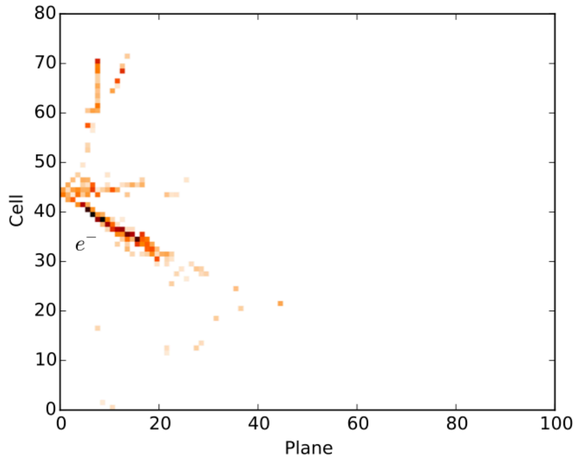
$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}^{\text{true}} \log(y_{ij}^{\text{pred}})$$

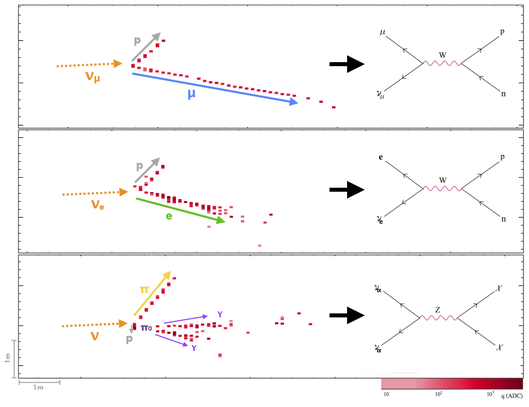
- Cross entropy~how surprised you are by the true answer given your predicted probability

Predict a discrete class

→Classification

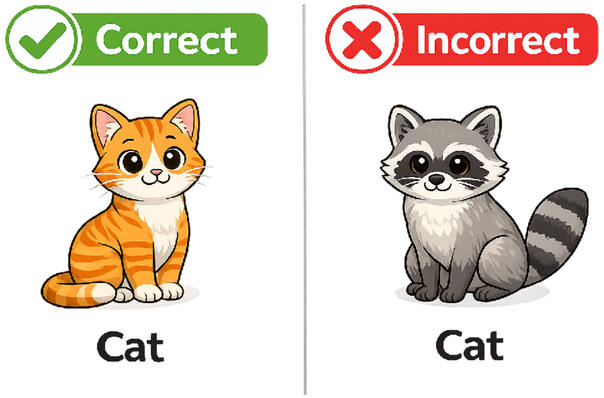
- Output layer $y = \phi(f(x))$
  - Binary $y = \sigma(f(x))$ →between (1,0)
  - Multi-class $y = \text{softmax}(f(x))$ →vector of probabilities summing to 1
- Loss ex. Cross-entropy
- Ex: Classify neutrino interactions



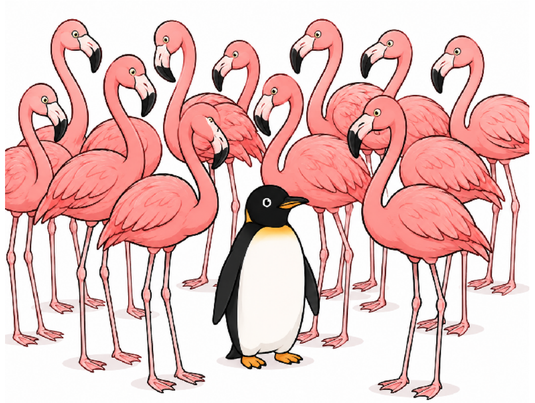


Supervised vs. Unsupervised Learning

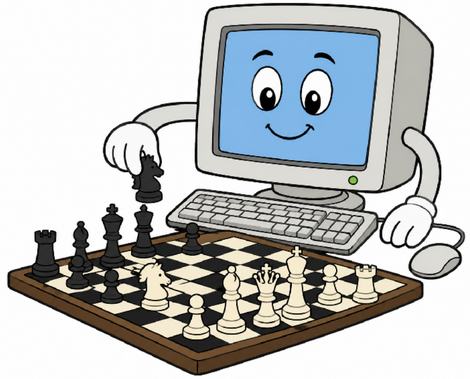
Labeled Data → **Supervised Learning** → Compare prediction vs truth



Unlabeled Data → **Unsupervised Learning** → Find structure or patterns in the data

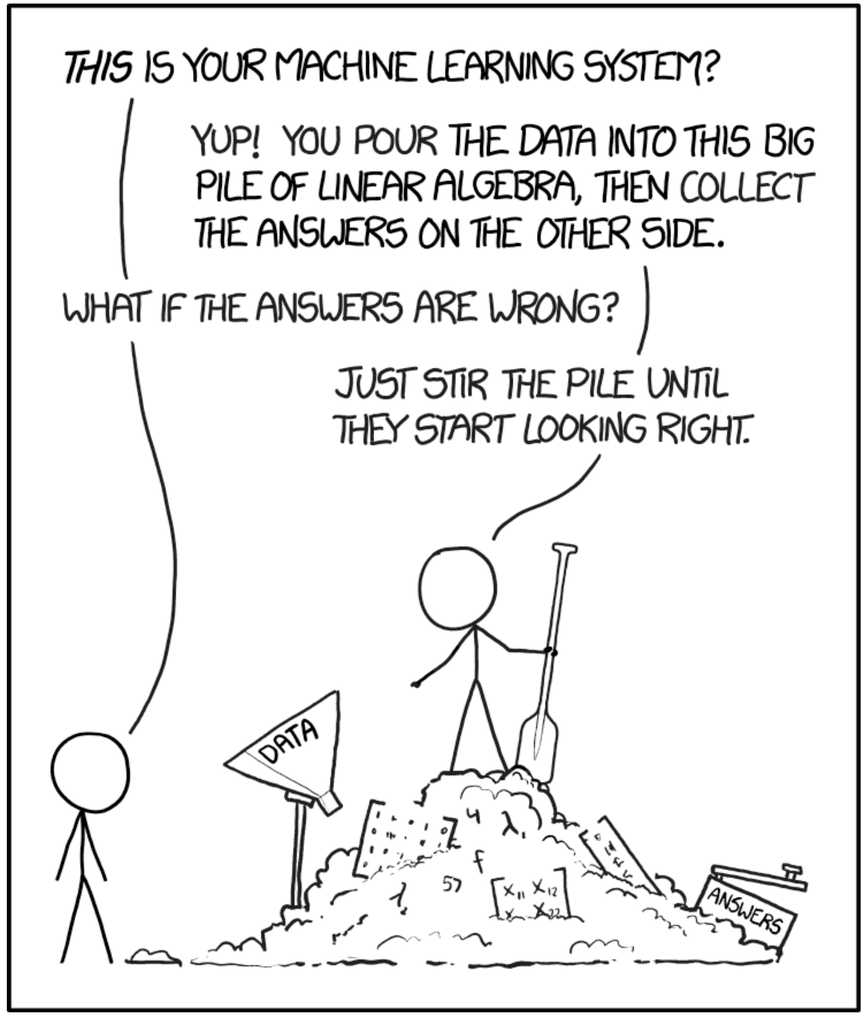


Unlabeled Data → **Reinforcement Learning** → Maximize reward signal over time



Takeaways + Conclusion

- Machine Learning: computer algorithms that **automatically** learn to approximate unknown functions from data
- Neural Networks are **universal function approximators**
  - →Linear models after inputs are mapped to **learned** features through **nonlinear transformations**
  - Optimize how to weight and combine features during training
- **Pick the model that best suits your task**

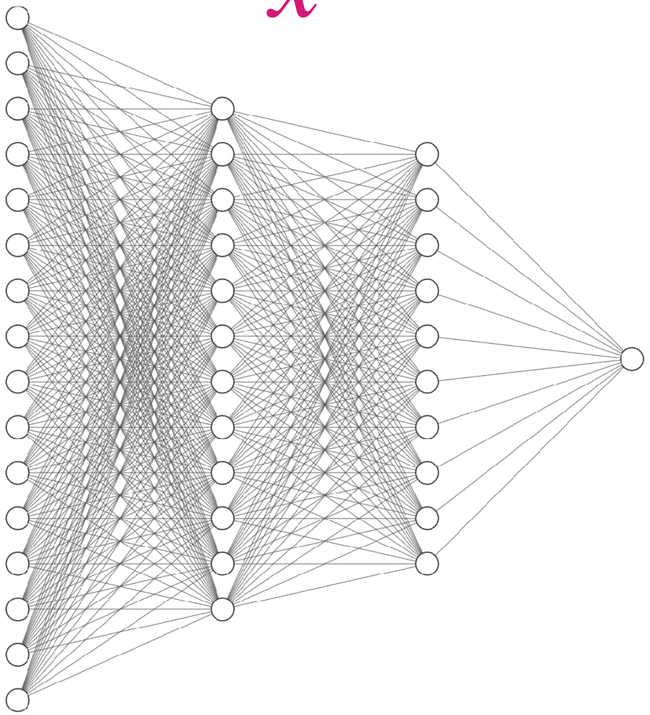


Thank you!

Recap

Data

x



Model

$f(x)$

Representation of the underlying function describing your data

Prediction

$y = f(x)$

Deep neural networks trained to represent extremely complex, nonlinear functions

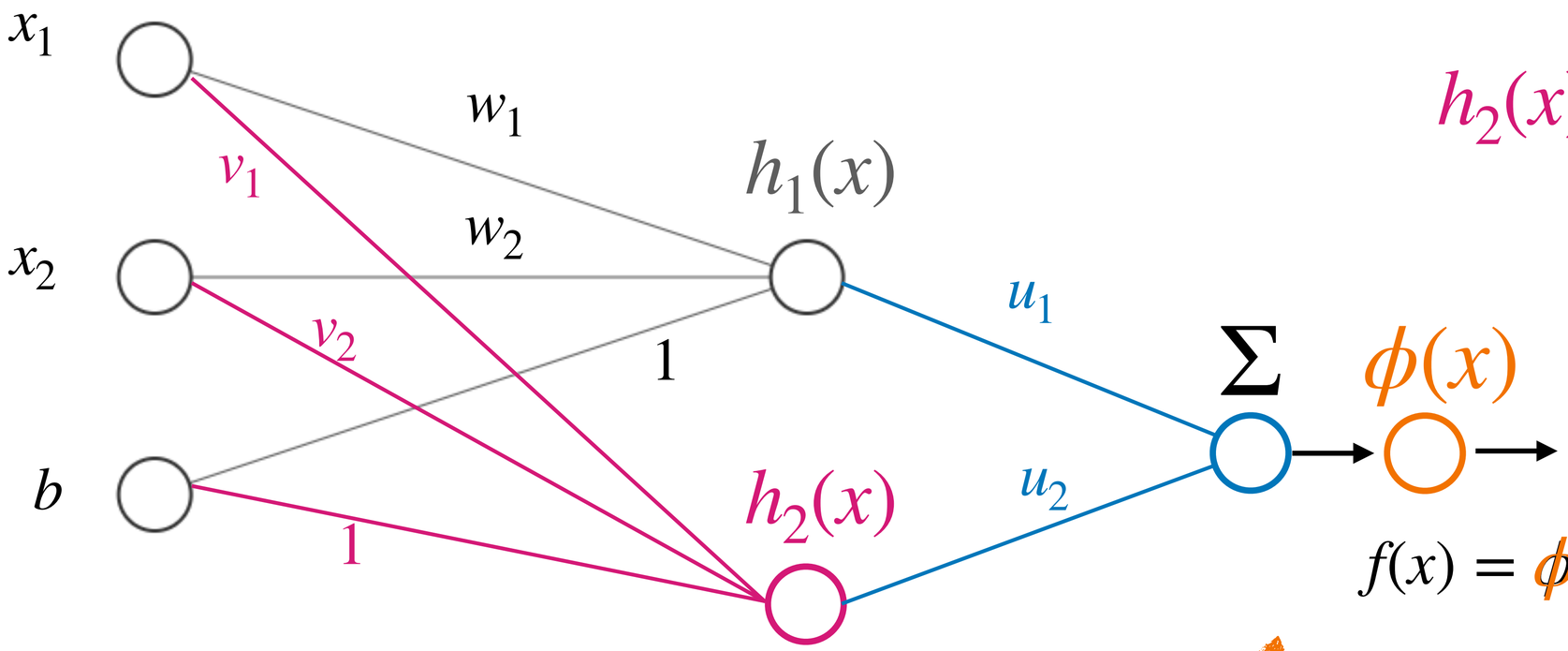
- → ***Neural networks are universal function approximators***

Building Neural Networks

- Let's add complexity: additional transformations for even more expressiveness!

$$h_1(x) = \phi(\Sigma_i w_i \cdot x_i + b)$$

$$h_2(x) = \phi(\Sigma_i v_i \cdot x_i + b)$$

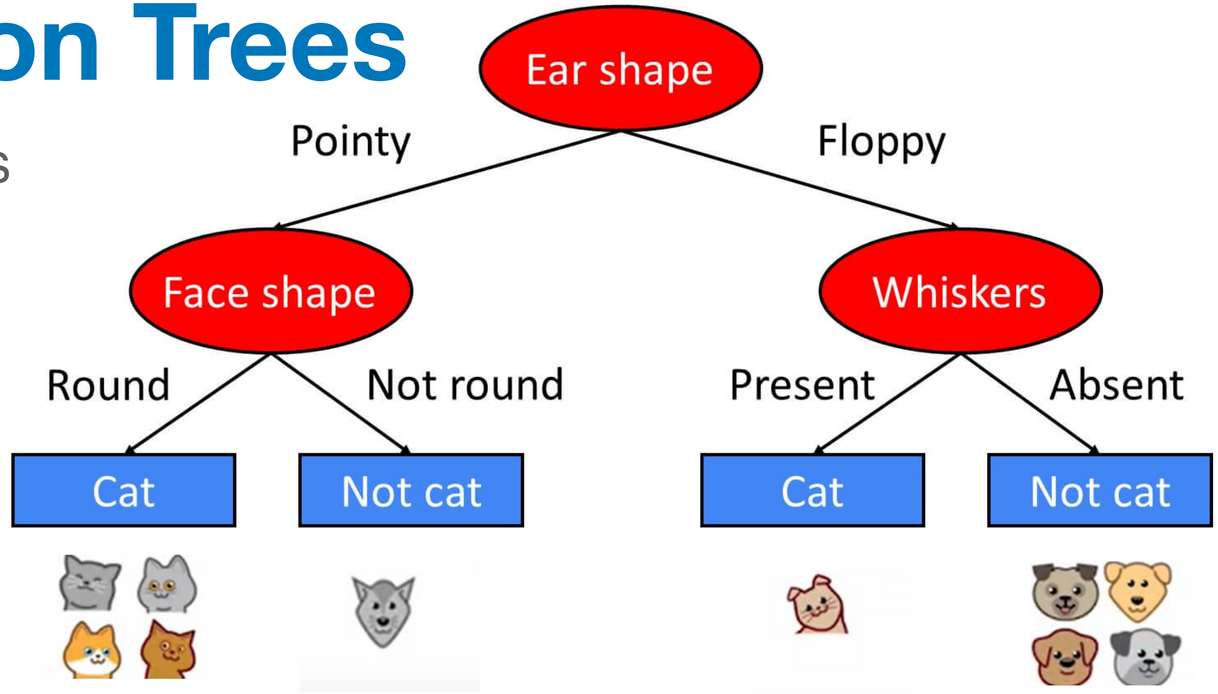


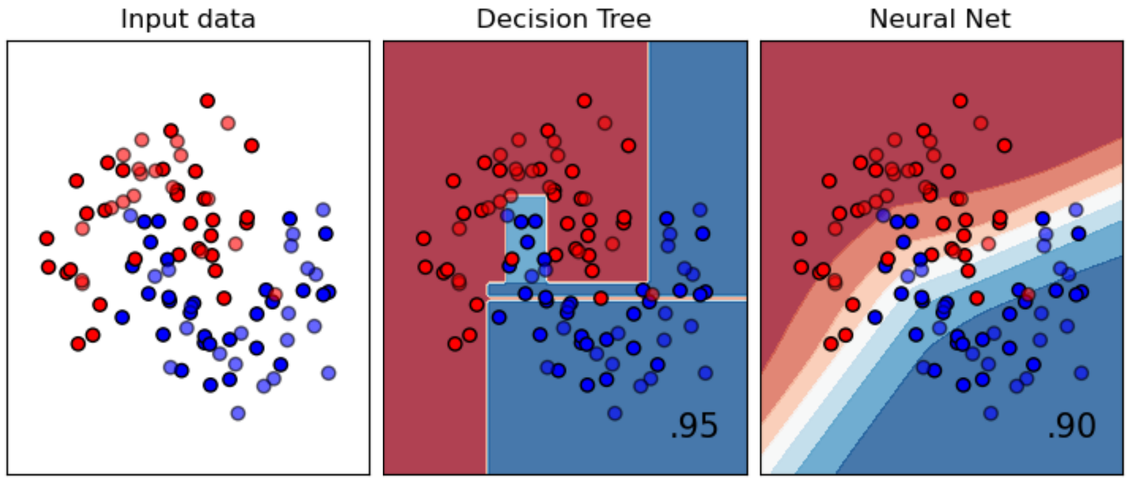
$$f(x) = \phi(u_1^T h_1(x) + u_2^T h_2(x))$$

Can transform output as well!

Sidenote: Boosted Decision Trees

- ~Flowchart of binary cuts on multiple variables
- Boosting: combine many trees:
  - When a tree misclassifies events, the next tree is trained with those misclassified events given higher weight
  - The final score is a weighted vote across all trees.
- Interpretable, easy to train
- Excellent results for appropriate problems!





→Pick the model that suits your task

BDT tutorial scikit-learn.org

Common Loss Functions

- Mean Squared Error (MSE)
  - ~average squared distance between your predictions and the true values

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i^{\text{true}} - y_i^{\text{pred}})^2$$

- Cross-entropy
  - ~how surprised you are by the true answer given your predicted probability

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i^{\text{true}}\log(y_i^{\text{pred}}) + (1-y_i^{\text{true}})\log(1-y_i^{\text{pred}})\right]$$

2 classes

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} y_{ij}^{\text{true}}\log(y_{ij}^{\text{pred}})$$

c classes