Politecnico di Bari

DIPARTIMENTO INTERATENEO DI FISICA

Dottorato in Fisica – XXXIX ciclo - 2024

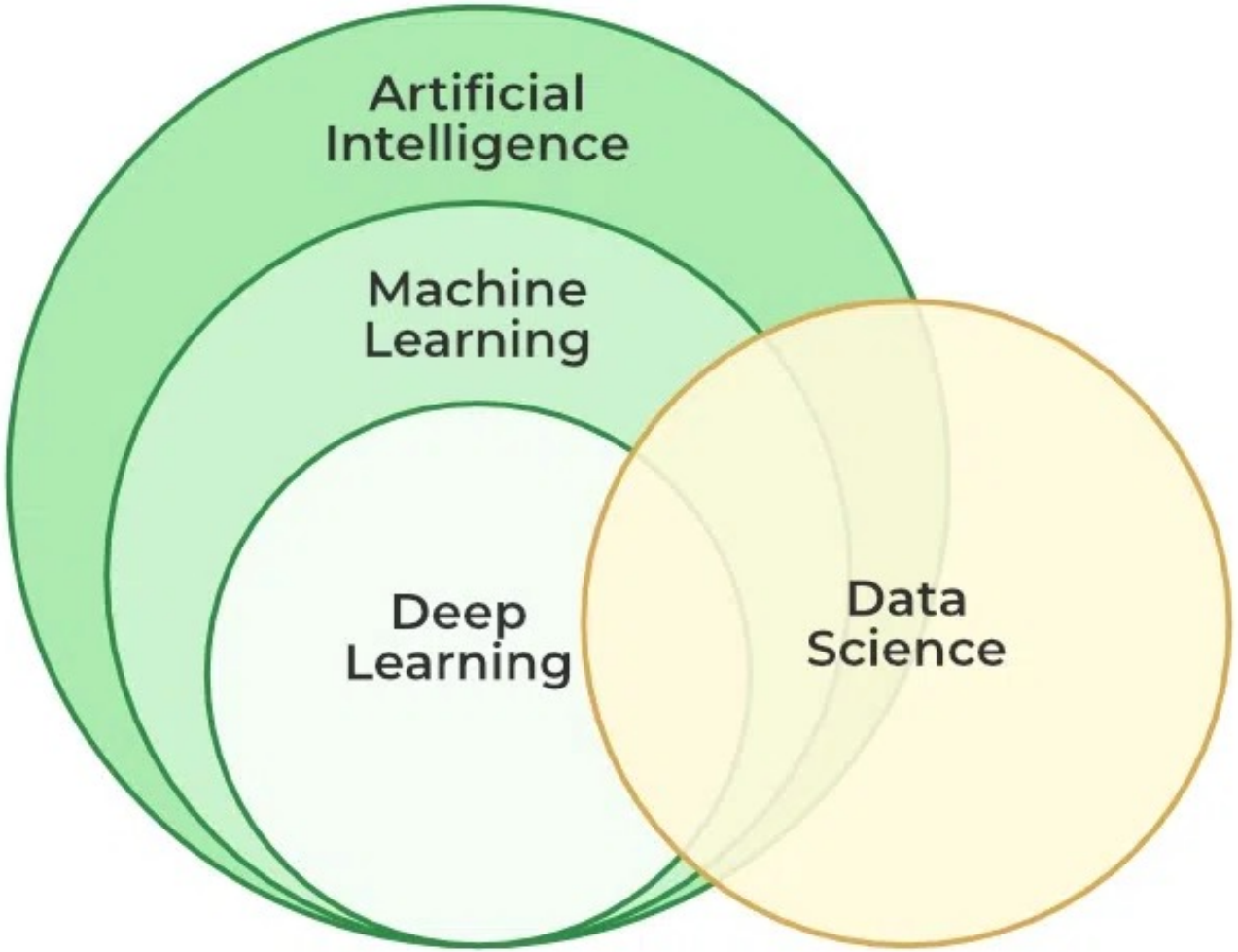
Machine Learning techniques for particle physics

Federica Maria Simone - federica.simone@poliba.it

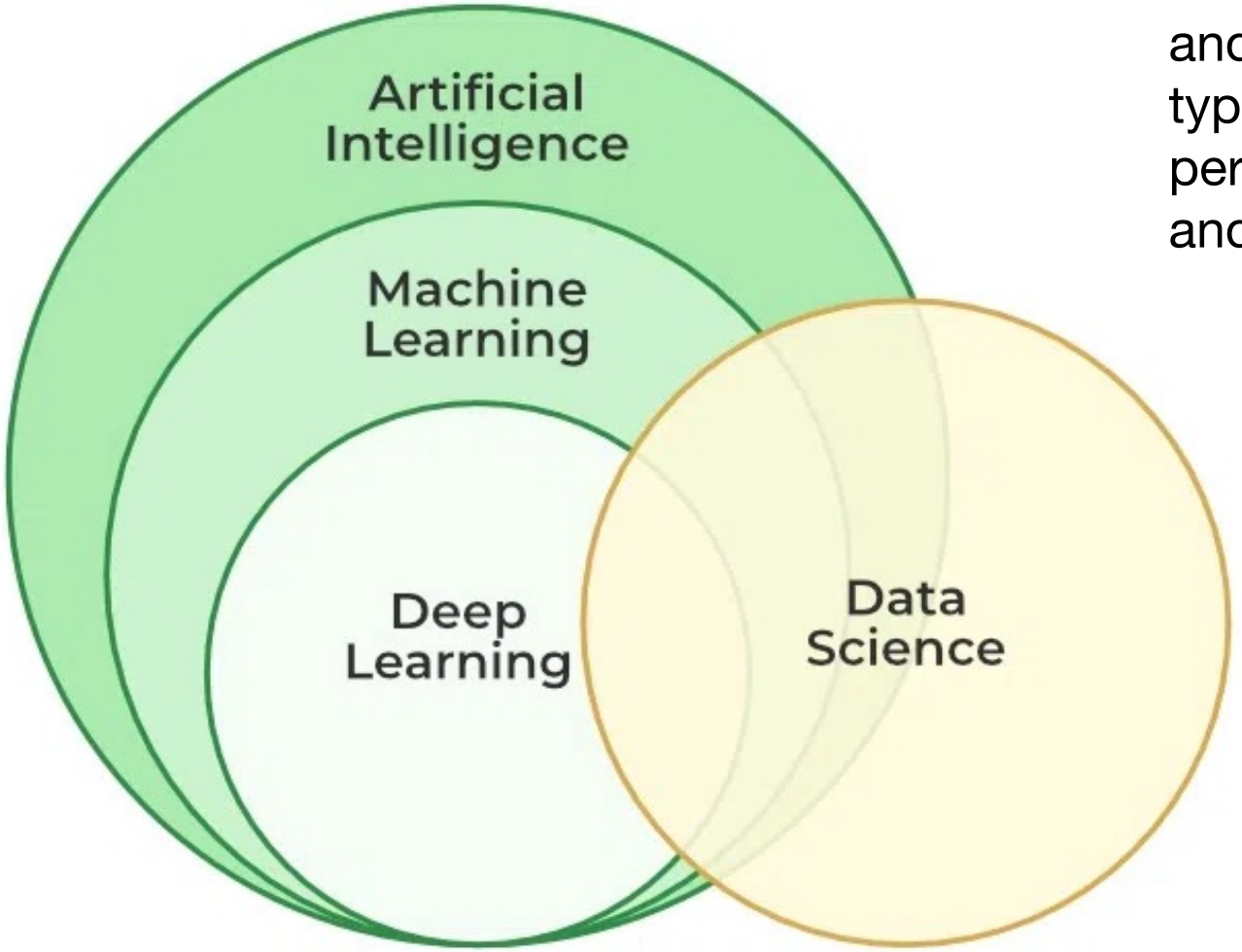
What is Machine Learning

Disclaimer: next slides are taken from many sources, please find useful links and credits on my last slide!

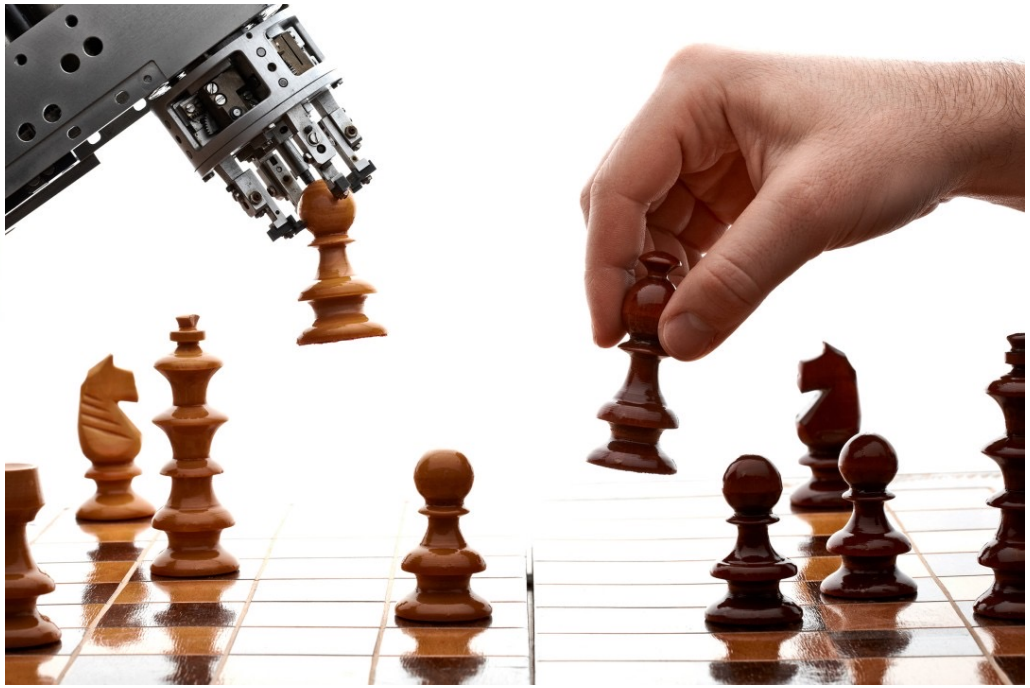
What is Machine Learning



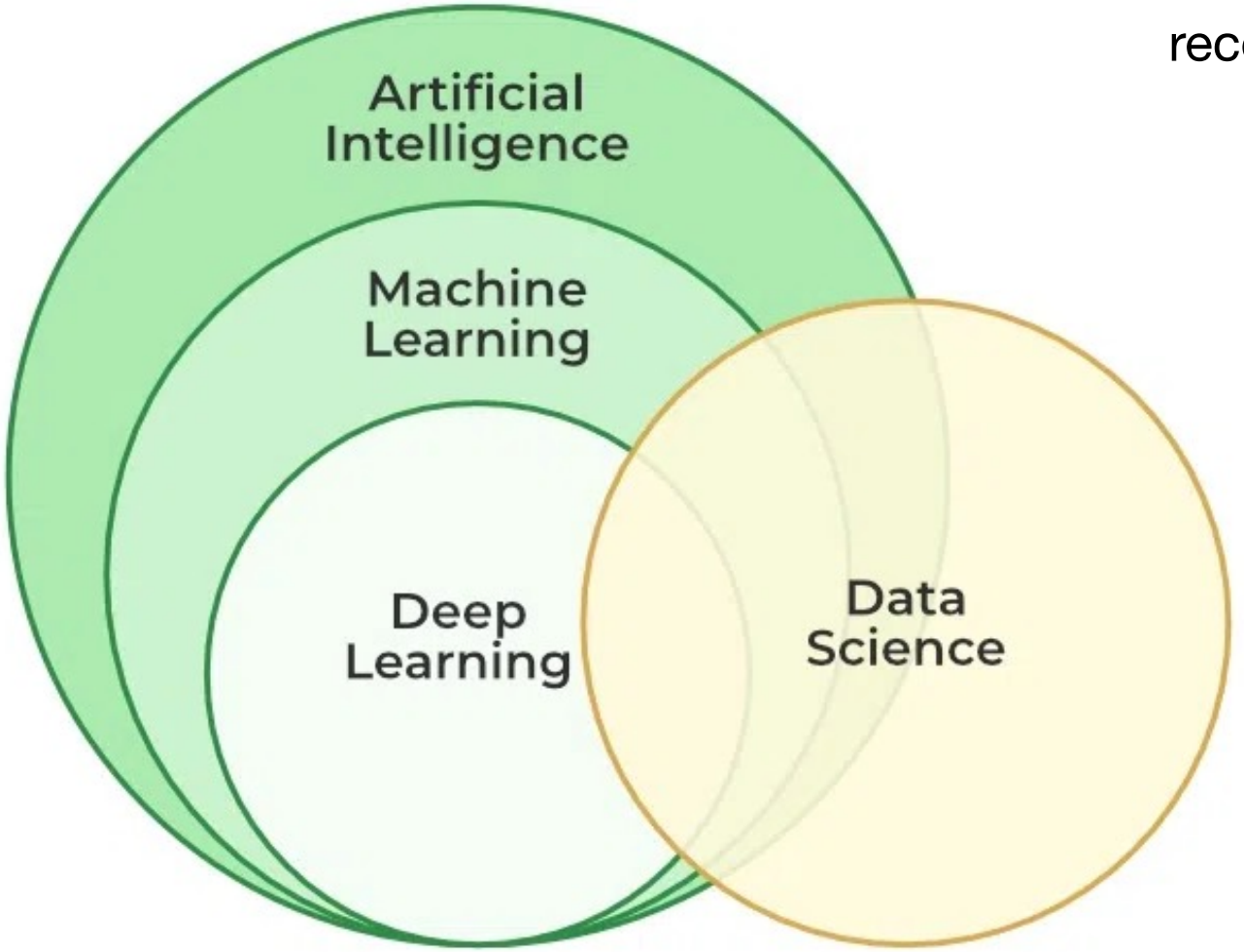
What is Machine Learning



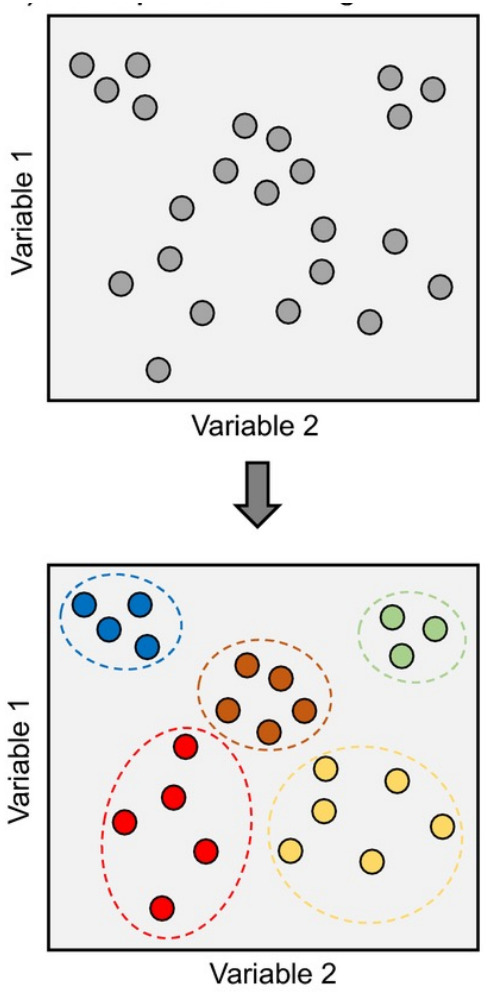
- **Artificial Intelligence:** development of algorithms and computer programs that can perform tasks that typically require human intelligence such as visual perception, speech recognition, decision-making, and language translation



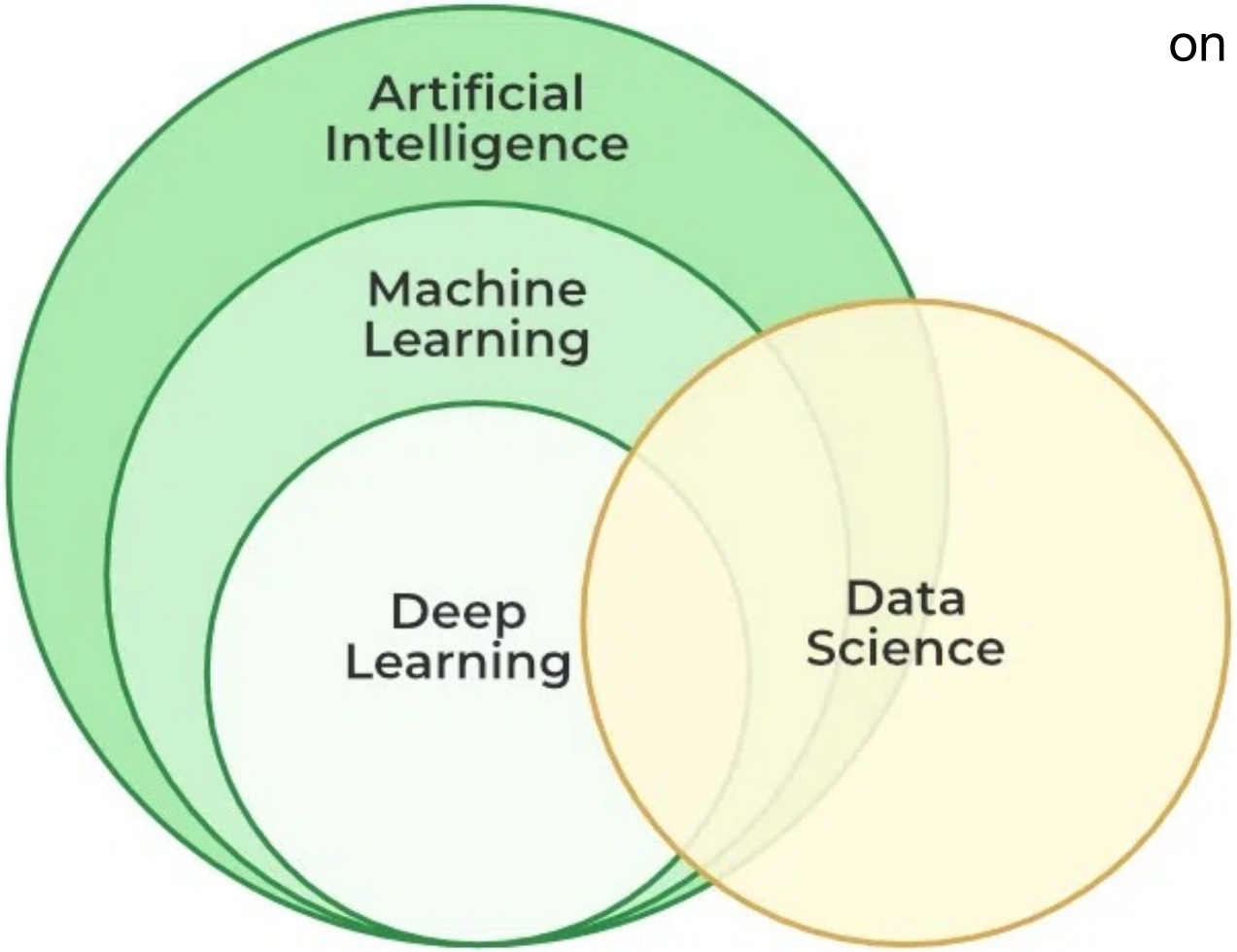
What is Machine Learning



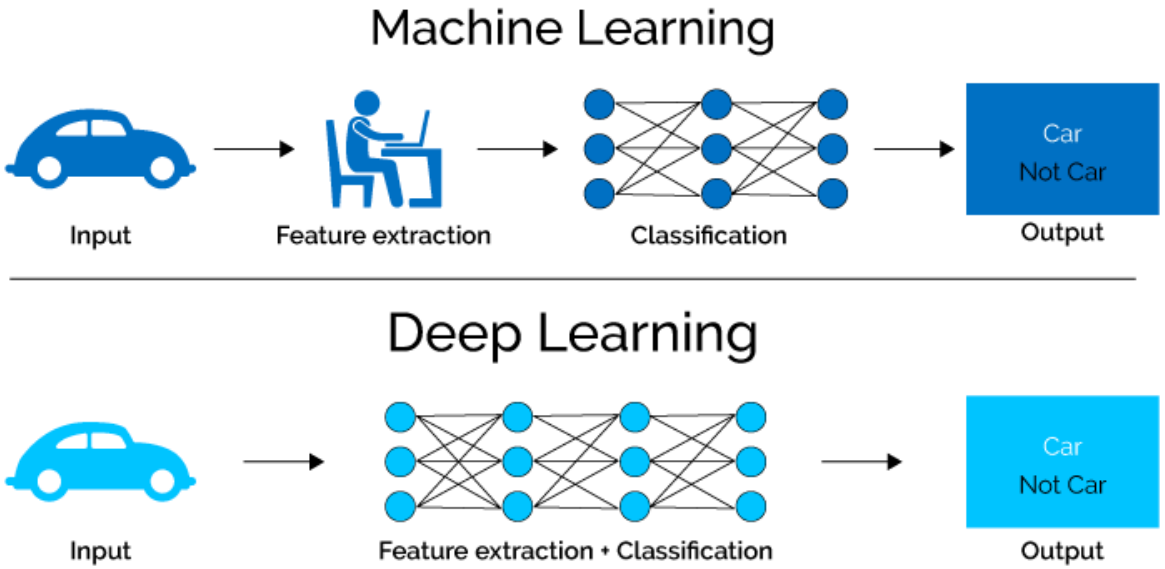
- **Machine Learning:** automatically learn insights and recognize patterns from data.



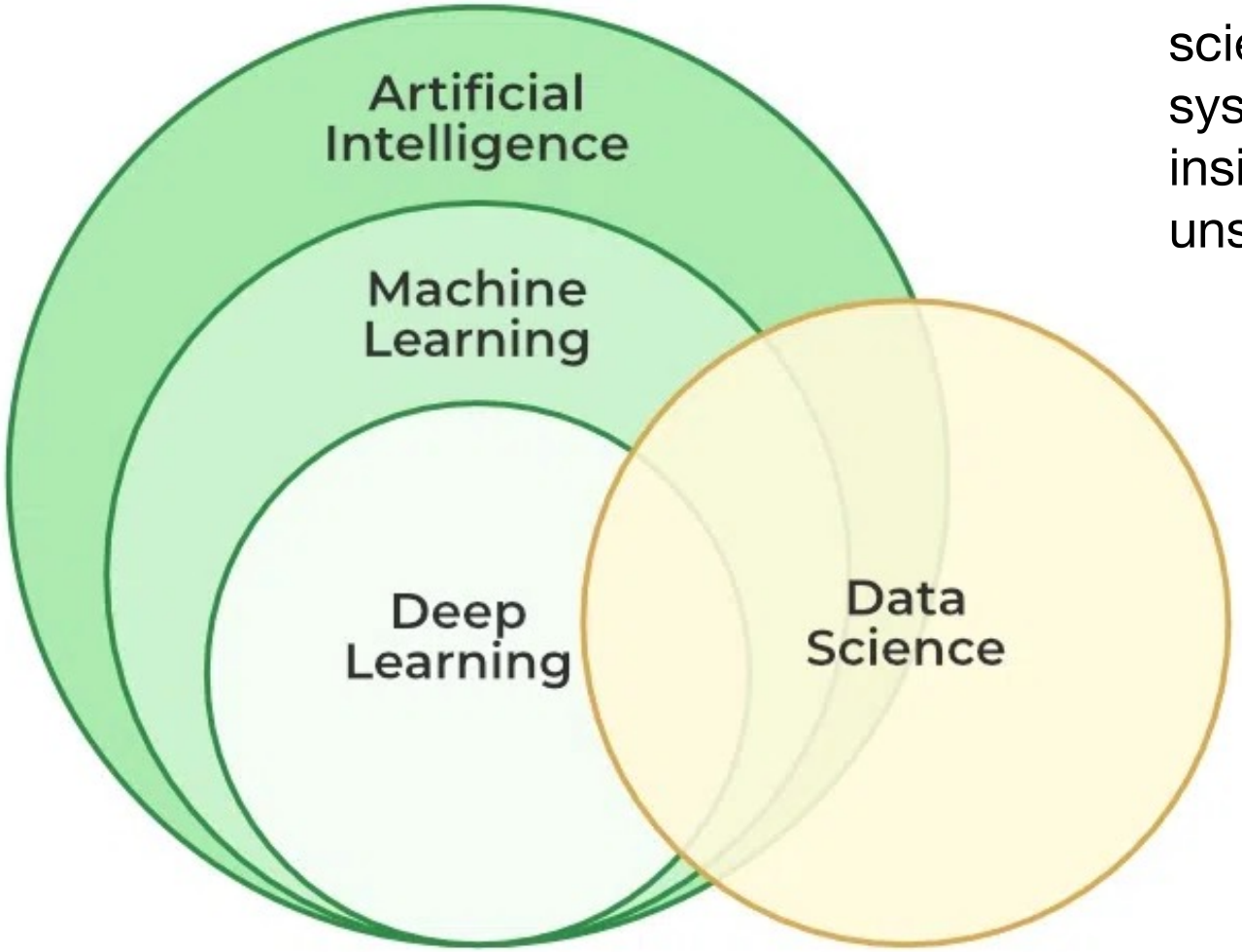
What is Machine Learning



- **Deep Learning:** complex neural networks trained on large amount of unstructured data.



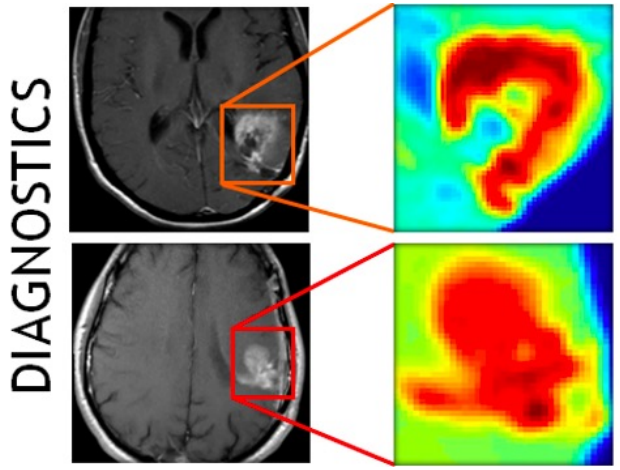
What is Machine Learning

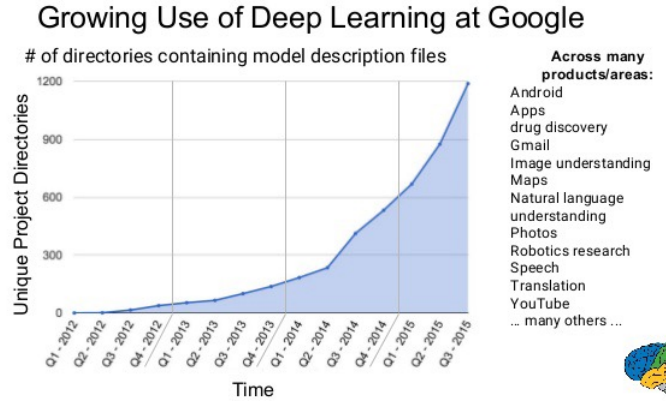


- **Data Science:** uses statistics, scientific computing, scientific methods, processes, algorithms and systems to extract or extrapolate knowledge and insights from potentially noisy, structured, or unstructured data.

Some applications …

- Natural Language Processing
- Speech and handwriting recognition
- Object recognition and computer vision
- Fraud detection
- Financial market analysis
- Search engines
- Spam and virus detection
- Medical diagnosis
- Robotics control
- Automation: energy usage, systems control, video games, self-driving cars
- Advertising
- Data Science



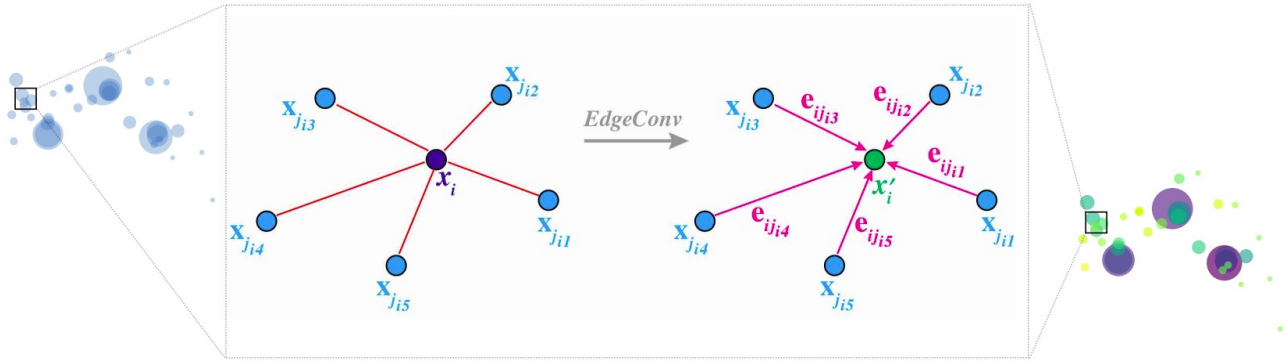


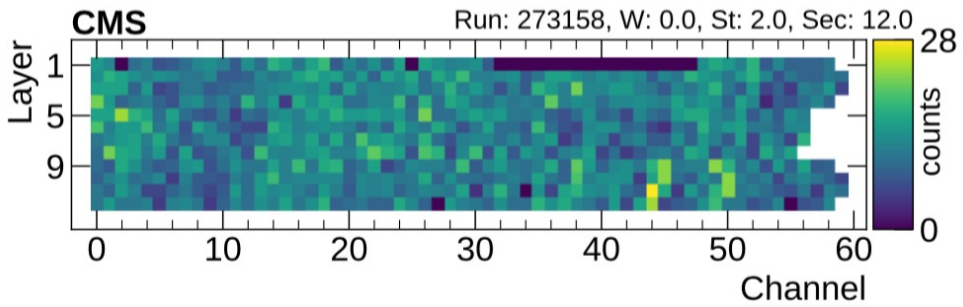
… in HEP

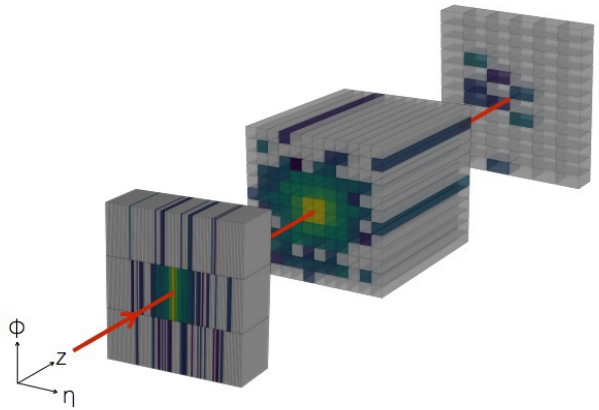
- Particle tagging
- Signal classification
- Anomaly detection
- Fast simulation
- Simulation-based inference
- Design optimisation
- Uncertainty mitigation

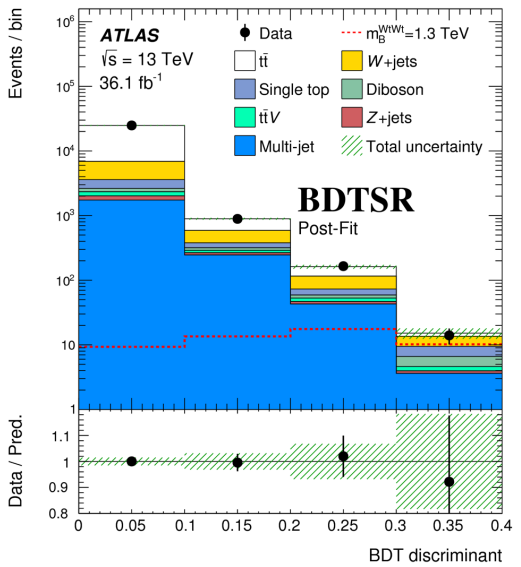
… and more

https://iml-wg.github.io/HEPML-LivingReview/









A rapidly evolving field

ChatGPT

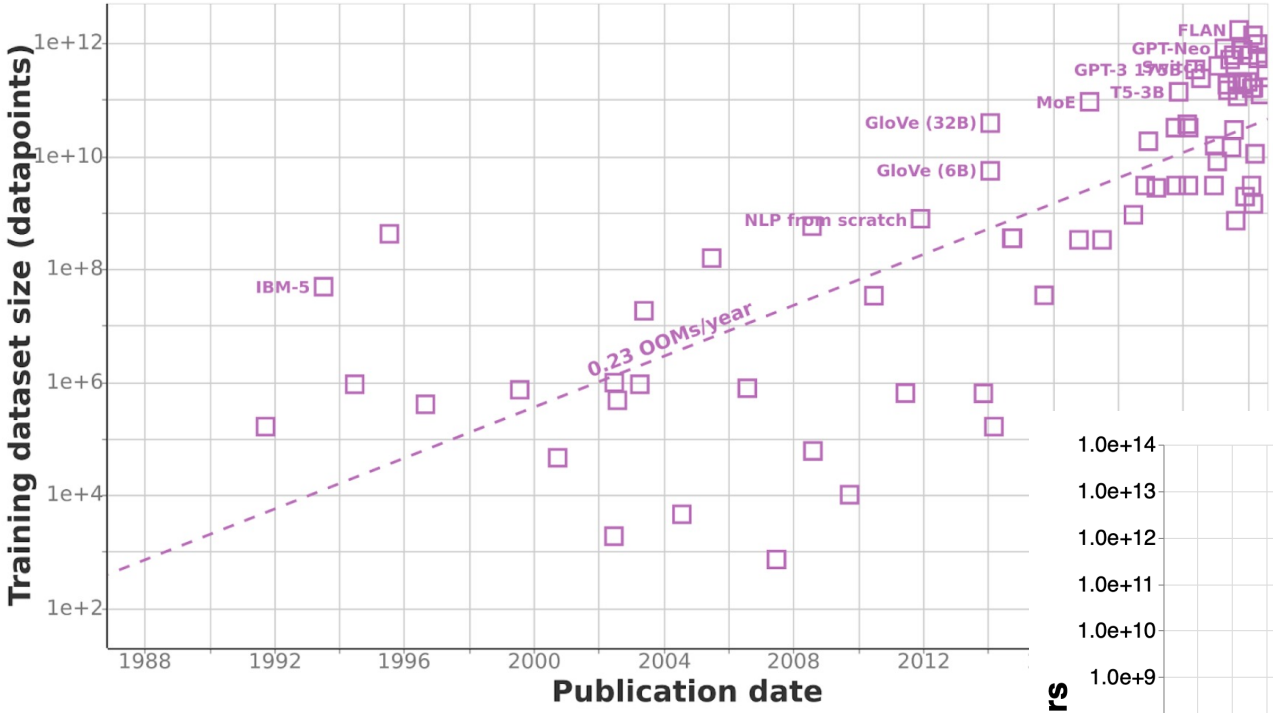
Stable Diffusion

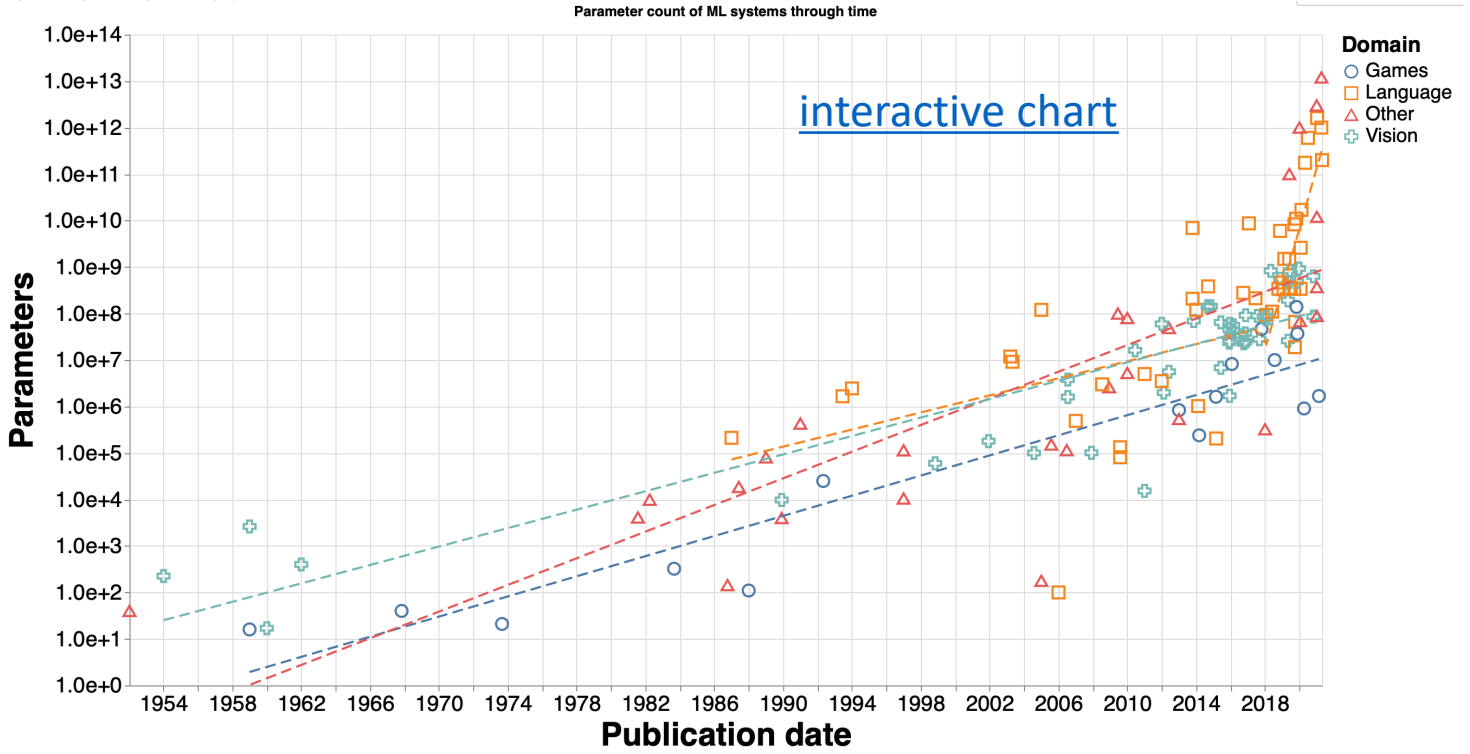
GitHub Copilot

Prompt:

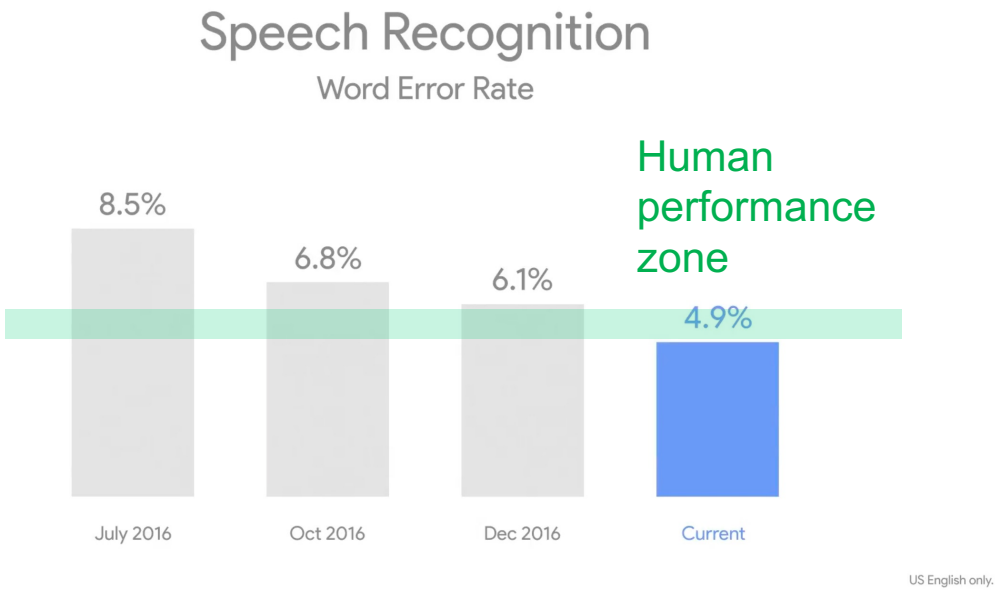
street style photo of a woman selling pho at a Vietnamese street market, sunset, shot on fujifilm

Increasing sizes and complexity …



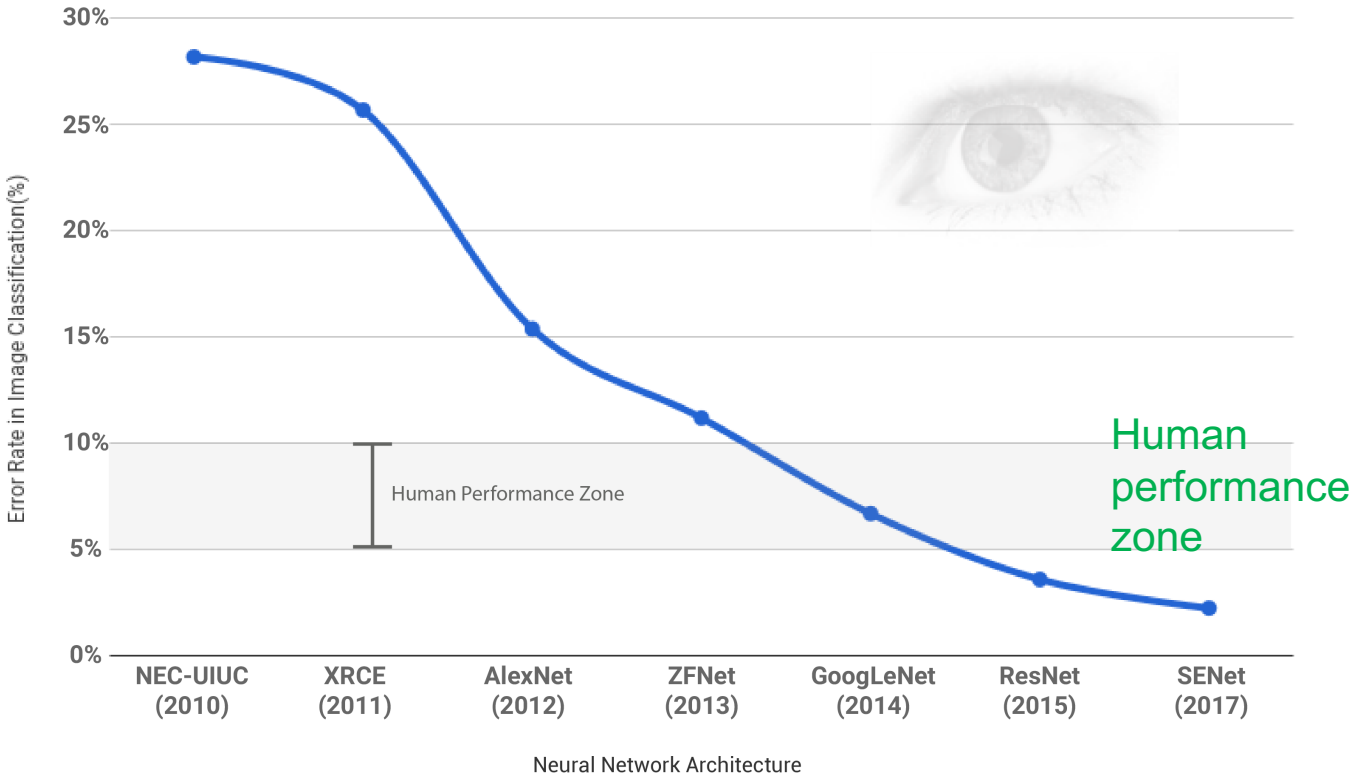


… and performance



Human performance zone

2017 Google results



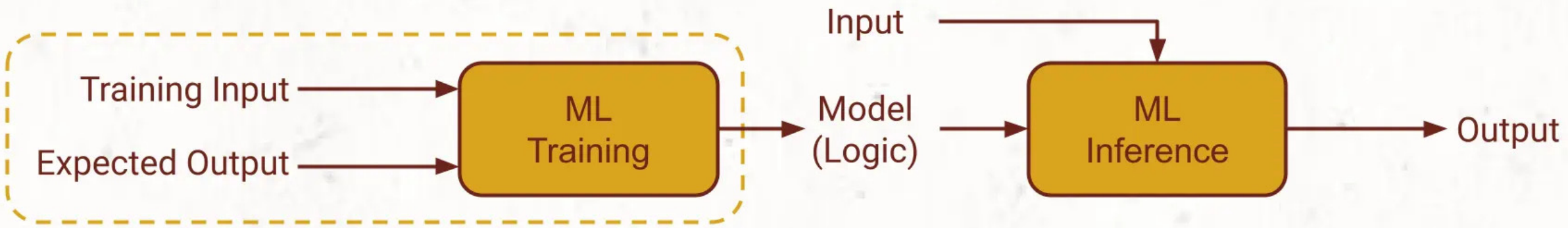
Human performance zone

https://arxiv.org/pdf/1409.0575.pdf

Let's start from basics: learning algorithms

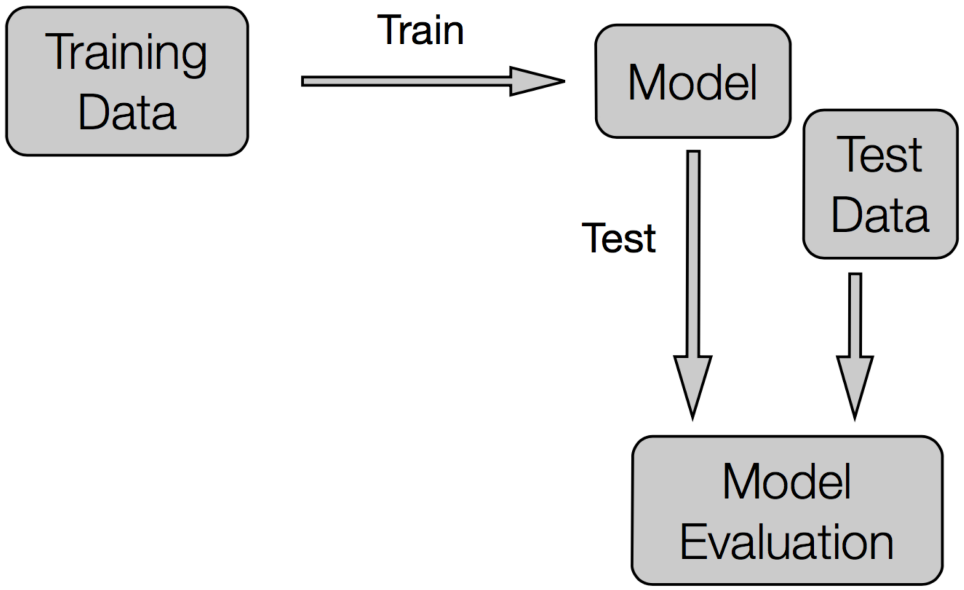
Traditional Programs: Define algo/logic to compute output

Machine Learning: Learn model/logic from data



Machine learning models

- Key element in machine learning is a **mathematical model**
- A mathematical characterization of system(s) of interest, typically via **random variables**
- Chosen model **depends on the task** / available data



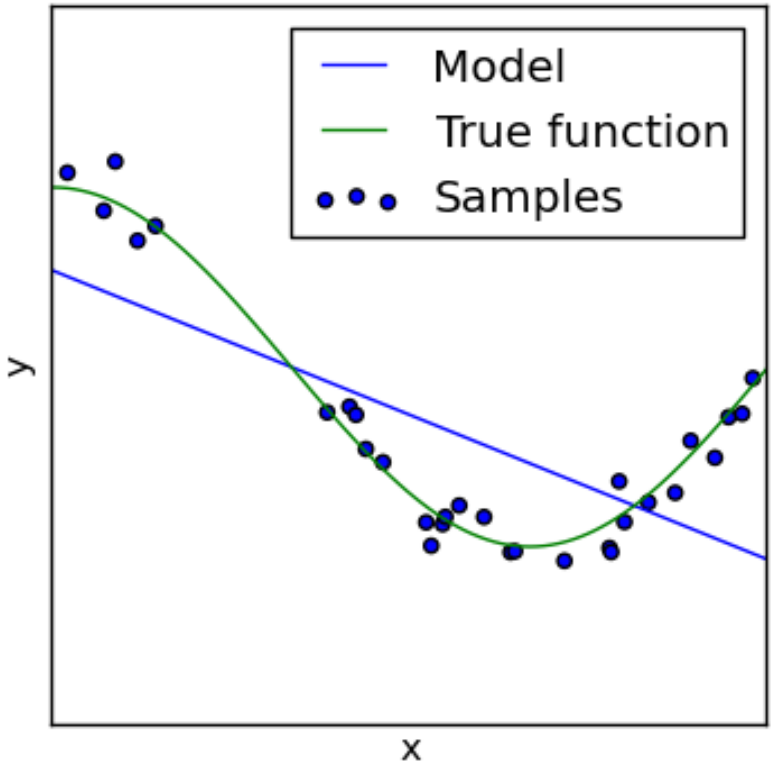
- **Learning**: estimate statistical model from data
- **Prediction and Inference**: using statistical model to make predictions on new data points and infer properties of system(s)

Parametric vs nonparametric models

Parametric: “fixed-size” models that do not “grow” with the data

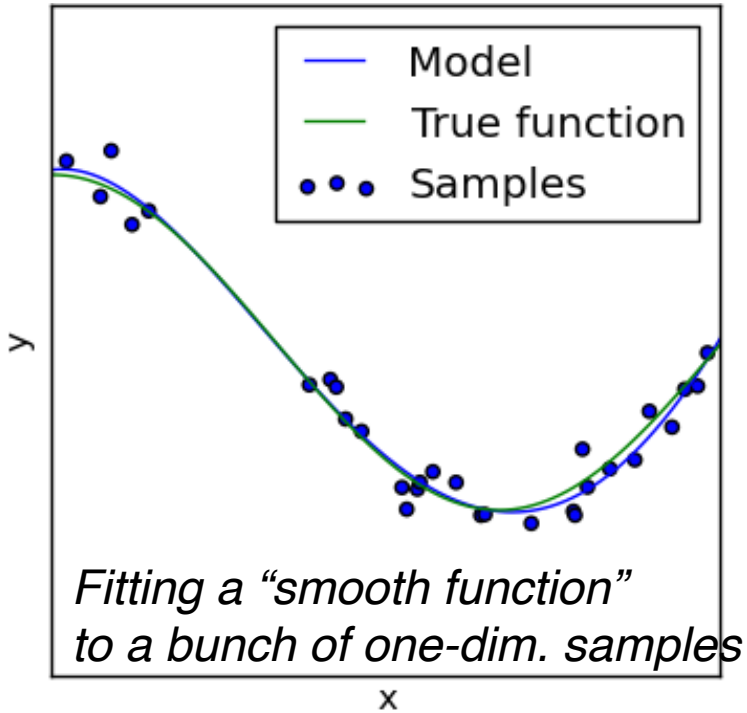
- More data just means you learn/fit the model better

Fitting a simple line (2 params)
to a bunch of one-dim. samples



Nonparametric: models that grow with the data

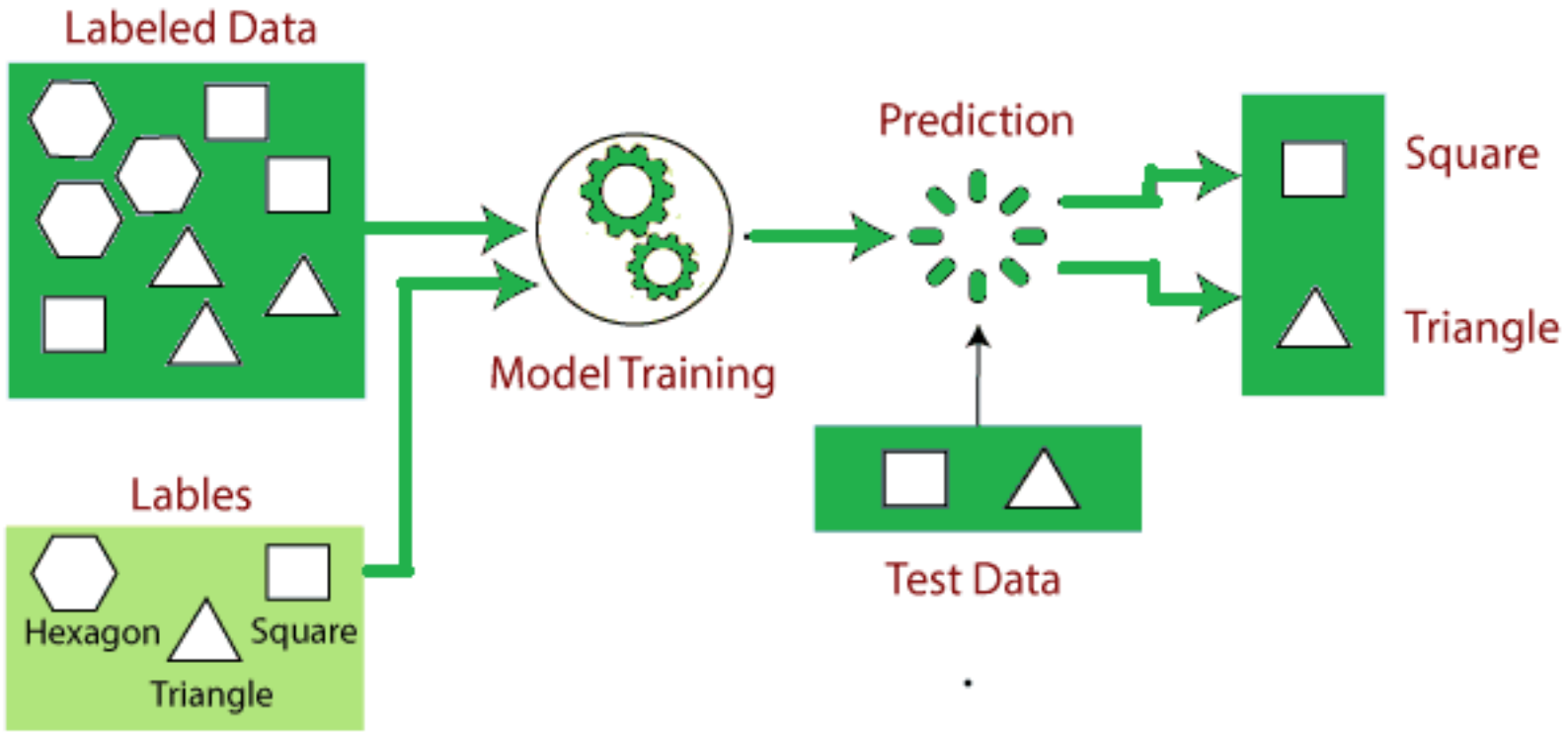
- More data means a more complex model



Fitting a “smooth function”
to a bunch of one-dim. samples

How: supervised learning

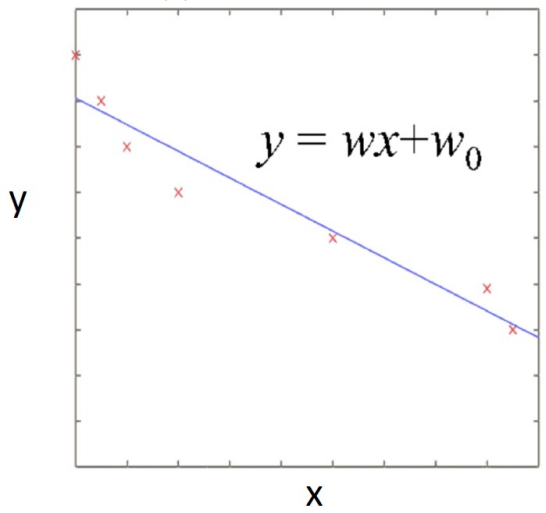
- The desired output (target) is known at training time.
  - calculate an error based on target output and algorithm output
  - use error to make corrections by updating the weights.
- Given N examples with features $\{x_i\}$ and targets $\{y_i\}$, learn function mapping **h(x)=y**



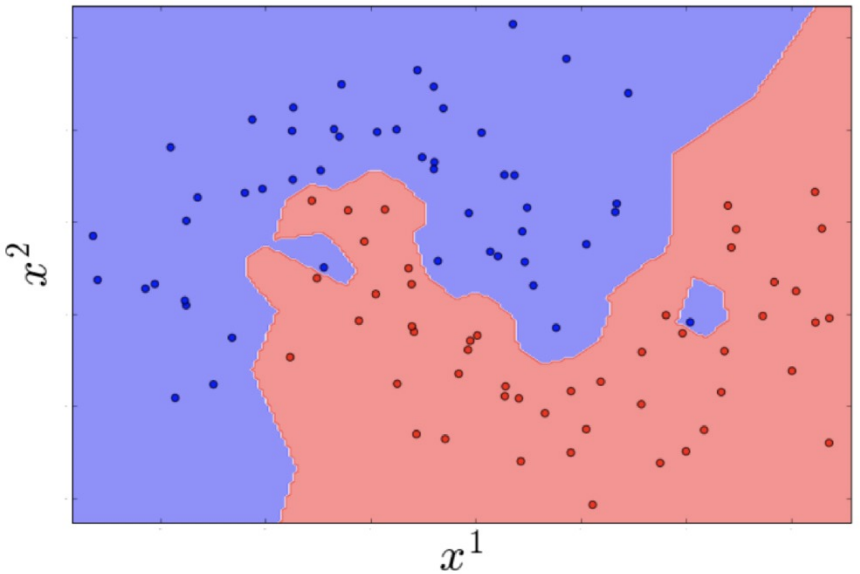
How: supervised learning

- The desired output (target) is known at training time.
  - calculate an error based on target output and algorithm output
  - use error to make corrections by updating the weights.
- Given N examples with features $\{x_i\}$ and targets $\{y_i\}$, learn function mapping **h(x)=y**

- **Regression:**
  - Y = Real Numbers

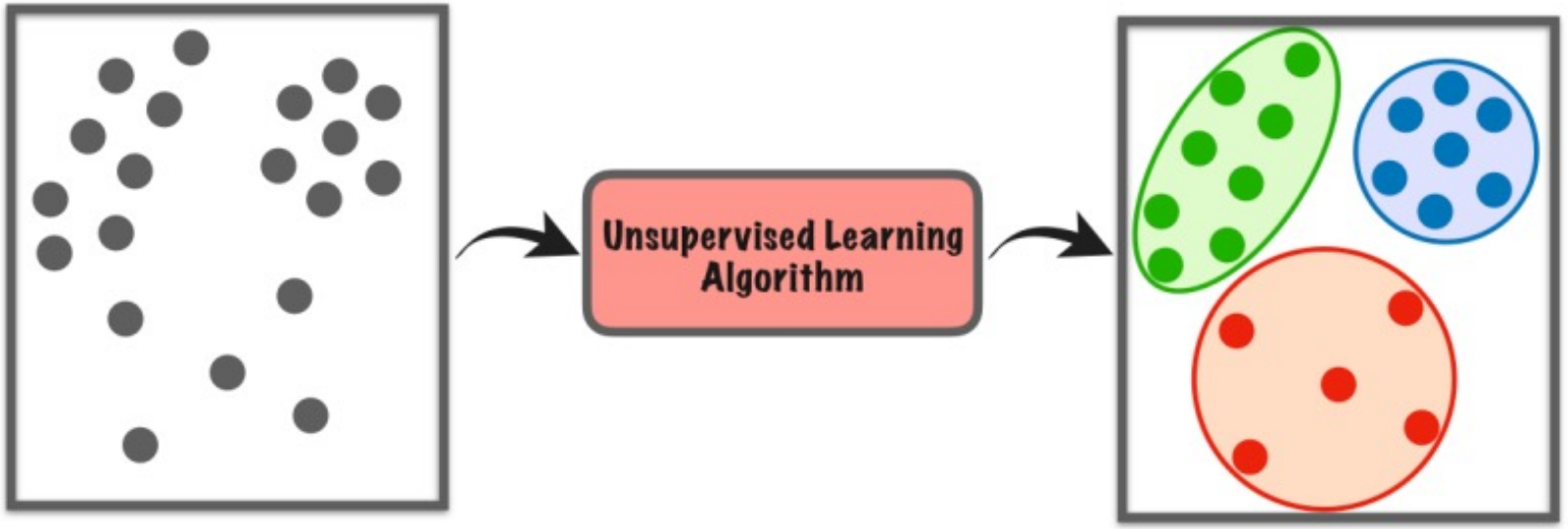


- **Classification:**
  - Y is a finite set of labels



How: unsupervised learning

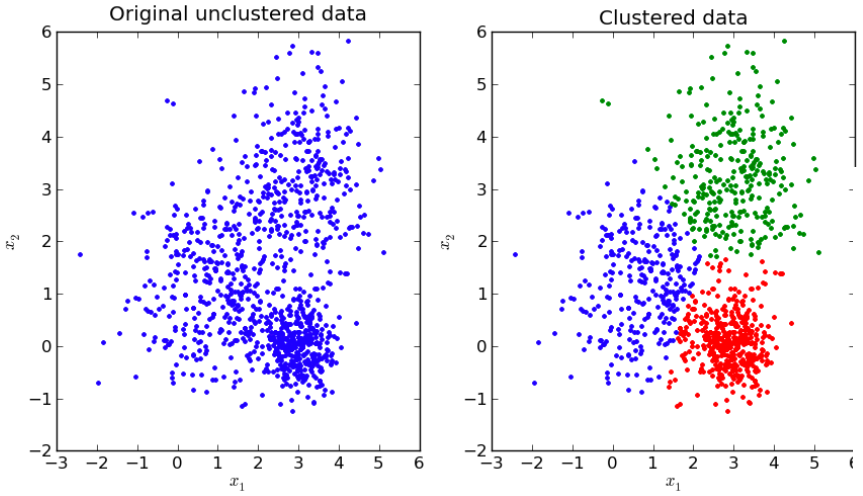
- No target output is used at training time
  - The ML algorithm finds pattern within the inputs
- Given some data D={x_i}, but no labels → find structure in the data



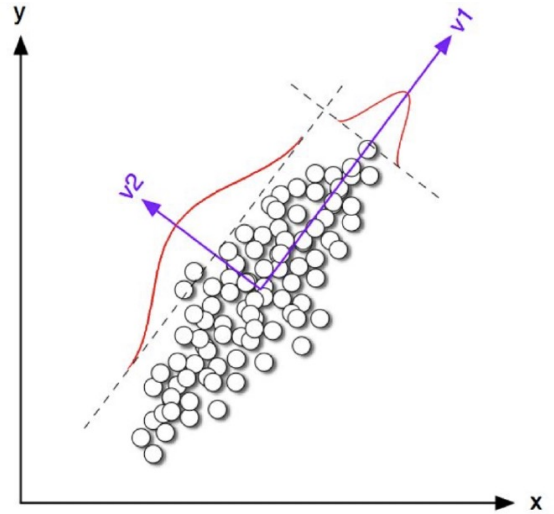
How: unsupervised learning

- No target output is used at training time
  - The ML algorithm finds pattern within the inputs
- Given some data D={x_i}, but no labels → find structure in the data

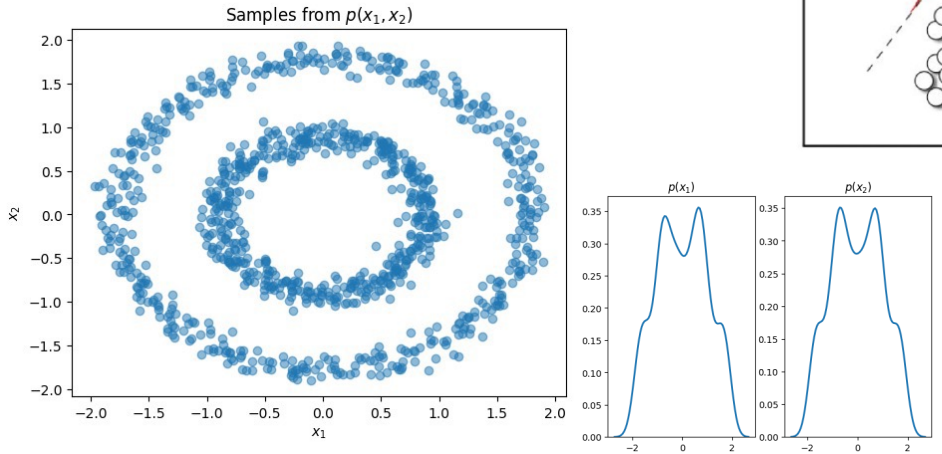
- **Clustering:**
  partition the data into groups

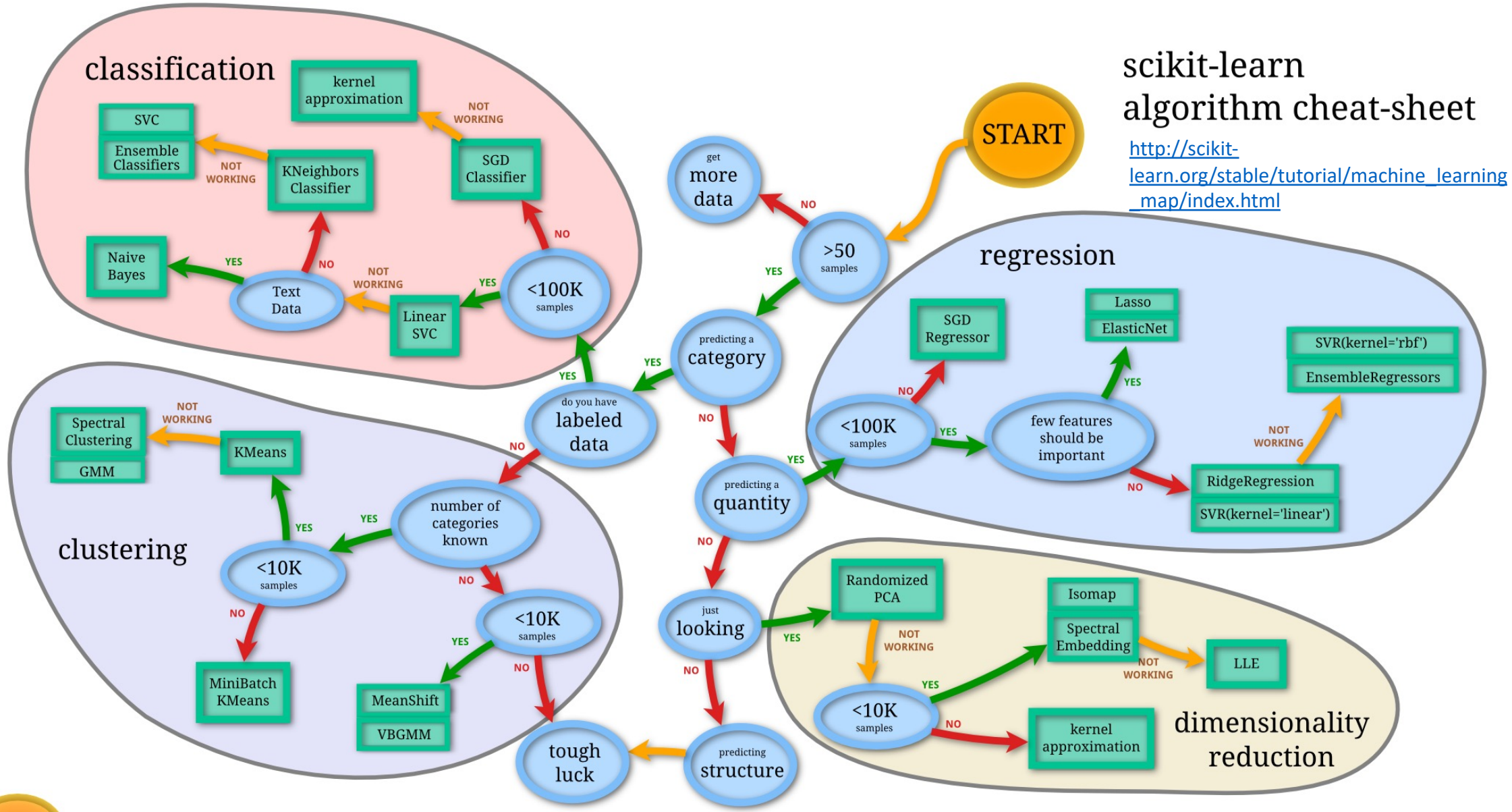


- **Dimensionality reduction:** find a low dimensional (less complex) representation of the data with a mapping Z=h(X)



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




scikit-learn
algorithm cheat-sheet
http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
START
get more data
NO
>50 samples
YES
predicting a category
YES
NO
do you have labeled data
YES
NO
predicting a quantity
YES
NO
just looking
NO
predicting structure
tough luck
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
NO
Text Data
NOT WORKING
Linear SVC
YES
<100K samples
YES
clustering
Spectral Clustering
GMM
NOT WORKING
KMeans
YES
number of categories known
YES
<10K samples
NO
MiniBatch KMeans
NO
<10K samples
YES
MeanShift
VBGMM
NO
regression
SGD Regressor
NO
<100K samples
YES
few features should be important
YES
Lasso
ElasticNet
NO
SVR(kernel='rbf')
EnsembleRegressors
NOT WORKING
RidgeRegression
SVR(kernel='linear')
dimensionality reduction
Randomized PCA
YES
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
YES
<10K samples
NO
kernel approximation
Back

Terminology: ML

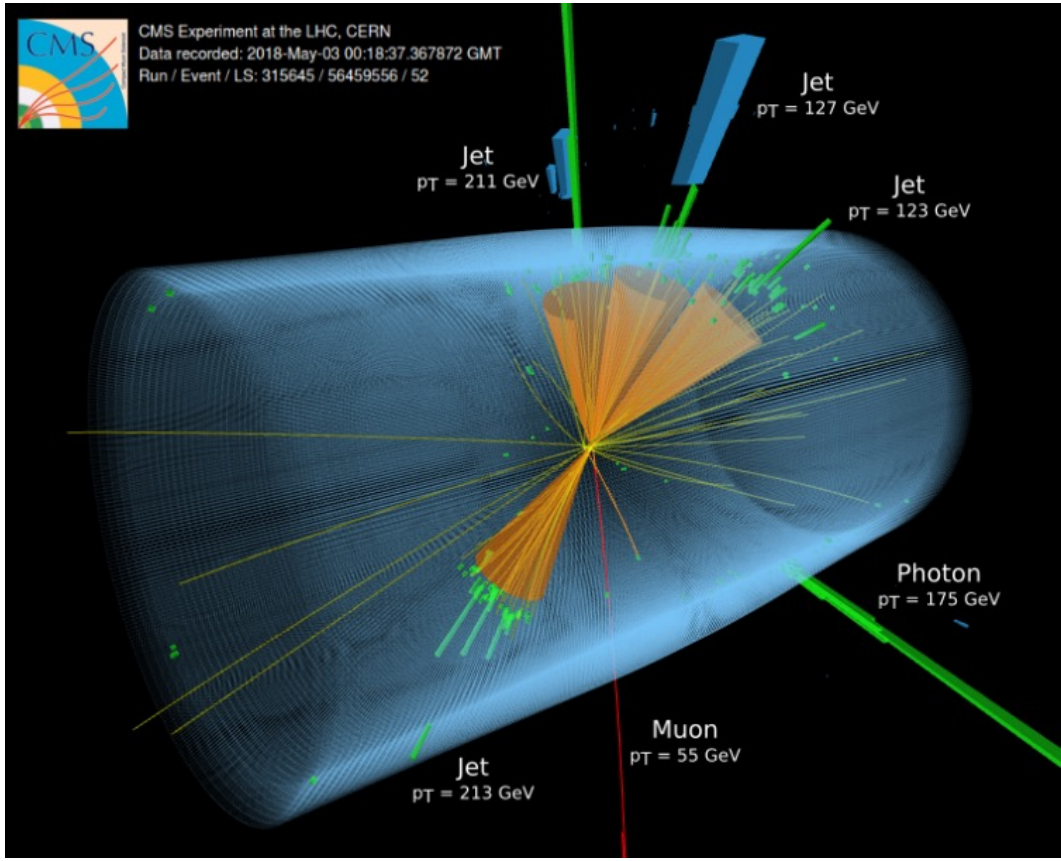
1. **Instance/example:** The values of one row of features and possibly a label.
2. **Dataset**: A collection of instances
3. **Feature**: Input variable to your ML model

Features			Label
Temperature	Humidity	Pressure	Test score
15	47	998	Good
19	34	1020	Excellent
18	92	1012	Poor

Terminology: HEP

(High Energy) Particle Physics

1. **Event:** All information collected during a collision inside a detector, or reproduced from a Monte Carlo simulation of such collisions (**instance/example**)
2. **Sample**: A collection of events, a **dataset**.
3. **Variable**: A property of the event or of one of its constituents (**feature**)
4. **Cut:** To cut on a variable is to apply a threshold on this variable and keep only events satisfying this condition. A cut-based analysis is applying such thresholds on several variables to select events.
5. **Event weight**: In high-energy physics events usually have an associated weight, which depends on how many events were generated (relating to the process cross section and collected luminosity) and various corrections applied to simulations to account for differences between data and Monte Carlo predictions (jet energy scale or object identification efficiency are such weights)



https://doi.org/10.1142/9789811234033_fmatter

Let’s dig into supervised training: how does it work?

Disclaimer: next slides are mostly taken from

- “Introduction to Machine Learning and Deep Learning, Micheal Kagan https://indico.cern.ch/event/1293858/”
- CS229, Ng, Stanford University http://cs229.stanford.edu/

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:

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

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:
  - $x \sim N(\mu, \sigma) \rightarrow p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$

Supervised training

- Given N examples with features $\{x_i \in \mathcal{X}\}$ and **targets** $\{y_i \in \mathcal{Y}\}$, learn function mapping **h(x)=y**
  - **Classification**: $\mathcal{Y}$ is a finite set of **labels** (i.e. classes)

$\mathcal{Y} = \{0, 1\}$ for **binary classification**,
encoding classes, e.g. Higgs vs Background

$\mathcal{Y} = \{c_1, c_2, \ldots c_n\}$ for **multi-class classification**

represent with "**one-hot-vector**"

$\rightarrow\ y_i = (0, 0, \ldots, 1, \ldots 0)$

were k^{th} element is 1 and all others zero for class c_k

Supervised training

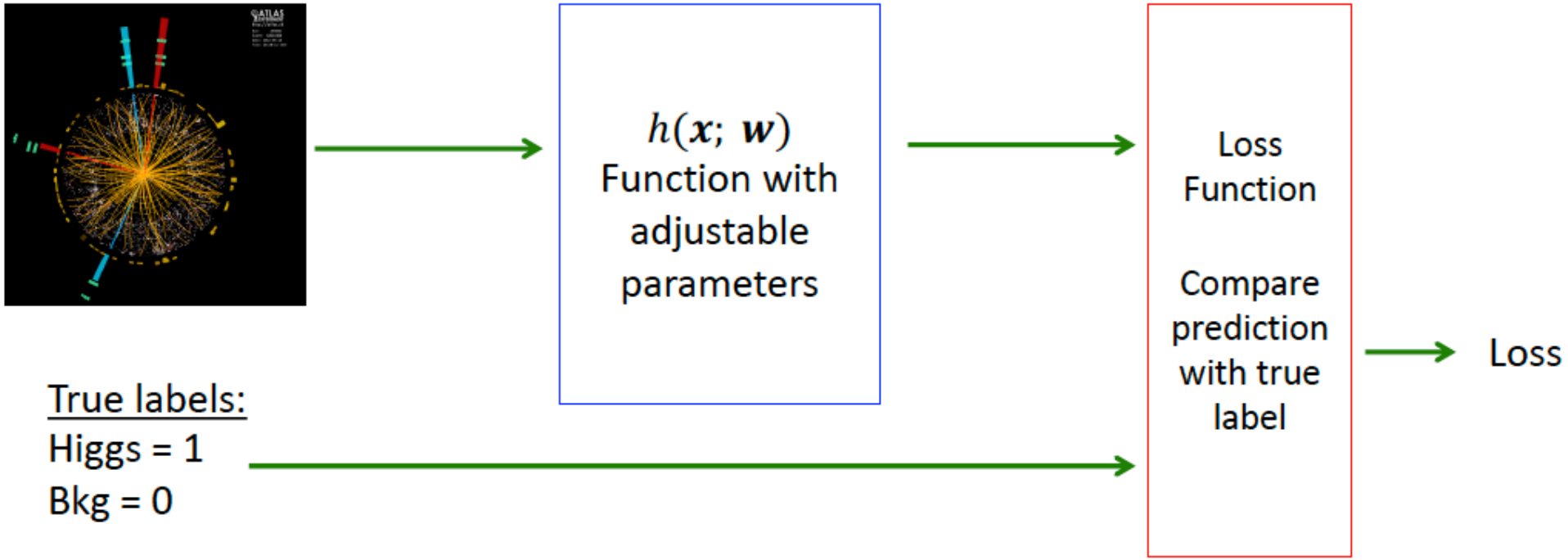
- Given N examples with features $\{\mathbf{x}_i \in \mathcal{X}\}$ and **targets** $\{y_i \in \mathcal{Y}\}$, learn function mapping **h(x)=y**
  - **Classification**: $\mathcal{Y}$ is a finite set of **labels** (i.e. classes)
  - **Regression**: $\mathcal{Y}$ = Real Numbers
- Often these are **discriminative models**, in which case we model:

$$h(x) = p(y \mid x)$$

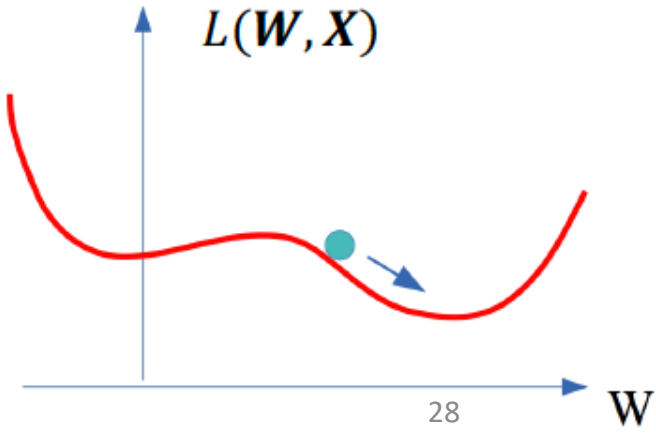
- Sometimes use **generative models**, estimate joint distribution p(y, x)
  - Could estimate class conditional density p(x | y) and prior p(y)
  - Use Bayes theorem to then compute:

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

Supervised training: how does it work?



- Initialization of the parameters w
- Design a Loss function
  - Differentiable with respect to model parameters
- Find best parameters which minimize loss



Supervised training: a simple regression example

- Suppose we have a dataset giving the living areas, number of bedrooms and prices of 47 houses from Portland, Oregon.
- We have to solve a regression problem: infer the price of the other houses in Portland.

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

- $\boldsymbol{x} \in \mathbb{R}^2$
- $(\boldsymbol{x}(i);\ y(i))$ is our training sample
- i = 1 .. N

Let's set an hypothesis on the function $h: X \to Y$ to be linear:

$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

θ are called **parameters** or **weights**

Supervised training: a simple regression example

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$$= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 = \sum_{i=0}^{d} \theta_i x_i =$$

$$= \theta^T \boldsymbol{x}$$

Problem: how do I set a method for learning the paramenters θ?

Solution: define a function measuring the distance between labels (y) and predictions (h(x))

This function is called **cost function**

In this simple example let's use the familiar **least-squares cost function**

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2.$$

Supervised training: a simple regression example

Cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Now I need an algorithm that starts with some "**initial guess**" for θ, and that repeatedly changes θ to make $J(\theta)$ smaller, until hopefully we converge to a value of that **minimizes** $\mathbf{J}(\boldsymbol{\theta})$

Let's use the **gradient descent algorithm**, which starts from some initial θ, and performs the update:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta).$$

α is called **learning rate**, it is an hyperparameter of my model.

Supervised training: a simple regression example

Model:

$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Gradient descent:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

Let's work out the derivatives in the case of only one example/instance:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{d} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y)\, x_j
\end{aligned}$$

We obtained a new update rule, called LMS (least mean squares):

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Supervised training: a simple regression example

Model:

$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

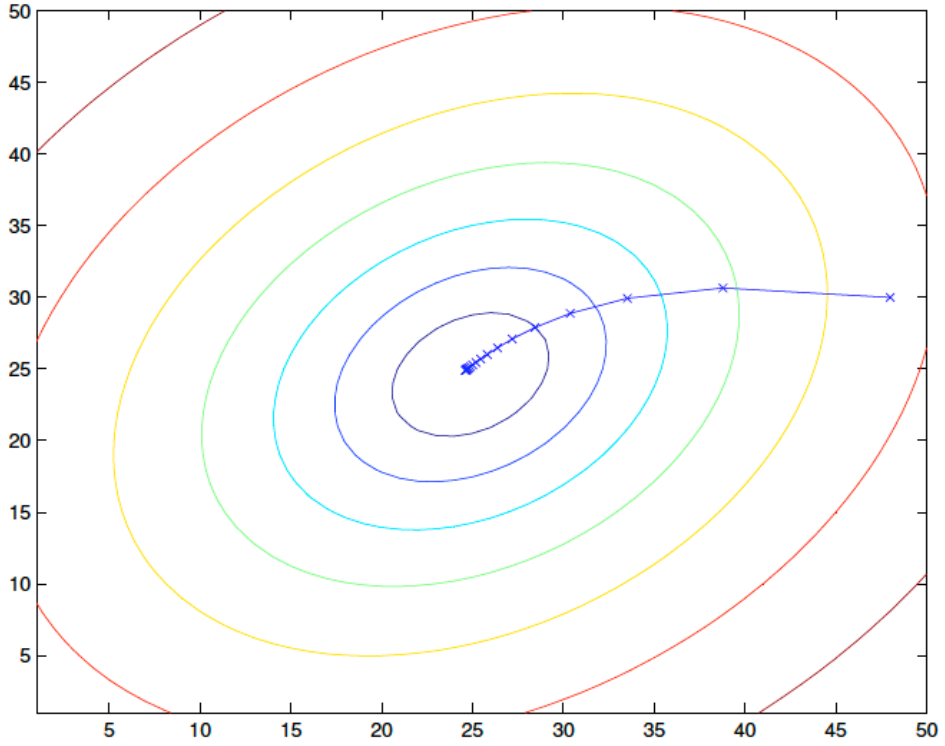
Gradient descent:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta).$$

Batch gradient descent LMS update rule:

$$\theta := \theta + \alpha\sum_{i=1}^{n}\left(y^{(i)} - h_\theta(x^{(i)})\right)x^{(i)}$$

Error term:



if we are encountering a training example on which our prediction nearly matches the actual value of y(i), then we find that there is little need to change the parameters; in contrast, a larger change to the parameters will be made if our prediction h(x(i)) has a large error (i.e., if it is very far from y(i)).

Supervised training: a simple regression example

Model:

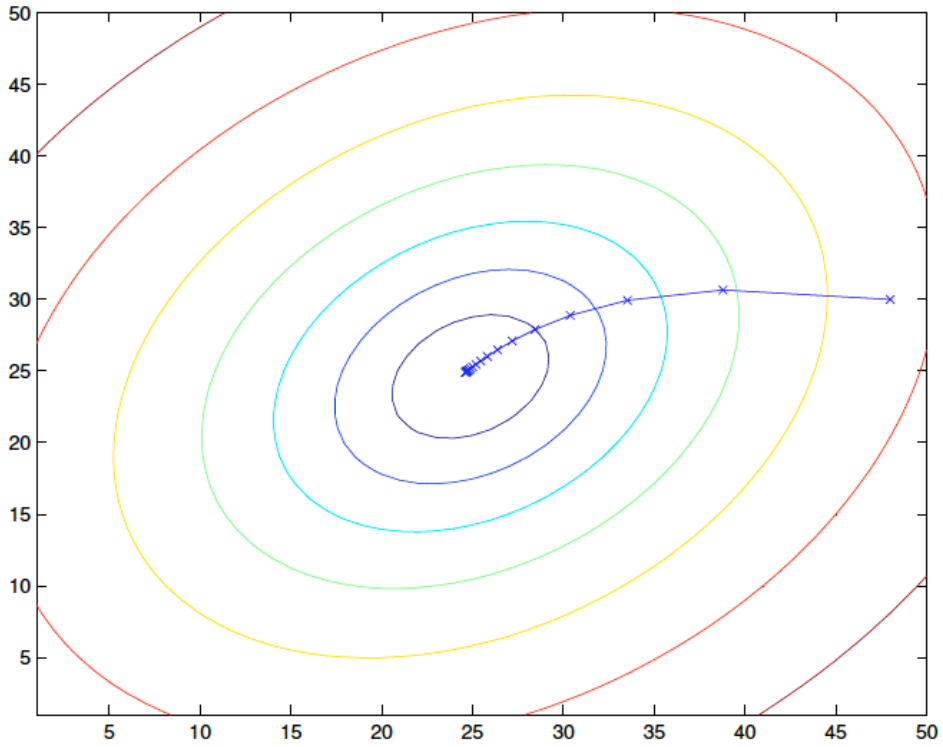
$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Cost function:

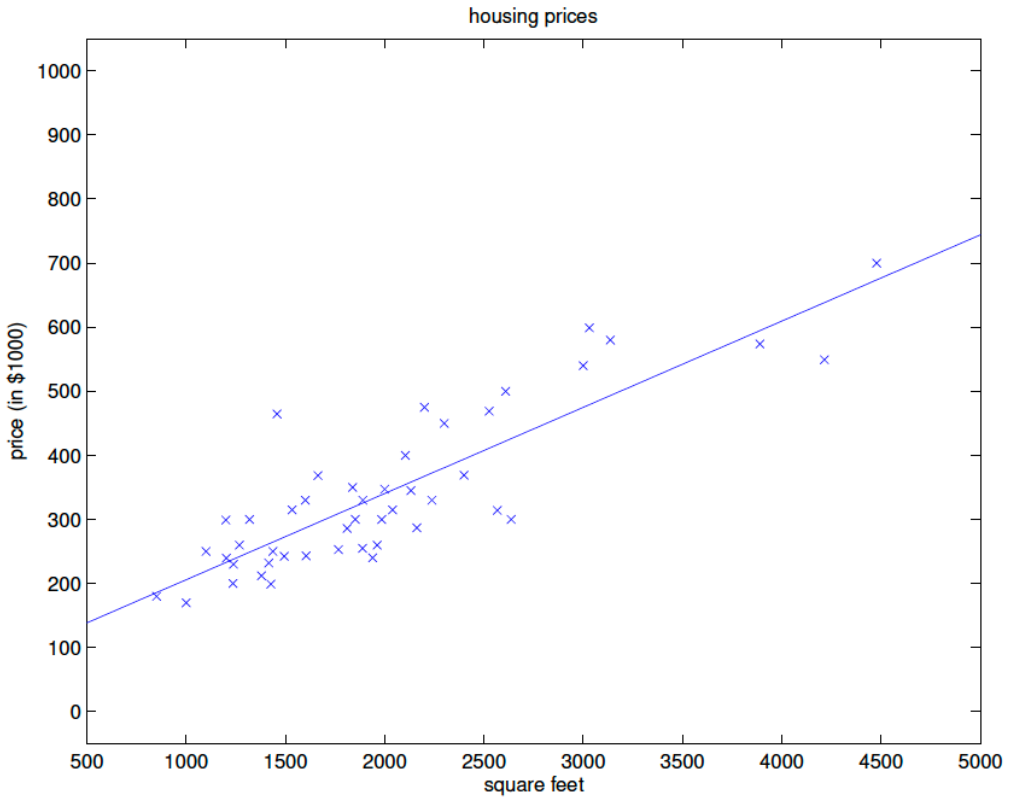
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Batch gradient descent:

$$\theta := \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})\right) x^{(i)}$$



Cost function minimisation



Fit θ to predict housing price as a function of living area

Supervised training: a simple regression example

Model:

$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Batch gradient descent:

$$\theta := \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})\right) x^{(i)}$$

The batch gradient descent goes through the whole dataset at each step! An alternative is the **stochastic (or incremental) gradient descent**, that updates the parameters according to the gradient of the error with respect to that single training example only.

Preferred for large datasets

Loop {

for $i = 1$ to n, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}, \quad \text{(for every } j\text{)}$$

}

}

Regression problem: probabilistic interpretation

Let's assume that the differences between labels and predictions (**error terms**) **are independent and gaussian distributed**:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)} \qquad p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

Therefore the probability of y given x and parametrized by θ is:

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

If seen as a function of θ, using the vector notation, this is the **likelihood**.

$$L(\theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)};\theta)$$
$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

It can be seen that maximising the log likelihood is equivalent to minimising the least-squares cost function!

$$\frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})^2$$

Summing up

Build a model:

$$h_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Define a cost (loss) function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Decide a method for minimising the cost function

$$\theta := \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})\right) x^{(i)}$$

This can be seen as maximising the likelihood of our parameters

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X;\theta)$$

More 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 ∈ {0,1}
  - Often used in classification

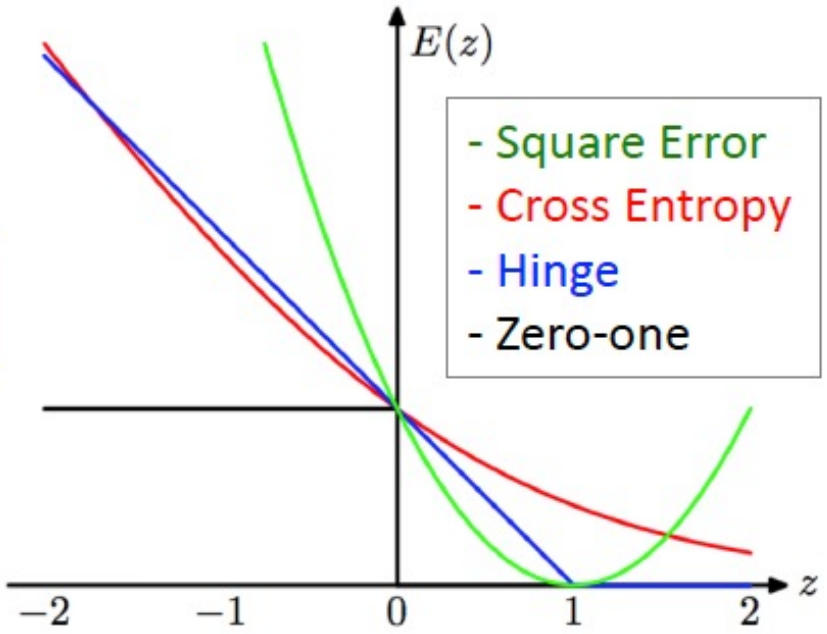
$$L(h(\mathbf{x};\mathbf{w}),y) = -\,y \log h(\mathbf{x};\mathbf{w}) - (1-y)\log(1-h(\mathbf{x};\mathbf{w}))$$

- Hinge Loss:
  - With y ∈ {-1,1}

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

- Zero-One loss
  - With h(x; w) predicting label

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



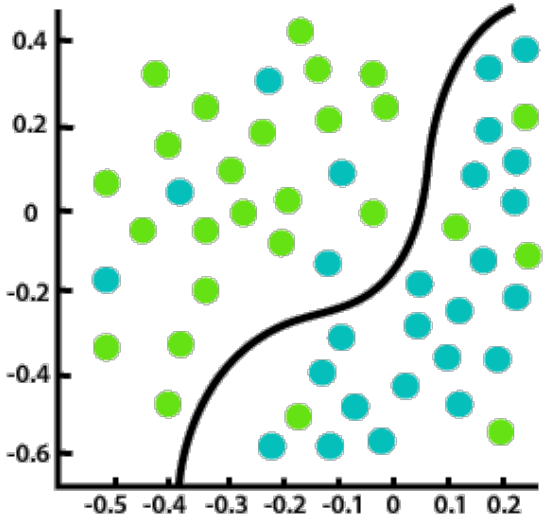
[Bishop]

Supervised training: a simple classification example

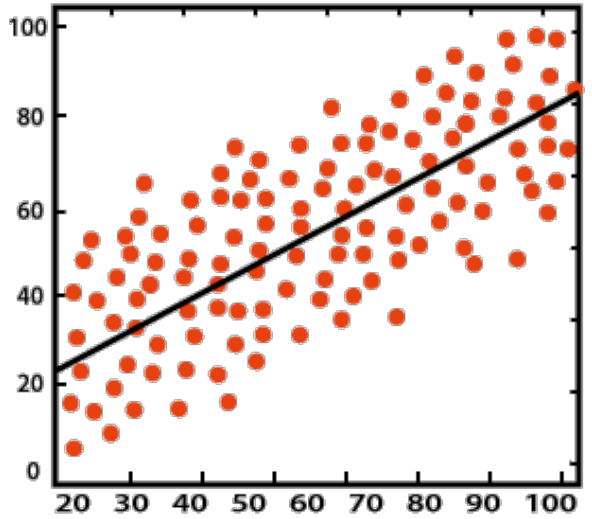
Can we solve a classification problem with linear regression?

In principle yes, but..

- A linear model treats the classes as numbers (0 and 1) and fits the best hyperplane (for a single feature, it is a line) that minimizes the distances between the points and the hyperplane.
- A linear model also extrapolates and gives you values below zero and above one. This is a good sign that there might be a smarter approach to classification.
- Linear models do not extend to classification problems with multiple classes.



Classification



Regression

Supervised training: a simple classification example

Let's change the form of our **model:** $h_\theta(x) = g(\theta^T x) = \dfrac{1}{1 + e^{-\theta^T x}}$

Where $g(z) = \dfrac{1}{1 + e^{-z}}$

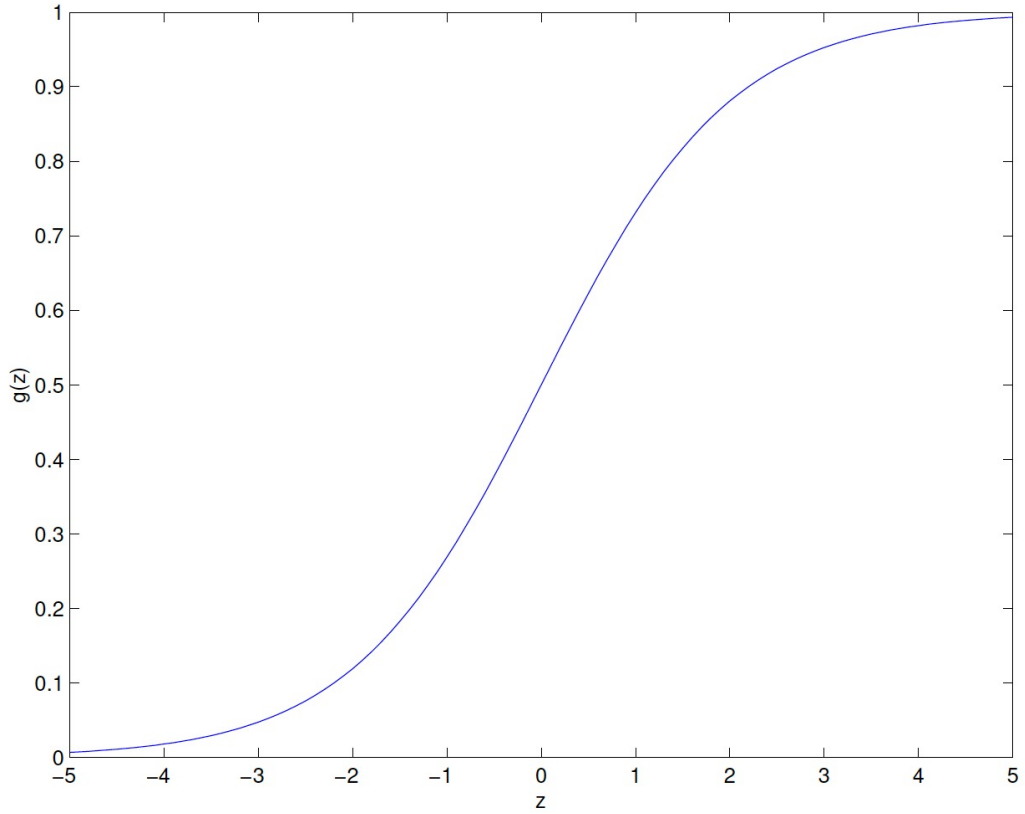
Is called **logistic** or **sigmoid function**

We can interpret $h_\theta(x)$ as a probability. In the simple case of **binary classification**, we can assume:

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

So we can concisely write p(y|x) as Bernoulli random variable:

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$



$$g'(z) = g(z)(1 - g(z))$$

Supervised training: a simple classification example

Therefore the likelihood of the parameters is:

$$L(\theta) = p(\vec{y} \mid X; \theta)$$

$$= \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \prod_{i=1}^{n} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}$$

Again, we want to maximise it. It is easier to maximise the log-likelihood:

$$\ell(\theta) = \log L(\theta) =$$

$$= \sum_{i=1}^{n} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

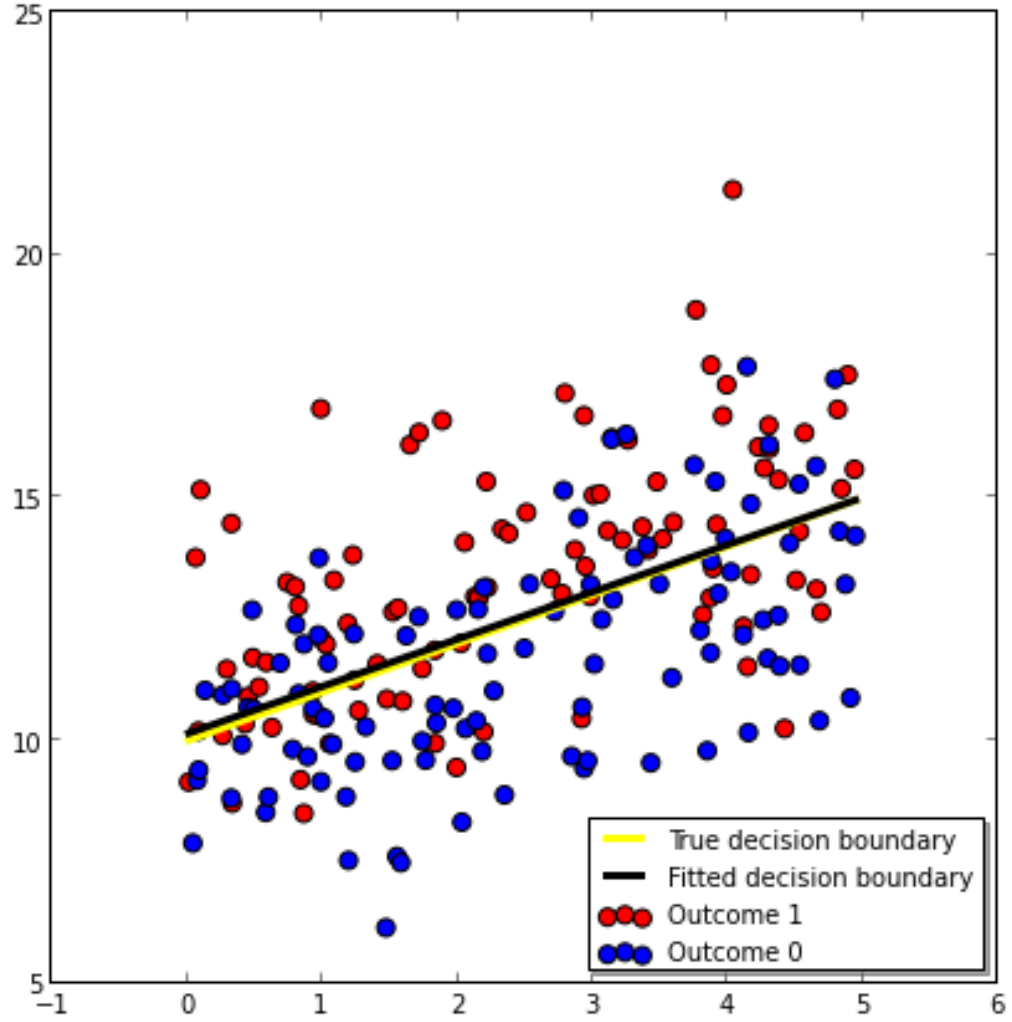
Let's use the gradient descent $\theta := \theta + \alpha \nabla_\theta \ell(\theta)$

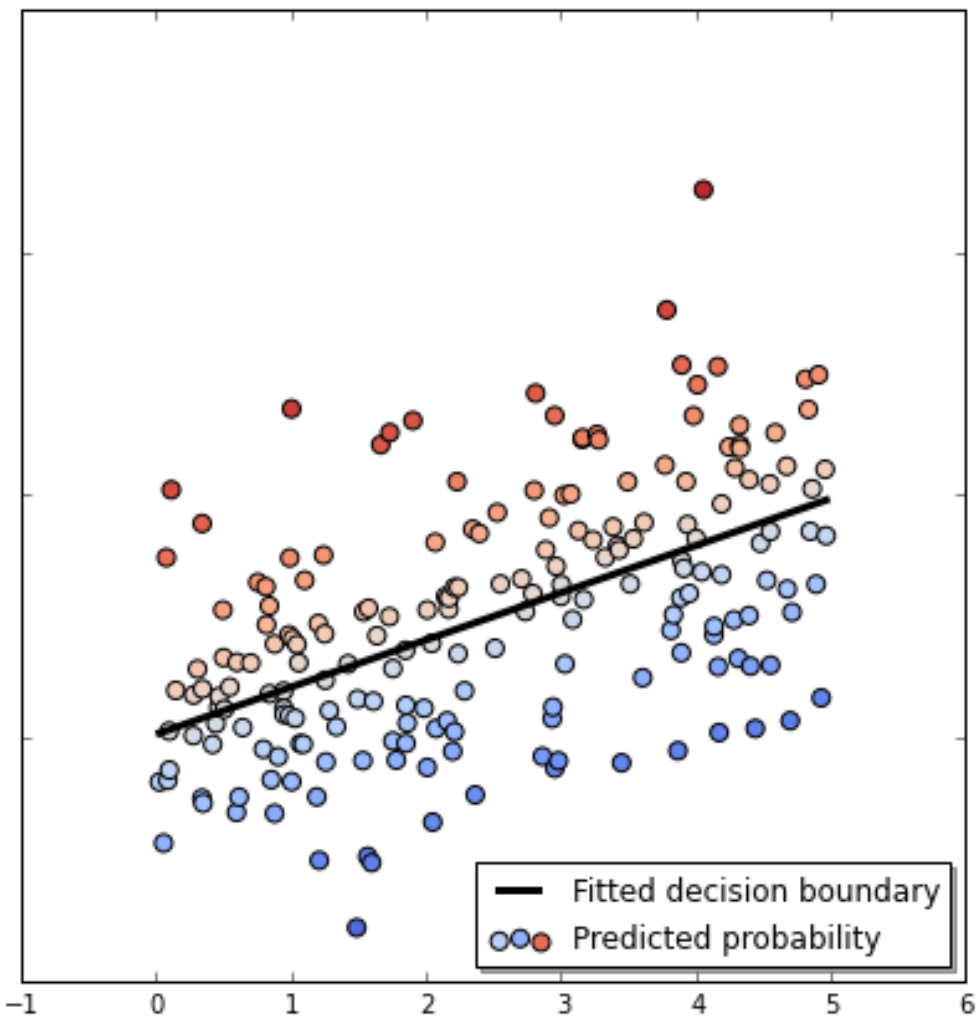
After some math, we find the **update rule for the parameters θ**

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

Note: it looks same as derived in the case of linear regression, but now h is a non-linear function of the features x

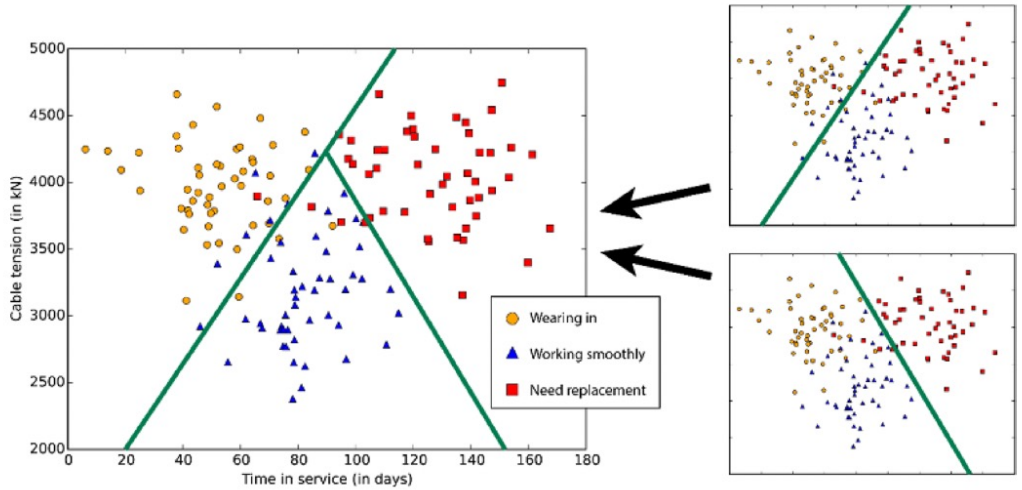
Supervised training: a simple classification example





Supervised training: 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

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

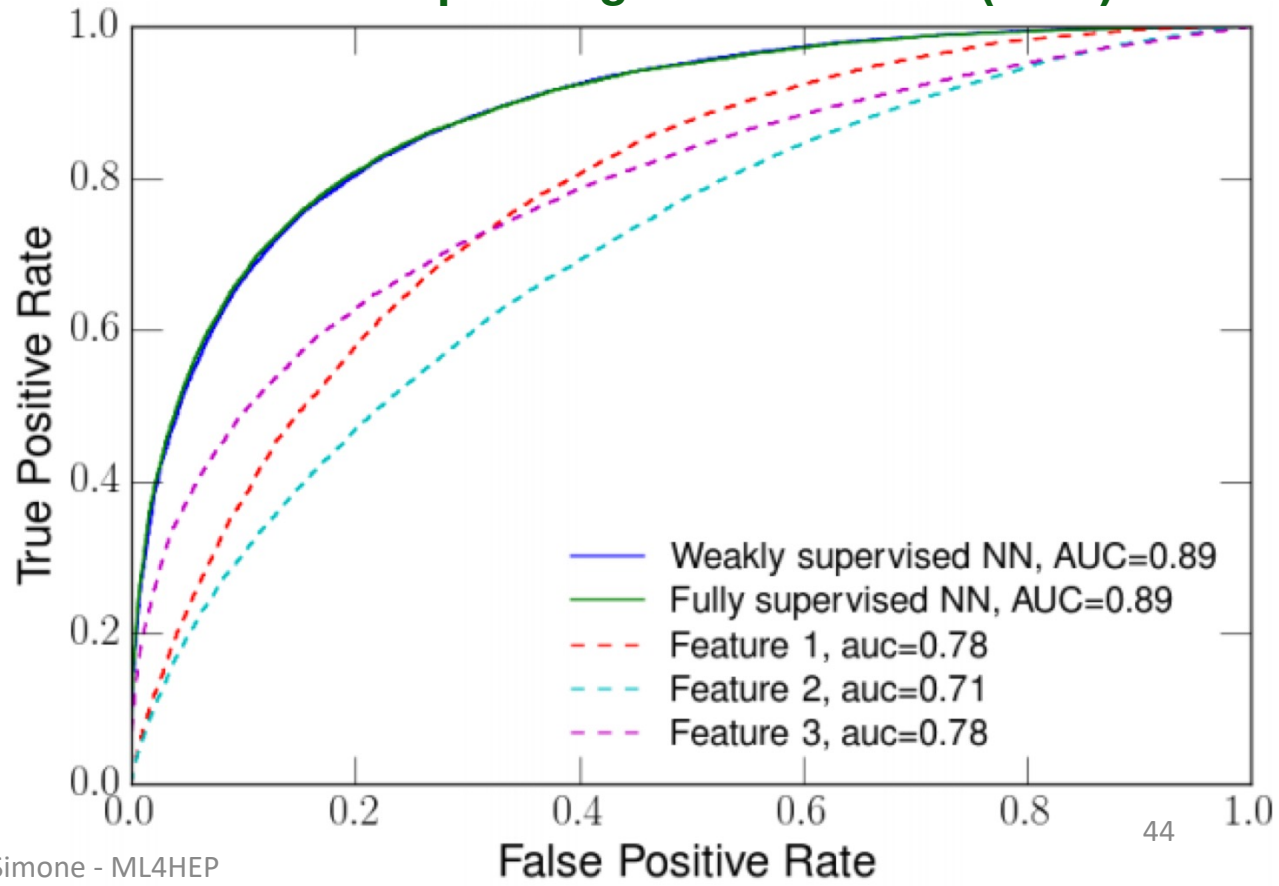
  - Gradient descent for each of the weights $\mathbf{w}_k$

How to evaluate classification performance

		Predicted: Positive	Predicted: Negative
True	Positive	True Positives (TP)	False Negatives (FN)
True	Negative	False Positives (FP)	True Negatives (TN)

CONFUSION MATRIX

Receiver Operating Characteristic (ROC) Curve



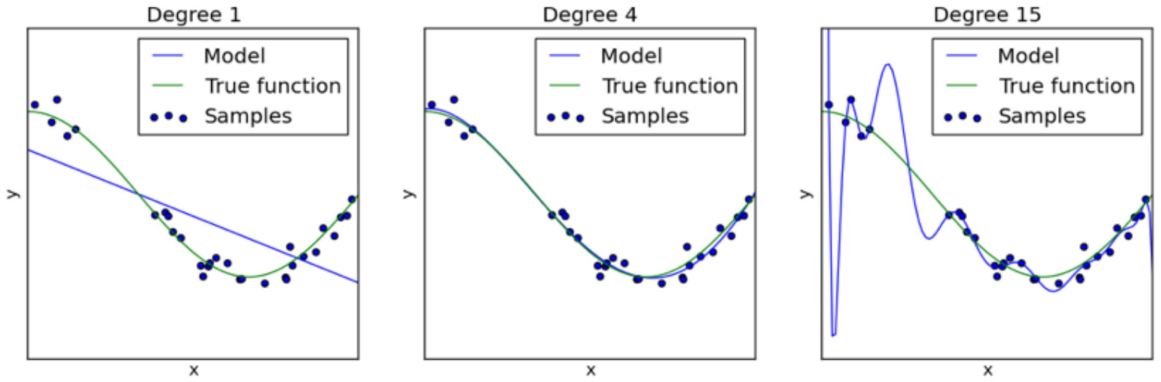
Model generalization

In both our examples **we minimized the loss function on the training data.**
However, this is NOT our ultimate goal!

The most important evaluation metric of a model is **the loss on unseen test examples**, referred to as the **test error**.

Training error is large → underfitting

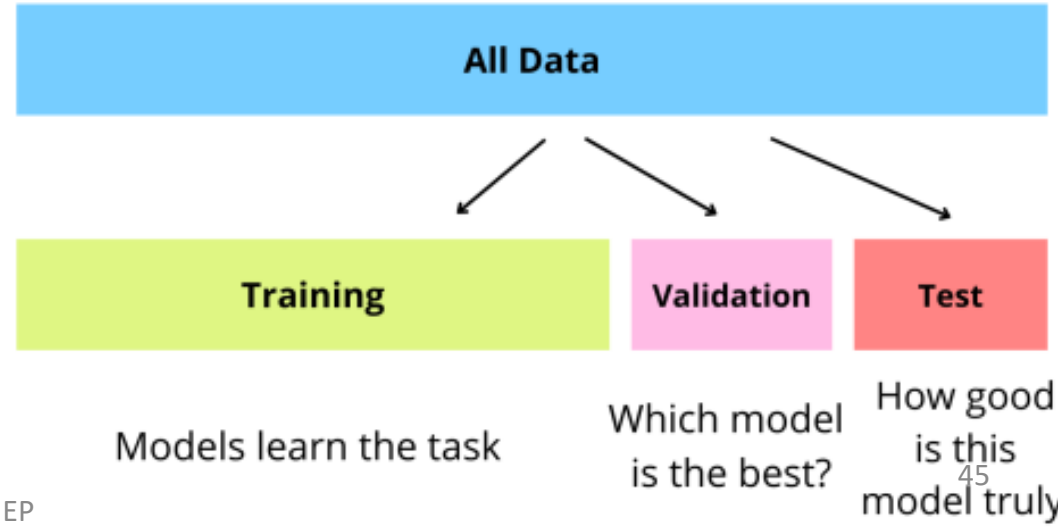
Training error is small but the test error is large → model predicts accurately on the training dataset but doesn't generalize well to other test examples → **overfitting**



Underfitting

Overfitting

http://scikit-learn.org/



Bias – variance tradeoff

Test (or generalisation) error = **systematic error** + **sensitivity of prediction**

bias *variance*

Given the model h, defined over the dataset x, modelling the random output variable y, the generalisation error can be decomposed into:

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

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

- Intrinsic noise in system or measurements
- Cannot be avoided or improved with modeling
- Lower bound on possible noise

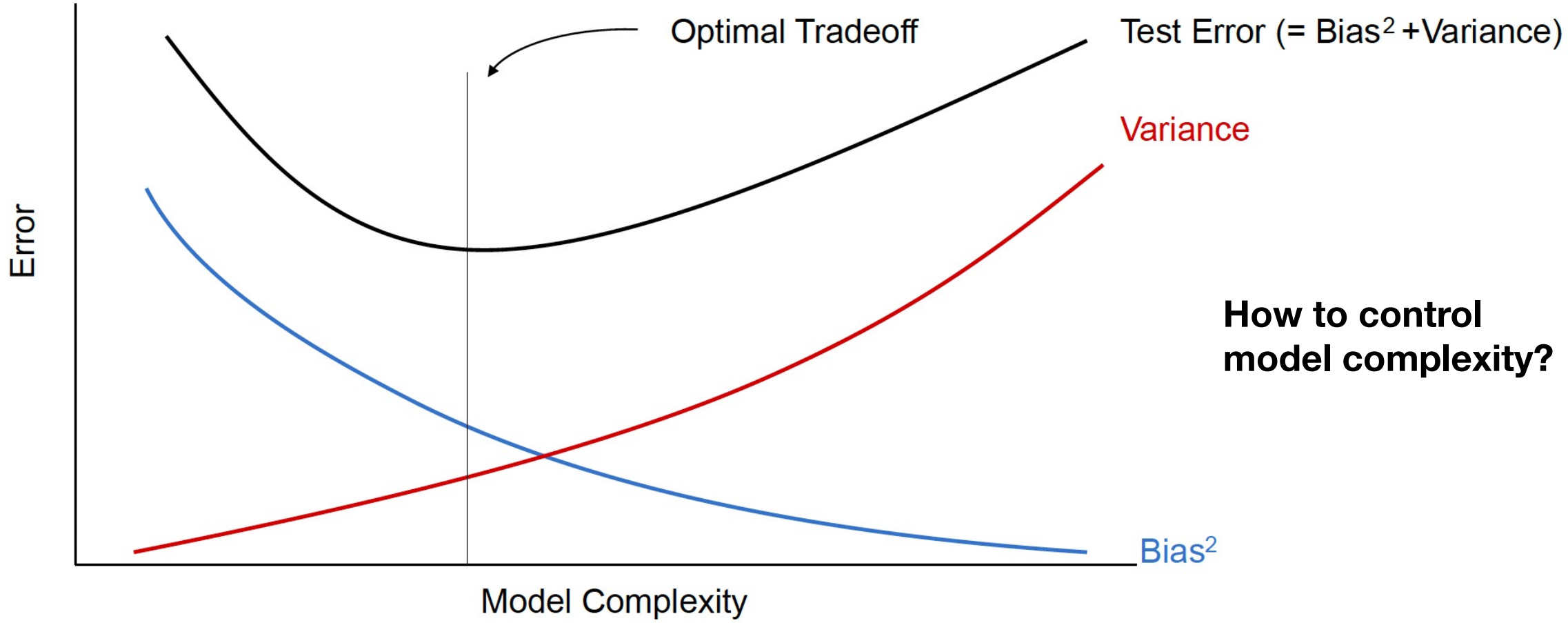
The more complex the model h(x) is, the more data points it will capture, and the lower the bias will be.

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

Bias – variance tradeoff

Test (or generalisation) error = **systematic error** + **sensitivity of prediction**

bias *variance*



How to control model complexity?

Empirical risk minimisation

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

Average expected loss

Model regularization

- Framework to design learning algorithms
  - L(…) is a loss function comparing prediction h(…) with target y
  - $\Omega(\mathbf{w})$ is a regularizer, penalizing certain values of $\mathbf{w}$
    - λ controls how much we penalize, and is a hyperparameter that we have to tune
    - We will come back to this later
- Learning is cast as an optimization problem

Regularisation terms

- Can control the complexity of a model by placing **constraints on the model parameters**
  - Trading some bias to reduce model variance

- **L2 norm**: $\Omega(\mathbf{w}) = ||\mathbf{w}||^2 = \sum_i w_i^2$
  - "Ridge regression", enforcing weights not too large
  - Equivalent to Gaussian prior over weights

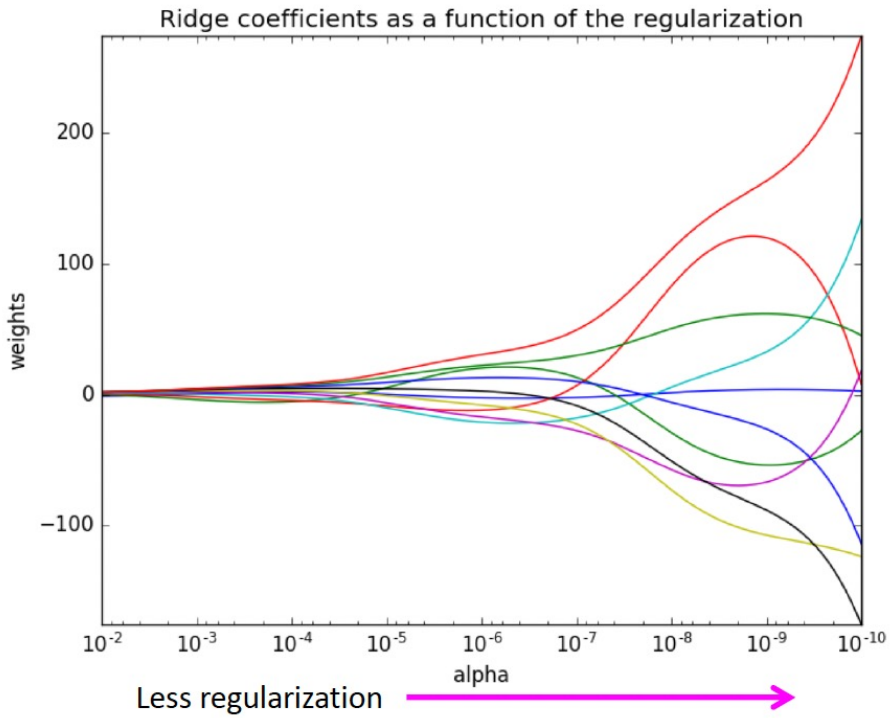
- **L1 norm**: $\Omega(\mathbf{w}) = ||\mathbf{w}|| = \sum_i |w_i|$
  - "Lasso regression", enforcing sparse weights

- Elastic net → L1 + L2 constraints

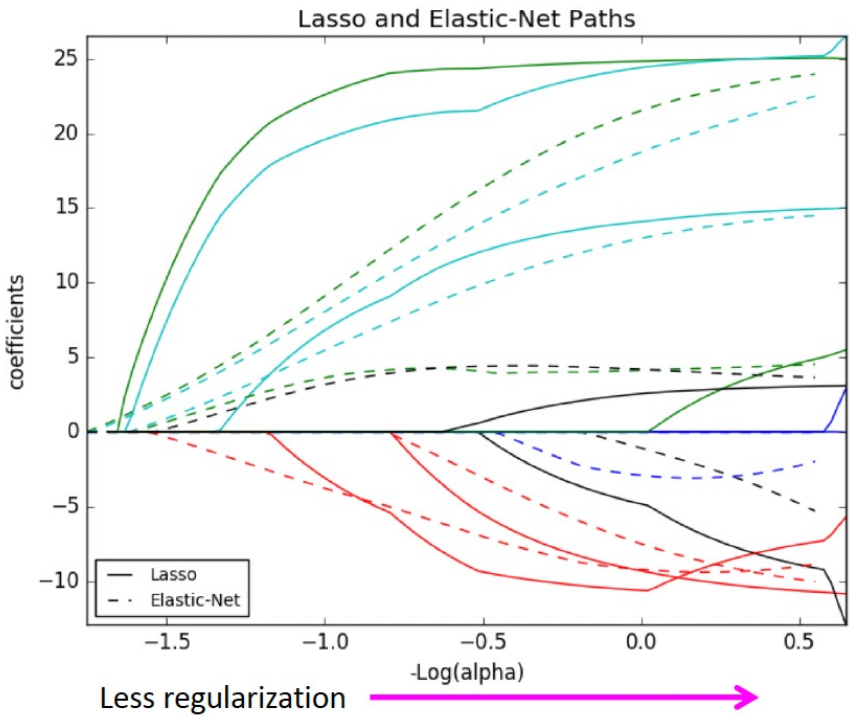
Regularisation term: linear regression example

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

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

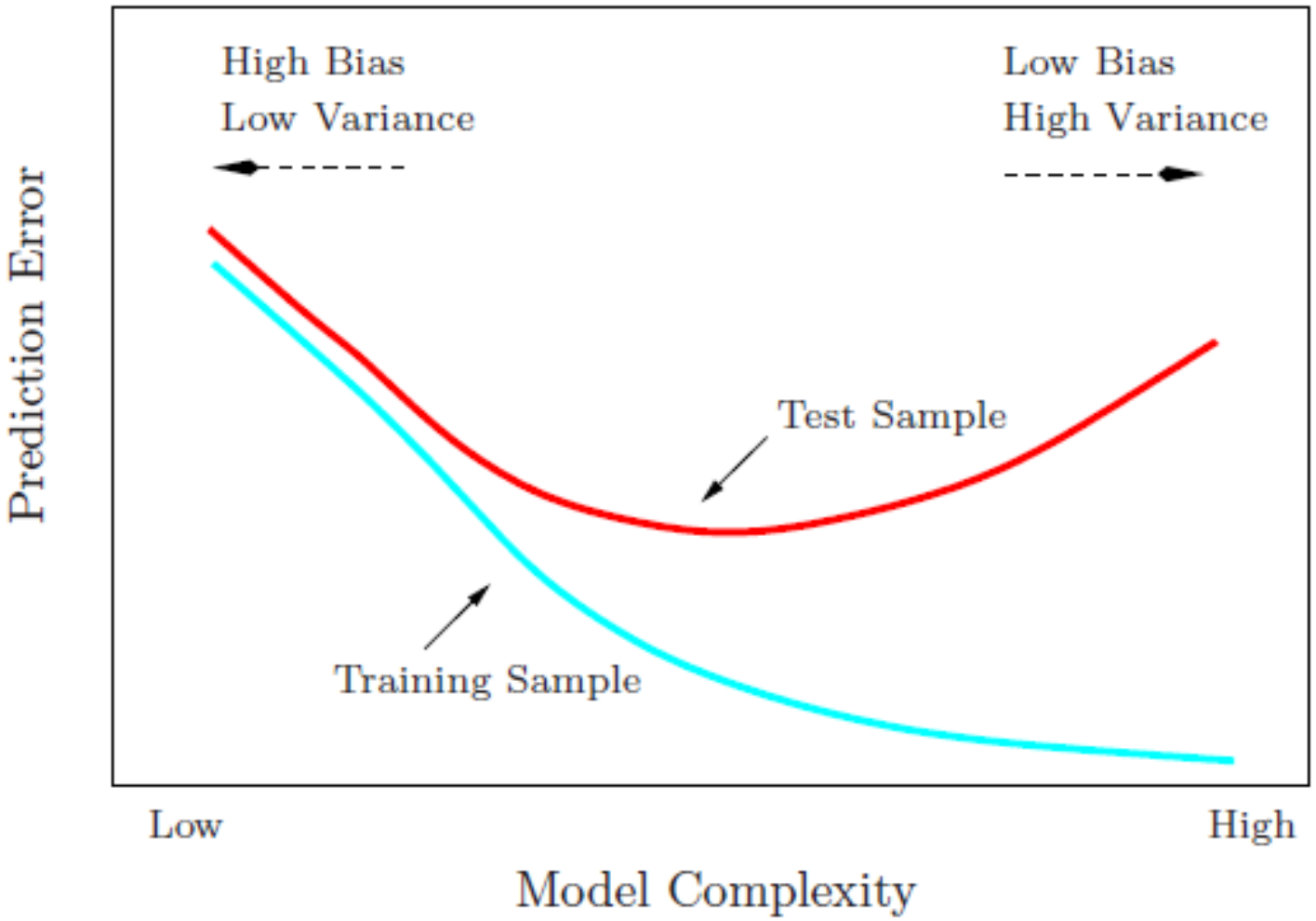


$$L1: \quad \Omega(\mathbf{w}) = ||\mathbf{w}||$$



- L2 keeps weights small, L1 keeps weights sparse!

Test sample to measure generalization error



Adding non-linearity

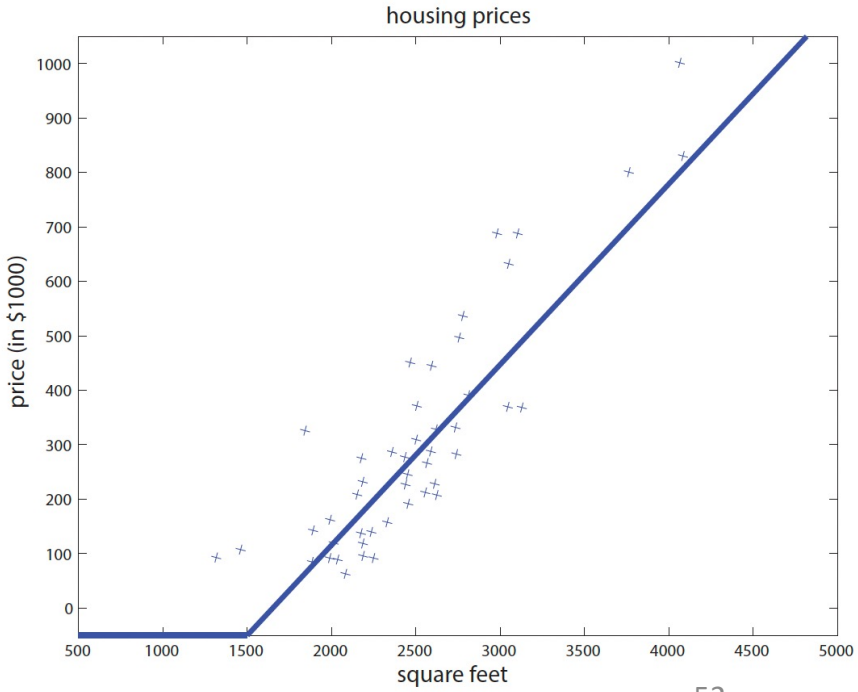
- Let's go back to the housing price prediction example. We introduce some non-linearity, for example forcing the prediction to be non negative

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

- This can be done by introducing the parametrised function:

$$\bar{h}_\theta(x) = \max(wx + b, 0), \text{ where } \theta = (w, b) \in \mathbb{R}^2$$

Rectified Linear Unit (*ReLU*)



- Generally, a non-linear function mapping $\mathbb{R} \to \mathbb{R}$ is called ***«activation function»***

Adding non-linearity: stacking neurons

- Let's **add extra features** to our familiar pricing problem: number of bedrooms, zip code, and wealth of the neighborhood

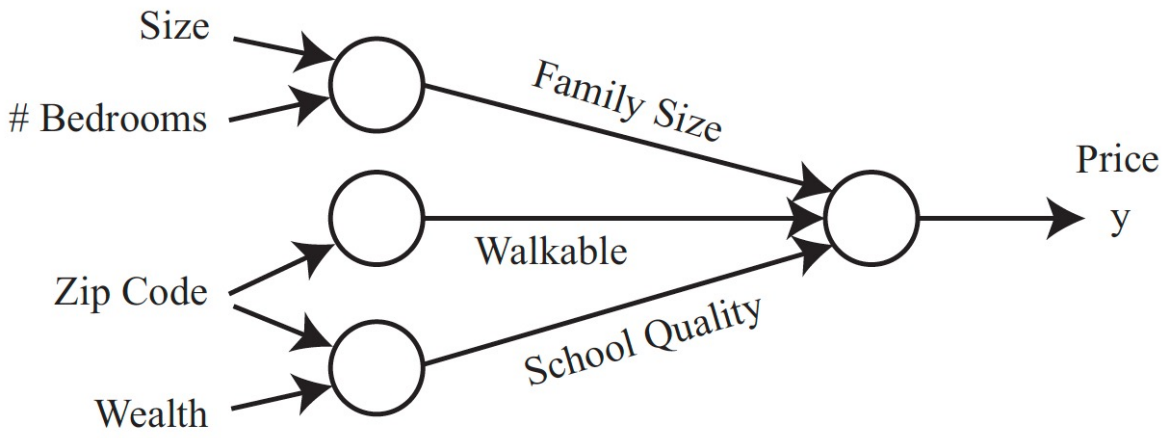
- Given these features (size, number of bedrooms, zip code, and wealth), we might then decide that the price of the house depends on the maximum family size it can accommodate.

- We may define some new features (family size, walkable, school quality) from which the final output ultimately depends.

$$a_1 = \mathrm{ReLU}(\theta_1 x_1 + \theta_2 x_2 + \theta_3)$$
$$a_2 = \mathrm{ReLU}(\theta_4 x_3 + \theta_5)$$
$$a_3 = \mathrm{ReLU}(\theta_6 x_3 + \theta_7 x_4 + \theta_8)$$

$$\longrightarrow \bar{h}_\theta(x) = \theta_9 a_1 + \theta_{10} a_2 + \theta_{11} a_3 + \theta_{12}$$

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

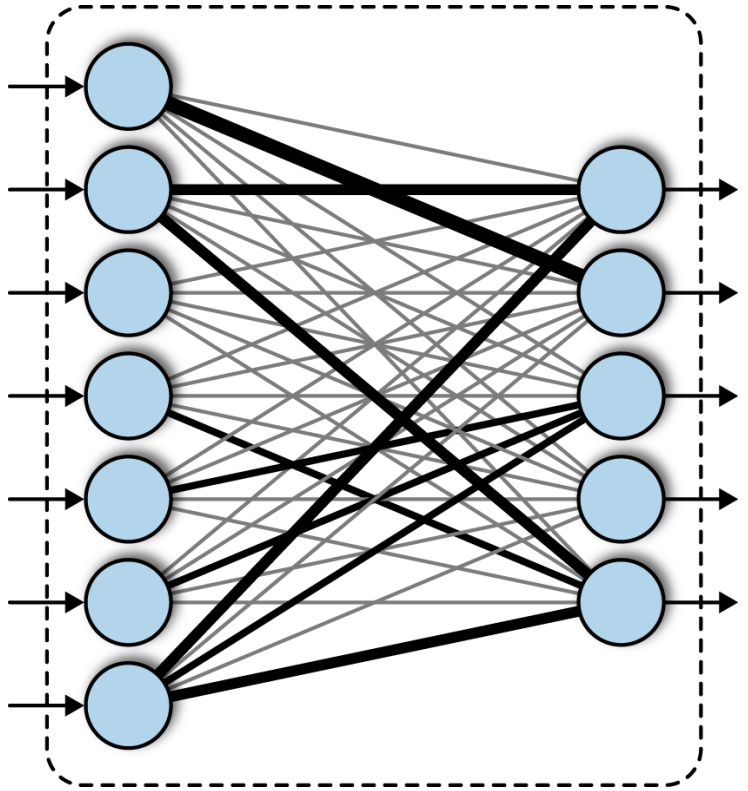


- Result: quite complex non-linear function of x
- Again, define loss function and minimise it

Two-layer Fully-Connected Neural Networks

- We tipically have no clue on the meaning of extra features obtained combining x_1, x_2 etc
- Let's then generalise the previous example:

$$a_1 = \mathrm{ReLU}(w_1^\top x + b_1), \text{ where } w_1 \in \mathbb{R}^4 \text{ and } b_1 \in \mathbb{R}$$
$$a_2 = \mathrm{ReLU}(w_2^\top x + b_2), \text{ where } w_2 \in \mathbb{R}^4 \text{ and } b_2 \in \mathbb{R}$$
$$a_3 = \mathrm{ReLU}(w_3^\top x + b_3), \text{ where } w_3 \in \mathbb{R}^4 \text{ and } b_3 \in \mathbb{R}$$



- All the intermediate variables (a) depend on all the features x
- In general, for m intermediate variables:

$$\forall j \in [1, ..., m], \quad z_j = {w_j^{[1]}}^\top x + b_j^{[1]} \text{ where } w_j^{[1]} \in \mathbb{R}^d, b_j^{[1]} \in \mathbb{R}$$
$$a_j = \mathrm{ReLU}(z_j),$$
$$a = [a_1, \ldots, a_m]^\top \in \mathbb{R}^m$$
$$\bar{h}_\theta(x) = {w^{[2]}}^\top a + b^{[2]} \text{ where } w^{[2]} \in \mathbb{R}^m, b^{[2]} \in \mathbb{R},$$

Two-layer Fully-Connected Neural Networks

- Vector notation

$$\forall j \in [1, ..., m], \quad z_j = {w_j^{[1]}}^\top x + b_j^{[1]} \text{ where } w_j^{[1]} \in \mathbb{R}^d, b_j^{[1]} \in \mathbb{R}$$

$$\underbrace{\begin{bmatrix} z_1 \\ \vdots \\ \vdots \\ z_m \end{bmatrix}}_{z \in \mathbb{R}^{m \times 1}} = \underbrace{\begin{bmatrix} — \; {w_1^{[1]}}^\top \; — \\ — \; {w_2^{[1]}}^\top \; — \\ \vdots \\ — \; {w_m^{[1]}}^\top \; — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{m \times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{x \in \mathbb{R}^{d \times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{m \times 1}}$$

$$a = \text{ReLU}(W^{[1]} x + b^{[1]})$$
$$\bar{h}_\theta(x) = W^{[2]} a + b^{[2]}$$

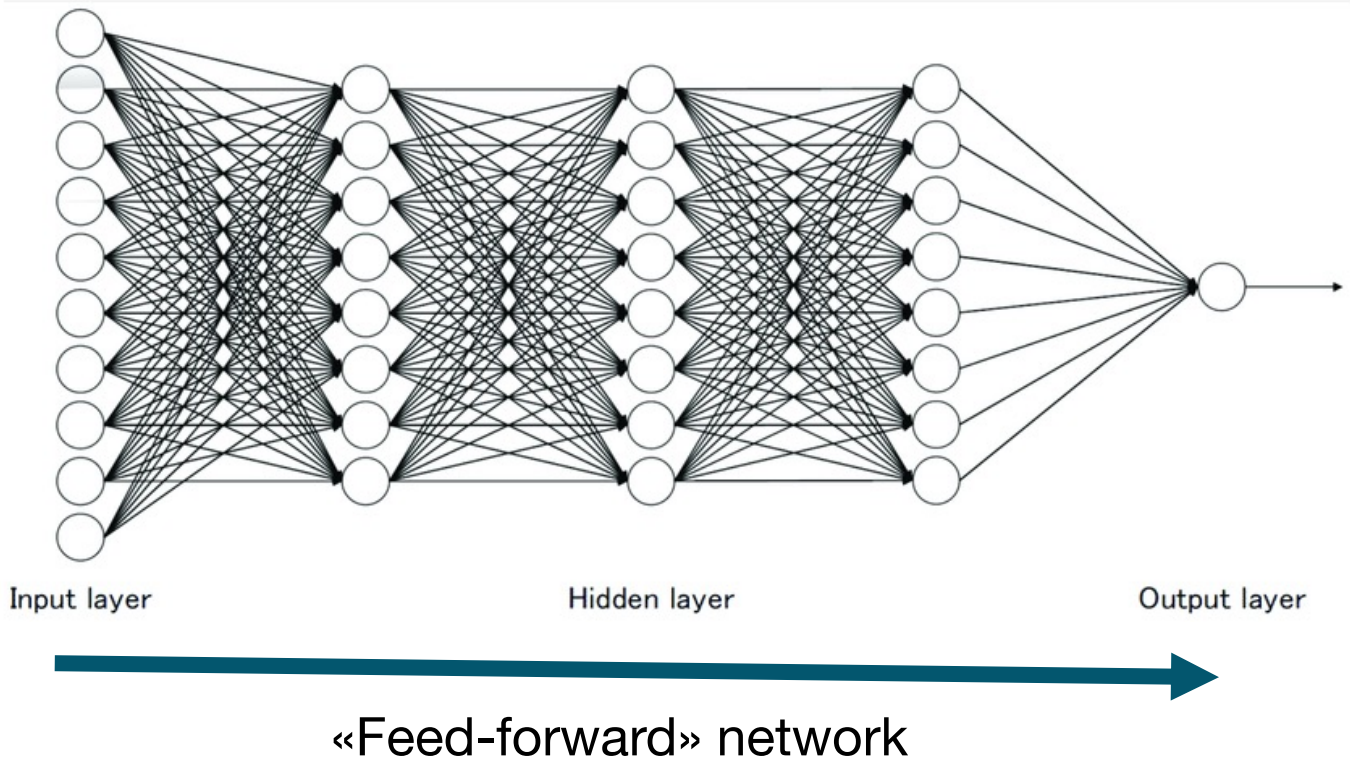
ReLU here is a vector

Multi-layer Fully-Connected Neural Networks

- r → number of layers
- Total number of neurons: $m_1 + m_2 + \ldots + m_r$

$$a^{[1]} = \mathrm{ReLU}(W^{[1]}x + b^{[1]})$$
$$a^{[2]} = \mathrm{ReLU}(W^{[2]}a^{[1]} + b^{[2]})$$
$$\ldots$$
$$a^{[r-1]} = \mathrm{ReLU}(W^{[r-1]}a^{[r-2]} + b^{[r-1]})$$
$$\bar{h}_\theta(x) = W^{[r]}a^{[r-1]} + b^{[r]}$$

Output layer: linear



More activation functions

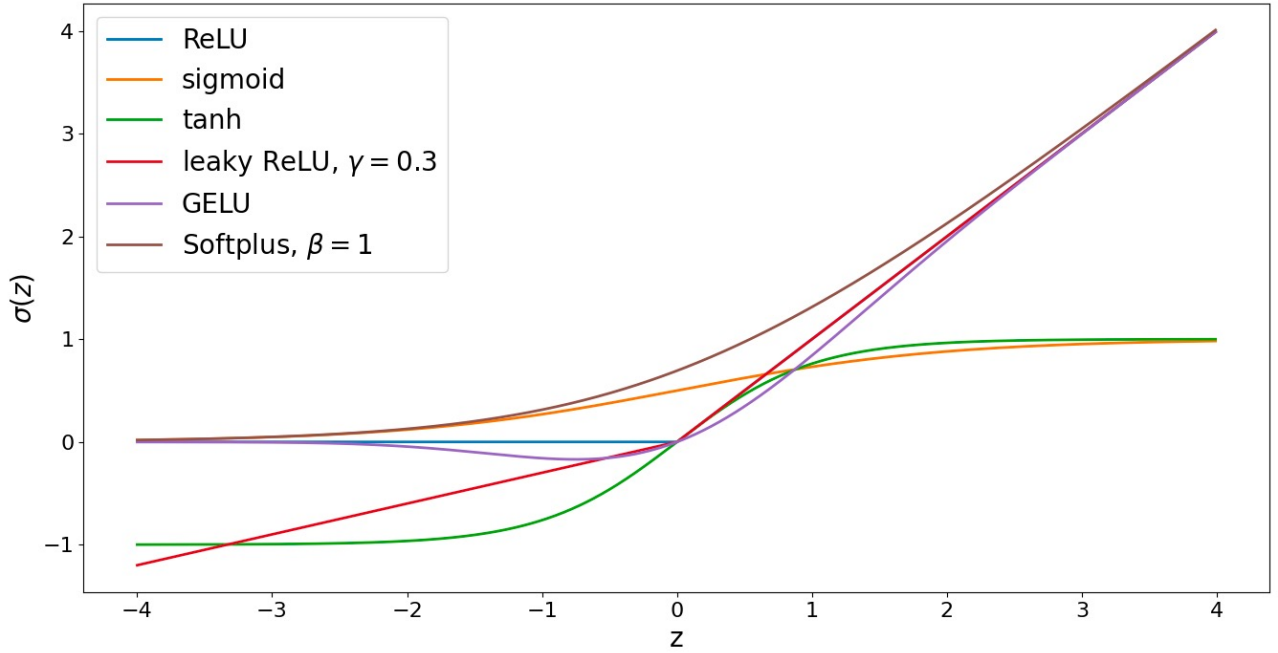
$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \text{(sigmoid)}$$

$$\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \qquad \text{(tanh)}$$

$$\sigma(z) = \max\{z, \gamma z\}, \gamma \in (0,1) \qquad \text{(leaky ReLU)}$$

$$\sigma(z) = \frac{z}{2}\left[1+\text{erf}(\frac{z}{\sqrt{2}})\right] \qquad \text{(GELU)}$$

$$\sigma(z) = \frac{1}{\beta}\log(1+\exp(\beta z)), \beta > 0 \qquad \text{(Softplus)}$$



Universal approximation

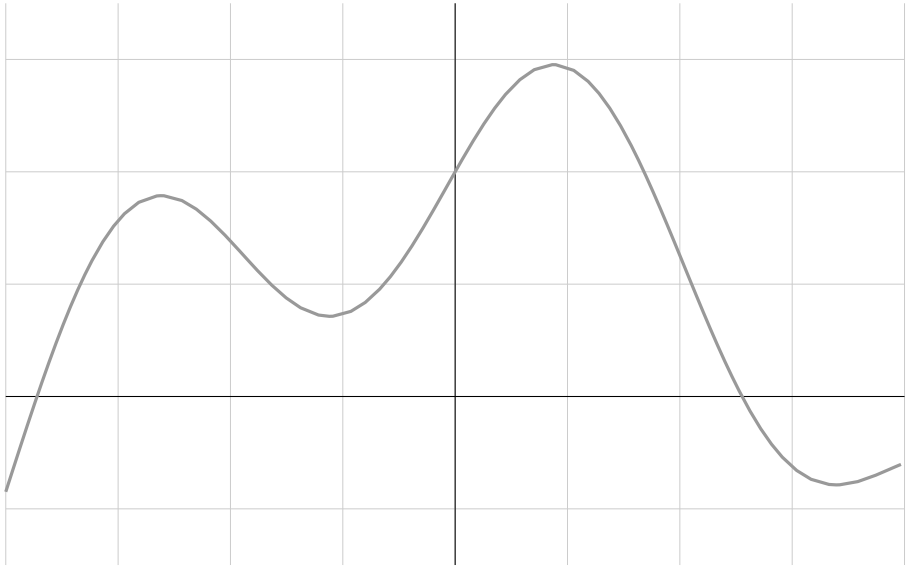
- Feed-forward neural network with a single hidden layer containing a finite number of neurons can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

  - Only mild assumptions on non-linear activation function needed. Sigmoid functions work, as do others

- But no information on how many neurons needed, or how much data!

Universal approximation

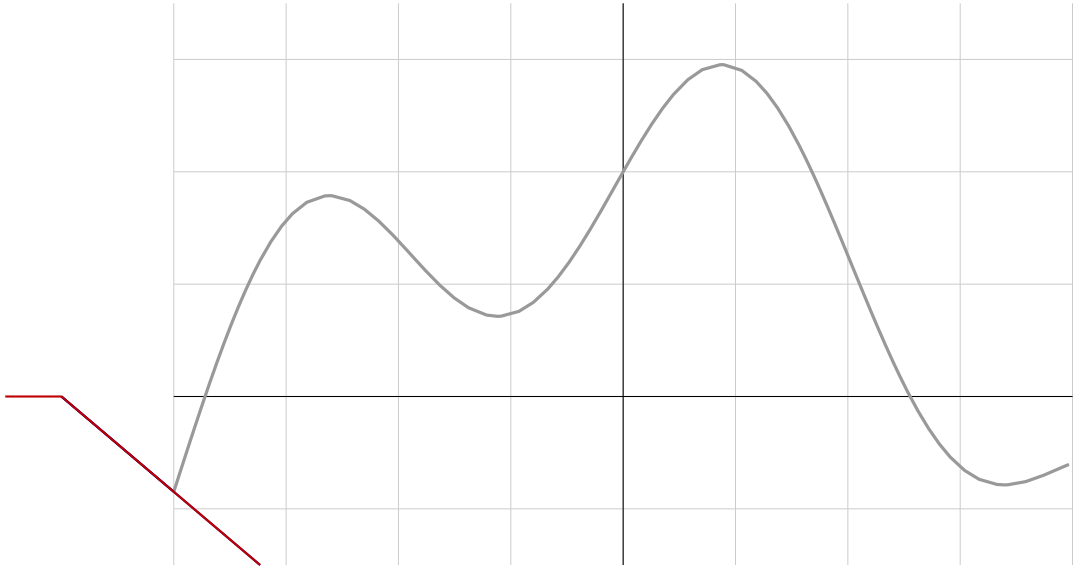
We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

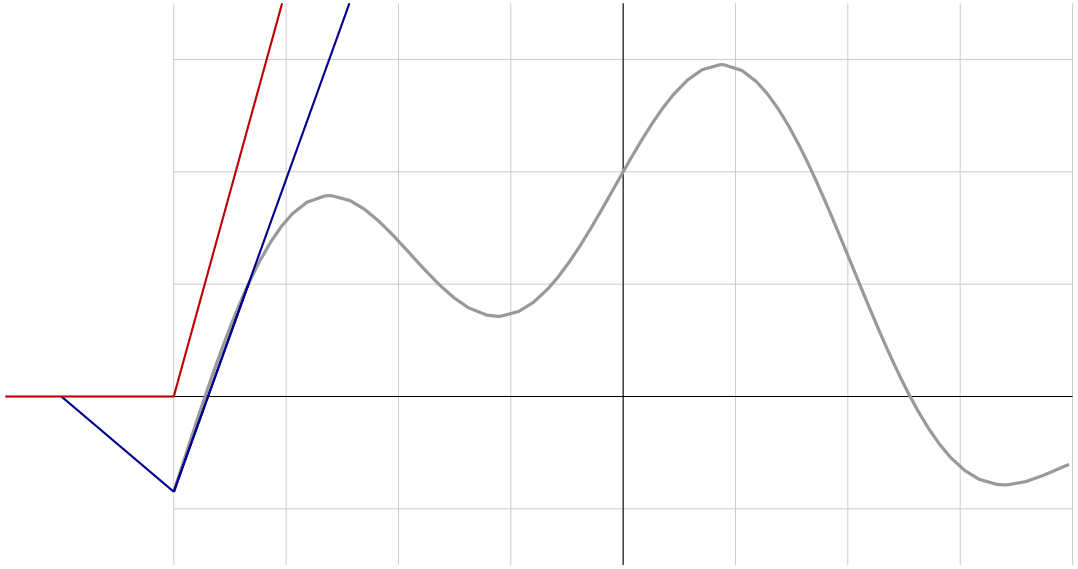
$$f(x) = \sigma(w_1 x + b_1)$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

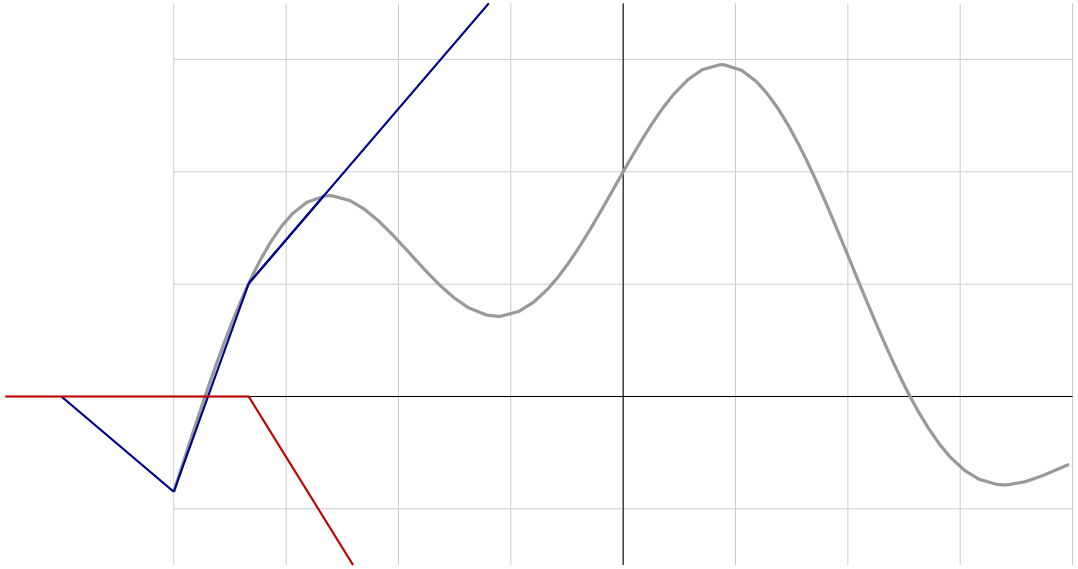
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2)$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

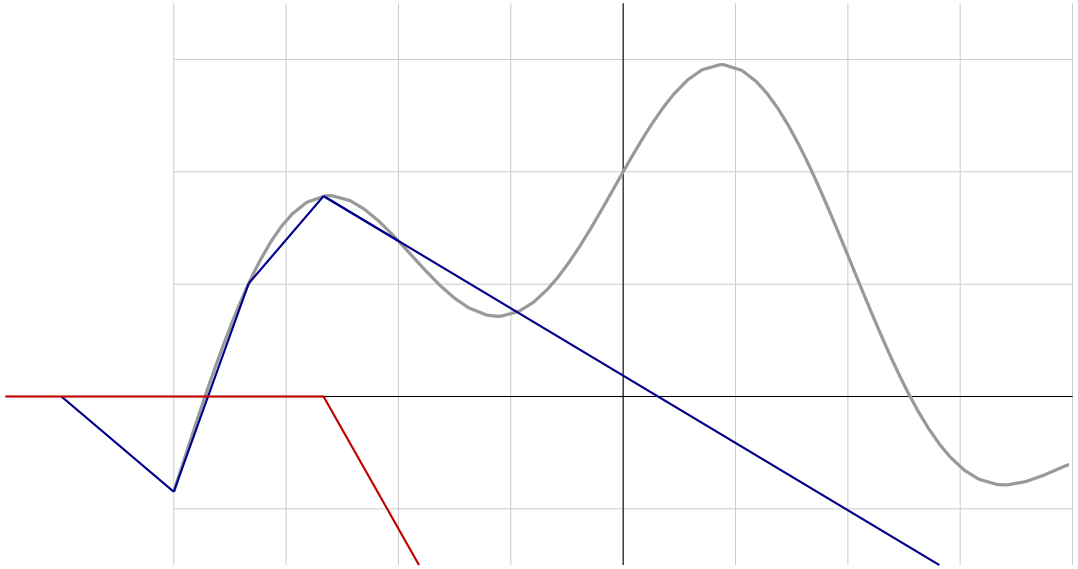
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3)$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

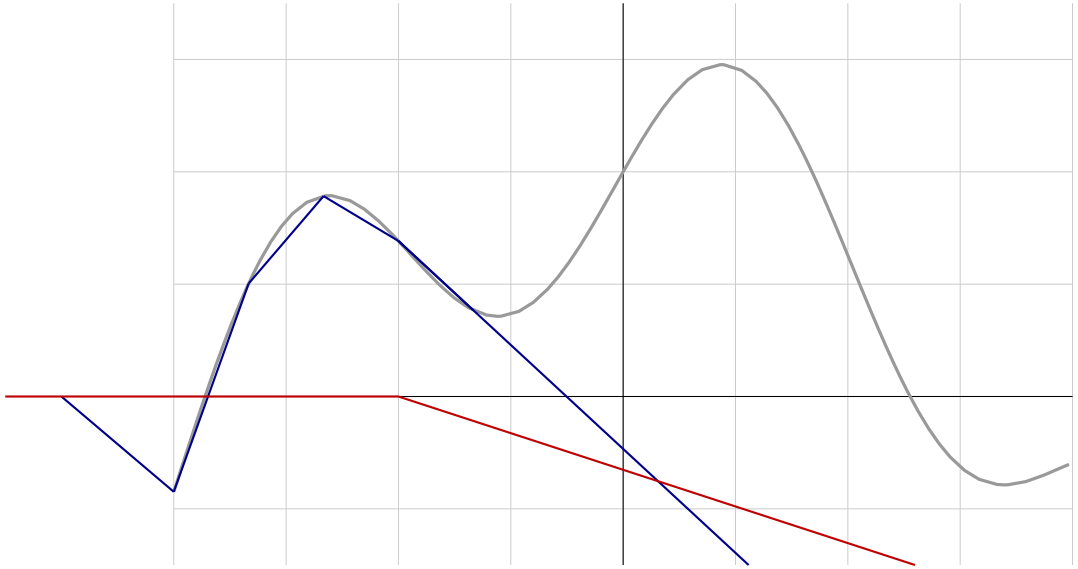
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

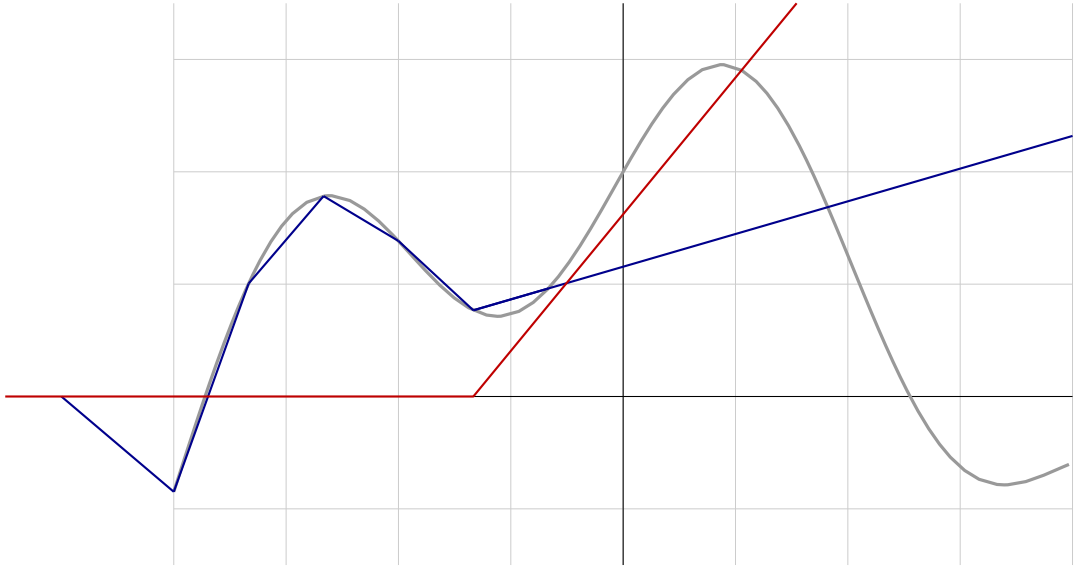
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

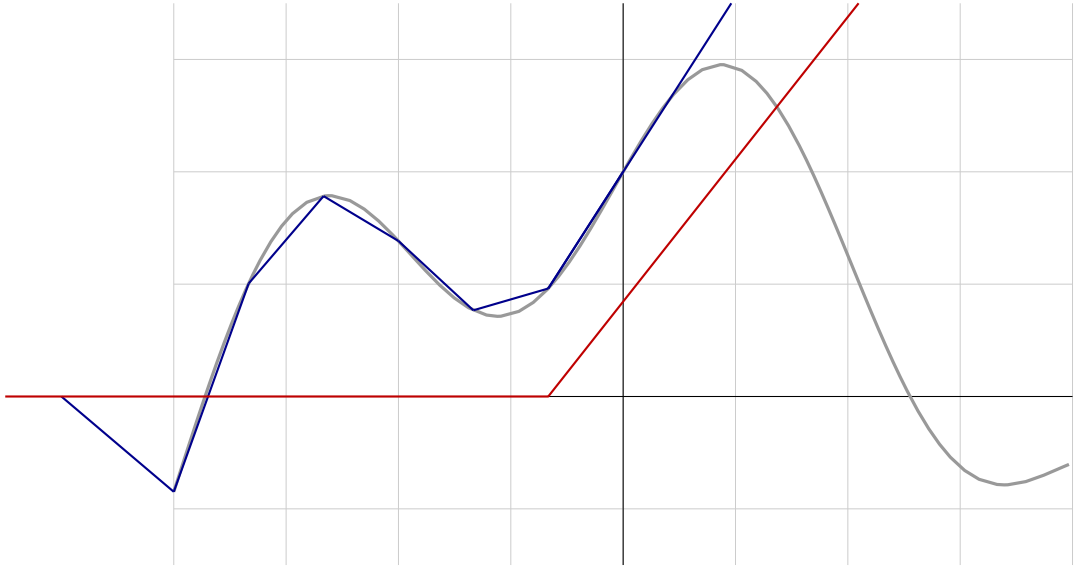
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

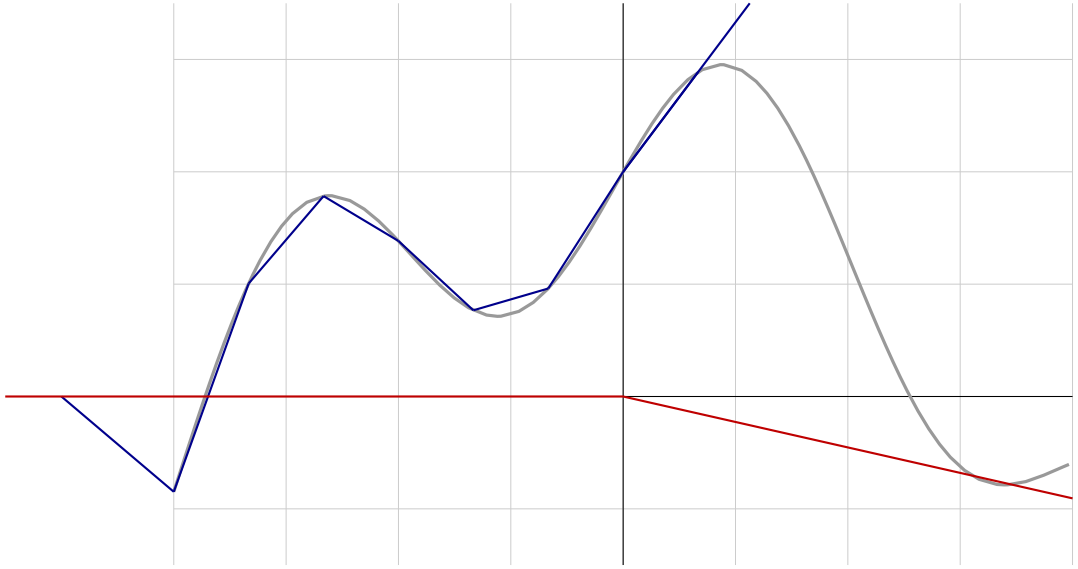
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

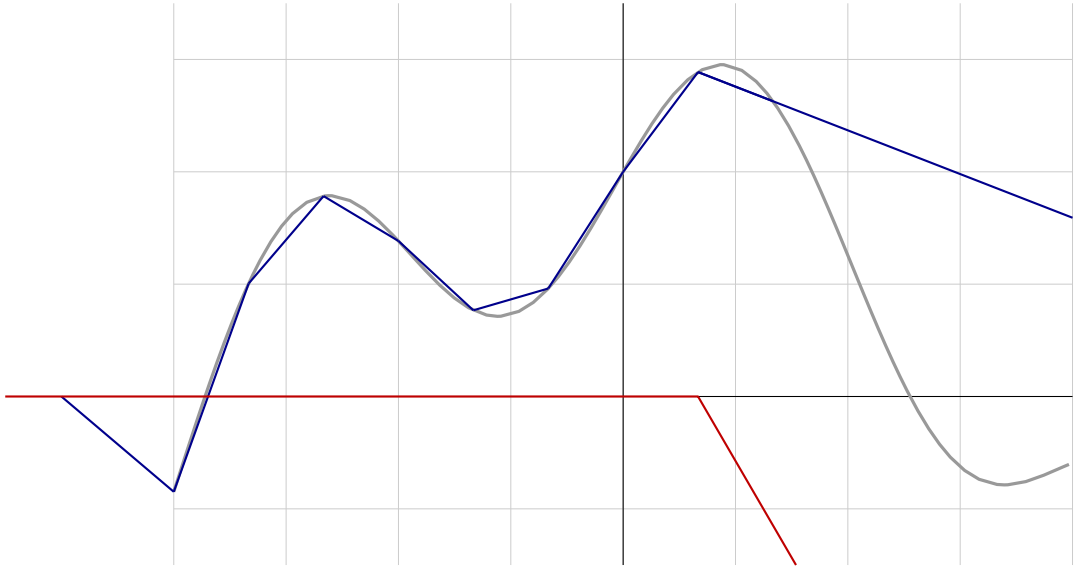
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

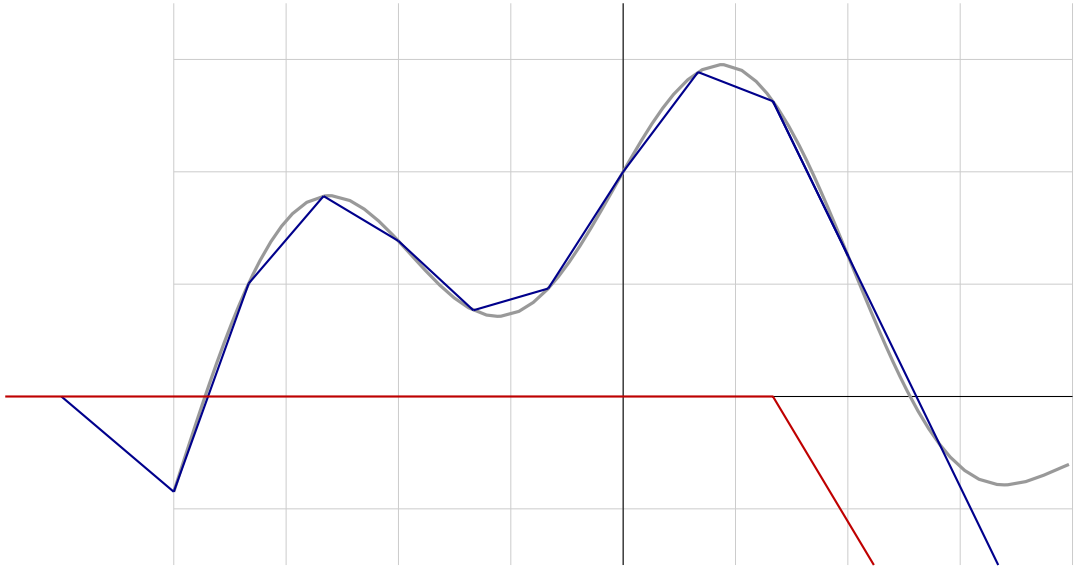
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

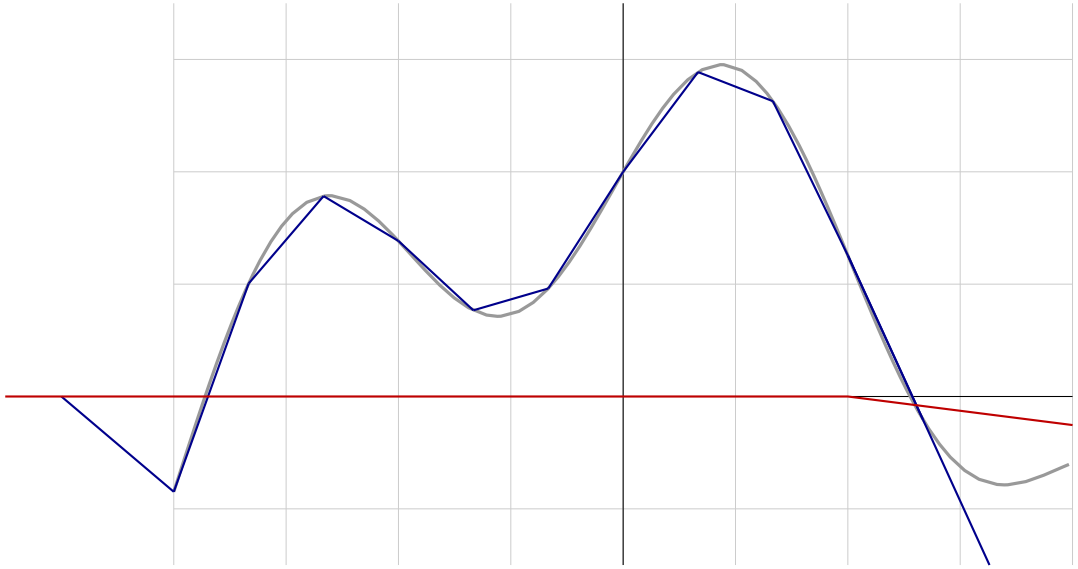
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

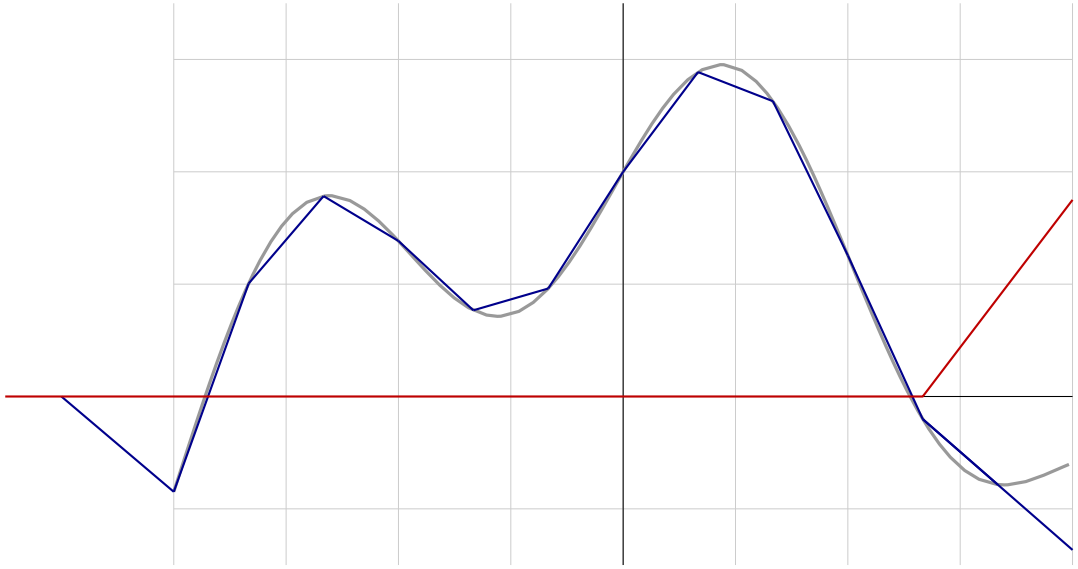
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

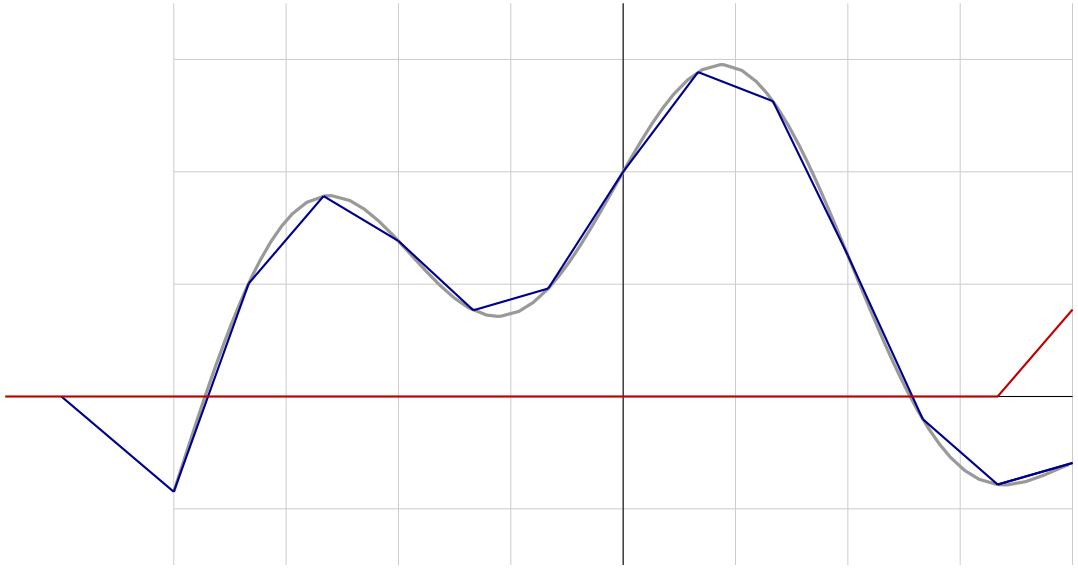
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

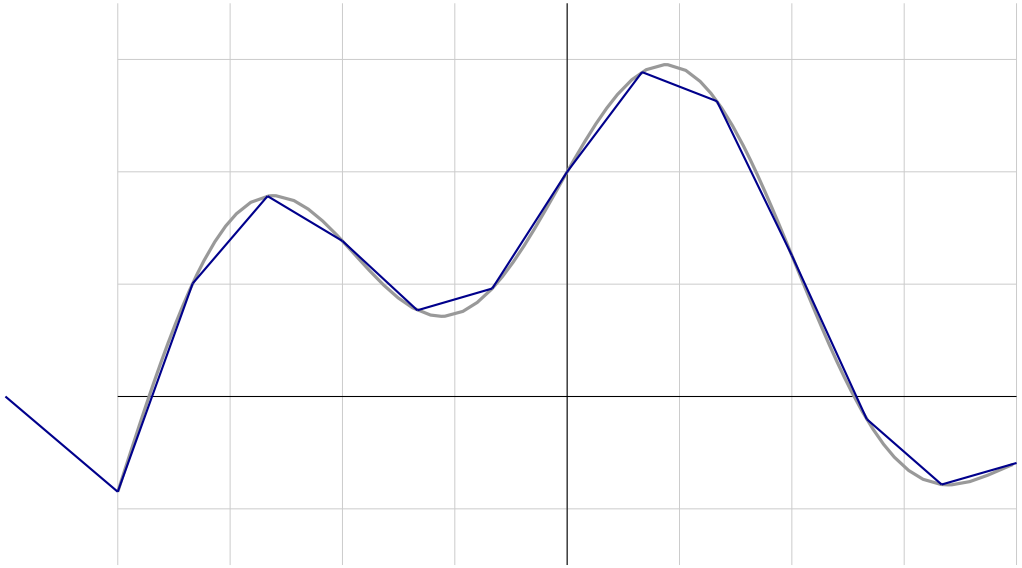
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

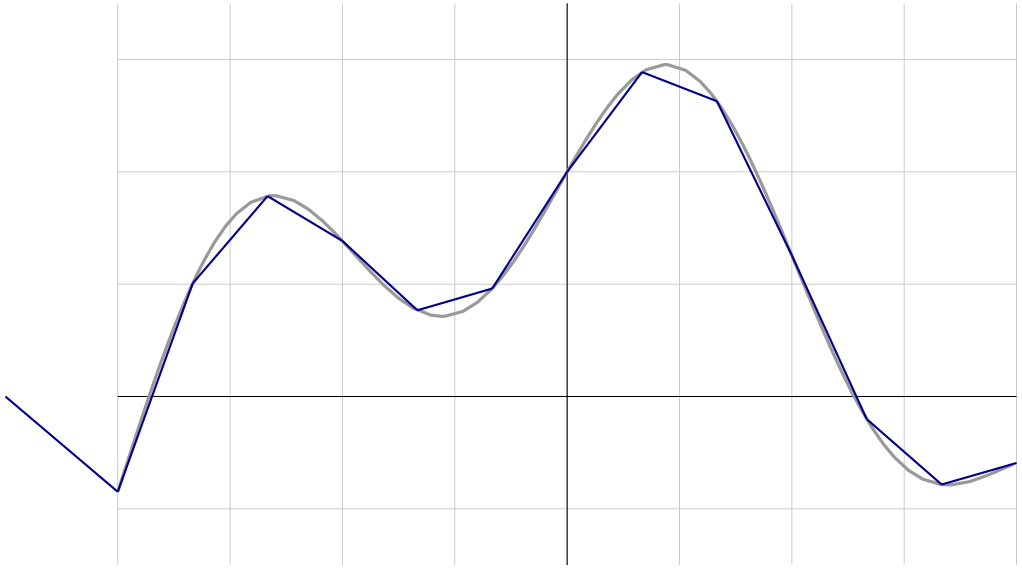
$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Universal approximation

We can approximate any $\psi \in \mathcal{C}([a, b], \mathrm{R})$ with a linear combination of translated/scaled ReLU functions.

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



This is true for other activation functions under mild assumptions.

Universal approximation

- Feed-forward neural network with a single hidden layer containing a finite number of neurons can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

  – Only mild assumptions on non-linear activation function needed. Sigmoid functions work, as do others

- But no information on how many neurons needed, or how much data!

- How to find the parameters, given a dataset, to perform this approximation?

Neural Network optimisation problem

The choice of a loss function in neural networks depends on the task.

- **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

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

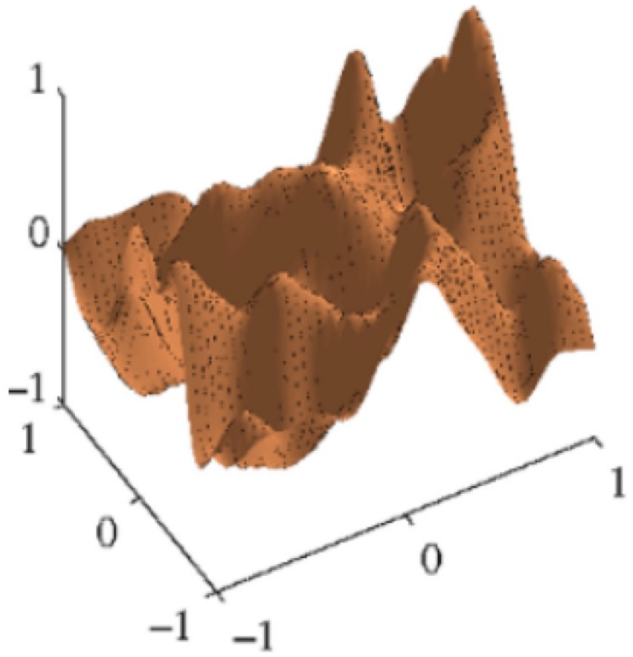
How to minimize the loss function?

Can we apply gradient descent?

– Compute gradient w.r.t. parameters: $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$

– Update parameters: $\mathbf{w}' \leftarrow \mathbf{w} - \eta \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$

- Now we need gradients w.r.t. two large set of parameters
- Gradients will depend on loss and network architecture
- Loss function is non-convex
  - – Gradient descent may get stuck in non-optimal stationary point



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

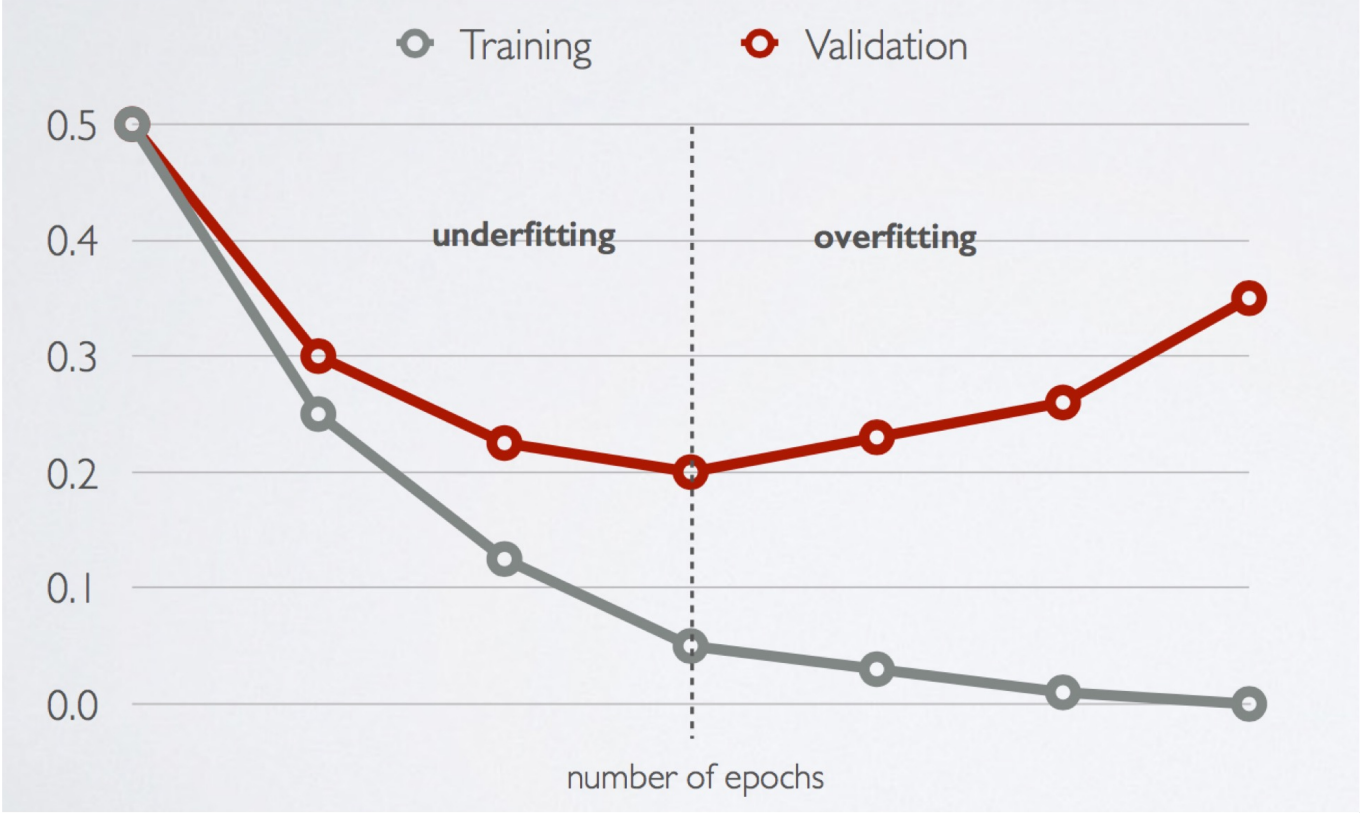
$$\phi^N\left(\ldots\phi^1(x)\right)$$

- Backward step (b-prop) $\dfrac{\partial L}{\partial \phi^a} = \sum_j \dfrac{\partial \phi_j^{(a+1)}}{\partial \phi_j^a} \dfrac{\partial L}{\partial \phi_j^{(a+1)}}$

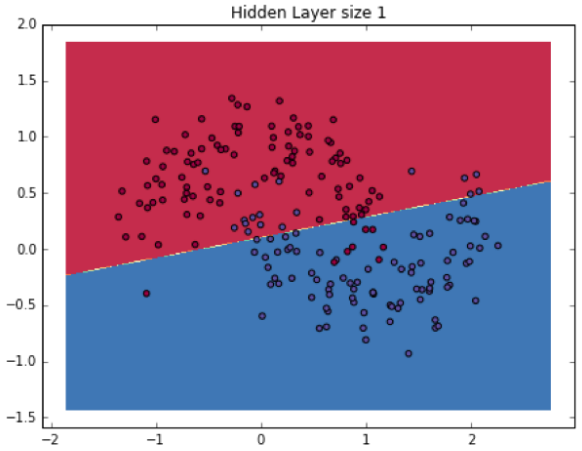
- Compute parameter gradients $\dfrac{\partial L}{\partial \mathbf{w}^a} = \sum_j \dfrac{\partial \phi_j^a}{\partial \mathbf{w}^a} \dfrac{\partial L}{\partial \phi_j^a}$

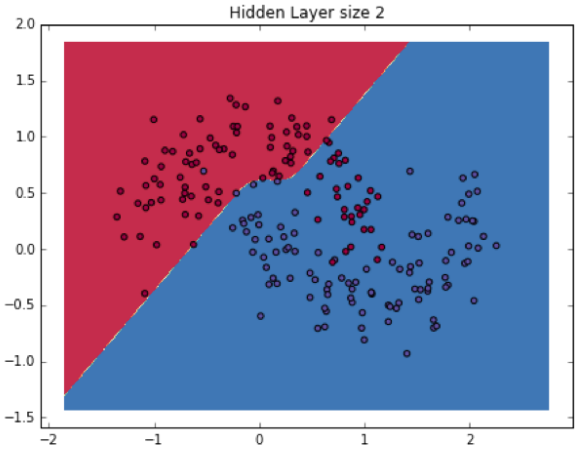
Backpropagation

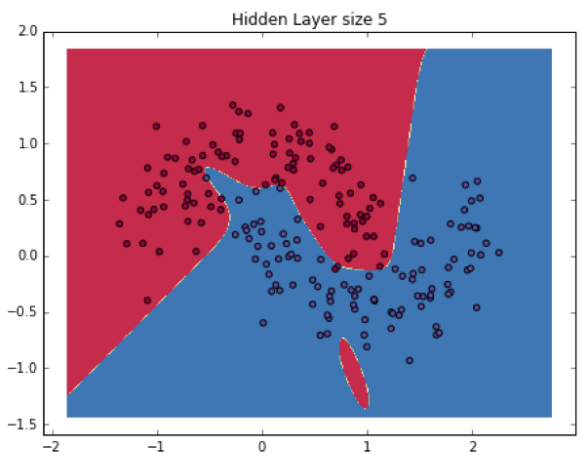
- Repeat gradient update of weights to reduce loss
  - Gradient descent + backpropagation
  - In practice, handled by optimise algorithm (Adam) → see exercise!
  - Each iteration through dataset is called an **epoch**
- Use validation set to examine for overtraining, and determine when to stop training

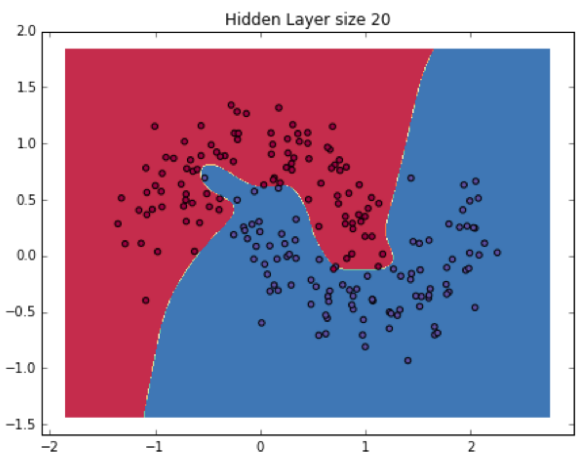


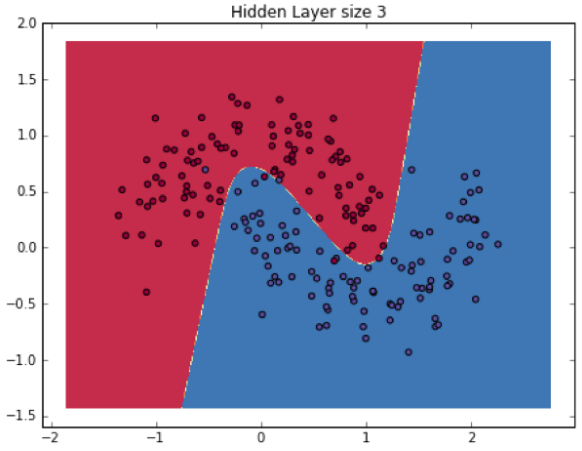
Neural Network decision boundaries

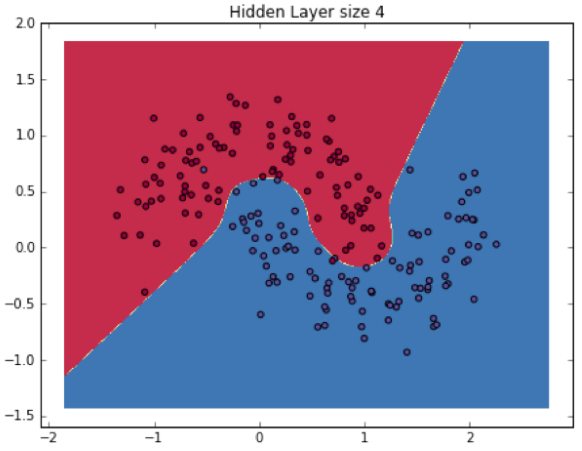


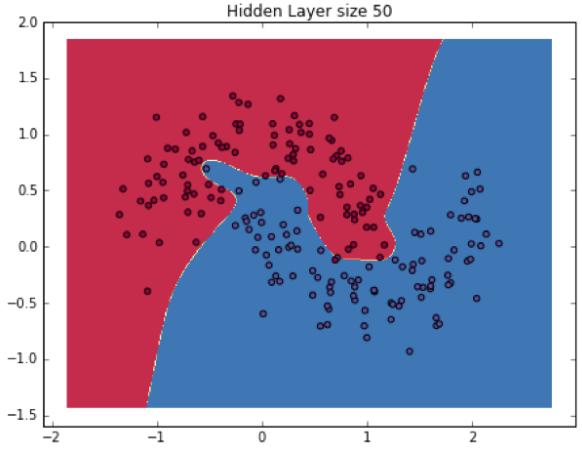








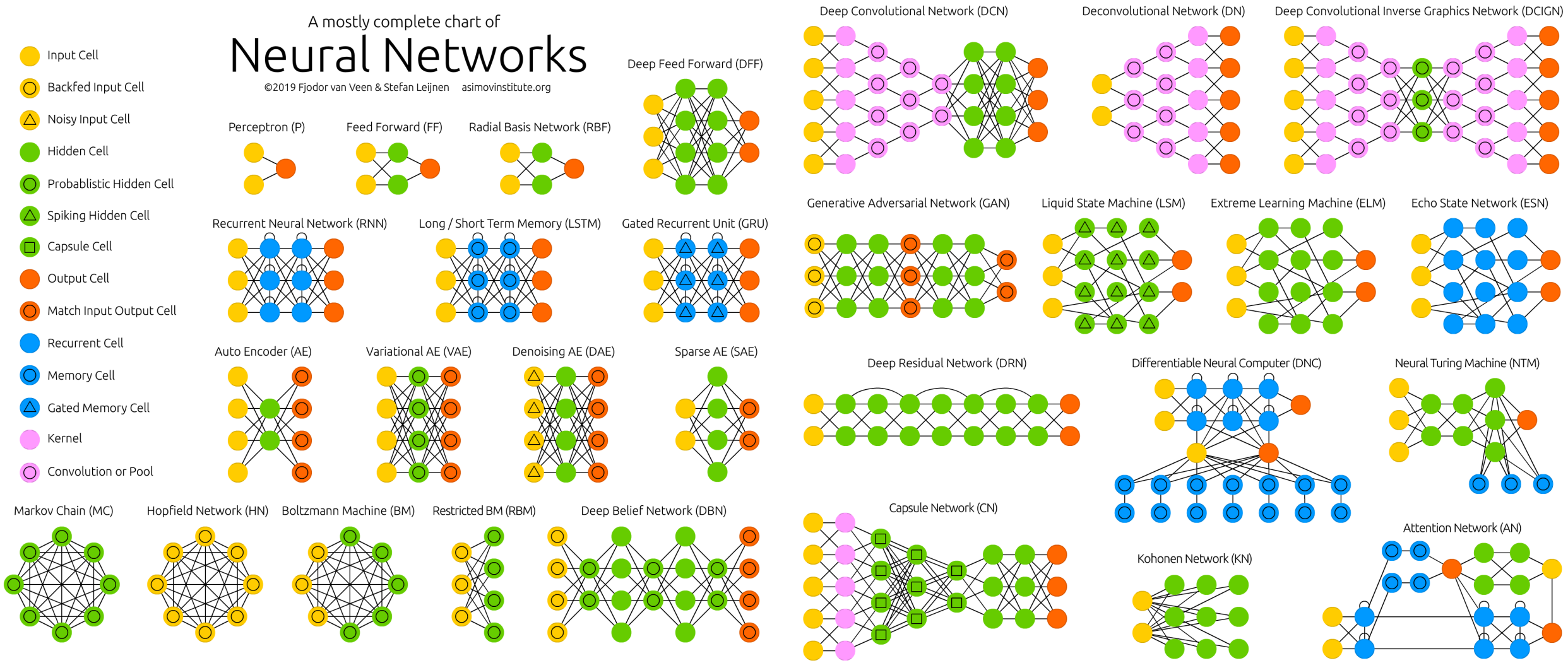




2-class classification
1-hidden layer NN
L2 norm regularization

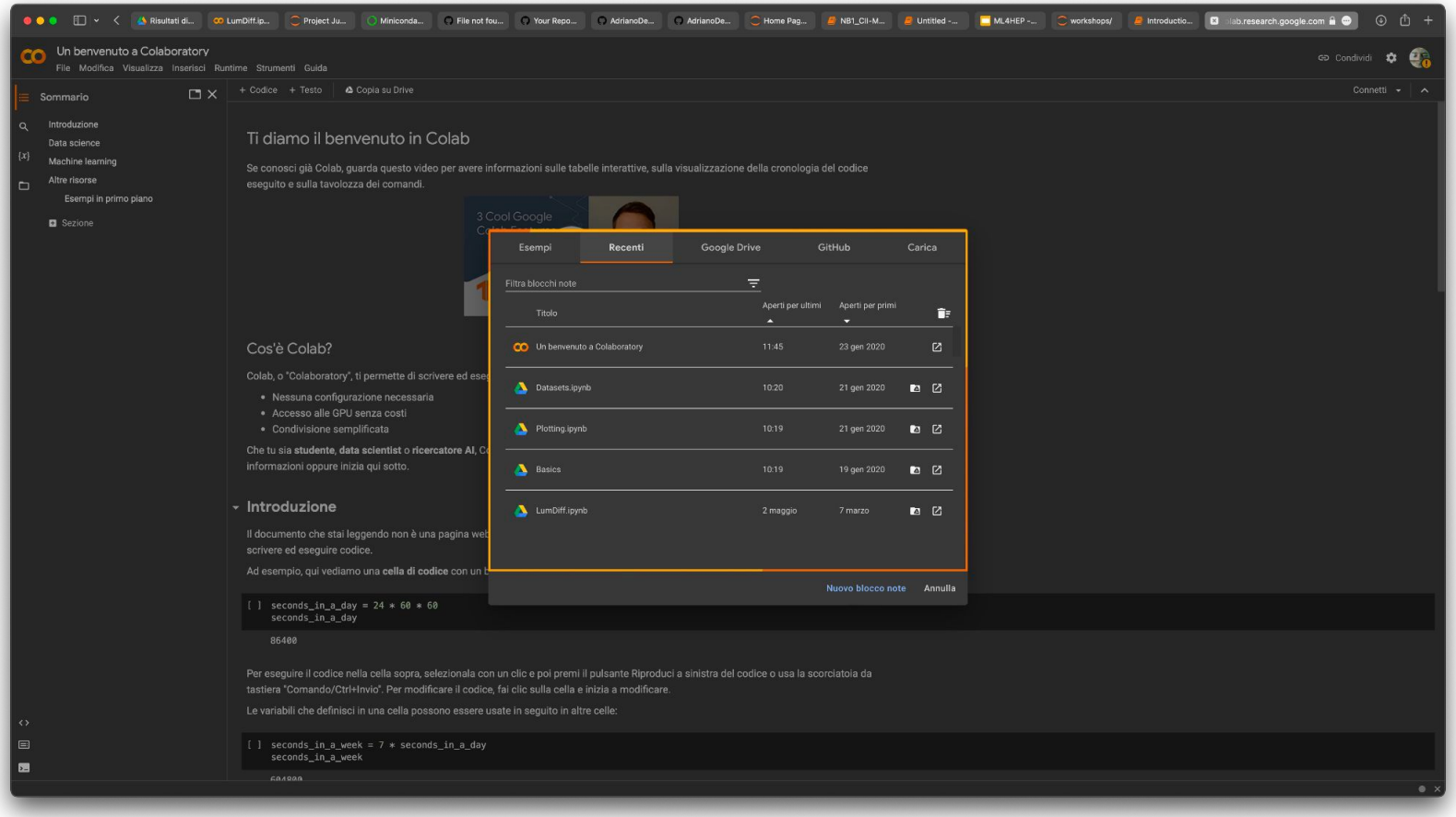
http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/

The NN zoo


A mostly complete chart of
Neural Networks
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org
Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Differentiable Neural Computer (DNC)
Neural Turing Machine (NTM)
Capsule Network (CN)
Kohonen Network (KN)
Attention Network (AN)

Hands-on!

GitHub repo: https://github.com/fsimone91/course_ml4hep/tree/main
Go to: http://colab.research.google.com/
Import notebook form github



Pre-exercises

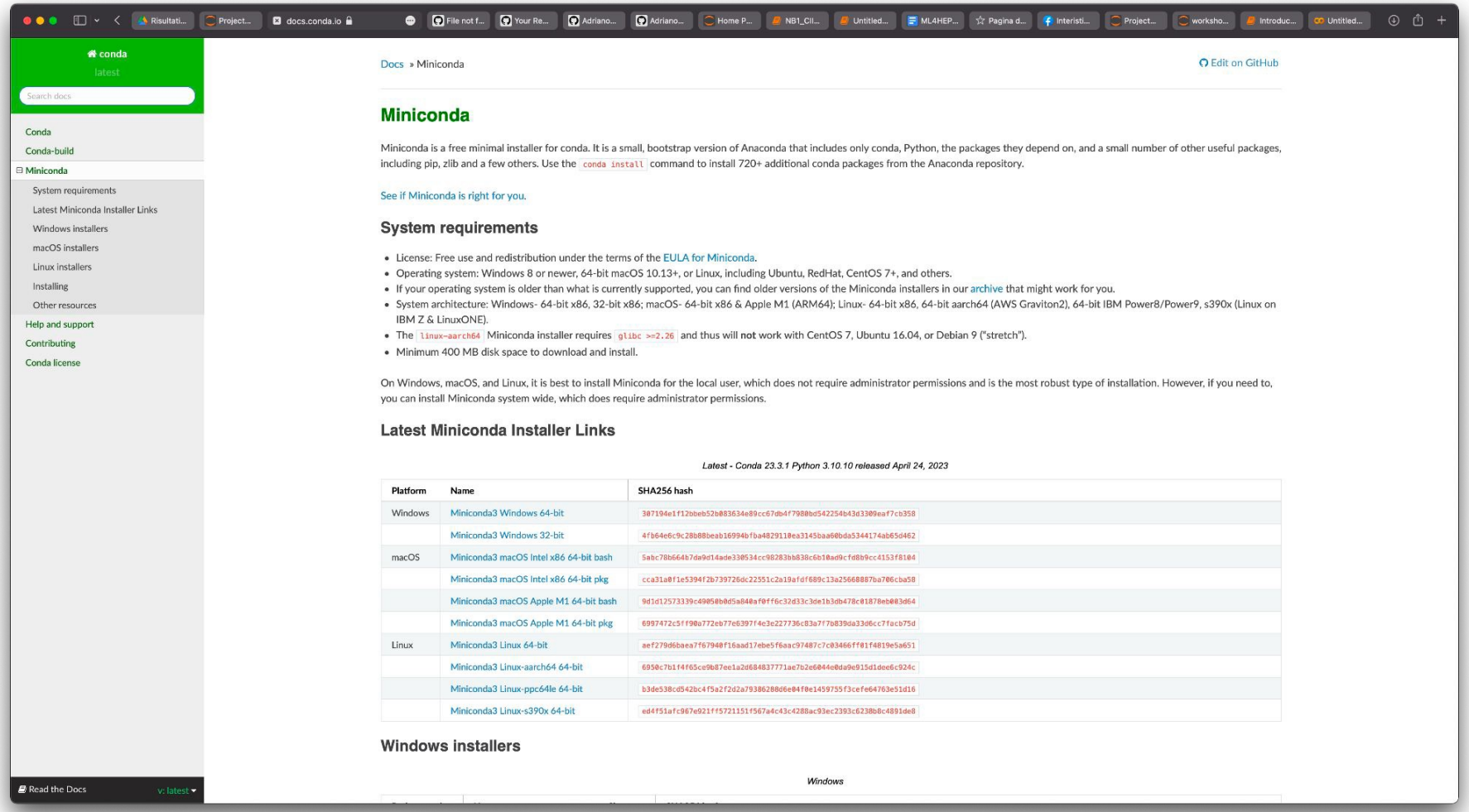
If you never played with python, you need to go through the following notebooks:
https://github.com/fsimone91/course_ml4hep/tree/main/notebooks/intro_notebooks

Exercise on gradient descent visualization

https://github.com/fsimone91/course_ml4hep/blob/main/notebooks/lesson2/4_Gradient_Descend_Visualize.ipynb

Jupyter notebook: install locally

1) Install [MiniConda]



Jupyter notebook: install locally

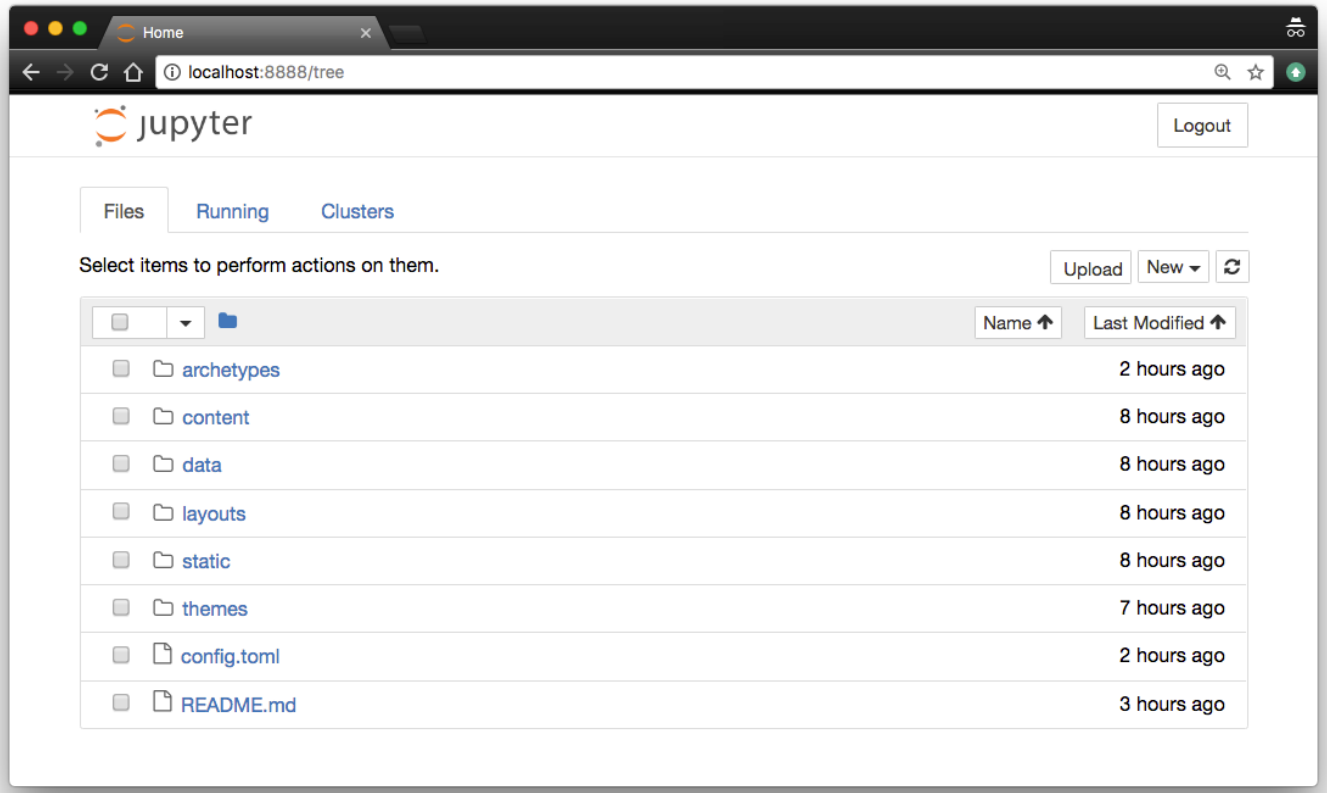
2) Create a python environment

conda create -n myenv anaconda python=3

3) Install jupyter-notebook

conda install jupyter

3) Run it! (Typically on http://**localhost**:8888)



Jupyter notebook using RECAS

Instructions:
https://www.recas-bari.it/images/manuali/JupyterNotebook_TensorFlow_GPU_EN.pdf

References and useful links

- http://scikit-learn.org/
- Pattern Recognition and Machine Learning, Bishop (2006)
- Elements of Statistical Learning (2nd Ed.) Hastie, Tibshirani & Friedman 2009
- Introduction to machine learning, Murray http://videolectures.net/bootcamp2010_murray_iml/
- What is Machine Learning, Ravikumar and Stone http://www.cs.utexas.edu/sites/default/files/legacy_files/research/documents/MLSSIntro.pdf
- CS181, Parkes and Rush, Harvard University http://cs181.fas.harvard.edu
- CS229, Ng, Stanford University http://cs229.stanford.edu/
- François Fleuret's deep-learning courses 14x050 of the University of Geneva, https://fleuret.org/dlc/#lectures

- Machine learning in high energy physics, Alex Rogozhnikov https://indico.cern.ch/event/497368/
- Introduction to Machine Learning and Deep Learning, Micheal Kagan https://indico.cern.ch/event/1293858/
- Public HEP datasets: https://iml.web.cern.ch/public-datasets
- A Living Review of Machine Learning for Particle Physics: https://iml-wg.github.io/HEPML-LivingReview/
- Machine Learning in High Energy Physics Community White Paper https://arxiv.org/abs/1807.02876

Python ML software

- Anaconda / Conda → easy to setup python ML / scientific computing environments
  - https://www.continuum.io/downloads
  - http://conda.pydata.org/docs/get-started.html
- Integrating ROOT / PyROOT into conda
  - https://nlesc.gitbooks.io/cern-root-conda-recipes/content/index.html
  - https://conda.anaconda.org/NLeSC
- Converting ROOT trees to python numpy arrays / panda dataframes
  - https://pypi.python.org/pypi/root_numpy/
  - https://github.com/ibab/root_pandas
- Scikit-learn → general ML library
  - http://scikit-learn.org/stable/
- Deep learning frameworks / auto-differentiation packages
  - https://www.tensorflow.org/
  - http://deeplearning.net/software/theano/
- High level deep learning package build on top of Theano / Tensorflow
  - https://keras.io/

Adding non-linearity

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