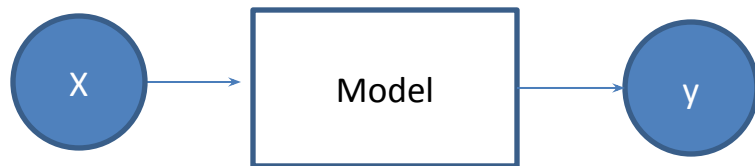


Introduction to Machine and Deep Learning

Dr. Ahmad El Sallab
Senior Expert

What AI can do?

Function approximation



Simple mapping

What AI can do?

Structured data

Price	Floor space	Rooms	Lot size	Apartment	Row house	Corner house	Detached
250000	71	4	92	0	1	0	0
209500	98	5	123	0	1	0	0
349500	128	6	114	0	1	0	0
250000	86	4	98	0	1	0	0
419000	173	6	99	0	1	0	0
225000	83	4	67	0	1	0	0
549500	165	6	110	0	1	0	0
240000	71	4	78	0	1	0	0
340000	116	6	115	0	1	0	0

Machine learning background survey 04.09.18

Please answer "Yes" to the following questions if:

This form is automatically collecting email addresses for VALEO users. [Change settings](#)

Know about basic linear algebra and matrices operations (multiplication, add, * transpose)?

☐ Yes

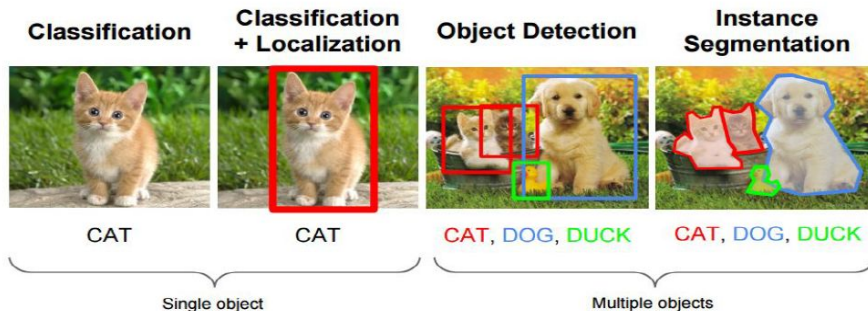
☐ No

Know how to apply differentiation and the chain rule? *

☐ Yes

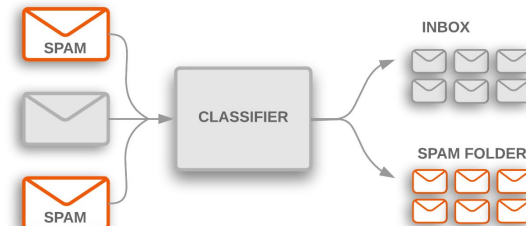
☐ No

Unstructured data



Source: Fei-Fei Li, Andrej Karpathy & Justin Johnson (2016) cs231n, Lecture 8 - Slide 8, *Spatial Localization and Detection* (01/02/2016). Available:

http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf



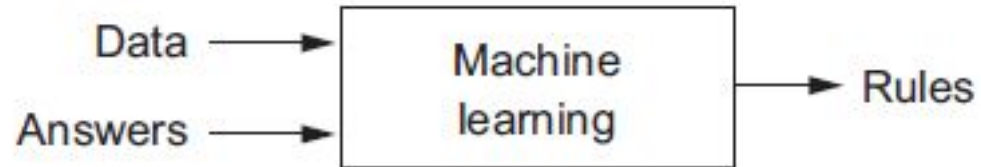
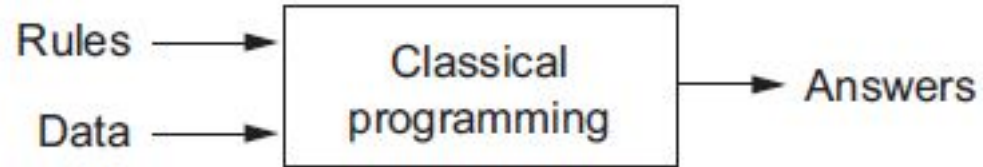
Artificial Narrow Intelligence: Structure or unstructured

Data formalization

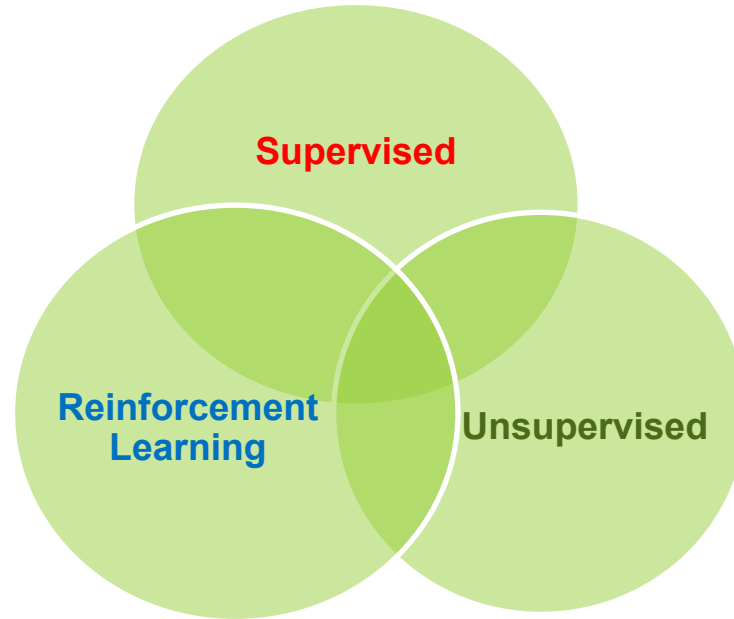
$$X = \begin{bmatrix} \text{---} (x^{(1)})^T \text{---} \\ \text{---} (x^{(2)})^T \text{---} \\ \vdots \\ \text{---} (x^{(m)})^T \text{---} \end{bmatrix}$$

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \leftarrow \text{Data}$$

Rule based vs. Learning



Learning approaches



Other types

- **Semi-supervised:** mix unsupervised and supervised
- **Self-supervised:** Self-supervised learning is supervised learning without human-annotated labels—you can think of it as supervised learning without any humans in the loop. There are still labels involved (because the learning has to be supervised by something), but they're generated from the input data, typically using a heuristic algorithm.
 - For instance, **autoencoders** are a well-known instance of self-supervised learning, where the generated targets are the input, unmodified.
 - In the same way, trying to predict the next frame in a video, given past frames, or the next word in a text, given previous words, are instances of self-supervised learning (**temporally supervised learning**, in this case: supervision comes from future input data).

Parametric Models

$$X = \begin{bmatrix} \text{---} (x^{(1)})^T \text{---} \\ \text{---} (x^{(2)})^T \text{---} \\ \vdots \\ \text{---} (x^{(m)})^T \text{---} \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \leftarrow \text{Data}$$
$$h_{\theta}(x^{(i)}) = (x^{(i)})^T \theta \quad \leftarrow \text{Model}$$

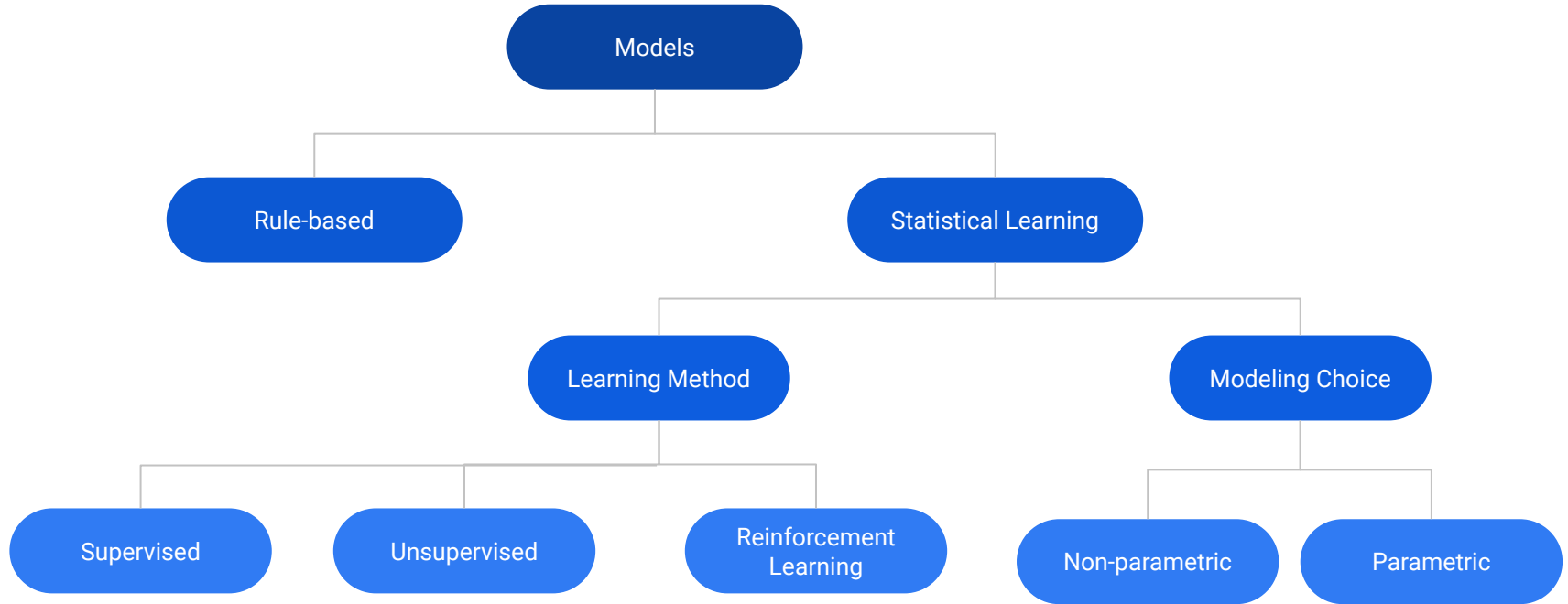
Parametric vs Non-parametric

- Parametric: assume a family of functions (=distribution) from which the model is searched. read as mapping function (model) of input X to output y , parametrized by θ

$$\hat{y} = h(X; \theta) = h_{\theta}(X)$$

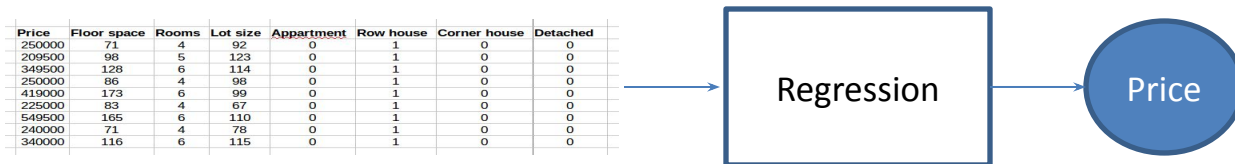
- Non-parametric: We don't assume any function family. Usually heuristics.
Example: K-NN.

Models Anatomy



What is AI is best at today?

Supervised Learning



Regression

Car Price	150K	200K	250K	?	350K
Horse Power	72	90	110	130	180

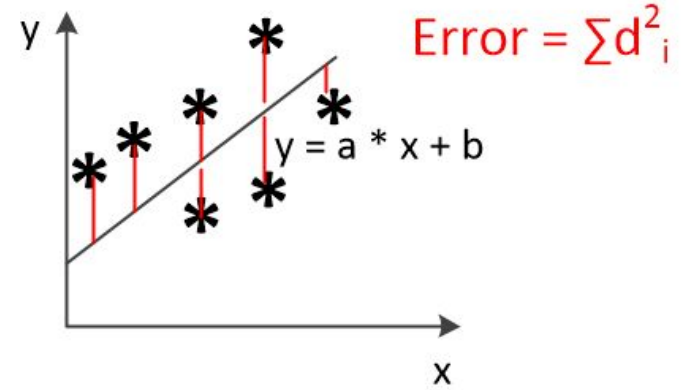
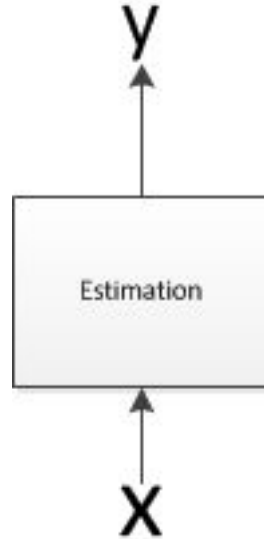
Pure DB query=fail

Need to interpolate

Job Salary	10K	20K	?	35K	50K
Domain	X	Y	X	Z	Z
Location	CAI	NY	PAR	NY	CAI
Grade	B	A	A	B	A

An output could depend on many factors = features

Functional approximation



Classification

Car Category	Eco	Low	Mid	High	Premium
Horse Power	72	90	110	130	180

Car Category



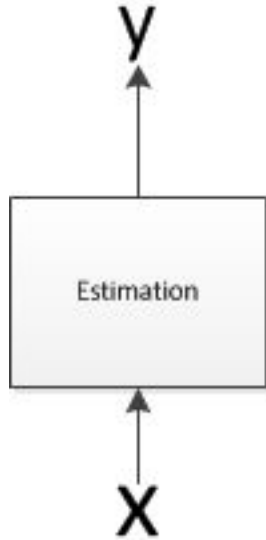
Eco=0
Low=1
Mid=2
High=3
Premium=4

Car Category



Low=0
High=1

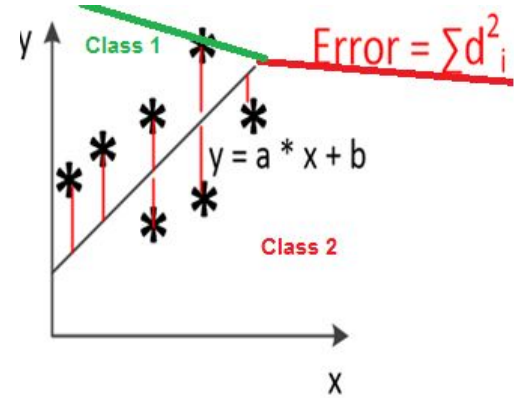
Discrimination = Classification



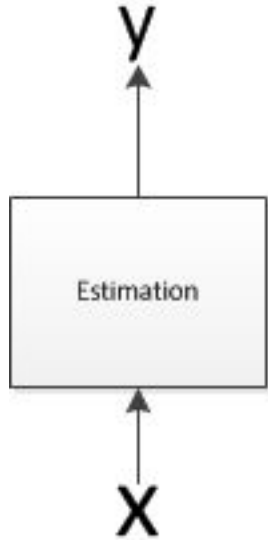
$$y_k = \varphi(x_k) + E$$

$$p(y | x) = \varphi(x)$$

$$E = \text{Desired} - \text{Obtained} = y - \varphi(x)$$

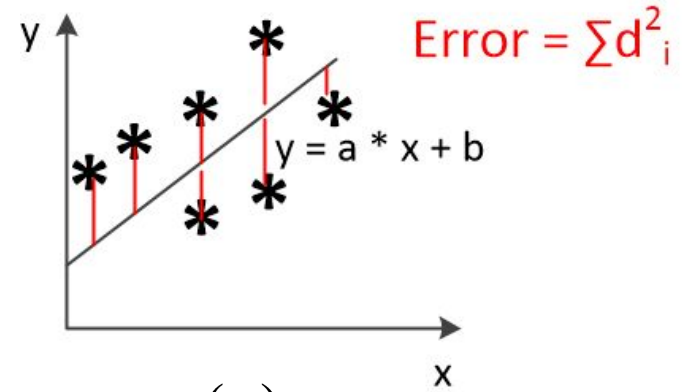


Optimization problem = Min. Error



$$y_k = \varphi(x_k) + E$$

$$p(y | x) = \varphi(x)$$



$$E = \text{Desired} - \text{Obtained} = y - \varphi(x)$$

$$J = \text{Min.} E = \frac{1}{2} \sum_{k=0}^{M-1} (y_k - \Gamma(x_k))^2 \equiv \text{Max.} p(y | x)$$

Ingredients of supervised learning

- Data:
 - X, Y (supervised)
- Model: function mapping:
 - $Y' = fw(X), w=1,2,\dots\text{etc},$
- Loss:
 - How good Model fits Data? $\rightarrow Y$ vs. Y'
- Optimizer:
 - search for best fw , that makes the Model fits the Data with min. Loss.

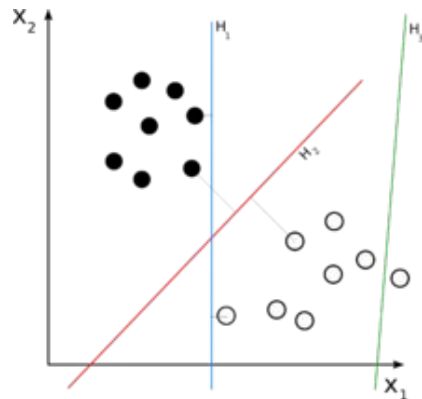
ML: How to learn w 's?

Assume only 2 q's:

Basically \rightarrow Search: $ax_1 + bx_2 + c$ (**Model**) $\rightarrow w_1 = a$,
 $w_2 = b$, $w_0 = c$ (bias, to be clear later)

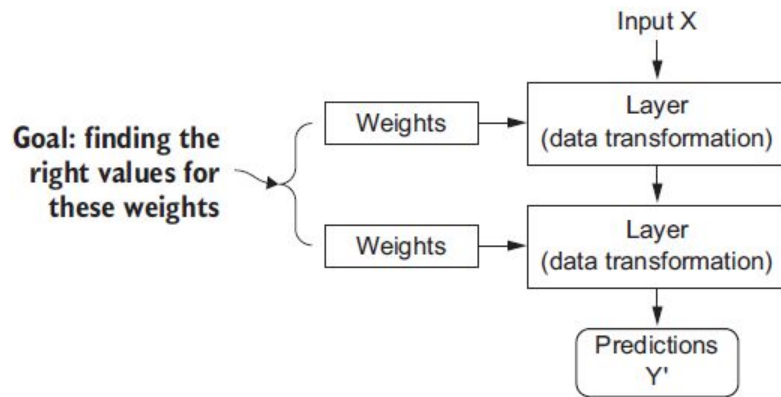
Try many lines, and get the one that separates the
Data better \rightarrow How good? **Loss**

Practically \rightarrow Smarter methods (**Optimizer**) are
used better than brute force or random search!



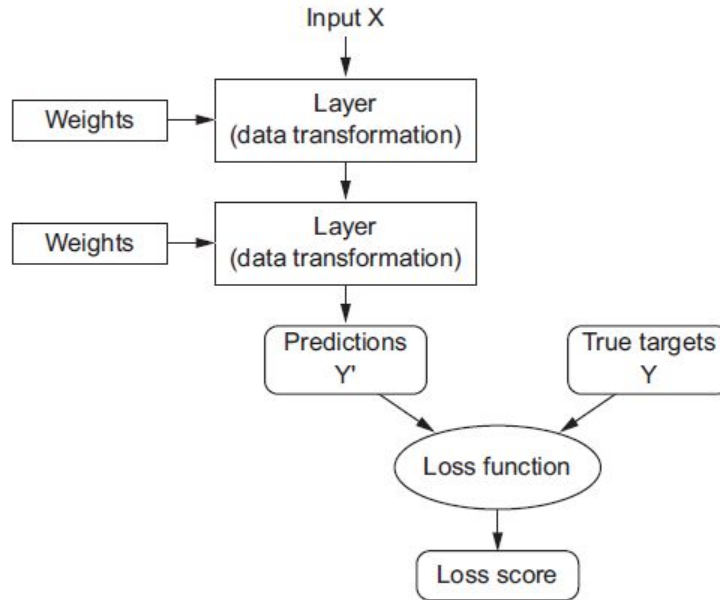
How DL works?

1. Model



How DL works?

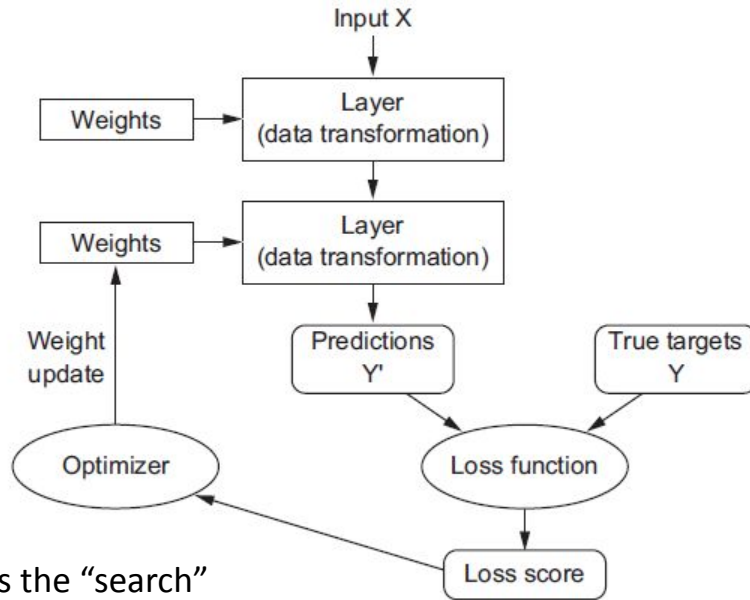
2. Data



How DL works?

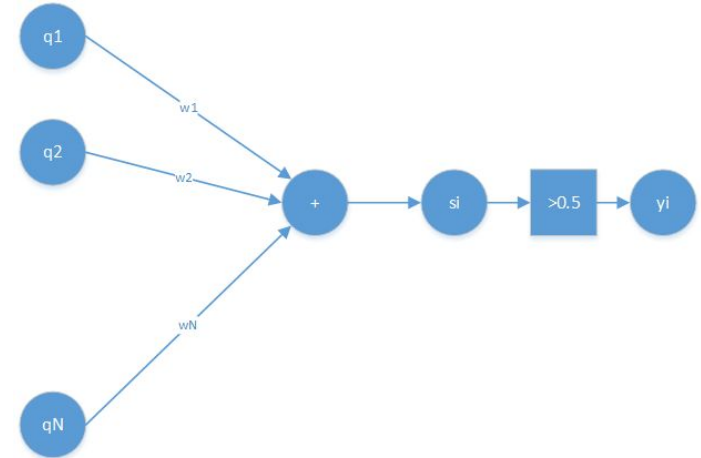
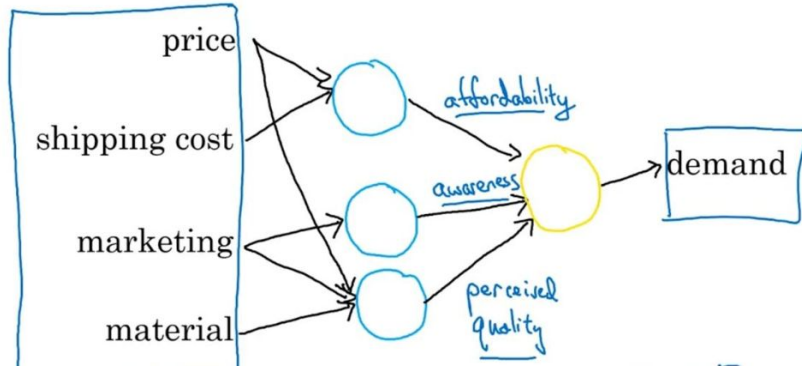
3. Loss

4. Optimizer

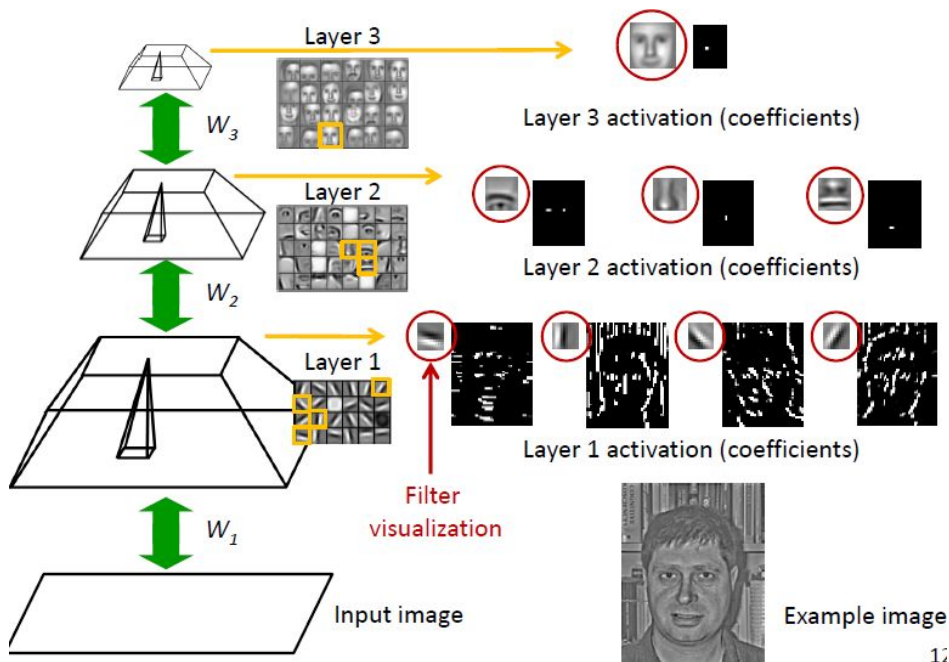
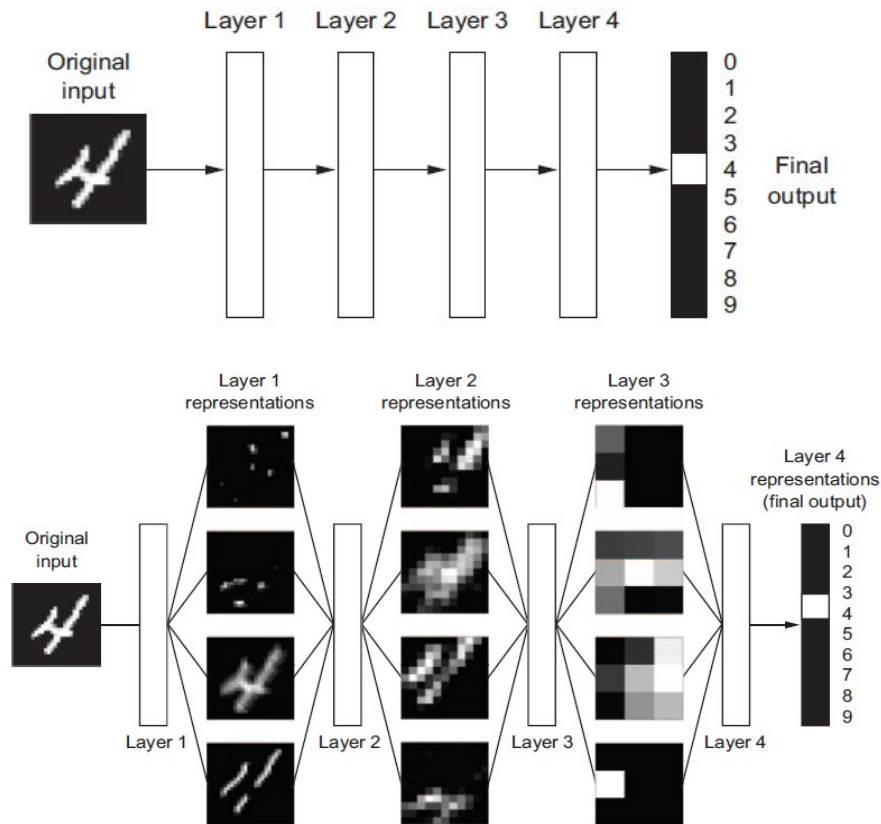


Optimizer does the “search”
Using **backpropagation**

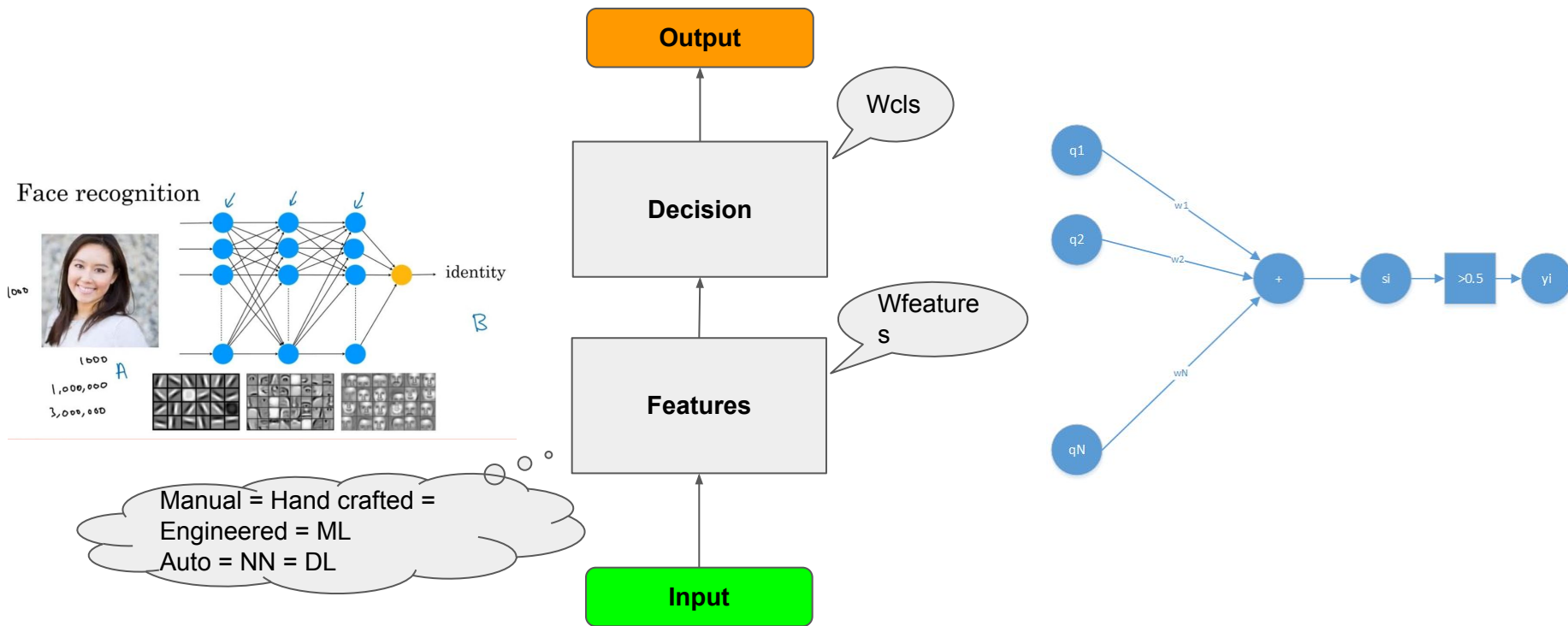
Deep or shallow?



Deep or shallow?



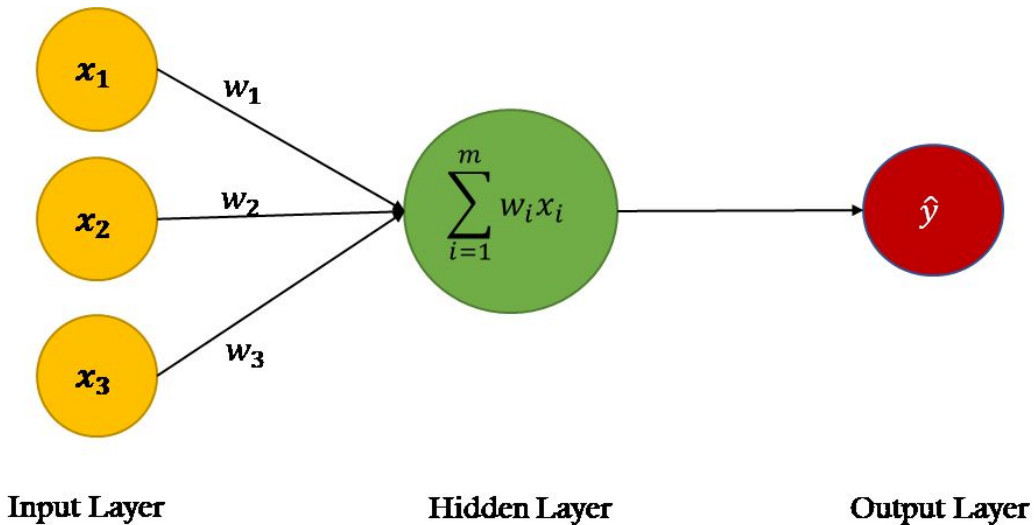
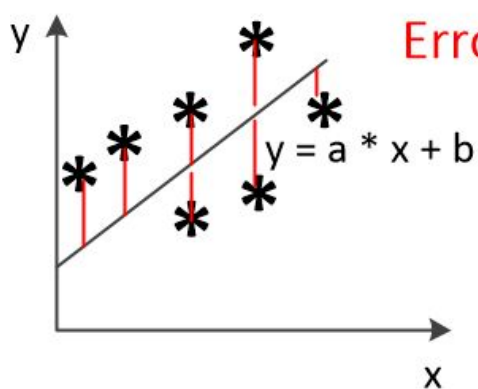
Supervised Learning Model Design Pattern



Regression Design Pattern

Model (Ex: linear regression)

How much knows ML?



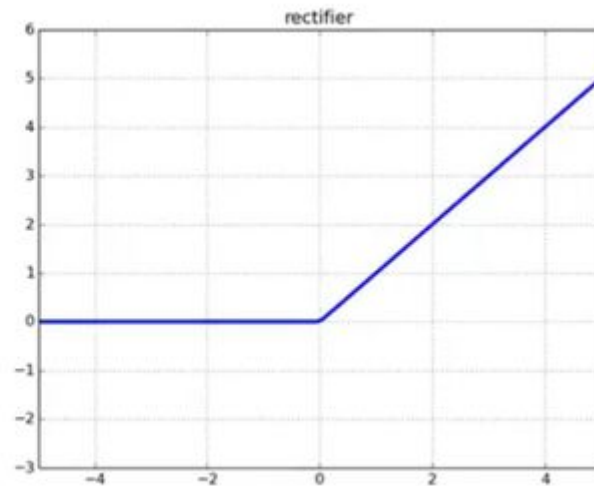
Regression Design Pattern

Decision layer/Output (Ex: linear regression)

ReLU (if negative is not permitted)

or Linear

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



Regression Design Pattern

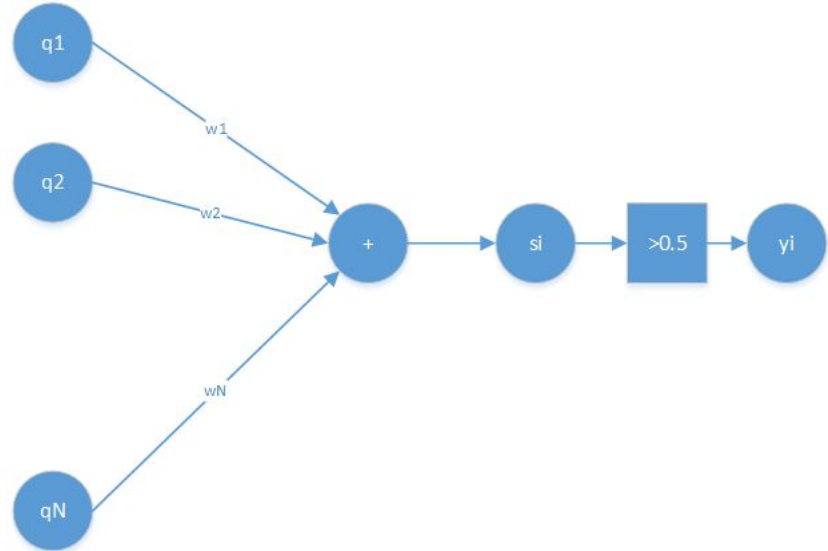
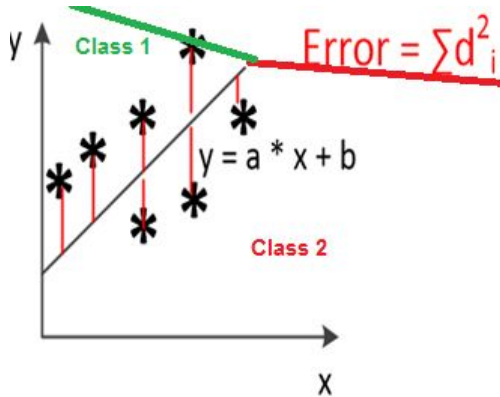
Loss (Ex: linear regression)

$$MSE = \frac{1}{n} \sum \left(\underbrace{y - \hat{y}}_{\substack{\text{The square of the difference} \\ \text{between actual and} \\ \text{predicted}}} \right)^2$$

Binary Classification

Model (Ex: Logistic regression)

Knows ML or not?



Binary Classification

Decision layer/Output (Ex: Logistic regression)

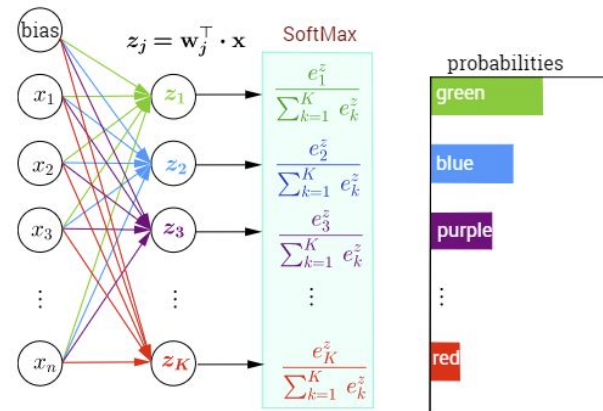
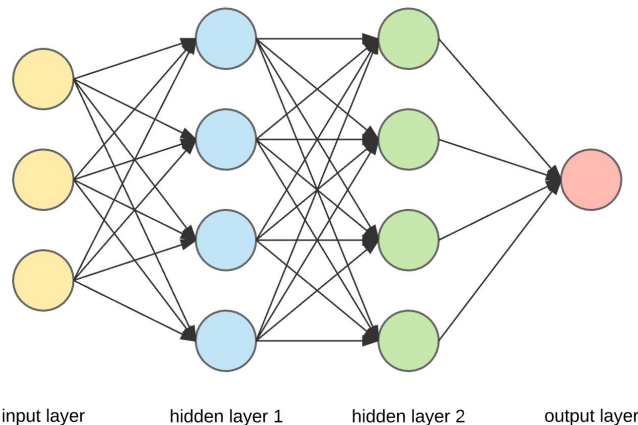
Knows ML or not?

. . . - - - - - . - - - - -

Multi-class classification

Model:

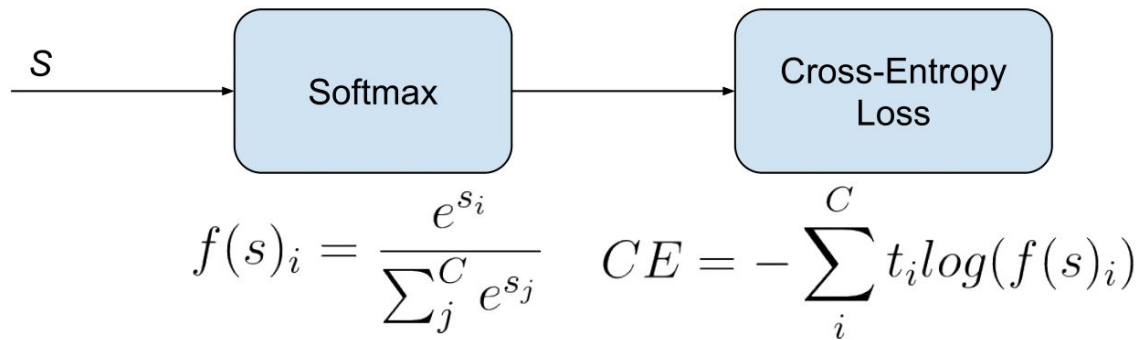
- Layers \rightarrow Wfeatures
- Output \rightarrow Multiple linear neurons (like linear regression)
- Each output neuron gives a score (unnormalized probability or logit)
- Softmax \rightarrow Normalize
- Prediction = argmax
- Ensures only 1 class is possible



$$\sigma(j) = \frac{\exp(\mathbf{w}_j^T \mathbf{x})}{\sum_{k=1}^K \exp(\mathbf{w}_k^T \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)}$$

Multi-class classification

Loss → Cross-entropy



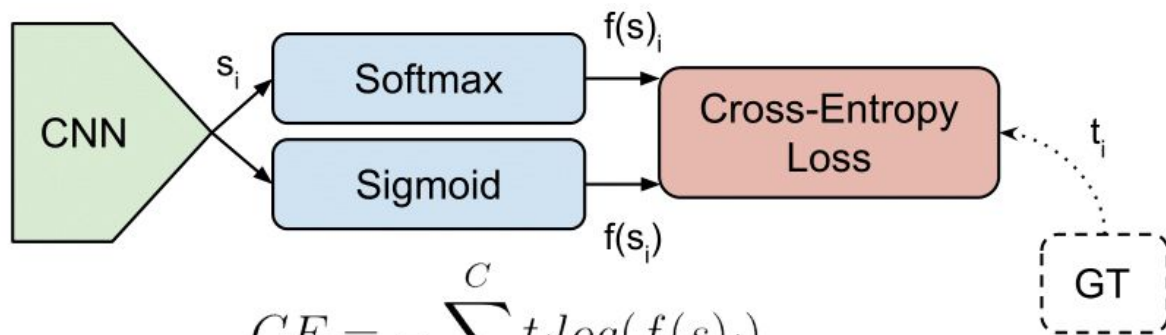
Only 1 class is possible

$t_i \rightarrow$ OHE \rightarrow We focus on the true class only \rightarrow If we maximize this one, the others will be damped, since they all sum to 1 in the softmax

$$CE = \sum_i^K t_i \times \log f(s)_i$$

$$CE = \sum_i^K f(s)_i^{t_i}$$

Binary Cross Entropy



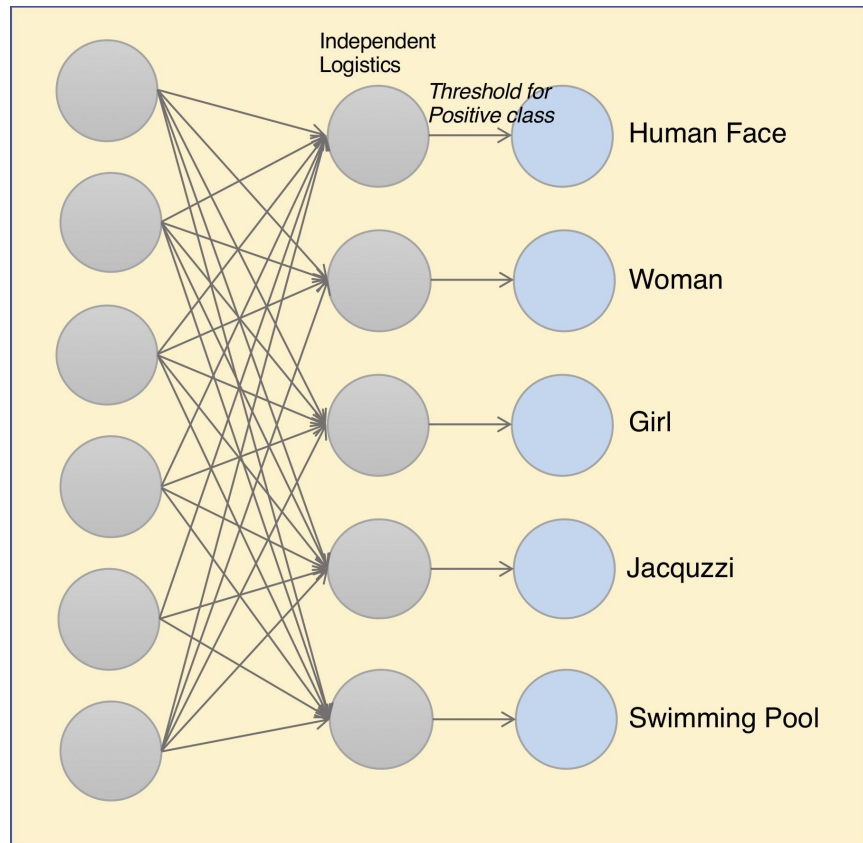
$$CE = - \sum_i^C t_i \log(f(s)_i)$$

$$CE = - \sum_{i=1}^{C'=2} t_i \log(f(s_i)) = -t_1 \log(f(s_1)) - (1 - t_1) \log(1 - f(s_1))$$

Multi-label Classification

Model:

- Layers
- Multiple sigmoids

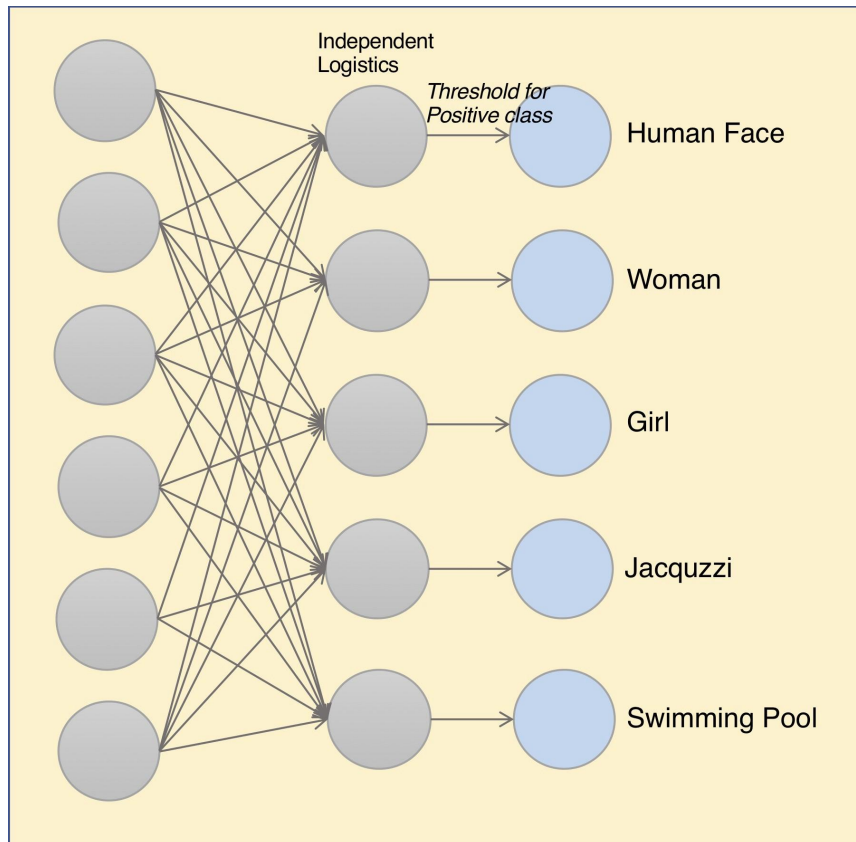


Multi-label Classification

Loss: BCE over each neuron

→ Ask every Neuron, is it Human? Is it person?,...

More than 1 is possible



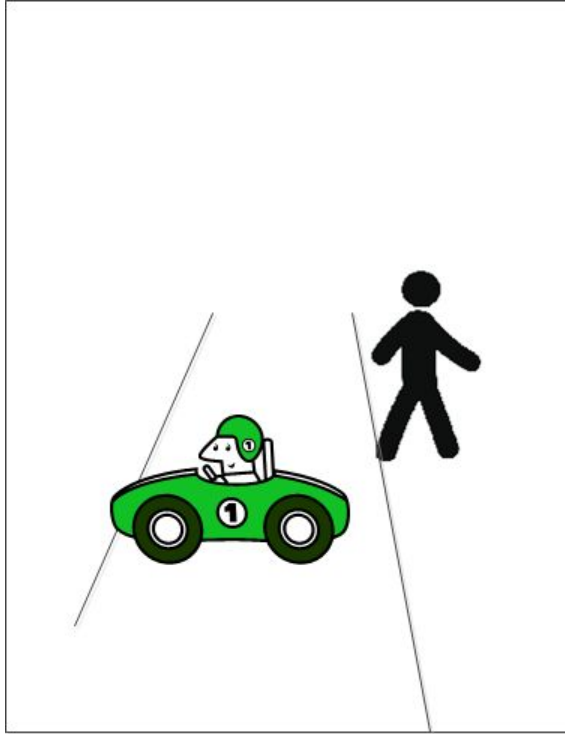
Model vs. Problem summary

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

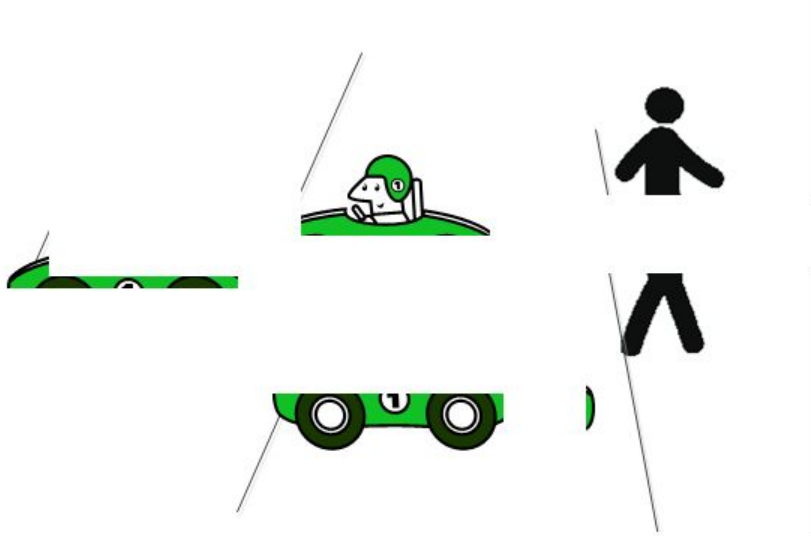
How optimizer works for Parametric models?

- How ? $\hat{y} = h(X; \theta) = h_{\theta}(X)$
 - **Hypothesis space**: We have many possible transforms: f_1, f_2, \dots
 - **Learning**: Search in the *hypothesis space*

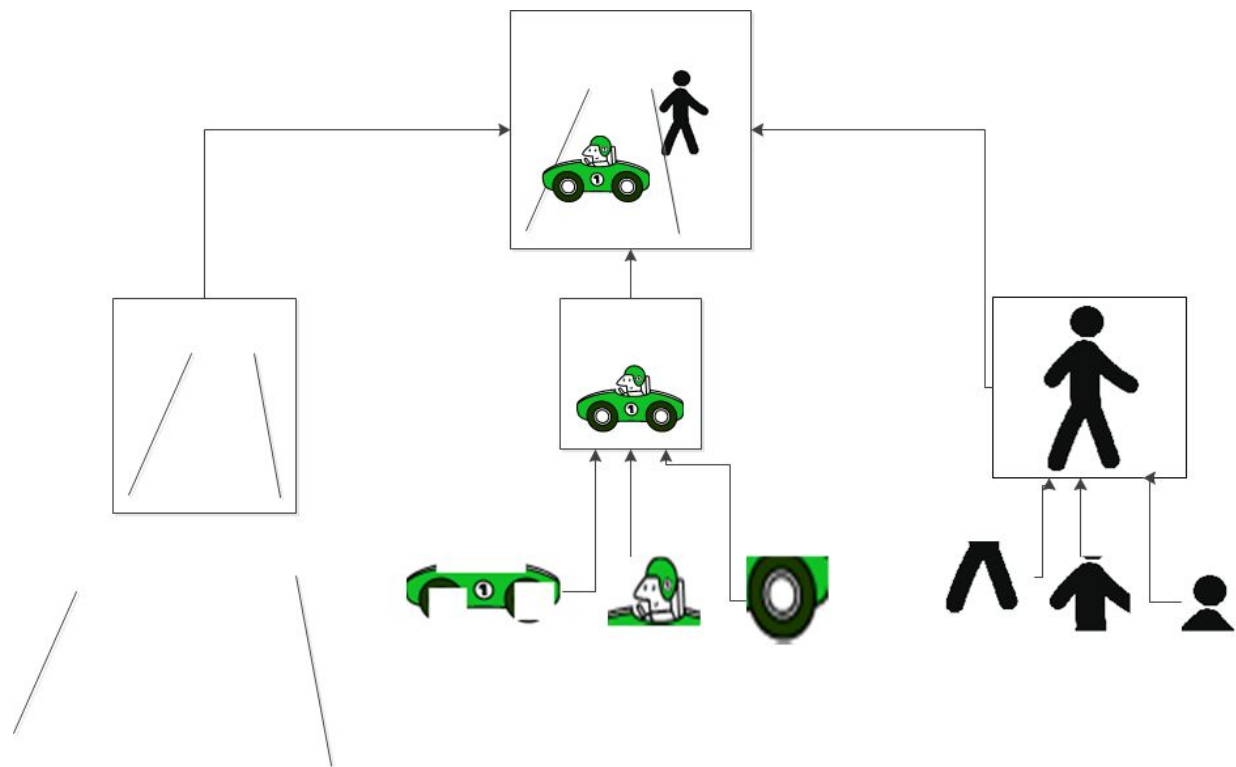
Puzzle Reconstruction



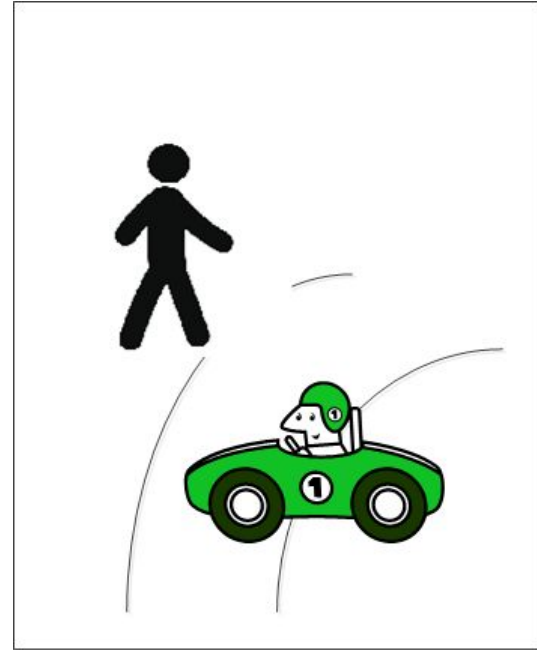
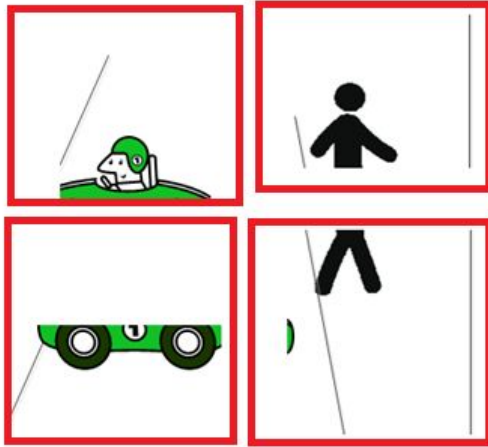
Puzzle Reconstruction



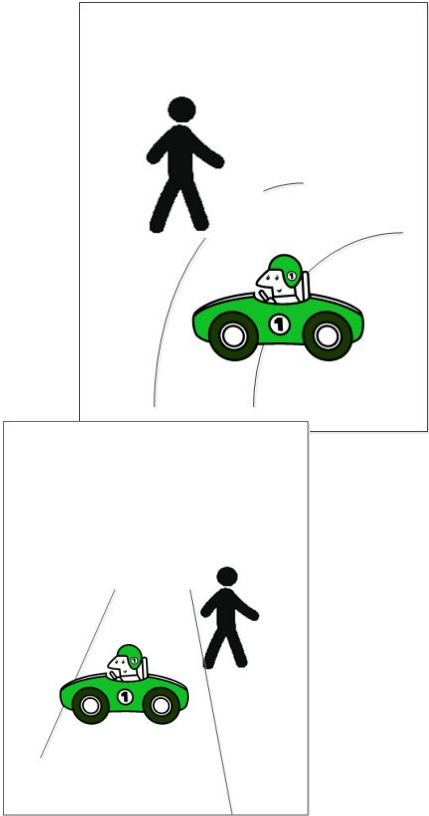
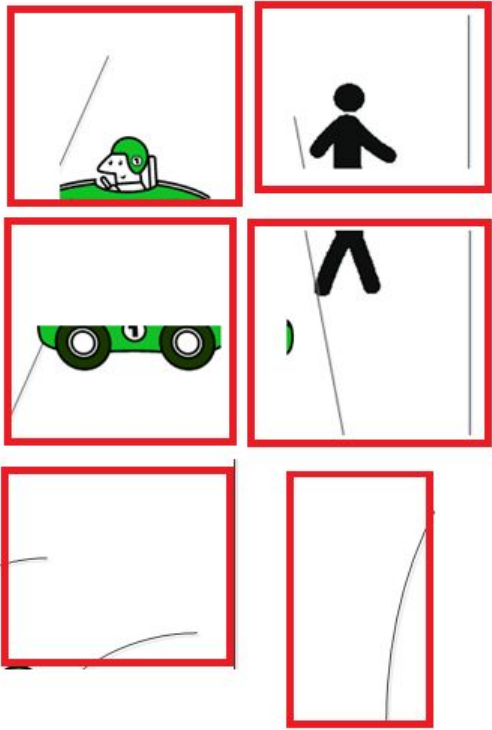
Deconstruction - Reconstruction



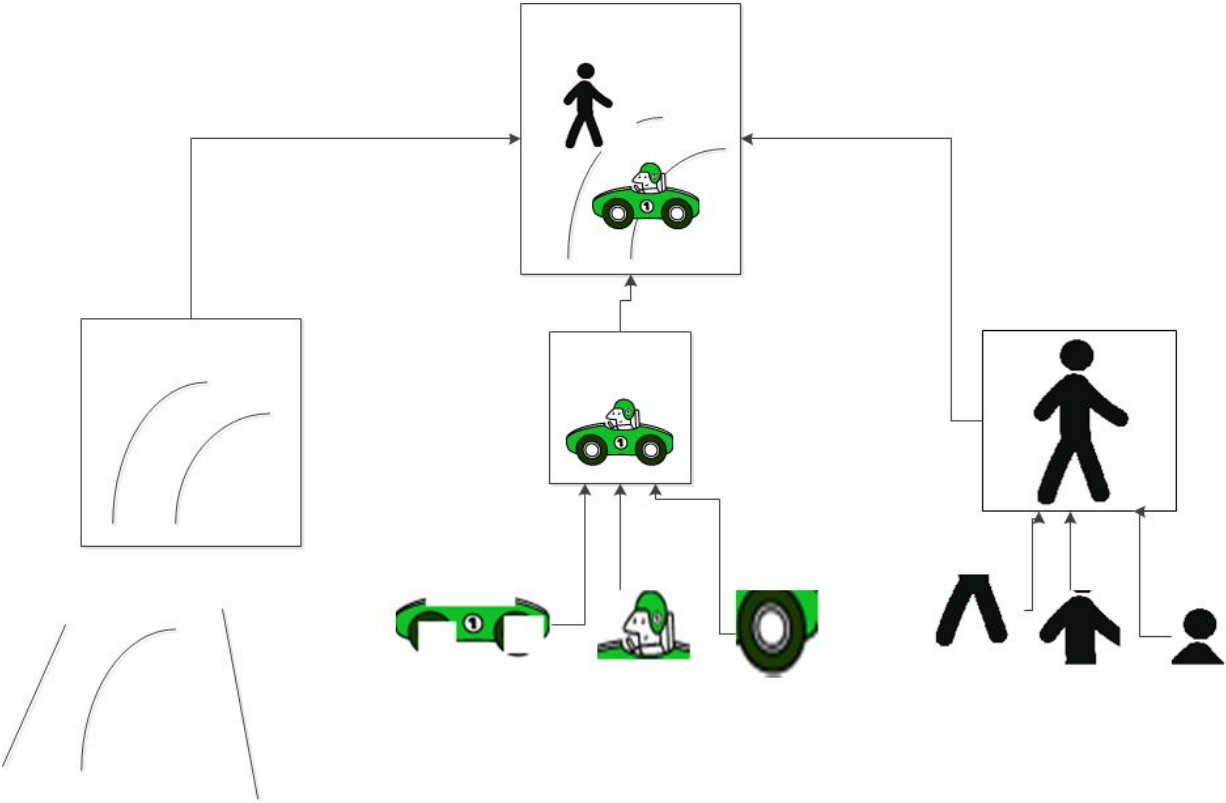
Generic puzzle?



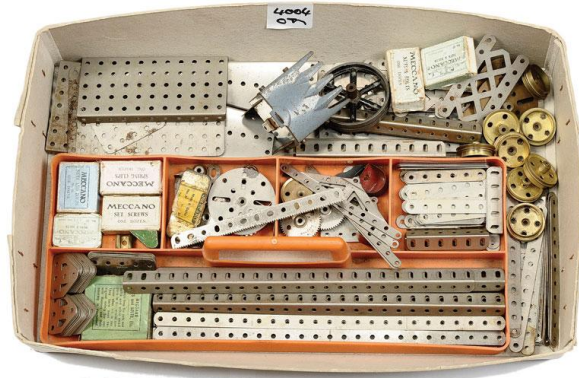
Generic puzzle?



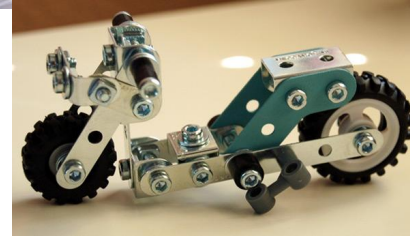
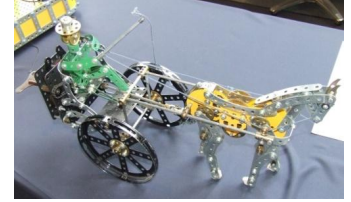
Deconstruction - Reconstruction



Generic re-usable units



Copyright © 2012 Vectra Auctions. All Rights Reserved.

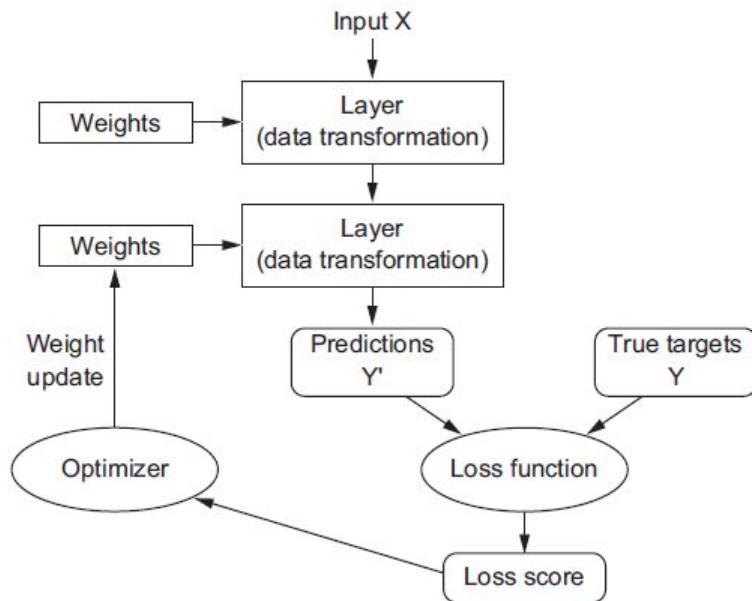


How optimizer works for Parametric models?

- Option 1: **Closed form solution** Let's solve the equations!
- Option 2: **Search !**
 - **Brute force**: try all possible values of w 's and choose the set with min loss → **Numerical solution = optimization**
 - **Random search**: same as brute force, but choose random subset of all possible w 's
 - **Guided search**: start from random w 's, measure the loss, then choose the next set of w 's, *guided by* the change in the loss.

On which data to compute the loss?

How to use the loss change as a guidance?



On which data to compute the loss?

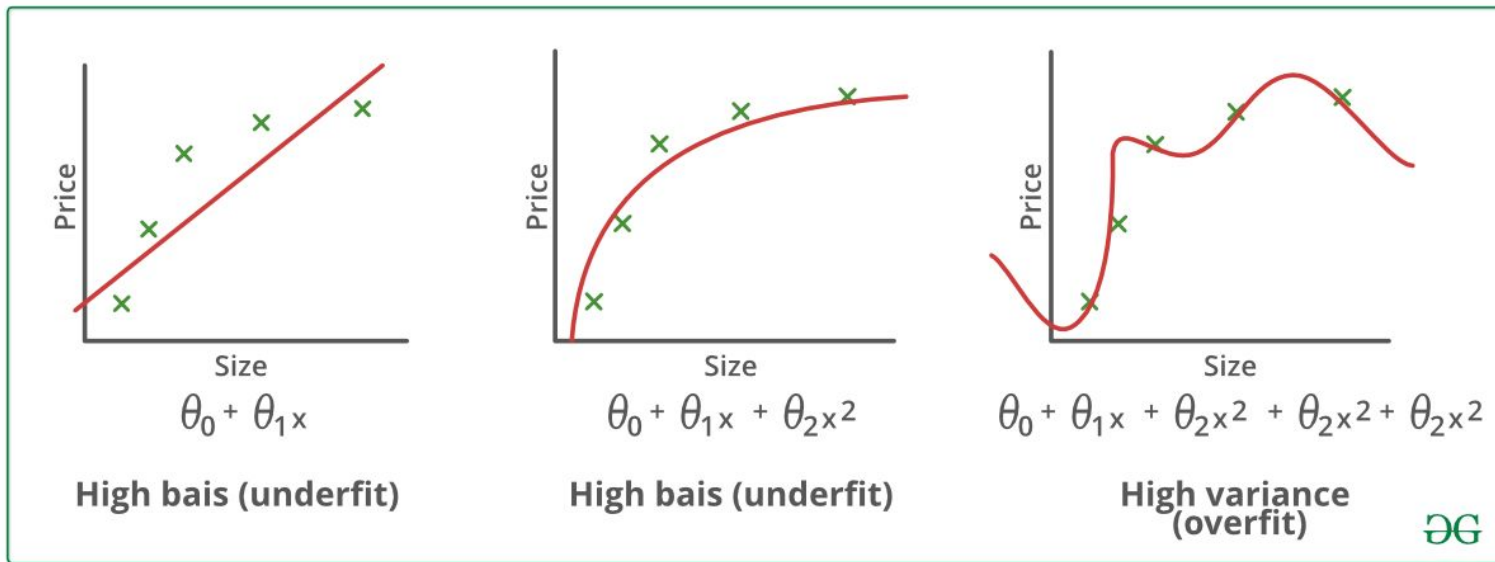
Search in training data → Loss evaluation + Parameters update

Evaluate on unseen testing data → Loss evaluation but No parameters update allowed

Model Choice for Parametric models - Regression

As we increase the number of features, and add more non-linearity (more layers), we get a better curve that fits the data.

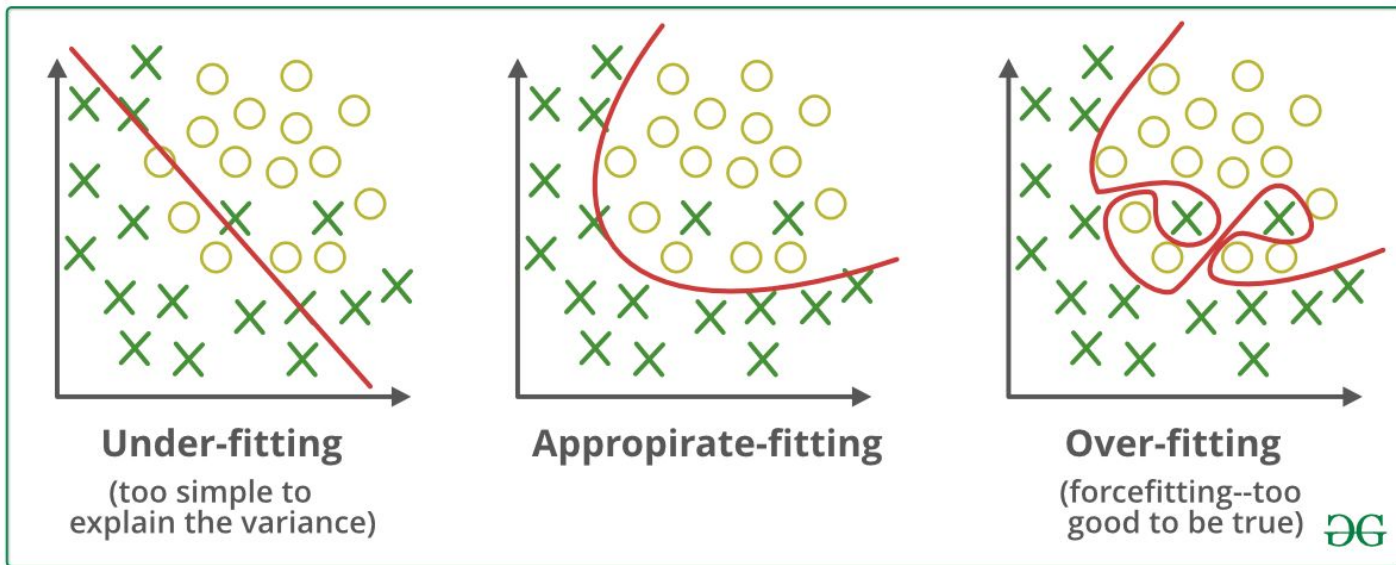
$$\hat{y} = h(X; \theta) = h_{\theta}(X)$$



Model Choice for Parametric models - Classification

As we increase the number of features, and add more non-linearity (more layers), we get a better curve that fits the data.

$$\hat{y} = h(X; \theta) = h_{\theta}(X)$$



Bias and Variance

The error of prediction is composed of two main contributions, the remaining **interference error** and the **estimation error** [39].

1- The **interference error is the systematic error** **__(bias)__** due to **unmodeled interference** in the data, as the calibration model is not complex enough to capture all the interferences of the relationship between sensor responses and analytes. **__training data modeling__**

__High bias means underfitting__

2- The estimation error is caused by **modeling measured random noise** of various kinds. In other words, the **__variance__** between the training and testing data (variance is a measure of noise, or how much variability is there between the training (mean) data and testing data).

__High variance means overfitting__

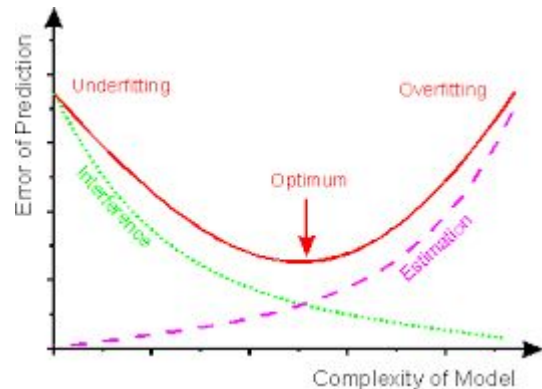
The optimal prediction is obtained, when the remaining **interference error** and the **estimation error** balance each other.

[src](http://www.frank-dieterle.de/phd/2_8_1.html)

The number of parameters is a degree of freedom!

Number of parameters = Model capacity

The optimizer is free to use whatever number.



However, optimization process could set some of them to 0's or small values.

Regularization

We could put some penalty on the loss:

- Loss is a measure of “badness”
- Extra term, means extra “badness”=penalty
- Even if the basic loss sees this choice of params as good, we increase its badness by extra term
- Ridge: L2
- Lasso: L1

$$L(x, y) = \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2$$

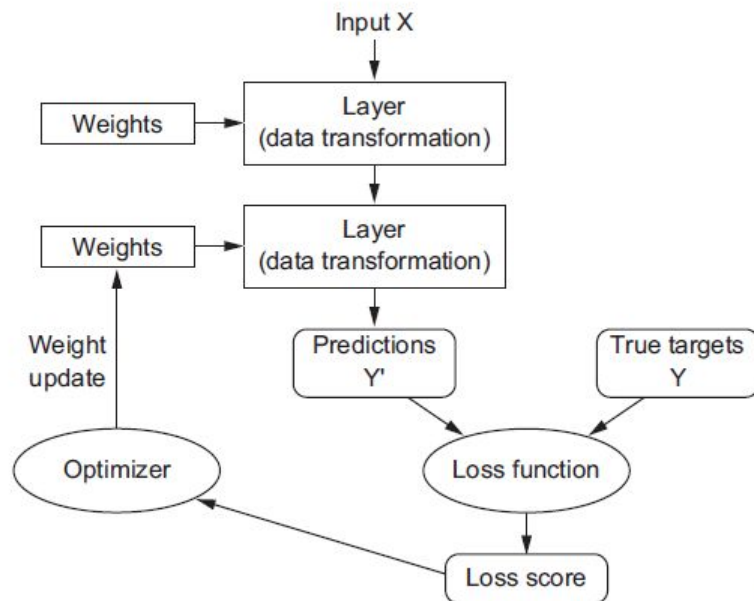
$$\text{where } h_{\theta}x_i = \theta_0 + \theta_1x_1 + \theta_2x_2^2 + \theta_3x_3^3 + \theta_4x_4^4$$

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$

How to use the loss change as a guidance?

Closed form solution (Normal equation)

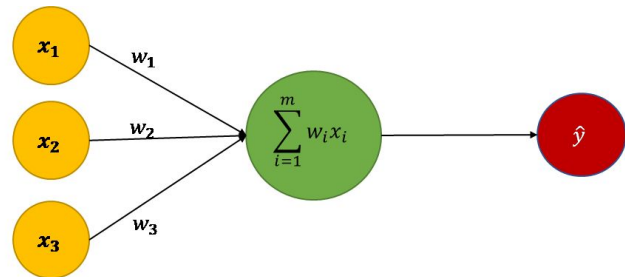
Numerical solution (Gradient based)



Normal equation

N-equations, in N unknowns

$$\frac{\partial J}{\partial \theta_j}, \text{ for } j = 1..N$$



Input Layer

Hidden Layer

Output Layer

$$X = \begin{bmatrix} - & (x^{(1)})^T & - \\ - & (x^{(2)})^T & - \\ & \vdots & \\ - & (x^{(m)})^T & - \end{bmatrix}$$

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

← Data

$$h_{\theta}(x^{(i)}) = (x^{(i)})^T \theta$$

← Model

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

← Objective

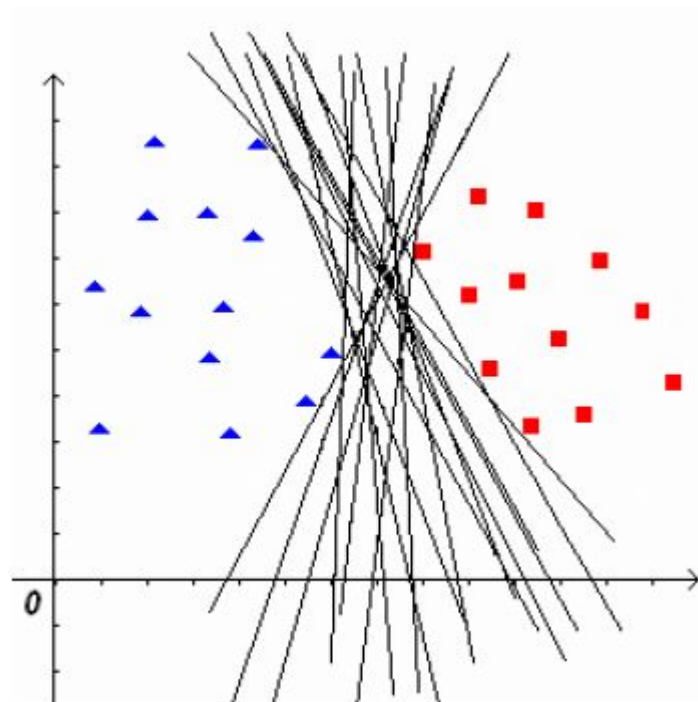
$$\theta = (X^T X)^{-1} X^T \vec{y}$$

← Normal equation

Normal equation

Disadv:

- Slow (X is a big matrix, inverse is costly and not always existing)
- Uses all the data points --> Might overfit (more on that later)
- It doesn't work for classification. Only regression, since some equations will vanish (correct classifications), leading to more degrees of freedom and less constraints, and we'll have infinite solutions (infinite lines can separate the classes with 0 MSE)



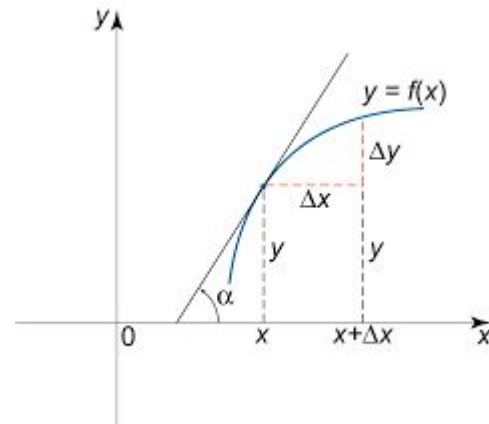
Gradient based optimization

A gradient is a of change of output, with change in input.

Meaning how much y changes when we change x by small amount.

In other words, it's the rate of change of y with x .

It also encodes the sensitivity of y to x



In case of loss, the gradient $\frac{\Delta \text{loss}}{\Delta w}$ is how much the loss changes when we change a weight with small amount. It guides our search of the the weights that minimizes the loss; if we change w a bit, and the loss is affected a lot, it means that the loss is sensitive to this change.

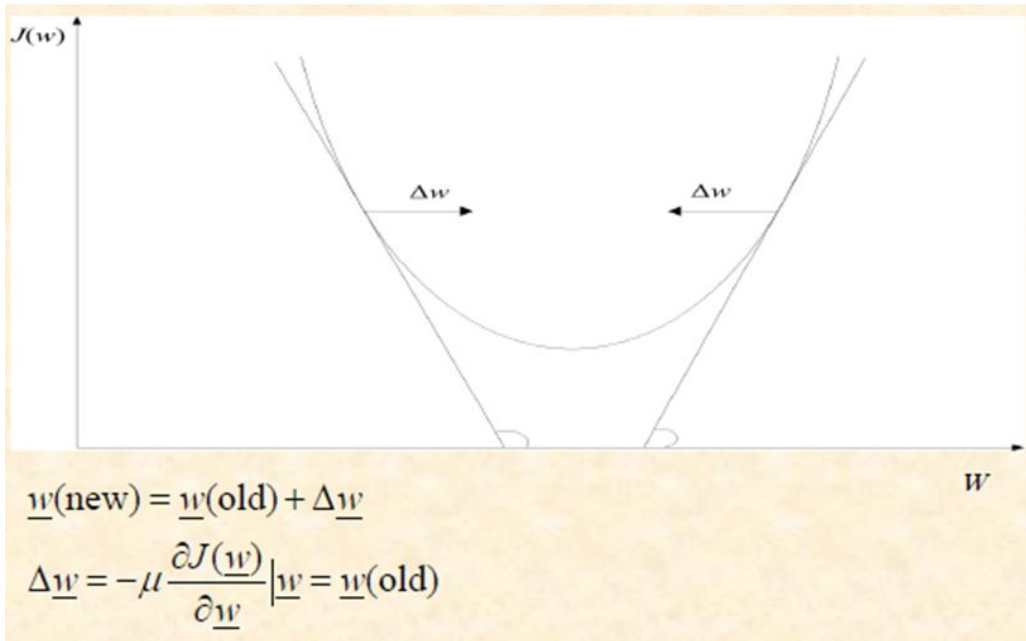
In case of MSE loss for example, since the loss is always convex, it means any change in w moves the loss away from the minimum. So if we move against it, we reach the minimum. If this change value is small, it means we need to move a small step (we are already at the min: $\frac{\Delta \text{loss}}{\Delta w} = 0$), and vice versa.

Gradient descent

When $w_{\text{new}} = w_{\text{old}}$, then the
grad = 0

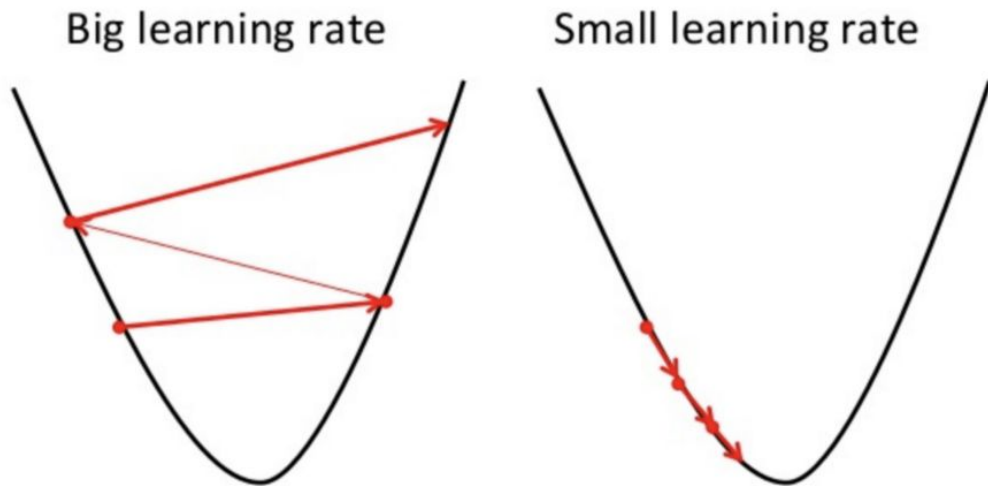
So we reach the min. (same as
we wanted in normal equation
and closed form solution)

grad = Δw → **Numerical
estimate of the true gradient
(dJ/dW)** → **How to calc that
estimate?**



Learning rate

As you can see, intuitively it's important to pick a reasonable value for the step factor. If it's too small, the descent down the curve will take many iterations, and it could get stuck in a local minimum. If step is too large, your updates may end up taking you to completely random locations on the curve.



Training loop

For number of epochs

For each example in the dataset:

1. Feedfwd: Compute err
2. Feedback: Compute Gradient
3. Update params

Repeated representation of data

- Training Loops
- For 1..Num epochs
 - For 1..Num examples (M)
 - GD Updates: $w(k+1) = w(k) + \text{delta_W (M)}$

Repeated representation of data

- Training Loops with batches
- For 1..Num epochs
 - For 1..Num batches (N_batches)
 - For 1..Num of examples per batch (B)
 - GD Updates: $w(k+1) = w(k) + \text{delta_W}(B)$

Let's code

Linear regression (Playing with lines):

https://colab.research.google.com/drive/1Oc0D_jr5Rmqvhf7315zjDY7Vj8Sr9RGV

Normal Equation vs SGD + Ridge:

<https://colab.research.google.com/drive/1lOtIEE9-V4mfViKF9yt602ROPesjX3uT>

Logistic Regression:

<https://colab.research.google.com/drive/1wIKM-QTJ02EGbk-YN9e05abAeswWzNRL>

Effect of batch size

- **Batch Gradient Descent.** Batch size is set to the total number of examples in the training dataset (M)
- **Stochastic Gradient Descent.** Batch size is set to one.
- **Minibatch Gradient Descent.** Batch size ($=B$) is set to more than one and less than the total number of examples in the training dataset.

For shorthand, the algorithm is often referred to as stochastic gradient descent regardless of the batch size.

Objective: $B=M$

However, given that very large datasets are often used to train deep learning neural networks, the batch size is rarely set to the size of the training dataset.

Batch size only affecting convergence speed, but not performance

Different training loop options

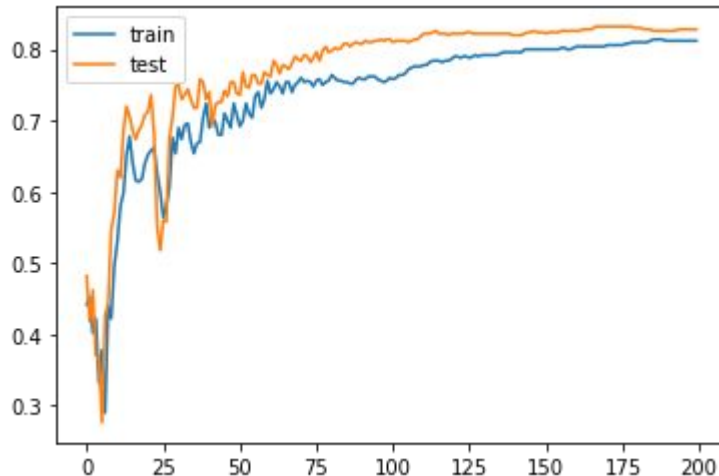
GD (batch SGD)

The above GD algorithm updates the gradient with every new sample of x .

If X is of 1000 samples, then the $\text{delta_W}[t+1] = \text{step} * \text{gradient}(f)(W[t])$ is calculated and accumulated every sample, but the application of weight update is done once after all 1000 samples are fed.

- Update once at the end
- High accumulation of error, Risk of saturation
- Slow feedback, slow convergence, but more stable
- Take advantage of parallelism in matrix operations (comp. arch.)

https://colab.research.google.com/drive/11_-SxhdtxvRYdPukURz-cojd7-JdpiKR?authuser=1



Different training loop options

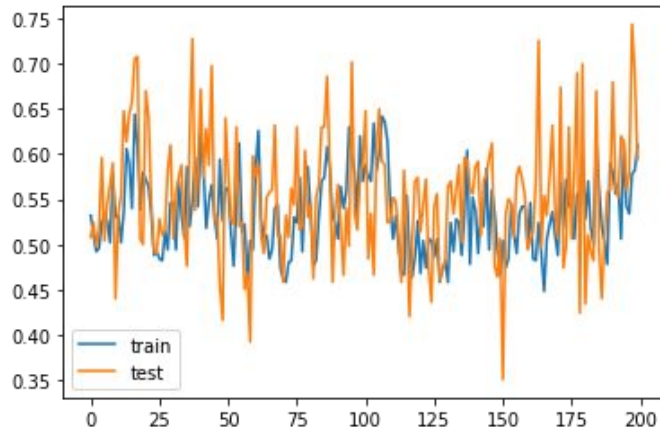
SGD

There are other options of updating:

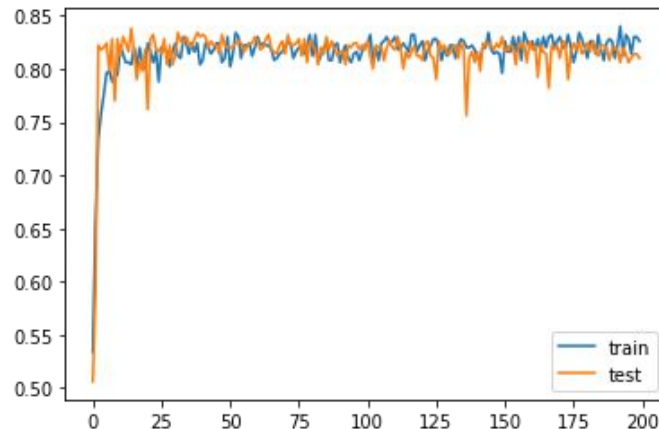
1. SGD: Each sample (batch_size=1)

- Fast feedback, fast convergence (corrects itself a lot)
- Could oscillate, unstable (tend to corrupt what it learnt) → To overcome reduce LR
- Not taking advantage of parallelism in matrix operations (comp. arch.)
- Not saturating
- Stochastic: every update is a representing sample of the true gradient

https://colab.research.google.com/drive/11_-SxhdtxvRYdPukURz-c_ojd7-JdpiKR?authuser=1



SGD + High LR



SGD + Low LR

SGD vs BDG

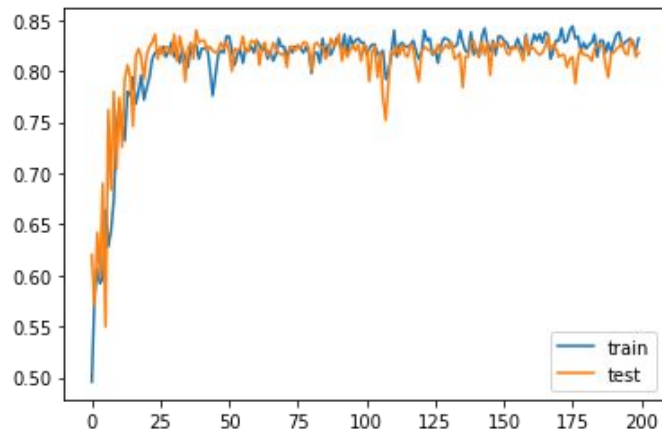
- **Batch Gradient Descent:** Use a relatively larger learning rate and more training epochs.
- **Stochastic Gradient Descent:** Use a relatively smaller learning rate and fewer training epochs. Mini-batch gradient descent provides an alternative approach.

Different training loop options

Minibatch SGD: Every group of samples (`batch_size=N`)



Accumulate M gradients, update every M

- More stable than 1 sample
- Faster than GD



https://colab.research.google.com/drive/11_-SxhdtxvRYdPukURz-cojd7-JdpiKR?authuser=1

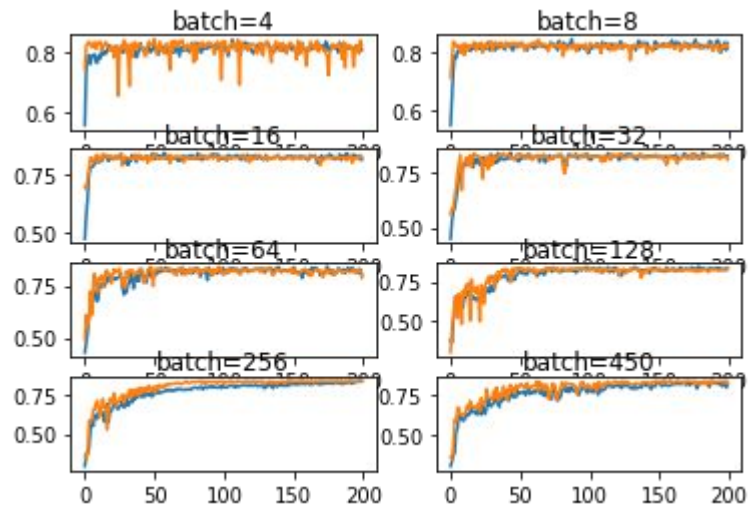
Training loop options/Effect of Batch size

	Batch_sz		
SGD	1	Fast feedback, fast convergence (corrects itself a lot) No risk of gradient saturation	Needs to reduce the LR since we make many corrections (could oscillate) Inaccurate gradient estimate
BGD	M (total samples)	Take advantage of parallelism in matrix operations (comp. arch.) → Better gradient estimate → More stable	Update once at the end → slow convergence High accumulation of error → Risk of saturation
MBGD	B ($N_{\text{batches}} = M/B$)	Good estimates than SGD Faster than BGD	Worse estimates than BGD (more oscillations) Slower than SGD

Effect of batch size in MBGD

The plots show that small batch results generally in rapid learning but a volatile learning process with higher variance in the classification accuracy.

Larger batch sizes slow down the learning process but the final stages result in a convergence to a more stable model exemplified by lower variance in classification accuracy.

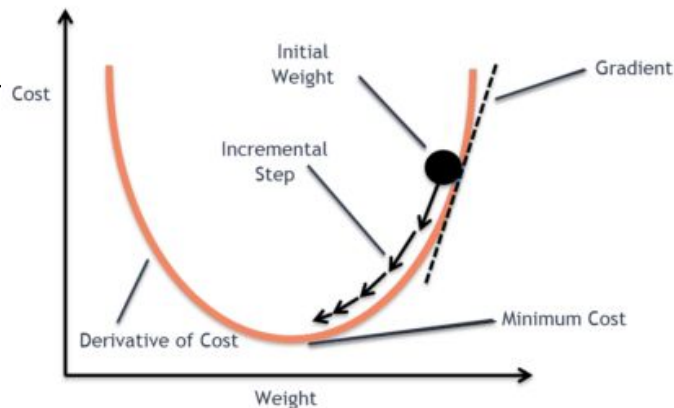


Momentum

Additionally, there exist multiple variants of SGD that differ by taking into account previous weight updates when computing the next weight update, rather than just looking at the current value of the gradients.

There is, for instance, SGD with momentum, as well as Adagrad, RMSProp, and several other Momentum draws inspiration from physics.

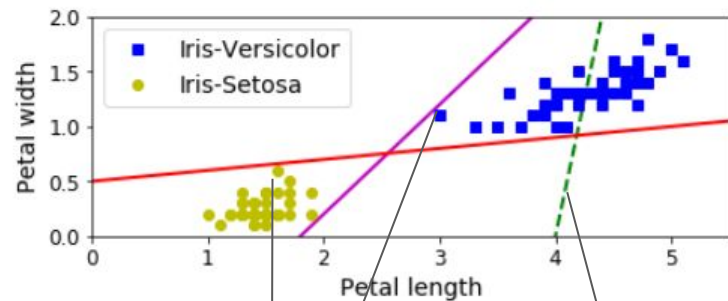
A useful mental image here is to think of the optimization process as a small ball rolling down the loss curve. If it has enough momentum, the ball won't get stuck in a ravine and will end up at the global minimum. Momentum is implemented by moving the ball at each step based not only on the current slope value (current acceleration) but also on the current velocity (resulting from past acceleration). In practice, this means updating the parameter w based not only on the current gradient value but also on the previous parameter update, such as in this naive implementation:



Large Margin Classifiers (Optional)

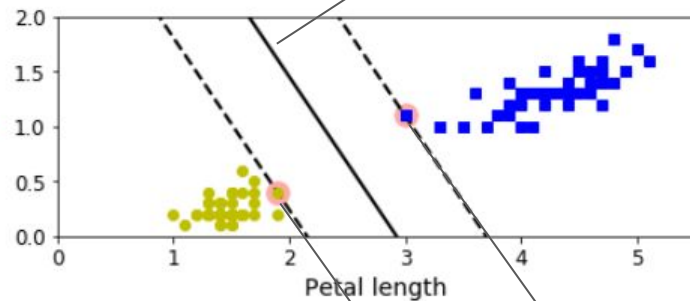
Large Margin Decision Boundary

Saving figure large_margin_classification_plot



High sensitivity
Outliers?

Worst boundary



this line not only separates the two
classes but also stays as far away from
the closest training instances as
possible.

SVM classifier as fitting the widest possible street
(represented by the parallel dashed lines)
between the classes.
This is called **large margin classification**.

Linear SVM

SVM classifier as fitting the widest possible street (represented by the parallel dashed lines) between the classes. This is called **large margin classification**.

```
from sklearn.svm import SVC
from sklearn import datasets

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)] # petal length, petal width
y = iris["target"]

setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]
y = y[setosa_or_versicolor]

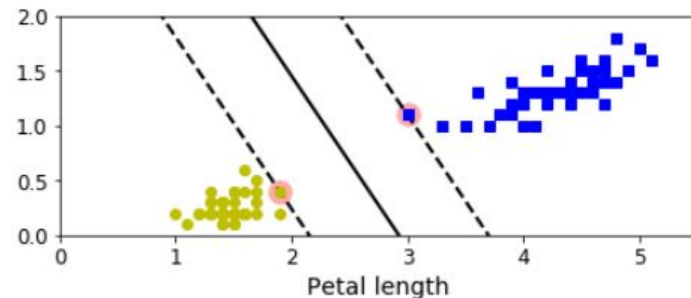
# SVM Classifier model
svm_clf = SVC(kernel="linear", C=float("inf"))
svm_clf.fit(X, y)
```

Support Vectors

Notice that **adding more training instances “off the street” will not affect the decision boundary at all:**

it is fully determined (or **“supported”**) by the instances located on the edge of the street.

These instances are called the **support vectors** (they are circled in the figure)



```
def plot_svc_decision_boundary(svm_clf, xmin, xmax):
    w = svm_clf.coef_[0]
    b = svm_clf.intercept_[0]

    # At the decision boundary, w0*x0 + w1*x1 + b = 0
    # => x1 = -w0/w1 * x0 - b/w1
    x0 = np.linspace(xmin, xmax, 200)
    decision_boundary = -w[0]/w[1] * x0 - b/w[1]

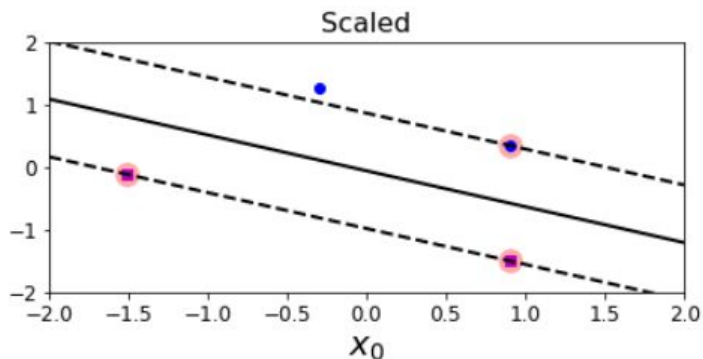
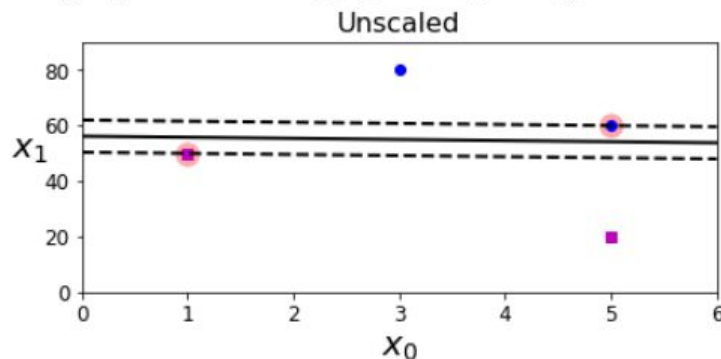
    margin = 1/w[1]
    gutter_up = decision_boundary + margin
    gutter_down = decision_boundary - margin

    sv = svm_clf.support_vectors_
    plt.scatter(svs[:, 0], sv[:, 1], s=180, facecolors='FFAAAA')
    plt.plot(x0, decision_boundary, "k-", linewidth=2)
    plt.plot(x0, gutter_up, "k--", linewidth=2)
    plt.plot(x0, gutter_down, "k--", linewidth=2)
```


Sensitivity to feature scales

SVMs are sensitive to the feature scales, as you can see below: on the left plot, the vertical scale is much larger than the horizontal scale, so the widest possible street is close to horizontal.

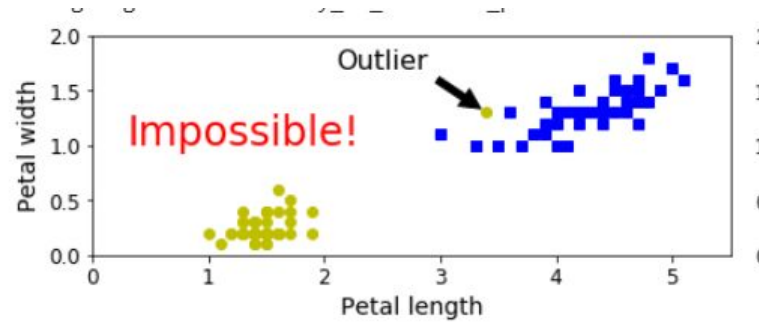
Saving figure sensitivity_to_feature_scales_plot



```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(Xs)
svm_clf.fit(X_scaled, ys)
```

Sensitivity to outliers

If we strictly impose that all instances be off the street and on the right side, this is called **hard margin classification**.

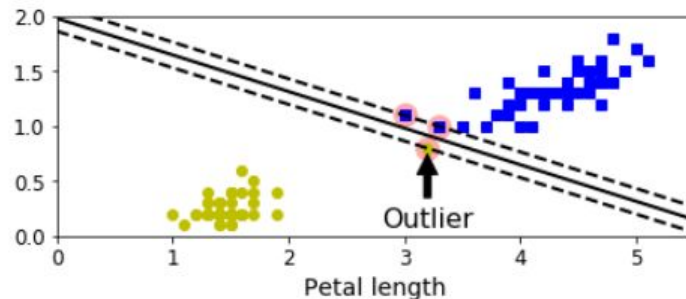
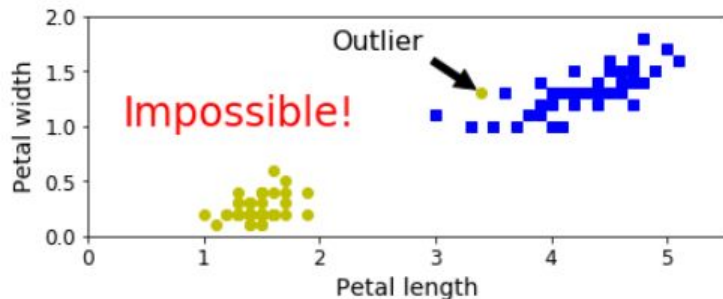


Hard margin classifiers

There are two main issues with hard margin classification.

- 1) First, it only works if the data is **linearly separable**, and
- 2) second it is quite sensitive to outliers. As shown below, the iris dataset with just one additional outlier: on the left, it is impossible to find a hard margin, and it will probably not generalize as well.

Saving figure sensitivity_to_outliers_plot

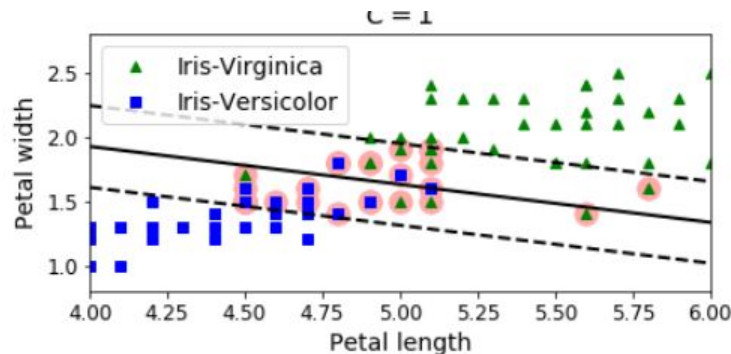


Soft margin classifiers

To avoid these issues it is preferable to use a more flexible model.

The objective is to find a good balance between keeping the street as large as possible and limiting the margin violations (i.e., instances that end up in the middle of the street or even on the wrong side).

This is called **soft margin classification**.



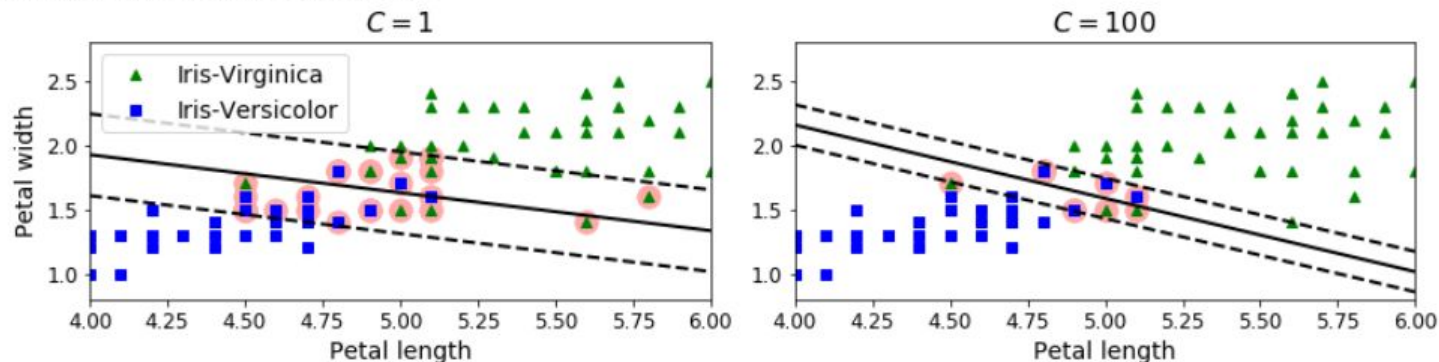
Soft margin classifiers

In Scikit-Learn's SVM classes, you can control this balance using the **C hyperparameter**:

a smaller C value leads to a wider street but more margin violations.

On the left, using a high C value the classifier makes fewer margin violations but ends up with a smaller margin. On the right, using a low C value the margin is much larger, but many instances end up on the street.

Saving figure regularization_plot



Hinge loss

We don't want to penalize out of margin cases

Let the SVM output be $y=wx+b$ (Linear SVC)

The target = $t = +/-1$

If y and t have same sign \Rightarrow correctly classified \Rightarrow 0 loss

If y and t have opposite signs \Rightarrow misclassified \Rightarrow increasing loss as the sample closer to the boundary

We keep the boundaries of support at $+/-1$

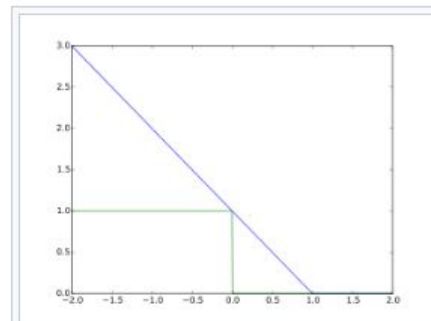
Hinge loss

$t = \pm 1$

t and y same sign $\Rightarrow l = 0$

t and y opposite $\Rightarrow 0 < l < 3$, if $|y| < 1$

$$\ell(y) = \max(0, 1 - t \cdot y)$$



Plot of hinge loss (blue, measured vertically) vs. zero-one loss (measured vertically; misclassification, green: $y < 0$) for $t = 1$ and variable y (measured horizontally). Note that the hinge loss penalizes predictions $y < 1$, corresponding to the notion of a margin in a support vector machine.

Hinge loss

We don't want to penalize out of margin cases

Hinge loss

From Wikipedia, the free encyclopedia

In machine learning, the **hinge loss** is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs).^[1]

For an intended output $t = \pm 1$ and a classifier score y , the hinge loss of the prediction y is defined as

$$\ell(y) = \max(0, 1 - t \cdot y)$$

Note that y should be the "raw" output of the classifier's decision function, not the predicted class label. For instance, in linear SVMs, $y = \mathbf{w} \cdot \mathbf{x} + b$, where (\mathbf{w}, b) are the parameters of the [hyperplane](#) and \mathbf{x} is the input variable(s).

When t and y have the same sign (meaning y predicts the right class) and $|y| \geq 1$, the hinge loss $\ell(y) = 0$. When they have opposite signs, $\ell(y)$ increases linearly with y , and similarly if $|y| < 1$, even if it has the same sign (correct prediction, but not by enough margin).

https://en.wikipedia.org/wiki/Hinge_loss#:~:text=In%20machine%20learning%2C%20the%20hinge,prediction%20y%20is%20defined%20as

Hinge loss

Always set hinge loss in sklearn, not the default!

```
svm_clf1 = LinearSVC(C=1, loss="hinge", random_state=42)
```

Let's code

<https://colab.research.google.com/drive/1eKtsuyw9LX-mQUKSPrd3cgc6lWWbWdC8?usp=sharing>

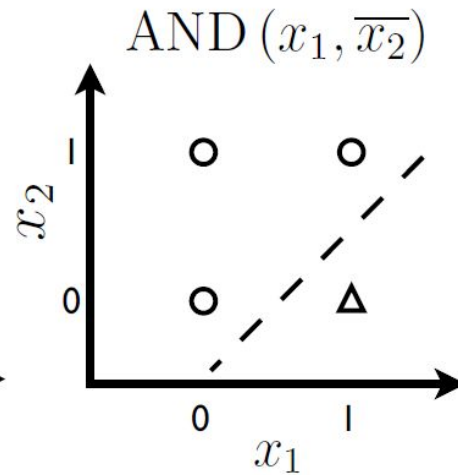
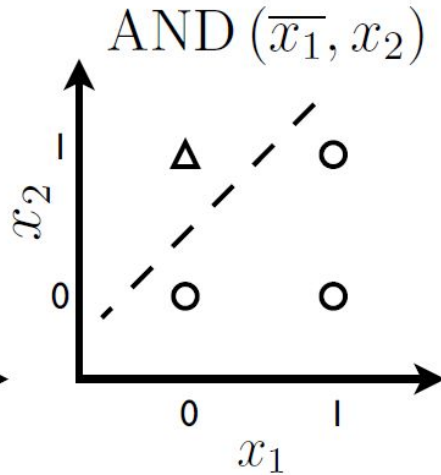
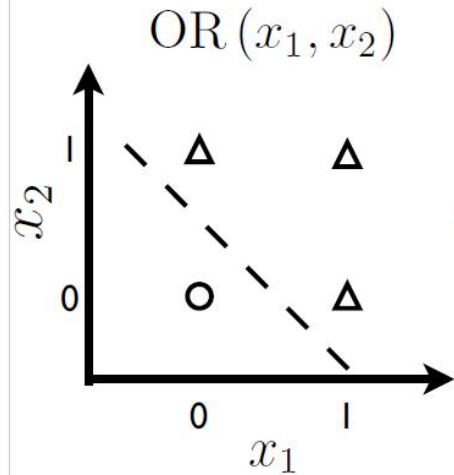
Linear vs Non-linear classifiers

Going Deeper

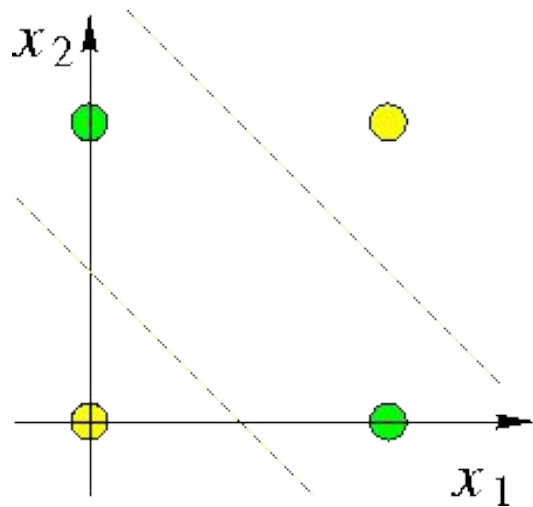
What kind of functions we can approximate with single Neuron?

A	B	out
0	0	0
0	1	1
1	0	1
1	1	1

A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

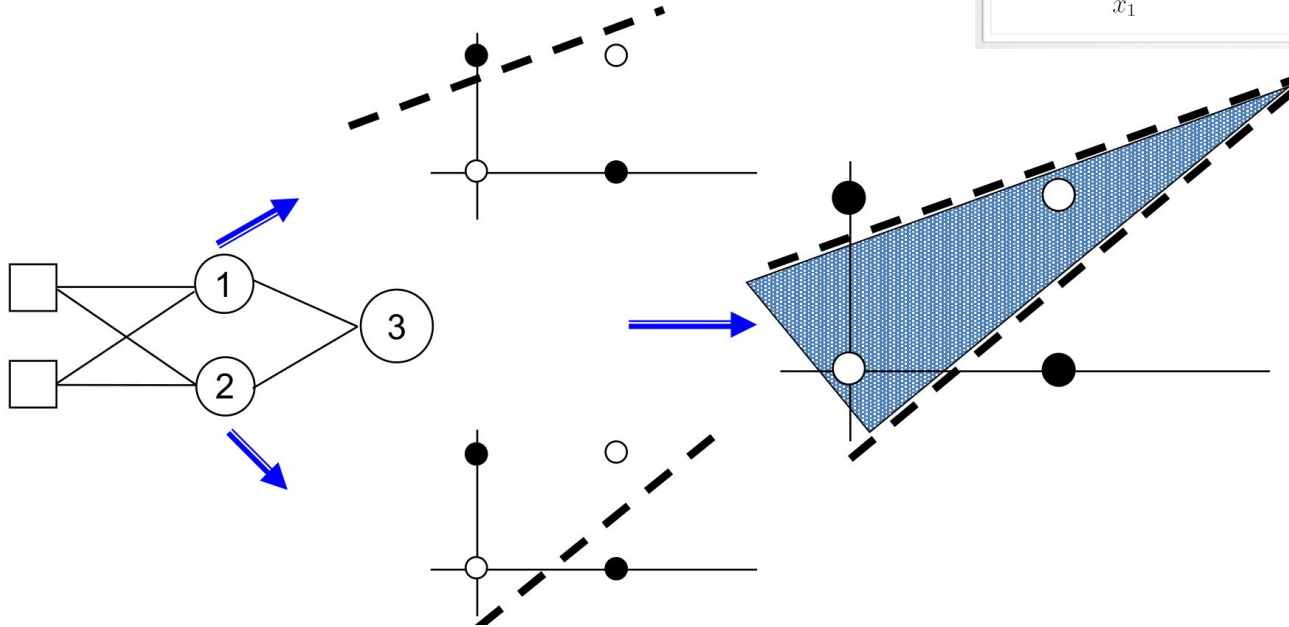
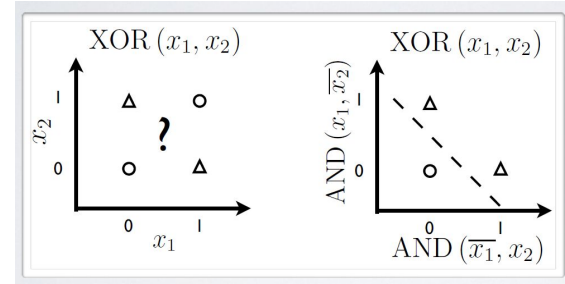


Is single layer enough?



x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Multi layer NN (perceptron)



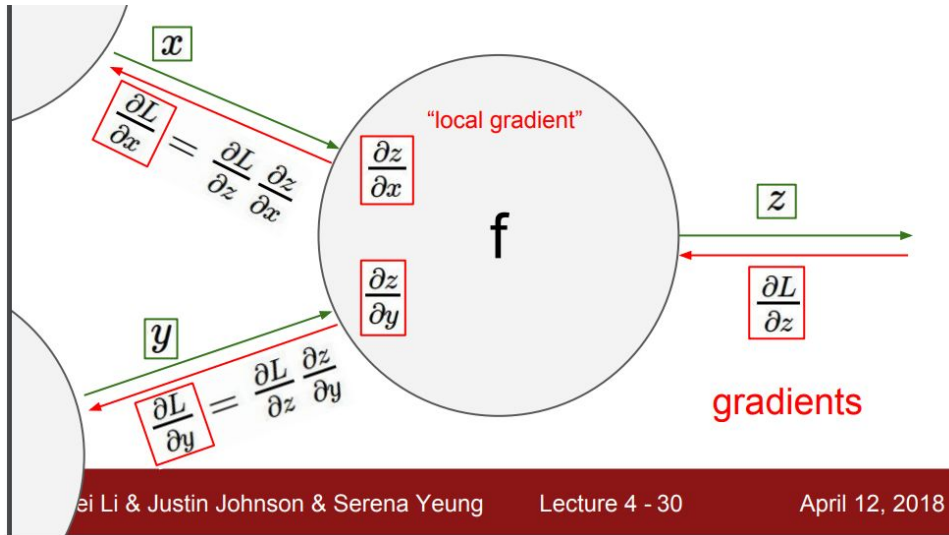
How to get gradients for hidden neurons 1 and 2?

$\frac{\partial L}{\partial z}$ is the error contribution due to z .

Now, what is the distribution of that error on x and y ?

The answer by chain is by weighting the upper error by the next gradients (=sensitivity) to each of x and y .

Continuing this chain, we reach the famous **Backprop algorithm**



Symbolic differentiation

Nowadays, and for years to come, people will implement networks in modern frameworks that are capable of symbolic differentiation, such as TensorFlow. This means that, given a chain of operations with a known derivative, they can compute a gradient function for the chain (by applying the chain rule) that maps network parameter values to gradient values. When you have access to such a function, the backward pass is reduced to a call to this gradient function. Thanks to symbolic differentiation, you'll never have to implement the Backpropagation algorithm by hand. For this reason, we won't waste your time and your focus on deriving the exact formulation of the Backpropagation algorithm in these pages. All you need is a good understanding of how gradient-based optimization works.

Why we need activation functions?

As we have seen above, one neuron creates one line (or plane in n-D).

For one layer of neurons, we need activation functions to *decide* on the output.

But why we need them in the intermediate neurons?

As we saw in the XOR example, we needed 2 lines.

However, if they are just 2 lines, without any decision actions on their outputs, we are still having one linear mapping!

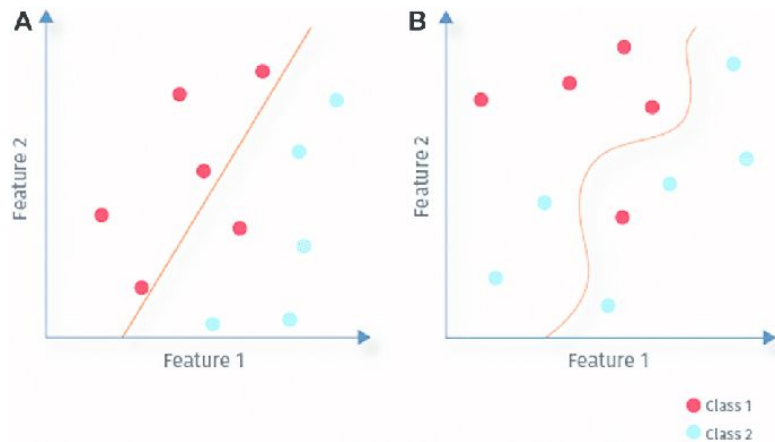
$$y = w_{31}(w_{11}x_1 + w_{12}x_2) + w_{32}(w_{21}x_1 + w_{22}x_2)$$

So we still have one line. But this is not what we wanted!

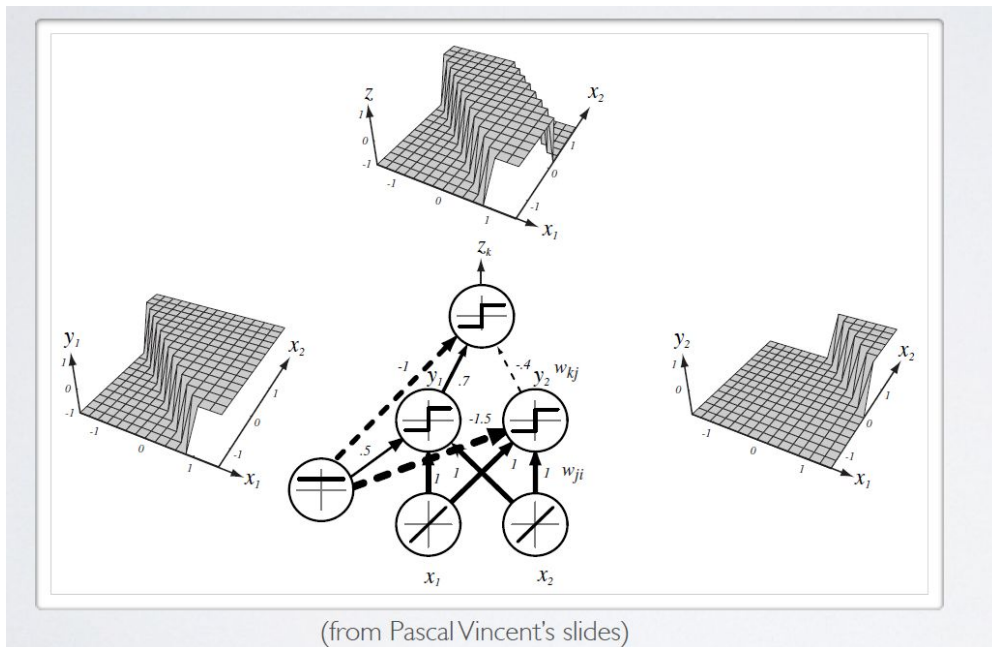
We wanted **three** mapped points in a **new space**

To achieve this mapping, we need an activation function, say sign or sigmoid.

Similarly, if the best boundary we want is **non linear**, we need different activation functions to achieve this non linearity:

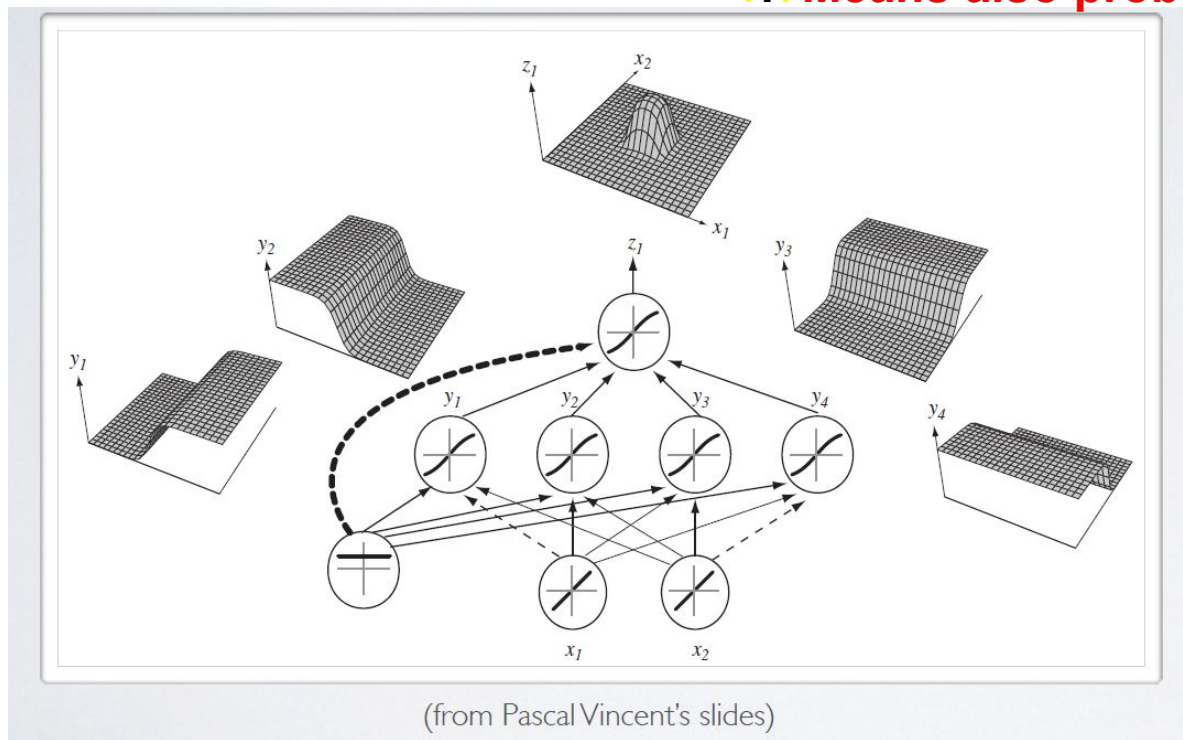


Multiple hidden layers capacity (decision boundaries)



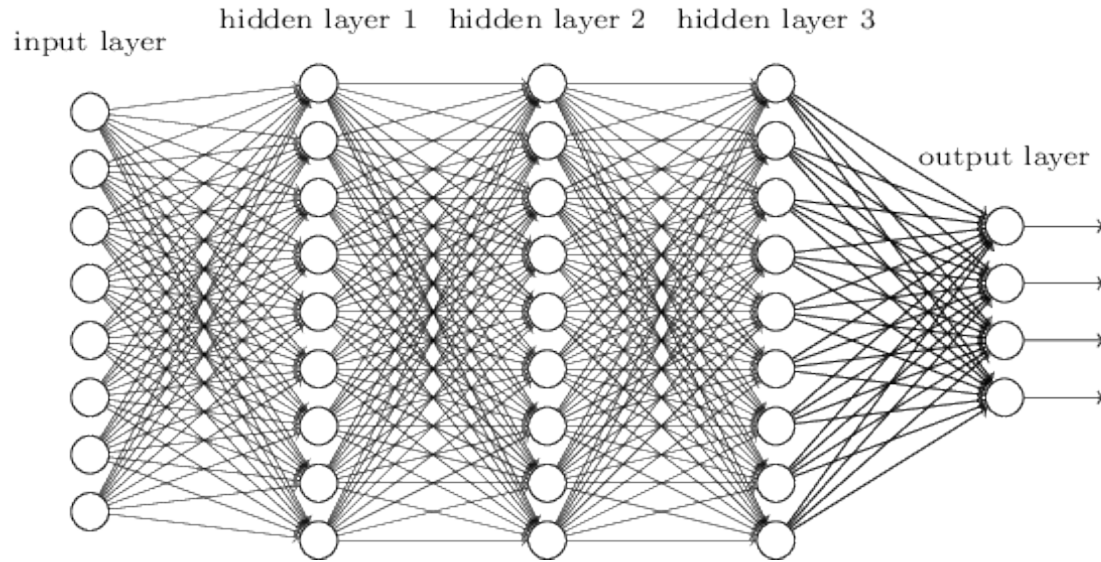
Multiple hidden layers capacity (decision boundaries)

!! Means also probability of overfitting



Going deeper

- Shallow vs. deep models



Let's code

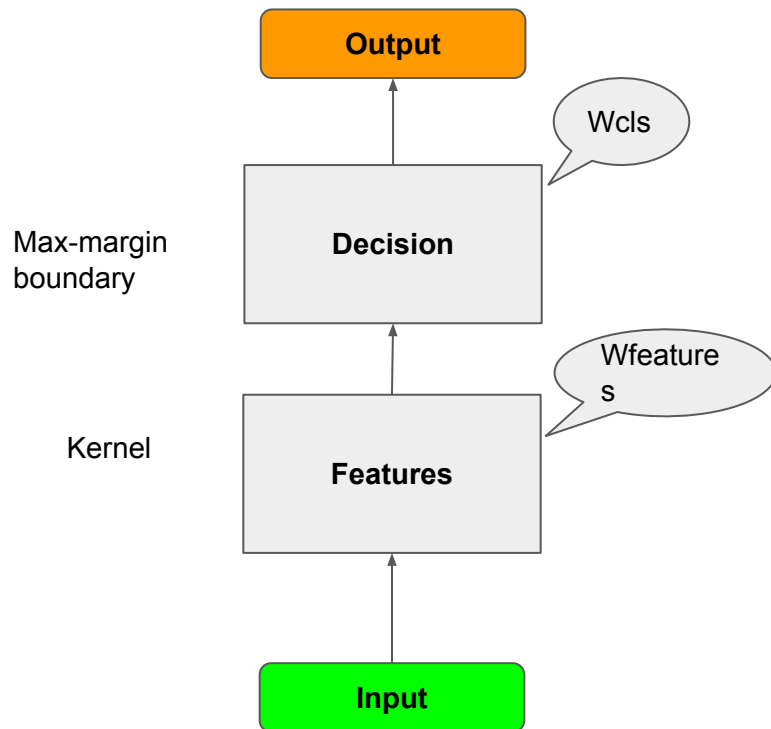
<https://colab.research.google.com/drive/1pn10aiFzYq9cUau0DTkJzMPnYNggtnpl#scrollTo=KtZy7BTth7Or>

Non-linear SVM (Optional)

Kernel SVM

Kernel trick enables the large margin classifier to “project” to a high dimensional space first before doing the max-margin classification \Rightarrow Called **kernel trick**

Roughly, this is equivalent to DL design pattern of features+decision blocks



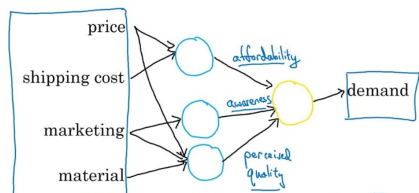
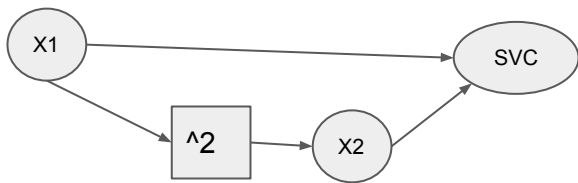
Kernel example

Consider the left plot below: it represents a simple dataset with just one feature X_1 . This dataset is not linearly separable, as you can see.

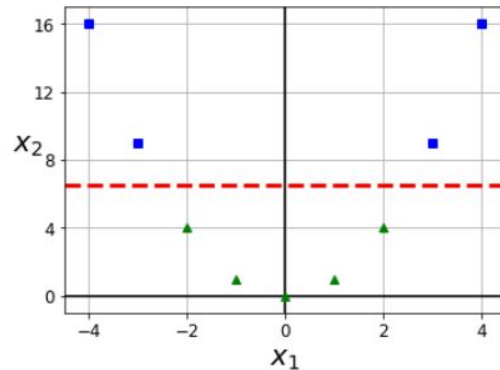
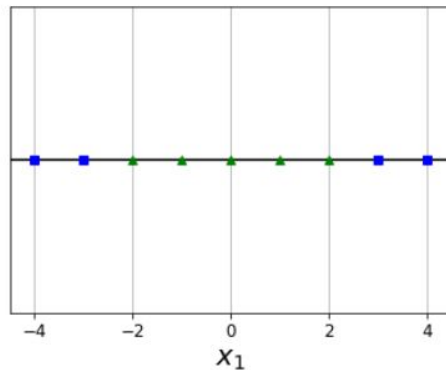
But if you add a second feature $X_2 = X_1^2$, the resulting 2D dataset is perfectly linearly separable.

Higher dimension projection:

As will be described later, this is equivalent to projecting the low dimensional features to another space (usually higher dimension), in which we hope the data is linearly (or easily) separable.



Saving figure higher_dimensions_plot

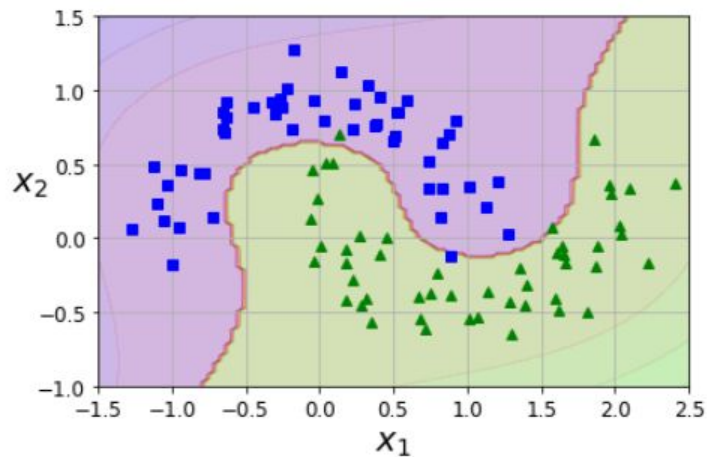
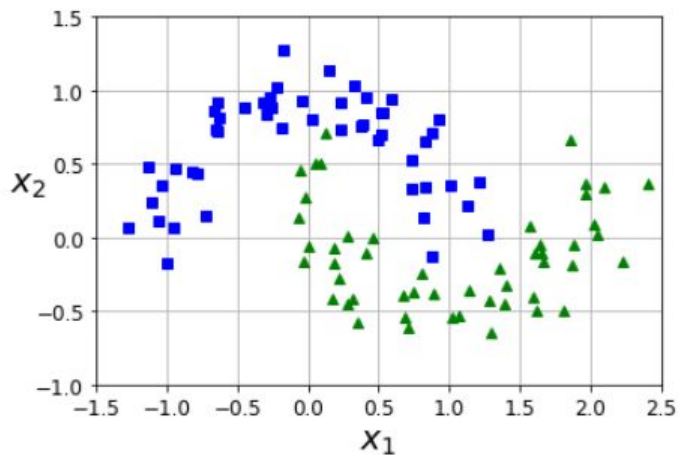


Kernel SVM

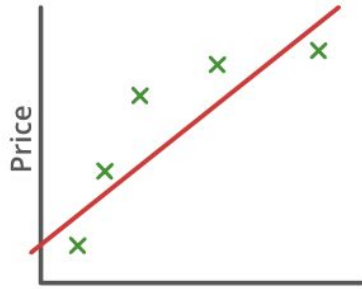
Polynomial

Gaussian = RBF

```
SVC(kernel="rbf", gamma=5, C=0.001)
```

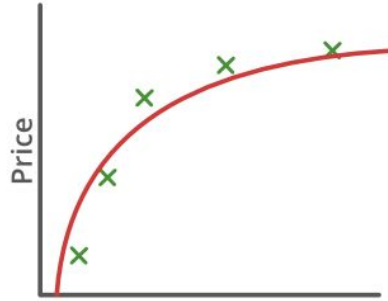


Risk of overfitting



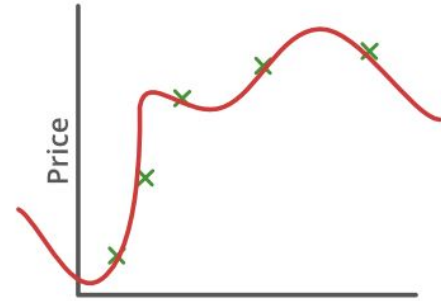
Size
 $\theta_0 + \theta_1 x$

High bias (underfit)



Size
 $\theta_0 + \theta_1 x + \theta_2 x^2$

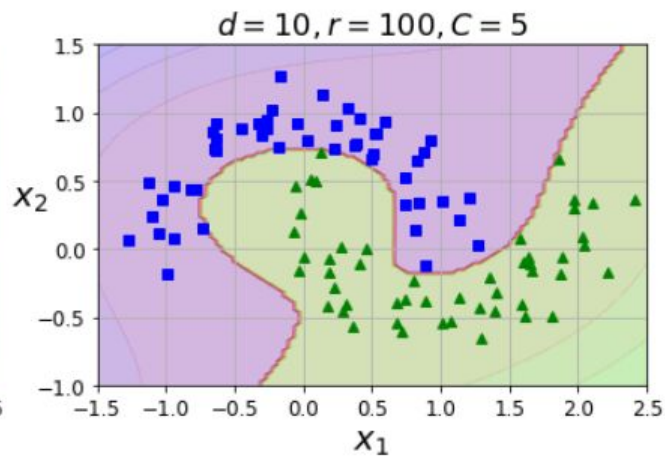
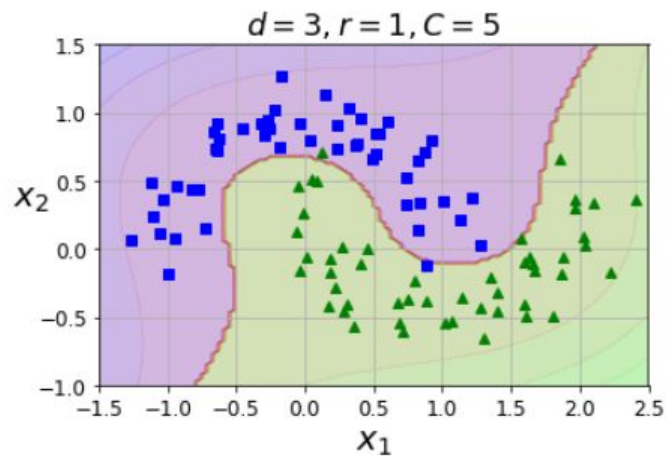
High bias (underfit)



Size
 $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$

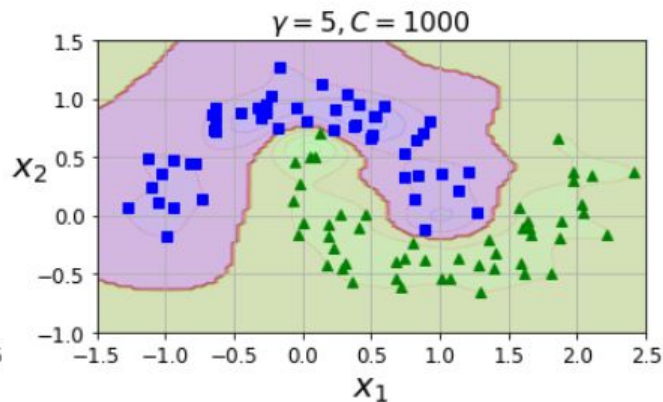
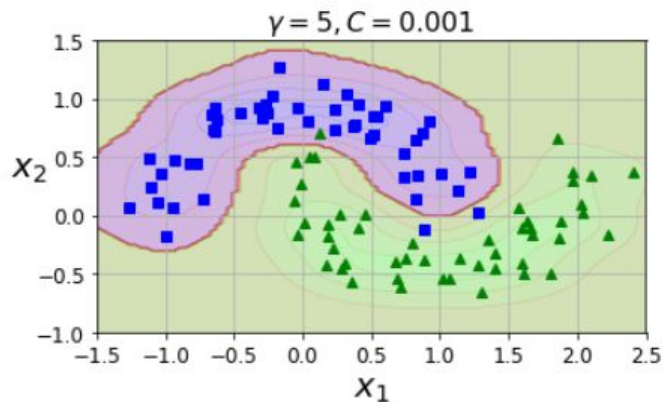
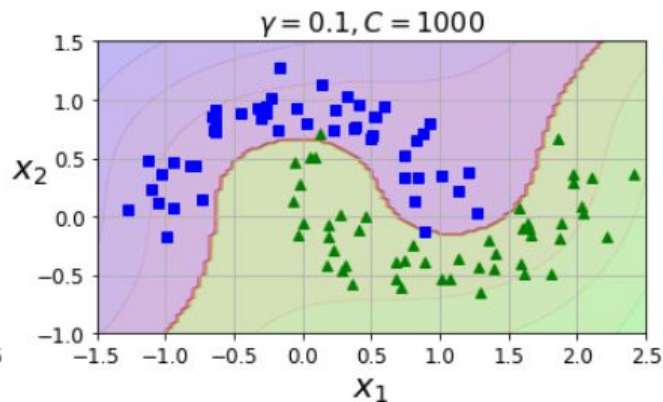
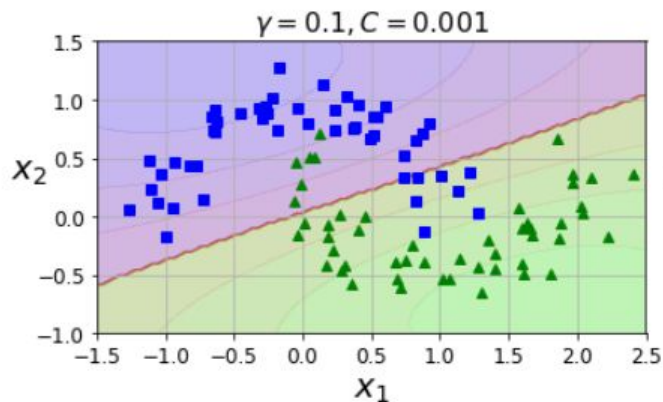
High variance
(overfit)

Risk of overfitting - Polynomial



Risk of overfitting - RBF

$$\phi_{\gamma}(x, l) = \exp(-\gamma(\|x - l\|^2))$$



Let's code

<https://colab.research.google.com/drive/1eKtsuyw9LX-mQUKSPrd3cgc6lWWbWdC8?usp=sharing>