ML recap

Lorenzo Leuzzi

Indice

1 Introduction

We want to investigate on the generalization capability of a Machine Learning model (measured as a risk or test error) thru **Statistical Learning Theory** (*SLT*). So we'll have to understand the role of model complexity and number of data *l*.
The goal is to find an hypotesis h in the hypotesis space H that minimize the *Empirical Risk* (training error E), finding the best values for the model free parameters $\mathbf{w}$. For the loss, we use the square of errors: **Least Mean Square**.

$$R_{emp} = Loss(h_w) = E(h_w) = \frac{1}{l}\sum_{1}^{l}(d_p - h_w(\mathbf{x_p}))^2 \tag{1}$$

Vapnik-Chervonenkis-dimension (VC-dim) is a measure of complexity of the hypothesis (flexibility to fit the data).

$$R \leq R_{emp} + \epsilon(\frac{1}{l}, VC, \frac{1}{\delta}) \tag{2}$$

ϵ (*VC-confidence*) is a function that grows with *VC-dim* and that decreases with l (size of the data set) and *delta* (confidence).

2 Linear Models

Supervised Learning: given a training set $< input, target >$ for an unknown function. Find a good approximation to the function, a hypothesis that can be used to predict on unseen data (generalization).

2.1 Regression

Process of estimating of a real-value function on the basis of finite set of noisy samples.
Given a set of l training examples (x_p, y_p) with $p = 1, ..., l$, we have to find $h(x_p)$ that minimizes the expected loss on the training data.

$$h_w(\mathbf{x}_p) = \mathbf{x}_p^T\mathbf{w} = \sum_{i=0}^{l} x_{p,i}w_i \tag{3}$$

How to solve? Since we know that a local minimum is a stationary point, we have to search $\mathbf{w}$ s.t the directional gradients $\frac{\partial E(w)}{\partial w_i} = 0$.

2.2 Classification

The same models used for regression can be used for classification: categorical targets y or d, for example 0/1, -1/+1, etc. We want to learn w such that we get a good classification accuracy. The decision boundary is $x^Tw = w^Tx = 0$ and we can introduce a threshold function which can be written in many ways.

$$h_w(\mathbf{x}_p) = sign(\sum_{i=0}^{l} x_{p,i}w_i) \tag{4}$$

2.3 Learning Algorithms

We are going to introduce two learning algorithms for the regression and for the classification task using a linear model, both based on LMS.

2.3.1 A direct approach based on normal equation solution

Differentiating E($\mathbf{w}$) with respect to $\mathbf{w}$ to get the normal equation $(\mathbf{X}^T\mathbf{X})\mathbf{w} = \mathbf{X}^T\mathbf{y}$. If $(\mathbf{X}^T\mathbf{X})$ is not singular then the solution is given by $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^+\mathbf{y}$. Else the solutions are infinite, so we can choose the *minnorm(w)* solution. The Singular Value Decomposition can be used for computing the pseudoinverse of a matrix ($\mathbf{X}^+$). With $\mathbf{X}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T$.

2.3.2 An iterative approach based on gradient descent

The derivation suggests an approach based on an iterative algorithm based on

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = -2\sum_{p=1}^{l}(y_p - \mathbf{x}_p^T\mathbf{w})x_{p,i} = -2\sum_{p=1}^{l}\delta_p x_{p,i} \tag{5}$$

The gradient $\Delta\mathbf{w}$ is the ascent direction, we can move toward the minimum.

$$\Delta\mathbf{w} = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial E(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial E(\mathbf{w})}{\partial w_n} \end{bmatrix} = \begin{bmatrix} \Delta w_1 \\ \vdots \\ \Delta w_n \end{bmatrix} \tag{6}$$

The gradient descent algorithm is a **local search**, we begin with a initial weights vector and modify it iteratively to minimize the error function. The movements will be made according to $\mathbf{w}_{new} = \mathbf{w} + \eta \cdot \Delta\mathbf{w}$ where η is the **learning rate**.
For batch version the gradient is the sum over all the l patterns, for the on-line/stochastic version we upgrade the weights with the error that is computed for each pattern.

2.4 Inductive Bias

Language bias: the H is a set of linear functions (may be very restrictive and rigid).
Search bias: ordered search guided by the Least Squares minimization goal.
That brings limitations to the resolvable problems.

2.5 Linear Basis Expansion (LBE)

In an higher dimensional space it's easier that the points are linearly separable. Augmenting the input vector with additional variables which are transformations of x according to a function $\phi_k : \mathbb{R} \to \mathbb{R}$ can help the problem.

$$h_w(\mathbf{x}) = \sum_{k=0}^{K} w_k \phi_k(\mathbf{x}) \tag{7}$$

This method is more expressive but with large basis of functions, increasing the complexity we risk overfitting.

2.6 Tikhonov regularization (Ridge Regression)

Adding a constraint to the sum of value of $|w_i|$ that penalizes models with higher values of the weights. This is the λ regularization hyper-parameter to control the complexity.

$$loss(\mathbf{w}) = \sum_{p=1}^{l} (y_p - \mathbf{x}_p^T \mathbf{w})^2 + \lambda ||\mathbf{w}||^2 \tag{8}$$

$$\mathbf{w}_{new} = \mathbf{w} + \eta \cdot \Delta \mathbf{w} - 2\lambda \mathbf{w} \tag{9}$$

3 K-NearestNeighbor

A natural way to classify a new point is to have a look at its neighbors. It is a lazy, memory based, instance-based, distance-based method. It stores the training data $\langle x_p, y_p \rangle$ and given an input x finds the k **nearest** training examples x_i, then outputs the average.

$$avg_k(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i \qquad h(\mathbf{x}) = \begin{cases} 1 \text{ if } avg_k(\mathbf{x}) > 0.5 \\ 0 \text{ otherwise} \end{cases} \tag{10}$$

3.1 Limitations

K-NN is computationally intensive, for each new input computes the distance with all stored vectors. Can fail when we have a lot of input variables due to the curse of dimensionality, it's hard to find nearby points in high dimensions.

K-NN	**Linear**
Flexible (high var)	Rigid (low var)
Lazy (store data)	Eager (analyze data)
Instance-based	Parametric
Local estimations	Global estimations

4 Neural Networks

A Neural Network (*NN*): can learn from examples; are universal approximators; give flexible approaches for arbitrary functions (including non-linear); can deal with noisy and incomplete data; can handle continuous and discrete data for both regression and classification tasks.

4.1 Artificial Neuron

Receive an input from an external source or an other unit, with weights $\mathbf{w}$ as free parameters and can be modified by the learning process. To the neuron's output is then applied an activation function f (linear, LTU, logistic, etc).

$$\begin{cases} net_i(\mathbf{x}) = \sum_j w_{ij} x_j \\ o(\mathbf{x}) = f(net_i(\mathbf{x})) \end{cases} \tag{11}$$

4.2 Learning for One-unit Models

4.2.1 Adaline

Adaptive linear neuron: LMS direct solution and gradient descent solution. An approach that we will generalize to Multi Layers Perceptron. It convergences asymptotically also for not linear separable problems and can be extended to network of units (NN).

4.2.2 Perceptron Learning Algorithm

Initialize weights, pick learning rate η (between 0 and 1), until stopping condition do. For each training pattern (x, d) compute output activation $out = sign(\mathbf{w}^T\mathbf{x})$

- $if\ out = d$ don't change weights
- $else$ change weights $\mathbf{w}_{new} = \mathbf{w} + \eta d\mathbf{x}$

Perceptron Convergence Theorem: the perceptron is guaranteed to converge (classifying correctly all the input patterns) in a finite number of steps if the problem is linearly separable. Else it doesn't converge and it is difficult to be extended to network of units (NN).

4.3 Least Mean Squared Error with Activation Functions

$$out(\mathbf{x}) = f(\mathbf{x}^T\mathbf{w}) \tag{12}$$

where f is a logistic function. Find $\mathbf{w}$ that minimizes the residual sum of squares.

$$E(\mathbf{w}) = \sum_p (d_p - out(\mathbf{x}_p))^2 = \sum_p (d_p - f(\mathbf{x}_p^T\mathbf{w}))^2 \tag{13}$$

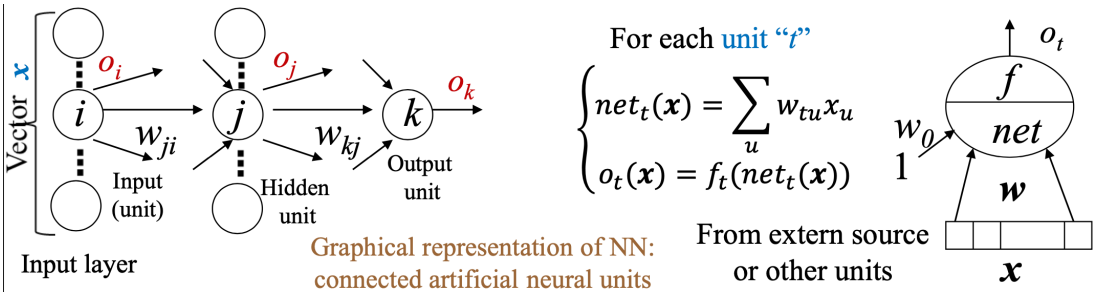
$$\frac{\partial E(w)}{\partial w_i} = -2\sum_{p=1}^{l} x_{p,i}(d_p - f(\mathbf{x}_p^T\mathbf{w}))f'(\mathbf{x}_p^T\mathbf{w}) = -2\sum_{p=1}^{l} x_{p,i}\delta_p \tag{14}$$

$$\delta_p = (d_p - f(\mathbf{x}_p^T\mathbf{w}))f'(\mathbf{x}_p^T\mathbf{w}) = (d_p - out(\mathbf{x}_p))f' \tag{15}$$

$$\mathbf{w}_{new} = \mathbf{w} + \eta\delta_p\mathbf{x}_p \tag{16}$$

4.4 Standard Feedforward Neural Networks

In a MLP (*Multi Layer Perceptron*) architecture units are connected by weighted links and organized in layers.
One hidden layer example:



Graphical representation of NN: connected artificial neural units

$$h(\mathbf{x}) = f_k\Big(\sum_j w_{kj} f_j\big(\sum_j w_{ji} x_i\big)\Big) \tag{17}$$

Neural Networks are **flexible** models where the hypothesis space is a continuous space. Depending on the class of values produced by the output units (discrete or continuous), the model can deal, respectively, with classification tasks (sigmoidal output f) or regression tasks (linear output f). Also multi-class and multi-regression, with multiple output units. The representational capacity of the model is related to the hidden layer, and it's essential that the hidden units are non-linear in order to transform the input pattern into the internal representation of the network. The flexibility is theoretically grounded by the **univeral approximation theorem** (Cybenko 1989): a single hidden-layer network with logistic activation functions can approximate (arbitrary well) every continuous function, provided enough units in the hidden layer.
The **expressive power** of NN is strongly influenced by the number of units and their configuration (architecture). The **learning algorithm** adapts the weights w of the model in order to obtain the best approximation of the target function.

5 Back-propagation Algorithm

The problem is to estimate the contribution of hidden units to the error at the output level. We do this by computing the Generalized Delta Rule.

Algorithm 1 Back-propagation

```
Initialize all the weights w in the network and η.
while E_tot < ε do
    Compute out and E_tot
    for all w ∈ w do
        Δw = −∂E_tot/∂w;
        w_new = w + ηΔw + ... ;
    end for
end while
```

$$\Delta\mathbf{w} = -\frac{\partial E_{tot}}{\partial \mathbf{w}} = -\sum_p \Delta_p \mathbf{w} \tag{18}$$

p: p-th pattern to be fed as input to the neural network. Then, for w_{tu} of a generic unit t, pattern p, and the input u from a generic unit u to the unit t, we have:

$$\Delta_p \mathbf{w}_{tu} = -\frac{\partial E_p}{\partial w_{tu}} = -\frac{\partial E_p}{\partial net_t} \cdot \frac{\partial net_t}{\partial w_{tu}} = \delta_t \cdot o_u \tag{19}$$

where δ_t is the error signal available to the unit t, and o_u is the input to t from a generic unit u, i.e. the input through the connection weighted by w_{tu}.

$$\delta_t = -\frac{\partial E_p}{\partial net_t} = -\frac{\partial E_p}{\partial o_t} \cdot \frac{\partial o_t}{\partial net_t} = -\frac{\partial E_p}{\partial o_t} \cdot f_t'(net_t) \tag{20}$$

The error signal is defined depending of which layer as follows.
output unit: $\delta_k = (d_k - o_k) \cdot f_k'(net_k)$
hidden unit: $\delta_j = (\sum_{k=1}^{K} \delta_k w_{kj}) \cdot f_k'(net_k)$

6 Issues in Training Neural Networks

A good interpretation is to see the back-propagation as a path trough the loss/weight space. The path depends on: data, neural network model, starting point (initial weight values), rate of convergence, final point (stopping rule). This defines a control for the search over the hypothesis space. In particular is import to pay attention to these hyper-parameters: starting values, on-line/batch, learning rate, numer of hidden unit. And other aspects as multiple minima, stopping criteria, overfitting and regularization.

6.1 Cascade Correlation

Start with a minimum networks, add units if they are needed until the error is low. The *CC* algorithm learns both the network weights and network topology (number of units).
Specifically, the algorithm works interleaving the minimization of the total error function (LMS), e.g. by a simple backpropagation training of the output layer, and the maximization of the (non-normalized) correlation, i.e. the covariance, of the new inserted hidden (candidate) unit with the residual error.

7 Validation

We are looking for the best solution, with minimal test error: looking for a balance between fitting (accuracy on training data) and model complexity. The training error is not a good estimate of the test error. So, in practice, is better to approximate the estimation on the data set by resampling, a direct estimation of error via cross-validation (hold-out, K-fold, etc).
Validation has two aims:

- **Model selection**: estimating the performance of different learning models in order to choose the best one (to generalize)
- **Model assessment**: having chosen a final model (or a class of models), estimating/evaluating its prediction error/ risk (generalization error) on new test data (measure of the quality of the ultimately chosen model). It returns an estimation value

Keep separation between goals and use separate datasets: TR for training, VL for model selection, TS for model assessment.

7.1 Grid Search

It is a technique for model selection: hyperparameters can be set by searching in a hyper-parameters space. Try every hyperparameter-value combination and record the result. The best result will corresponds to the best hyperparameter values. The cost can be high so a first coarse grid search can be performed to find good intervals of values, then a finer grid-search over smaller intervals.

7.2 Hold-out

Split the data in 3 disjoint sets: TR, VL, TS (e.g. 50% TR, 25% VL, 25%).

7.3 K-fold Cross Validation

Split the data set D into K mutually exclusive subsets $D_1, ..., D_k$, we train on $D - D_i$ and test it on D_i. This process does NOT give us a unique model, but a variance (std. dev.) over different folds for the class of models or algorithm.

K-fold Double Cross Validation

After dividing the data set into K mutually exclusive subsets $D_1, ..., D_k$, for each D_i: do another internal cross validation using the other $K - 1$ subsets. It doesn't provide a model but an estimation of the risk, because can provide a different model per external step cycle.

8 Statistical Learning Theory

We could approximate the estimation of validation set error (for model selection) or test set error (for model assessment) using analytical concepts like Structural Risk Minimization and VC-dimension.

8.1 VC-dimension

A measure of complexity (capacity/expressive power/flexibility) of a class of hypothesis.

Shattering

Given X input space, n size of the instance space (number of instances), l number of data available and H the hypothesis space. The dichotomies are the partitions or labeling of the N points in X.
H shatters X $\Leftrightarrow$ H can represent all the possible dichotomies on X (0 errors). The points in X can be separated by an $h \in H$ in all the possible ways, and for every possible dichotomy of X there exists a consistent hypothesis $h \in H$.

Definition

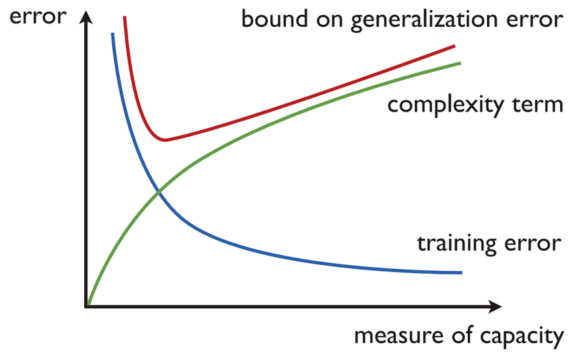
The VC-dim of a class of functions H is the maximum cardinality of a set (configuration) of points in X that can be shattered by H.
In general the VC dimension of a class of linear (separator/decision) hyperplanes (LTU) in a n-dimensional space is $n + 1$.

$$R \leq R_{emp} + \underbrace{\epsilon(\frac{1}{l}, VC, \frac{1}{\delta})}_{\text{VC-confidence}} \tag{21}$$

If we have large l (number of data) then we have low VC-confidence and the bound is close to R. Too simple models (low VC-dim) can not be not sufficient due to high R_{emp} (*underfitting*). If the model is too complex, so high VC-dim, the R_{emp} is low but the VC-confidence and hence R may increase (*overfitting*).

8.2 Structural Risk Minimization

SRM uses the VC-dim as a controlling parameter for minimizing the generalization bound on R (find a trade-off on the bound).

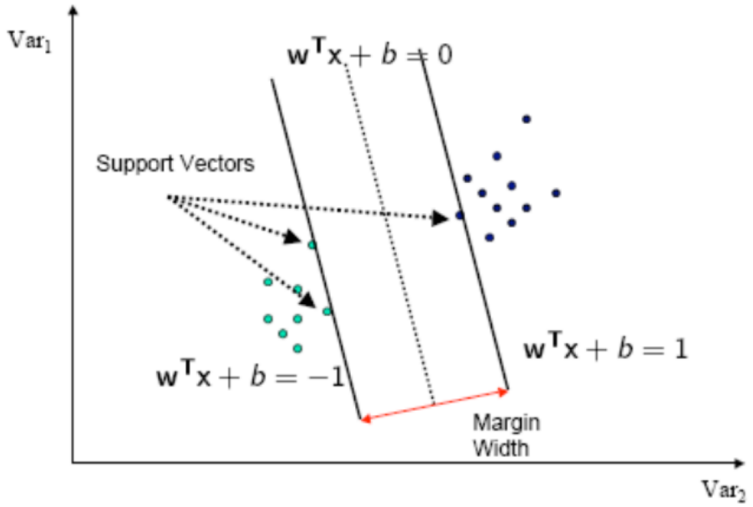


9 Support Vector Machines

9.1 Suppport Vector Machines for Binary Classification

It is a linear machine (LTU, perceptron, ...), with maximization of the separation margin and structural risk minimization.
Given the training set $T = (\mathbf{x}_i, d_i)_{i=1}^N$ we want to find an hyperplane of equation $\mathbf{w}^T\mathbf{x} + b = 0$ to separate the examples so the hypothesis $h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + b)$.
The **separation margin** (p) is evaluated as the double of the distance between the linear hyperplane and the closest data point (a "safe zone"). The optimal hyperplane is the hyperplane which maximizes $p = \frac{2}{||\mathbf{w}||}$. So **maximize** $p \Leftrightarrow$ **minimize** $||\mathbf{w}||$.
The **support vectors** are the closest data points to the hyperplane and "lay on the margin". A support vector $\mathbf{x}^{(s)}$ satisfies the previous equation exactly:

$$d^{(s)}(\mathbf{w}^T\mathbf{x}^{(s)} + b) = 1 \tag{22}$$



With the discriminant function $g(x) = \mathbf{w}^T\mathbf{x} + b$, recalling that $\mathbf{w}_O$ is a vector orthogonal to the hyperplane. Let's denote with r the **distance** between $\mathbf{x}$ and the optimal hyperplane.

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}_o}{||\mathbf{w}_o||} \qquad\qquad r = \frac{g(\mathbf{x})}{||\mathbf{w}_o||} \tag{23}$$

9.2 Hard Margin SVM

For the hard margin SVM, the quadratic optimization problem asks to find the optimum values of $\mathbf{w}$ and b in order to maximize the margin.

$$\Psi(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{24}$$

satisfying the constraints

$$d_i(\mathbf{w}^T\mathbf{x} + b) \geq 1 \ \forall i \text{ (zero classification error)} \tag{25}$$

The objective function $\Psi(\mathbf{w})$ is quadratic and convex in $\mathbf{w}$, while the constraints are linear in $\mathbf{w}$: solving this problem scales (the computational cost) with the size of the input space $\mathbf{w}$.

Solving the Problem

To solve this problem the Lagrangian multipliers method is used and we will find:

$$\mathbf{w}_o = \sum_{i=0}^{N} \alpha_{o,i} d_i \mathbf{x}_i \tag{26}$$

where $\alpha_i \geq 0$ are the Lagrangian multipliers. We also find $b_O = 1 - \mathbf{w}_O^T\mathbf{x}^{(s)}$ corresponding to a positive support vector $\mathbf{x}^{(s)}$. Thus the optimal hyperplane is expressed as:

$$\mathbf{w_o}^T\mathbf{x} + b_o = 0 \Longleftrightarrow \sum_{i=0}^{N} \alpha_{o,i} d_i \underbrace{\mathbf{x}_i^T\mathbf{x}}_{\text{dot product}} + b_o = 0 \tag{27}$$

Kuhn-Tucker Conditions: we can restrict the computation to N_s (the set of SVs): $\mathbf{w}_o = \sum_{i=0}^{N_s} \alpha_{o,i} d_i \mathbf{x}_i$. **The hyperplane depends only from the support vectors**.

How to Use it

We solve the optimization problem and obtain the Lagrangian multipliers α_i. Given the input patter $\mathbf{x}$:

1. computer $g(\mathbf{x}) = \sum_{i=0}^{N} \alpha_{o,i} d_i \mathbf{x}_i^T\mathbf{x} + b_o$
2. classify $\mathbf{x}$ as $h(\mathbf{x}) = sign(g(\mathbf{x}))$

Note that it's not necessary to compute $\mathbf{w}_o$, also the sum can be restricted to the number of support vector N_s

Why this Improves Generalization?

We fixed the training error on linearly separable problems. Minimizing the norm of $\mathbf{w}$ is equivalent to minimizing the VC-dim and thus to minimizing the VC confidence. **Vapnik Theorem**: Let D be the diameter of the smallest ball around the data points. For the class of separating hyperplanes described by the equation $\mathbf{w}^T\mathbf{x} + b = 0$, the upper bound to the VC-dim is:

$$VC \leq min(\lceil \frac{D^2}{p^2} \rceil, m_o) + 1 \tag{28}$$

since $\frac{D^2}{p^2} = Radios^2 ||\mathbf{w}||^2$, VC can be less than m_o (dim. of the input) +1 for general hyperplanes by constrainig the search to the "regularized" one with maximum margin thus minimum $\mathbf{w}$.

9.3 Soft Margin SVM

For noisy or not linearly separable data, you can have a soft margin, support vectors are still on the border, but some data may fall closer to the hyperplane. We introduce a non-negative variable $\xi_i \geq 0$ called **slack variable**. Then a support vector satisfy:

$$d_i(\mathbf{w}^T\mathbf{x} + b) = 1 - \xi_i \tag{29}$$

Note that Vapnik does not hold anymore.
The primal problems becomes find the optimum values of $\mathbf{w}$ and b which minimizes:

$$\Psi(\mathbf{w}, \xi_i) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \tag{30}$$

We introduced C as a **regularization hyperparameter**: low C means many TR errors allowed (possible underfitting), while high C means less or no TR errors allowed (smaller margin, possible overfitting).
Kuhn-Tucker Conditions:

- $0 < \alpha_i < C \rightarrow \xi_i = 0$ (on the margin)
- $\alpha_i = C \rightarrow \xi_i \geq 0$ (inside the margin)

Solving the Problem

Solve the dual problem w.r.t to α_i. Find:

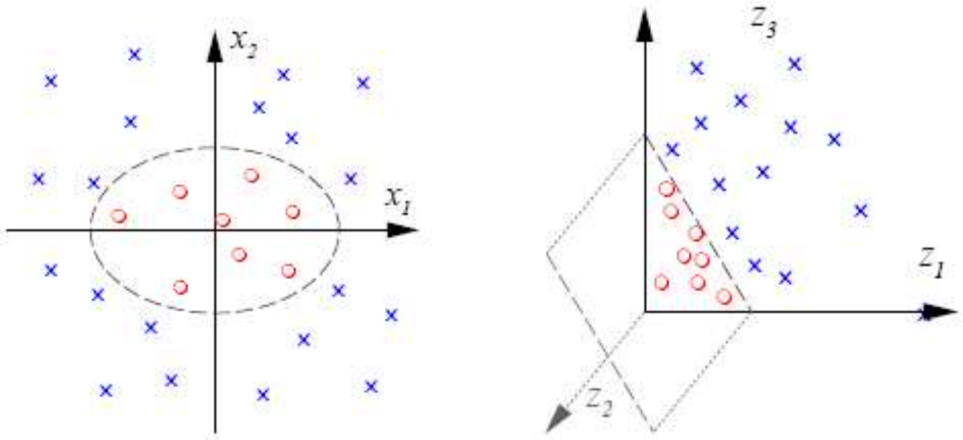
$$\mathbf{w}_o = \sum_{i=0}^{N}\alpha_{o,i}d_i\mathbf{x}_i \tag{31}$$

$$b_o = d_j - \sum_{i=0}^{N}\alpha_{o,i}d_i\mathbf{x}_i^T\mathbf{x}_j \text{ for a pattern } j \text{ s.t. } 0 < \alpha_i < C \tag{32}$$

The non-zero Lagrangian multipliers correspond the the support vectors. We use it as before.

9.4 Mapping to a Higher-dimensional Space

The idea is to map the data points to a high-dimensional feature space where they can possible be linearly separable.



Kernel Approach

Approach to implicitly manage the feature space while regularizing where $\mathbf{x} \mapsto \Phi(\mathbf{x})$. The problem is formulated as before but the hyperplane is now:

$$\mathbf{w}^T\Phi(\mathbf{x}) + b = 0 \Longleftrightarrow \sum_{i=1}^{N} \alpha_i d_i \underbrace{\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x})}_{\text{dot product}} = 0 \tag{33}$$

and the weight vector (incorporates the bias):

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \Phi(\mathbf{x}_i) \tag{34}$$

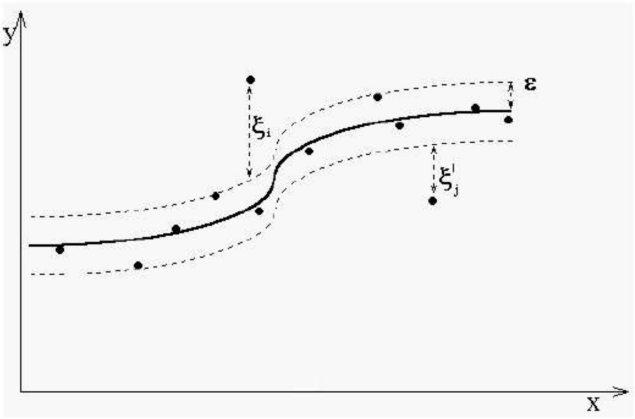
We do not even need to know the feature space itself. This is possible using a function to compute directly the dot products in the feature spaces: k, called **inner product kernel function**: $k(\mathbf{x}_i, \mathbf{x}) = \Phi(\mathbf{x}_i)\Phi(\mathbf{x})$ (symmetric function). We can arrange the dot products in the feature space between the images of the input training patterns in a N×N matrix called kernel matrix $K = \{k(\mathbf{x}_i, \mathbf{x}_j)\}_{(i,j)=1}^{N}$. Not any kernel function computes the inner product in a feature space only if $K \succeq 0$ (positive semidefinite).

9.5 SVM for Non-linear Regression

We estimate d using a linear expansion of non-linear functions $y = h(\mathbf{x}) = \mathbf{w}^T\Phi(\mathbf{x})$. **$\epsilon$-insensitive loss function**:

$$L_\epsilon(d, y) = \begin{cases} |d - y| - \epsilon \text{ if } |d - y| \geq \epsilon \\ 0 \text{ otherwise} \end{cases} \tag{35}$$

We introduce the non-negative slack variables ξ_i' and ξ_i



Solving the Problem

The dual problem yields the optimal α_i' and α_i. With those, we can compute the optimal $\mathbf{w}$:

$$\mathbf{w} = \sum_{i=1}^{N} \underbrace{(\alpha_i' - \alpha_i)}_{\gamma_i} \Phi(\mathbf{x}_i) \tag{36}$$

In this case support vectors correspond to non-zero values of γ_i. The estimate is defined as

$$h(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i k(\mathbf{x}_i, \mathbf{x}) \tag{37}$$

Select values for the user-specified parameters C and ϵ. Choose an inner product kernel function k. Compute the kernel matrix K. Solve the dual form and get the optimal values of the Lagrangian multipliers.

9.6 SVM other information

+: The regularization is embedded in the optimization problem (margin). Approximation of the theoretical structure risk minimization. Convex problem (training always finds a global minimum). Implicit feature transformation using kernels.
−: Must choose the kernel and its parameters. It's a batch algorithm. Very large problems are computationally intractable.
In practice there is no theory which guarantees that a given family of SVMs will have high accuracy on a given problem. Rigorous **selection** of the **kernel** and the C requires an estimation of the complexity (VC-dim).

10 Bias and Variance

A training set is only one possible realization from the universe of the data: different sets can provide different estimates. The expected error (on various training) for a point $\mathbf{x}$ is decomposed as:

- Bias, quantifies the discrepancy between true function and $h(\mathbf{x})$, with $h(\mathbf{x})$ averaged over different data (systematic error);
- Variance, quantifies the variability of the response of the model h for different realizations of the TR data;
- Noise, error in the labels, even the optimal solution can be wrong.

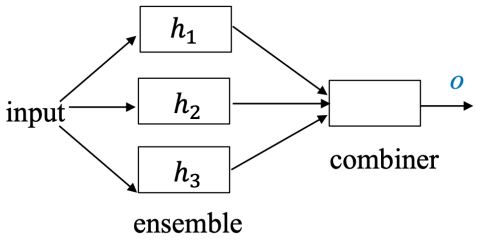
An over-regularized model (large λ) will have a high bias, while an under-regularized model (small λ) will have a high variance.

10.1 Bias, Variance Decomposition

$$\begin{aligned} E_P[(y - h(\mathbf{x}))^2] \quad = \quad & E_P[(h(\mathbf{x}) - E_P[h(\mathbf{x})])^2] && \text{variance} \\ & +(E_P[h(\mathbf{x})] - f(\mathbf{x}))^2 && \text{bias}^2 \\ & +E_P[(y - f(\mathbf{x}))^2] && \text{noise}^2 \end{aligned}$$

10.2 Ensemble Learning

Take advantage of multiple models. In regression, simple average committee, mean square error of a committee. In classification, take a vote over many classifier.



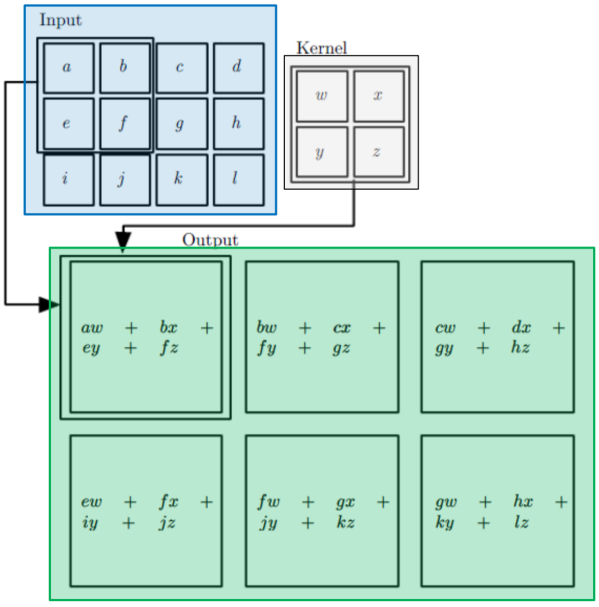
Bagging: Bootstrap Aggregating, combine many classifiers. Train n classifiers on different subsets of TR and differentiate each training using bootstrap (resampling with replacement)
Boosting: differentiate each training concentrating on errors, combine results by output weights.

11 Convolutional Neural Network

CNN are a specialized type of artificial neural networks that use a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers. They are specifically designed to process pixel data and are used in image recognition and processing. CNN exploits weight sharing to set-up a shifting window of the units over a segment of the input signal.

11.1 2D Convolution

Using a $n * n$ kernel (**local receptive field**), we traverse the image with a number of pixels shifts (**stride**) to produce the **feature map**.

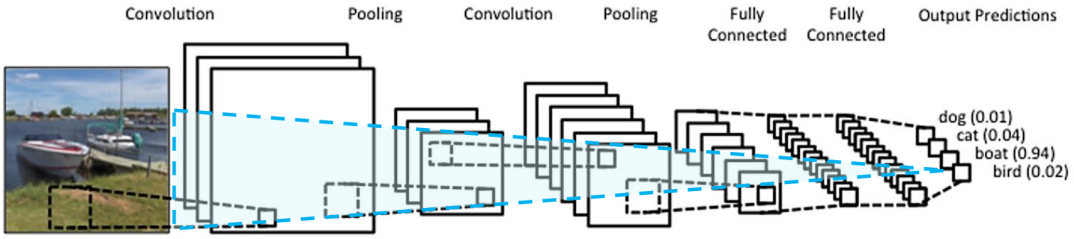


The units' weights are a filter, trained to detect some features in the image: with the architecture constraints of having local and small receptive field (kernel), learned "filters" produce the strongest response to a spatially local input pattern. The same unit/filter is applied across the entire image allowing for features to be detected regardless of their position in the visual field, thus constituting the property of translation invariance.

11.2 Pooling

We can reduce the feature map by subsampling (stride > 1), or by operating a mean, or by taking the max over the neighbors of each input.
Pooling also further helps to make the representation become approximately invariant to small translations of the input.

11.3 Overview



CNNs detect **local patterns** in images and they're invariant to translations. **Weight sharing** reduces the number of free-parameters. Pooling helps reducing the dimension of the representation to small shifts and distortions.
Historical instance of deep neural network, progressively abstracting features from images.
Training is typically made by back-propagation, with the many heuristics that we studied and some specializations for weight sharing, pooling etc. Given the typical usage of huge networks and large amount of data, many hyper-parameters are fixed by experience and expert suggestion.

12 Deep Learning

Deep Learning (*DL*) allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of **abstraction**. Starting from 2010, deep learning architectures are no longer unfeasible, thanks to the availability of large datasets, powerful computing systems, etc.

12.1 General View

Deep models are built by a MLP with multiple layers of nonlinear processing units and using different new techniques (as pre-training). The learning phase can be supervised or unsupervised. The **feature extraction** is **hierarchical**, a series of hidden layers extracts increasingly abstract features from the image. The abstract features can be "reused" at the higher levels, combining to generalize to unseen during training. A series of hidden layers extracts increasingly abstract features from the image. DL NNs break the complicated mapping into a serie of nested simple ones.

12.2 A Many Layers Architecture

The general idea (**no-flattening**) is that when a function can be compactly represented by a deep architecture, in might need a very large architecture to be represented by a shallow one. Deep models can exploit the compositionality of internal representation of feature, learn every pattern but not the combination of them.

So, deep networks can be seen as **compact models** with respect to potentially larger shallow models for the task. Hence more efficient, not just from the computational point of view but from the learning point of view too: **less units** and **less weights** help learning on complex tasks to achieve good generalization.
Choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions. If our task matches this **inductive bias** of course the deep shape of the learner is suitable.
Deeper network often need less units, so less parameters, so less training data. But many layers are harder to optimize during learning (exploding/vanishing gradients).

12.3 Representational Learning

Deep learning methods are representation-learning methods with multiple levels of representation.
How to obtain hidden representation? The representation can be learnt for the unlabeled data and then use it to solve supervised tasks (**pretraining**).
How to exploit the hidden representation? The learned representation can be used for other tasks (**transfer learning**).

Pre-training

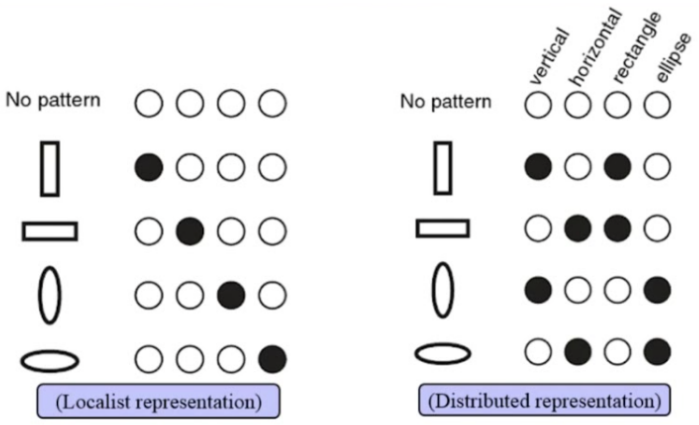
The first approach to pre-training is called Greedy Layer-Wise Unsupervised Pre-training where each layer is optimized independently in an unsupervised way, constituting a pre-training for the final fine-tuning of the network. So the pre-training captures the shape of the input distribution. It is good for **initialization** and **regularization**. Despite many initial successful cases, the general role of pre-training is nowadays under critical revision by researchers since it is difficult to be managed.
An **autoencoder** is a neural network that is trained to attempt to copy its input to its output. It has two part: encoder ($h = f(x)$) and decoder that produces a reconstruction ($r = g(h)$). They can be useful for unsupervised learning and pre-training: train the layers as an auto-associator to minimize the reconstruction error.

Transfer Learning

Using the representation discover in a model to improve another one. Like for example AlexNet which is a model that has learned rich feature representations for a wide range of images and now used for different tasks.

12.4 Distributed representation

In a distributed representation their elements (the features) are not mutually exclusive and their many configurations correspond to the variations seen in the observed data. It is a richer and smooth representation: **shared attributes** allows generalization among different concepts. The distance between two concepts reflects the meaning, and their similarity (similar concept have similar values). Distributed representation can represent n features with k values to describe k^n different concepts.
A drawback of the many advantages of distributed representation is that is less easy to interpret.



DL models learn a distributed representation of the data by disentangling shared features that generates the data through different level of abstraction.

12.5 Techniques

Originally pre-training techniques were used but nowadays the followings: stochastic gradient descent with momentum, ReLU activation functions, cross-entropy loss with soft-max for output layers, regularization, etc.

Gradient Issue

The magnitude of the gradients, as its back-propagated through many layers, suffers from the issues of **vanishing** or **exploding gradient**. Some approaches include the gradient clipping: if $||\mathbf{g}|| > v$ then $\mathbf{g} = \frac{v\mathbf{g}}{||\mathbf{g}||}$ with v the norm threshold. A new approach is to use the **ReLU** (Rectified Linear Unit) activation function: $f(x) = max(0, x)$. It is efficient for gradient propagation, makes computation and training faster for DL networks but it's NOT differentiable at 0.

Dropout

Method that randomly selects and "ignores" a subset of the network units during its training. Bagging is the combination of many classifiers/models, dropout aims to approximate this process, with an exponentially large number of sub neural networks, also aiming at maximizing the diversity of the ensemble preventing perform well on average. There's parameter sharing among sub networks and it regularizes each unit to be not merely a good feature but a feature that is good in many contexts (the different sub-networks).

13 Random Weights Neural Network

A randomized approach employs a degree of randomness as part of its constructive strategy. It can enhance predictive performance and alleviate difficulties of classical ML methodologies.

13.1 Overview

A network containing randomly connected hidden layers, weights are fixed after random initialization, train only the output weights.

13.2 Structure

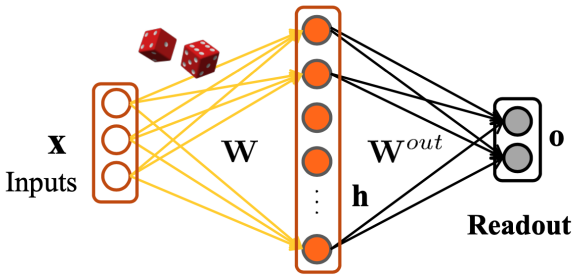
Input $\rightarrow$ **Untrained hidden layer**: non-linearly embeds the input into a high-dimensional feature space, where the problem is more likely to be solved linearly (LBE). Theoretically grounded on the Cover's Theorem. $\rightarrow$ **Feature representation** Φ $\rightarrow$ **Trained readout layer**: combines the features in the hidden space for output computation. Typically implemented by linear models. $\rightarrow$ **Output**

13.3 Feed-forward Randomized NNs

Input-output relation is implemented through an untrained hidden layer, which implements a randomized basis expansion (Φ).

$$o = \mathbf{W}^{out} f(\mathbf{W}\mathbf{x}) \tag{38}$$

$$\mathbf{W}^{out} = argmin_{\mathbf{W}} ||\mathbf{W}\mathbf{h} - y||^2 \tag{39}$$



14 Unsupervised Learning

The training set is a set of unlabeled data $< x >$.

14.1 Clustering

Partition of data into clusters, subsets of "similar" data. There's different clustering algorithms, but we go directly to a simple, historical approach.

Vector Quantization

A data vector $x \in V$ is described by the best matching or "winning" reference vector $\mathbf{w}_{i*(\mathbf{x})}$ of $\mathbf{w}$ which the **distortion error** $d(x, \mathbf{w}_{i*(\mathbf{x})})$ is minimal. This procedure divides the manifold V into a number of subregions.

Clustering and Vector Quantization

The goal is to find the optimal partitioning of unknown data distribution in x-space into region (cluster) approximated by a cluster center or prototype ($c(\mathbf{x})$ or $\mathbf{w}_{i*(\mathbf{x})}$).
Loss function (discrete version):

$$E = \sum_{i}^{l} \sum_{j}^{K} ||\mathbf{x}_i \mathbf{w}_j||^2 \delta_{winner}(i, j) \tag{40}$$

where δ is the characteristic function of the receptive field of $\mathbf{w}_j$.
The set of reference vectors $\mathbf{w}$ that minimizes E is the solution of the VQ problem.

14.2 Online K-means

The online version of K-means for any x_i is:

$$\delta \mathbf{w}_{i*} = \eta \delta_{winner}(i, i*)(\mathbf{x}_i - \mathbf{w}_i) \tag{41}$$

where η is the learning rate. The basic algorithm is:

1. Choose k cluster centers to coincide with k randomly chosen patterns or k randomly defined points inside the hypervolume containing the pattern set.
2. Assign each pattern to the closest cluster center (the winner).
   $i^*(\mathbf{x}) = argmin_i ||\mathbf{x} - \mathbf{w}_i||^2$ ($\mathbf{x}$ belong to cluster i^x).
3. recompute the cluster centers. The new centroid is $\mathbf{w}_i = \frac{1}{|cluster_i|} \sum_{x_j \in cluster_i} \mathbf{x}_j$.
4. If a convergence criterion is not met, go to step 2.

This method is easy and in general efficient but the number k must be provided (depends on it) and it has no visual properties. Use **softmax** to avoid confinement to local minima.

14.3 Self Organizing Map

They consist of n neurons located on a regular low-dimensional, usually 2D, grid. Each unit can be identified by the coordinates on the map, receives the same input $\mathbf{x}$ and has a weight $\mathbf{w}$.
Usage of SOM: clustering, pattern recognition, data compression, projection and exploration.
Competitive Learning is an adaptive process in which neurons gradually become sensitive/specialized to different input categories/ set of samples (the winner learns more).

Model and Algorithm

The aim of SOM algorithm is to learn a map from vectors to a discrete output space (the coordinates of the unit on the map).

1. The weights of the map are randomly initialized.
2. A sample input $\mathbf{x}$ is drawn.
3. Competitive stage: the current input vector $\mathbf{x}$ is compared with all the unit weights, the winner is $i^*(\mathbf{x}) = argmin_i ||\mathbf{x} - \mathbf{w}_i||$, where i is an index on the map.
4. Cooperative stage: the weight of the winning unit and the weight of the units in its neighborhood are moved closer to the input vector.
5. Continues until convergence, when there are no changes

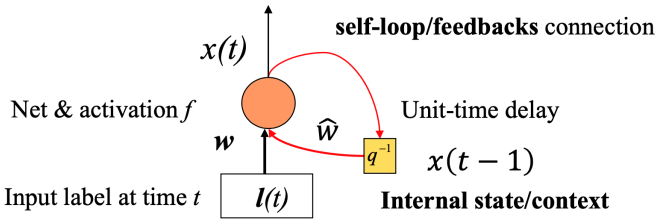
Topological information is supplied to the map because both the winning units and its lattice neighbors receive similar weight updates that allow them, after learning, to respond to similar inputs.

15 Recurrent Neural Network

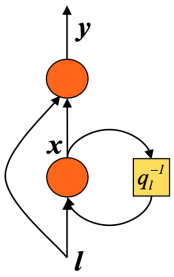
"The Neural Network That Remembers", with short-term memory, recurrent neural networks are able to remember, thus gaining amazing abilities. A recurrent neural network includes connections between neurons in the same hidden layer, some of which feed back on themselves. RNN nowadays are reference approach for sequence (structured data) processing.

15.1 Recurrent Unit

Uses the current unit plus states information. Recurrent neural unit with feedbacks in the architecture. The state summarizes the past information (context). The encoding of the past is adaptive (du to the free weights of the model).



15.2 State Transition System



$$\begin{cases} \mathbf{x}(t) = \tau(\mathbf{x}(t-1), l(t)) \\ \mathbf{y}(t) = g(\mathbf{x}(t), l(t)) \end{cases} \tag{42}$$

where τ is the state transition function realized by a neural network. The states summarizes the past inputs. $\mathbf{x}$ can be a set of states.

15.3 Simple Recurrent Neural Network

They are universal approximators of non-linear dynamical systems. RNN are based on the following assumptions:

- Causality: a system is causal if the output at a time only depends on inputs at previous times.
- Stationarity: time invariant (after model training).
- Adaptivity: transition functions are realized by neural networks with free weights, hence they are learned from data.

15.4 Learning Algorithms

The learning algorithm must take into account the set of encode transitions developed by the model for each step of the inputs. **Backpropagation Through Time** (BPTT) and **Real Time Recurrent Learning** (RTRL) are supervised learning algorithms designed for recurrent neural networks. In different style, they compute the same gradient values of the output errors across an **unfolded** network that is equivalent to the recurrent one (encoding network). The vanish of gradients over this deep network can make difficult to learn long-term dependencies.