Machine Learning Resumé

Inhaltsverzeichnis

Part I

Three pillars

Unsupervised Learning: inferring a function to describe hidden data from **unlabeled** data.
(e.g., Clustering)
Supervised Learning: inferring a function to describe hidden data from **labeled** training data.
(e.g., Classification, Regression)
Reinforcement Learning: Learning how software agents should take actions **in an environment** as to **maximize** some notion of cumulative **reward**.
(e.g., playing games)

	Input	**Objective**
Supervised Learning	(x_1, y_1), …., (x_N, y_N) observations of the following measurements: X: vector of p predictor measurements Y: outcome measurement (Regression: Y quantitative, Classification: Y from finite, unordered set)	Predict unseen test data. Understand which inputs affect the outcome and how.
Unsupervised Learning	Sample of unlabeled data	Fuzzier: find groups in sample, who behave similarly. (can be useful as pre-processing step for supervised learning)

Example:
We have *Sales* (Y) for different features *TV*, *Radio*, *Newspaper* (X_1, X_2, X_3). We want to predict *Sales*:

$$\text{Sales} \sim f(\text{TV}, \text{Radio}, \text{Newspaper})$$

Thus, we write with a measurement error:

$$Y = f(X) + \epsilon$$

Regression function:

$$f(x) = \mathbb{E}(Y|X = x)$$

⇒ **Problem:** we have normally few if any values for X = x

Nearest Neighbour Average:

We replace x by some neighbourhood of x:

$$\tilde{f}(x) = \mathbb{E}(Y|X \in \mathcal{N}(x))$$

Works well for little p and large N (few features, many data).

⇒ **Problem:** gets lousy for large p, since the nearest neighbour tend to be far away in high dimensions

Cross Validation and Bootstrap

Problem: We have only the error to the training data, but not to the test data.
Solution: Largely designed test data set (often not available). Hence, we consider a class of methods that estimate the test error by **holding out a subset of the training observations** from the fitting process, and **then applying the statistical learning method to those** held out observations.

Validation Set Approach

Randomly divide the available set in two samples: training set and validation (hold out) set.
Problem: estimate can be highly variable depending on which observations have left out, only half of the available data is used to fit the model

⇒ Overestimate test error

K-fold Cross Validation

Randomly divide the data into K equal-sized parts. We leave out part k, t the model to the other K − 1 parts (combined), and then obtain predictions for the left-out kth part. This is done in turn for each part k = 1, 2, ...K, and then the results are combined.
Problem: bias minimized for K = N, but high variance. Therefore, often K = 5, 10 used as a compromise.
Provides estimate of test error (no overlap between the different sets).

Bootstrap

Mimic new data sets with the computer. Each of these bootstrap data sets is created by sampling with replacement and is the same size as our original dataset. As a result, some observations may appear more than once in each bootstrap data set and some not at all.
Problem: The significant overlap between the samples will cause a serious underestimation of the test error.
Primarily used for standard error.

Pre-Processing

Feature Scaling: Centering and standardization of all/certain features x_i.
Feature Engineering: Introduction of dummies for categorical features.

Methods

Linear Regression

Model:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon$$

We estimate $\beta_0, \ldots, \beta_p$ as the values that minimize the squared residuals:

$$\text{RSS} = \sum_{1}^{n} (y_i - \beta_0 - \beta_1 x_1 - \ldots - \beta_p x_p)^2$$

Problem: The ideal scenario is when predictors are uncorrelated (each coefficient can be tested and estimated independently). If not, the variance tends to increase dramatically, and interpretations become hazardous.

Problem of ordinary squared: can lack accuracy of prediction (especially when n > p) and difficult to interpret.

Upgrades to ordinary squared residuals

Subset selection: We use only a subset, which we think is related to the response.

- ⇒ Best subset selection (try everyone and evaluate) not always possible for large p (possible overfitting, high variance).
- ⇒ Stepwise selection (iterate: add one predictor to subset and chose best one)

Shrinkage: The estimated coefficients are shrunken towards zero relative to the least square estimates. Reduces variance.

Dimension Reduction: We project the p predictors into a M-dimensional subspace with M<p (by computing M different linear combinations or projections).

Shrinkage

Ridge Regression

Model: We estimate $\beta_0, \ldots, \beta_p$ as the values that minimize:

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$

Where λ tuning parameter (chosen by cross-validation). Second term, called *shrinkage penalty*, is small when $\beta_0, \ldots, \beta_p$ small to zero.

Problem: Will include all p predictors.

Lasso

Model: We estimate $\beta_0, \ldots, \beta_p$ as the values that minimize:

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| = RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

Advantage: Yields sparse models = models, where only a fraction of parameters is non-zero (the L_1 penalty will force some β to be equal to zero and includes therefore only a subset of predictors).

! There is no model between these two who generally dominates other!

Dimension Reduction

Model: Let $Z_1 \ldots, Z_M$ be the linear combinations of our p original predictors:

$$Z_m = \sum_{1}^{p} \phi_{mj} X_j$$

We fit then the OLS Regression with

$$y_i = \theta_0 + \sum_{m=1}^{p} \theta_m Z_{im} + \epsilon_i$$

Advantage: Can outperform OLS Regression and win in the bias-variance trade-off.

Principal Component Analysis

To determine the linear combinations of the original predictors:

1. First principal component is (normalized) linear combination of the variables with the largest variance.
2. Second principal component has largest variance, subject to being uncorrelated with the first.
3. …

Hence with many correlated original variables, we replace them with a small set of principal components that capture their joint variation.

Support Vector Machines (SVM)

Approach two-class classification problem (separating data in two classes: 'normal' and 'abnormal', e.g., spam and no spam).

Model: Find a plane that separates the classes in feature space (sometimes necessary to soften separation or enlarge the feature space).

- ⇒ **Maximal Margin Classifier:** Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.
- ⇒ **Support Vector Classifier:** Maximizes a soft margin (noisy data).
- ⇒ **Feature Expansion:** Enlarge the space of features by including transformations. Fit a SVC afterwards.
- ⇒ **Kernel trick:** To find nonlinear separations.

More than two (K) classes?

OVA: One versus all. Fit K different 2-class SVM classifiers. Classify the class, for which the classifier is the largest.

OVO: One versus One. Fit all pairwise classifiers. Classify the class, that wins the most pairwise competitions.

! If K is not too large, chose OVO!

Tree-based Methods

Can be applied to both regression and classification problems.

Idea: We divide the predictor space X_1…, X_p into J distinct and not overlapping regions R_1…, R_J. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

Model: We will use boxes R_1…, R_J, that minimizes:

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

Where $\widehat{y_{R_j}}$ is the mean for the training observations within the j-th box.

Problem: Infeasible to calculate every possible combination of J boxes. Thus, tree-algorithm.

Tree Algorithm:

1) Recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2) Apply **cost complexity pruning** to the large tree to obtain a sequence of best subtrees, as a function of γ:

Subtrees indexed by nonnegative tuning parameter γ, for each value for γ there corresponds a subtree T which minimizes:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \gamma |T|$$

|T|: number of terminal nodes of the tree T,
R_m: subset corresponding to the m-th terminal node.
y_{Rm}: is the mean of the training observations in R_m.

3) Use K-fold cross-validation to choose γ. For each k = 1..., K:
   a. Repeat Steps 1 and 2 on the K−1/K-th fraction of the training data, excluding the k-th fold.
   b. Evaluate the mean squared prediction error on the data in the left-out k-th fold, as a function of γ.

   Average the results and pick γ to minimize the average error.
4) Return the subtree from Step 2 that corresponds to the chosen value of γ.

For classification trees, adapt the used error. $\hat{p}_{mk}$ represents the proportion of training observations in the m-th region that are from the k-th class.

Gini Index: considered as node purity- a small value indicates that a node contains predominantly observations from a single class:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Cross-Entropy: very similar numerically

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Bagging

Build multiple Training sets (for example with Bootstrapping), make predictions for each training set, and build afterwards the mean of all predictions (classifications trees: majority vote).

Random Forests

When building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors (often $m \sim \sqrt{p}$ used).

Boosting

Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Summary

Decision trees are simple and interpretable models for regression and classification. **But** not competitive with other methods in terms of prediction accuracy

⇒ Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. The latter two methods- random forests and boosting -are among the state-of-the-art methods for supervised learning. **But** their results can be difficult to interpret.

XGBoost

Idea: Optimize the tree structure for a general loss function l and $\hat{y}_i^{(B)} = \sum_{b=1}^{B} f_b(x_i)$ by minimizing the objective:

$$\sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{b=1}^{B} \Omega(f_b)$$

$$\text{where } \Omega(f_b) = \gamma|T| + \frac{1}{2}\sum_{j=1}^{|T|} \omega_j^2$$

$$|T| = d + 1 \text{ number of regions}$$

$$\omega_j \text{ regression weight of region } R_j$$

Which function for f_b: f will describe the weights.

$$f_t(x_i) = w_{J(x_i)}$$

$$\text{where } J : \mathbb{R}^p \to T, x \mapsto R_j \text{ for } j : x \in R_j$$

Optimization of weights:

We want to optimize the objective from before.

$$\text{Obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + C$$

We define:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

And apply a Taylor Expansion for l:

$$\text{Obj}^{(t)} = \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t) + C$$

For a **general l^2-loss**:

$$\text{Obj}^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))^2 + \Omega(f_t) + C$$

$$= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + C$$

And we use for Taylor:

$$g_i = \partial_{\hat{y}_i^{(t-1)}} (\hat{y}_i^{(t-1)} - y_i)^2 = 2(\hat{y}_i^{(t-1)} - y_i)$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} (\hat{y}_i^{(t-1)} - y_i)^2 = 2$$

By regrouping the terms by region R_j (page 53, week 3), we get finally:

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$\text{Obj}^{(t)} \sim -\frac{1}{2} \sum_{j=1}^{|T|} \frac{G_j^2}{H_j + \lambda} + \gamma|T|$$

$$\text{where } G_j = \sum_{i : x_i \in R_j} g_i \text{ and } H_j = \sum_{i : x_i \in R_j} h_i$$

Model Stacking

Idea: extends the idea of the weighted ensemble and uses a meta learner to and the best combinations of the ensemble members (e.g., many models of the same type but different hyper-parameters or all tricks from before- bagging, column/feature sampling).

Interpret a Model

Partial Dependence Plot

Idea: shows the marginal effect of a feature on the predicted outcome of a previously fit model. Can show if the relationship between the target and a feature is linear, monotonic, or more complex.

Problem: can be misleading when there are interactions to other predictors which average out.

Individual Conditional Expectation Plot

Idea: Rather than plot the target predictors' average partial effect on the predicted response, plot the n estimated conditional expectation curves.

Additive Feature Attribution

Idea: explain an original prediction model f by an explanation model g (which is a linear function of binary inputs):

$$g(z') = \phi_0 + \sum_{j=1}^{m} \phi_j z'_j$$

$$\text{where } z' \in \{0,1\}^m, \; m \text{ is the number of simplified input features, and } \phi_j \in \mathbb{R}$$

SHAP (Shapley Additive explanations)

Idea: uses game theory to derive an optimal attribution to features for explaining a predicted outcome.

LIME (Local interpretable model-agnostic explanations)

Idea: fits local, interpretable models to explain a prediction.

4. G class of potentially interpretable models (such as additive features attributions)
5. Ω a measure of complexity (maybe some elements of G are not simple enough)
6. z' acts over absence (0)/presence (1) of the interpretable components
7. $\pi_x(z)$ proximity measure between an instance z to x
8. $L(f, g, \pi_x)$ measure of how unfaithful g is in approximating f in the locality defined by π_x

Then LIME is defined by

$$\xi(x) = \text{argmin}_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

For linear g we use normally

$$L(f, g, \pi_x) = \sum_{(z,z') \in Z} \pi_x(z)(f(z) - g(z))^2$$

$$\pi_x(z) = \exp(-D(x, z)^2 / \sigma^2)$$

For some distance function D and width σ.

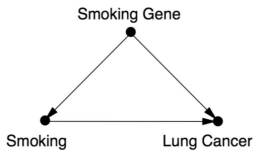
Problem: The interpretation models presented in the following part give just associations and no causality (causality-vs-correlation problem), we also have no guarantee on statistical error bounds for the feature attributions.

Causal Inference

Idea: Attempts to draw causal conclusion from observational data.

Causal Diagrams:

Chain A -> B -> C: B is a mechanism that transmits the effect of A to C (e.g., fire -> smoke -> alarm).
Fork A <- B -> C: B is often called a confounder of A and C, it makes A and C statistically correlated even though there is no direct causal link between them (e.g., shoe size <- age of child -> reading ability).
Collider A -> B <- C: if A and C are independent in the beginning, conditioning on B will make them dependent (e.g., talent -> celebrity <- beauty).
Confounding: The true causal effect X -> Y is mixed with the spurious correlation between X and Y induced by the fork X <- Z -> Y. For example: Exists a smoking/lung cancer gene?



Randomized Controlled Trial

Idea: randomly assign the test/control group. Thus, the assignment is completely independent of all other variables, confounders cannot act on the result in a systematic way anymore.
Problem: Not always possible (e.g., for ethical or cost reasons).

Do-Operator

Idea: recreate RCT artificially. $do(X = x)$ means, that we force X to be x. We have generally

$$\mathbb{P}(Y|do(X) \neq \mathbb{P}(Y|X)$$

We call everything, which makes the probability differ confounding.

Backdoor-Criterion

Backdoor path from X to Y: path (regardless of the arrow directions), which are not directed from X to Y.
(Directed Path from X to Y: chain of causes, Backdoor-Path: Creates correlation but not causality).
Backdoor-Criterion: A set of variables Z satisfies the backdoor criterion relative to variables (X, Y) in a G if:

9. no node in Z is a descendant of X.
10. Z blocks every path between X and Y that contains an arrow pointing into X (= blocks every Backdoor path).

Backdoor Theorem: If a set of variables Z satisfies the backdoor criterion relative to (X, Y), then the causal effect of X on Y is identifiable, and it holds:

$$\mathbb{P}(Y|do(X), Z) = \mathbb{P}(Y|X, Z)$$

Adjustment: If Z is a set of variables, blocking all the backdoor paths for the causal effect X -> Y, then:

$$\mathbb{P}(Y|do(X)) = \sum_z \mathbb{P}(Z = z)\mathbb{P}(Y|X, Z = z)$$

Front door-Criterion:

Front door Theorem: If Z is a set of mediators for the causal effect X -> Y, then Fundamental

$$\mathbb{P}(Y|do(X)) = \sum_z \mathbb{P}(Z = z) \sum_x \mathbb{P}(X = x)\mathbb{P}(Y|X = x, Z = z)$$

rules for Do-Operator

I. If W is independent of Y conditional on Z, then:

$$\mathbb{P}(Y|do(X), Z, W) = \mathbb{P}(Y|do(X), Z)$$

II. If Z blocks all back-door paths from X to Y, then:

$$\mathbb{P}(Y|do(X), Z) = \mathbb{P}(Y|X, Z)$$

III. If there are no causal paths from X to Y, then:

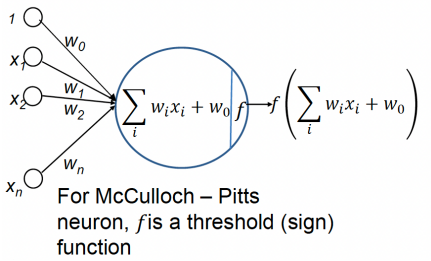
$$\mathbb{P}(Y|do(X)) = \mathbb{P}(Y)$$

Part II

Historical Models

McCulloch – Pitts (1943): First model with the main idea, that synchronous assembly of neurons is capable of universal computations (aka equivalent to a Turing machine)

Perceptron (1958): Decision cell is a threshold function (= McCulloch-Pitts Neuron).



For McCulloch – Pitts neuron, f is a threshold (sign) function

Input: Labelled data set

$$\left\{(x^i, y^i), i = 1, \ldots, N, x \in \mathbb{R}^n, y \in \{-1, 1\}\right\}$$

Output: Classifier ω (from R^n) and decision

$$F(x) = \text{sgn}\left(\sum_{i=0}^{n} \omega_i x_i\right)$$

Algorithm:

Initialize $\omega(0)$

Repeat:

$\text{Choose } (x^i, y^i)$

$\text{If } y(t)\omega(t).x(t) \leq 0 \text{ then } \omega(t+1) = \omega(t) + \epsilon y(t)x(t)$

$\Rightarrow$ learning rule stochastic gradient algorithm for minimizing the number of wrongly predicted labels

Adaline (1959): Linear unit as decision and as a loss c Least Mean Square

$$c(x, y) = ||y - F(x)||^2 = ||y - (\sum_i \omega_i x_i + \omega_0)||^2$$

Algorithm (Stochastic gradient algorithm):

Repeat:

$\text{Choose } (x^{(t)}, y^{(t)})$

$\omega(t+1) = \omega(t) - \epsilon \nabla_\omega c(x, y)$

Summary: Supervised learning algorithms, stochastic optimization algorithms (for NNs, most are based on Stochastic gradient descent). But Learning is not equal optimization (we want a function which generalizes well).

Optimization

Gradient Methods

Batch Gradient Algorithms

Loss function: Sum of individual losses for each example.

Step in algorithm: descent direction Δ and gradient step ε.

$$\omega(t+1) = \omega(t) + \epsilon(t)\Delta_\omega(t)$$

Stochastic Gradient Algorithms

Objective: Training NNs is finding the parameters ω by optimizing a loss.
Problem: Deep NN have many parameters and meta-parameters, the loss is most often a non-linear function of these parameters: the optimization problem is nonconvex.
Solution: No unique answer!

-> The most common family of optimization methods for Deep NN is based on stochastic gradient algorithms (Exploit the redundancy in data, at the cost of high variance in gradient estimates)

Loss function: Sum of individual losses for each example.
Basic SGA: Iterate until stop criterion (but might produce many oscillations):

$$\omega(t+1) = \omega(t) - \epsilon \nabla_\omega c(x^{(t)}, y^{(t)})$$

Momentum:

$$m(t) = \gamma m(t-1) + \epsilon \nabla_\omega c(x^{(t)}, y^{(t)})$$
$$\omega(t+1) = \omega(t) - m(t)$$

Adaptive Learning Rate: Has a learning rate for each parameter ω_I (e.g., Adagrad and RMS prop). Can be combined with Momentum (e.g., Adam).

Numerical Integration of ODEs

Allows us to use the results from numerical analysis to characterize useful properties e. g. stability / consistence.

Summary

No one solution for all.

11. For large scale applications, Adam is often used today as a default choice together with minibatches (but simple SGD with heuristic learning can sometimes be competitive).
12. Stochastic methods exploit data redundancy.
13. Mini Batch well suited for GPU.

Regression and Logistic Regression

Linear Regression

Linear Model:

$$F(x) = \omega.x = \sum_{i=0}^{n} \omega_i x_i$$

Loss Function: Mean square error.

$$C = \frac{1}{2} \sum_{i=1}^{N} (y^i - w.x^i)^2$$

Steepest descent gradient:

$$\omega = \omega(t) - \epsilon \nabla_\omega C$$
$$\omega = \omega(t) + \epsilon \sum_{i=1}^{N} (y^i - \omega.x^i) x^i$$

Probabilistic Interpretation: We introduce random variable (error term) ε.

$$F(x) = w.x + \epsilon$$

1) ε is iid. Gaussian

$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

Then we have as a posterior distribution

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

2) (Log-)Likelihood
   Then maximize L or l = log L

$$L(\omega) = \prod_{i=1}^{N} p(y^i|x^i;\omega)$$

Logistic Regression

Linear regression for classification. Here y in our sample not from R, but {0,1}.
Linear Model: We use the logistic (or sigmoid) function σ.

$$F(x) = \sigma(w.x) = \frac{1}{1+\exp(-w.x)}$$

Probabilistic Interpretation: as before with a Bernoulli-Hypothesis for the posterior distribution

$$p(y=1|x;\omega) = F(x),\ p(y=0|x;\omega) = 1 - F(x)$$

Can be generalized for p classes:

$$p(y=i|x;\omega) = \frac{\exp(w_i.x)}{\sum_{j=1}^{p}\exp(w_j.x)}$$

Non-Linear Regression

The results extend to non-linear functions, e.g., F(x) as a NN. For linear regressions models equivalently to Gaussian hypothesis (and resulting likelihood), for classification problems equivalently to Bernoulli/Multinomial-Hypothesis.
Mean square loss: Optimal solution given by:

$$F_{\omega^*}(x) = \text{argmin}_{\omega}\mathbb{E}_{x,y}\left[\left(\mathbb{E}_y(y|x) - F_\omega(x)\right)^2\right]$$

Multilayer perceptron

Neurons arranged into layers, each neuron non-linear unit.

Back-Propagation

We note $\odot$ the pointwise operator: $f \odot x = (f(x_1), f(x_2))$

Forward pass:

$$\hat{y} = F(x) = f \odot \big(w(2) f \odot (W(1)x)\big)$$

Compute error: $c(y, \hat{y})$ e.g., mean square error or cross-entropy.

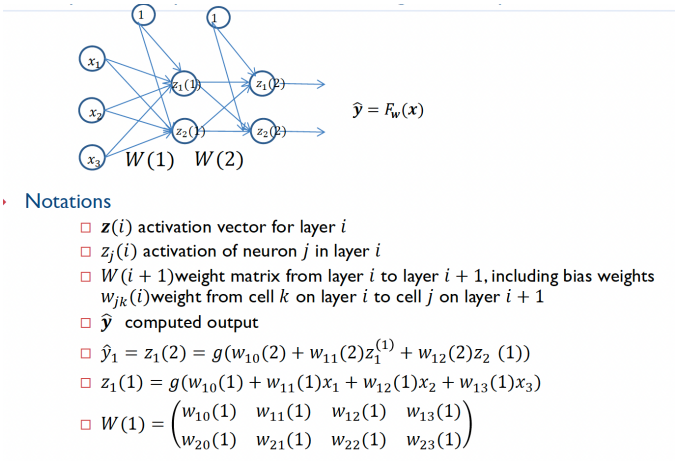
Backward pass:

$$w_{i,j} = w_{i,j} - \epsilon\frac{\partial c(y,\hat{y})}{\partial w_{i,j}}$$

Automatic/Algorithmic Differentiation

Back-Propagation is an instance of AD. All modern Deep Learning implement AD.
Idea: A mathematical expression can be written as a computation graph, i.e., graph decomposition of the expression into elementary computations. AD allows to compute efficiently the derivatives of every element in the graph w.r.t. any other element.

⇒ AD transforms a program computing a numerical function into the program for computing the derivatives.



In more detail

Forward pass ->

The activations of cells on layer 1 are then used as inputs for layer 2:

$s(2) = W(2)z(1)$ and $\hat{y} = z(2) = g(s(2))$

Summary

Back-Propagation instance of more general technique *Adjoint Method*:

- has been designed for computing efficiently the sensitivity of a loss to the
- parameters of a function (e.g., weights, inputs, or any cell value in a NN).
- Used in many fields like control or geosciences.

(Missing Adjoint Method, pages 69-76)

Complexity Control

Problem: Complex problems perform sometimes worse than simple linear ones (Overfitting). Thus, the model complexity should be adjusted to the task and the given information.

Bias-Variance

Bias-Variance-Problem: Conflict in trying to simultaneously minimize:

- **Bias error**: error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
- **Variance:** error from sensitivity to small fluctuations in the training set. High variance may result from an algorithm modelling the random noise in the training data (overfitting).

Bias-Variance-Decomposition: Analyses a learning algorithm's expected generalization error with respect to a particular problem as a sum of three terms, the bias, variance, and a quantity called the irreducible error, resulting from noise in the problem itself.

⇒ Requires compromise between flexible model (low bias, high variance) and simple model (strong bias, low variance)

Regularisation

A problem is **well posed** if there is one and only one solution.

Idea: control the solution variance by constraining function F.

Model: We will optimize (find weights w) the compromise C between C_1 (objective) and C_2 (constraints on solution, λ constraint weight):

$$C = C_1 + \lambda C_2$$

Update as before:

$$w = w - \epsilon \nabla_w C$$

For the multivariate regression model (q = 1: l_1-regularization - Lasso, q=2: l_2-regularization - Ridge):

$$C = \frac{1}{N}\sum_{i=1}^{N}(y^i - w.x^i)^2 + \lambda\frac{1}{2}\sum_{j=1}^{n}|w_j|^q$$

Consequence: Network with high weights has probably learned the statistical noise (corrects high but far away data). By introducing the constraint, high weights are penalized (by constraint weight lambda) and it works against overfitting (to noise). But high weights nevertheless still possible.
But modern Deep learning does not follow in general the common complexity wisdom (Double descent phenomena)!

Deep Learning

Activation Function

Calculates the output of a node in a neural network. Some properties and their effects:
Nonlinear: then a two-layer neural network can be proven to be a universal function approximator (= approximate any continuous function provided rather mild assumptions about the form of the activation function).
Range: if **finite**, gradient-based training methods tend to be more stable, because pattern presentations significantly affect only limited weights. If **infinite**, training is generally more efficient because pattern presentations significantly affect most of the weights.
Continuously differentiable: desirable for gradient-based optimisations.

Normalization

Batch Normalization: Normalize the activations of the units (hidden units) to coordinate the gradients across layers.
Gradient Clipping: Avoid very large gradient steps when the gradient becomes very large.
Dropout: Randomly drop units at training time

Types of Networks

Convolutional Neural Networks (CNN)

Developed for speech and image recognition.
Idea: Exploit local characteristics via local connections. Also possible on multiple channels (for example for coloured pictures with red-green-blue input).

Generative Adversarial Networks (GAN)

Learns a probability distribution model form data samples.
Issue: How to compare detected (proposed) distribution with real (unknown) distribution.
Solution: Here binary classification (but many different methods).

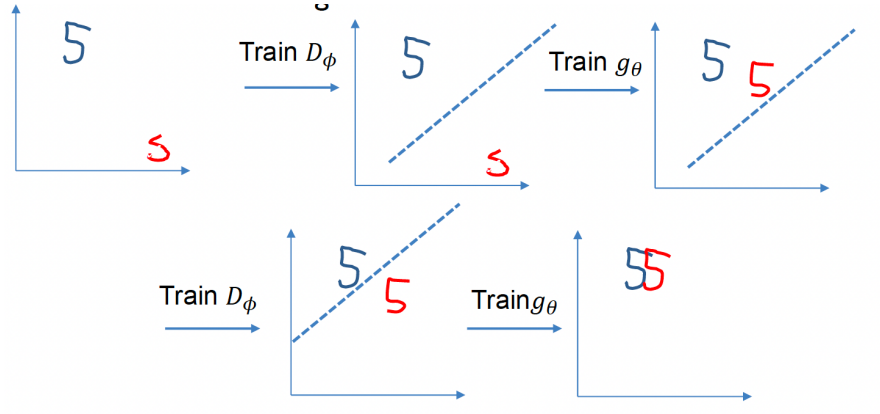
Principle:

1) Generative network generates data after sampling from a latent distribution.
2) Discriminant network tells if the data comes from the generative network or from real samples. The discriminator will be used to measure the distance between the distributions.
3) The two networks are trained together: The generative network tries to fool the discriminator, while the discriminator tries to distinguish between true and artificially generated data (MinMax game).

⇒ The Discriminator will force the Generator to be « clever » and learn the data distribution.

Remark: No hypothesis on the existence of a density function.

Training alternates between Discriminator and Generator:



Problems:

- Highly unstable: no convergence in oscillation, sometimes G collapses and produces repeatedly same distribution.
- Not explained: lot of theory, lot of heuristics, but no real explanation.
- Low dimensional supports: the real and proposed distribution may lay on low dimensional manifold without intersection. Then easy to separate them for the discriminator.

Extension: Conditional GANs, where the distributions are conditional to a latent variable.

Summary:

- Likelihood free training
- Optimized quality of generated sample
- **But** difficult to train.
    - Mode collapse, convergence
    - No best method, relies on heuristics.
    - Works on very high dimensional spaces.

Variational Auto-Encoders

Problem:

We will approximate posterior distribution of unobserved variables Z by using some

$$Q(Z) \sim P(Z|X)$$

given data X:

We restrict Q to simpler distribution (e.g., gaussian distribution).

Variational Methods: $p_\theta(x, z)$

We suppose having a joint model: . And we want to maximize the log-likelihood for our observed data x.

Kullback-Leibler-Divergence:

Measure of the difference between two probability distributions. For continuous variables:

$$D_{\mathrm{KL}}(p(y)||q(y)) = \int_y \log\left(\frac{p(y)}{q(y)}\right)p(y)dy$$

For discrete variables:

$$D_{\mathrm{KL}}(p(y)||q(y)) = \sum_i \log\left(\frac{p(y_i)}{q(y_i)}\right)p(y_i)$$

Evidence Lower bound:

To maximize the log-likelihood:

$$\log p_\theta(x) = D_{\mathrm{KL}}(q_\phi(z|x)||p_\theta(z|x)) + \mathrm{ELBO}(\theta, \phi, x)$$

We will maximize ELBO:

$$\mathrm{ELBO}(\theta, \phi, x) = -D_{\mathrm{KL}}(q_\phi(z|x)||p(z)) + \mathbb{E}_{q_\phi(Z|X)}(\log p_\theta(x|z))$$

Here, KL-divergence is a regularization term (forces the learned distribution q to stay close to the prior p). And the second term is a reconstruction term, which measures how well x can be reconstructed from latent representation z.

Link to NN implementation:

The generative $p_\theta(x|z)$ and inference $q_\phi(z|x)$ are implemented by NNs. They will be trained to maximize the second (expectation) term (often estimated by MC) with the constraint $D_{\mathrm{KL}} \sim 0$.