Machine learning

Deep Learning

Sven-Patrik Hallsjö

University of Glasgow | Experimental Particle Physics

- Based slides on material from and images from:
- Artificial Intelligence, a modern approach, Stuart Russell and Peter Norvig.
- Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville. http://www.deeplearningbook.org
- Python Machine Learning - Sebastian Raschka
- www.machinelearningmastery.com

- Prerequisites
- Linear Algebra
- Mathematical optimisation
- General programming, preferably Python

- Lecture 1
- Introduce ML
- Supervised vs unsupervised learning.
- Linear regression example
- Neurons
- Motivating Deep learning

- Lecture 2
- Deep learning
- XOR example
- Forward feed
- Backfitting
- Learning vs pure optimization
- CNN

- Lecture 3
- Real life applications / Current research
- Start discussing labs

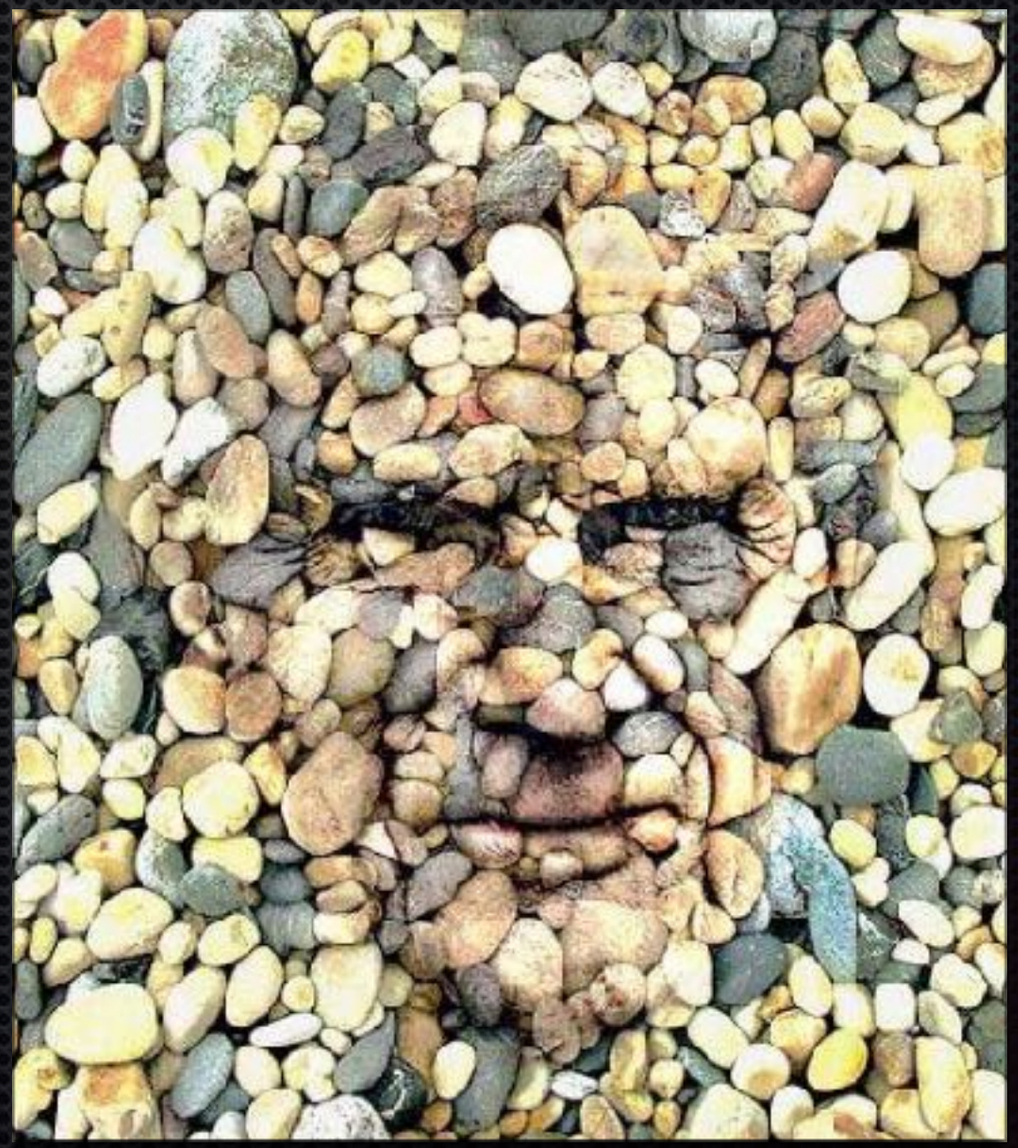
Lecture 1

Introduction

- Even the ancient Greeces dreamt of thinking machines
- They were thinking fully alive automatons. Now we are thinking of making use of our programmable computers.
- Artificial intelligence (AI) is a huge field
- Trying to develop software to automate “simple” labour
- Speech and image recognition

Introduction

- Initially computers were used to solve complicated for humans but straight forward mathematical problems.
- Challenge now is understanding the softer issues, easy/intuitive for humans such as recognising “hidden images” or dialects.
- https://www.moillusions.com/wp-content/uploads/2007/06/hiddenman-580x504.jpg



Introduction

- The present solution is to allow computers to learn from experience and to understand things as concepts with each concept defined in relation to simpler concepts.
- From this, using layers of layers of layers of concepts we get the approach of AI deep learning.
- By gathering knowledge from experiences there is a method that avoids the need for humans to formally specify all of the needed knowledge.
- Imagine how any program with enough if statements could be considered intelligent.

Introduction

- Many early AI projects, for instance deep blue (Chess) did not know anything about the “world”.
- Chess can be completely described with a short list of formal rules.
- Similar thing with GO!
- We want to move away from this box and go to things closer to human life.
- Requiring information, knowledge and intuition.
- Also difficult to write up formally.
- Key challenge.

Introduction

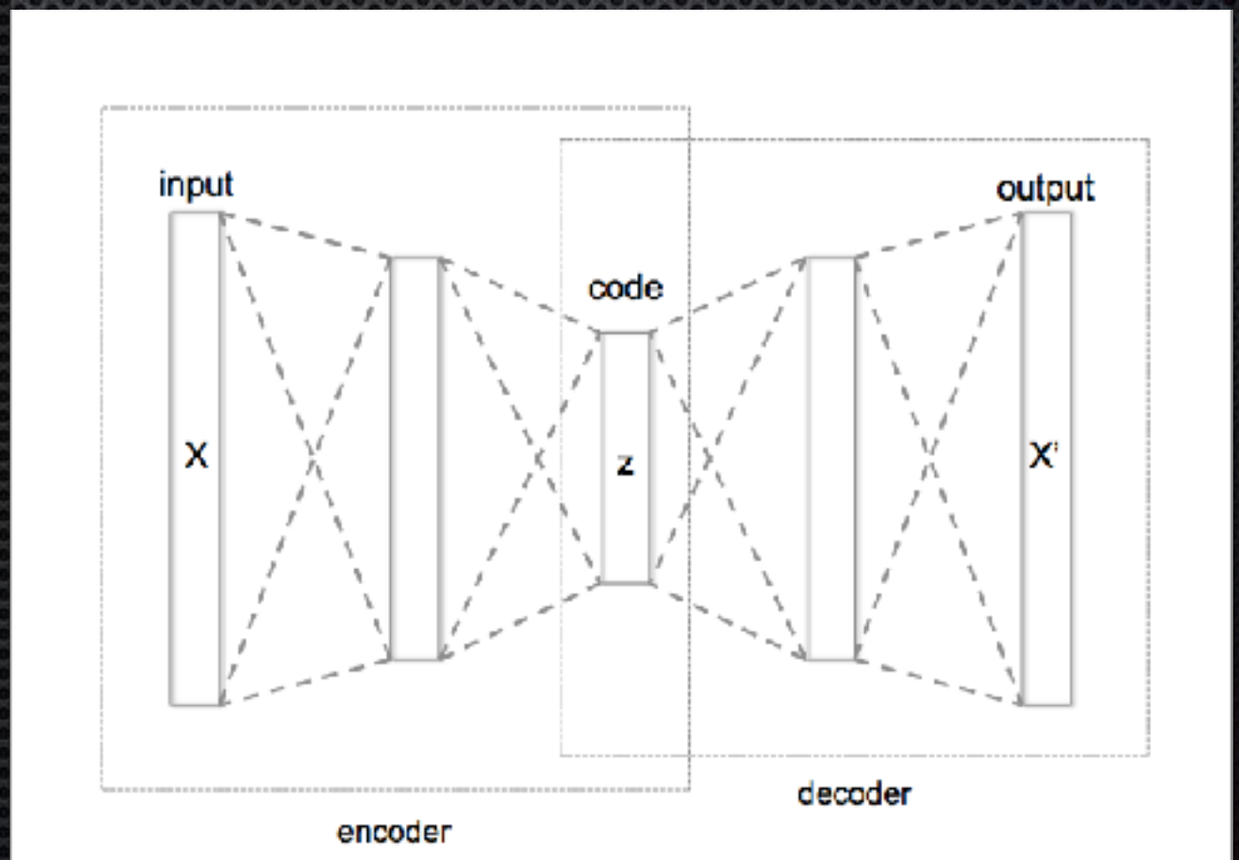
- Ai needs the ability to acquire knowledge, patterns from raw data. This is known as Machine learning.
- Can now teach computers knowledge of the real world and make seemingly subjective decisions.
- Logistic regression -> Medical decision making.
- Naive Bayes, spam filtering
- Requires representation of data, each provided feature
- These methods can not understand a picture, but can understand a technical report.
- Pixels do not have as big a correlation.

Introduction

- One thing to never forget is the representation.
- If the data is easily accessible, computers can collect it exponentially faster than having to type it in.
- Handle numbers vs strings, names vs bank id.
- Compare voice samples or use samples to estimate size of vocal cords?
- Very difficult to know a priori, identify cars by wheels or pixels?
- Is it a wheel or a shadow?
- Same at night as daytime?

Introduction

- One solution, use machine learning for representation.
- Representation learning, unsupervised
- Think sudo inverse, get as much information with as simple a representation.
- Autoencoder, both encoder and decoder.
- Trained to preserve as much information as possible.



Introduction

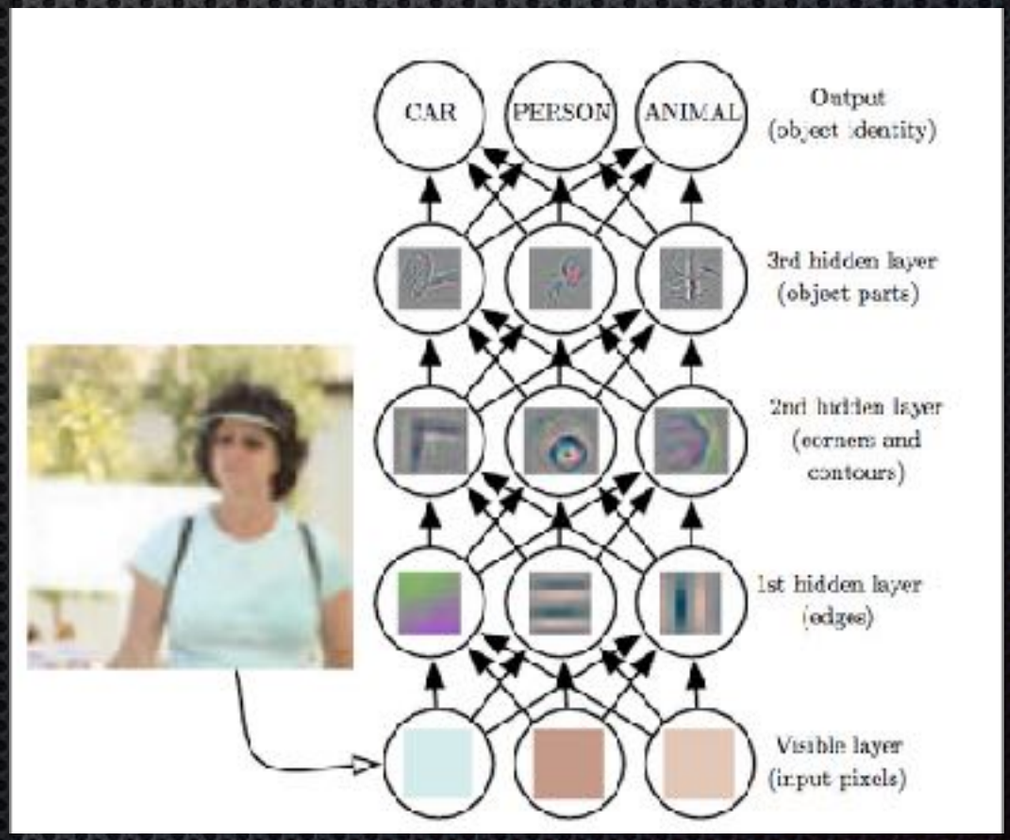
- Another difficulty, many factors or variation in real world applications.
- Think we can use images to classify horses vs zebras
- How handle variations in images?
- Angle, shape, size?

Introduction

- Almost as hard to find representation as solving the problem.
- Deep learning lets us express difficult representations as simpler representations.
- However, not always needed, could be too complicated for what we are trying to do.
- Could simply need a rotation or mapping.

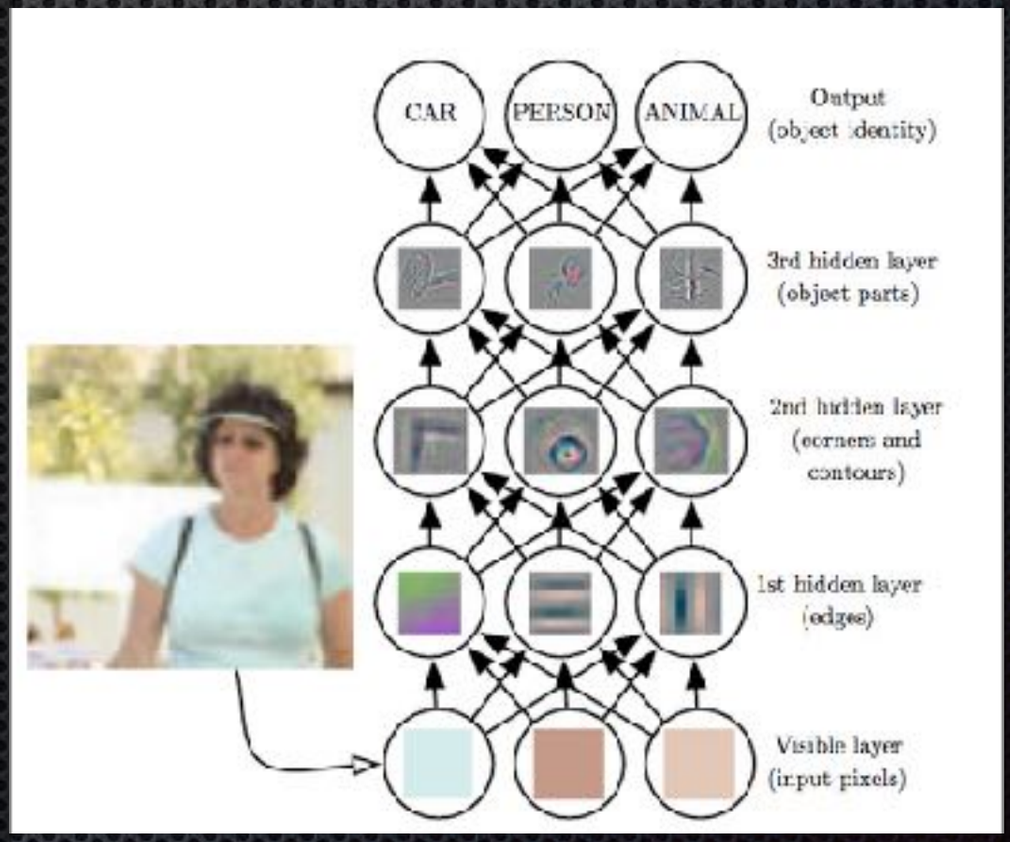
Introduction

- Illustrating how a deep neural net can identify an image.
- Raw data is just pixels. Perhaps impossible to tackle straight away.
- Use several different simpler mappings in each net.
- Notation, visible and hidden layers.
- What we can “observe” vs internal data layers and not directly seen in the output.



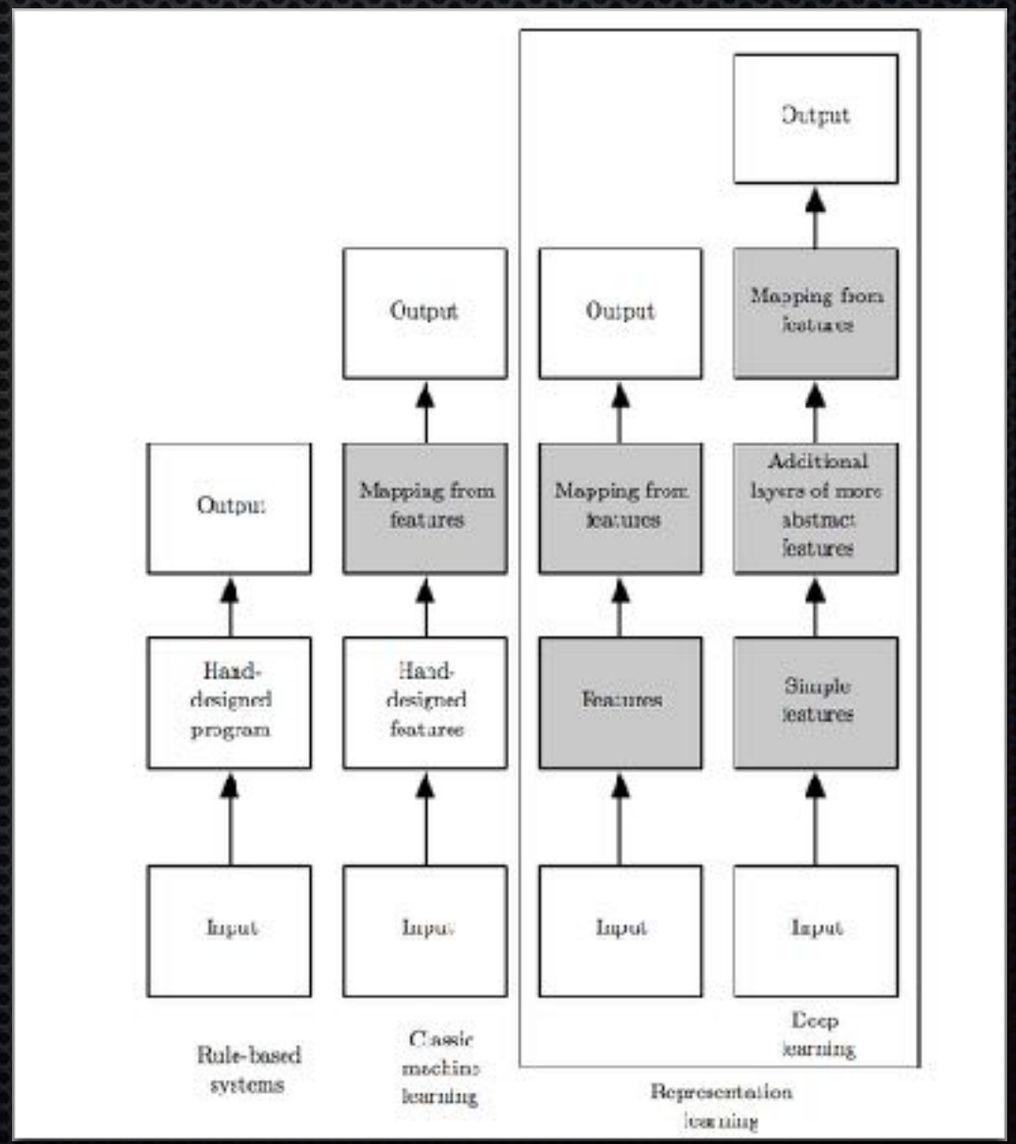
Introduction

- First layer finds edges by using brightness.
- Using corners, contours can be found as several edges.
- Using edges, can start looking finding objects.
- With objects, an easier task to classify.



Introduction

- How different parts relate to each other.
- Difference between models and planning.
- Shaded boxes are where we learn from data.



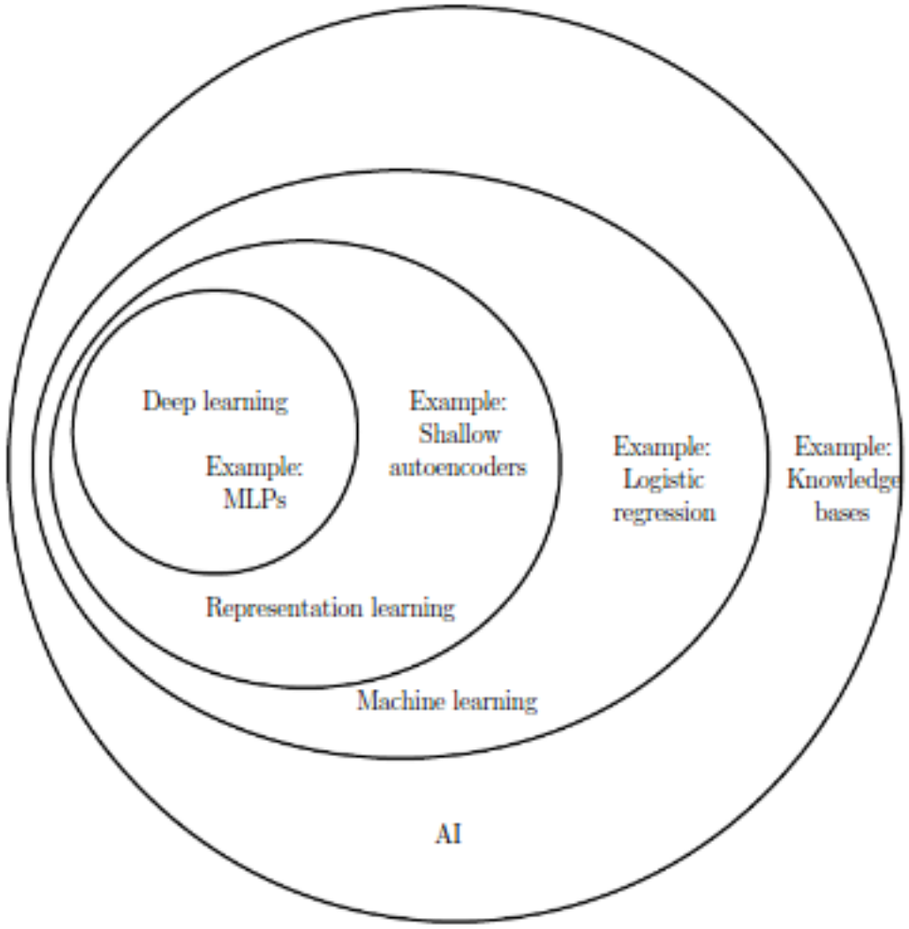


Figure 1.4: A Venn diagram showing how deep learning is a kind of representation learning, which is in turn a kind of machine learning, which is used for many but not all approaches to AI. Each section of the Venn diagram includes an example of an AI technology.

ML vs Ai - http://www.deeplearningbook.org

Don’t forget

- Logistic Regression (LR) is classified under Machine Learning (ML). One should not use Deep Learning (DL) for every possible problem.
- Don’t use Machine Learning (ML) or Deep Learning (DL) if you can write a simple program to solve the problem: *Using ML or DL for such a problem will be inaccurate or very inefficient*
- Always start with simple models before moving to complex once: Complex models have too many free parameters. It is very difficult to know how they will behave in unexplored cases.
- Don’t use DL if the problem is simple or if you get sufficient accuracy with a ML model: DL models are complex. Using DL for a simple problem is an overkill, the model will be inefficient
- Don’t use DL if you have small amount of data: You can’t train a DL model with small amount of data. The results will be really bad.
- Best situation is a lot of data with few very distinct parameters.
- This is not a magical method that solvs all problems.

Practically

- A machine learning algorithm is an algorithm that is able to learn from data.
- But what do we mean by learning?
- “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Task T

- Classification (With missing input)
- Regression
- Transcription (Find structure in text)
- Machine translation
- Parsing
- Anomaly detection
- Generators/Predictions of missing values/Denoising
- Density estimators

Performance P

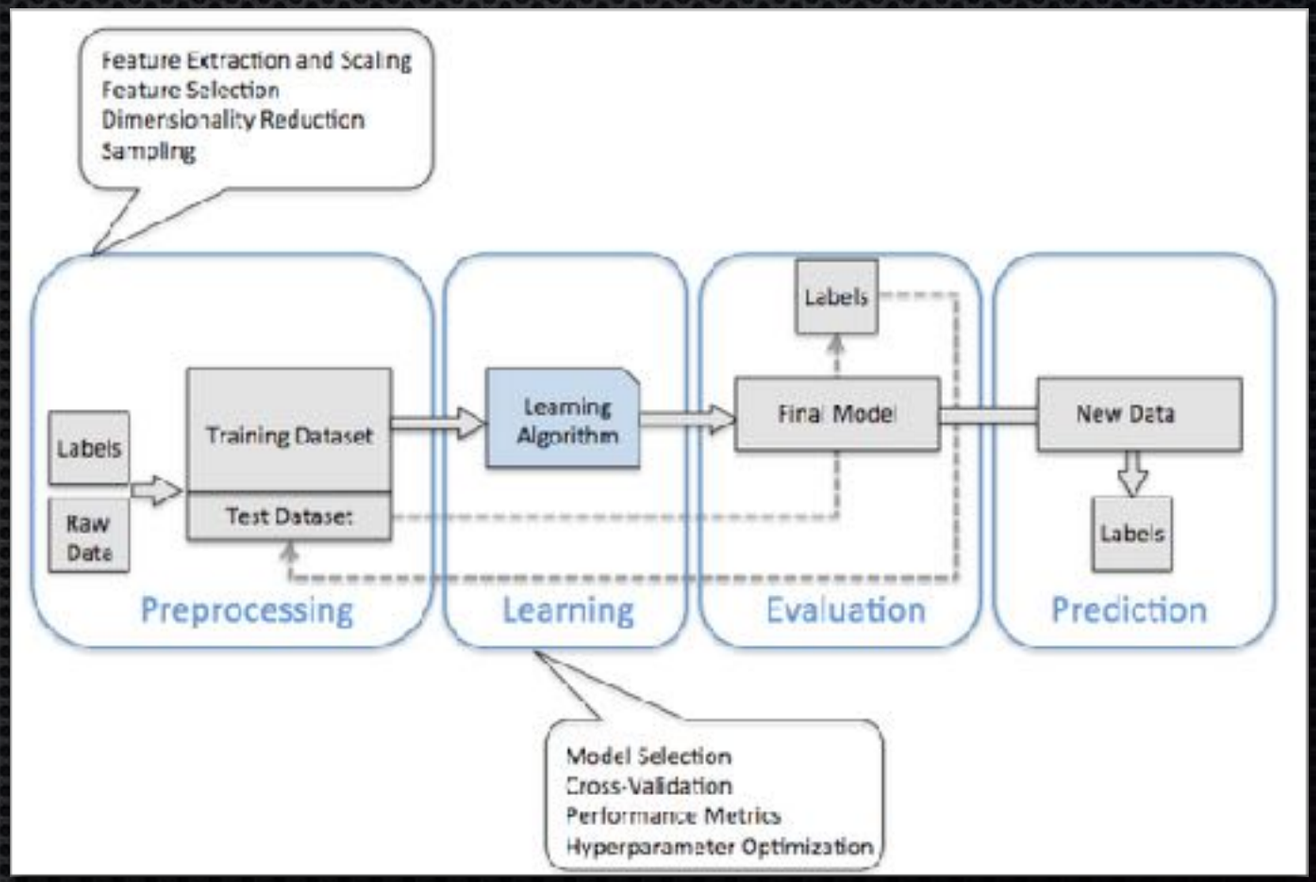
- Measure accuracy, proportion of examples of correct output vs incorrect.
- Can also output error rate as 0-1 loss.
- However all this needs to be performed on data that the algorithm was not trained on to give a good estimate.
- Can be quite difficult choosing the right performance index in some cases.

Experience E

- Unsupervised vs Supervised
- Want to find clusters, unknown patterns.
- Known pattern, want to classify.

How do we do it?

- Splitting data into Training, Testing and “Real”.
- Training used for the algorithm
- Testing to evaluate
- “Real” is yet unclassified data



Hopefully you now have an idea of the background
Lets move on to an example:

Learning linear regression on the board

Have the notes in the slides as well

- Linear regression is a very simple and limited example of ML
- It shows the how an algorithm can work and shows the basic principles.
- It also shows fundamentally what ML actually is.

- Task: Given a vector (position of our points) x, predict output y on the form : $y_{pred} = \vec{w}^T \vec{x}$
- Where w is our fitting parameters. Think of them weighing features w.r.t. predictions.
- How do we evaluate performance? In this example, evaluate the test set using MeanSquaredError:

$$MSE_{test} = \frac{1}{m} |\vec{y_{pred}^{test}} - \vec{y^{test}}|^2$$

- In our example we can simply derive our best W, only on train data.

$$\nabla_W MSE_{train} = 0$$

$$\rightarrow \nabla_W \frac{1}{m} |y_{pred}^{train} - y_{train} MSE_{train} = 0$$

$$\rightarrow \nabla_W \frac{1}{m} |X_{train} - y_{train}| = 0$$

$$\rightarrow \nabla_W (w^T X_{train}^T X_{train} w - 2w^T X_{train}^T y_{train} + y_{train}^T y_{train}) = 0$$

$$\rightarrow 2X_{train}^T X_{train} w - 2X_{train}^T y_{train} = 0$$

$$\rightarrow w = (X_{train}^T X_{train})^{-1} X_{train}^T y_{train}$$

- This may not seem like learning, but it is! Learn about w.

- We have to assume training error is expected test error.
- Assume this by assuming sampling from the same set.
- Have to ensure that they are both large enough samples.
- Thus minimising MSE train should be the same as minimising MSE test.

- This leads to two different problems, underfitting and overfitting.
- Essentially the gap between training and test error.
- Relating to our example, limit the order of the equation.
- Remember any linear curve fitting you've ever done.

- A big problem in ML when is trying to work with to few data points.
- Easiest solution is to preprocess the data, combine features.
- Too many data points, could be missing a feature, some underlaying assumption/variable.

- Logistical regression, y = 1 if < ϵ from our line given some w.
- How do we improve? Stochastic Gradient Descent
- Normal optimisation method, think Newton-Ralphson
- Step depending on derivative of error loss function.

- How improve data from test?
- In general, total error is J for a parameter theta

$$J(\theta) = E_{x,y\ \hat{p}_{data}} L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$$

- For an example loss function

$$L(x, y, \theta) = -logp(y|x; \theta)$$

- Could also be our MSE.

- Thus, derive wrt W or theta.

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

- In our fixed mini batch M', assuming it is representativ of M.

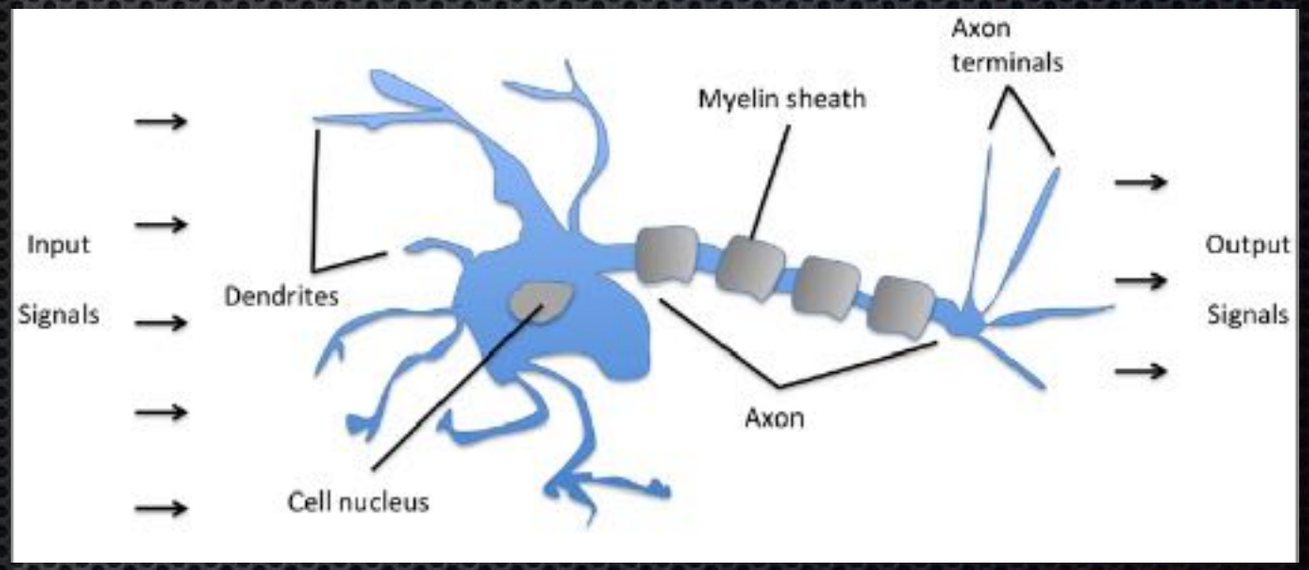
$$g = \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta L(x^{(i)}, y^{(i)}, \theta)$$

- Improve parameter theta by stepping with some size e

$$\theta \leftarrow \theta - \epsilon g$$

Neurons

- For some input x
- Create Z with weight W as Z =W^T*Z
- Return 1 if Z> U
- Else return 0 or -1
- U is the threshold
- Usually implemented with the heavy side function.



Neurons

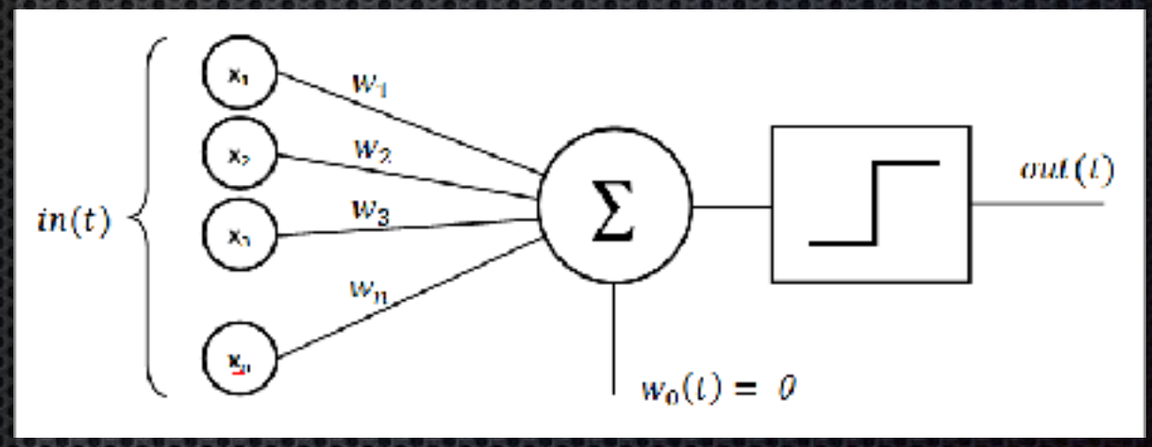
- Can put these in a chain
- Can throw the threshold into w0 = -U and x0 = 1
- Steps:
- Initialize all the weights to 0 or small random numbers
- For each sample x, Compute output value \hat{y}, update the weights given expected y.

$$w_j = w_j + \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$$

- eta is the learning rate, some value in range {0,1}

Perceptrons

- Find some features
- Our example quite simple, only one feature. one layer.
- Either “fire” or doesn’t.
- Will walk through a simple code version.



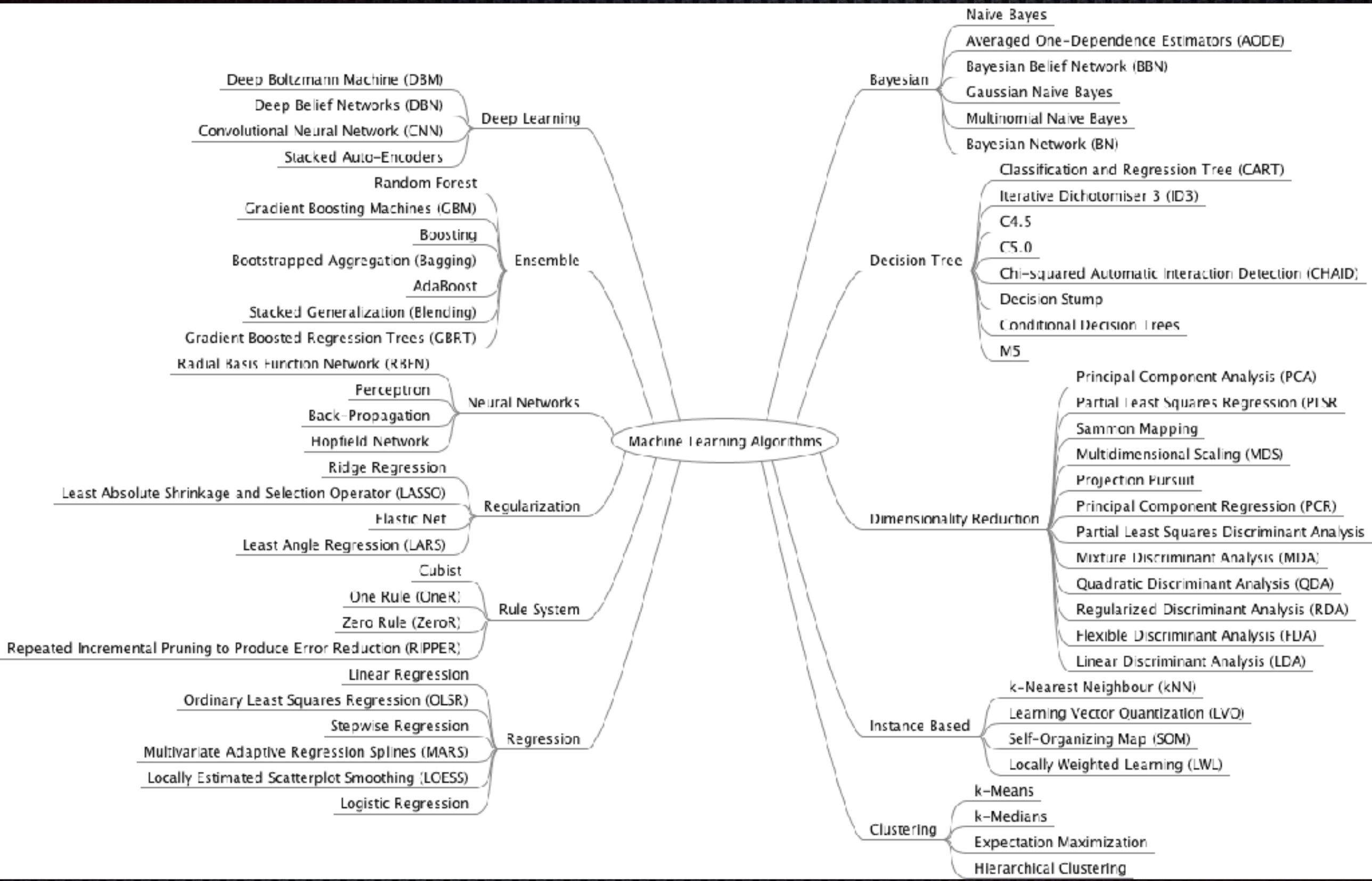
Example of perceptron with data.

You will implement this for the iris data set

Based on Python Machine Learning - Sebastian Raschka

Deep learning

- What happens if we set several neurons after each other?
- Good place to stop for today.


Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR)
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Any questions?

Otherwise just approach me at any time or send me an email.
p.hallsjo.1@research.gla.ac.uk

End lecture 1

Lecture 2

- Lecture 2
- Deep learning
- Forward feed
- XOR example
- Universal approximation
- Back-propagation
- Learning vs pure optimization
- CNN

I will not talk about optimisation.
Too problem specific, will just give an overview and how to use it.

See more in for instance the deep learning book.

Deep learning

- Deep learning lets us express difficult representations as simpler representations.
- Goal is approximate a function f which maps input x to a category y given parameters m (theta)
- Networks, given that often we have f(g(h(x))). The chain is denoted depth.

Deep learning

- Deep feedforward networks, same as multilayer preceptrons. Several of what we used in last lecture.
- Hidden layers since can not directly see output due to layer.
- Feedforward, information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y.
- There are no feedback connections in which outputs of the model are fed back into itself.
- When feedforward neural networks are extended to include feedback connections, they are called recurrent neural network.

Deep learning

- Finally, these networks are called neural because they are loosely inspired by neuroscience.
- Each hidden layer of the network is typically vector-valued.
- The dimensionality of these hidden layers determines the width of the model.
- Each element of the vector may be interpreted as playing a role analogous to a neuron.
- Rather than thinking of the layer as representing a single vector-to-vector function

Deep learning

- We can also think of the layer as consisting of many units that act in parallel, each representing a vector-to-scalar function.
- Each unit resembles a neuron in the sense that it receives input from many other units and computes its own activation value. Analogy with neuroscience.
- The choice of the functions $f^{(i)}(x)$ used to compute these representations is also loosely guided by neuroscientific observations about the functions that biological neurons compute.
- However, modern neural network research is guided by many mathematical and engineering disciplines, and the goal of neural networks is not to perfectly model the brain.
- It is best to think of feedforward networks as function approximation machines that are designed to achieve statistical generalization, occasionally drawing some insights from what we know about the brain, rather than as models of brain function.

Deep learning

- One way to understand feedforward networks is to begin with linear models and consider how to overcome their limitations.
- Linear models, such as logistic regression and linear regression, are appealing because they may be fit efficiently and reliably.
- Linear models also have the obvious defect that the model capacity is limited to linear functions, so the model cannot understand the interaction between any two input variables.

Deep learning

- Think of a linear model, but being extended to apply the model to not x, but f(x) where f(x) is just a mapping.

- Many initial methods did this. Choose a generic f, if it has high enough dimension it can fit the training set. But generalisation is poor.

- Often manually generate f, requires human effort. Before deep learning.

Deep learning

- Deep learning aims to learn this initial $y=f(x;\theta,w)=\phi(x;\theta)w$ mapping.
- Use θ and learn ϕ from a broad class of functions, have w for mapping.
- This is an example of a deep feedforward network with ϕ as a hidden layer.
- One of many approaches, more benefits than difficulties.

Deep learning

- Parametrise $\phi(x;\theta)$ and use optimisation to find θ for a good representation.
- Use a broad family of functions, $\phi(x;\theta)$, to make the approach generic.
- Gives us the advantage that it is simpler to fine a family of functions than precisely the right function.

XOR example

Have the notes in the slides as well

Example

- Definitely not linear
- XOR image from
- http://www.vlsiinterviewquestions.org/wp-content/uploads/2012/04/xor.jpg

- Take loss function J = 1/4 MSE(m)
- f(x,w,b) = x^T w + b, does not work
- Choose y = f(h,w,b), with h = g(x^T W + c)

- Default recommendation, use the max function.
- This activation function is the default activation function recommended for use with most feedforward neural networks

- Close to linear, but lets us move to nonlinearity.
- Thus, our network is taken as:
- $y=f(x;W,c,w,b) = w^T \max\{0,W^t x +c \} + b$
- Thus our m is composed of W,c,w,b and will need to be trained through J.

- Could work through, should start with random values
- For XOR, take W = [1 1 ; 1 1], c = [0 ; 1], w = [1, -2] and b =0
- Let walkthrough for binary input: X = [0 0 ; 0 1 ; 1 0 ; 1 1]
- If not good, could use our normal gradient decent for all m variables, W, c, w, b
- Minimise all of the derivatives.
- Bonus, have developed xor model for non binary input!

Gradient decent

- As we can see, designing and training a neural net is not much different from any other method using gradient decent.
- It requires an optimisation, cost function and a model family.
- The largest difference between linear models and neural networks
- Nonlinearity of a neural network causes most interesting loss functions to become non-convex.

Gradient decent

- This means that neural networks are usually trained by using iterative, gradient-based optimisers that merely drive the cost function to a very low value.
- Compare to linear equation solvers used to train linear regression models or the convex optimisation algorithms with global convergence guarantees used to train logistic regression or SVMs.
- Convex optimisation converges starting from any initial parameters (in theory—in practice it is very robust but can encounter numerical problems).
- Stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee, and is sensitive to the values of the initial parameters.
- For feedforward neural networks, it is important to initialise all weights to small random values. The biases may be initialised to zero or to small positive values.

Gradient decent

- For the moment, it suffices to understand that the training algorithm is almost always based on using the gradient to descend the cost function in one way or another.
- Linear models can also be trained with gradient decent
- Common when the training set is large.
- Training a neural network is not much different, but gradient computation is more complex.

Hidden units

- The design of hidden units is an extremely active area of research and does not yet have many definitive guiding theoretical principles.
- Rectified linear units (RLUs) are an excellent default choice of hidden unit.
  - Examples are the max(0,x) function
  - Many other types of hidden units are available. Essentially anything that goes from -1 to 1, tanh(x)
  - It is usually impossible to predict in advance which will work best.
  - The design process consists of trial and error, intuiting that a kind of hidden unit may work well, and then training a network with that kind of hidden unit and evaluating its performance on a validation set.

Hidden units

- Hidden units do not, as you may assume be differentiable at all input points.
- For example, the rectified linear function g (z) = max{0, z} is not differentiable at z = 0.
- This may seem like it invalidates g for use with a gradient- based learning algorithm.
- In practice, gradient descent still performs well enough for these models to be used for machine learning tasks.
- This is in part because neural network training algorithms do not usually arrive at a local minimum of the cost function, but instead merely reduce its value significantly.

Hidden units

- Because we do not expect training to actually reach a point where the gradient is 0, it is acceptable for the minima of the cost function to correspond to points with undefined gradient.
- A function is differentiable at z only if both the left derivative and the right derivative are defined and equal to each other.
- The functions used in the context of neural networks usually have defined left derivatives and defined right derivatives.
- In the case of g(z) = max{0, z}, the left derivative at z = 0 is 0 and the right derivative is 1.

Hidden units

- Software implementations of neural network training usually return one of the one-sided derivatives rather than reporting that the derivative is undefined or raising an error.
- This may be heuristically justified by observing that gradient-based optimisation on a digital computer is subject to numerical error anyway.
- The important point is that in practice one can safely disregard the non-differentiability of the hidden unit
- Deeper networks often are able to use far fewer units per layer and far fewer parameters and often generalise to the test set, but are also often harder to optimise.
- The ideal network architecture for a task must be found via experimentation guided by monitoring the validation set error.

Universal approximation

- A linear model, mapping from features to outputs via matrix multiplication, can by definition represent only linear functions.
- It has the advantage of being easy to train because many loss functions result in convex optimisation problems when applied to linear models.
- Unfortunately, we often want to learn nonlinear functions.
- At first glance, we might presume that learning a nonlinear function requires designing a specialised model family for the kind of nonlinearity we want to learn.
- Fortunately, feedforward networks with hidden layers provide a universal approximation framework.

Universal approximation

- Specifically, the universal approximation theorem states that a feedforward network with a linear output layer and at least one hidden layer with any activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.
- This also holds for the derivatives.
- Any continuous function on a closed and bounded subset of R^n is Borel measurable and therefore may be approximated by a neural network.

Universal approximation

- A neural network may also approximate any function mapping from any finite dimensional discrete space to another.
- Universal approximation theorems have also been proved for a wider class of activation functions, even RLUs
- The universal approximation theorem means that regardless of what function we are trying to learn, we know that a large MLP will be able to represent this function.
- This does not guarantee that the algorithm can learn the function, only represent.

Universal approximation

- Even if the MLP is able to represent the function, learning can fail for two different reasons.
- The optimisation algorithm used for training may not be able to find the value of the parameters that corresponds to the desired function.
- We get the same issues in neural net optimisation as any other optimisation problem, local minima, saddle points etc.
- The training algorithm might choose the wrong function due to overfitting.
- There is no free lunch, there is no universally superior machine learning algorithm.
- Feedforward networks provide a universal system for representing functions, in the sense that, given a function, there exists a feedforward network that approximates the function.
- There is no universal procedure for examining a training set of specific examples and choosing a function that will generalise to points not in the training set.

Universal approximation

- The universal approximation theorem says that there exists a network large enough to achieve any degree of accuracy we desire, but the theorem does not say how large this network will be.
- Unfortunately, in the worse case, an exponential number of hidden units may be required.
- In summary, a feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalise correctly.
- In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalisation error.

Back-propagation

- Feels more intuitive to use the cost to calculate the derivative directly.
- In essence, we use the chain rule to traverse our whole network.
- When we use a feedforward neural network to accept an input x and produce an output y, information flows forward through the network.
- The inputs x provide the initial information that then propagates up to the hidden units at each layer and finally produces y.
- During training, forward propagation can continue onward until it produces a scalar cost J(θ).

Back-propagation

- The back-propagation algorithm allows the information from the cost to then flow backwards through the network, in order to compute the gradient.
- Computing an analytical expression for the gradient is straightforward, but numerically evaluating such an expression can be computationally expensive.
- The back-propagation algorithm does so using a simple and inexpensive procedure.
- Back-propagation refers only to the method for computing the gradient, while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient.
- Furthermore, back-propagation is often misunderstood as being specific to multi-layer neural networks, but in principle it can compute derivatives of any function.

Back-propagation

- Specifically, we will describe how to compute the gradient $\nabla_x f(x, y)$ for an arbitrary function f , where x is a set of variables whose derivatives are desired, and y is an additional set of variables that are inputs to the function but whose derivatives are not required.

- In learning algorithms, the gradient we most often require is the gradient of the cost function with respect to the parameters, $\nabla J(\theta)$.

- Many machine learning tasks involve computing other derivatives, either as part of the learning process, or to analyse the learned model.

- The back-propagation algorithm can be applied to these tasks as well, and is not restricted to computing the gradient of the cost function with respect to the parameters.

Back-propagation

- The back-propagation algorithm is very simple. To compute the gradient of some scalar z with respect to one of its ancestors x in the graph, we begin by observing that the gradient with respect to z is 1.
- We can then compute the gradient with respect to each parent of z in the graph by multiplying the current gradient by the Jacobian of the operation that produced z.
- We continue multiplying by Jacobians traveling backwards through the graph in this way until we reach x.
- For any node that may be reached by going backwards from z through two or more paths, we simply sum the gradients arriving from different paths at that node.

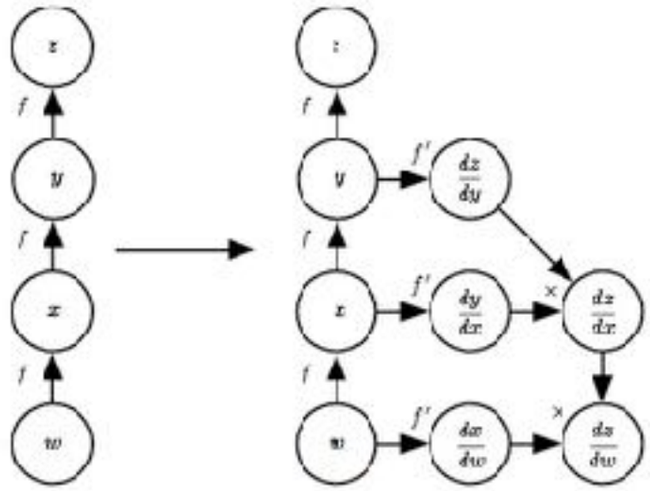


Figure 6.10. An example of the symbol-to-symbol approach to computing derivatives. In this approach, the back-propagation algorithm does not need to ever access any actual specific numeric values. Instead, it adds nodes to a computational graph describing how to compute these derivatives. A generic graph evaluation engine can later compute the derivatives for any specific numeric values. *(Left)*In this example, we begin with a graph representing $z = f(f(f(w)))$. *(Right)*We run the back-propagation algorithm, instructing it to construct the graph for the expression corresponding to $\frac{dz}{dw}$. In this example, we do not explain how the back-propagation algorithm works. The purpose is only to illustrate what the desired result is: a computational graph with a symbolic description of the derivative.

Summary

- From this point of view, the modern feedforward network is the culmination of centuries of progress on the general function approximation task.
- Following the success of back-propagation, neural network research gained popularity and reached a peak in the early 1990s. Afterwards, other machine learning techniques became more popular until the modern deep learning renaissance that began in 2006.
- The core ideas behind modern feedforward networks have not changed substantially since the 1980s. The same back-propagation algorithm and the same approaches to gradient descent are still in use.
- Most of the improvement in neural network performance from 1986 to 2015 can be attributed to two factors.
- First, larger datasets have reduced the degree to which statistical generalisation is a challenge for neural networks.
- Second, neural networks have become much larger, due to more powerful computers, and better software infrastructure. However, a small number of algorithmic changes have improved the performance of neural networks noticeably.
- One of these algorithmic changes was the replacement of mean squared error with the cross-entropy family of loss functions. Mean squared error was popular in the 1980s and 1990s, but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community.

Optimisation vs training

- Optimisation algorithms used for training of deep models differ from normal optimisation.
- Machine learning usually acts indirectly.
- In most machine learning scenarios, we care about some performance measure P, that is defined with respect to the test set.
- We therefore optimise P only indirectly. We reduce a different cost function J(θ) in the hope that doing so will improve P.
- This is in contrast to pure optimisation, where minimising J is a goal in and of itself.
- Optimisation algorithms for training deep models also typically include some specialisation on the specific structure of machine learning objective functions.

Convolutional Networks, CNN

$$s(t) = \int x(a)w(t-a)da$$

$$s(t) = (x * w)(t)$$

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

- Convolutional neural networks or CNNs.
- Specialised kind of neural network for processing data with a, grid-like topology.
- Time-series data, 1D grid taking samples at regular time intervals
- Image data, a 2D grid of pixels.
- Convolutional networks have been tremendously successful in practical applications.
- Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

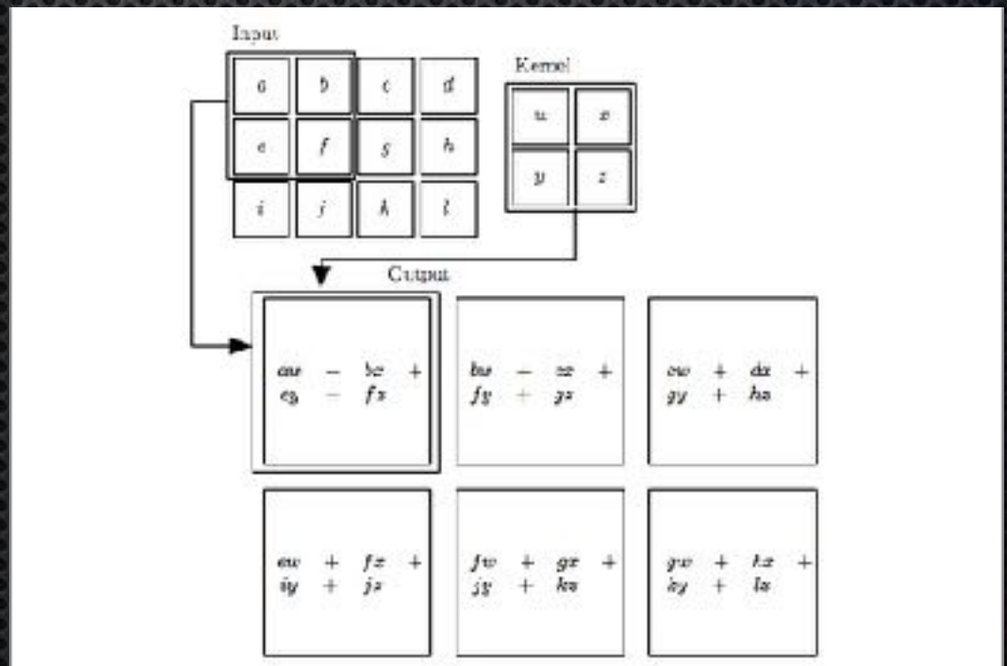


Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

CNN

- Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivalent representations.
- Moreover, convolution provides a means for working with inputs of variable size.
- Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit.
- This means every output unit interacts with every input unit.
- Convolutional networks, however, typically have sparse interactions This is accomplished by making the kernel smaller than the input.
- For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels.

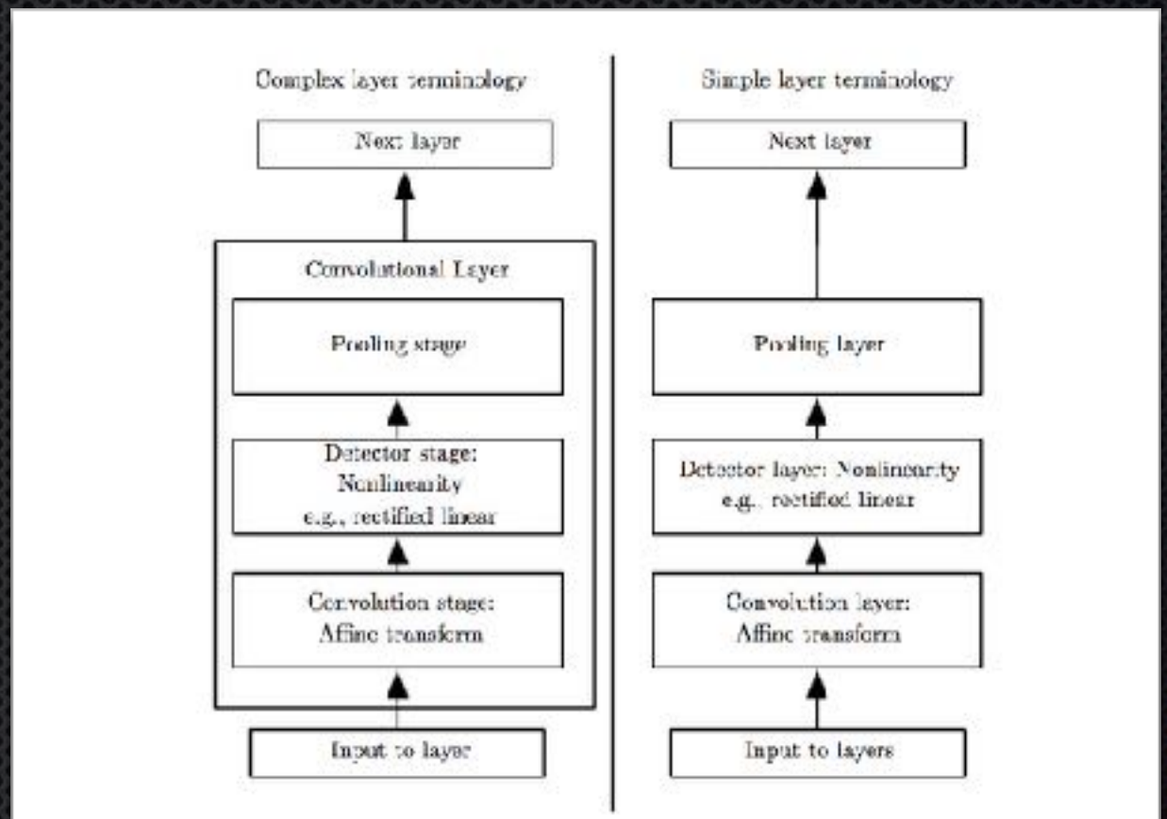


Figure 9.7: The components of a typical convolutional neural network layer. There are two commonly used sets of terminology for describing these layers. *(Left)*In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many "stages." In this terminology, there is a one-to-one mapping between kernel tensors and network layers. In this book we generally use this terminology. *(Right)*In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every "layer" has parameters.

CNN

- Store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency.
- It also means that computing the output requires fewer operations.
- These improvements in efficiency are usually quite large
- For many practical applications, it is possible to obtain good performance on the machine learning task while keeping several orders of magnitude less data.
- This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions.

Figure 9.6: *Efficiency of edge detection.* The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all of the vertically oriented edges in the input image, which can be a useful operation for object detection. Both images are 280 pixels tall. The input image is 320 pixels wide while the output image is 319 pixels wide. This transformation can be described by a convolution kernel containing two elements, and requires $319 \times 280 \times 3 = 267,960$ floating point operations (two multiplications and one addition per output pixel) to compute using convolution. To describe the same transformation with a matrix multiplication would take $320 \times 280 \times 319 \times 280$, or over eight billion, entries in the matrix, making convolution four billion times more efficient for representing this transformation. The straightforward matrix multiplication algorithm performs over sixteen billion floating point operations, making convolution roughly 60,000 times more efficient computationally. Of course, most of the entries of the matrix would be zero. If we stored only the nonzero entries of the matrix, then both matrix multiplication and convolution would require the same number of floating point operations to compute. The matrix would still need to contain $2 \times 319 \times 280 = 178,640$ entries. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small, local region across the entire input. (Photo credit: Paula Goodfellow)

Any questions?

Otherwise just approach me at any time or send me an email.
p.hallsjo.1@research.gla.ac.uk

End lecture 2

Lecture 3

- Lecture 3
- Real life applications / Current research
- Start discussing labs

Bootstrap aggregating

- Bagging, bootstrap aggregating
- Reduces generalisation errors. Combine several methods.
- Train several models separately, then combine for final result.

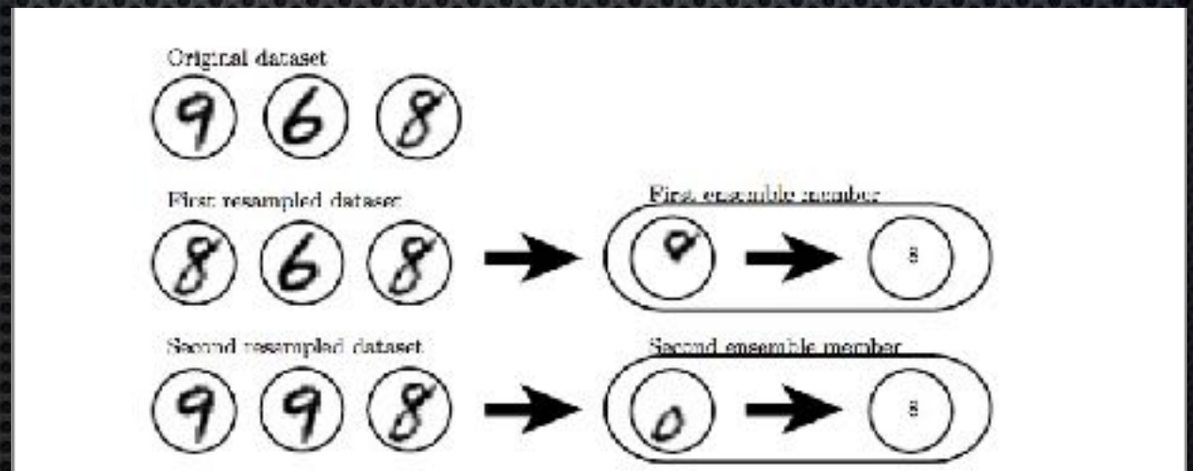


Figure 7.5: A cartoon depiction of how bagging works. Suppose we train an 8 detector on the dataset depicted above, containing an 8, a 6 and a 9. Suppose we make two different resampled datasets. The bagging training procedure is to construct each of these datasets by sampling with replacement. The first dataset omits the 9 and repeats the 8. On this dataset, the detector learns that a loop on top of the digit corresponds to an 8. On the second dataset, we repeat the 9 and omit the 6. In this case, the detector learns that a loop on the bottom of the digit corresponds to an 8. Each of these individual classification rules is brittle, but if we average their output then the detector is robust, achieving maximal confidence only when both loops of the 8 are present.

Adversarial training

- In many cases, neural networks have begun to reach human performance.
- Wonder whether these models have obtained a true human-level understanding of these tasks.
- In order to probe the level of understanding a network has of the underlying task, we can search for examples that the model misclassifies.
- Neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed by using an optimisation procedure, seen in figure.
- Adversarial examples have many implications, for example, in computer security.
- However, they are interesting in the context of regularisation because one can reduce the error rate on the original i.i.d. test set via adversarial training.
- Adversarial training discourages this highly sensitive locally linear behaviour by encouraging the network to be locally constant in the neighbourhood of the training data.

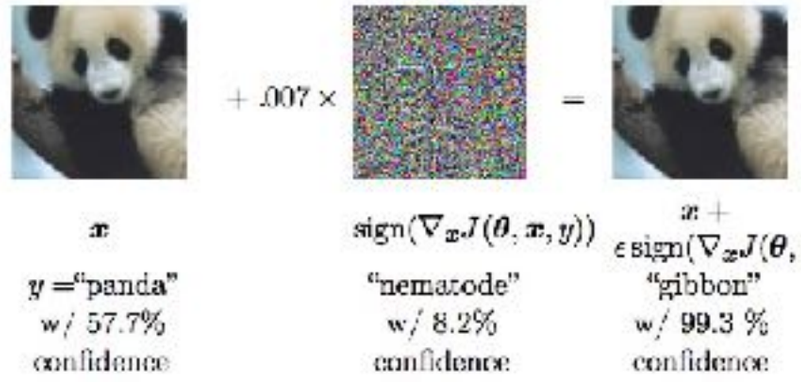


Figure 7.8: A demonstration of adversarial example generation applied to GoogLeNet (Szegedy *et al.*, 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Reproduced with permission from Goodfellow *et al.* (2014b).

Online applications

- Give some specific problems and solutions, computer vision.
- How do we deal with images? Pixel by pixel. Neural nets give images.
- Great link
- http://playground.tensorflow.org/

Online applications

- ConvNetJS is a Javascript library for training Deep Learning models (Neural Networks) entirely in your browser. Open a tab and you're training. No software requirements, no compilers, no installations, no GPUs, no sweat.

- https://cs.stanford.edu/people/karpathy/convnetjs/

Online applications

- How do we apply our knowledge to time series? One good tutorial for how to do it in tensorflow.

- https://mapr.com/blog/deep-learning-tensorflow/
- https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
- https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/

Online applications

- https://www.kaggle.com/patha325/titanic-data-science-solutions
- https://www.kaggle.com/patha325

https://youtu.be/I_owi316cE8

Predicting human pose in 3D using any camera

https://youtu.be/aircAruvnKk
Based on
http://neuralnetworksanddeeplearning.com
And code
https://github.com/mnielsen/neural-networks-and-deep-learning

Youtube series, discussing ML.

https://youtu.be/qv6UVOQ0F44

Apply neutral nets to Super Mario

Lab information

- Lab1
- Perceptron Simple and elegant use of a ML algorithm. Getting into writing own implementations in python. Using sklearn. Quite free form using the tools and understanding them.
- In tensorflow, use image analysis and understand how difficult noise can be.
- Lab2
- Free form translation. Build a translator and translate “My hovercraft is full of eels” and “Can you direct me to the station?” from English to Hungarian or to German. Use any resources, can for instance find a dictionary at: http://www.manythings.org/anki/
- I will add details to help you though the latter, there are many resources online.

Further resources

- http://www.ppe.gla.ac.uk/~phallsjo/files/CardiffSTFC/
- http://tmva.sourceforge.net - Particle physics
- https://www.kaggle.com
- https://www.openml.org
- https://www.tensorflow.org/tutorials/
- http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/
- https://medium.freecodecamp.org/every-single-machine-learning-course-on-the-internet-ranked-by-your-reviews-3c4a7b8026c0
- https://github.com/josephmisiti/awesome-machine-learning
- https://github.com/josephmisiti/awesome-machine-learning/blob/master/books.md

Further resources

- http://cs231n.github.io/classification/ - Stanford lecture
- http://www.mit.edu/~rakhlin/6.883/ - MIT Online Methods in Machine Learning

- **EDX:**
- https://www.edx.org/course/applied-machine-learning-microsoft-dat203-3x-3
- https://www.edx.org/course/principles-machine-learning-microsoft-dat203-2x-5

Further resources

- **Coursera:**
- https://www.coursera.org/learn/machine-learning/home/welcome
- https://www.coursera.org/learn/convolutional-neural-networks/home/welcome
- https://www.coursera.org/learn/neural-networks/home/welcome
- https://www.coursera.org/learn/neural-networks-deep-learning/home/welcome
- https://www.coursera.org/learn/python-machine-learning/home/welcome

Further resources

- **CERN lecture series**
- https://indico.cern.ch/category/9320/
- **Excellent talk from google Deep Mind**
- https://indico.cern.ch/event/673350/
- https://danilorezende.com
- **Deep Mind publications**
- https://deepmind.com/research/publications/

Any questions?

Otherwise just approach me at any time or send me an email.
p.hallsjo.1@research.gla.ac.uk

End lecture 3