Overview of IN3050

1. Search and Optimization

1.1 Introduction

- **Optimization:** A numerical representation *x* for all possible solutions to the problem.
- **Continuous optimization:** Concerned with finding the maxima and minima of functions, possibly subject to constraints. E.g. mechanics, economics, and control engineering.
- **Discrete optimization:** Activity of looking thoroughly to find an item with specified properties among a collection of items. E.g. travelling salesman problem, chip design and timetabling.

1.2 Exhaustive search

- Pure exploration
- Test all possible solutions and pick the best – guaranteed to find the optimal solution.
- Only works for discrete problems but can be apx. in continuous problems with grid search or sample the space randomly **N** times.

1.3 Greedy search and hill climbing

- **Greedy search:** Only generates and evaluates a single solution. Makes several locally optimal choices, hoping the result will be near a global optimum. Does not look ahead much further than the closest next option, leading to potential pitfalls like getting "trapped" by having to make long trips at the end of the itinerary.
- **Hill climbing:** Pick a solution as the current best (e.g. a random solution). Compare it to neighbour solution(s) – if better, replace with current best.

1.4 Exploitation and exploration

- **Exploration:** Trying completely new solutions.
- **Exploitation:** Trying to improve the current best solution by **local search**.
- The best strategy to find the global optimum is to combine exploration and exploitation.
- Greedy search, hill climbing, and gradient ascent/descent can **only find local optima.**

1.5 Simulated annealing

Algorithm:

- Set an initial temperature T (high T leads to high exploration, low T leads to high exploitation)
- Pick an initial solution.
- Repeat:
  - Pick a solution neighbouring the current solution.
    - If the new one is better, keep it.

        - Otherwise, keep a new one with probability *p* (*p* depends on the difference in quality and the temperature).
    - Reduce T

1.6 Continuous optimization and gradient descent

- **Gradient:** Tells us in which direction *f(x)* increases the most.
- **Gradient ascent:** $\mathrm{x}^{(\mathrm{k}+1)} = \mathrm{x}^{(\mathrm{k})} + \gamma \nabla f\left(x^{(k)}\right)$
- **Gradient descent:** $\mathrm{x}^{(\mathrm{k}+1)} = \mathrm{x}^{(\mathrm{k})} - \gamma \nabla f\left(x^{(k)}\right)$

Algorithm:

- Start with a point (random).
- Repeat:
    - Determine a descent direction.
    - Choose a step (lr).
    - Update.
- Until stopping criterion is satisfied.

- A **large** learning rate may accelerate convergence but risks overshooting the minimum, causing instability. A **small** learning rate ensures more reliable convergence but can be slow and gets easily stuck in local minima.

2. Evolutionary Algorithms

2.1 Introduction to evolution

- **Evolutionary Computing (EC):** Mimic the biological evolution to optimize solutions to a wide variety of complex problems. In every new generation a new set of solutions is created using bits and pieces of the fittest of the old.

2.2 Evolutionary algorithms

Algorithm:

- Initialise population with **random** candidate solutions.
- **Evaluate** each candidate.
- Repeat until termination condition is satisfied:
    - **Select** parents.
    - **Recombine** pairs of parents.
    - **Mutate** the resulting offspring.
    - **Evaluate** new candidates.
    - **Select** individuals for the **next generation**.

Two pillars of evolution:

- **Push towards novelty:** Increasing population diversity by the genetic operator's mutation and recombination.
- **Push towards quality:** Decreasing population diversity by selection of parents and survivors.

Representation:

- **Gene:** One **element** of the array.

- **Locus:** The **position** of a gene (*index*).
- **Allele:** What **values** a gene can have.
- **Genotype:** A **set** of gene values.
- **Phenotype:** What could be **built/developed** based on the genotype.

2.3 Components of an evolutionary algorithm

- **Evaluation (fitness) function:** Represents the task to solve and enables selection (provides a basis for comparison). Assigns a single real valued fitness to each phenotype.

- **Population:** The candidate solutions (individuals) of the problem. Population is the basic unit of evolution – it is the population that is evolving, not the individuals. Selection operators act on population level, variation operators act on individual levels.

- **Selection mechanisms:** Identify individuals to become parents and to survive. Pushes population towards higher fitness. High quality solutions more likely to be selected than low quality, but not guaranteed. This **stochastic** nature can aid escape from local optima.

- **Variation operators:** Generate new candidate solutions. Usually divided into two types based on their arity (number of inputs):
  - **Arity 1:** Mutation operators.
  - **Arity > 1:** Recombination operators.
  - **Arity = 2:** Crossover.
  - **Arity > 2:** Seldom used in EC.

- **Mutation:** Cause small, random **variance** to a genotype. Element of randomness is essential and differentiates it from other unary heuristic operators. E.g. 1111 to 1101.

- **Recombination:** Merges information from parents into offspring (stochastically) with hope that some offspring are better by combining elements of genotypes that lead to good traits. E.g. one parent is 5555 and the other is 6666 – offspring is 5656.
- **Termination:** Termination condition is checked every generation, e.g. ideal fitness, max. number of generations and min. level of diversity.

2.4 Binary, integer, and real-valued representations

- **Binary representation:** Genotype consists of a string of binary digits.
  - **Mutation:** Alter each gene independently with a probability, the probability is called the mutation rate.
  - **1-point crossover:** Choose random point on two parents, split parents at this crossover point and create children by exchanging tails.
  - **n-point crossover:** Choose *n* random crossover points, split along those points, glue parts alternating between parents.

    - **Uniform crossover:** Assign 'heads' to one parent and 'tails' to the other. Flip a coin for each gene of the first child. Make an inverse copy of the gene for the second child. Breaks more links in the genome.

- **Integer representation:** Naturally, some problems have integer variables. Other take categorical values – can convert to integers.
    - **n-point crossover**
    - **Uniform crossover**
    - **Creep:** Adding a small (positive or negative) value to each gene with probability *p*.
    - **Random resetting (esp. categorical variables):** A new value is chosen at random with a certain probability.

- **Real-valued or floating-point representation**
    - **Uniform mutation:** New value is drawn randomly from a user specified lower boundary and upper boundary.
    - **Non-uniform mutation:** Most common method is to add random deviate to each variable separately, taken from **Gaussian distribution** and then curtail to range.
    - **Crossover operators:**
        - **Discrete recombination:** Each allele in offspring comes from one of its parents with equal probability – could use **n-point** or **uniform**.
        - **Intermediate recombination:** Exploits idea of creating children between parents.
    - **Simple arithmetic crossover:** Pick a random gene (k) and after this point mix values. E.g. parent one is 4444 and parent two is 5555 – child one is 4455 and child two 5544.

2.5 Permutation and tree-based representations

- **Permutation representations:** Useful in sequencing problems. Task is arranging some objects in a certain order. E.g. production scheduling and TSP. If there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once.

    - **Mutation:** Mutating a single gene destroys the permutation, therefore must change at least two values. *P* reflects that some operator is applied once to the whole string rather than individually in each position.
        - **Swap mutation:** Pick two alleles at random and swap their positions. E.g. [1,2,3,4] to [1,3,2,4].
        - **Insert mutation:** Pick two allele values at random, move the second to follow the first, shifting the rest along to accommodate. E.g. [1, 2, 3, 4, 5, 6, 7, 8, 9] to [1, 2, 5, 3, 4, 6, 7, 8, 9].
        - **Scramble mutation:** Pick a subset of genes at random, randomly rearrange the alleles in those positions. E.g. [1, **2, 3, 4, 5**, 6, 7, 8. 9] to [1, **3, 5, 4, 2,** 6, 7, 8, 9].

    - **Inversion mutation:** Pick two alleles at random then invert the substring between them. E.g. [1, **2, 3, 4, 5**, 6, 7, 8. 9] to [1, 5, 4, 3, 2, 6, 7, 8, 9].

- **Crossover operators:** Normal crossover operators will often lead to inadmissible solutions. We use **specialised operators** which focus on combining order or adjacency information from the two parents.
    - **Conserving order:** Important for problems order of elements decide performance (e.g. production scheduling).
        - **Order Crossover:** Idea is to preserve relative order that elements occur.
          Informal procedure:
            - Choose an arbitrary part from the first parent.
            - Copy this part to the first child.
            - Copy the numbers that are not in the first part, to the first child:
                - Start right from cut point of the copied part,
                - Using the **order** of the second parent
                - And wrapping around at the end.
            - Analogous for the second child, with parent roles reversed.
        - **Cycle Crossover:** Each allele comes from one parent together with its position.
          Informal procedure:
            - Make a cycle of alleles from P1 in the following way:
                a) Start with the first allele of P1.
                b) Look at the allele at the same position in P2.
                c) Go to the position with the same allele in P1.
                d) Add this allele to the cycle.
                e) Repeat step b through d until you arrive at the first allele of P1.
            - Put the alleles of the cycle in the first child on the positions they have in the first parent.
            - Take next cycle from second parent.
            - Identify cycles, copy alternate cycles into offspring.

    - **Conserving adjacency:** Important for problems where adjacency between elements decides quality (e.g. TSP).
        - **Partially Mapped Crossover:**
          Algorithm:
            - Choose random segment and copy it from P1.
            - Starting from the first crossover point look for elements in that segment of P2 that have not been copied.
            - For each of these *I* look in the offspring to see what element *j* has been copied in its place from P1

        - Place *l* into the position occupied in P2, since we know that we will not be putting *j* there (as is already in offspring).
        - If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *l* in the position occupied by *k* in P2.
        - Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2.
    - **Edge recombination:** Works by constructing a table listing which edges are present in the two parents, if an edge is common to both, mark it a +.
      Informal procedure: once edge table is constructed:
        - Pick an initial element, entry, at random and put it in the offspring.
        - Set the variable current element = entry.
        - Remove all references to current element from the table.
        - Examine list for current element:
            - If there is a common edge, pick that one to be the next element.
            - Otherwise, pick the entry in the list which itself has the shortest list.
            - Ties are split at random.
            - In the case of reaching an empty list:
                - A new element is chosen at random.

2.6 Fitness, Selection and Population Management

- **Selection:** Second fundamental force for evolutionary systems (*variation/mutation is the first*). Selection operators and preserving diversity. Selection can occur in two places: **Parent selection** (selects mating pairs) and **survivor selection** (replaces population).
- **Selection pressure:** As selection pressure increases, fitter solutions are more likely to survive, or be chosen as parents.
- **Parent selection:**
    - **Fitness-Proportionate Selection (FPS):** Also called roulette wheel selection, e.g. fitness(A) = 3, fitness(B) = 1 and fitness(C) = 2, then A has is 50%, B is 17% and C is 33%. Relies on global population statistics, could be bottleneck when population is very large. Can lead to premature convergence if there are individuals with much higher fitness, since they can dominate the wheel.
    - **Tournament Selection:** Pick *k* members at random then select the best of these. Repeat to select more individuals. Either stochastic (wins with probability *p*) or deterministic.
    - **Rank-based selection:** Individuals are ranked based on their fitness, and selection is based on these ranks rather than their actual fitness

values. Assigns selection probabilities according to rank, which helps mitigate what we see in FSP.

- **Elitism:** Always keep at least one copy of the N fittest solutions so far.

2.7 Diversity preservation

- **Multimodality:** Want to identify several possible peaks because they might be different good ways to solve the problem. Need methods to preserve diversity instead of converging to one peak.
- **Preserving diversity:** Explicit vs. implicit
  - **Implicit approaches:**
    - Impose an equivalent of **geographical separation**
    - Impose an equivalent of **speciation**
    - **Automatic speciation:** Either only mate with genotypically/phenotypically similar members or add species tags to genotype. Initially randomly set. When selecting partner for recombination, only pick members with a good match.
    - **Island model**: Periodic migration of individual solutions between populations. Run multiple populations in parallel. After several generations, exchange individuals with neighbours. Repeat until ending criteria is met.
  - **Explicit approaches:**
    - Make **similar individuals compete** for resources (**fitness**)
    - Make **similar individuals compete** for each other for **survival.**
    - **Fitness sharing:** Restrict the number of individuals within a given niche by "sharing" their problem. Need to set the size of the niche in either genotype or phenotype space.
    - **Crowding:** New individuals replace similar individuals. Randomly shuffle and pair parents, produce two offspring. Each offspring competes with their nearest parent for survival (using a distance measure). Result is an even distribution among niches.

2.8 Hybridization

- **Why hybridise?**
  Looking to improve on existing techniques (non-EA) and might be looking at improving EA search for good solutions.
- **What is a Memetic algorithm?**
  The combination of EA with **local search operators** that work within the EA loop has been termed "memetic algorithms". Term also applies to EAs that use instance specific knowledge. Memetic algorithms have shown to be orders of magnitude faster and more accurate than EAs on some problems and are the "state of the art" on many problems.
- **Local search**
  - **Lamarckian: Acquired traits are inherited.** This type of direct inheritance of acquired traits is not possible, according to modern evolutionary theory.
  - **Baldwinian: Learned behaviors or adaptations acquired during an individual's lifetime could influence the direction of evolution.**

- **Where to hybridise:** Initial population, crossover stage, mutation stage and selection stage.

2.9 Multi-objective optimization

- **Multi-objective optimisation problems:** Wide range of problems can be categorised by the presence of a number of *n* **possibly conflicting objectives:**
  - **Buying a car:** Speed vs. price vs. reliability
  - **Engineering design:** Lightness vs. strength
  - **Inspecting infrastructure:** Energy usage vs. completeness
  - **Two problems:**
    - Finding a set of good solutions
    - Choice of best for the particular application

- **Two approaches to multi-objective optimisation:**
  - **Weighted sum (scalarisation):**
    - Transform into a single objective optimisation method
    - Compute a weighted sum of the different objectives
  - **A set of multi-objective solutions (Pareto Front):**
    - The population-based nature of EAs used to simultaneously search for a set of point apx. Pareto Front.
    - **Dominance relation:** A solution X dominates solution y if:
      - X is better than Y in at least one objective.
      - X is not worse than Y in all other objectives.
    - **Pareto optimality:** Solution X is **non-dominated** among a set of solutions Q if no solution from Q dominates X. A set of non-dominated solutions from the entire feasible solution space is the **Pareto set,** or **Pareto front,** its members Pareto-optimal solutions.
  - **Goal of multi-objective optimisers:** Find a set of non-dominated solutions (approximation set) following the criteria of:
    - **Convergence** (as close as possible to the Pareto-optimal front)
    - **Diversity:** Spread, distribution.
    - **EC approach:**
      1. Way of assigning fitness and **selecting individuals** (usually based on dominance)
      2. Preservation of a **diverse set of points** (similarities to multi-modal problems)
      3. Remembering all the **non-dominated points** you have seen (usually using elitism or an archive)
  - **NSGA-II:** Diverse population of solutions, using a non-dominated sorting approach to classify solutions based on dominance criteria, and a crowding distance mechanism to promote solution diversity. Iteratively selects and breeds individuals using crossover and mutation to improve the population across generations towards a set of Pareto-optimal solutions.

3. Supervised Learning

Introduction to supervised learning

- **ML:** Study of computer algorithms that improve automatically through experience.
- **Generalization:** Provides sensible outputs for inputs not encountered during training.
- **Formalizing the Learning Problem:**
    - Improve over task T
    - With respect to performance measure P
    - Based on experience
- **Supervised learning:** Predict the label on unseen items.
- **Classification:** Categorize data points (spam/not spam)
- **Regression:** Predicts continuous values (housing prices, weather forecasts),

Classification and features:

- **Labels:** A finite set of labels.
- **Classifier:** A mapping which maps each object to a unique label.
- **Binary classification:** Two classes (the perceptron).
- **Multiclass classification:** Three or more classes (decision tree).
- **Features:** To be able to classify objects, we have to make a representation of them. Decide on a set of attributes/features we can observe, decide on the set of possible values for each attribute, extract the values of the attributes for each object.

Supervised learning:

- **Structure:** Well-defined set of observations (possible inputs), set of label values (possible output values).
- **Goal:** Determine a mapping from O to L.
- **Training:** A training set of examples from O x L.
- **Training phase:** Learn mapping from the training set. Ideally, learn mapping from input to output on the training data. We the mapping is learned it can be used to predict values for new items.
- **Feature extraction:** We first decide on the features, their possible values, extract them from the observations and replace each observation with its features.
- **Feature types:** Some algorithms can only take numerical features (*kNN, perceptron)* and some take categorical features (decision trees).
- **Training set:** Train on training set.
- **Test set:** Predict labels on the test set (after removing the labels). Compare the prediction to the given labels. When development training is done.
- **Development set (dev-test set):** Repeated testing during development. Tune the hyperparameters used during training.
- **Confusion matrix:** Run the classifier on the labelled test set and compare the predicted labels to the example labels and count.
- **Accuracy:** Ratio of correctly classified instances to the total number of instances. (TP + TN) / N

- **Precision:** Ratio of correctly predicted positive instances to all predicted positive instance. TP / (TP + FP)
- **Recall (sensitivity):** Ratio of correctly predicted negative instances to all actual negative instances. TP / (TP + FN)
- **F1-score:** Harmonic mean of precision and recall, combining both metrics into a single score. 2 * (precision * recall) / (precision + recall)

kNN

- **Algorithm:**
  1. Calculate the distance to all the training instances
  2. Pick the *k* nearest ones
  3. Choose the majority class for this set
- **Distance:** Points that are close together in feature space are similar. Normally Euclidean distance (L2).
- **Small *k*:** Good for training data but danger of overfitting.
- **Larger *k*:** More general.
- **Properties of *k*NN:** Instance-based, no real training. Inefficient in predicting the label of new instances, since it must consider all the training data. Only one parameter (*k*), the distance measure may influence the result. The **scaling** of the axes might influence the result.
- **More classes:** Binary classifier with odd *k* always reach a decision. With more than 2 classes, there might be a draw. Possible ways out are back-off to k-1, etc. until there is a majority class or weight points by inverse distance from the target, take the weighted max.

Scaling

- **Use case:** Scaling adjusts the range of feature values. It is important because different features can have different units of measurement and scales. Scaling can be important for efficiency for some classifiers.
- **Max-min-scaling:** Scales features between a specified minimum and maximum value (usually 0 and 1)
- **Standard scaler (normalizer):** Subtracts the mean of each feature and then divides them by the SD.
- **Test set:** Not good idea to normalize the dataset before splitting into training and testing. Use the training data to determine the scaler/normalizer. Scale the training data before training. Whenever you apply the system: scale the data before they are given to predict, using the same scaler as on the training data. A

The Perceptron Algorithm

- **The perceptron:** A set of inputs $x_1, x_2, \ldots, x_m$ and a set of weights $w_1, w_2, \ldots w_m$. We multiply each respective weight with each the input and sum it up in an adder. The last step is typically an activation function (step function).
- **Update weights:** $w_i = w_i - lr(y - t)x_1$

- **Linear separability:** A set is linearly separable if there is a straight line in the feature plane such that all points in one class fall on one side and all points in the other class fall at the other side. For more than two features, this generalizes to a hyper-plane.
- **Linear classifier:** A linear classifier will always propose a linear decision boundary (point, line, plane, hyper-plane). The perceptron is a **linear classifier**.
- **Perceptron Convergence Theorem:** If the training set is linearly separable, the perceptron will (sooner or later) find a linear decision boundary and stop updating unless *lr* is too large.

Linear Regression and Classification

- **Regression:** Assign a numerical value to an observation (e.g. temperature tomorrow). The target set is real numbers. The goal is to predict a $y_j = f(x_j)$ which is close to the true t_j.
- **Inductive bias:** To learn from data, you must have idea regarding how the data is distributed. You choose a model and try to find parameters which makes the model fit the training data well. Models carry with them inductive biases, e.g. linear regression can only learn straight lines and perceptron can only learn linear decision boundaries.
- **Hyper-plane:** $f(x) = w_0 + w_1x_1 + w_2x_2 + \cdots w_mx_m$ describes a hyper-plane.
- **Mean Square Error (loss function):** We need a way to tell whether a line is a good fit to the data, and whether one line is better than another. The goal is to minimize this error by finding the appropriate weights. MSE is convex, there is only one global minimum and no problem with local optima. $\frac{1}{2}\Sigma_{j=1}^{N}\left(t_j - y_i\right)^2$

Gradient descent

Algorithm:

- Start with x_0
- Iteratively find $x_1, x_2, \ldots, x_i$ with decreasing $f(x_i)$ by setting $x_{i+1} = x_i - lrf'(x_i)$

Logistic Regression

- **Introduction:** A classifier and not for numerical regression. It is the best purely linear classifier. Probability-based. There are two classes t = 1 and t = 0. For an observation, we wonder how probable is it that this observation belongs to class 1, and how probable is it that it belongs to class 0?
- **The sigmoid curve:** An approximation to the ideal decision boundary. Mistakes further from the decision boundary are punished harder. Squeezes/maps anything to (0, 1) and is monotone.
- **The logistic function:** $\frac{1}{1+e^{(-x)}}$. We apply the logistic function to the sum of weights multiplied by inputs. This looks like this: $\frac{1}{1+e^{(-w*x)}}$
- **The derivative of the logistic function:** $y(1-y)$
- **The activation of logistic regression:** The **logistic function**. The perceptron has the **step function** and linear regression has **identity**.
- **Cross-Entropy Loss:** In machine learning we decide on an objective for the training. We can do that in terms of a loss function. The goal of the training is to minimize the loss function (MSE for linear regression). We can choose between various loss

functions. The choice is partly determined by the learner. **For logistic regression we choose Cross-Entropy Loss**. We want to maximise the joint probability of all the predictions we make.

Training the Logistic Regression Classifier

When training our supervised ML model, we need a way to measure how well our model's predictions match up with the true labels. This measurement is done using a loss function. For classification problems, we use **Cross-Entropy loss**. This function compares the predicted probabilities that the model outputs for each class (the higher the probability, the more confident the model is that this is the right class) with the actual class labels that should be predicted. A good model will have a high predicted probability for the correct class, leading to a low cross-entropy loss.

The goal of training the model is to make this loss as small as possible, which means the model's predictions get closer and closer to the true labels. Now that we have our loss, we need a way to improve our model. This is where "gradient descent" comes in. It's not a measure of the model's performance, but rather a method to improve it. The gradient descent minimizes the loss. The Cross-Entropy Loss is convex, so we're not in a local minimum and know which way to go.

Variants of gradient descent

- **Batch Training:**
    - Calculate the loss for the whole training set, and the gradient for this.
    - Make **one** move in the **correct direction**.
    - Repeat (an epoch).

  Can be slow.
- **Stochastic Gradient Descent:**
    - Pick one item.
    - Calculate the loss for this item.
    - Calculate the gradient for this item and move in the opposite direction.

  Each move does not have to be in towards the direction of the gradient for the whole set, but the overall effect may be good. Can be faster.
- **Mini-batch Training:**
    - Pick **a subset** of the training set of a certain size.
    - Calculate the loss for this subset.
    - Make one move in the direction opposite of this gradient.
    - Repeat (an epoch).

  A good compromise between the two extremes – the two other are subcases of this.

Multi-class classification

So far, many algorithms and examples have been binary (yes/no, 1/0). In the real world, many classification tasks are multi-class: To each observation *x* choose one label from a finite set C.

1-of-N (One Hot Encoding)
Labels might be categorical: 'apple', 'tomato', 'dog', 'horse'. The algorithms demand numerical attributes. We might be tempted to say that 'apple' = 1, 'tomato' = 2, 'dog' = 3 and 'horse' = 4. This isn't a good idea, because we then say that tomato is between apple and dog. Better to encode them like e.g. 'apple' = (1,0, 0, 0, 0). Both the target and the predicted value are vectors.

Multi-label classifier:

- Make *n* different classifiers, one for each class
- For classifier *j*:
    - Consider class *j* the positive class
    - All other items in the negative class
    - Train a classifier f_j
- Application: Assign a label c_j to an item if and only if it is classified as positive by f_j.

For the multi-class task, the goal is to predict one of the labels. In multi-label: For a given items, for each of the labels, to decide if it is true or not. How can a multi-label classifier be turned into a multi-class classifier?

Two approaches:

- **One vs. Rest classifier (one vs. all):**
    - Start like the multi-label classifier: Make one classifier for each class.
    - If each classifier predicts a score, compare the scores for the classes.
    - Choose the class with the highest score.
    - E.g. log. reg.:
        - Probability of being red: 0.8
        - Probability of being blue: 0.7
        - Choose red.

- **Multinomial Logistic Regression (softmax regression):**
    - Statistical method used for predicting the categorization of an input variable into multiple mutually exclusive classes.
    - Extends the concept of logistic regression used for binary classification.
    - Find the best fit parameters to predict the probability of an input belonging to each class.
    - Probabilities for all classes are calculated simultaneously using the softmax function, which normalizes the outputs into a valid probability distribution.
    - The highest probability class is then predicted as the outcome.

4. Feed-forward neural networks and backpropagation

Neural networks

A NN is a machine learning program that makes decisions in a manner similar to the human brain, by using processes that mimic the way biological neurons work together to identify phenomena, weigh options and arrive at conclusions.

Feed-forward neural networks (Muli-layer Perceptron)

- An input layer
- An output (final) layer: The predictions
- One or more hidden layers (not exposed to input/output, considered the computational engine of the NN),
- Connections from nodes in one layer to nodes in the next layer (left to right)
- The connections are weighted

Going forward

- One input node for each feature/dimension in input vector: $x_1, x_2, \dots, x_m$.
- In addition, an input bias node $x_0 = -1$ or any other non-zero value.
- The input values are multiplied with the weights and summed into each hidden node.
- There is some processing in the hidden node (activation function: ReLU, logistic (Sigmoid)).
- The output values of the hidden nodes are fed to the next layer.

One hidden node

1. First sum of the weighted inputs: $z = \Sigma_{i=0}^{m} w_i x_i = w * x$
2. Then the result is run through an activation function, g, to produce $g(z) = g(w * x)$
3. Then $y = g(z) = \log(z) = \frac{1}{1+e^{-w*x}}$

Output layer

- Several possibilities, depending on the task, including:
  - **Regression (can predict non-linear functions)**
    - One output node.
    - No activation function in the output layer (activation function is the identity function).
  - **Binary classification (can produce non-linear decision boundaries)**
    - One output node.
    - Logistic function in the output layer.
    - Similar to logistic regression.
  - **Multi-label classification**
    - Several output nodes.
    - Logistic activation function.
    - Can be made multi-class classification by one vs. rest.
  - **Multi-class classification**
    - Several output nodes.
    - Sum the weighted inputs at each nodes.
    - The sums are brought together in the softmax.
- From the last layer to the output layer is like the same tasks without multiple layers!

Learning by backpropagation

Training

- Given a set of training instances $\{(x_1, t_1), (x_2, t_2), \dots, (x_n, t_n)\}$
- **Forward:** Run them forward and get predictions $\{y_1, y_2, \dots, y_n\}$

- **Backward:** Use a suitable loss function and compare these to the target values $\{t_1, t_2, \dots, t_n\}$. Apply gradient descent to update the weights (partial derivatives). We take the derivate of the loss function and use the chain rule to derive the different steps.

How do we update the weights?

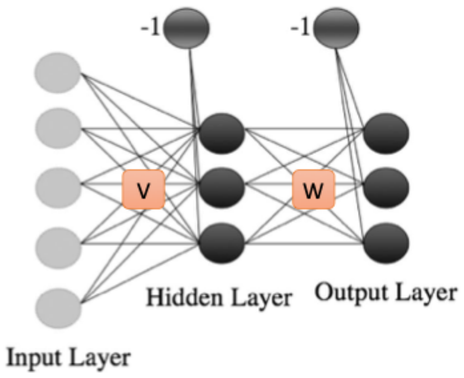
- **Last layer:** Easy – like the same problems for linear regression or logistic regression without a hidden layer.
- **The first layer:** Backpropagation. Correction on first layer may affect all output nodes – interdependencies between all the nodes.

Backpropagation

- **Activation functions:**
    - **Hidden layer:** *g*
    - **Output layer:** *f*
- We consider **SGD** where we update for one input $x = (x_1, x_2, \dots, x_m)$. We will run one input through the network and update for only one.

Forward

1. Add bias and send: $x = x_0, x_1, \dots, x_m$
2. Through the first layer to get: $h_j = \Sigma_{i=0}^{m} x_i v_{i,j}$
3. Apply activation to get: $a = g(h)$
4. Add bias and send: $a = a_0, a_1, a_2, \dots, a_k$
5. Through the second layer to get: $z_j = \Sigma_{i=0}^{k} a_i w_{i,j}$
6. Apply activation function to get: $y = f(z) = (y_1, y_2, \dots, y_3)$ where $y_j = f(z_j)$.



Backward: Regression and updating the last layer

We consider only one node. We should not update the weights in the last layer until we have found out how to update the loss in all the other layers. We have to calculate the delta term for all layers to update first.

Putting it together: the Algorithm (ESSENTIAL)

- We use this to calculate the delta term at the hidden nodes, from the delta term(s) and the hidden layer and the weights at the connections:
    - $d_h = (d_{final}) * (w_{h,final}) * a_h * (1 - a_h)$
    - a_h is the output from hidden node and the derivative of the activation function at the hidden layer.
    - We now have the delta terms, time to update the weights by the deltas
    - $w_{l,r} = w_{l,r} - lr * d_r * output_l$ (I've generalized this so that the formula can be applied to update weights between the hidden layer and the final layer and the hidden layer and the input layer).

Binary classification:

- Only difference to regression is the logistic activation function.

- SE: The derivative of this is $y(1-y)$, we get $delta = (y-t)y(1-y)$
- The rest is as for regression
- For logistic regression, we use Cross-Entropy Loss.
- Since we use Cross-Entropy Loss (logistic activation), we use the derivative of this: $(y-t)$

Multi-label classification:

- Several output nodes
- Logistic activation function
- SE: The derivative of this is $y(1-y)$, we get $delta_{final} = (y-t)y(1-y)$
- We compute the delta term at each output node.
- **First:**
    - The delta term on the hidden layer will take the delta term of each of the output nodes and multiply them with the weight between them. In other words, sum of the delta at output weighted by the connections between them.
- The rest as for the others.

Multi-label/class classification:

- For the multi-class task, the goal is to predict one of the labels.
- In multi-label: For a given items, for each of the labels, to decide if it is true or not.
- **Representation of multi-class classification:** (0, 0, 1, 0).
- **Representation of multi-label classification:** (1, 0, 1, 0).
- **Model:** Logistic regression and MSE-loss.
- Same learning by backpropagation.
- **Difference in application:**
    - Multi-class (one vs. rest): argmax
    - Multi-label: Prediction for each class on the basis of the y is greater or smaller than the threshold value (typically 0.5).

Multinomial Logistic Regression:

- Also known as the softmax-classifier
- A function that "softens" logits to probabilities, typically used in the final layer of a network for multi-class problems.
- **Activation function:** Produced by exponential + softmax.

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}$$

- **Loss function:** Cross-entropy loss.
- The multinomial logistic regression, or softmax classifier is an essential tool in modern (deep) neural networks: E.g. Natural Language Processing (which word comes next).

Finer details:

- **Scaler:** The $z = w * x$ should not be too large, z should not be much more than 1.
- **Initializing the weights:** The weights should not be initialized to 0, but to random number between -1 and 1.
- **Local minima:** The loss function for MLP is **not convex.** It can be caught in local minima. That is why we make several runs with different initializations and compare the results (mean and std. dev.)
- **Early stopping:** The loss on the training data will decrease during training. There is danger of overfitting by training for too long – the network knows the training set very well but does not generalize.
- **Number of hidden nodes and hidden layers:** Run with different setting which give the best results (called **hyper-parameter tuning – set manually and not learnt**).
- **Alternative activation functions in the hidden layer:** ReLU and tanh.

Overfitting and regularization

- **Overfitting:** The goal is to construct models that fit the training set. Sometimes, we succeed too well. Model fits the training data very well but does not generalize to other data.
- **Regularization:** Punish large weights.

Bias term:

- **About a classifier:** Unfairness, prejudice.
- **Bias-variance trade-off:**
    - **Bias:** The classifier systematically misses the target, e.g. searching for linear classifiers to a non-linear problem. Leads to **under-fitting.**
    - **Variance:** Picking up some noise, being too sensible to small variation in the input data. Leads to **over-fitting.**

Training and test set (repetition)

- To measure improvement, we need at least two disjoint labeled sets:
    - Training set
    - Test set
- We train on the training set.
- Predict labels on the test set (after removing the labels).
- Compare to the given label.
- For repeated development we need two test sets:
    - One for repeated testing during development
    - One for final testing

Cross validation

- Small test set results in large variation in results
- **N-fold cross validation:**
- Split-off a final test set
- Split the development set into *n* equally sized bins (e.g. *n* = 10)
- Conduct *n* many experiments:
    - In experiment m, use part m as test set and the n-1 other as training set

- This yields *n* many results:
    - We can consider the mean of the results
    - We can consider the variation between the results

Ensemble learning
Train several different classifiers:
    - E.g. LogReg, one re more *k*NNs etc.
    - For prediction:
        - Run all classifiers
        - Pick the majority vote (hard vote)
        - Or the average if they provide probabilities (soft vote)
- The ensemble classifier may perform better than any individual classifiers. Even weak classifiers (accuracies just above 0.5) may perform well together, provided they are **independent** and there are **sufficiently** many of them.

Bagging (Bootstrap Aggregating)
- An alternative to different learners:
- Train the same learner on slightly different data
- With much training data, you may train on subsets of the data
- One way to produce different datasets is called bootstrap
- **Bootstrap:**
    - Given a set of training data $D = (X, t)$ of size *n*.
    - Produce a new set by picking *n* samples from D with replacement.
    - Produce several such sets, at least 50.
    - Fit a model to each set.
    - **Prediction** use hard or soft vote.
    - **Subagging:** Similar, but pick sets of smaller size than D (all of same size)
    - **Pasting:** Similar but sampling without replacement.

Random forest
Randomness, step 1
- Given training data $D = (X, t)$:
- Use Bootstrap to construct sets of training data
- Train a decision tree on each bootstrap sample

Randomness, step 2
- At each step in the construction of the tree, consider only a subset of the available features.
- In the algorithm:
    - A parameter m
    - = the number of features to consider at each step
- **Random forest – properties:** Good results both on big and small datasets and embarrassingly parallel.

Reinforcement learning

Introduction

- Training set generated dynamically.
- Told if actions are good or bad.
- Exploration to find the right actions.
- **Application:**
    - **Robot with sensors and motors**: The observations (camera image, touch sensor reading, sound, accelerometers, microphones etc.) result in action (motor torque). The goal could be to keep balance, walk past obstacles or reach a destination.
    - **Computer playing a game:** Observation is the current state, and the action is the next move. The goal is to win the game.

Policy and states

- **Policy:** The way observations are mapped to actions defines a **policy.** Could be deterministic or stochastic (exploitation vs. exploration).
- **States:** All RL systems work within an **environment**. At any time the environment has a particular **state** and this is what we observe. **Actions** can change the state.
- An RL session involves a series of state changes each being the result of an action.

Learning from rewards

- Instead of supplying the learner with the correct outputs (which we do not possess), we will tell if the result of an action was **good** or **bad**.
- **Static policy:** No improvement over time.
- **Agent:** By allowing the policy to change over time, the system can adapt to the environment. An **agent** repeatedly adjusts the policy based on the previous state, the current state, and the reward received.
- Reinforcement learning involves exploration, which is essential to explore different actions, see where they lead, and accumulate knowledge. As training progresses, exploration can be replaced by exploitation.

Defining the policy

- The policy decides what action(S, w) to take in a given state S, using prior knowledge collected in a parameter vector *w*.
- **There are two common approaches:**
    - **Represent the policy as a neural network:** The weights in the network are given by the parameter vector *w* in action(S, w).
        - **Policy-gradient methods:** Execute current policy and collect rewards. After several iterations adjust the weights in the direction of increased expected reward.
        - **Actor-critic methods: Two** neural networks interact and are trained together. **The critic network** learns the values of different (state, action) pairs from the received rewards. **The actor network** learns correct actions using feedback from the critic network.
    - **Represent the policy as a table:** Store all previous learning experience in a table with one entry for each (state, action) combination.

        - **SARSA (on-policy learning)**
            - Choose action *a* using the **current policy**
            - No assumption on the action we are to take, update the Q value by looking at reward and new state. We will follow the policy that the agent has learned, not necessarily the maximum action.
            - E.g. cliff example, makes choice based on policy, frequently ends up outside the cliff and makes it "safer".
            - **Formula:** $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + lr * (r_{t+1} + df * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$

        - **Q-learning (off-policy learning – assuming greedy policy)**
            - Assumes we'll choose the best action.
            - E.g. cliff example, path is efficient, but risky.
            - **Formula:** $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + lr * (r_{t+1} + df * maxQ(s_{t+1}, a) - Q(s_t, a_t)$

        Which is better? Depends on the problem we're trying to solve. Higher risk tolerance: Q-learning and lower risk tolerance: SARSA.

- **Three different ways of using the Q-table:**
    - **Action selection:** Need to decide whether to exploit or explore, need a balance of this to explore the environment.
    - **Greedy:** Choose the highest value.
    - **e-Greedy:** The choice between the greedy action and a random one depends on a **randomly drawn number** compared to a threshold epsilon (ε). Let's say epsilon is 0.1 (10% of the time, we'll explore.
        - With probability 1 - ε (90% of the time), we choose a3.
        - With probability ε (10% of the time), we choose randomly between a1, a2, and a3.
    - **Softmax:** The softmax output values can be interpreted as the predicted probabilities. Sum of all softmax outputs adds up to 1.

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}$$

Unsupervised learning

1. **Introduction**

- **Unsupervised learning:** We have a training set consisting of only a feature vector $x = x_1, \dots, x_m$ for each sample. The goal is to learn something interesting about the **distribution** of the *x*'s.
- **What can we learn in unsupervised learning?**
    - How to generate similar data
        - Density estimation, generative models
    - How many (and which) groups are present
        - Hierarchical clustering, k-means clustering
    - How to compress data
        - Autoencoders
    - How to visualize the data in low dimension
        - PCA, learning vector quantization, self-organizing maps
    - **Evaluating the performance:** No "truth" to compare with, performance more open to interpretation. Still: Often possible to define a useful loss function and to determine which of the two solutions is best.

2. **Learning how to generate similar data**
    - We start with the training set data. We learn the data generation mechanism with a model describing how to generate data similar to the training data set. We apply the data generation mechanism to get the generated data (as many as we want). At the end, is the validation phase, where we compare the distribution of generated data with distribution of training data.
    - E.g. generating a digital picture, if learning is successful, we get somewhat similarly generated pictures.
    - **For remaining part of unsupervised learning:**
        - **Discrete random variable:** Can take a discrete set of different values. E.g. x = 0, 1, 2, 3. A probability distribution for a discrete random variable describes the probability of different values occurring. E.g. rolling a fair dice with six sides.
        - **Continuous random variable:** Can take a continuous set of different values. E.g. all real values on an interval [a, b]. A probability distribution for a continuous random variable must be formulated slightly differently. E.g. measure the time it takes for a soap bubble to burst measured in hours with decimals. Taking any particular time point in the time interval, the probability that the bubble bursts at exactly that time point is 0. So not a good measure of distribution of events.
        - **Density functions:** The problem in the last example was that the probability of X taking a particular value *x* is infinitely small. The solution is to define a density function *f(x)* always non-negative and the total area under the curve on the interval. It summaries some important aspect of the data.
        - **Learning a density:**
            - **Histogram**

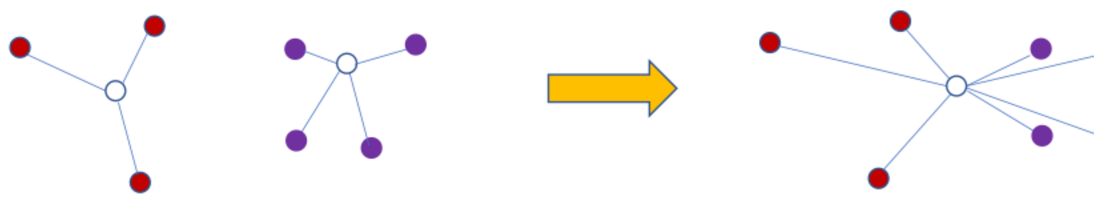
- **Smooth density estimator**
- **Generative adversarial networks (GANs)**

3. **Learning how data are grouped**

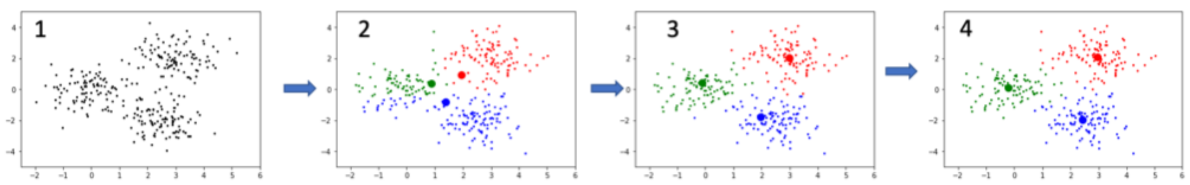
- Learning how the data are grouped is called **clustering.**

Clustering methods:

- **Hierarchical clustering**
  - **Algorithm:** 1. Initially, each sample is a separate cluster. 2. Find the clusters that are closest together and merge them into one cluster. 3. Repeat point 2 until there is only one cluster left.
  - **Important aspects:**
    - Must keep track of the whole history of cluster merges in order to plot the tree (dendrogram).
    - In each iteration we have to compute the distance between every pair of the remaining clusters.
    - We have to define what we mean by the distance between two clusters.
  - How to define the distance between two clusters $c_1$and c_2 is referred to as linkage method and has great influence on the result. The distance between two samples is usually measured using the **Manhattan, Euclidean or Cosine distance.** Most common linkage methods are:
    - **Single linkage:** Smallest distance between a sample in c_1 and a sample in c_2
    - **Average linkage:** Average distance between samples in c_1 and a samples in c_2
    - **Complete linkage:** Maximal distance between sample in c_1 and a sample in c_2
    - **Ward linkage:** Increase in within-cluster variance if the clusters are merged.



- **K-means clustering**
  - **Algorithm:** 1. Randomly select k cluster centres. 2. Assign each sample to the nearest cluster centre. 3. Update cluster centres to mean of samples in cluster. 4. Repeat steps 2-3 until convergence.



- **Voronoi diagram**: Part of the plane that is closer to one particular cluster centre than to any other cluster centre.
- **K-means clustering (algorithmic details):**
  - Initial cluster centre selection:
    - Random (uniform, normal) coordinates within data range
    - Random selection of the data points
    - Manual selection
  - Distance between a sample x_1 and a cluster centre c_i
    - $d(x_i, c_i) = \Sigma\left(x_{ij} - c_{ij}\right)^2$
  - Cluster centres are updated by calculating the average of all the sample vectors assigned to it
    - $c_i^{new} = \frac{1}{m}\Sigma x_i$

- **Competitive learning**

4. **Learning how to compress data**
   - **Autoencoder:** An autoencoder is a neural network that learns to produce an output similar to the input. If the hidden layer is smaller than the input/output layers, the network learns a compact representation of the data. For autoencoders, we use **reconstruction loss.**
     Special types of autoencoders:
     - **Sparse autoencoders:** Trained to obtain a sparse hidden layer representation (= only a few hidden nodes active at a time).
     - **Denoising autoencoders:** Trained to recapture a denoised version of the input vector.
     - **Contractive autoencoders:** Trained to obtain a hidden layer representation that is robust to small changes in the input.
     - **Variational autoencoders:** Trained to obtain a representation useful also for generating new data from same distribution.

5. **Learning how to represent data in low dimensions**

Purpose for dimensionality reduction:
- To plot high-dimensional data
- To visually explore structural features of a data set
- To reduce the number of features before further analyses
- To reduce noise

How is dimension reduction different from compression?
- Autoencoder can also be used to compress data – but the aim is different.
- **Compression:** Encoded version of data should be compact and possible to decode to original format. Not for human interpretation.
- **Dimension reduction:** Encoded version of data should preserve only important structure and be useful for human interpretation or input to other analyses. Decoding is not the aim.

Dimensionality reduction methods

The goal is to find Low-Dimensional Representation (LDR) of data.

- **Principal Components Analysis (PCA):** Find LDR maximizing the variance.
- **Multidimensional Scaling (MDS):** Find LDR preserving pairwise distance.
- **T-distributed Stochastic Neighbour Embedding (t-SNE):** Find LDR preserving neighbour relations in probabilistic sense.
- **Uniform Manifold Approximation and Projection (UMAP):** Find LDR preserving topological structure of data.

Principal Components Analysis (PCA):

- Start with a data set.
- Centre the data around mean.
- Find weights maximizing the length of Y**w.**

Deep Neural Networks

The deep learning revolution

- **Deep learning:** A sub-class of neural nets. There is no exact definition, but more an **attitude/approach** than a defined class. There are normally at least two hidden layers and often a much more specific architecture.
- **The revolution:** Deep learning had a breakthrough 12 years ago. It spawned the great interest in AI we have seen the last years (AI and not only ML). E.g. self-driving cars, chatbots and image classification.
- **Image net competition:** The image classification challenge: Alex Net won in 2012 based on deep NNs.
- **The beginning ($\rightarrow$ 1969)**:
    - **1958:** Rosenblatt invented the Perceptron.
    - **1969:** Minsky & Papert, The perceptron:
        - Networks without hidden layers can only learn linear classifiers.
        - Networks with hidden layers are probably impossible to train.
    - Less interest in perceptron's afterwards.
- **Backpropagation (1986 -)**
    - **1986:** Rummelhart, Hinton and Williams reinvented backpropagation. An immediate enormous interest by researchers. But the practical results weren't impressing, and the interest diminished.
- **Deep learning:**
    - In the 1990s and 2000s other approaches to ML was preferred in usage and developed, e.g., log.reg. and SVM.
    - Some brave and stubborn researchers continued the work on neural nets, including Hinton, and got their rewards in 2010s.
    - NNs finally succeeded because of better models, more data and more powerful machines (GPUs).

Deep feed-forward neural networks

- Several hidden layers
- The number of nodes in each layer may vary
- Fully connected: edges from each node in one layer to each node in the next layer.
- **Weight matrices:** One matrix of weights for each layer.

- **The hidden nodes – forward:**
    - Same activation function at all hidden layers (e.g. logistic or ReLU)
    - At node j in layer k:
        - First sum of weighted inputs
        - Then apply the activation to the sum of the weighted inputs
- **Forward:** Each hidden layer behaves like the hidden layer when there is only one. The output layer behaves like the output layer when there is only one hidden layer.
- **Update:** Compare the output values to target values: $L(y, t)$
    - L is a loss-function (there are alternative loss functions)
    - If y = t then $L(y, t) = 0$ so no update. The larger the difference between y and t, the larger the loss and update.
- Formula for computing the delta rems in the hidden layer:
  $$delta_{hidden} = a_{hidden}(1 - a_{hidden})\Sigma_{i=1}^{n} delta_{right} * weight_{hidden,i}$$
- **Vanishing gradient problem:** The derivative of the logistic function is close to 0 except in a small interval around 0. At each backwards step calculating the deltas, we multiply with the derivative of the activation function. The gradient comes close to 0. Unbearable slow update, or no update at all.
- **Rectified Linear Unit (ReLU):** Alternative activation functions in the hidden layers:
    - ReLU(x) = max(x, 0)
    - ReLU'(x) = 1 for x > 1
    - ReLU'(x) = 0 for x < 1
    - Use 0 for ReLU'(0)
    - **ReLU** is the preferred method in deep networks.
    - **At the hidden layer with activation function g = ReLU**

  $delta_{hidden} = a_{hidden}(1 - a_{hidden})\Sigma_{i=1}^{n} delta_{right} * weight_{hidden,i}$ for x >0 and + for <= 0.

Convolutional NNs and image processing

- Designed to process data with a grid-like topology.
- Image classification: An image can be represented as m * n many pixels, e.g. 28 * 28. If it is in colors, each pixel can be three numbers e.g. between 0 and 255. We can represent this in a neural net with m * n * 3 input nodes.
- **The problem:** Positive class if it contains at least one subfigure exactly the shape and size. How can a classifier which takes pixels as input recognize this? There is no similarity in the pixels.
- **Approach:** Positive class if it contains at least one subfigure of exactly the shape and size. We split the task into two: For each 5 * 5 subpicture, decide whether it has this shape or not. Answer whether the picture has at least one such subpicture.
- **The filter:** We slide a 5 * 5 windows over the picture and report the result each time. We can solve this task: determine whether the picture contains this 5 * 5 subfigure.
- This can also be applied to text. The learning is done as for other multi-layer neural nets by backpropagation. Demands more training data and more machine power.
    -

Recurrent NNs and language processing

Introduction to Recurrent Neural Networks

- Recurrent networks (RNNs) are heavily used in Language Technology.
- Model sequences/temporal phenomena. A cell may send a signal back to itself – at the next moment.
- Processes data sequentially, where order of the elements is important such as sentences in text or time series data.

Applications in Language technology

- **One-hot encoding:**
    - A word Is a categorical feature.
    - We assume vocabulary of e.g. 100000 different words.
    - We could use a "one-hot" encoding ("one of *n*")
        - (0, 0,0 ..., 1,..., 0)
        - One 1 and 99999 many 0-s
        - Different words, different positions.
- **Embeddings:** Represent each word with a vector of reals and a fixed number of positions, e.g. 100. Try to get similar vectors for similar words. Words can be considered similar if the occur in similar positions. These representations (**embeddings)** can be learned from a form of language modelling task: predict whether a neighbouring word can occur together with this word.
- **Machine translation:**
    - **Bi-text:** Text translated between two languages. The translated sentences are aligned into sentence pairs.
    - Machine learning based translation systems are trained on large amounts of bitext.
- **Encoder-decoder based translation:**
    - Concatenate the two sentences in a pair:
        - Source sentence_<\s>_target sentence
        - Train an RNN on these concatenated pairs
        - Apply by reading a source sentences and from there predict a target sentence.

Developmental Systems and Swarm Intelligence

Developmental Systems

Morphogens

- **Morphogens:** Chemicals that influence development. E.g. embryogenesis, or more morphogen, more beak on birds.
- Powerful indirect mapping of genotype to phenotype.
- **Reaction-diffusion models through self-inhibition** describe how chemical substances spread and react over space and time, typically resulting in patterns or structures due to inhibition of rection products on their own formation.
- **Gray-Scott algorithm** is a particular computational simulation model that uses a set of partial differential equations to mimic the behavior of reaction-diffusion systems, often producing complex, self-organized patterns akin to those found in nature.
- **Calculation with Laplace Transform** is a mathematical technique used to simplify the analysis of linear time-invariant systems by transforming differential equations into algebraic equations in the frequency domain, making them easier to manipulate and solve.
- **Cellular Automata** are discrete, algorithmic simulations that consist of a grid of cells, each of which can be in one of a finite number of states, where the state of each cell changes over time according to a set of rules that depend on the states of neighbouring cells.

Rewrite Systems

- **L-system:** An L-system is a recursive string rewriting mechanism used for modelling plant growth and fractal patterns.
- **Matrix Rewrite System:** A computational framework where matrices are manipulated by applying transformation rules to produce new matrices.
- **Graph Rewrite System:** A set of rules for transforming graphs by changing their nodes and edges to model complex structures.

CPPNs and Curves

- **CPPN (Compositional Pattern Producing Network)**: A neural network type that generates patterns or spatial structures through the composition of simple functions, used in evolutionary computation.
- **Bezier curve:** A parametric curve used in computer graphics and related fields to model smooth curves that can be scaled indefinitely.

Swarms, Particle Swarm Optimization

- **Swarm intelligence:** “The study of large collections of relatively **simple agents that can collectively solve problems** that are too complex for a single agent or that can display the robustness and adaptability to environmental variation displayed by biological agents.
- **Hive mind:** Brain of brains
- Emergent intelligence forms
- Swarms can enhance the intelligence in groups.

- **Emergent collective behaviour:** Aggregation, clustering, foraging, nest construction.
- **Extended phenotype:** How an animal's genes can affect the world, e.g. is a beehive an extended part of a bee?
- **Boids: 'bird-oid object':**
    - **Separation:** Boids steer away from close neighbours.
    - **Alignment**: Boids steer towards the average heading of their neighbours.
    - **Cohesion:** Boids steer towards the average position of its neighbours.
- **Particle Swarm Optimization (PSO):**
    - Population based metaheuristic like evolutionary algorithms.
    - Candidate solutions are particles.
    - A particle contains positional and velocity parameters:
        - Position defines the adjustable/mutable parameters of an individual.
        - Velocity represents how these parameters are updated.
    - The positional parameters can be viewed as the *genotype* of individuals.

Ant Colony Optimization and Reconfigurable Robots

Ant Colony Optimization (ACO)

- **Stigmergy:** Social communication through modification of the environment.
- Why **swarm systems?**
    - **Continuous adaptation:** Dynamic network routing and urban transportation.
    - **Decentralized, asynchronous**
    - **Collective decision making**
- **Reconfigurable robots:** Robots composed of modules that can change the shape of the robot to adjust its functionality. Passive approaches vs. active swarm approaches.
- **Useful evolutionary robotics:** Shape-changing: Evaluation of different body shapes.

The History and Philosophy of Artifical Intelligence

The birth of AI

- **Darthmouth Summer Research Projects (1956):** John McCarthy and Marvin Minsky. MIT, Standford. Birth of AI.
- **General Problem Solver (1957):** Goals-mean analysis. Could not handle the computational power needed.

The Turing test and a little more philosophy related to AI

- **The Turing test:**
    - Try to fool the interrogator to think it is a human
    - Try to help the interrogator to see that he/she is a human
    - The interrogator should guess who is human and who is machine
- **Evaluating the Turing test:** Is it adequate? Can a computer pass the test? Is it a goal that a machine passes the test?
- **Chat bots:** Eliza, domain specific chat bot (e.g. psychotherapy)

- **Has the Turing test been passed?** Now and then stories in the news that the Turing test has been passed. According to the rules, the human should be cooperative (in which they are not in the news stories).
- **Philosophy of AI:**
    - Can a machine pass the Turing test?
    - Can a machine act intelligently?
        - If a machine is considered intelligent bby performing tasks considered intelligent by humans, then yes. But it can't solve any problems.
    - Can it solve any problem that a person would solve by thinking?
    - Can a machine have a mind and consciousness?
        - **Strong AI:** Not only do the same as humans, but also do it the same way, which includes having a mind and consciousness that machines don't have.
    - Are human intelligence and machine intelligence the same?
        - A physical symbol system. Implies that machines are intelligent.
    - **Connectionism:** After the introduction of backpropagation, some pshycologist and philosopgers argued for modelling the human mind in terms of neural networks.

Traditional approaches in AI

- **Symbolic:** Could use other representations than logic, e.g. GPS.
- **Rule-based:** Expert systems
- **Combined with search:** TSP, challenge is large search space.
- **Logic:** Rationality, intelligence, correct reasoning. Computer is based on logic – logic seems to be the perfect link between human intelligence and computers. Challenge is that logic provides proofs, and to find a proof is a challenge.

More recent trends

- **What is AI?** A machine is considered intelligent if it can perform taks which are considered intteligent when carried out by a human being. Hence AI focused on language and mathematics.
- **Intelligence**
    - **Higher-level intelligence:** Playing chess, first year university mathematics – machines are good at this.
    - **Lower-level intelligence:** Face recognition, moving around – machines are not good at this.
- **Nouvelle AI/behavior based robotics:** Brook's behavior-based robotics was inspired by evolution. Animal-inspired robots acting in the real world. Decomposed by activity: One system for avoiding collision and another system for goal-directedness. Commercialized iRobot, these ideas are also essential for the development of e.g. self-driving cars.
- **AI becomes an empirical science:** E.g. Natural Language Processing. Real-world data, a bottom-up approach compared to a top-down approach which were common in AI/NLP. Machine learning, rigid ecalution, large amounts of data available (data science).

- **Deep Learning:** Third large change to AI since 1990 is the DL revolution since 2012. The revival of neural networks made AI popular again. Even more popular with the advent of generative language models (ChatGPT).
- **Symbolic, Rule-based vs. ML, NNs:** NNs good at manyt tasks, but not all. NNs are often black boxes, they give prediction but no explanations. There is a demand for explainable AI. Rule-based symbolic AI may strike again.

Ethical Issues and Future Perspectives

- **Which jobs are at risk?**
  - Routine and repetetive tasks
  - Low-skill jobs
- Large difference between countries, because some already have a high skilled workers with high level of automation.
- Technology removing human jobs is not a new concern
- Technology may create more jobs than it removes, still a lot of people will be removed. Possibly handled with re-education programs, robot tax, UBI.
- **Superintelligence:** We build more and more intelligent machines and not likely to hit a wall soon. **Intelligence explosion:** At some point, we will reach a machine so intelligent it can make more intelligent copies of itself rapidly. Intelligence will **explode**, increasing rapidly in perhaps hours/days.
- **Evil machines?** If the machine's goals are not precisely aligned with ours, we're in trouble. Machines would be so intelligent it's goals would win any disagreement, why not just program in its goals very specifically?
- Current AI research is so far from superintelligence that worrying/planning for this is not very constructive.
- **Risks and ethical issues:**
  - **Biases:** Biased training data.
  - **Consciousness:** Rights.
  - **Ethical Dilemmas:** Self-driving cars: hit pedestrian to save driver?
  - **Privacy:** When you train predictive models on input from users, it can leak information.
- **Fundamental Limitations to Current AI:**
  - **Main message:** DL has revolutionized many areas of AI, and greatly increased the range of problems we can solve with AI. However, there may be a large set of problems deep learning cannot solve that the brain can solve. Some of the most hyped AI successes of recent years suffer from surprisingly fundamental difficulties.
  - **Robustness:** Deep RL: RL straight from pixels – one of recent year's most discussed successes. However, learned policies may be extremely sensitive to changes to the input. Adversarial attacks on image-classification networks.
  - **Understanding Human Language:** Extremely impressive progress has been made in modelling human language in recent years, with applications such as:
    - **Translation**
    - **Text summarization**

        - **Automated image description**
        - **Automated text generation**
    - **But what does it "understand" about the world?** Impressive DL systems learn in a very different way from humans. They are vulnerable to simple attack indicating they do not *understand* the images they are looking at. Perhaps we should be more worried about these systems not being smart enough.
    - **More limitations to current AI:**
        - Explainability
        - Continuous learning
        - Extremely data-inefficient learning
    - **How do we address these limitations?**
        - DL part of the solution, not the entire solution.

- **Fair machine learning:** A fair machine learning system is designed to minimize bias and make equitable predictions across different groups. It involves careful data analysis, bias detection and mitigation, transparency, and accountability in implementing and monitoring the algorithms to prevent discriminatory outcomes based on sensitive attributes.