Lecture 1 - Anda

Historical note

↳ The lectures are flipped classroom. Watch videos before lecture

Perceptom 1943 from McCulloch and Pitts. They discovered single nueron and single neurons. They thought they could bring structures from brain to computer.

Donald Hebbs, neurons co-acted work together. ADALINE basically founded SGD. They discovered delta-rule.

Rosenblatt Percepton, weights could be re-valued. Built it mechanical and said it work learn by it's own. First 1969 → 1981 dead silence.

Hopfield' came in and came with backpropagation. 1986 meant you could solve hard problems. Around 90s everything joined up and you got advanced models.

2010 you could go deeper. The more deeper, the more better the model becomes. A billion wide scope of applications.

Learning activations

↳ 11 Lectures (13 sessions)

↳ 12 regular lab review sessions (5 bonus point review dates) → Help sessions

↳ 5 mandatory labs
↳ 3 Kontroll Skrivninger
↳ 1 Tentamen

} To pass the course: Hard course

Attend the Fucking lectures!

Lecture 2 - Perceptons, backpropagation

$y' = w_1 x_1 + w_2 x_2$ — y' => Before step function

$y = f_{step}(y')$ — y = Step function

Weighted data goes in, check threshold then check

$y' = w_1 x_1 + w_2 x_2$ if $y' > 0$ then $y = 1$

else $y' < 0$ then $y = 0$

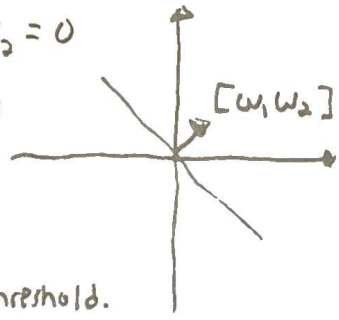
→ ~~the~~ Geometric interpretation

$w_1 x_1 + w_2 x_2 - y' = 0$ => This is a hyperplane.

$ax + by - cz + d = 0$

Let's say threshold is zero: $y' = w_1 x_1 + w_2 x_2 = 0$

the "normal vector" becomes w_1 and w_2. With this information: * You get the decision boundary



There is a Symbol θ which represents the threshold.

Basically it moves the "starting line" from Origo to elsewhere.

Called "bias". He mentions you also set a weight on it

and represented as $w_0 1$. where $w_0 = -\theta$

You can do binary Classification with this model.

$y' = w_1 x_1 + w_2 x_2 = w^T x$

Lecture 2 -

→ Principle of Perceptron Learning

How do we go about iteratively adjusting weight, w

$$w^{new} = w^{old} + \Delta w$$ Ok what is the intuition

1. If correct classified, do nothing
2. If data output should be 1 but we get zero => $\Delta w = x$
3. If data output should be -1 but we get 1 => $\Delta w = -x$

Basically modify weights on how "wrong" your were with the data

$w^{new} = w^{old} + \Delta w$ where $\Delta w = -\eta x$. η = learning rate, eta

Weights are only changed when missclassified. Learning are adjusted during the training. Convergence theorem

[Convergence theorem: As long as the data is linearly separatable then the percepton training will converge after a large number of steps]

↳ You might have premature termination. Basically a shit solution. Happens when patterns "approximately similar" to those used in training. Gives bad generalisation.

→ Delta Rule (widrow-huff rule, ADALINE)

1. Symmetric target values $\{-1, 1\}$
2. Error is measured before thresholding $[e = t - \vec{w}^T \vec{x}]$
3. Find weights that minimise the cost function $[\mathcal{E} = \frac{e^2}{2}]$

Very similar to mean square error

The goal is to minimize the cost function $E = \frac{e^2}{2}$. The simple algorithm is <u>steepest descent</u>

→ Gradient defines the direction in which error increases the most

→ In this algorithm move in the opposite direction.

→ $\frac{dE}{dw} = \frac{de}{dw} = \frac{d(t - \vec{w}^T \vec{x})}{dw} = -e\vec{x}$ here $[e = t - y]$

→ Delta rule becomes $\Delta\vec{w} = \eta e \vec{x}$

Difference between Perceptron Learning and delta rule, error is $\epsilon - y$ in Perceptron and in Delta $t - \vec{w}\vec{x}$. Makes large difference.

<u>What is Multilayer-Layer-Perceptron</u>

↳ Feed forward networks that feeds output of perceptron to another Perceptron. You cannot train it with Perceptron learning or Delta Rule. For Perceptron we feed data to input hence why not possible. For delta Rule we need to know output before thresholding → not possible.

↳ Dilemma to solve when training MLPs. Thresholding destroys informations needed for learning. Without thresholding we loose advantage of multiple layers. The solution is to use threshold-like but differentiable function.

Backpropagation overview

1. Forward pass where you calculate biases and output of each layer
2. Backpass where you calculate output of each layer and their error
3. Update the backpass and forward pass

There are few limitations with Backpropagation:

* Slow convergence
* Getting stuck in local minima
* Many parameters must be tuned
* Challenging Scaling
* Biologically unrealistic

Practicalities when dealing with the algorithm

→ Sequential (online) versus batch mode

→ You update network after each sample during sequential.

→ During the batch mode, you update ONCE however you accumulate the updates and apply it ONCE per epoch, not once per data.

Sequential updates is faster especially for large datasets. Also less susceptible to getting stuck in local minima. Referred to Stochastic Gradient Descent

→ Weights initialisation

→ Random initialisation, Gaussian Distribution or uniform distribution with zero mean and $\sigma^2 = \frac{1}{fan\_in}$ (heuristic)

→ Learning rate adaptation

→ Cyclic rate, Adam, exponential and other.

Lecture 3 - Generalization, Regularisation, Model selections and valid

Generalization: How well concepts learned by the model apply to new examples not seen by the model during training. Factors that have an impact are

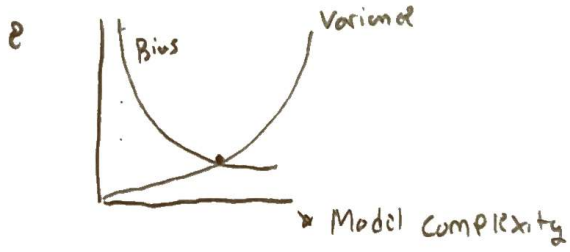
1. Size and quality of the data
2. Complexity of the machine
3. Physical complexity of the underlying problem

Let data - $D = \{(x,t)\}_{i=1}^{N}$ $\quad X \to T : T = f(x) + \varepsilon \quad \mathbb{E}[\varepsilon f(x)] = 0$

We want to find F. We approach the problem with a NN estimator, $F(x,w)$ and define the L2-Norm Risk $R[F]$

$$R[F] = \mathbb{E}_D\left[(t - F(x,w))^2\right]$$

The need to balance bias (approximation error) and variation (estimation error) on a limited data sample.



- Occams Razor. Not too complex, simple to explain the model.

Ok why do we estimate generalization?

↳ This is a criterion on model selection

↳ Good generalization is important.

We do this by splitting training and testing dataset. One popular way to estimate bias is Bootstrapping. Rests on the concept of re-sampling from available F data to estimate bias. There is also n-fold cross-validation.

Lecture 3

Regularisation: Methods to minimize generalization errors but not training data.

- **Early stopping**: The network is trained with BP until the error monitored on the validation set reaches minimum. As soon as we hit minimum we stop training.
- **Network growing and pruning**: Prune the model if network becomes too big and grow it if network is too simple. Model size matters.
- **Penalty term**: Penalty for adding too much weight. We penalize complex models. Two ways
  - L2 Regularisation (weight decay). Weights become smaller or close to zero but not really zero
- **Bayesian Regularisation**: All quantities are treated as random variables. There is a prior distribution over unknown parameter, Bayes rule produces a posterior distribution for these parameters - Prior evidence is taken into account.
  - Give intuitive solutions. No need for hold out set to estimate generalize. Give confidence intervals
- **Ensemble learning**: Combine weak learners and boost performance. Basically democracy of ML models
  - Ensemble
    - Bagging
    - Boosting

Lecture 3

Bagging: Draw a lot of bootstrap samples

↳ Ensemble Each resample can be treated with additive Gaussian noise

Transfer learner for each bootstrap sample

Combine the outputs of all learners

→ Elevates variances, reduces biases

- Adaboost XGBoost are popular methods

Drop-out: Powerful regularisation methods. Basically "cut off" some parts of the networks to avoid tuning network. The parts you drop are "randomly

Regularisation Video 2 Extendent

L2 and L1 penalize large weights but do so differently

- L2 shrinks weights proportional to the size of weights
- L1 encourages sparsity as it leaves some weights to zero.
- L1 drops weights at a constant rate

Weight Elimination: kill weights entirely and keep a few large weights.

Lecture 4 - Anns approaches to pattern recognition problem

Loss functions optimization <- video

- Loss function should correspond to find the optimum values for the parameters by maximizing a likelyhood function derived from the training data
- ML loss functions how closely we were the correct ones with our models guesses. Guess right.

The most popular choices

- Cross-entropy with sigmoid activation function in a single or multi-classification task. Softmax as well
- MSE as loss function with linear activation function for regression problems.
- Mean squared logaritmic error, Mean absolute error loss, Hinge loss for classification.
- There's is Precision => Ratio of true positive / All guesses F-score=
- ROC curve is also good to use. Tells you how accurate you are between false positives and false negatives.

Outline of optimisation algorithms → Optimization of eta

- Stochastic Gradient Descent versus Mini-batch GD
  - ↳ You can add "momentum" that is SGD with Nesterov momentum (NAG)
  - ↳ Adaptive Gradient Descent (AdaGrad) - Adative learning rate
  - ↳ AdAM → exponentially decaying average of past squares. (good one)
  - ↳ Quick prop
  - ↳ Newton method → Speeds up converge but takes time
  - ↳ Levenberg-Marquardt algorithm
  - ↳ Conjugate gradient method

Practical Aspects of ANNs to pattern recognition problem <- video

Pipeline recognition problem

1. Preprocessing
2. Features, Low-level data representation
3. Classification
4. Postprocessing

=> **Preprocessing**

=> What is the objective of the data

=> What data is available

=> How are/were data generated!

Look in data quality, de-noising, outlier analysis. Data transformation, normalisation and data analysis, Data augmentation

Lecture 5 - Radial basis function

What kind of data

Discrete or continous data?
Amount of data?
Missing data?
Noise or correlation

Lecturer

Erik talked about importance to view data and properly understanding it before building your model. Your model is after all dependent on your data

→ Something called kernel trick where you blow up your problem into a higher space where it becomes easier to map things for classification.

→ When training MLP, you are training/modifying weights and biases. Now when training RBF networks, you are changing RBF location.

Lecture 5 slides starts now

- Cover's theorem basically states that a projection to a higher space is easier to solve compared to a lower space

- You need a kernel function such that: $\phi(\|x - x_i\|)$ x_i = RBF center
  This basically projects into higher space ϕ = Kernel function
  $\| \|$ = Vector norm

$$F(x) = \sum_{i=1}^{N} w_i \phi(\|x - x_i\|)$$

- In exact interpolation, the size of the hidden layer N is equal to the number of samples. ← This is not robust if data is large and noise in data.

1. $N < n$
2. Centres x_i different sum samples
3. Widths, σ, also differ across RBF nodes
4. Possible to include biases

Lecture 5

RBF learns by doing Hybrid Learning on the output learning. It does least squres to find and adjust the errors

RBF nets verses MLPs

- Hidden units in MLP rely on weighted linear summations of inputs. RBFs rely on the distance to prototype vectors
- MLP function approximation is defined as a nested sum of weighted summations. In RBFs the approximations is defined by single weighted sum.
- MLP form distributed activations. In RBFs, very few local basis functions are activated for a given input.
- In MLP usually all parameters/weights are trained at the same time.
  In RBF, hybrid two-stage training is used. =>
  - ↳ How to position RBF
  - ↳ Regression of outputs
- MLP rely on complex multi-layer architecture
  In RBF nets have one simple hidden layer architecture

Lecture 5

Hybrid Learning in RBFs

- Clustering algorithms
  - ↳ k-means
  - ↳ Clustering with Kohonen feature maps (SOMs), vector quantization (VQ)
  - ↳ Estimate Cluster centers (variance)
- Gaussian mixture models: Expectation Maximation problem
- Gradient descent (super computational heavy)

Learning

RBF has a "winner takes all" policy where the best one get's to learn and change.

- By change, it can change perceptron-like or other way Delta way.
- ↳ How do we do initialisation? One datapoint can win all the time. This is the core of competitive learning.

Lecture 6 - Self-organising maps (SOMs)

Vector Quantisation

The idea borrowed from information coding, data compression. Represent data in terms of few limited typical data vector - code vectors.

→ Compression

→ Noise reduction

For unsupervised RBF networks you have winner takes all mechanism. Only one output gets to learn - hard competition.

↳ Fundamental principle: update the winning unit making it more specialised.

Properties

↳ Algorithm find cluster in the data
↳ Each unit protects it's territory

} Dead unit problem. Prototype vectors far from actual data will never become better.

To avoid dead units:

↳ Initialize algorithm with data samples
↳ Leaky learning (soft competition) - Same updates for all
↳ Learning with consience - balance update allowing losers to win
↳ Introduce noise to the data

Topographic map (Kohonen maps) <- not

Not used for classification or regression.

Learning principle: Competitive learning where winning "spills" over to neighbours.

$$\Delta w = \eta h (x - w)$$

w - weights

η - learning rate

h - winning or losing rate

SOM Learning

Following initialisation there are three learning stages

1. Competition (mapping continuous input space onto a discrete output space of unit)
2. Cooperation (lateral interaction through topographic neighbourhood)
3. Weight (synaptic) adaption

Summary from chatgpt

Unsupervised algorithm used to produce low dimensional discrete representation of the input space of the training samples called a map, and therefore a method of dimension reduction. Operate on the principle of organising a set of neurons in a grid in such a way similar input patterns are mapped to adjacent nodes in the grid, preserving the topological properties of the input space. Achieved through a competitive learning process.

Useful for visualisation of high data since they can reveal clusters, relationships.

Used for tasks like clustering, visualisation and feature extraction.

Learning Vector Quantization (LVQ)

Supervised competitive learning algorithm. You do data compres

Lecture 7a - Hopfield network and associative memories

Most memory in brain is recurrent connected neurons. Most Recurrent Neural Networks is inspired by Hebbian learning. A vector consisting of $[-1, 1]$ if describing neurons is describing memory.

Associative pattern recognition

Heteroassociate - Maps patterns in input X into input space Y. Associate boat with a landscape

Pattern recognition - Identify a picture of a boat to the word boat

Auto-associative - Identify boat as boat. "Remembers it"

To make Recurrent networks better, we add bipolar coding $\{-1, 1\}$ with Sgn transform $\begin{cases} 1, & x>0 \\ -1, & x<0 \end{cases}$

Becomes Hebbian learning (Correlation learning, outer product)

$$W = W^1 + W^2 + W^3 + \dots W^m$$

$$W^k = [w_{ij}] = [x_j^k \, y_i^k] \quad \text{(outer product)}$$

$$\vec{Y}(\vec{x}_p^T \vec{x}_p) + \underbrace{\sum_{p \neq k}^{m} \vec{Y}_k (\vec{x}_p^T x_k)}_{\text{Cross-talk - Hinderance}}$$

For Hebian learning

Our goal: $\text{Sgn}(Wx) = \vec{x}$, $\text{Sgn}(W\vec{x}) = X$

Essentially, $\vec{x}$ are the eigenvectors of nonlinear Sgn operator so the idea is to find W for which $\text{Sgn}(Wx)$ has these patterns as eigenvectors. But we do not want $W=I$ as a trivial solution of $\text{Sgn}(Wx) = X$

W describe non-orthogonal projection onto subspace spanned by $\vec{x}$.

Continous Hopfield networkes does not work with [-1,1], training is larger and basically a theoretical model. Hopfield networks are good at solving constraint satisfaction problems. If you build the constraint into the model as energy then you can use hopfield to converge to a minima solution

Boltzmann Machine

- Not used in Engineering but theoretical systems. Underlying of Deep Belief network.
- Generalization of Continous Hopfield Network. You have some probability on when a neuron fires, the model is not deterministic anymore.

$$P(v) = \frac{1}{1+e^{-v}} \quad \text{where} \quad v = \frac{1}{T}\sum w_i x_j \qquad x \begin{cases} 1 & \text{with probability } p \\ -1 & \text{with } 1-p \end{cases}$$

Thinking of Simulated annealing

- Energy of this stochastic network is the same as before

$$E = -\frac{1}{2}\vec{x}^T W \vec{x} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}\, x_i x_j$$

- key difference is $P_{s_1 \to s_2} = \frac{1}{1+e^{\Delta E/T}}$

Quiz 2 - Preparation

RBF networks

- An input layer, a single hidden layer of RBF neurons
- Neuron has radial basis function as its activation function.
- The output of each Neuron is a measure of similarity between input and a prototype vector
- You can choose learning for hidden-layer
  - ↳ K-means clustering ↳ Som nets ↳ Vector quantization ↳ GMMS ↳ Gradient Descent
- Least Square fitting on output
- You COULD use competitive learning

SOM

↳ Learning is done at three stages

1. Competition 2. Cooperation 3. Weight (Synamic) adaption

↳ Unsupervised model, used for Clustering, Data mining, Visualization, feature mapping

↳ Basically a model that maintains the topological data ordering. Used to see patterns in the dataset

Hopfield networks

↳ Serve as "associative" memory system with binary thresholds.

↳ Learn through Hebbian learning. Weights between Neurons are adjusted according to the rule that the change in Synaptic weight is proportional to the product of the pre and post-Synaptic neuron. Both weights are calculated as the sum of the outer product of each patter with itself

↳ Used for pattern recognition, associative memory and solving optimization problems

Lecture 9 - Deep Learning Fundamentals

The goal of AI is that Computers exhibit what we call Intelligence.

So we need:

- Knowledge
- Learning
  - Complex functions
  - From unlabeled data
  - little human input
- Generalization
- Understanding the underlying factor

Factors that contribute Deep learning:	Where are we now? (applicability)
- Computing power:	- No human expertise or expertise hard to formalize
- Data availability:	- No underlying physical/math models
- More effective algorithm:	- Problems with search space exceeding human capability
- Open Source tools:	- Tendency to automate and reduce human involvement

Critical of Deep learning

- Shallow meaning of "depth". Not deep problems
- Lack of Transparency
- Causality vs Correlation

Trouble with Classical Multi-layer Ams

- They are hard to train. The problem of vanishing gradients in backprop algorithm. The chain of multiplication becomes bigger. Non-Convex optimisation

Learning high-level features - data representation

We expect Machine to learn distinct representation. Potential to learn multiple levels of representation in DL algorithms.

Caveat: Extracting low-level specific to the problem domain helps a lot.

Good predicts are important but so are data representation.

What is depth in ML?

The number of levels of composition of non-linear operations in the functions learnt

Ok why go deep?

Expressive power and compactness of model. (Expressibility vs efficiency)

Enchances generalization, especially with limited training examples. less degree of freedom when handling complexity and non linearity. - Exponential gain

Also if we have shallow but super "bredth" network then the model becomes too costly to train. Also because we take inspiration from the brain.

Facilitate Multi-task learning and Transfer learning

- Transfer learning: You train a model for one task then take the learned weights and train it on a similar data → It will converge faster

- Multi-task learning - Simultaneously improve learning across multiple related tasks by leveraging shared information and representation

Representation Learning

↳ Learning features as part of DL algorithms

↳ Multiple levels of abstraction and complexity

↳ Multi-task or Transfer Learning

↳ Facilates non-local generalization (multi clustering)

↳ Sparse Coding

General theme of early Deep Learning Protocol - Deep Belief Network, Stacked Autoencoder

↳ Greedy layer-wise unsupervised pre-train. & Supervised tuning

- Pretrain minimizes Variance
- Helps to Control Complexity for architecturals with large sets of hidden layers
- Acts like a penalisation term ↳ Regularisation
- Pretraining finds better "initial" condition for further gradient based optimisation

What's the fate of Pre-training now?

Less common as you have ReLu, Dropout. Only use with Deep Belief networks

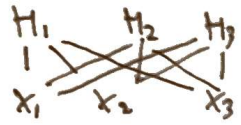
Fundamental Network Architectures

Restricted Boltzman Machines — Greedy-wise unsupervised training which is increasingly omitted once ReLu units are employed

Autoencoders

Convolutional Neural Networks — Network can be initialised without any pretraining. Transfer learning is often explored

Restricted Boltzmann Machine



simple energy model

The idea is to optimise log-likelyhood with the use of approximate Gibbs Sampling - Constrative Divergence /Hyor.

x_i and h_i are binary Random Variables

Visible and Hidden units are conditionally independent given one another:

$$P(h|v) = \prod_i p(h_i|v)$$

$$P(v|h) = \prod_j p(v_j|h)$$

To maximize logg likelyhood do Gibbs sampling, Contrasive Divergence does not optimize the likelihood but it works effectively. You are trying to fit the model than the data

Gibbs Sampling

1. Clamp the visible units with an input vector and update hidden unit

$$p(h_j = 1 \mid v) = \left(1 + \exp\left(-\text{bias}\, h_j - v^T w_j\right)\right)^{-1}$$

2. Update all the visible unit in parallel to get reconstruction

$$P(v_j = 1 \mid h) = \left(1 + \exp(-\text{bias} - w_j h)\right)^{-1}$$

3. Collect the stats for correlation after k-steps using mini-batches and update weights

$$\Delta w_{ij} = \frac{1}{N} \sum_{n=1}^{N} v_j^{(n)} h_j^{(n)} - \hat{v}_j^{(n)} \hat{h}_j^{(n)}$$

Generative vs Discriminate approach

Generative

- Describe Statistical distributions of data and associated classes $P(x,y)$
- Characterise higher-order correlational structure of data for pattern analysis
- Energy based models including auto-encoders

Discriminately

- HMM, CMM, DGN, DNN
- Characterising the postener dist $P(y|x)$
- Tells you if it is a cat or not

- Hybrid deep approach

- Good is discrimination but helped by outcomes of generative modelling in deep architecture

Cheap learning (Why does Deep Learning seem to work?)

- Exponentially fewer parameters than "generic" degrees of freedom
- We take advantage of the special nature of problems at hand

No flat theorem: Deep architectures are better, computationally, compared to shallow but bredth network.

Hierachical Structure: Structure of the physical world. Hierarchy of the objects and hierachy of generative process to untangle

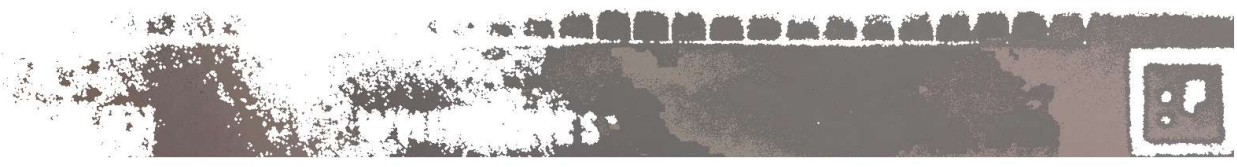
Key challenges

(1) Too much theory but cannot implement it. Cannot fully understand uncertainty bounds and what causes them in the network

(2) Interpretability: We have no idea how these neurons actually works

(3) How can we process the learning such that we can monitor and controll it better

(4) Too god damn costly

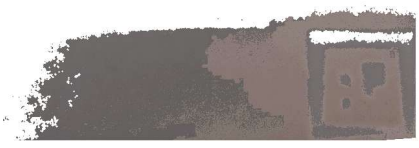


CNN networks

Convolutional layers then pooling layer

Convolution layer does convolution, does spatial filtering. Masks or weights that moves. They extract complex stuff by adding more convolution. Relies on spatial correlation.

Pooling layer downsamples spatial dimensions which reduces overfitting and computational costs as well.



Lecture 10 - Representation Learning

Why do we think Deep Learning powerful

- Expressively and efficiency
- Cheap learning, no flattening hierarchical struct
- Multiple levels of abstraction and hierarchical brain organisation

Algorithmic development

↳ Regularisation techniques ↳ Hyperparameter selection techniques

Learning representations as a hallmark of DL

↳ Deep learning automatically learns representations on their own. Has a lot of implications

- Multitasks
- Multiclustering
- Zero-shot learning

Why are representation learning important

↳ Trade-off between minimising "information" loss and obtaining "nice" property.

They are crucial as they determine how the data is understood and manipulated by the model to make predictions or decision.

Distributed Representation versus Local Representation

Information is distributed across many units that accounts for information about features that are not mutually exclusive.

Local : Each unit in the network is responsible for one specific feature or concept

Distributed Representation : Information is encoded among different units in the network

More recap on Deep learning (lecture 8-11 more better recap)

Multitask learning: Learning paradigm where single model is trained to perform multiple tasks simultaneously. Instead of seperate models for each task, one model learns it all.

Transfer learning: Transferring knowledge from one task into another related tasks. Instead of training from scratch, you load the trained model and fine-tune it.

- **Why care about representation?**

Learning data representations offers several advantages

- Improved performance: Models that learn distributed representations perform better than sparse models or symbolic representations
- Dimensionality Reduction: Often lower dimensionality compared to original data
- Semantic understanding: They encode semantic information about data allowing models to understand the underlying meaning and relationships

Learning representations is the holy grail of Machine learning. You can save a lot of space and computational power by representing data correctly.

Super vs unsupervised approach

Supervised - Distilling information relevant to concrete task defined by label. The issue of Supervised learning is that they rely heavy on abundance of labelled data. Panda + noise = Totally wrong image of Panda. ← With supervised learning you might not always catch the representation you want. → This has made unsupervised learning popular ⟶ Learn the world before the task

Unsupervised learning - Learn representation before doing anything else. You learn through statistics why the data acts the way it does

You can do representation learning through a probabilistic perspective

↳ learn probability distribution for data with the use of latent variables (PCA, ICA, GMM etc) → Explain data

↳ P(data | latent var) for generation P(latent var | data) for Recognition

Zeroshot learning: Model is trained to recognize or classify objects or without any direct supervision or labeled examples for some of the classes.

The proffessor will go and talk about Deep Generative modelling.

Autoencoder (Stacked): The objective is to extract/learn representation (latent code) of data without any labels. Consists of an encoder and decoder function. They first compress data into lower dimension then reconstructs it back to the original input form

Why autoencoder over PCA: They are non-linear and does not assume Gaussian data which allows them to capture more complex data

Undercomplete Autoencoders: - Autoencoders with hidden layer dimension less than input are called this.
- Information bottleneck is realized by the lower dimensionality of the hidden layer than that of input & output - No need for explicit regularisation term

Overcomplete Autoencoders: Hidden layer dimension higher than input data. ↳ Risk of overfitting

Sparse Autoencoder: Hard penalty $\mathcal{L}1$-Regularisation on autoencoder. They aim to learn where only a small subset of neurons are firing or "active"

Denoising Autoencoders: Denoising autoencoders are fed with corrupted versions of the input data. The autoencoder is trained to reconstruct the original clean input data from these corrupted inputs

Variational Autoencoders: Generative model that extend the capabilities of traditional Autoencoders to learn probabilistic representations of data. Designed in a way to learn <u>latent space representation</u> of input data in such a way that it allows for generating new samples.

$x \rightarrow P(z|x) \longrightarrow z \longrightarrow P(x|z) \quad \hat{x}$

latent space representation

VAE is a probabilistic twist on an autoencoder with regularised training:
- ↳ To minimise overfitting
- ↳ To ensure "nice" properties of latent space

<u>Variational Autoencoder</u> replaces deterministic hidden layer (latent) z with stochastic sampling

The aim is to estimate a probabilistic encoder $P(z|x) = \frac{P(x|z)\,P(z)}{P(x)} = \frac{P(x|z)\,P(z)}{\int P(x|u)P(u)du}$

Done by $\text{Min}KL(q(z|x) \,||\, P(z|x))$

$\downarrow$

$\max_{q(z|x)} \log P(x|z) - KL(q(z|x) \,||\, P(z)) \leq \log(P(x))$

Long story short try to maximize $P(z|x)$

Bidirectional feedback imposes extra challenges that need to be solved

→ Synchronous vs Asynchronous

→ Different properties depending on updating mode!

Concept of energy in BAM

If a point is stable, then nearby points should converge (x_0, y_0)

$$\vec{y_0} = Sgn(W\vec{x_0}) \quad \text{next} = \vec{e} = W^T\vec{y_0} \text{ (next value)}$$

For convergence we expect

$$Sgn(\vec{e}) = \vec{x_0}$$

Hopfield network – Memory patterns → Frontend or Deep learning

You are looking for stable points such that "energy" is low in the network. Once the network is stable then no activity is done. That means that the network "memorized" all the patterns. 2^n where n is the nodes is the amount of states. However important that among all the possible states some are stable.

Spurious States => You end up in a stable state but that solution is not a good one. They are problematic, you don't get good memory patterns

Catastrophic forgetting => Your model suddenly forgets all memories, completely everything