DD2437 Artificial Neural Networks and Deep Architectures

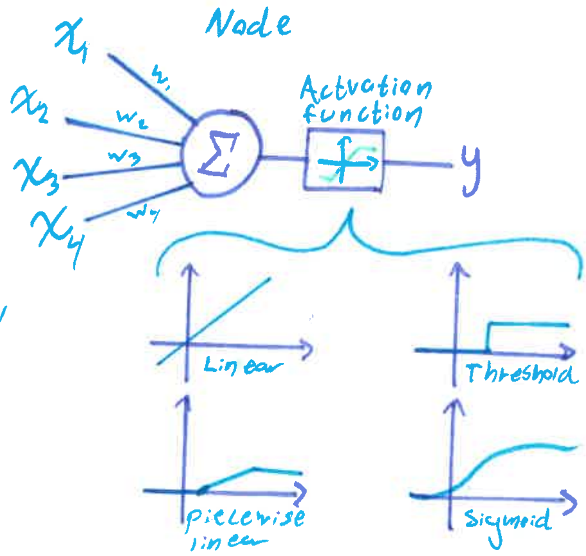
Fundementals

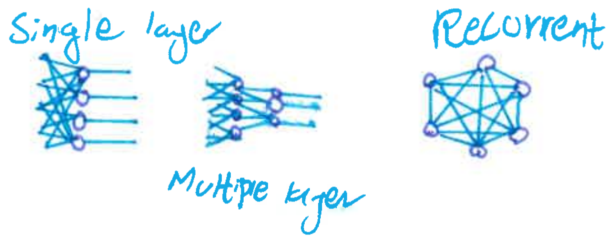
Node - Smallest computational Unit.

Activation function - When should a Node "Fire"

Learning Rule - How should success And fails be handeled

Topology/Architecture - How the networks is structured





Learning Principles

Supervised - Input with labels

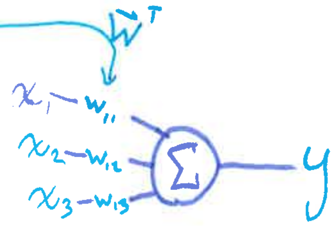
Unsupervised - Input without labels.

Reinforcement - Reward given on success.

Linear Networks

Weight matrix

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \qquad W_{xy} \quad x = \text{sender}, \; y = \text{receiver}$$



$$\vec{W}^T \cdot \vec{x} = y$$

$$W \times \vec{x} = \vec{y}$$

Note: chained single layer networks with weights W_1, W_2, W_3 have a total weight:

$$\vec{y} = W_3(W_2(W_1\vec{x})) = \underbrace{(W_3 W_2 W_1)}_{W}\vec{x}$$

in short, this is just W another single layer network.

Hebbs Learning Hypothesis

Neurons which activate together should strengthen their connection.

$$\Delta W_{ji} = x_i y_j$$

if sign of x y is the same their product is positive, else it is negative.

" Fire Together Wire Together "

Same as

$$\Delta W_{ji} = (x_i - \underset{\text{mean}}{\bar{x}})(y_j - \underset{\text{mean}}{\bar{y}})$$

Storing Mappings

$$W = \sum_{p=1}^{n} \vec{y}^{(p)} \cdot \vec{x}^{(p)T}$$

$$y_{out} = W\vec{x}^{(K)} = \sum_{p=1}^{n} \vec{y}^{(p)} \left(\vec{x}^{(p)T} \vec{x}^{(K)}\right)$$

$$= y^{(K)}\left(\vec{x}^{(K)T} x^{(K)}\right) + \underbrace{\sum_{p \neq K}^{n} \vec{y}^{(p)}\left(\vec{x}^{(p)T} x^{(K)}\right)}_{\text{Close to zero if } x^p \text{ and } x^k \text{ are orthogonal.}}$$

$$\approx \alpha y^K$$

Preception Learning

1 Data is correct; Do nothing.

2 Data is false positive $\Delta W = -\vec{x}$

3 Data is false negative $\Delta W = +\vec{x}$

Convergence

If solution exists it will be found in finite number of steps.

Delta Rule (Widrow Hoff)

While preception learning will always terminate with linearly seperable data, it is not always the most intuetive solution. Delta rule will always converge on an intuitive solution by minimizing an error function.

1 Symetric targeting values $\{-1, 1\}$

2 Error is measured before threshold $e = t - \vec{W}^T\vec{x}$

3 Minimize weights according to $\mathcal{E} = \frac{e^2}{2}$ (steepest decent)

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = -e\vec{x}$$

$$\Delta \vec{w} = -\eta \frac{\partial \mathcal{E}}{\partial \vec{w}} = \eta e \vec{x}$$

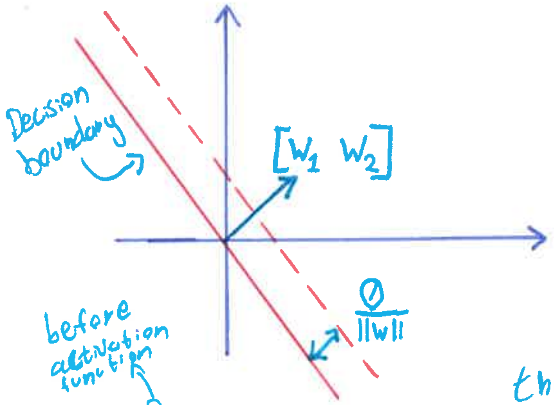
learning rate

Threshold Logic Unit

McCulloch Pitts neuron (1942)

Introduction of the threshold function.

Add function to output to map some values to 0 and some to 1. Also creates a decision boundary.



before activation function

$$y' = w_1x_1 + w_2x_2 \ldots w_nx_n - \theta$$

threshold

$$0 = w_1x_1 + w_2x_2 \ldots w_nx_n + y'$$ Plane equation

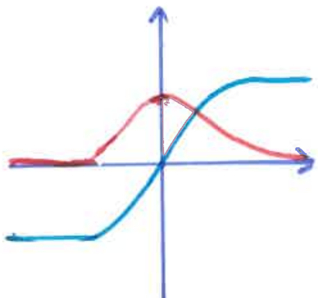
θ becommes an additional input to the node and acts as a bias.

Multilayer Networks

Dillemma:
> Thresholding destroys information heeded for decide.
> No thresholding loses functionallty of multiple layers

Solution:
Use threshold-like but differentiable function.

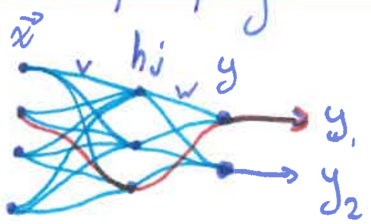
φ φ'



$\varphi(x) = \frac{1-e^{-x}}{1+e^{-x}}$

$\varphi'(x) = \frac{1-\varphi^2(x)}{2}$

Backpropagation



1 forward pass; compute h_j & y_k

$$h_j = \varphi\left(\sum_i \vec{v}_{ji}\vec{x}_i\right) \quad y_k = \varphi\left(\sum_j \vec{w}_{kj}\vec{h}_j\right)$$

2 Backward pass; compute δ_k & δ_j

$$\delta_k = (t_k - y_k)\cdot\varphi'(y_k^{in})$$

$$\delta_j = \sum_k \delta_k \cdot w_{kj}\cdot\varphi'(h_j^{in})$$

Convinient if φ' can be expressed in terms of φ

3 Update; ΔW & ΔV

$$\Delta w_{kj} = \eta\delta_k h_j \quad \Delta v_{ji} = \eta\delta_j \vec{x}_i$$

Problems:

Convergence is slow
getting stuck in local minima.
Many parameters must be tuned
Challangeing scaling
Biologically unrealistic.

Sequential & Batch

Sequential → stocastic gradient decent

Either error is accumilated and then updated or after every sample.

Sequential:
faster but needs higher learning rate.
less likely to get stuck in local minima.
Is not true gradient decent

1 coose data x
2 feed and get error e
3 calc update ΔW

sequential → Apply update

Batch → accumilate ΔW → apply update

epoch

weight should be randomly initialized

$N(0, \sigma)$

$\sigma = \frac{1}{\text{fan\_in}}$ ← number of inputs for node

Learning Rate η

too low → learning slow
too large → overshoot convergence
Adaptive → $\eta(t) = \eta(1)/t$ or $\eta(t) = \eta(1)/(1+t/\gamma)$
↳ depending of sign of consecutive iterations

Momentum $\Delta\vec{w} = \beta\Delta\vec{w} - (1-\beta)\frac{\partial e}{\partial w}$
↳ previous update affects new update
keeps old changes for a while.

overrepresenting hard data.
Adding noise during training.
Hyperparameter selection.

Generalization

Our model should work on data never seen before.

Key factors:
↳ quality & size of training data
↳ complexity of model
↳ complexity of problem

Classification RBF

Training for MLP involves changeing weights and biases.

Training for RBF means changeing locations and radii

Cover's Theorem

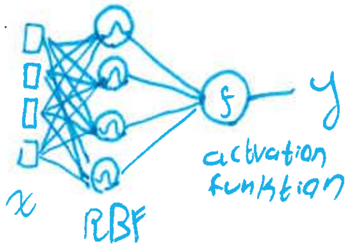
Projecting a pattern to a higer dimention will cause problems to be solved easily.

RBF

Use non-linear mapping function.

$\varphi_i(\|\bar{x} - \bar{x}_i\|)$ → RbF centre

$$F(x) = \sum_{i=1}^{N} w_i \varphi(\|X - X_i\|)$$



Competitive Learning

Winner takes all.
node closest to the input Pattern learns by the rule.

$\Delta W_a = \eta \vec{x}$

Pos of RBF

or

$\Delta W_a = \eta(\vec{x} - W_a)$

Learning Algorithm.

fundemental principle:
Update winning node to make it more specialized.

Algorithm.

- Initialdize clusters randomly $W_A \dots W_C$
- For each input vector, find winner weight (in the input space)
- update weight using rule.

Result
clusters in data is found and protected by nodes

Problems
Dead or unuseful nodes.

Vector Quantization

Represent data in terms of a limited number of typical data vectors.
>compression
>noise Reduction.

Spills-over Learning

When learning, nodes connected to winning nodes (neigbour) also learn a bit.

$\Delta W = \eta h(\vec{x} - W)$

Spill rate depening on neigbourness

Self Organizing Maps (SOM)

1 competition } output space
2 cooperation }
3 weight updates } input space

↳ Topological ordering
↳ high learning rate
↳ large neigborhood
} global Order

↳ Convergence
↳ low learning rate
↳ small neigborhood
} Local fit

Important!

neighbours are determined by the output space not input Space
(winner is still in the input space)

Learning Vector Quantization

Supervised version of VQ when classes are known.

$\Delta \vec{W}_i = \eta(\bar{x} - \vec{w}_i)$ when classified correctly

$\Delta \vec{w}_i = -\eta(\bar{x} - \vec{w}_i)$ when classified incorrectly

Hopfield Network (hebbian)

All nodes are connected to every other node.

$$\vec{x} = \mathrm{sgn}(w x + \theta)$$

$$E(\vec{x}) = -\frac{1}{2} \sum_i^n \sum_j^n w_{ij} x_i x_j + \sum_i^n \theta_i x_i$$

Asynchronious update:

$$x_i(t+1) = \mathrm{sgn}\left(\sum_j^n w_{ij} x_j(t)\right)$$

i is random

$$\Delta E_{x_i \to x_i^*} = -\frac{1}{2}\left(\sum_j^n w_{ij} x_j x_i^* - \sum_j^m w_{ij} x_i x_j\right)$$

$$= -\frac{1}{2}(x_i^* - x_i) \sum_j^n w_{ij} x_j \leq 0$$

Synchronics can flip

Memory capacity

Memory capacity M is proportional to $0{,}138 n$ (nodes)

$$M \leq \frac{n}{4 \ln(n)}$$

Sparse patterns $n \times \log(n)$

Associative Pattern Recognition

Hetero Associative: boat → water
Auto Associative: boat ↻
P.R. networks: boat1 → boat2

Works when $\vec{x}$ are orthogonal

Auto associative case:

$W = XX^T$

$\mathrm{sgn}(W\bar{x}) = \bar{x} \rightarrow$ $\bar{x}$ is eigenvector of W

W describes non-orthogonal projection of subspace spanned by $\bar{x}$

Resonance, Energy (BAM)

$\vec{y}_0 = W\vec{x}_0 \qquad \vec{e} = W^T\vec{y}_0$

how far is e from x?

$$E = -\bar{x}_0^T \vec{e} = -\bar{x}_0^T W^T \bar{y}_0 = -\vec{y}_0^T W \vec{x}_0$$

Converges if W is symetric
Memory state are fixed point attractor

Batch mode is faster

Pattern Recognition Pipeline

Data → Preprocessing → Feature Analysis → Classification / Regression → Postprocessing → output

Preprocessing

- Familiarize yourself with Data → Prior knowledge, How it was generated, Type of atributes, Test & training same distribution
- Quality assesment → Dimentions & amount
- De-noising → Info abot noise/remove noise, outlier detection/removal
- Transformation/normalization → Scaling/normalization
- Missing Data → Replace with mean, generate missing atributes
- Data Augmentation → up/down sampling

Feature Analysis

- Dimentionality reduction → PCA, SOM, ICA, Decorrelation
- Transformation to new space
- Feature Selection

Classification / Regression

Choosing correct error and loss function → relu, sigmoid
- ↳ Depends on: problem type (regression/classification)
- ↳ Expected value in output layer (function/num outputs)
- ↳ learning algoritm

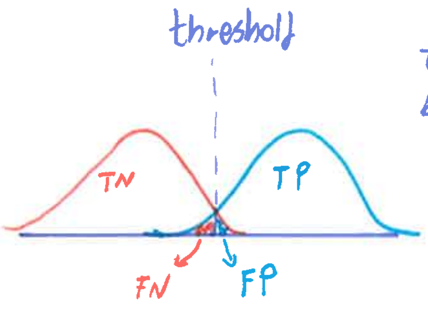
Maximum likelyhood
↳ How close the distributions of predictions by model match with data
- ↳ Cross-Entropy between emperical distrubutions defined by training set, and Probability distrubution of model.
- ↳ MSE is the same for regression

Cross Entropy
↳ sigmoid activation of output

Binary classification → $L(y,t) = \sum_{i=1}^{N} t^i \log y^{(i)} + (1-t^i)\log(1-y^i)$

multiple class → $L(y,t) = \sum_{i=1}^{N} \sum_{c=1}^{C} t_c^i \log(y_c^i)$

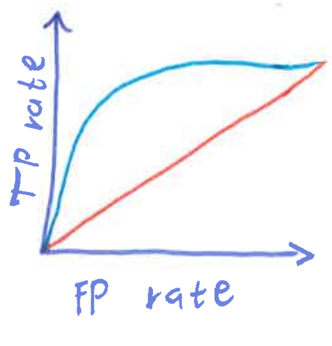
Receiver Operator Characteristics



Threshold can be moved to decrease FP or FN, but increase the other.

$$\text{Specificity} = \frac{TN}{TN+FP}$$

$$\text{Sensitivity} = \frac{TP}{TP+FN}$$



Why deep learning

Deep learning performs better with more data while other learning plateues eventually.

Depth - longest path from x to y

Layer-Wise Training

Only Emperical proof

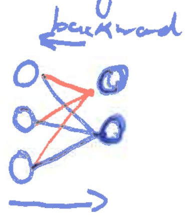
Train one layer at the time while keeping others locked. Unsupervised.
Then tune with backprop.

2010 Pretraining minimizes varianse
Controls complexity
implicit penalization (Regularization)

2007 Pretraining has better initial condition
training of entire architecture

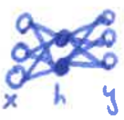
RBM

Two layer autoencoder



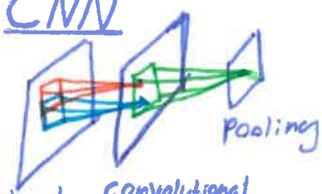
features are extracted in hidden layer and then reconstructed in backward pass

Autoencoder



Unsupervised (still gradient desent)
x should equal y despite dimention reduction.

CNN



Multiple layer allow for hirarical feature extraction.
Multiple sampling.

Problems

- Vanishing gradients in backprop algorithm with more layers added.
- Unstable gradients
- local minima
- overfitting

2006
↳ pre-train deep Architectures with layer-wise unsupervised learning.

Classical Architectures

Restricted Boltzmann Machine
↳ Contrastive divergence for pre-training

Autoencoder
↳ gradient decent based alg for pre-training

Convolutional Neural Network
↳ No need for pretraining
↳ often uses transfer learning

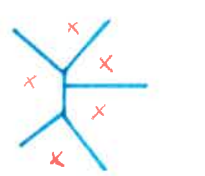
Representation Learning

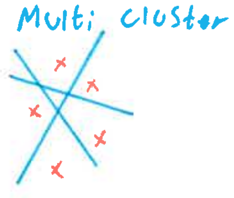
Many levels of representation allow multitask-learning

Distributed representation

$P(\text{data} \mid \text{latent})$ - generation
$P(\text{latent} \mid \text{data})$ - recognition

↳ learn features
↳ layers of abstraction
↳ multitask/transfer learning
↳ non-local generalization (sparse code) (multi clustering)





Class is not determined by a single feature.

Residual activity prevent overfitting

Filters convolve (slide) across input to analyse said input and sub sample

Distributed vs local representation

Distributed

Information is spread across multiple units

Represents similarity between objects

More easily seperatable (orthogonal)

implications
↳ Transfer learning
↳ Multitask
↳ zero shot

Supervised

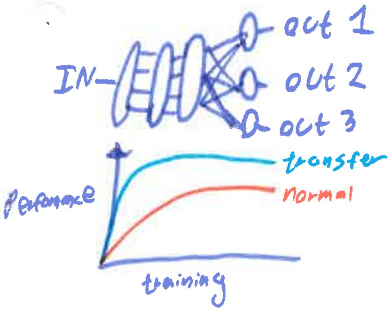
Very useful at solving specific tasks
Relies on labeled data
Information bottleneck
Lacking "common sense"

Unsupervised

Learning world before task
Less susceptable to information loss

Transfer learning

A model developed for one task is reused as the starting point of another

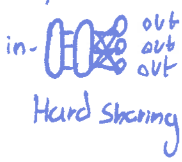


Development mode
Pre-trained model

higher slope
higher asymptote

Multitask learning

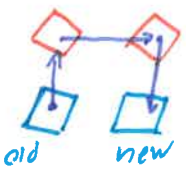
Colearn multiple tasks simultaneously
Improves generalization by using domain specific information contained in training signals



Zero-Shot learning

Recognize objects from unseen classes.

Available:
↳ seen data (labels)
↳ Unseen data (no labels)
↳ Auxiliary information from seen and unseen classes
↳ bridge between seen and unseen



Autoencoders

$Loss(x, g(f(x)))$

Encoder $x' = f(x) = W'h$
Decoder $h = g(f(x)) = Wx$

Undercomplete
↳ hidden layer < input

overcomplete
↳ hidden layer > input
↳ needs regularisation
↳ sparse & denoising
↳ Deep autoencoder

} Fixed by adding latent variable with a prior and maximise likelyhood

Generative modeling

Objective is to capture hidden structure in data

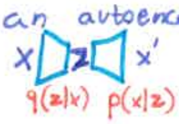
Generative Adversarial network
↳ Zerosum game between generator & discriminator

Real data → D → Label $p(x \text{ is real}) = d(x, \theta^{d})$

Noise → G → Fake data $x = g(z, \theta^{gen})$

Variational Autoencoders

an autoencoder with generator capabilities



last layer of q define mean and variance of latent space

Vector aritmetic in latent space

Linear Networks.

Supervised learning

$y = \vec{W}^T \cdot \vec{x}^T$

Stacking multiple does not work (becomes new liear mapping)

Storing Mappings

↳ Program Resides in weights

↳ Learning Adapts weights

Hebbian Learning

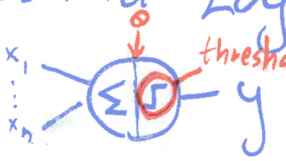
↳ $\Delta w_{ij} = x_j y_i$

↳ Fire together wire together.

↳ $\Delta W = \vec{y} \times \vec{x} = \vec{y}\vec{x} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} [x_1 \; x_2 \cdots x_n] = \begin{bmatrix} x_1 y_1 & \cdots & x_3 y_1 & \cdots \\ x_1 y_2 & \cdots & x_3 y_2 & \cdots \end{bmatrix} = \begin{bmatrix} \Delta w_{11} & \cdots & \Delta w_{15} \\ \Delta w_{21} & & \Delta w_{25} \end{bmatrix}$

↳ Remember orthogonal patterns

Threshold Logic Unit

↳ 

$y' > 0 \rightarrow y = 1$

$y' <= 0 \rightarrow y = 0$

0

Decision Boundary

$W_0 = 0$ Threshold or Bias

↳ Preceptron learning

↳ update weights depending on classification correctness

↳ Result=0, label=1: $\Delta W = \eta \vec{x}$ (learning rate) Result=1, label=0: $\Delta W = -\eta x$ Correct classifcation do nothing

↳ Always converges for linearly seperable data.

↳ Delta Rule

↳ Results $\in \{-1, 1\}$, $error = t - \vec{W}^T \vec{x}$ minimize $\mathcal{E} = \frac{e^2}{2}$

↳ Steepest descent move down in gradient $\Delta \vec{W} = -\eta \frac{\partial \mathcal{E}}{\partial \vec{w}}$ $= \frac{\partial}{\partial w} \mathcal{E}(e(w)) = e\dot{e}$

↳ Finds lowest erron (not neccesarely correct classification)

$\frac{\partial}{\partial w} = -ex$

$\Delta w = \eta e x$

↳ Cannot solve xor problem

Hopfield Networks

Learn and remember patterns. Autoassociative ~~$Y = WX$~~ $\vec{x}(i+1) = Wx(i)$

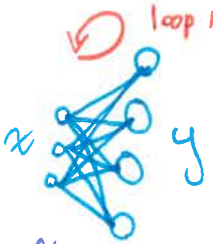
recurrent network

Bipolar encoding $x \in \{1, -1\}$

$$y = Wx_p = \sum (y_k x_k^T) x_p = \sum y_p (x_p^T x_p) = y_p (x_p^T x_p) + \sum^m \vec{y}_k \underbrace{(x_p^T x_k)}_{0 \text{ if pattes are orthogonal}}$$

Hebbian learning

loop many times



$W = XX^T$

↳ matrix of pattens

after many iterations x will converge to an "attractor"

$sgn(W\vec{x}) = \vec{x}$ $\quad sgn(WX) = X$

↳ $\vec{x}$ are eigenvectors

$\forall_i w_{ii} = 0$

Bidirectional Associative memory

$$E(\vec{x}) = -\frac{1}{2} \sum_{i,j}^{n} w_{ij} x_j x_i + \emptyset$$

energy

Will always converge since energy decreases (when w is symetric)

$$\Delta E(x_j \to x_j^*) = -\frac{1}{2}(x_j^* - x_j) \sum_i^n w_{ij} x_i \geq 0$$

opositos of attractors are stable

[-1] & [1]

Synchronious & asynchronious

given patterns $x_1 \; x_2 \; \dots \; x_n$

$\hat{x}_k$

$$W_{ij} = \underbrace{\begin{cases} \frac{1}{n} \sum_{k=1}^{n} x_{ki} \cdot x_{kj} \\ 0 & \text{if } i = j \end{cases}}_{\text{Hebb lerning}}$$

Patterns must be orthogonal for best perfomance

$M \leq 0.138n$ (patterns; nodes)

sparse patterns $\quad n\log(n)$

$$M \leq \frac{n}{4 \ln n}$$

one-shot learning

Temporal Processing

Descrete & continious. Time domain data is important

MLP

Time lagged feed forward network

↳ Multiple different times in the input

↳ only some exact times are allowed

↳ gives ability to add <u>static nodes</u>

Recurrent Architecture

Backprop through time

↳ Multiple layers tacking to each other

$x \to o \to o \to o$
  ↳ $o \to o$
  ↳ $o \to o$

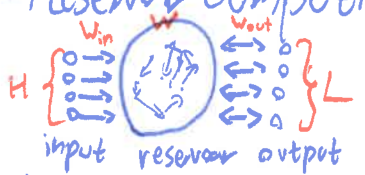
For training the entire time series is needed since you cannot know future samples.

Kalman filters

Vanishing gradient (very bad) since w is multiplied many times and eigenvalues λ are usually less then 1 so $\lambda^n \to 0$ for large n.

Echo State Network

↳ Reservoir computing



Reservoir is <u>Very</u> large 10 k nodes

sparse 2-20% of nonzero elements

spectral radius $\rho < 1$ forgetting speed

Nodes have leaky memory modulated by α $h(t) = \alpha h(t-1) + (1+\alpha) x(t)$

One shot learning with least squares (LMS)

fill and activate reservoir with $x(n)$, compute $h(t)$, fit y wit LMS

↳ <u>Long Short-term Memory</u>

Very good, hard to understand

backprop is possible because everything is differentiable

Multilayer Networks

$\varphi(x) = \frac{1-e^{-x}}{1+e^{-x}}$ $\varphi'(x) = \frac{1-\varphi^2}{2}$

$\varphi(x) = \frac{1}{1+e^{-x}}$ $\varphi'(x) = \varphi(x)(1-\varphi(x))$

$\varphi = \tanh(x)$ $\varphi'(x) = 1-\varphi^2$

$\varphi = \arctan(x)$ $\varphi' = \frac{1}{1+x^2}$

Instead of thresholding use differentiable function

1 Define error function $\varepsilon = \frac{1}{2}\|\vec{t}-\vec{y}\|^2 = \sum_k (t_k - y_k)^2$

2 Minimize ε with steepest desent (towards minima)

$$\Delta w = -\eta \frac{\partial \varepsilon(\vec{x}, \vec{w})}{\partial \vec{w}} \qquad \Delta w_{kj} = -\eta \frac{\partial \varepsilon}{\partial w_{kj}} \qquad \Delta v_{ji} = -\eta \frac{\partial \varepsilon}{\partial v_{ji}}$$

Error of previous layer depends on the next. however it diminishes with each layer (vanishing gradient)

$$\delta_k = (t_k - y_k) \cdot \varphi'(y_k^{in}) \qquad \delta_j = \sum \delta_k \cdot W_{kj} \cdot \varphi'(h_j^{in})$$

$$\Delta W_{kj} = \eta \delta_k h_j \qquad \Delta V_{ji} = \eta \delta_j x_i$$

1 forward pass: compute each $\underbrace{\text{hidden layer}}_{h_j}$ and $\underbrace{\text{output}}_{y}$

2 Backward pass: compute δ_k & δ_j $\delta_k = (t_k - y_k) \cdot \varphi'(y_k^{in})$ $\delta_j = \sum_k \delta \cdot W_{kj} \cdot \varphi'(h_j^{in})$

3 Update weights: $\Delta w_{kj} = \eta \delta_k h_j$ $\Delta v_{ji} = \eta \delta_j x_i$

Sequential

- update w after each sample
- faster but needs lower learning rate
- Less likely to get stuck in local minima
- not true gradient descent, but rather stochastic gradient descent

Adaptive Learning Rate

- num iterations $n(t) = \frac{n(1)}{t}$ $n(t) = \frac{n(1)}{1+t/N}$
- Momentum $\Delta W = \beta \Delta W_{\text{old } W} - \underbrace{(1-\beta)\frac{\partial \varepsilon}{\partial \vec{w}}}_{\text{small change}}$
- antisymmetric activation function converge faster.

Radial Basis Function $R^M \to R^N \to R^1$ $N > M$

Seperate mappings that are nonliear after mapping them to a higher dimension using a linear separability

$\varphi_i(\|X - X_i\|)$ gaus of distance from X_i

gauß RBF centre

linear operation in N dimentional space: $F(x) = \sum_i^N w_i(\text{RBF})$

Unsupervised learning, mainly for clustering

Number of RBF need to be less than num samples but larger than input size. ⇒ more samples than dimention is needed

Winner takes all, winner is the closest RBF to the input data $\|x - w\|$

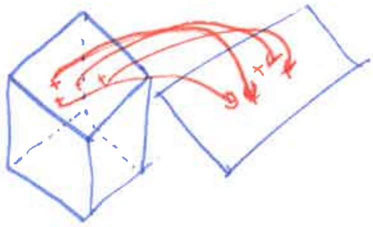
Dead units are units that never win. (Fixed by reinitilize, or give a node a point to start at)

Self Organizing Map

hyperparameter: neighbourhood size
learning rate

Vector Quantization - let single vector represent a large area
↳ compression / Noise reduction.

When a node wins it will spill over to its neighbours in the node space. This means nodes close in input will remain close in output despite (output space) the dimention up or down scale.



Learning Vector quantization

↳ $\Delta W = +\eta(\vec{x} - \vec{w})$ when correctly classifed $\Delta W = -\eta(\vec{x} - \vec{w})$ when incorrectly classified

Decreasing learningrate and neigbourhood size

Deep Belief Networks

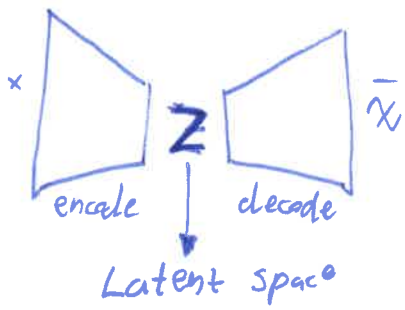
Create a network that maximises $p(x)$ not $p(y|x)$
Make inferences instead of leerning training data.

Stacked RBM → Energy based

Train layer by layer usein unsupervised methods
Tune after, using supervised learning and labels.
↳ works because pretraining minimizes variance
↳ Controls complexity
↳ implicit penalty and acts as regularisation (less susceptible to overfit)
↳ Better initial position in terms of local minima

Contrastive Divergence

Variational autoencoders



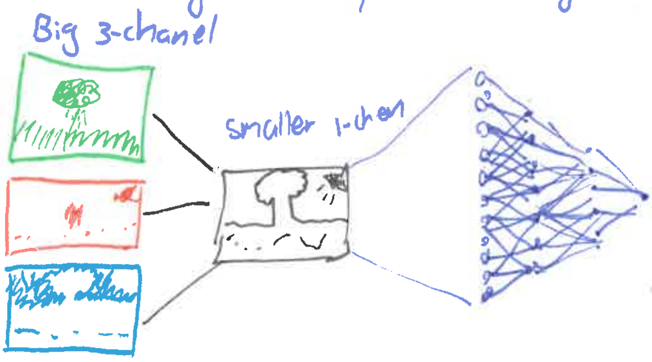
generation capabilitis comes from having probability distribution in the latent space
$P(x|y)$

Vectors in latent space can be decoded into the real space

Undercompleete - Dense
overcompleete - Sparse
↳ avoid trivial connection
↳ L1 regulaization

Convolutional Neural Networks

Take original input (image) and apply filters across. (Convolution)



Representational Hierarchy

Generative Adversarial Network

Real image → Discriminator → label

Generator → Fake image

Probabalistic

Regularisation & Bayesian Techniques

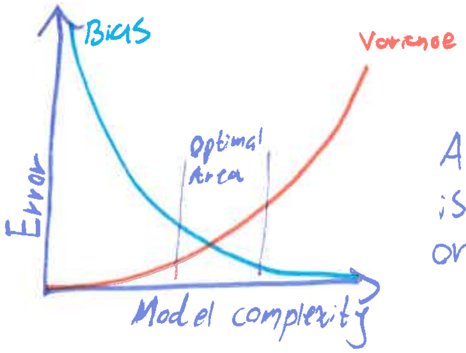
Generalization is the ability to apply learned knowledge in unseen situation
↳ Training size, complexity NN, complexity of problem
↳ Underfitting / overfitting

Bias: Difference between average prediction and the correct value. High bias implies the model oversimplifies the problem and underfits training results giving high error in training

Variance: The variability of model prediction on a given data point. High variance is very good with the training data but does not generalise well on unseen data
overfit

True Relationship $Y = f(x) + \underset{\text{gaussian error}}{e}$ Modelled relationship $Y = f'(x) + e$

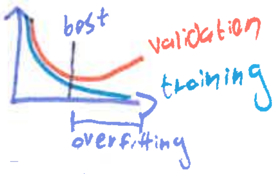
Error (x) $E[(Y - \hat{f}(x))^2] = \underbrace{(E[\hat{f}(x)] - f(x))^2}_{\text{Bias}} + \underbrace{E[(\hat{f}(x) - E[\hat{f}(x)])^2]}_{\text{Variance}} + \underbrace{\sigma_e^2}_{\text{irreducible error}}$



Adding noise in <u>every</u> sample right before it is seen minimises the variance since network cannot overfit to data seen once.

Early stopping

Create an additional dataset called Validation set
Train network on training data until validation error is minimum



Early stopping can aso be based on problem-specific limits.

Network Growing and pruning

Growing - Add more units or layers when specification is not fulfilled
Pruning - remove parts of network based on saliency

Penalised learning

Add a penalty term to error function $\tilde{E} = E + \lambda \Omega$ (Penalty: Ω, factor: λ)

L2 Regularisation

$\Omega = \|W\|^2 = \sum_j w_j^2 \Rightarrow \tilde{E} = \frac{1}{N}\sum_i (t_i - y_i)^2 + \lambda \sum_j w_j^2$

gradient: $\frac{\partial \tilde{E}}{\partial w_j} = \frac{\partial E}{\partial w} + 2\lambda w_j$

L1 regularisation

$\Omega = \|W\| = \sum |W_j|$

$\tilde{E} = \frac{1}{N}\sum_i (t_i - y_i)^2 + \lambda \sum_j |w_j|$

gradient $\frac{\partial \tilde{E}}{\partial w_j} = \frac{\partial E}{\partial w_j} + 2\lambda \, \mathrm{sgn}(w_j)$

Both L2 & L1 penalise large weights

L2 penalty is proportional to the weights size

L1 is constant penalty

L1 leaves few large connection, encourages sparsity

Bayesan Regularisation

bayesan belief is to model underlying (multiple) probability distribution

frequentist belief there is a single true model and data is just realization

Bayesan: probability of hypothesis given data : can assume a distribution

Frequentist: probability of data given hypothesis : only real data is relevant

$P(H/D) = \frac{P(H)\,P(D|h)}{P(D)}$

Ensemble learning

Dropout - approximative bagging, drop some nodes randomly

n-fold Cross-validation -> group data into n-subsets, train on n-1 samples and use the nth sample for validation, repeat n times and use avg test error

Ensemble learning -> combine multiple networks, assumes networks are independent → Decrease Variance

↳ Boosting -> Treat previously misclassified samples more serious

↳ on different weak learner

Bagging reduces variability among weak learners

Network	Algorithm	Properties	Aplplication
Preceptron	Preceptron learning	always converges for linearly sepecble, Premature termination	linear classification
Multilayer Preception	Backprop with Deltarule	Local minima vanishing gradient, supervised global classification	
Radial Basis Function	Position RBF with winner takes all. Delta rule train.	Clustering, classification	
Self Organizing Maps	Competitive cooperative learning, with map in outputspace	neighbourhood preserving not distance preserving. vector Quantization	
Hopfield Networks	Hebbian learning Unsupervised	One-shot learning orthogonal vectors energy based	Denoising
Restricted Boltsman Machine	Contrastive Divergence, Gibbssampling	Energy based Stochastic	
Deep Belief Network	greedy-layerwise training, sleep-wake fine tuning	Generative, Discriminative	
Convolutional neural network	Shared weights Convolutional. Pooling backprop	Translational invariance, weight sharing	Image classification
Generative Adversarial Networks	0-Sum game train.	Probability is modeled implicitly	Image generation
Autoencoder	backprop with original as label (not supervised kinda) (MSE)	over/undercomplete sparse Dense	compression Denoise
Variational Autoencoder		Stocastic latent Space generative vector arithmetic in latent space	

Classification
needs many samples
images
CNN
Sigmoidal outputs
(transfer)
Labeled data
inputspace = size of image
Backprop
regression
generalization loss
reduce image size
when few data samples
Cross entropy

Hebbian
Hopfield Memory
Pattern

Approximate capabilities
Cluster
Competitive learning
RBF
feed forward.
Sigmoid output
for classification
Delta rule
for linear W
hyperparameter:
· Num RBF
· width (maybe)
· learning rate

Regression
10-fold cross validation
MLP
L2 regularization
dropout

Sampling without label
SOM
Representations
(no images)

discriminative
p(out|in) ←
Probabilistic (input + output) ← generativ
generate Data
probabilistic
Label in output
DBN
one_hot encoding
Contrastive divergence
Gibbs sampling
contrastive-
wake_sleep tuneing
underlying characteristics
Probabalistic model over data

MSE
Backprop
AE de-noising
Probablistic

Probabilistic (input)
GAN
P(x,y)
Unsupervised
Underlying characteristics

Probabalistic
VAE Unsupervised
backprop
latent variables
Underlying Characteristics

multiple time scales
recurrent
Backprop
LSTM
Mean sq error
L2 regularization
Temporal prediction
& Regression

Size depends on Performance
Time Delayed FFN
MLP
Backprop
Cyclic pattern

Train & validation
RNN
Backprop through
time

Stochastic g.d.
ESN