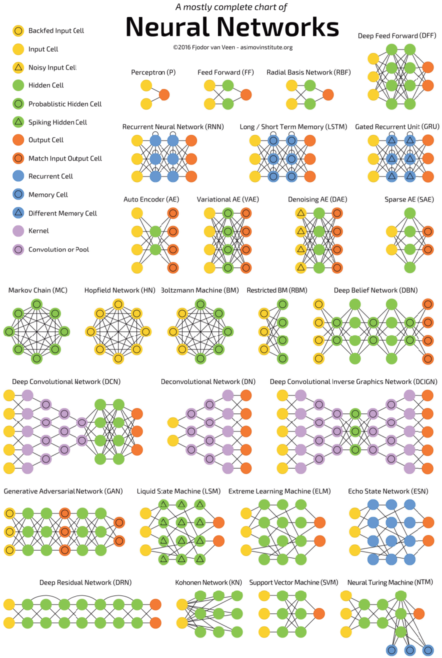
Aalto University

Probabilistic machine learning | Intro (E)

Introduction to machine learning

Francesco Corona

Chemical and Metallurgical Engineering
School of Chemical Engineering


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Intro

Many neural network models are made of neurons

$\rightsquigarrow$ A single lonely ☹ neuron can learn, too

It _must_ be interesting to understand it in detail

Formulation and training

The neuron

Formulation

The neuron

Formulation

Architecture

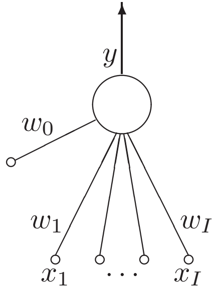
A neuron has a number I of inputs x_i and one output y

Associated with each input is a weight w_i $(i = 1, \ldots, I)$

- There may be an additional parameter w_0
- The bias, the weight for the input x_0
- Input x_0 is permanently set to 1

The neuron is a feedforward computational device

- Connections go from inputs to output



Formulation (cont.)

Activity

The activity of the neuron consists of two steps

1. In response to the presented input $\mathbf{x} = (x_1, \dots, x_I)$, the **activation**

$$a = \sum_i w_i x_i$$

   Sum is over $i = 0, \dots, I$ if there is a bias
   $\rightsquigarrow$ ($i = 1, \dots, I$ otherwise)

2. The output y, or **activity**, is set as a function $f(a)$ of the activation

$$y = f(a) = f(\sum_i w_i x_i)$$

   There are several activation functions
   $\rightsquigarrow$ Deterministic and stochastic

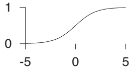
Formulation (cont.)

Popular activation functions (deterministic)

- Linear (identity)

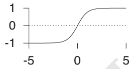
$$y(a) = a$$

- Sigmoid (logistic function)



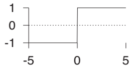
$$y(a) = \frac{1}{1+e^{-a}}, \quad y \in (0,1)$$

- Sigmoid (tanh)



$$y(a) = \tanh(a), \quad y \in (-1,+1)$$

- Threshold (sign)



$$y(a) = \text{sign}(a) = \begin{cases} +1, & a > 0 \\ -1, & a \leq 0 \end{cases}$$

Formulation (cont.)

Popular activation functions (stochastic)

Stochastic activation functions, y is randomly selected from $\{-1, +1\}$

- Heat bath

$$y(a) = \begin{cases} +1, & \text{with probability } \dfrac{1}{1+e^{-a}} \\ -1, & \text{with probability } 1 - \dfrac{1}{1+e^{-a}} \end{cases}$$

- ...

Formulation (cont.)

The neuron implements a function $y(\mathbf{x}|\mathbf{w}) = y[a(\mathbf{x}, \mathbf{w})]$, with $a = \sum_i w_i x_i$

The output y is often a nonlinear function of the inputs $\mathbf{x}$

- The function is parameterised by weights $\mathbf{w}$

The logistic sigmoid

We study a neuron which produces an output $y \in (0, 1)$, as function of $\mathbf{x}$

- We consider the logistic function (sigmoid)

$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

Formulation (cont.)

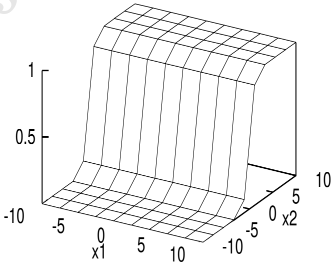
The case where input vector and parameter vector are two-dimensional

$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

- $\mathbf{x} = (x_1, x_2)$ and $\mathbf{w} = (w_1, w_2)$
- The implemented function y

$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}$$

- (Temporarily, no bias)



$\mathbf{w} = (0, 2)$

Along a line perpendicular to the direction of $\mathbf{w}$, the output is constant

Along a line in the direction of $\mathbf{w}$, the output is a sigmoid function

Formulation (cont.)

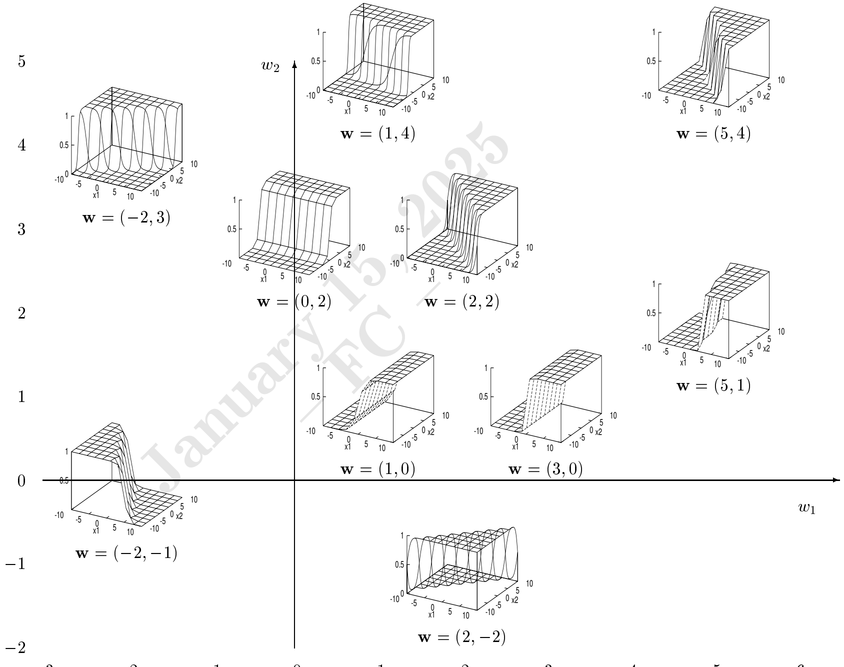
The weight space

The parameter/weight space of the neuron is a space whose dimensionality equals the number of weights and onto which weights can take on values

- Each point $\mathbf{w}$ in weight space corresponds to a function of $\mathbf{x}$

In our case study, there are two parameters (weights) w_1 and w_2

- The weight space is two-dimensional
- We see functions $y(\mathbf{x}|\mathbf{w})$ in place


w_2
w_1
5
4
3
2
1
0
−1
−2
$\mathbf{w} = (1, 4)$
$\mathbf{w} = (5, 4)$
$\mathbf{w} = (-2, 3)$
$\mathbf{w} = (0, 2)$
$\mathbf{w} = (2, 2)$
$\mathbf{w} = (5, 1)$
$\mathbf{w} = (1, 0)$
$\mathbf{w} = (3, 0)$
$\mathbf{w} = (-2, -1)$
$\mathbf{w} = (2, -2)$
x1
x2

Training

The E-uD-uE-uA-uF-AN(u)N

Training

The central idea of a supervised neuron (and also neural networks, I suppose)

$\rightsquigarrow$ Given examples of the relation between input vectors $\mathbf{x}$ and targets t

$$\left\{\left(\mathbf{x}^{(n)}, t^{(n)}\right)\right\}_{n=1}^{N}$$

$\rightsquigarrow$ We wish to make the neuron $y(\mathbf{x}|\mathbf{w})$ learn the map from $\mathbf{x}$ to t

For any given $\mathbf{x}$, a successfully trained neuron will return some output y

- Output is expected to be close (in some sense) to target value t

---

Training the neuron means searching in weight space for some optimal $\hat{\mathbf{w}}$

- A value $\hat{\mathbf{w}}$ that produces a function $y(\mathbf{x}|\hat{\mathbf{w}})$ that fits the data well
- That is, output values y that are close to target values t

Training (cont.)

The error function

A function that measures how well a neuron with weights $\mathbf{w}$ solves the task

Often, the error function is a sum of terms, one for each input/target pair

- It measures how close output $y(\mathbf{x}|\mathbf{w})$ is to target t
- Each term is a function of $\mathbf{w}$, given the input $\mathbf{x}$

Training the neuron is an exercise in function minimisation (optimisation)

$\rightsquigarrow$ Find the $\mathbf{w}$ so that the error (objective) function is minimal

Training (cont.)

Binary classification

We have a lonely neuron whose output $y(\mathbf{x}, \mathbf{w})$ is bounded in $(0, 1)$

- The activation function is the logistic sigmoid

$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

They gave us a set of input data with binary labels

$$\{(\mathbf{x}^{(n)}, t^{(n)})\}_{n=1}^{N}$$

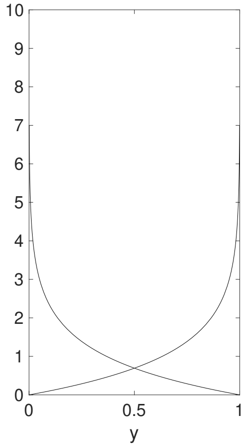
How do we train the neuron to binary classify the data?

- Firstly, we need to define some error function
- Then, we need to find a $\mathbf{w}$ that minimises it

Training (cont.)

We consider the following error function

$$G(\mathbf{w}) = -\sum_n \left\{ t^{(n)} \ln\left[y(\mathbf{x}^{(n)}|\mathbf{w})\right] + (1 - t^{(n)}) \ln\left[1 - y(\mathbf{x}^{(n)}|\mathbf{w})\right] \right\}$$



The term in error function

- Bounded below, by zero
- When $y(\mathbf{x}^{(n)}|\mathbf{w}) = t^{(n)}$
- (for each n)

The neur(on)al model

$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w}\cdot\mathbf{x}}}$$

Training (cont.)

$$G(\mathbf{w}) = -\sum_n \left\{ t^{(n)} \ln \left[\underbrace{y(\mathbf{x}^{(n)}|\mathbf{w})}_{y^{(n)}} \right] + (1 - t^{(n)}) \ln \left[1 - \underbrace{y(\mathbf{x}^{(n)}|\mathbf{w})}_{y^{(n)}} \right] \right\}$$

The derivative $\mathbf{g} = (\cdots, g_j, \cdots)$ of $G(\mathbf{w})$ with respect to $\mathbf{w}$

$$\begin{aligned} g_j &= \frac{\partial G(\mathbf{w})}{\partial w_j} \\ &= \sum_{n=1}^{N} -\left[t^{(n)} - y^{(n)} \right] x_j^{(n)} \\ &= \sum_{n=1}^{N} -e^{(n)} x_j^{(n)} \end{aligned}$$

Quantity $e^{(n)} \equiv \left[t^{(n)} - y^{(n)} \right]$ is the mismatch on case n

The derivative $\dfrac{\partial G(\mathbf{w})}{\partial \mathbf{w}}$, the gradient $\nabla_{\mathbf{w}} G(\mathbf{w})$, is a sum of terms $\mathbf{g}^{(n)}$

$$g_j^{(n)} \equiv -\left[t^{(n)} - y^{(n)} \right] x_j^{(n)}, \quad \text{for } n = 1, \ldots, N$$

An online algorithm is designed by feeding each input to the neuron, one at a time, and then adjusting $\mathbf{w}$ a bit in the direction opposite to $\mathbf{g}^{(n)}$ (a stochastic gradient descent)

Training (cont.)

Sequential training

Architecture: A lonely ☹ neuron with I inputs x_i and one output y

- Associated with each input is a weight w_i $(i = 1, \dots, I)$

Activity: In response to inputs $\mathbf{x}$, we compute the neuron activation

$$\begin{aligned} a &= \mathbf{w} \cdot \mathbf{x} \\ &= \sum_i w_i x_i \end{aligned}$$

The output y is set as a logistic sigmoid of the activation $a = \mathbf{w} \cdot \mathbf{x}$

$$y(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Training (cont.)

Training

The teacher supplies a target value $t \in \{0, 1\}$

- The correct label t of input $\mathbf{x}$

We compute the neuron error with weights $\mathbf{w}$

$$e = t - y$$

We adjust all the weights to reduce the error

$$\Delta w_i = \eta \underbrace{(t - y) x_i}_{-g_i^{(\cdot)}}$$

- They call η the 'step-size'

Activity and training are repeated for each supplied pair $(\mathbf{x}, t)$

- A change in weight is made after every pair is presented

With fixed-size sets of data, we can cycle thru multiple times

Training (cont.)

An alternative training paradigm is to go through a batch of examples

- Compute all the outputs $y^{(n)}$ and errors $e^{(n)}$
- Accumulate all the changes, $\Delta w_i = \eta e x_i$
- Apply cumulative change at the end

Batch training

For each input-output pair $\{\mathbf{x}^{(n)}, t^{(n)}\}$, we compute $y^{(n)} = y(\mathbf{x}^{(n)}|\mathbf{w})$

$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + \exp\left(-\sum_i w_i x_i\right)}$$

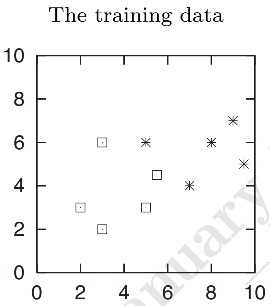
Define the term $e^{(n)} = \left[t^{(n)} - y^{(n)}\right]$ and compute for each weight w_i

$$g_i^{(n)} = -e^{(n)} x_i^{(n)}$$

Then, let

$$\Delta w_i = -\eta \sum_n g_i^{(n)}$$

Training (cont.)



Batch-mode (gradient descent)

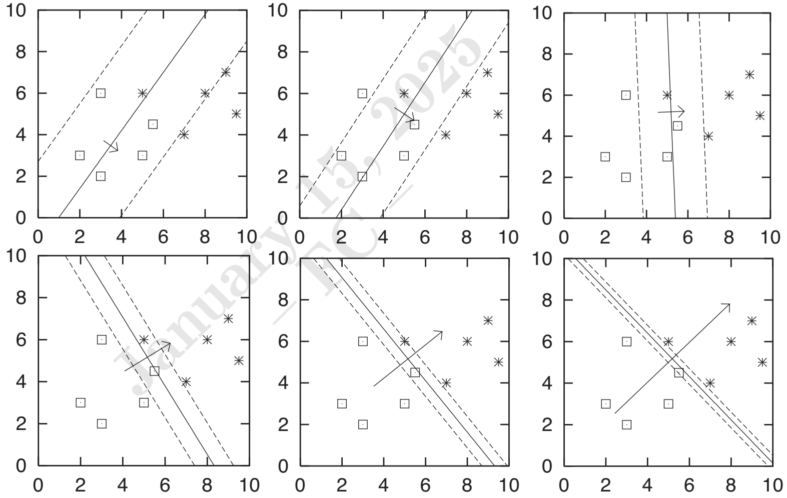
- Step-size $\eta = 0.01$

Each data point consists of a two-dimensional input vector $\mathbf{x}$ and a t value

- $\times$ for $t = 1$
- $\square$ for $t = 0$

Training (cont.)

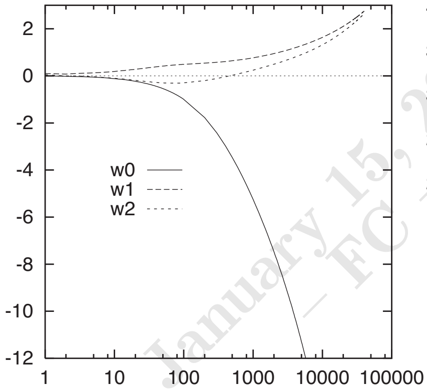
The performed function after 30, 80, 500, 3K, 10K and 40K iterations

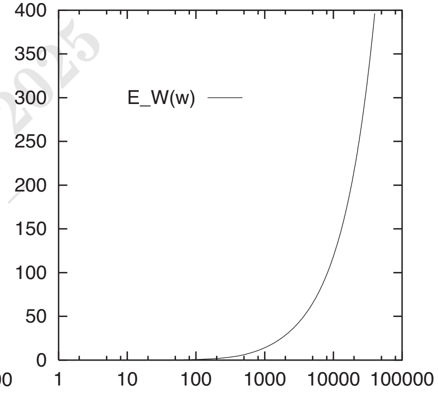


Contours correspond to $a \in \{-1, 0, +1\}$, namely at $y \in \{0.27, 0.50, 0.73\}$

Training (cont.)

(a) Evolution of the weights as a function of the (log) number of iterations





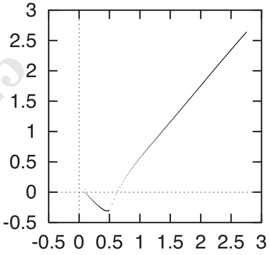
(b) Magnitude of the weights as a function of the (log) number of iterations

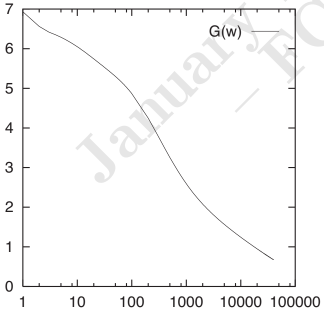
$$E_W(\mathbf{w}) = 1/2 \sum_i {w_i}^2$$

Training (cont.)

Evolution of the weights

- In weight-space





The error function as a function of number of iterations (on log scale)

Training (cont.)

This training algorithm works, if η is set to an appropriate value

- It finds a $\mathbf{w}$ that correctly classifies examples

For linearly separable examples, the neuron finds the separation

- With time, weights diverge to ever-larger values
- It is a manifestation of overfitting (undesirable)

Note to self: It's dumb to early-stop an algorithm meant to do minimisation

- It is more principled to use regularisation

Training (cont.)

Regularisation

We augment the error function so that we penalise solutions we dislike (large weights)

- Them sharp boundaries arising from large weight values
- We penalise large (half-)norms of the parameter vector

$$E_W(\mathbf{w}) = \frac{1}{2}\sum_i {w_i}^2$$
$$= \frac{1}{2}||\mathbf{w}||_2^2$$

We augment the error function $G(\mathbf{w})$ with a weight-decay regulariser $E_W(\mathbf{w})$

$$M(\mathbf{w}) = G(\mathbf{w}) + \alpha E_W(\mathbf{w})$$

- α is a regularisation constant (it is a hyper-parameter)

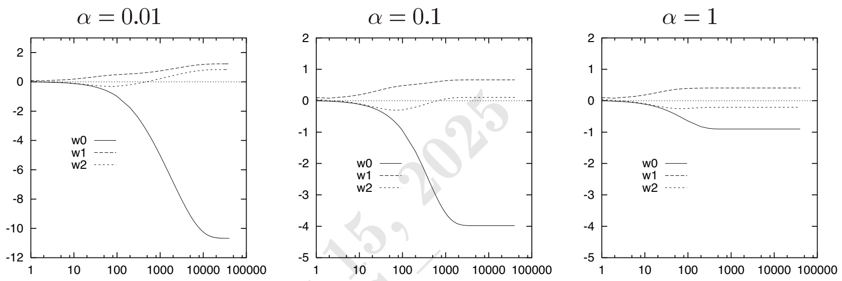
Training (cont.)

$$M(\mathbf{w}) = \underbrace{-\sum_n \left\{ t^{(n)} \ln \left[y(\mathbf{x}^{(n)}|\mathbf{w}) \right] + (1 - t^{(n)}) \ln \left[1 - y(\mathbf{x}^{(n)}|\mathbf{w}) \right] \right\}}_{\text{Error function: } G(\mathbf{w})} + \alpha \underbrace{\frac{1}{2} \sum_i {w_i}^2}_{\text{Regularizer: } E_W(\mathbf{w})}$$

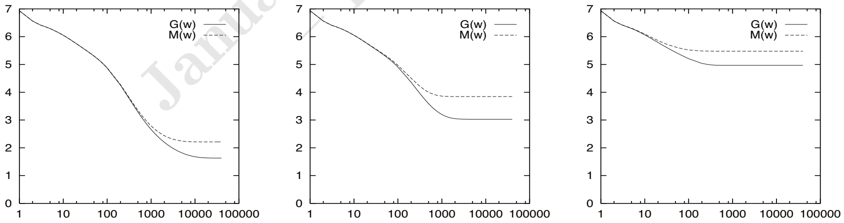
Influence of weight decay on the neuron batch training, gradient descent

- $\alpha \in \{0.01, 0.1, 1\}$

Training (cont.)



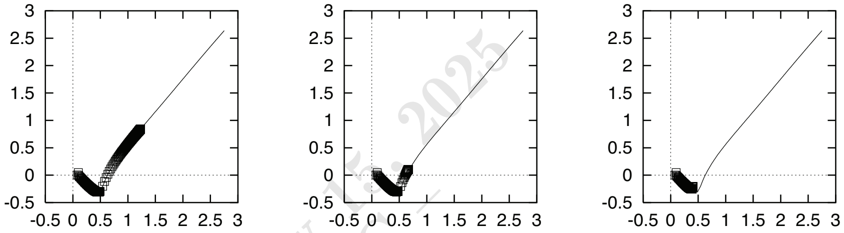
(a) Evolution of weights w_0, w_1 and w_2, as a function of number of iterations



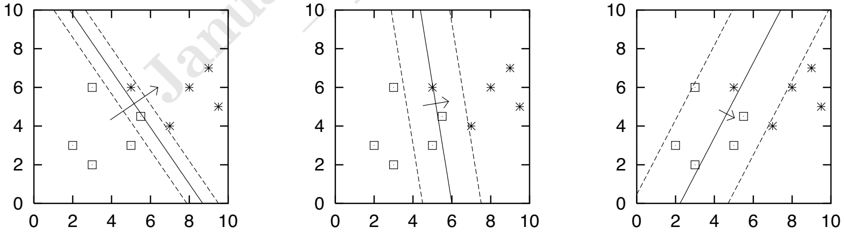
(c) Objective $M(\mathbf{w})$ and error function $G(\mathbf{w})$, against number of iterations

Training (cont.)

(b) In weight space, with trajectory in the case of zero weight-decay



(d) The function performed by the neuron, after 40 000 iterations



Learning and predictions

The neuron

Learning

The neuron

Learning

We trained the neuron to behave as a linear classifier

- Minimisation of an objective function

$$M(\mathbf{w}) = \underbrace{-\sum_n \left\{ t^{(n)} \ln \left[y(\mathbf{x}^{(n)}|\mathbf{w}) \right] + (1 - t^{(n)}) \ln \left[1 - y(\mathbf{x}^{(n)}|\mathbf{w}) \right] \right\}}_{\text{Error function: } G(\mathbf{w})}$$

$$+ \alpha \underbrace{\frac{1}{2} \sum_i {w_i}^2}_{\text{Regularizer: } E_W(\mathbf{w})}$$

The neuron's output $y(\mathbf{x}, \mathbf{w})$ defines the probability that an input $\mathbf{x}$ belongs to class $t = 1$, rather than to the alternative $t = 0$, when the parameter values $\mathbf{w}$ are all given

---

Values of $\mathbf{w}$ define the different hypothesis about the probability of class 1

- Relative to class 0, as function of input $\mathbf{x}$

Learning (cont.)

Assume the inputs $\{\mathbf{x}_n\}_{n=1}^{N}$ to be given (sure variables, not to be modelled)

Let D be the observed target data $D = \{t_n\}_{n=1}^{N}$

To infer parameters $\mathbf{w}$ given data D, we require a likelihood function

- The joint probability of all of the data, given parameters

Plus some prior probability over $\mathbf{w}$

Learning (cont.)

The likelihood measures how well various parameters $\mathbf{w}$ predict the observed data

- It is the probability assigned by the model to the observed data t

$$p(t = 1|\mathbf{w}, \mathbf{x}) = y(\mathbf{w}, \mathbf{x})$$
$$p(t = 0|\mathbf{w}, \mathbf{x}) = 1 - y(\mathbf{w}, \mathbf{x})$$

Each observed datum is assumed to be Bernoulli with parameter y

$$\begin{aligned} p(t|\mathbf{w}, \mathbf{x}) &= y^{t}(1-y)^{1-t} \\ &= \exp\left[t \ln(y) + (1-t)\ln(1-y)\right] \end{aligned}$$

For independent and identically distributed data, the probability of the data

$$\begin{aligned} p(\{t\}|\mathbf{w}, \{\mathbf{x}\}) &= \prod_n \left\{ y(\mathbf{w}, \mathbf{x}_n)^{t_n} \left[1 - y(\mathbf{w}, \mathbf{x}_n)\right]^{t_n} \right\} \\ &= \prod_n e^{\left\{ t_n \ln[y(\mathbf{w},\mathbf{x}_n)] + (1-t_n)\ln[1-y(\mathbf{w},\mathbf{x}_n)] \right\}} \\ &\rightsquigarrow \exp\Bigg(\underbrace{\sum_n \left\{ t_n \ln[y(\mathbf{w}, \mathbf{x}_n)] + (1-t_n)\ln[1 - y(\mathbf{w}, \mathbf{x}_n)] \right\}}_{-G(\mathbf{w})} \Bigg) \end{aligned}$$

Learning (cont.)

$$p(\{t\}|\mathbf{w},\{\mathbf{x}\}) = \exp\Bigg(\underbrace{\sum_n \{t_n \ln[y(\mathbf{w},\mathbf{x}_n)] + (1-t_n)\ln[1-y(\mathbf{w},\mathbf{x}_n)]\}}_{-G(\mathbf{w})}\Bigg)$$

This is the probabilistic interpretation of the cross-entropy objective

---

The error function $G(\mathbf{w})$ can be interpreted as negative log likelihood

$$p(\{t\}|\mathbf{w}) = \exp\big[-G(\mathbf{w})\big]$$

Learning (cont.)

The regulariser is interpreted as log prior probability over the parameters **w**

$$P(\mathbf{w}|\alpha) = \frac{1}{Z_W(\alpha)} \exp\Big[-\alpha E_W\Big]$$

If E_W is quadratic, then the corresponding prior distribution is Gaussian

$$\begin{aligned} P(\mathbf{w}|\alpha) &= \frac{1}{Z_W(\alpha)} \exp\Big[-\alpha E_W\Big] \\ &= \frac{1}{Z_W(\alpha)} \exp\Big[-\frac{\alpha}{2}\sum_i {w_i}^2\Big] \end{aligned}$$

- I is the number of parameters in **w**
- $Z_W^{-1}(\alpha)$ is equal to $(\alpha/2\pi)^{I/2}$
- The variance $\sigma_W^2 = 1/\alpha$
- The mean $\boldsymbol{\mu} = \mathbf{0}$

Learning (cont.)

Why is it natural to interpret the error functions as log probabilities?

- Probabilities are multiplicative, for independent events
- Error functions are often additive, over multiple data
- The log fixes the correspondence

Generalised Gaussian priors

$$E_W(\mathbf{w}) = \frac{1}{2}\sum_i |w_i|^q$$

If E_W is a q-norm, the prior distribution is a generalised Gaussian

$$p(\mathbf{w}|\alpha) = \left[\frac{q}{2}\left(\frac{\alpha}{2}\right)^{1/q}\frac{1}{\Gamma(1/q)}\right]^I \exp\left[-\frac{\alpha}{2}\sum_i |w_i|^q\right]$$

Learning (cont.)

The objective function $M(\mathbf{w})$ is inference of parameters $\mathbf{w}$, given data D

$$\begin{aligned} P(\mathbf{w}|D,\alpha) &= \frac{P(D|\mathbf{w})P(\mathbf{w}|\alpha)}{P(D|\alpha)} \\ &= \frac{1}{P(D|\alpha)} e^{-G(\mathbf{w})} \frac{1}{Z_W(\alpha)} e^{-\alpha E_W(\mathbf{w})} \\ &= \frac{1}{Z_M(\alpha)} \exp\big[-M(\mathbf{w})\big] \end{aligned}$$

The $\mathbf{w}$ by minimising $M(\mathbf{w})$ is interpreted as the most probable vector $\hat{\mathbf{w}}$

The partition function (normalisation constant)

$$Z_M(\alpha) = \int d\mathbf{w} \exp\big[-M(\mathbf{w})\big]$$

Learning (cont.)

Estimator $\hat{\mathbf{w}}$, the product of traditional learning is a point in the weight-space

- $\hat{\mathbf{w}}$ maximises the posterior probability density

In a sensible sense, the product of learning is an ensemble of plausible values

- We do not choose one particular hypothesis $\mathbf{w}$
- We rather evaluate the posterior probabilities

The posterior distribution, the likelihood times a prior distribution over $\mathbf{w}$

Learning (cont.)

For a neuron with two inputs and no bias

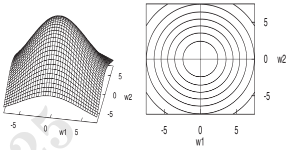
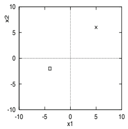
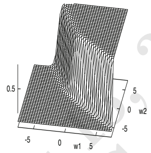
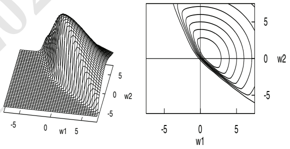
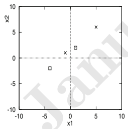
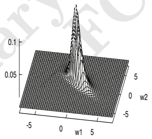
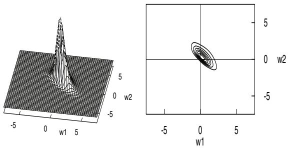
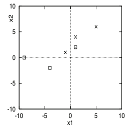
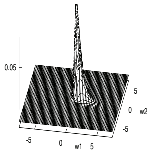
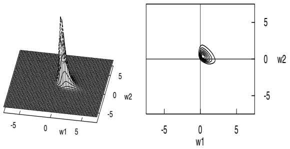
$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}$$

We can plot the posterior probability of $\mathbf{w}$

$$p(\mathbf{w}|D, \alpha) \propto \exp\big[- M(\mathbf{w})\big]$$

Each data point consists of a two-dimensional input vector $\mathbf{x}$ and a t value

- $\times$ for $t = 1$
- $\square$ for $t = 0$

Data set	Likelihood	Probability of parameters
$N = 0$	*(constant)*	
$N = 2$ 		
$N = 4$ 		
$N = 6$ 		

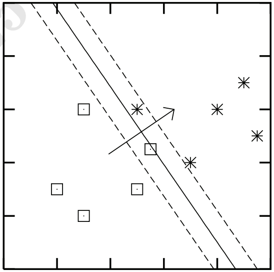
Prediction

The neuron

Prediction

The task is making predictions using the neuron we trained as classifier

$$y(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}}$$

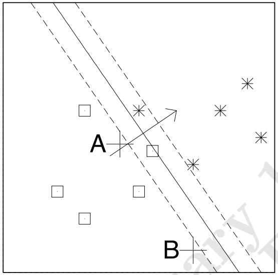


Training by minimising $M(\mathbf{w}) = G(\mathbf{w}) + \alpha E(\mathbf{w})$, optimized for $\alpha = 0.01$

We consider the task of predicting class $\mathbf{t}^{(N+1)}$ for a new input $\mathbf{x}^{(N+1)}$

- We could just use the neuron, weights set to $\hat{\mathbf{w}}$

Prediction (cont.)



Consider two new data points

- A and B

Both are assigned to class 1

- With probability 0.2
- $\hat{\mathbf{w}}\mathbf{x}^{(\mathrm{A})} = \hat{\mathbf{w}}\mathbf{x}^{(\mathrm{B})}$

These predictions would be correct, if we really really knew $\mathbf{w}$ was $\hat{\mathbf{w}}$

- But we do not, parameters are uncertain
- We placed a prior over them
- We even got its posterior

Prediction (cont.)

Prediction of a new datum $\mathbf{t}^{(n)}$ involves marginalising over the parameters

- Well, over anything else that is endowed with uncertainty
- We assume that we only have the weight $\mathbf{w}$ uncertain
- Weight-decay α and model $\mathcal{H}$ are assumed to be sure

$$p(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, D, \alpha) = \int d^K\mathbf{w} P(\mathbf{t}^{N+1}|\mathbf{x}^{(N+1)}, \mathbf{w}, \alpha)P(\mathbf{w}|D, \alpha)$$

Predictions are weighted by weighting the predictions, for all possible $\mathbf{w}$

- $P(\mathbf{t}^{(N+1)} = 1|\mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) = y(\mathbf{x}^{(N+1)}|\mathbf{w})$
- $P(\mathbf{t}^{(N+1)} = 0|\mathbf{x}^{(N+1)}, \mathbf{w}, \alpha) = 1 - y(\mathbf{x}^{(N+1)}|\mathbf{w})$

The weights are given by the posterior probabilities of $\mathbf{w}$

- $P(\mathbf{w}|D, \alpha) = 1/Z_M \exp[-M(\mathbf{w})]$
- $Z_M = \int d^K\mathbf{w} \exp[-M(\mathbf{w})]$

Prediction (cont.)

We get predictions if we find a way of computing the integral

$$p(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, D, \alpha) = \int d^K \mathbf{w} y(\mathbf{x}^{(N+1)}|\mathbf{w}) \frac{1}{Z_M} \exp\Big[-M(\mathbf{w})\Big]$$

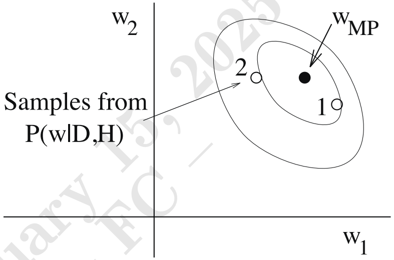
Expectation of function y under the posterior distribution

$$\langle y(\mathbf{w}) \rangle \simeq \frac{1}{R} \sum_r y(\mathbf{w}_r)$$

Average the output at $\mathbf{x}^{(N+1)}$, under the posterior of $\mathbf{w}$

$$\{\mathbf{w}_r\}_{r=1}^{R}, \quad \text{simulated}$$

Prediction (cont.)



We average together predictions made by each possible value of weights **w**

- Each value receives a(nother) weight proportional to its probability
- The probability is under the posterior ensemble

Simulation

The neuron

Simulation

Simulation

Need to simulate a density $P(\mathbf{x})$, known to within a multiplicative constant

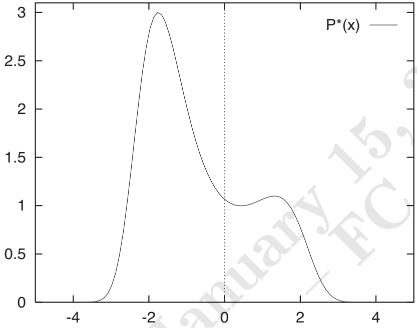
- We can evaluate a function $P^*(\mathbf{x})$ such that $P(\mathbf{x}) = P^*(\mathbf{x})/Z$

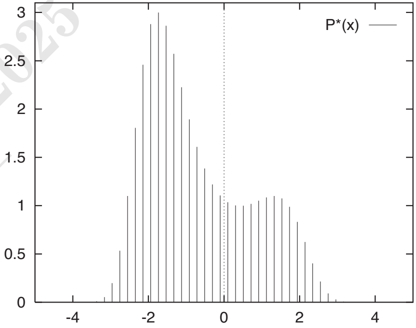
$Z = \int d^N \mathbf{x} P^*(\mathbf{x})$ is unknown, and even if we knew it we would not know how to simulate P w/o listing ALL states (may take a few universe ages)

- Best evaluations come from places where P is big

Simulation (cont.)

We wish to draw samples from some $P(x) = P^*(x)/Z$, we can plot $P^*(x)$





Potentially, we could discretise the support of variable x, and ask for samples from the discrete probability distribution over the finite set of uniformly spaced points $\{x_i\}$, no?

- $p_i^* = P^*(x_i) \quad \longrightarrow Z \simeq \sum_i p_i^* \quad \longrightarrow p_i = p_i^*/Z$

$\rightsquigarrow$ Samples from the empirical distribution $\{p_i\}$

Simulation (cont.)

What's the cost of evaluating (just) Z?

- We have to visit every site x_i?
- Yes! Well, 50 P^* evaluations

- In 1000 dimensions, 50^{1000}
- We do not need a 50-grid!
- A 2-grid is good enough
- 2^{1000}, a lot better
- Not really!

1. **Energy-based simulations**
2. **Shape-based simulations**

Energy-based simulation

The neuron

Energy-based simulations

We do not simulate $P(\mathbf{w}|D,\alpha)$, but another (simpler) density $Q(\mathbf{w}|\Theta)$

- $Q(\mathbf{w}|\Theta)$ must be easy to simulate and similar to $P(\mathbf{w}|D,\alpha)$
- We pick a new $Q(\mathbf{w}|\Theta)$ at each simulated state $\mathbf{w}_r$

The $Q(\mathbf{w}|\Theta,\mathbf{w}_r)$ depends on the system's energy

$$H(\mathbf{w},\mathbf{p}) = \underbrace{E(\mathbf{w})}_{\mathbf{w}^T\mathbf{x}} + \underbrace{K(\mathbf{p})}_{\mathbf{p}^T\mathbf{p}/2}$$

$\mathbf{p}$ are momentum variables

Energy-based simulations

Langevin thermostats

We create, asymptotically, samples from the joint distribution

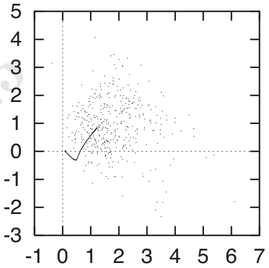
$$\begin{aligned} P_H(\mathbf{w},\mathbf{p}) &= \frac{1}{Z_H}\exp\left[-H(\mathbf{w},\mathbf{p})\right] \\ &= \underbrace{\frac{1}{Z_H}\exp\left[-E(\mathbf{w})\right]}_{\text{desired density}}\exp\left[-K(\mathbf{p})\right] \end{aligned}$$

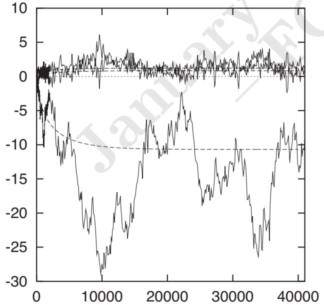
1. Draw $\mathbf{p} \sim \mathcal{N}(\mathbf{p}|\boldsymbol{\mu},\mathbf{I})$
2. Calculate gradient $\mathbf{g} = \partial E(\mathbf{w})/\partial\mathbf{w}$
3. Make a step in $\mathbf{w}$-space $\Delta\mathbf{w} = -\gamma^2\mathbf{w} + 2\gamma\mathbf{p}$
4. Accept/reject proposal $\mathbf{w}$ based on changes in $M(\mathbf{w})$ and $\mathbf{g}$

Energy-based simulations (cont.)

Weights' 'evolution'

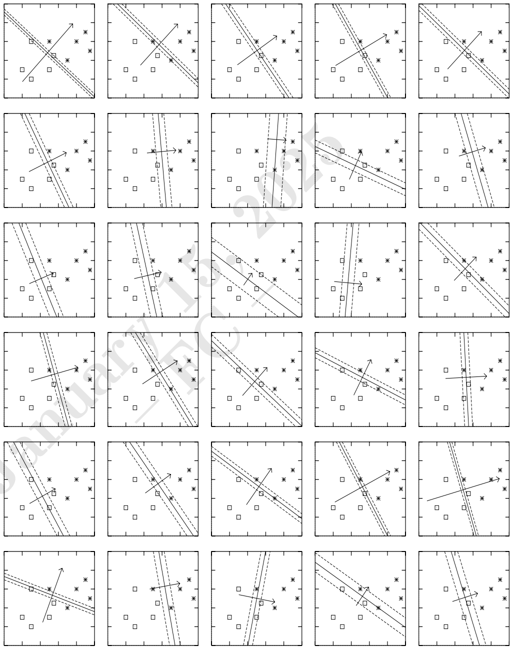
- In weight-space





Energy-based samples

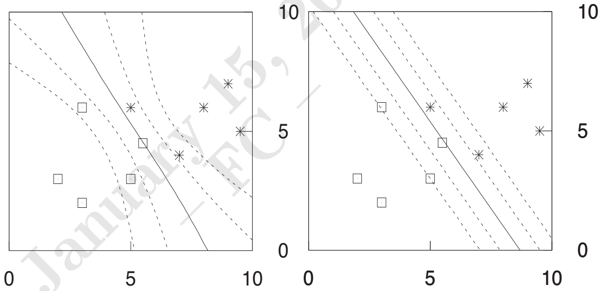
- Convergence in 10 K
- Acceptance rate 93%



Energy-based simulation (cont.)

Samples from iteration 10 000 to 40 000, every 1000 iterations

(a) The predictive function by averaging predictions from 30 samples



- Contours at $a = \{0, \pm 1, \pm 2\}$, $y = \{0.5, 0.27, 0.73, 0.12, 0.88\}$

(b) Predictions by the ‘most probable’ setting of neuron parameters

Energy-based simulation (cont.)

The behaviour of functions $G(\mathbf{w})$ and $M(\mathbf{w})$ during sampling

- Compared with values of G and M at $\hat{\mathbf{w}}$

Function $G(\mathbf{w})$ fluctuates around $G(\hat{\mathbf{w}})$, unsymmetrically

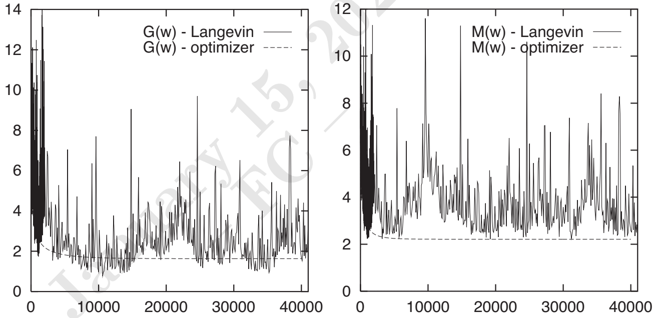
Function $M(\mathbf{w})$ also fluctuates, but not around $M(\hat{\mathbf{w}})$

- It cannot go lower than the optimum
- Actually, it rarely even gets closer

Energy-based simulations (cont.)

(a) The error function as a function of the (log) number of iterations

- The error function during optimisation



(b) The objective function as a function of the (log) number of iterations

- The objective function during optimisation

Shape-based simulation

The neuron

Near-Gaussian simulations

An old trick with nonlinearities is to take them and locally linearise them

An old trick with distributions is to approximate them with Gaussians

- Such approximations may be a good alternative to evaluate

$$p(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, D, \alpha) = \int \mathrm{d}^K \mathbf{w}\, y(\mathbf{x}^{(N+1)}|\mathbf{w}) \frac{1}{Z_M} \exp\left[-M(\mathbf{x})\right]$$

$\rightsquigarrow$ The actual name of the method is saddle-point approximation

$\rightsquigarrow$ The actual name of the method is Laplace's approximation

$\rightsquigarrow$ The actual name of the method is Gaussian approximation

Near-Gaussian simulations

Interest in an unnormalised probability density $P^*(x)$, peak at some x_0

- $Z_P \equiv \int P^*(x)\mathrm{d}x$ is the normalising constant

We Taylor-expand the logarithm of $P^*(x)$ around its peak

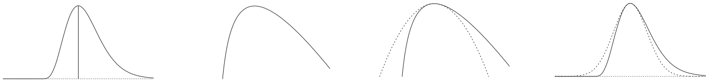
$$\ln\left[P^*(x)\right] \simeq \ln\left[P^*(x_0)\right] - \frac{1}{2}\Big\{\underbrace{-\frac{\partial^2}{\partial x^2}\ln[P^*(x)]\Big|_{x_0}}_{c}\Big\}(x-x_0)^2 + \cdots$$

$$\simeq \ln\left[P^*(x_0)\right] - \frac{c}{2}(x-x_0)^2 + \cdots$$

We approximate $P^*(x)$ by an unnornalised Gaussian

$$Q^*(x) \equiv P^*(x_0)\exp\left[-\frac{c}{2}(x-x_0)^2\right]$$

Z_P is approximated by its normalising constant $\quad Z_Q \equiv P^*(x_0)\sqrt{\frac{2\pi}{c}}$



Shape-based simulations (cont.)

We start by making a Gaussian approximation to the posterior probability

- We go to the bottom of $M(\mathbf{w})$ and there we Taylor-expand it

$$M(\mathbf{w}) \simeq M(\hat{\mathbf{w}}) + \frac{1}{2}(\mathbf{w}-\hat{\mathbf{w}})^{\mathsf{T}}\mathbf{A}(\mathbf{w}-\hat{\mathbf{w}}) + \text{ h.o.t.}$$

  $\mathbf{A}$ is the matrix of second derivatives (the Hessian) of $M(\mathbf{w})$

$$A_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} M(\mathbf{w})\Big|_{\hat{\mathbf{w}}}$$

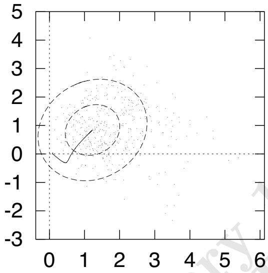
- Then, we define the Gaussian approximation

$$Q(\mathbf{w}|\hat{\mathbf{w}}, \mathbf{A}) = \left[\det\left(\frac{\mathbf{A}}{2\pi}\right)\right]^{1/2} \exp\left[-\frac{1}{2}(\mathbf{w}-\hat{\mathbf{w}})^{\mathsf{T}}\mathbf{A}(\mathbf{w}-\hat{\mathbf{w}})\right]$$

  Q is a normal with covariance matrix $\mathbf{A}^{-1}$

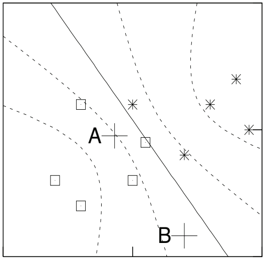
Shape-based simulations (cont.)

(a) A projection of the Gaussian approximation onto the (w_1, w_2)-plane



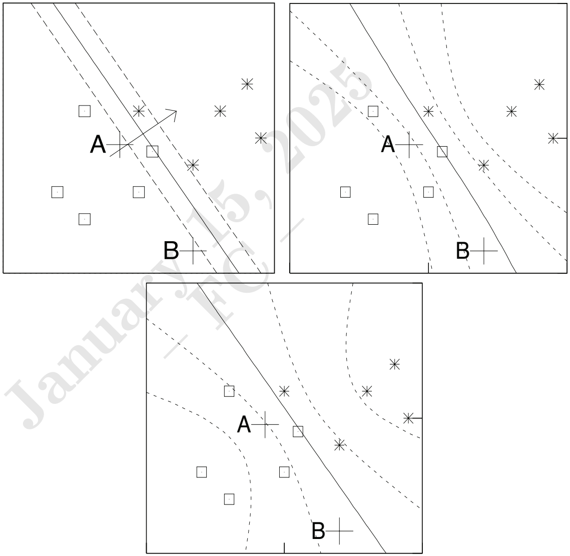
- One and two standard deviation contours
- The trajectory of the optimizer
- Near-normal samples

$$p(\mathbf{t}^{(N+1)}|\mathbf{x}^{(N+1)}, D, \alpha)$$



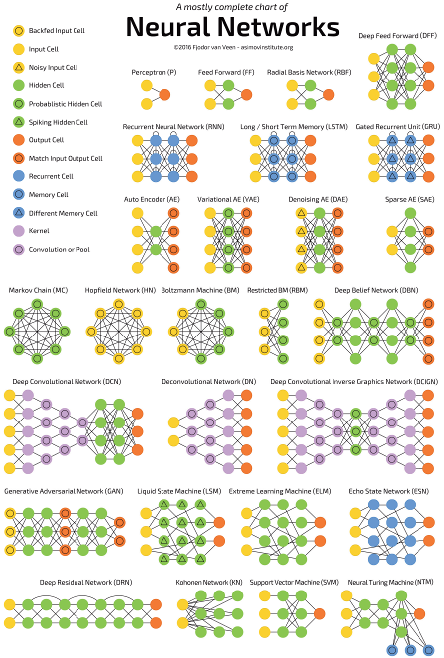
(b) The predictive function obtained from the Gaussian approximation

Wrap-up



Outro

The neuron


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)