Machine Learning in Graphics and Vision

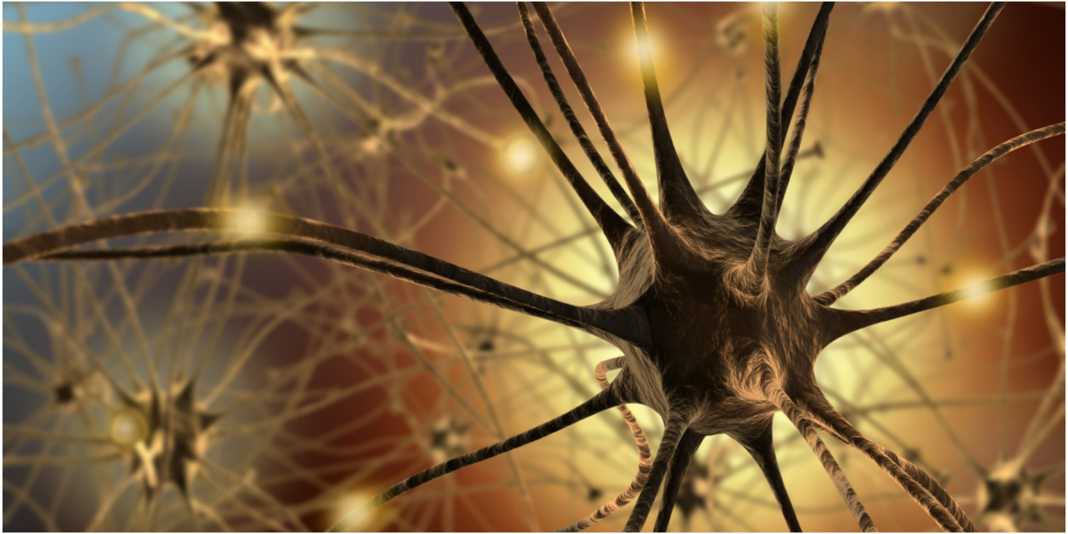
Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group
MPI-IS / University of Tübingen

May 3, 2018

University of Tübingen
MPI for Intelligent Systems
Autonomous Vision Group

Deep Learning



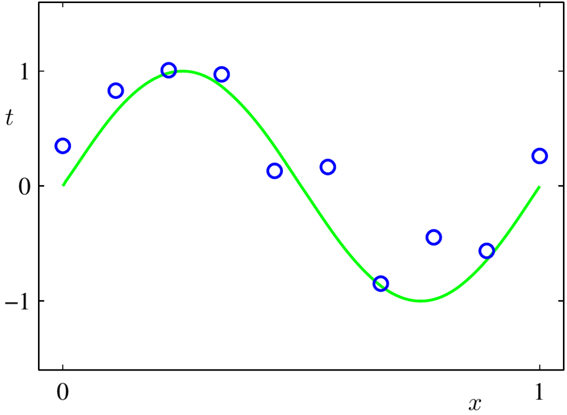
Quiz

Polynomial Curve Fitting

Polynomial Curve Fitting

Let $\mathcal{X}$ denote a dataset of size N and let $(x_n, t_n) \in \mathcal{X}$ denote its elements.

Goal: Predict t for a previously unseen input x.



Example with true model (green) and 10 noisy samples (blue).

Polynomial Curve Fitting

Model:

- Polynomial of order M

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Tasks:

- **Training:** Estimate parameters $\mathbf{w}$ by minimizing error function, *e.g.*:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2$$

- **Inference:** Given parameters $\mathbf{w}$ and novel x, predict $t = y(x, \mathbf{w})$

Quiz

Polynomial Model:

$$y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j$$

- Linear in x?
- Linear in $\mathbf{w}$?

Error function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2$$

- Functional form of E wrt. $\mathbf{w}$?
- How many local optima when optimizing for $\mathbf{w}$?

Quiz

Linear Model:

$$y(x, \mathbf{w}) = w_0 + w_1 x$$

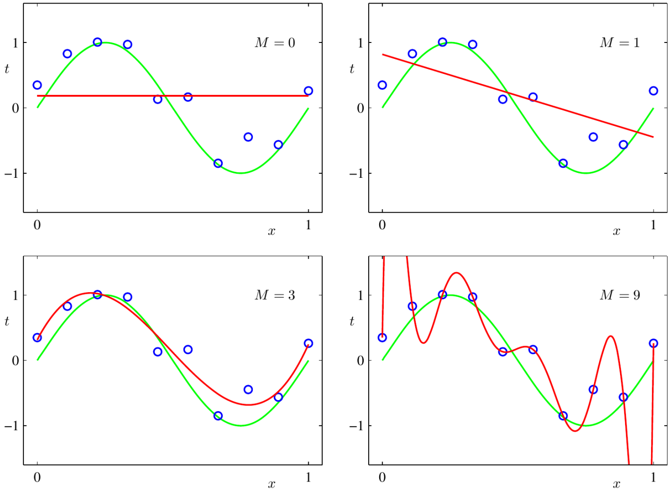
- How many observations required?
- In which situations will the error be 0?

Quadratic Model:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2$$

- How many observations required?
- In which situations will the error be 0?

Quiz

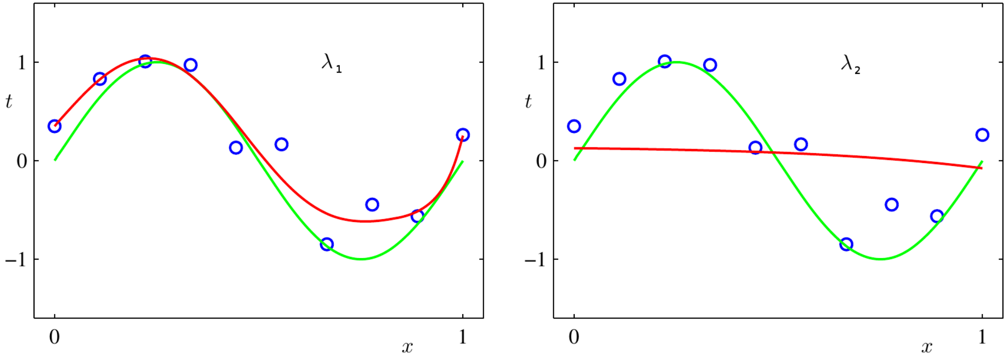


Which polynomial order M is right? How to determine in practice?

Quiz

Ridge Regression:

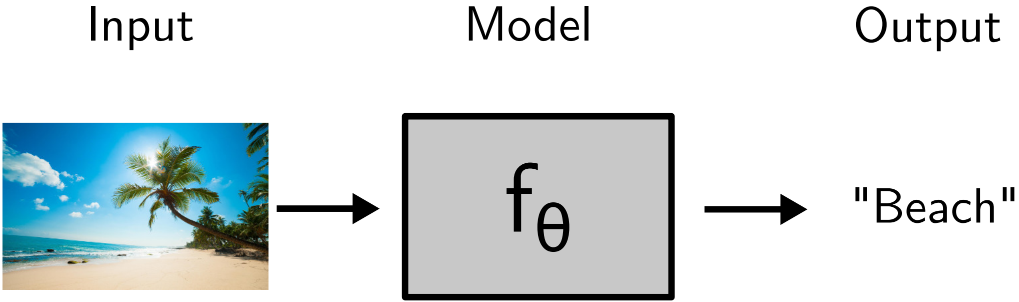
$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n,\mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



$\lambda_1 > \lambda_2$ or $\lambda_2 > \lambda_1$?

Overview

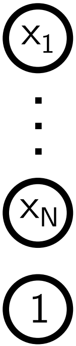
Multi-Class Classification



- $\cancel{f_\theta : \mathbf{x} \in \mathbb{R}^{W \times H} \mapsto y \in \{1, \dots, L\}}$
- $f_\theta : \mathbf{x} \in \mathbb{R}^{W \times H} \mapsto \mathbf{y} = [0, \dots, \textcolor{red}{1}, \dots, 0] \in \{0, 1\}^L$

Linear Regression

- Mapping:
  $\mathbf{x} =$ Image

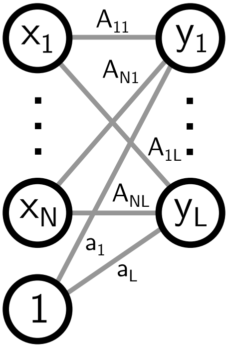


Linear Regression

- Mapping:

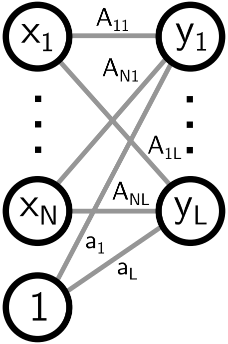
  $\mathbf{x}$ = Image

  $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{a}$



Linear Regression

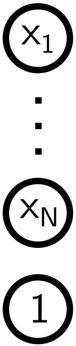
- Mapping:
  $\mathbf{x} =$ Image
  $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{a}$
- Classification:
  $L^* = \text{argmax}_l \; y_l$



Linear Regression

- 3 Layers:

  $\mathbf{x}$ = Image

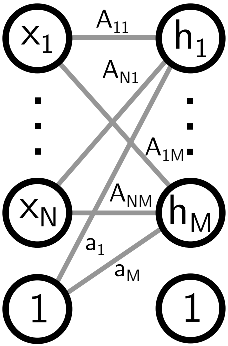


Linear Regression

- 3 Layers:

  $\mathbf{x} = \text{Image}$

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$



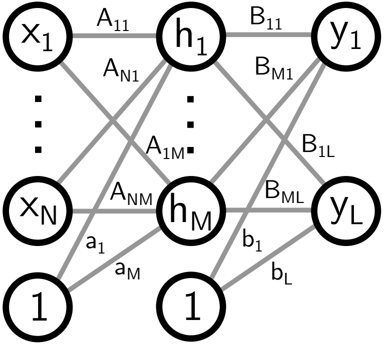
Linear Regression

- 3 Layers:

  $\mathbf{x} =$ Image

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$



Linear Regression

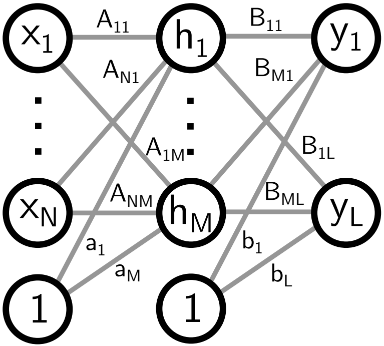
- 3 Layers:

  $\mathbf{x} =$ Image

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?



Linear Regression

- 3 Layers:

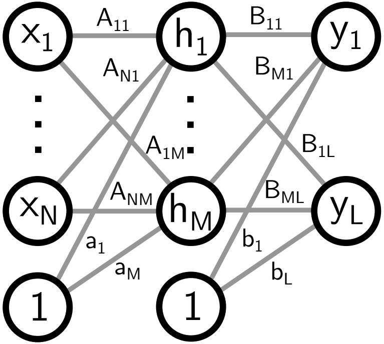
  $\mathbf{x} =$ Image

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$



Linear Regression

- 3 Layers:

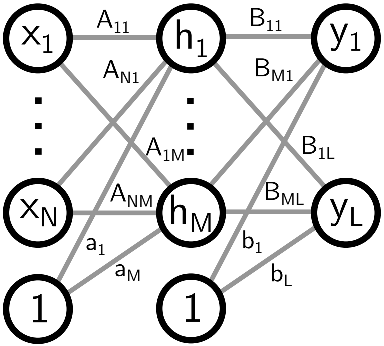
  $\mathbf{x} =$ Image

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

  $\mathbf{y} = \mathbf{B}(\mathbf{A}\mathbf{x} + \mathbf{a}) + \mathbf{b}$



Linear Regression

- 3 Layers:

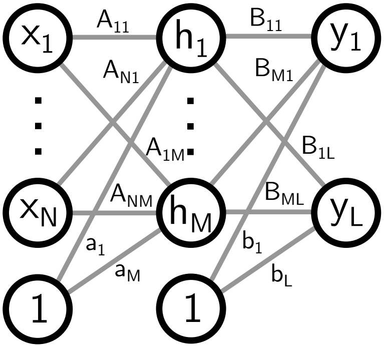
  $\mathbf{x} =$ Image

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

  $\mathbf{y} = \mathbf{B}\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{a} + \mathbf{b}$



Linear Regression

- 3 Layers:

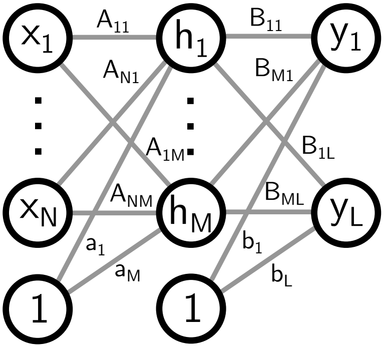
  $\mathbf{x} =$ Image

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

- Is this model better?

  $\mathbf{y} = \underbrace{\mathbf{B}\mathbf{A}}_{=\mathbf{C}}\mathbf{x} + \underbrace{\mathbf{B}\mathbf{a} + \mathbf{b}}_{=\mathbf{c}}$



Linear Regression

- 3 Layers:

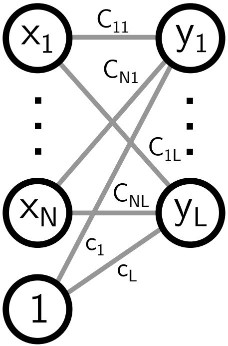
  $\mathbf{x} =$ Image

  $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$

  $\mathbf{y} = \mathbf{B}\mathbf{h} + \mathbf{b}$

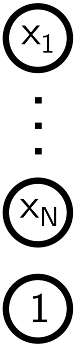
- Is this model better?

  $\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{c}$



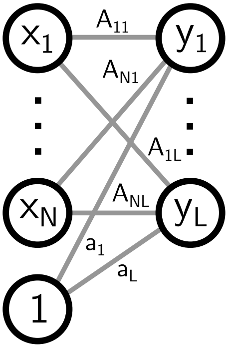
Logistic Regression

- Mapping:
  $\mathbf{x} = \text{Image}$



Logistic Regression

- Mapping:
  $\mathbf{x} =$ Image
  $\mathbf{y} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$



Logistic Regression

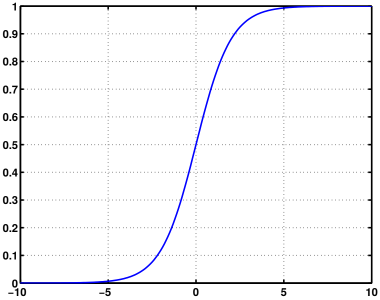
- Mapping:

  $\mathbf{x}$ = Image

  $\mathbf{y} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$

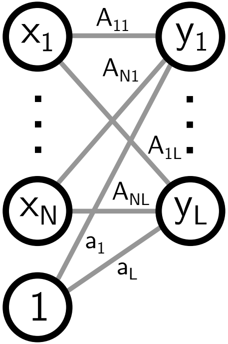
- With (elementwise):

  $\sigma(x) = \frac{1}{1+\exp(-x)}$



Logistic Regression

- Mapping:
  $\mathbf{x} =$ Image
  $\mathbf{y} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$
- With (elementwise):
  $\sigma(x) = \frac{1}{1+\exp(-x)}$
- Classification:
  $L^* = \mathrm{argmax}_l \; y_l$



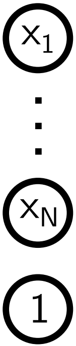
Multilayer Perceptron

- 3 Layers:

  $\mathbf{x} = \text{Image}$

  $\mathbf{h} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$

  $\mathbf{y} = \sigma(\mathbf{B}\mathbf{h} + \mathbf{b})$



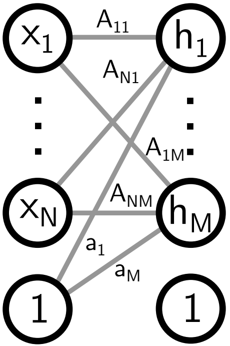
Multilayer Perceptron

- 3 Layers:

  $\mathbf{x} = \text{Image}$

  $\mathbf{h} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$

  $\mathbf{y} = \sigma(\mathbf{B}\mathbf{h} + \mathbf{b})$



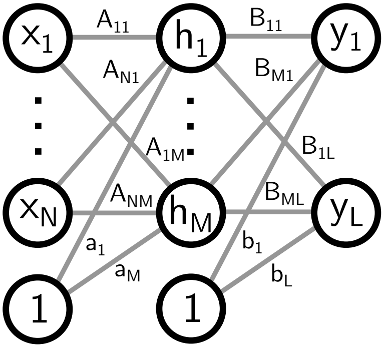
Multilayer Perceptron

- 3 Layers:

  $\mathbf{x} =$ Image

  $\mathbf{h} = \sigma(\mathbf{Ax} + \mathbf{a})$

  $\mathbf{y} = \sigma(\mathbf{Bh} + \mathbf{b})$



Multilayer Perceptron

- 3 Layers:

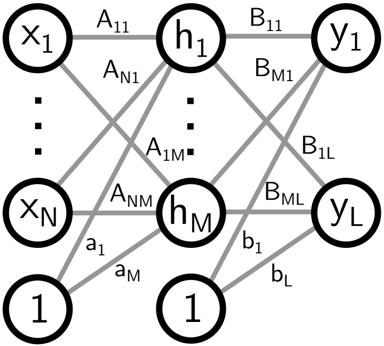
  $\mathbf{x} = \text{Image}$

  $\mathbf{h} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{a})$

  $\mathbf{y} = \sigma(\mathbf{B}\mathbf{h} + \mathbf{b})$

- Now:

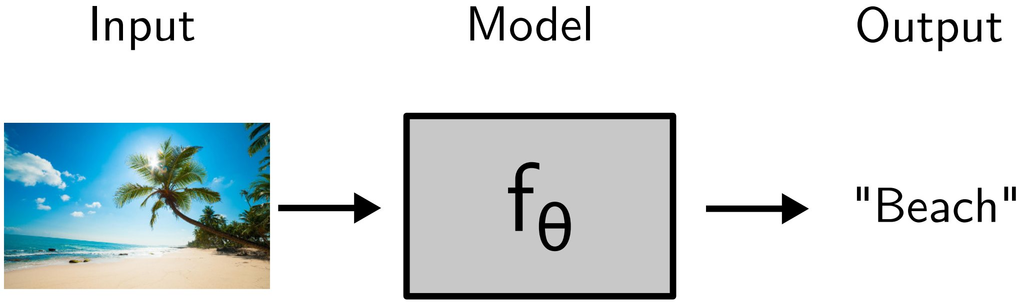
  $\mathbf{y} = \sigma(\mathbf{B}(\sigma(\mathbf{A}\mathbf{x} + \mathbf{a})) + \mathbf{b})$



Logistic Regression

Binary Classification

Let's go back to the binary classification problem again.



- $f_\theta : \mathbf{x} \in \mathbb{R}^{W \times H} \mapsto y \in \{\text{“Beach”}, \text{“No Beach”}\}$

Generative vs. Discriminative Models

So far we have considered the regression problem.

Now consider the classification problem with two classes: c_1 and c_2.

Generative Models:

- We model the class conditional distributions $p(\mathbf{x}|c_k)$
- Classification by computing the class posterior $p(c_k|\mathbf{x})$ using Bayes rule

Discriminative Models:

- We model the class posterior $p(c_k|\mathbf{x})$ directly
- No need to fit the class conditional distributions well

We will shortly see how we can obtain a discriminative model from a generative model!

Recap: Probability Theory

Random Variables

- Discrete random variable: $X \in \{x_1, \ldots, x_M\}$
  - Probability that X takes value x_i: $p(X = x_i)$
- Continuous random variable: $X \in \mathbb{R}$
  - Probability that X takes value in $\mathcal{X} \subset \mathbb{R}$: $p(X \in \mathcal{X})$
- Distribution over X: $p(X)$

Properties

- Joint distribution: $p(X = x_i, Y = y_i)$, short notation: $p(X, Y)$
- Sum rule (marginal): $p(X) = \sum_Y p(X, Y)$ / $\int_Y p(X, Y)$
- Product rule: $p(X, Y) = p(Y|X)p(X)$
- Bayes rule: $p(Y/X) = \frac{p(X|Y)p(Y)}{p(X)}$

Logistic Regression

The posterior probability for class 1 given the observation $\mathbf{x}$ is:

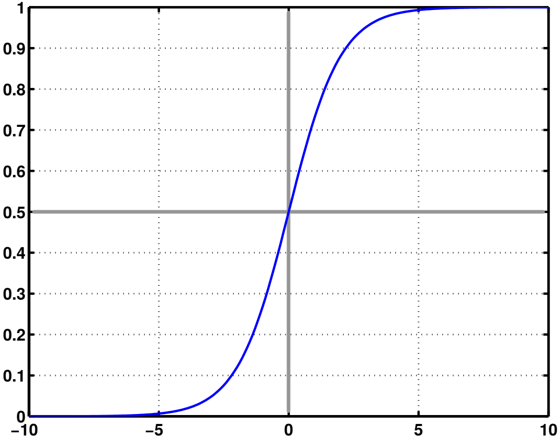
$$\begin{aligned} p(c_1|\mathbf{x}) &= \frac{p(\mathbf{x}|c_1)p(c_1)}{p(\mathbf{x})} \\ &= \frac{1}{1+\exp(-a)} =: \sigma(a) \end{aligned}$$

with log posterior probability ratio:

$$a = \log \frac{p(c_1|\mathbf{x})}{p(c_2|\mathbf{x})} = \log \frac{p(\mathbf{x}|c_1)p(c_1)}{p(\mathbf{x}|c_2)p(c_2)}$$

The posterior probability for class 2 is given by $p(c_2|\mathbf{x}) = 1 - p(c_1|\mathbf{x})$

Logistic Regression



Sigmoid Function: $$\frac{1}{1+\exp(-a)} =: \sigma(a)$$

Logistic Regression

Let's consider multivariate class conditionals with shared covariance $\mathbf{\Sigma}$:

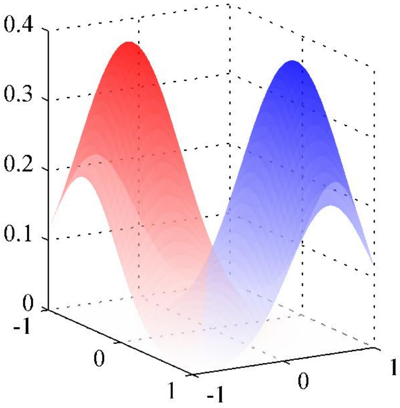
$$p(\mathbf{x}|c_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\mathbf{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right\}$$

Thus:

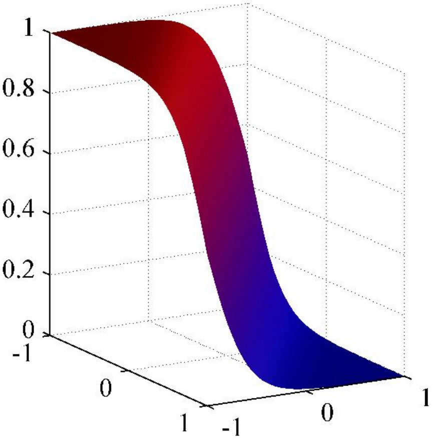
$$\begin{aligned}
p(c_1|\mathbf{x}) &= \sigma\left(\log\left(\frac{\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1)p(c_1)\right\}}{\exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_2)p(c_2)\right\}}\right)\right) \\
&= \sigma\left(\underbrace{\left(\boldsymbol{\mu}_1^T - \boldsymbol{\mu}_2^T\right)\mathbf{\Sigma}^{-1}}_{\mathbf{w}^T}\,\mathbf{x}\,\underbrace{-\frac{1}{2}\boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_2 + \log\frac{p(c_1)}{p(c_2)}}_{w_0}\right) \\
&= \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)
\end{aligned}$$

Note: We have obtained a discriminative model from a generative one!

Logistic Regression

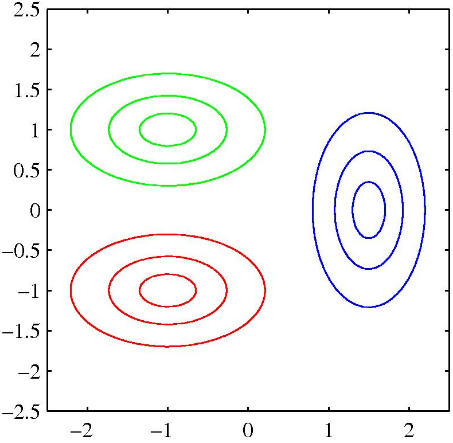


Class conditional densities $p(\mathbf{x}|c_k)$

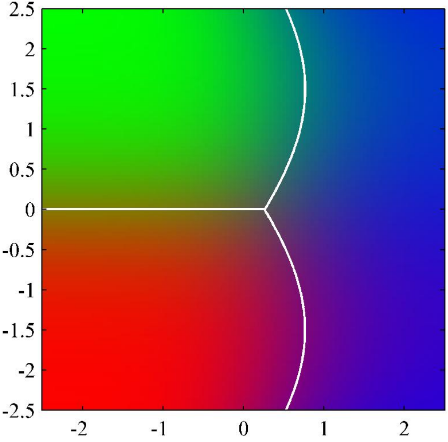


Posterior $p(c_1|\mathbf{x})$

Logistic Regression



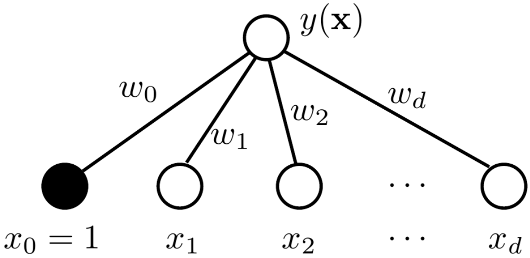
Class conditional densities $p(\mathbf{x}|c_k)$



Posterior $p(c|\mathbf{x})$

Logistic Regression

Interpretation as 2-Layer Neural Network:



output layer (here: single node)

weights

input layer

$$\boxed{y(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)}$$

- Classification result obtained by checking for $y(\mathbf{x}) > 0.5$ / $y(\mathbf{x}) < 0.5$
- We obtain linear regression as a special case iff $\sigma(x) = x$

Logistic Regression

Inference:

- Given $\mathbf{w}, w_0$ and $\mathbf{x}$, predict: $p(c_1|\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x} + w_0\right)$ and $p(c_2|\mathbf{x}) = 1 - p(c_1|\mathbf{x})$
- Choose class with largest posterior probability: $c = \mathrm{argmax}_k \; p(c_k|\mathbf{x})$

Learning:

- Given dataset $\mathcal{X} = \{(\mathbf{x}_n, t_n)\}_{n=1}^N \quad \mathbf{x}_n \in \mathbb{R}^D \quad t_n \in \{0,1\} \quad (t_n = 1 \Leftrightarrow \mathbf{x}_n$ in $c_1)$
- Maximize likelihood:

$$
\begin{aligned}
p(\mathbf{T}|\mathbf{X};\mathbf{w},w_0) &= \prod_{n=1}^N p(t_n|\mathbf{x}_n;\mathbf{w},w_0) \\
&= \prod_{n=1}^N p(c_1|\mathbf{x}_n;\mathbf{w},w_0)^{t_n} p(c_2|\mathbf{x}_n;\mathbf{w},w_0)^{1-t_n} \\
&= \prod_{n=1}^N \sigma(\mathbf{w}^T\mathbf{x}_n + w_0)^{t_n}(1-\sigma(\mathbf{w}^T\mathbf{x}_n + w_0))^{1-t_n}
\end{aligned}
$$

Logistic Regression

Learning:

- Given dataset $\mathcal{X} = \{(\mathbf{x}_n, t_n)\}_{n=1}^N \quad \mathbf{x}_n \in \mathbb{R}^D \quad t_n \in \{0,1\} \quad (t_n = 1 \Leftrightarrow \mathbf{x}_n$ in $c_1)$
- Maximize likelihood:

$$p(\mathbf{T}|\mathbf{X};\mathbf{w},w_0) = \prod_{n=1}^N \sigma(\mathbf{w}^T\mathbf{x}_n + w_0)^{t_n}\left(1-\sigma(\mathbf{w}^T\mathbf{x}_n + w_0)\right)^{1-t_n}$$

- Minimize negative Log-likelihood (=cross entropy loss):

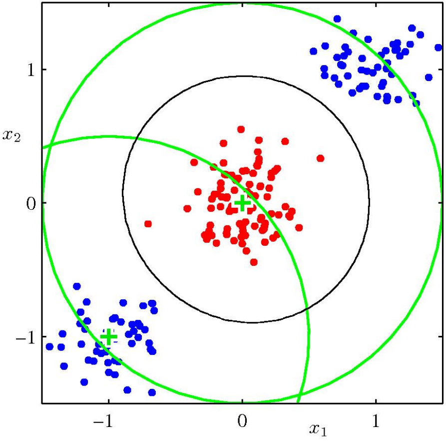
$$-\log p(\mathbf{T}|\mathbf{X};\mathbf{w},w_0) = -\sum_{n=1}^N t_n \log\sigma(\mathbf{w}^T\mathbf{x}_n + w_0) + (1-t_n)\log\left(1-\sigma(\mathbf{w}^T\mathbf{x}_n + w_0)\right)$$

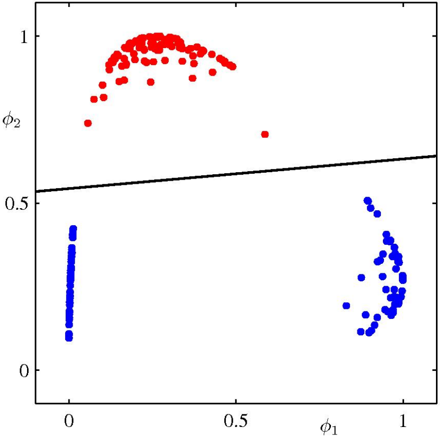
- Use gradient descent. Gradient can be derived as (absorbing w_0 into $\mathbf{w}$):

$$\frac{\partial}{\partial \mathbf{w}} - \log p(\mathbf{T}|\mathbf{X};\mathbf{w},w_0) = \sum_{n=1}^N \left(\sigma(\mathbf{w}^T\mathbf{x}_n) - t_n\right)\mathbf{x}_n$$

Logistic Regression

Logistic regression in non-linear feature space to model non-linear separations:

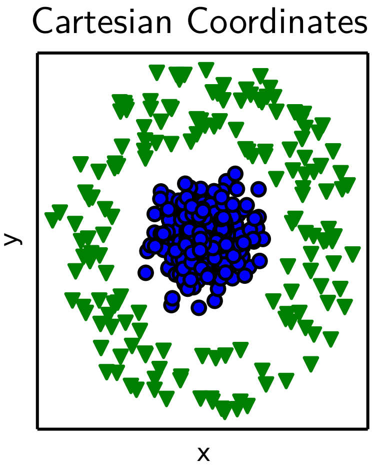


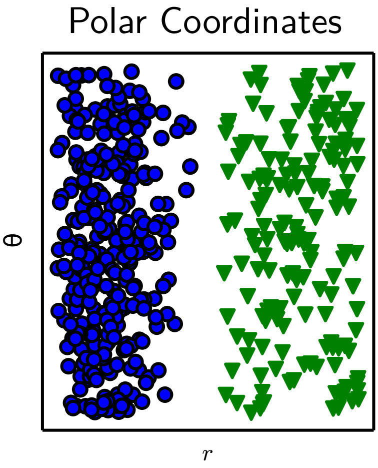


Input space w/ Gaussian bases $\phi_{1/2}(\mathbf{x})$ Feature space w/ decision boundary

Multilayer Perceptron

Representation Learning



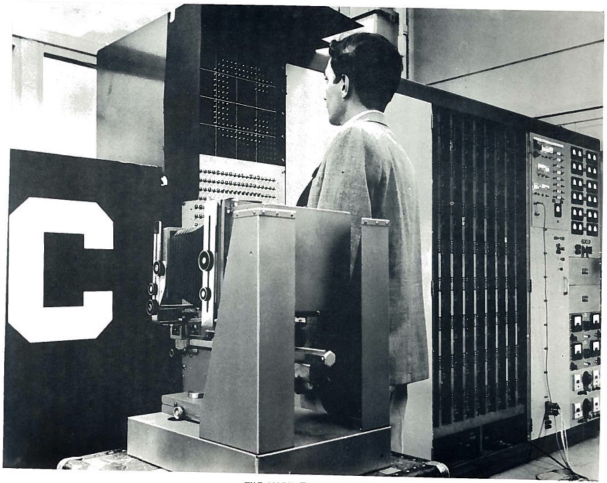


Deep Learning

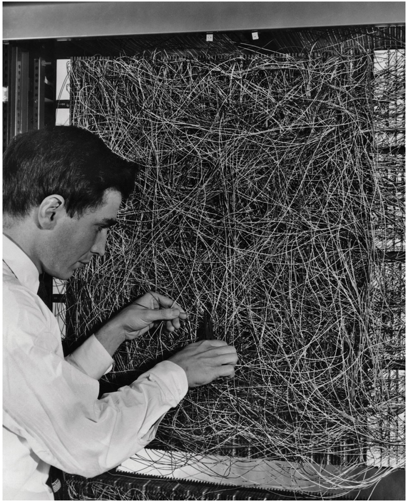
"Deep learning is just a buzzword for neural nets, and neural nets are just a stack of matrix-vector multiplications, interleaved with some non-linearities. No magic there."

Ronan Collobert (developed Torch), 2011

Perceptron

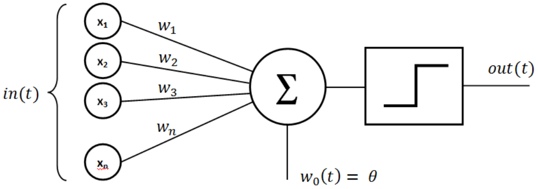


THE MARK I PERCEPTRON



Mark 1 Perceptron, Frank Rosenblatt, 1958

Perceptron



$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + w_0 > 0 \\ 0 & \text{otherwise} \end{cases}$$

- 2 layers (input & output layer) like logistic regression
- We obtain logistic regression if we replace $f(\mathbf{x})$ with a sigmoid $\sigma(\mathbf{x})$
- Activation function $f(\mathbf{x})$ not differentiable wrt. $\mathbf{w}, w_0$
- $\mathbf{w}$ can be optimized using the perceptron algorithm (see Bishop, p.193)

Multilayer Perceptron

Generalizes Perceptron to

- Arbitrary activation functions (sigmoid, tanh, ReLU)
- Multiple outputs
- Multiple layers: simplest form of deep net (≥ 3 layers = "deep")

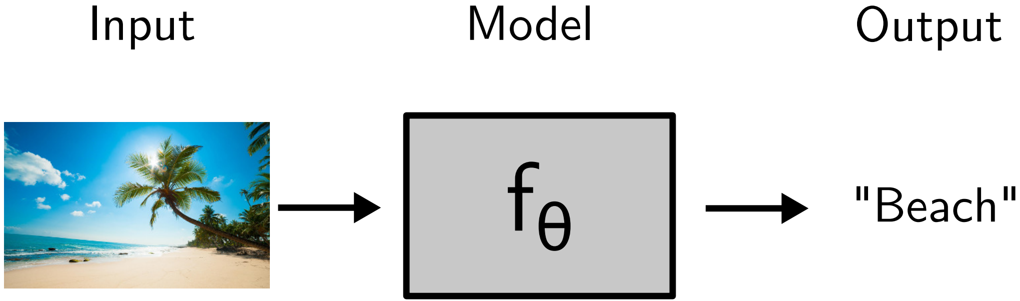
Layer j in a Multilayer Perceptron:

$$y_j = \underbrace{\frac{1}{1+\exp\{-x_j\}}}_{\text{activation function}} \qquad x_j = \underbrace{\sum_i w_{ji} y_i}_{\text{weighted sum}}$$

- Linearly combines *all* outputs of prev. layer i
- Applies non-linear activiation function (here: sigmoid $y_j = \sigma(x_j)$)

Multilayer Perceptron

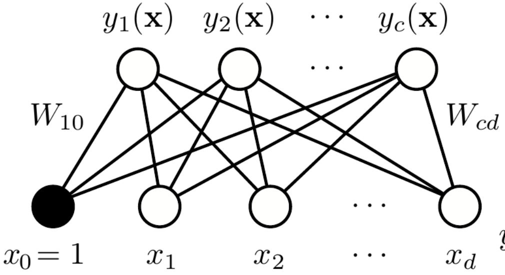
So far: 2 classes. Now consider the multi-class classification problem.



- $\cancel{f_\theta : \mathbf{x} \in \mathbb{R}^{W \times H} \mapsto y \in \{1, \ldots, L\}}$
- $f_\theta : \mathbf{x} \in \mathbb{R}^{W \times H} \mapsto \mathbf{y} = [0, \ldots, 1, \ldots, 0] \in \{0, 1\}^L$

Multilayer Perceptron

Multi-class Neural Network:



$$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki} x_i$$

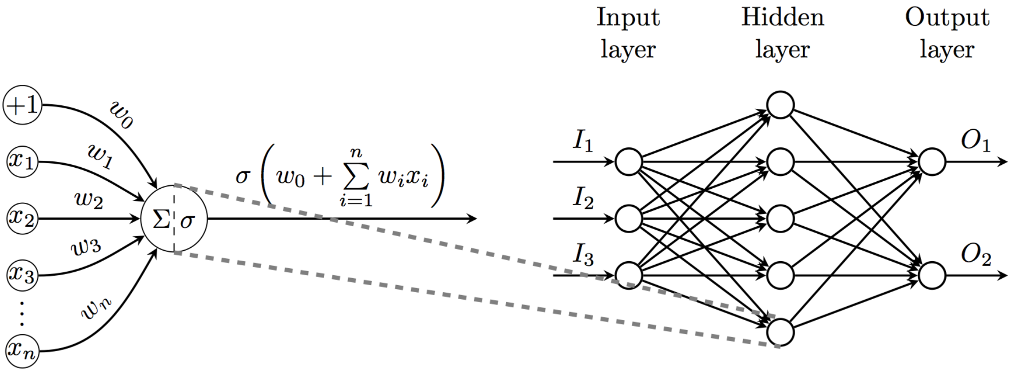
or

$$y_k(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} W_{ki} x_i\right)$$

- Extension of logistic regression/perceptron model to multiple classes is easy
- Can be used for multidimensional linear regression (with linear activation)

Multilayer Perceptron

Extension to Multiple Layers (here: 3):



Inference:

- Apply the equations layer-by-layer, reuse intermediate results
- Resembles processing in the brain (activations indeed correlate)

Multilayer Perceptron - Learning

Multilayer Perceptron - Learning

Learning:

- Define error function over all data points $\{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^N$, *e.g.*:

$$E = \frac{1}{2} \sum_n \sum_j (y_{jn} - t_{jn})^2$$

  - Here y_{jn} is an element of the last layer j of the network
- Calculate $\nabla_{\mathbf{W}} E = \sum_n \nabla_{\mathbf{W}} E_n$ with $E_n = \frac{1}{2} \sum_j (y_{jn} - t_{jn})^2$
- Gradient descent: $\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \nabla_{\mathbf{W}} E|_{\mathbf{W}^t}$ (η = “learning rate”)

Problem?

- Gradient needs to be computed for all data points (problematic when N is large)
  - Solution: Perform Stochastic Gradient Descent (SGD) using mini batches:
  - Approximate gradient with $M << N$ randomly picked data points at each iteration (typically $M = 16, \ldots, 128$ as gradients too noisy for $M = 1$)
- In other words: $\nabla_{\mathbf{W}} E \approx \sum_m \nabla_{\mathbf{W}} E_m$. But how to calculate $\nabla_{\mathbf{W}} E_m$?

Backpropagation

[Rumelhart *et al.*, Nature 1986]

Backpropagation

Multilayer Perceptron (MLP):

$$y_j = \frac{1}{1+\exp\{-x_j\}} \qquad x_j = \sum_i w_{ji} y_i$$

Goal: Calculate gradient $\nabla_{\mathbf{w}} E_m = \nabla_{\mathbf{w}} \frac{1}{2} \sum_j (y_{jm} - t_{jm})^2$ (dropping m for simplicity)

$$\begin{aligned} \frac{\partial E}{\partial y_j} &= y_j - t_j \\ \underbrace{\frac{\partial E}{\partial x_j}}_{=:\delta_j} &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j} = (y_j - t_j) y_j (1 - y_j) \end{aligned}$$

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial w_{ji}} = \delta_j y_i \\ \frac{\partial E}{\partial y_i} &= \sum_j \frac{\partial E}{\partial x_j}\frac{\partial x_j}{\partial y_i} = \sum_j \delta_j w_{ji} \end{aligned}$$

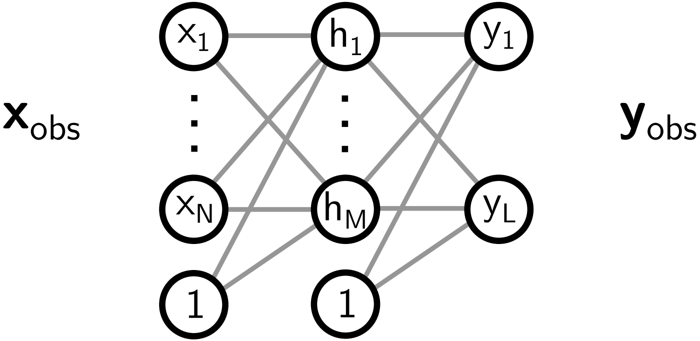
$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i} = \underbrace{\boxed{\delta_i := y_i (1 - y_i) \sum_j \delta_j w_{ji}}}_{\text{Backpropagation formula}}$$

Backpropagation

Algorithm for training a neural network using SGD:

1. Initialize $\mathbf{W}$, pick learning rate η and mini batch size M
2. Randomly pick M data points $\mathcal{X}_{\text{batch}} \subset \mathcal{X}$
3. For each data point $\mathbf{x}_m \in \mathcal{X}_{\text{batch}}$ do:
   3.1 Forward propagate $\mathbf{x}_m$ through network: $y_j = \sigma\left(\sum_i w_{ji} y_i\right)$ (here: $\{y_0\} = \mathbf{x}_m$)
   3.2 Evaluate errors δ_j at output nodes: $\delta_j = (y_j - t_j) y_j (1 - y_j)$
   3.3 Backpropagate errors δ_i through network: $\delta_i = y_i(1 - y_i) \sum_j \delta_j w_{ji}$
   3.4 Calculate derivatives: $\frac{\partial E_m}{\partial w_{ji}} = \delta_j y_i$
4. Update gradients: $\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \sum_{m=1}^{M} \nabla_{\mathbf{W}} E_m |_{\mathbf{W}^t}$
5. If validation error decreases, go to step 2, otherwise stop

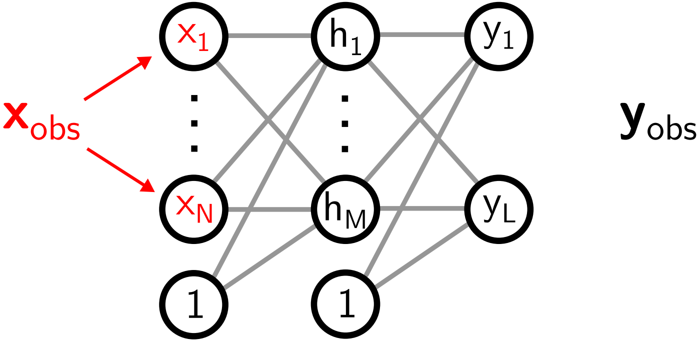
Forward Pass



[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

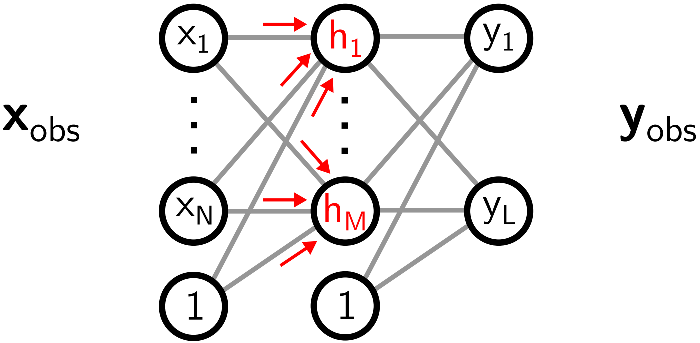
Forward Pass



[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

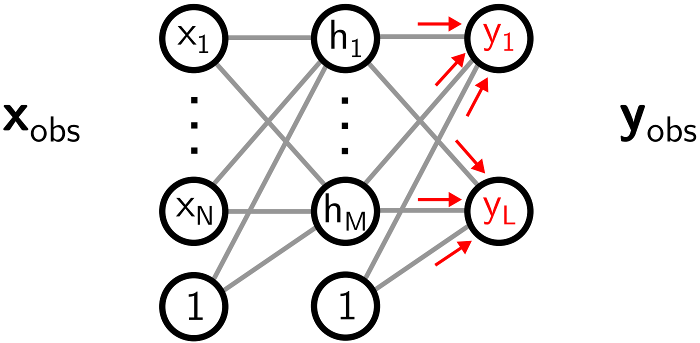
Forward Pass



[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

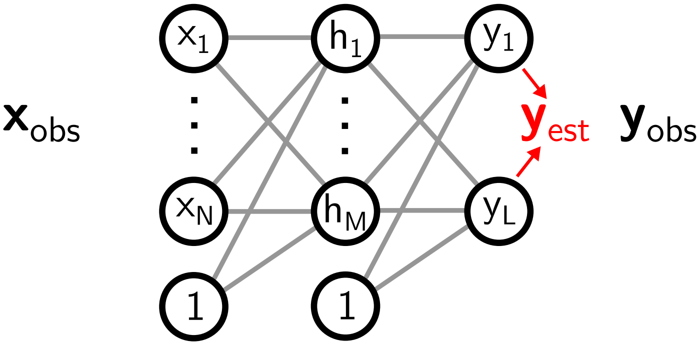
Forward Pass



[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

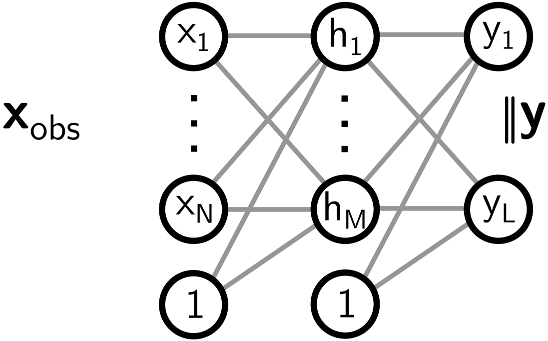
Forward Pass



[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

Error Calculation

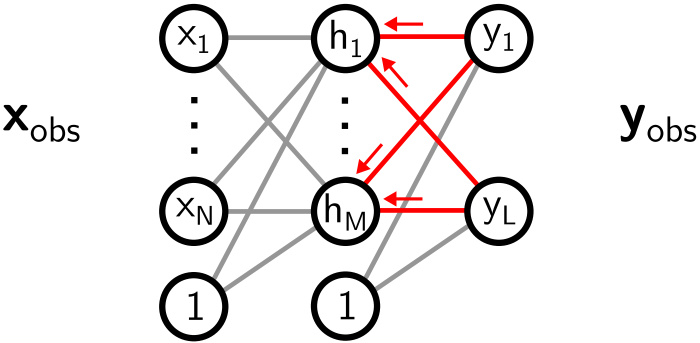


$$E = \|\mathbf{y}_{est} - \mathbf{y}_{obs}\|$$

[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

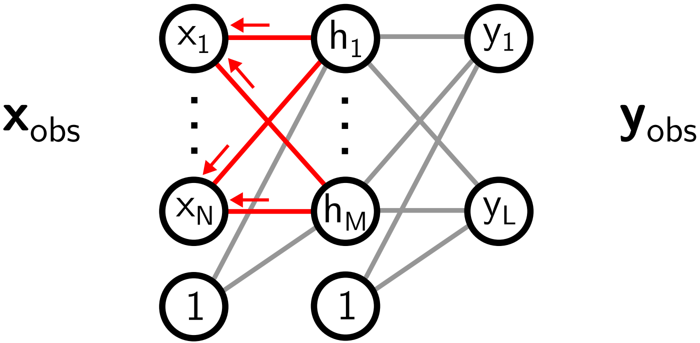
Error Backpropagation



[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

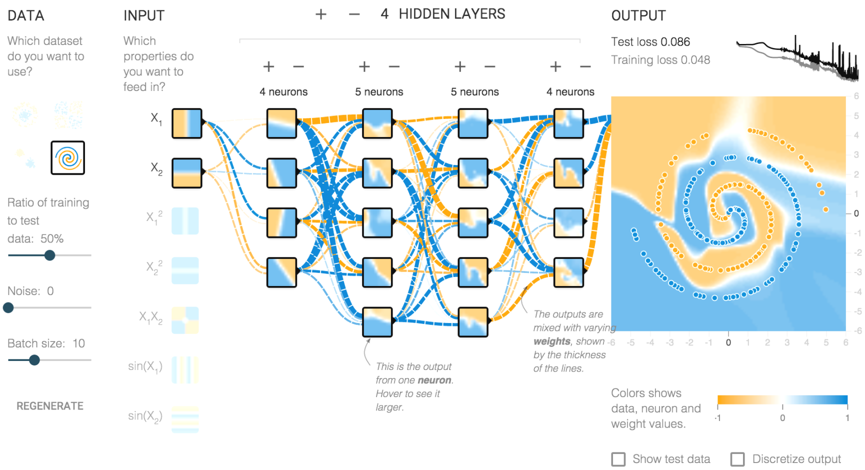
Error Backpropagation



[D. Rumelhart, G. Hinton and R. Williams: Learning representations by back-propagating errors. Nature, 1986]

- Forward pass: resembles processing in the brain
- Backward pass: relationship to neural learning less clear / less well understood

Tensorflow Playground



playground.tensorflow.org

Questions?