02457 Signal Processing in Non-linear Systems: Lecture 6

Perceptrons for signal detection

Ole Winther

Technical University of Denmark (DTU)

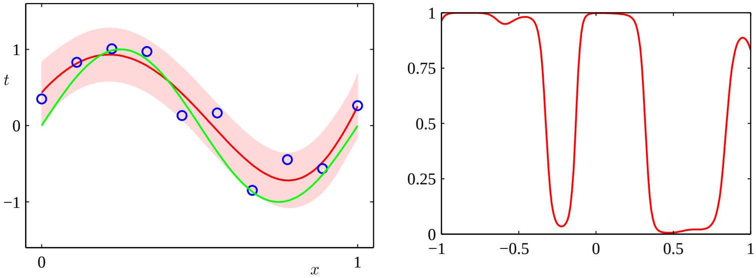
October 9, 2017

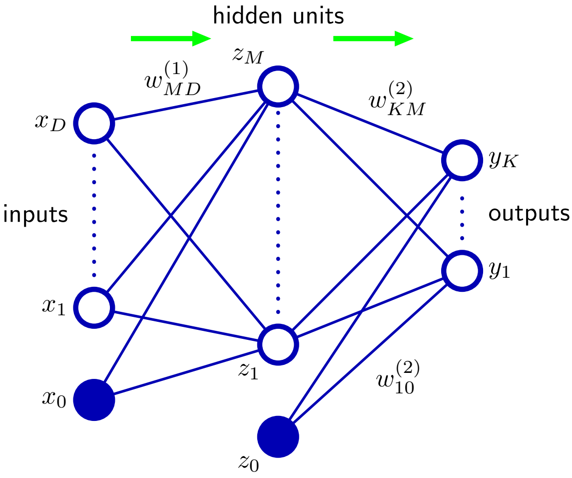
- Hour 1
  - Video 1 – Andrew Ng: “curve fitting”
  - Advanced non-linear optimization
  - Your turn! Exercise 5 quiz
- Hour 2
  - Video 2 – “Speech recognition breakthrough”
  - Neural networks – tricks of the trade
  - Exercise 5 walk through
  - Your turn! Groups go through backpropagation exercise
- Hour 3
  - Decision theory
  - Your turn! Loss function for traffic
  - Neural Networks for classification (Signal detection)
  - Your turn! Construct neural network for AND and XOR

- Video 1 – Andrew Ng: “curve fitting”
- http://www.youtube.com/watch?v=n1ViNeWhC24

Supervised learning

- Learning the conditional distribution $p(\mathrm{output}|\mathrm{input})$.
- Regression – output continuous
- Classification – output discrete (e.g. positive diagnosis)




hidden units
z_M
$w^{(1)}_{MD}$
$w^{(2)}_{KM}$
x_D
y_K
inputs
outputs
y_1
x_1
z_1
$w^{(2)}_{10}$
x_0
z_0

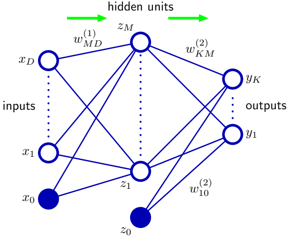
- Add an additional activation (x_0 or z_0) clamped to 1
- Input to hidden unit j:

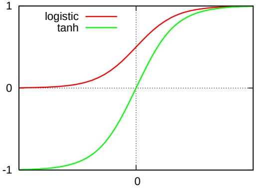
$$\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

- Output *k* two-layer network:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

- Activation functions tanh, logistic or identity.
- How do we count layers?



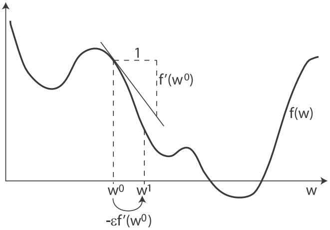


- Gradient descent learning:
- Iterate:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \Delta\mathbf{w}^{\tau}$$

with

$$\Delta w_{ji}^{\tau} = -\eta \frac{dE(\mathbf{w}^{\tau})}{dw_{ji}}$$



- Regression cost function

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

- Classification: same procedure, new def. of cost/likelihood.

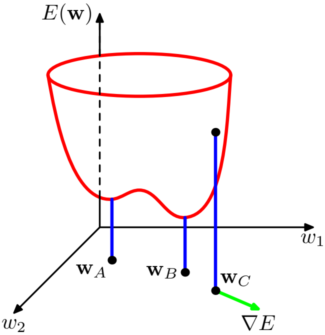
- Objective: to solve the equation

$$\nabla E = 0$$

- Gradient descent:

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \\ \Delta\mathbf{w}^{(\tau)} &= -\eta \nabla E|_{\mathbf{w}^{(\tau)}} \end{aligned}$$

- η is the learning parameter (rate)
- η can be too small: convergence very slow
- η can be too large: oscillatory behavior



- 1d - let w^* be a minimum: $\left.\frac{\partial E(w)}{\partial w}\right|_{w=w^*} = 0$
- We want to find w^* from current value w
- Approximate with quadratic function:

$$E(w) = E(w^*) + \frac{1}{2}H(w - w^*)^2$$

- The derivative is given by

$$\frac{\partial E(w)}{\partial w} = H(w - w^*)$$

- Solve with respect to w^*:

$$w^* = w - H^{-1}\frac{\partial E(w)}{\partial w}$$

- Hence the optimal step is $\Delta w = -H^{-1}\frac{\partial E(w)}{\partial w}$

- Now we to consider multivariate case:
- Second order Taylor expansion of the cost function

$$
\begin{aligned}
E(\mathbf{w}) \quad &\approx \quad E(\mathbf{w}_0) + \sum_j \frac{\partial E}{\partial w_j} \left(w_j - w_{0,j} \right) \\
&+ \frac{1}{2} \sum_{j,k} \frac{\partial^2 E}{\partial w_j \partial w_k} \left(w_j - w_{0,j} \right) \left(w_k - w_{0,k} \right)
\end{aligned}
$$

$$
E(\mathbf{w}) \approx E(\mathbf{w}_0) + \frac{\partial E}{\partial \mathbf{w}} (\mathbf{w} - \mathbf{w}_0) + \frac{1}{2} (\mathbf{w} - \mathbf{w}_0)^T \frac{\partial^2 E}{\partial \mathbf{w} \partial \mathbf{w}^T} (\mathbf{w} - \mathbf{w}_0)
$$

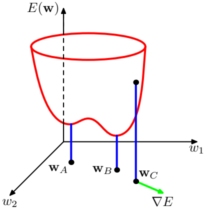
- The symmetric matrix $\mathbf{H} = \frac{\partial^2 E}{\partial \mathbf{w} \partial \mathbf{w}^T}$ is called the *Hessian*

- Zero *gradient*: $\frac{\partial E}{\partial \mathbf{w}} = \nabla E(\mathbf{w}) = 0$ at minimum $\mathbf{w} = \mathbf{w}^*$

$$E(\mathbf{w}) \approx E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w}^*)(\mathbf{w} - \mathbf{w}^*)$$

- Eigenvectors of Hessian:

$$\mathbf{H}\mathbf{u}_j = \lambda_j \mathbf{u}_j \qquad \mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$$



- At a minimum the Hessian is positive definite:

$$\mathbf{v}^T \mathbf{H} \mathbf{v} > 0$$

- in particular for all eigenvectors

$$\mathbf{u}_j^T \mathbf{H} \mathbf{u}_j = \lambda_j > 0$$

- 2nd order expansion $\nabla E(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^*} = 0$ around minimum:

$$E(\mathbf{w}) \approx E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H} (\mathbf{w} - \mathbf{w}^*)$$

$$\nabla E(\mathbf{w}) \approx \mathbf{H} (\mathbf{w} - \mathbf{w}^*)$$

- We find the optimal multivariate step is given by

$$\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1} \nabla E(\mathbf{w})$$

- This is the Newton direction, for a quadratic problem this solves the optimization problem in one iteration!

- The least squares cost function

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

- The first derivative is

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{n=1}^{N}(y_n - t_n)\frac{\partial y_n}{\partial \mathbf{w}}$$

- The second derivative is

$$\frac{\partial^2 E}{\partial \mathbf{w} \partial \mathbf{w}^T} = \sum_{n=1}^{N}\frac{\partial y_n}{\partial \mathbf{w}}\frac{\partial y_n}{\partial \mathbf{w}}^T + \sum_{n=1}^{N}(y_n - t_n)\frac{\partial^2 y_n}{\partial \mathbf{w}\partial \mathbf{w}^T}$$

- ...

- ...
- The Gauss-Newton or outer product approximation is

$$\frac{\partial^2 E}{\partial \mathbf{w} \partial \mathbf{w}^T} \approx \sum_{n=1}^{N} \frac{\partial y_n}{\partial \mathbf{w}} \frac{\partial y_n}{\partial \mathbf{w}}^T$$

- The pseudo-Gauss-Newton approximation is to ignore the off-diagonal terms
- Many other methods: line search, conjugate gradients, gradient-free, Hessian-free.

Week 5 exercise recap

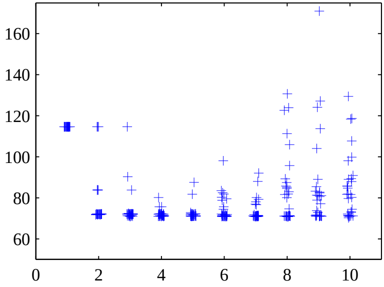
- Exercise 5 walk through
- Your turn! Exercise 5 quiz

- Video 2 – “Speech recognition breakthrough”
- http://www.youtube.com/watch?v=Nu-nlQqFCKg

Local minima

- Because of the highly non-linear nature of the networks there are usually many local minima for the error function
- The more hidden units, the worse
- Weights are initialized at random. The larger the initial weights, the easier to get stuck.

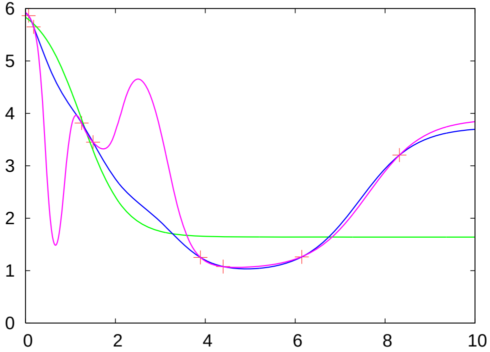
The value of the sum-of-squares error plotted against the number of hidden units with 30 random starts for each network.



Overfitting

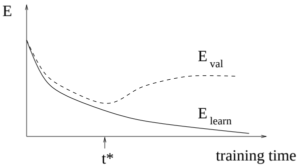
- The eight points shown by plusses lie on a parabola (apart from a bit of "experimental" noise).
- One input, one output unit and

Green: one hidden
Blue: 10 hidden
Purple: 20 hidden



Regularization for improved performance

- 2-norm weight penalty: $||\mathbf{w}||^2$
- 1-norm weight penalty: $|\mathbf{w}| = \sum_i |w_i| \Rightarrow$ sparse solutions
- Dropout (new techniques)
- Ensembles (averaging)
- Pruning of weights – "optimal brain damage"
- Early stopping: stop when the test error is lowest
- Bayesian NNs by sampling - weights priors



Optimal brain damage

- How much does the training error increase if we delete a weight?
-
- Second order expansion:

$$E(\mathbf{w}) \approx E(\mathbf{w}^*) + \frac{\partial E}{\partial \mathbf{w}}(\mathbf{w} - \mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H} (\mathbf{w} - \mathbf{w}^*)$$

- Deletion of the jth weight: $w_j = 0$ and $w_{j'} = w^*_{j'}, j' \neq j$:

$$\begin{aligned} \mathbf{w} - \mathbf{w}^* &= -w_j^* \mathbf{e}_j \\ E(\mathbf{w}) &\approx E(\mathbf{w}^*) - \frac{\partial E}{\partial \mathbf{w}} w_j^* \mathbf{e}_j + \frac{1}{2} w_j^* \mathbf{e}_j^T \mathbf{H} w_j^* \mathbf{e}_j \end{aligned}$$

$$E(\mathbf{w}) \approx E(\mathbf{w}^*) - \frac{\partial E}{\partial w_j} w_j^* + \frac{1}{2} \mathbf{H}_{jj} (w_j^*)^2$$

- However, in the minimum the first derivative is zero, hence

$$\Delta E(\mathbf{w})_{\mathrm{obd}} \approx \frac{1}{2}\mathbf{H}_{jj}(w_j^*)^2$$

defining the optimal brain damage (OBD) *saliency*

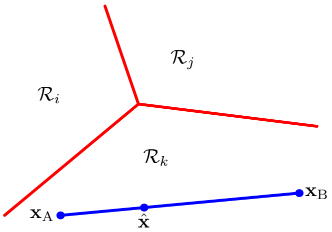
- If the retraining contribution is included (the un-pruned weights are not optimal after pruning) we get instead the optimal brain surgeon (OBS) saliency

$$\Delta E(\mathbf{w})_{\mathrm{obs}} \approx \frac{1}{2}\frac{(w_j^*)^2}{(\mathbf{H}^{-1})_{jj}}$$

- You will use OBD in the Exercise 6 this week
- as a tool to find a solution that can be interpreted.

Backprop exercise

- Your turn! Groups go through backpropagation exercise



- A signal detection system (or pattern classifier) provides a rule for assigning a measurement to a category (class)
- Hence, a classifier divides measurement space into disjoint regions $\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_c$, such that measurements that fall into region $\mathcal{R}_k$ are assigned with class $\mathcal{C}_k$.
- Boundaries between regions are denoted decision surfaces or decision boundaries


www.australianportraits.com
SAMPLE IMAGE - COPYRIGHT

- Imagine that we have a model (for example a neural network) that make probabilistic predictions:

$$p(\mathcal{C}_k|\mathbf{x})$$

- Example - traffic crossing road in situation **x**.
- Model predicts probability for

$$\{\text{hit by car}, \text{missed by car}\}$$

- Our decision about passing the road or waiting will depend not only on the predicted probabilities but also on the cost of the different possible outcomes.

Loss function

- Quantify how much a wrong action costs!
- In this example a two-by-two matrix: L_{kj}
- Other example: make wrong medical diagnosis
- In general at $\mathbf{x}$ we should choose action j which minimizes

$$\mathrm{E}[\mathrm{Loss}(j)] = \sum_k P(\mathcal{C}_k|\mathbf{x})\, L_{kj} \ .$$

Your turn! Loss function for traffic

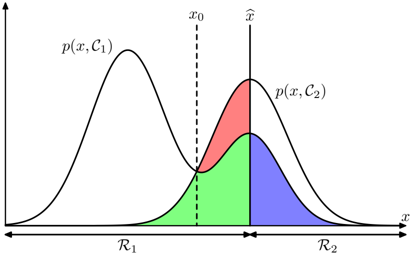
1. Assign a cost (or loss) to the possible outcomes of our action (wait/pass). How many possible outcomes are there?
2. Use the probabilities

$$p(\text{hit}|\mathbf{x}) \;\; \text{and} \;\; p(\text{missed}|\mathbf{x}) = 1 - p(\text{hit}|\mathbf{x})$$

   and the cost to make optimal decisions on our action.
3. What quantity should we optimize here?

- How should we divide $\mathbf{x}$ into regions $\mathcal{R}_1, \ldots, \mathcal{R}_K$ in order to maximize the expected probability of making correct classification?

$$p(\text{correct}) = \sum_{k=1}^{K} p(\mathbf{x} \in \mathcal{R}_k, \mathcal{C}_k) = \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(\mathbf{x}, \mathcal{C}_k)\, d\mathbf{x} \,.$$



- How should we divide $\mathbf{x}$ into regions $\mathcal{R}_1, \ldots, \mathcal{R}_K$ in order to maximize the expected probability of making correct classification?

$$p(\text{correct}) = \sum_{k=1}^{K} p(\mathbf{x} \in \mathcal{R}_k, \mathcal{C}_k) = \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(\mathbf{x}, \mathcal{C}_k)\, d\mathbf{x} \ .$$

- Write $p(\mathbf{x}, \mathcal{C}_k) = p(\mathcal{C}_k|\mathbf{x})p(\mathbf{x})$:

$$p(\text{correct}) = \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(\mathbf{x}, \mathcal{C}_k)\, d\mathbf{x} = \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(\mathcal{C}_k|\mathbf{x})p(\mathbf{x})\, d\mathbf{x} \ .$$

- Choose $\mathcal{R}_k$ such that

$$\operatorname*{argmax}_{k'} p(\mathcal{C}_{k'}|\mathbf{x}) = k \ \text{ for } \ \mathbf{x} \in \mathcal{R}_k \ .$$

- Training data $\mathcal{D} = \{(\mathbf{x}_n, t_n) | n = 1, \ldots, N\}$
- Likelihood function for independent identically distributed (iid) examples, factorizes

$$p(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N} [p(t_n|\mathbf{x}_n, \mathbf{w}) p(\mathbf{x}_n|\mathbf{w})] = \underbrace{p(\mathbf{t}|\mathbf{X}, \mathbf{w})}_{\text{supervised}} \quad \overbrace{p(\mathbf{X}|\mathbf{w})}^{\text{unsupervised}}$$

- For regression, we can use least squares learning

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

- More general learning principle maximum likelihood

- Maximum likelihood, that is maximize

$$\log p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \sum_{n=1}^{N} \log p(t_n|\mathbf{x}_n, \mathbf{w})$$

- New convenient definition of cost function

$$E(\mathbf{w}) = -\log p(\mathbf{t}|\mathbf{X}, \mathbf{w})$$

- The *training error per example*

$$e_{\mathrm{tr}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} -\log p(t_n|\mathbf{x}_n, \mathbf{w})$$

- A *good generalizer* assigns high probability to the true output for a given *new input*:
- We define *the generalization error*:

$$\begin{aligned} e_{\mathrm{gen}} &= \lim_{M\to\infty} \frac{1}{M}\sum_{m=1}^{M} -\log p(t_m|\mathbf{x}_m, \mathbf{w}) \\ &= \int\int -\log p(t|\mathbf{x}, \mathbf{w})\, p(t|\mathbf{x})\, dt\, p(\mathbf{x})\, d\mathbf{x} \end{aligned}$$

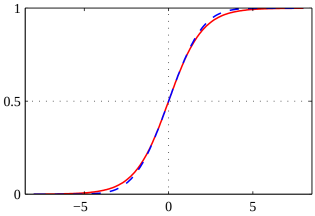
This is the average (expected) error on a test datum $(\mathbf{x}, t)$.

- Labels two class problem: $t_n = 1$ for class one and $t_n = 0$ for class two
- Logistic regression recap – start with real valued function of inputs:

$$a(\mathbf{x}; \mathbf{w}) = \mathbf{w} \cdot \mathbf{x} + w_0$$

- and apply logistic transformation

$$P(t = 1|\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) = \sigma(\, a(\mathbf{x}; \mathbf{w})\,) \quad \text{with} \quad \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$



Two class problem - cost function

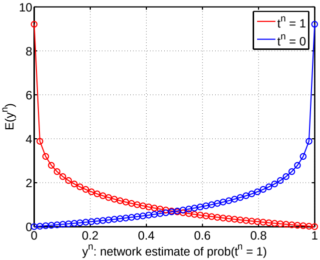
- Labels: $t_n = 1$ for class one and $t_n = 0$ for class two
- Let the network output $y \in [0, 1]$ be the probability of $t = 1$,
- then we can write the likelihood as

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^{N} \left\{ y(\mathbf{x}_n|\mathbf{w})^{t_n} [1 - y(\mathbf{x}_n|\mathbf{w})]^{(1-t_n)} \right\}$$

- and the cost function becomes

$$E(\mathbf{w}) = -\sum_{n=1} \{ t_n \log y(\mathbf{x}_n|\mathbf{w}) + (1 - t_n) \log[1 - y(\mathbf{x}_n|\mathbf{w})] \}$$

- This is called the *entropic cost function*



- The cost is minimal if $y_n = t_n$

$$E(\mathbf{w}) = -\sum_{n=1} \{ t_n \log y(\mathbf{x}_n|\mathbf{w}) + (1 - t_n) \log[1 - y(\mathbf{x}_n|\mathbf{w})] \}$$

- the derivative wrt y is

$$\frac{\partial E}{\partial y_n} = -\left[\frac{t_n}{y_n} - \frac{1 - t_n}{1 - y_n} \right] = \ldots = \frac{y_n - t_n}{y_n(1 - y_n)}$$

- How the entropic cost function penalizes wrong predictions:

$$E(\mathbf{w}) = -\sum_{n=1} \{ t_n \log y(\mathbf{x}_n|\mathbf{w}) + (1 - t_n) \log[1 - y(\mathbf{x}_n|\mathbf{w})] \}$$

- Let $t_n = 1$ and $y_n = 1 - \epsilon_n$ (correct decision)

$$E_n = -\log y_n = -\log[1 - \epsilon_n] \approx \epsilon_n$$

- Let $t_n = 0$ and $y_n = 1 - \epsilon_n$ (very wrong decision!)

$$E_n = -\log[1 - y_n] = -\log[1 - (1 - \epsilon_n)] = -\log \epsilon_n$$

- For comparison the squared error cost function:

$$E_n = (1 - (1 - \epsilon_n))^2 = (\epsilon_n)^2$$

$$E_n = (0 - (1 - \epsilon_n))^2 = (1 - \epsilon_n)^2$$

- MLP w linear output: $a(\mathbf{x};\mathbf{w}) = \mathbf{w}^{(2)} \cdot \mathbf{z}$:

$$y(\mathbf{x}|\mathbf{w}) = \frac{1}{1 + \exp(-a(\mathbf{x};\mathbf{w}))}$$

- Backprop rule

$$\frac{\partial E_n}{\partial w_{jk}} = \delta_{nj} z_{nk}$$

- Derivative of logistic function:

$$\frac{\partial y_n}{\partial a_n} = \frac{\partial}{\partial a_n} \frac{1}{1 + \exp(-a_n)} = y_n(1 - y_n)$$

- Output unit δ-rule

$$\delta_n = \frac{\partial E_n}{\partial a_n} = \frac{\partial E_n}{\partial y_n} \frac{\partial y_n}{\partial a_n} = \frac{y_n - t_n}{y_n(1 - y_n)} y_n(1 - y_n) = y_n - t_n$$

Multiple classes

- We use $0 \leq y \leq 1$ coding for C classes and we want the outputs to be the posterior probabilities $P(C|\mathbf{x})$, hence they "should sum to one"

$$y_k(\mathbf{x}) = \frac{\exp a_k(\mathbf{x})}{\sum_{k'} \exp a_{k'}(\mathbf{x})}$$

- Targets are represented by '1 of K'-vectors. If class k:

$$\mathbf{t} = [0, 0, 0, ..., \underbrace{1}_{k}, 0, \ldots, 0]$$

- The likelihood function is given by

$$p(\mathbf{t}|\mathbf{x}) = \prod_{k=1}^{C} y_k(\mathbf{x})^{t_k}$$

- The likelihood and cost function are given by

$$p(\mathbf{t}|\mathbf{x},\mathbf{w}) = \prod_{k=1}^{C} y_k(\mathbf{x})^{t_k} \qquad E = -\sum_n \sum_k t_{nk} \log y_{nk}$$

- The derivatives are relatively simple again

$$\begin{aligned}
\frac{\partial E_n}{\partial a_k} &= \sum_{k'} \frac{\partial E_n}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial a_k} \\
\frac{\partial y_{k'}}{\partial a_k} &= \delta_{kk'} y_k - y_{k'} y_k \\
\frac{\partial E_n}{\partial y_{k'}} &= -\frac{t_{k'}}{y_{k'}}
\end{aligned}$$

$$\frac{\partial E_n}{\partial a_k} = \sum_{k'} -\frac{t_{k'}}{y_{k'}} (\delta_{kk'} y_k - y_k y_{k'}) = -(t_k - y_k \sum_{k'} t_{k'}) = y_k - t_k$$

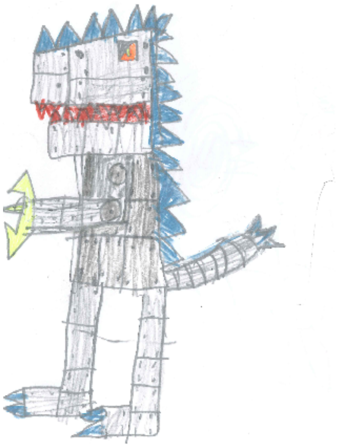
Your turn! Neural networks for AND and XOR

- Consider 2d inputs $\mathbf{x} = (x_1, x_2)$.
- Represent AND and XOR in truth table & graphically (2d)
- The decision boundary is defined as those points in input space with $p(t = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{2}$
- What is the shape of the decision boundary for logistic regression

$$P(t = 1|\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) = \sigma(\ \mathbf{w} \cdot \mathbf{x} + w_0\ )$$

- Try to find **w**-values to solve the AND and XOR problems.
- XOR – use hidden layer and two hidden units
- Hint: each hidden unit acts logistic regressor.

- Advanced non-linear optimization
- Tricks of the trade
- Decision theory
- Perceptrons for signal detection aka classification
    - Cost and likelihood functions
    - Error backpropagation
    - Multiple classes



- Neural networks – Bishop 4.2, 4.3.4, 5.1-5.4
- Decision theory – Bishop 1.5
- Alternative free pdf books:
- Hastie, Tibshirani and Friedman, The Elements of Statistical Learning, Springer and
- MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge