Projects

- **3-4** person groups
- Deliverables: Poster, Report & main code (plus proposal, midterm slide)
- Topics: your own or chose form **suggested topics / kaggle**
- **Week 3 groups** due to TA Nima. Rearrangement might be needed.

- **May 2** proposal due. TAs and Peter can approve.
- Proposal: One page: Title, A large paragraph, data, weblinks, references.
- Something **physical and data oriented.**
- **May ~16** Midterm slides. Likely presented in 4 subgroups (3TA+Peter).
- **5pm 6 June** Jacobs Hall lobby**,** final poster session. Snacks

- Poster, Report & main code. Report due Saturday 16 June.

Final Report

Poster on **June 6** from each group is mandatory. Upload poster as well.

For the Final project (Due Saturday 16 June). Delivery Dropbox request <2GB (details to follow).:

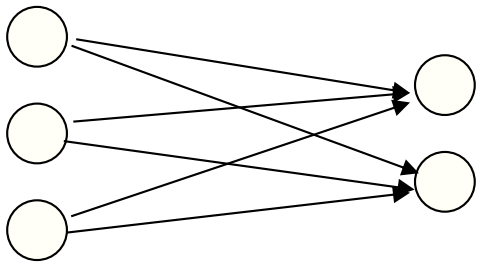
Deliver a code:

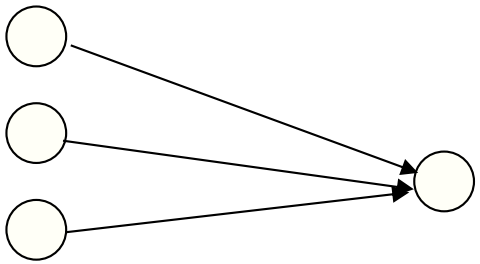
- Assume we have reasonable compilers installed (we use Mac OsX)
- Give instructions if any additional software should be installed.
- You can ask us to download a dataset. Or include it in this submission
- Don't include all developed codes. Just key elements.
- We should not have to reprogram your code.

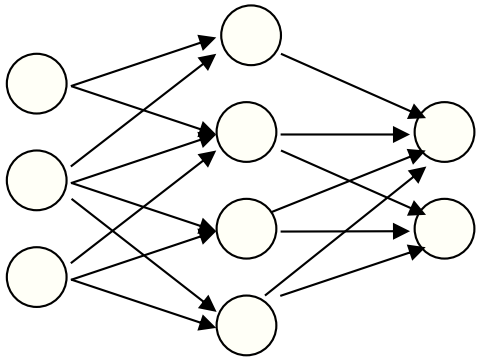
Report

- The report should include all the following sections: Summary -> Introduction->Physical and Mathematical framework->Results.
- Summary is a combination of an abstract and conclusion.
- Plagiarism is not acceptable! When citing use " " for quotes and citations for relevant papers.
- Don't write anything you don't understand.
- Everyone in the group should understand everything that is written. If we do not understand a section during grading we should be able to ask any member of the group to clarify. You can delegate the writing, but not the understanding.
- Use citations. Any concepts which are not fully explained should have a citation with an explanation.
- Please be concise. Equations are good. Figures essential. Write as though your report is to be published in a scientific journal.
- Last year's reports are on class website. Especially good projects 2,11,12,13,

Lecture 8: Backpropagation







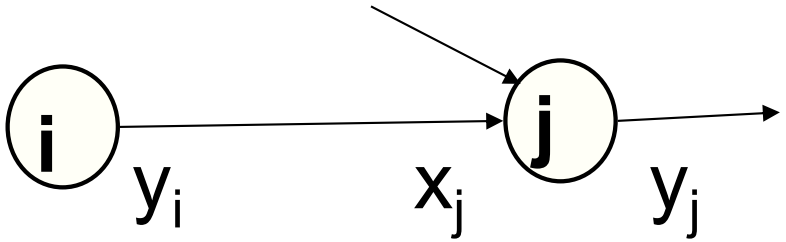
A difference in notation

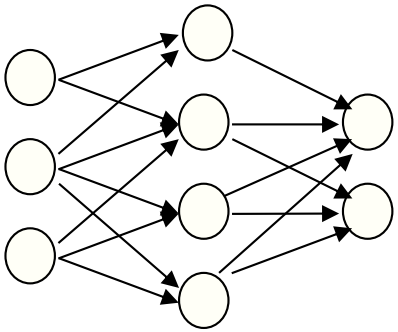
- For networks with multiple hidden layers Bishop uses an explicit extra index to denote the layer.
- The lecture notes use a simpler notation in which the index denotes the layer implicitly.

  y is the output of a unit in any layer

  x is the **summed** input to a unit in any layer

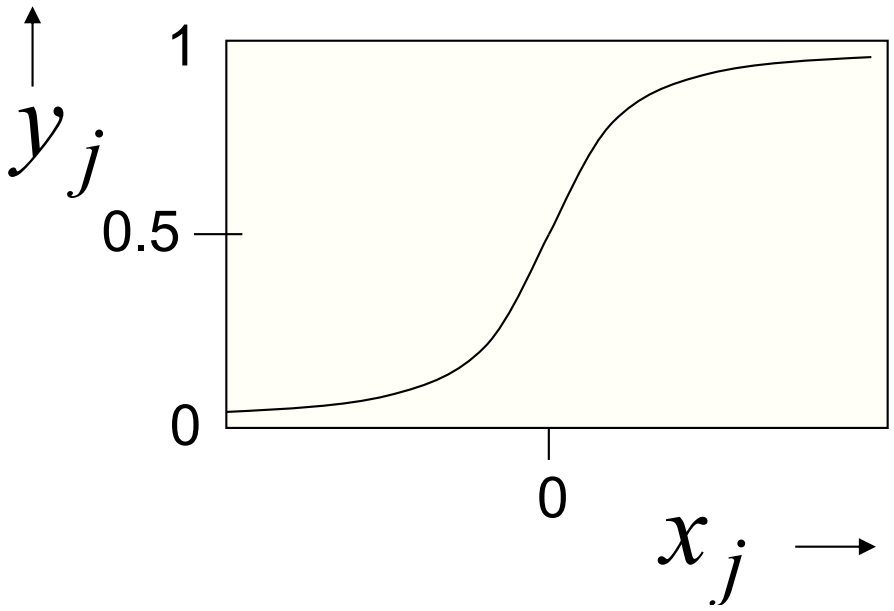
  The index indicates which layer a unit is in.





Non-linear neurons with smooth derivatives

- For backpropagation, we need neurons that have well-behaved derivatives.
  - Typically they use the logistic function
  - The output is a smooth function of inputs and weights.



$$x_j = b_j + \sum_i y_i w_{ij}$$

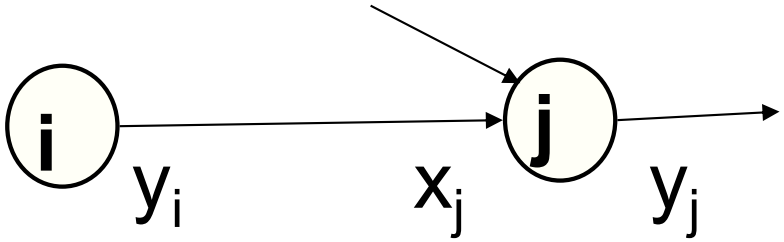
$$y_j = \frac{1}{1 + e^{-x_j}}$$

$$\frac{\partial x_j}{\partial w_{ij}} = y_i \qquad \frac{\partial x_j}{\partial y_i} = w_{ij}$$

$$\frac{dy_j}{dx_j} = y_j (1 - y_j)$$

Backpropagation

- J nodes
- Observations t_j
- Predictions y_j
- Energy function $E =$
- $\frac{dE}{dy_j} =$
- $\frac{dE}{dx_j} =$
- $\frac{dE}{dw_{ij}} =$
- $\frac{dE}{dy_i} = \sum_j^J$
- $\frac{dE}{dx_i} =$



TensorFlow and Matlab training

- `tf.train.Optimizer`
- `tf.train.GradientDescentOptimizer`
- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- `tf.train.MomentumOptimizer`
- `tf.train.AdamOptimizer`
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`

Choose a Multilayer Neural Network Training Function

R2018a

It is very difficult to know which training algorithm will be the fastest for a given problem. It depends on many factors, including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the network, the error goal, and whether the network is being used for pattern recognition (discriminant analysis) or function approximation (regression). This section compares the various training algorithms. Feedforward networks are trained on six different problems. Three of the problems fall in the pattern recognition category and the three others fall in the function approximation category. Two of the problems are simple "toy" problems, while the other four are "real world" problems. Networks with a variety of different architectures and complexities are used, and the networks are trained to a variety of different accuracy levels.

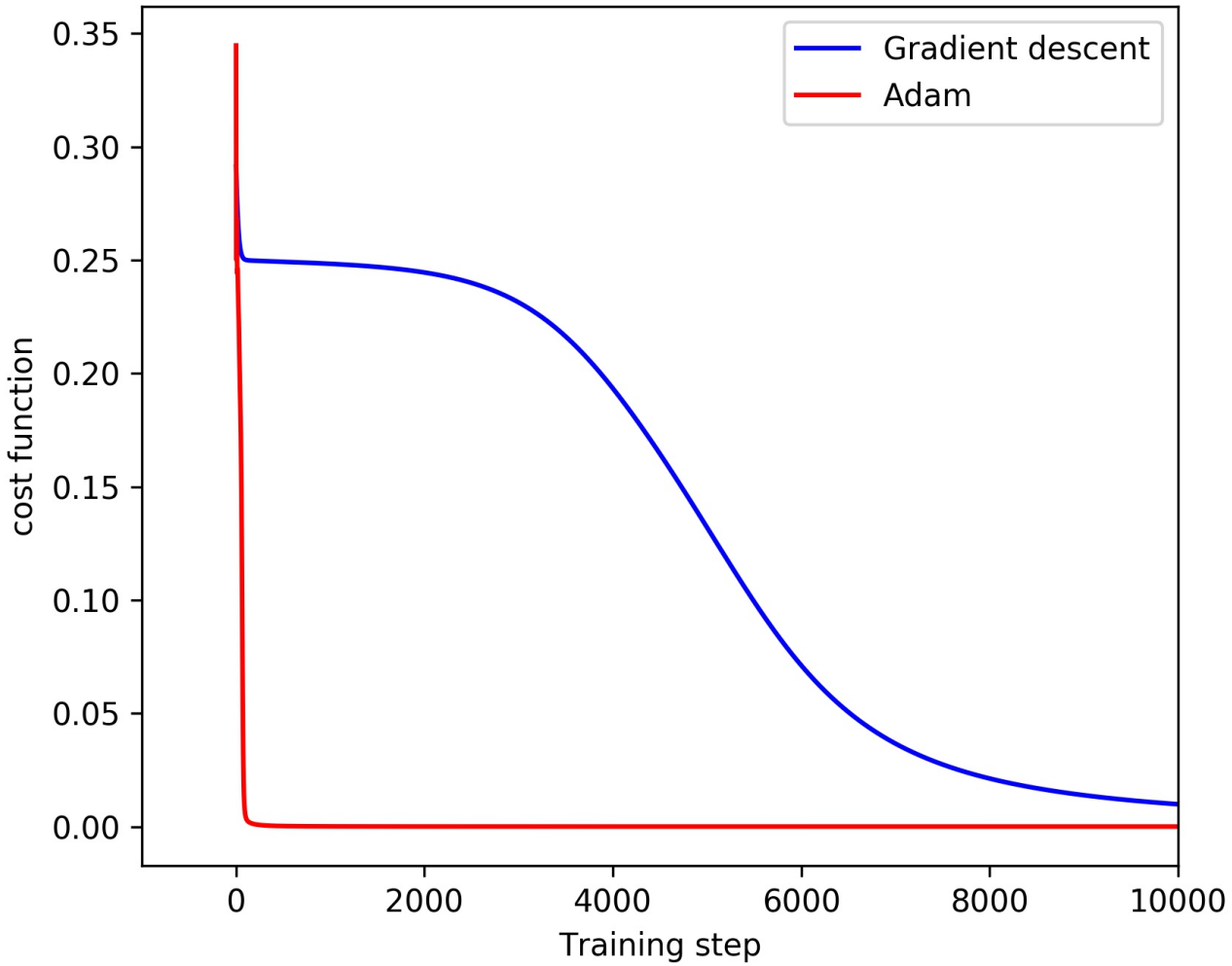
The following table lists the algorithms that are tested and the acronyms used to identify them.

Acronym	Algorithm	Description
LM	`trainlm`	Levenberg-Marquardt
BFG	`trainbfg`	BFGS Quasi-Newton
RP	`trainrp`	Resilient Backpropagation
SCG	`trainscg`	Scaled Conjugate Gradient
CGB	`traincgb`	Conjugate Gradient with Powell/Beale Restarts
CGF	`traincgf`	Fletcher-Powell Conjugate Gradient
CGP	`traincgp`	Polak-Ribiére Conjugate Gradient
OSS	`trainoss`	One Step Secant
GDX	`traingdx`	Variable Learning Rate Backpropagation

The following table lists the six benchmark problems and some characteristics of the networks, training processes, and computers used

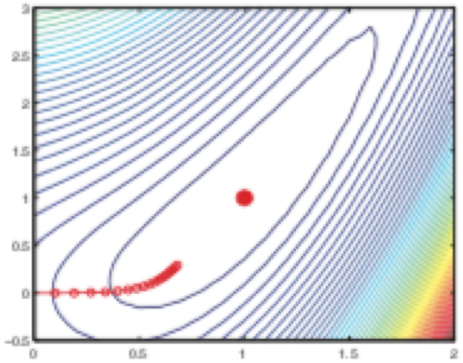
Gradient results

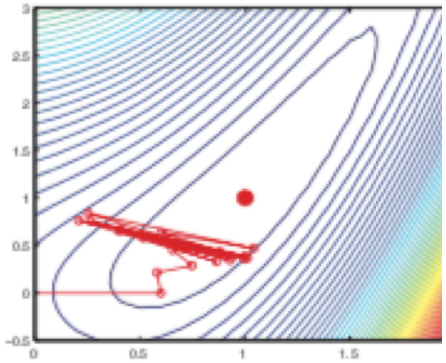
Maybe from tensorflow with ocean acoustics data



Gradients

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla E$$





- Batch
  - $E(\boldsymbol{w}_k) = \sum_n^N E_n(\boldsymbol{w}_k)$
- Stochastic gradient descent:
  - Pick just one data sample n (of N)
  - $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla E_n(\boldsymbol{w}_k)$
  - Less sensitive to global minima

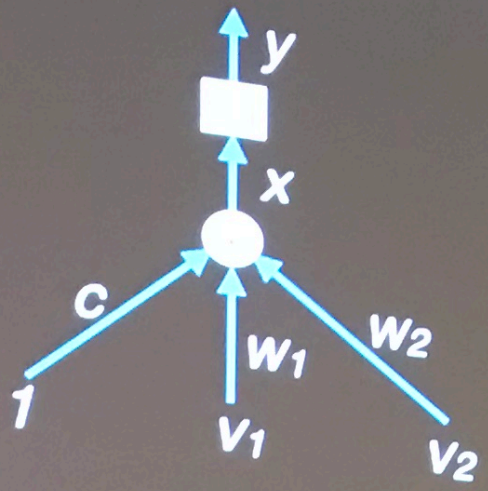
- Momentum
  - $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \nabla E + \mu(\boldsymbol{w}_k - \boldsymbol{w}_{k-1})$

- Avoid Overfitting (regularization, dropout, early stopping)

- 100's PhD thesis on how to optimize. Always backpropagation.

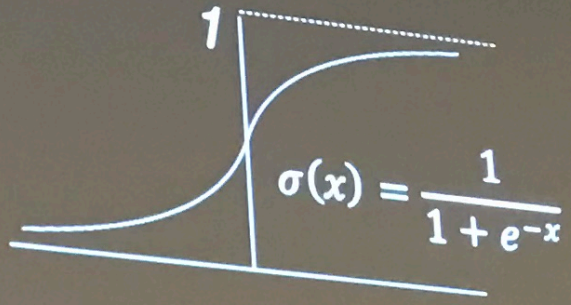
ICASSP 2018 Plenary

Stochastic Gradient Descent (SGD)



$$y = \sigma(x)$$

$$x = c + \mathbf{v}^T\mathbf{w}$$



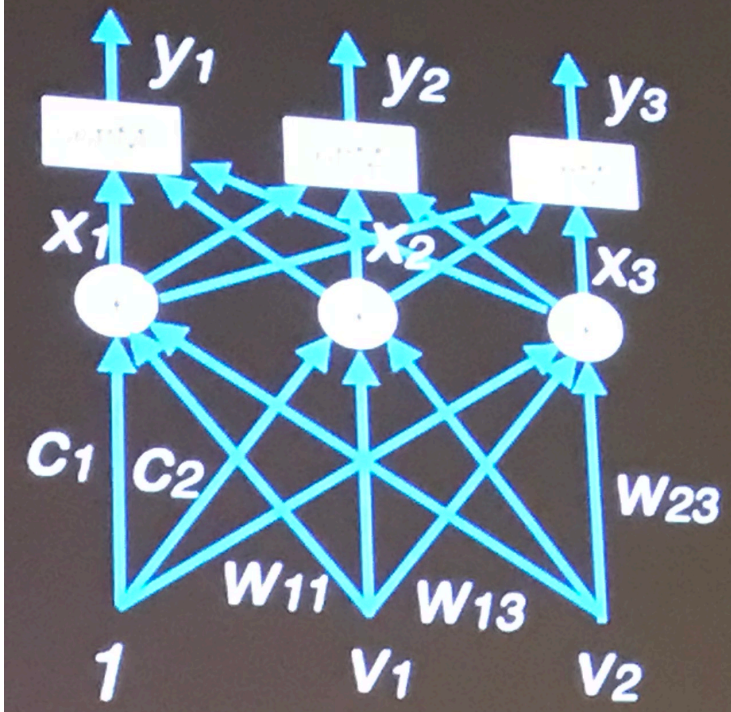
$$p(t|\mathbf{v}) = y^t(1-y)^{1-t}$$

$$L = \ln p(t|\mathbf{v}) = t\ln y + (1-t)\ln(1-y)$$

$$\frac{\partial L}{\partial w_1} = \left(\frac{\partial L}{\partial y}\right)\left(\frac{\partial y}{\partial x}\right)\left(\frac{\partial x}{\partial w_1}\right) = (y-t)v_1$$

ICASSP 2018 Plenary

N-ary Classification



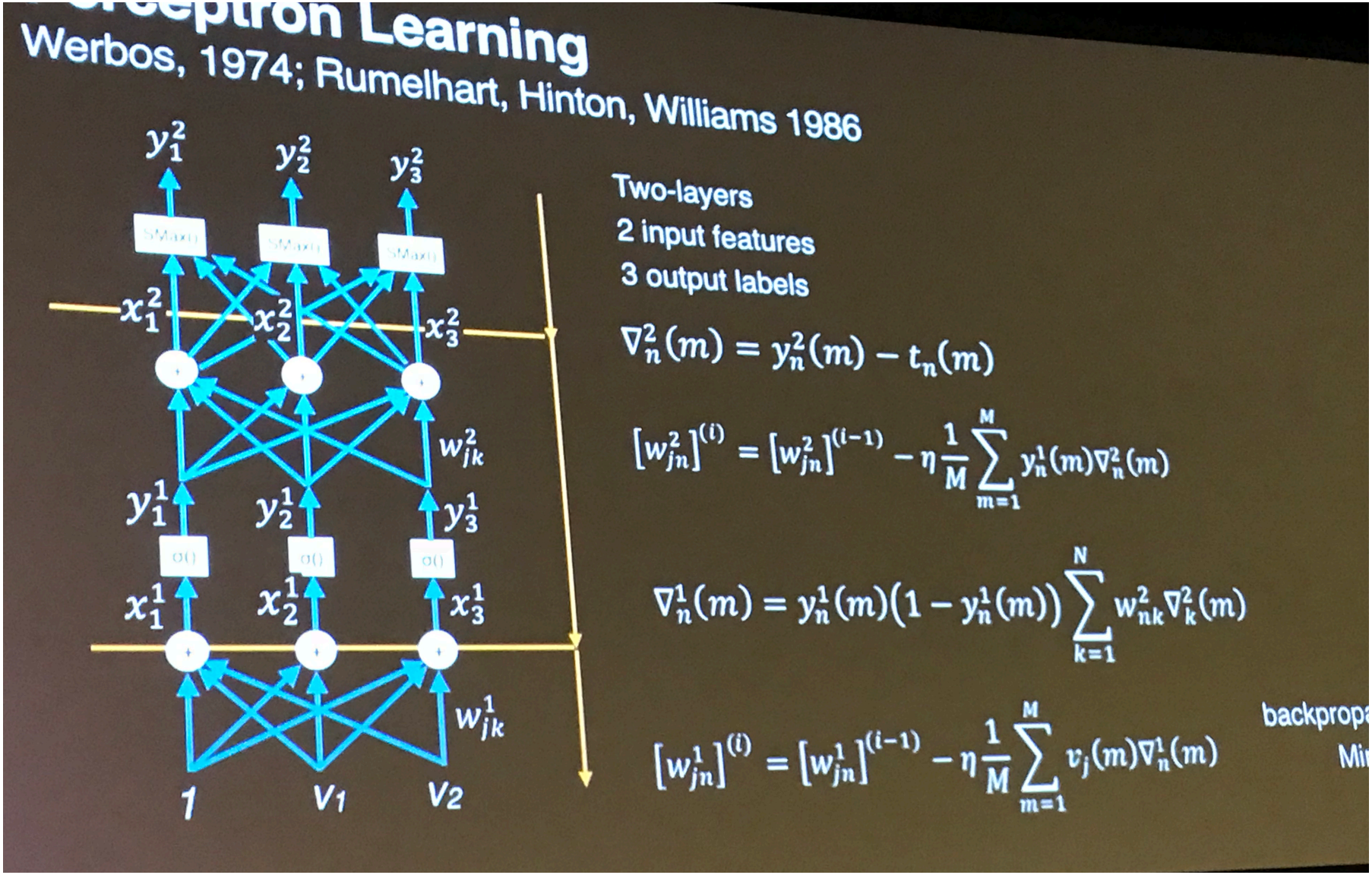
$$y_i = p(i|\mathbf{v}) = \frac{e^{x_i}}{\sum_{l=1}^{N} e^{x_l}}$$

Softmax

$$L = \sum_{i=1}^{N} t_i \ln y_i$$

$$w_{nj}^{(i)} = w_{nj}^{(i-1)} + \eta v_n^{(i-1)} \left(t_j - y_j^{(i-1)} \right)$$

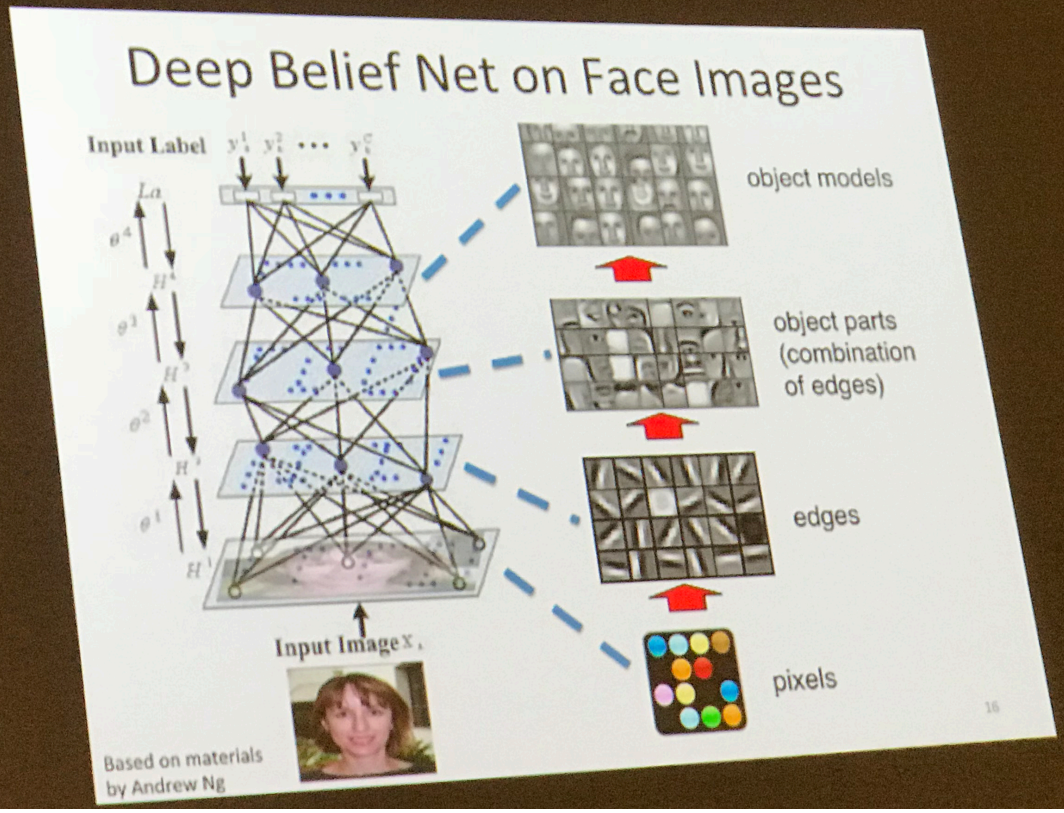
ICASSP 2018 Plenary



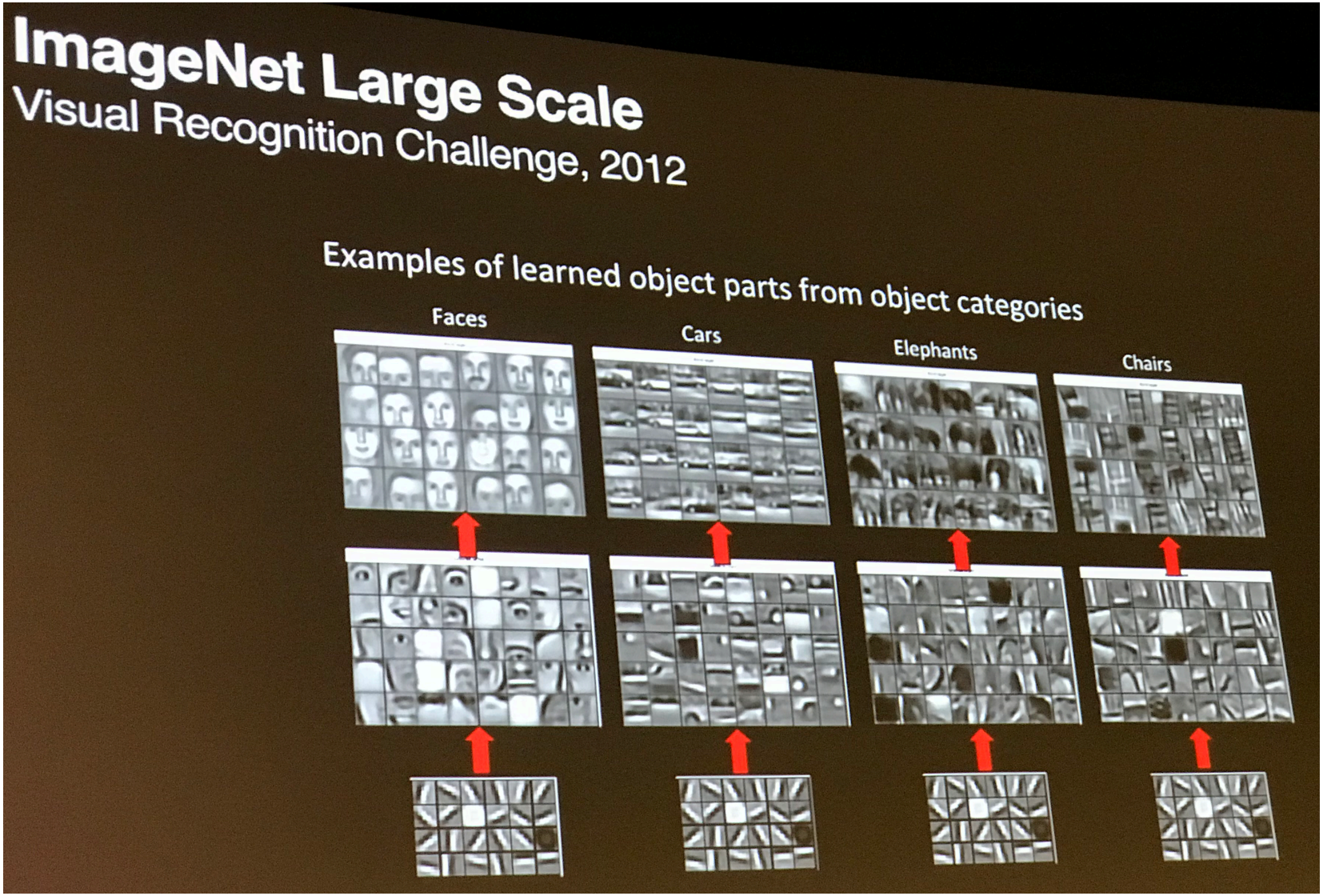
ICASSP 2018 Plenary

CNN on Face Images

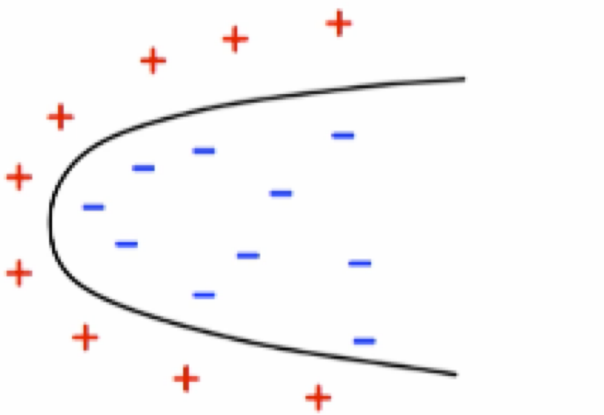
2012

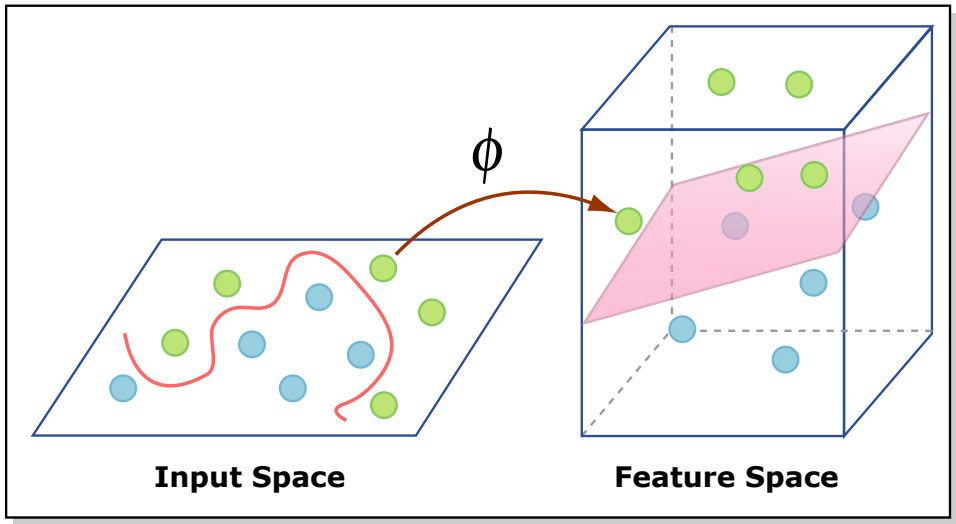


ICASSP 2018 Plenary



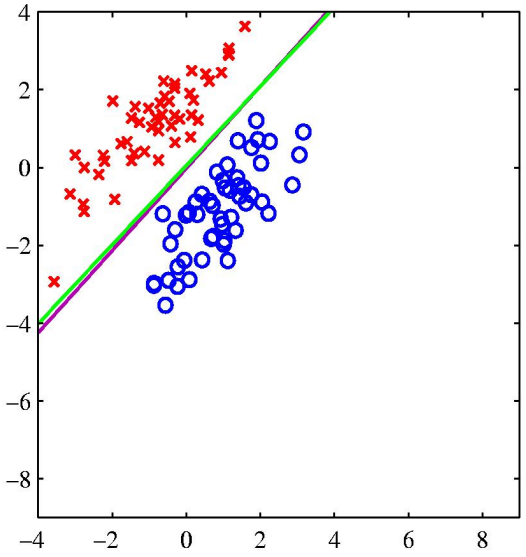
Lecture 9: Kernels

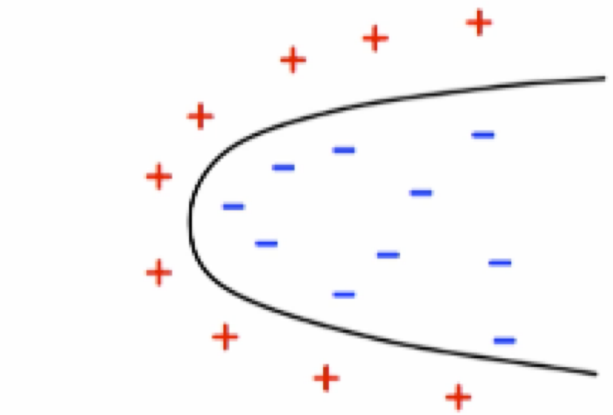




Basis expansion
Kernel trick

Basis expansion





LSQ for classification

Each class $\mathcal{C}_k$ is described by its own linear model so that

$$y_k(\mathbf{x}) = \mathbf{w}_k^{\mathrm{T}}\mathbf{x} + w_{k0} \tag{4.13}$$

where $k = 1, \ldots, K$. We can conveniently group these together using vector notation so that

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{x}} \tag{4.14}$$

Consider a training set $\{\boldsymbol{x}_n, \boldsymbol{t}_n\}, n = 1$ ...N
Define **X** and **T**

LSQ solution:

$$\widetilde{\mathbf{W}} = (\widetilde{\mathbf{X}}^{\mathrm{T}}\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^{\mathrm{T}}\mathbf{T} = \widetilde{\mathbf{X}}^{\dagger}\mathbf{T} \tag{4.16}$$

And prediction

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^{\mathrm{T}}\widetilde{\mathbf{x}} = \mathbf{T}^{\mathrm{T}}\left(\widetilde{\mathbf{X}}^{\dagger}\right)^{\mathrm{T}}\widetilde{\mathbf{x}}. \tag{4.17}$$

Dual representation, Sec 6.2

Primal problem: $\min_{\boldsymbol{w}} E(\boldsymbol{w})$

$$E = \frac{1}{2}\sum_n^N \{\boldsymbol{w}^T \boldsymbol{x}_n - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

Solution $\boldsymbol{w} = \boldsymbol{X}^+\boldsymbol{t} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}_M)^{-1}\boldsymbol{X}^T\boldsymbol{t}$

$= \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{t} = \boldsymbol{X}^T(\boldsymbol{K} + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{t} = \boldsymbol{X}^T\boldsymbol{a}$

The kernel is $\mathbf{K} = \boldsymbol{X}\boldsymbol{X}^T$

Dual representation is : $\min_{\boldsymbol{a}} E(\boldsymbol{a})$

$$E = \frac{1}{2}\sum_n^N \{\boldsymbol{w}^T \boldsymbol{x}_n - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \|\boldsymbol{K}\boldsymbol{a} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

Prediction

$$y = \boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{a}^T\boldsymbol{X}\boldsymbol{x} = \sum_n^N a_n \boldsymbol{x}_n^T\boldsymbol{x} = \sum_n^N a_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

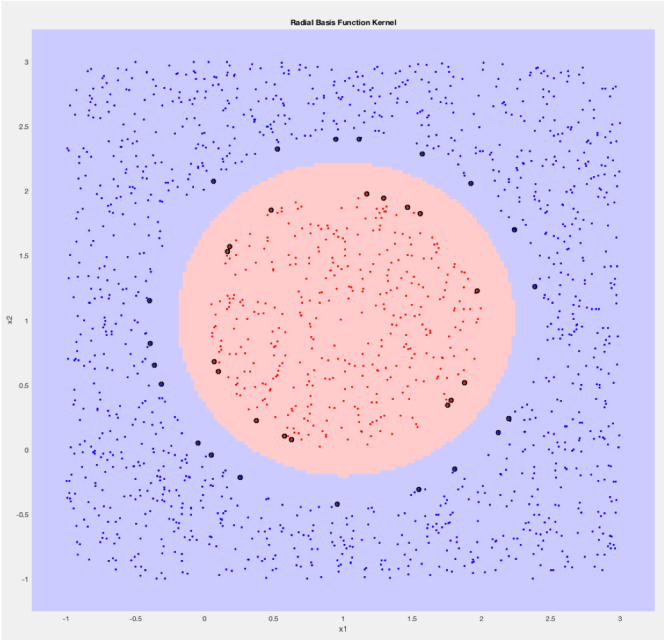
Dual representation, Sec 6.2

Prediction

$$y = \boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{a}^T\boldsymbol{X}\boldsymbol{x} = \sum_n^N a_n \boldsymbol{x}_n^T\boldsymbol{x} = \sum_n^N a_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- Often **a is sparse (…** Support vector machines**)**
- We don't need to know **x or** $\boldsymbol{\varphi}(\boldsymbol{x})$**.** $\boldsymbol{Just\ the\ Kernel}$

$$E(\boldsymbol{a}) = \|\boldsymbol{K}\boldsymbol{a} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$



Gaussian Kernels

- Gaussian Kernel

$$k(x, x') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}')\right)$$

Diagonal Σ: (this gives ARD)

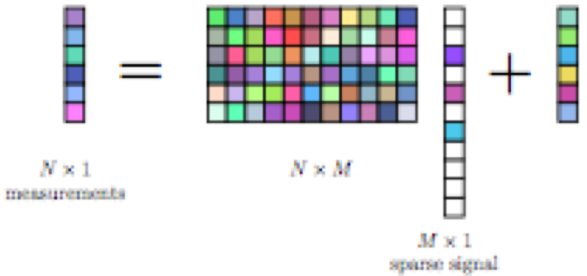
$$k(x, x') = \exp\left(-\frac{1}{2}\sum_{i}^{N}\frac{(x_i - x_i')^2}{\sigma_i^2}\right)$$

Isotropic σ_i^2 gives an RBF

$$k(x, x') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right)$$

Sparse Bayesian Learning (SBL)

$$\text{Model}: \mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}$$
$$\text{Prior}: \mathbf{x} \sim \mathcal{N}(\mathbf{x}; 0, \mathbf{\Gamma})$$
$$\mathbf{\Gamma} = \text{diag}(\gamma_1, \ldots, \gamma_M)$$
$$\text{Likelihood}: p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; \mathbf{A}\mathbf{x}, \sigma^2 \mathbf{I}_N)$$



$$\text{Evidence}: p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} = \mathcal{N}(\mathbf{y}; 0, \mathbf{\Sigma}_\mathbf{y})$$
$$\mathbf{\Sigma}_\mathbf{y} = \sigma^2 \mathbf{I}_N + \mathbf{A}\mathbf{\Gamma}\mathbf{A}^H$$

$$\text{SBL solution}: \hat{\mathbf{\Gamma}} = \arg\max_{\mathbf{\Gamma}} p(\mathbf{y})$$
$$= \arg\min_{\mathbf{\Gamma}} \left\{ \log|\mathbf{\Sigma}_\mathbf{y}| + \mathbf{y}^H \mathbf{\Sigma}_\mathbf{y}^{-1} \mathbf{y} \right\}$$

M.E.Tipping, "Sparse Bayesian learning and the relevance vector machine," Journal of Machine Learning Research, June 2001.

Gaussian Kernels

- Gaussian Kernel

$$k(x, x') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}')\right)$$

Diagonal Σ: (this gives ARD)

$$k(x, x') = \exp\left(-\frac{1}{2}\sum_{i}^{N}\frac{(x_i - x_i')^2}{\sigma_i^2}\right)$$

Isotropic σ_i^2 gives an RBF

$$k(x, x') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right)$$

Commonly used kernels

Polynomial: $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}.\mathbf{y}+1)^p$

Gaussian radial basis function: $K(\mathbf{x},\mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$

Neural net: $K(\mathbf{x},\mathbf{y}) = \tanh(k\,\mathbf{x}.\mathbf{y} - \delta)$

Parameters that the user must choose

For the neural network kernel, there is one "hidden unit" per support vector, so the process of fitting the maximum margin hyperplane decides how many hidden units to use. Also, it may violate Mercer's condition.

Kernels

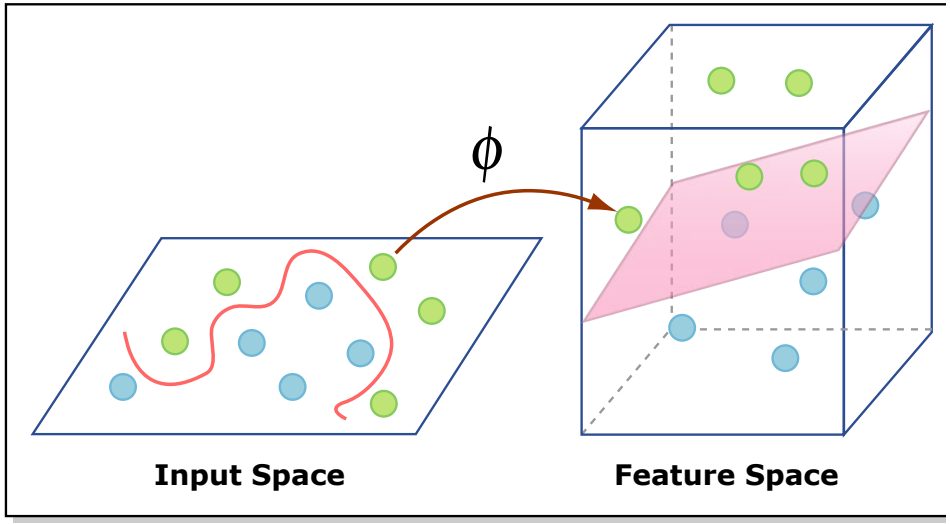
We might want to consider something more complicated than a linear model:

Example 1: $[x^{(1)}, x^{(2)}] \rightarrow \mathbf{\Phi}\left([x^{(1)}, x^{(2)}]\right) = \left[x^{(1)2}, x^{(2)2}, x^{(1)}x^{(2)}\right]$

Information unchanged, but now we have a **linear** classifier on the transformed points.

With the kernel trick, we just need kernel

$k(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{\Phi}(\boldsymbol{a})^T \, \boldsymbol{\Phi}(\boldsymbol{b})$



Example 4:

$$\begin{aligned} k(\mathbf{x},\mathbf{z}) &= (\mathbf{x}^T\mathbf{z}+c)^2 = \left(\sum_{j=1}^{n} x^{(j)}z^{(j)} + c\right)\left(\sum_{\ell=1}^{n} x^{(\ell)}z^{(\ell)} + c\right) \\ &= \sum_{j=1}^{n}\sum_{\ell=1}^{n} x^{(j)}x^{(\ell)}z^{(j)}z^{(\ell)} + 2c\sum_{j=1}^{n} x^{(j)}z^{(j)} + c^2 \\ &= \sum_{j,\ell=1}^{n} (x^{(j)}x^{(\ell)})(z^{(j)}z^{(\ell)}) + \sum_{j=1}^{n} (\sqrt{2c}x^{(j)})(\sqrt{2c}z^{(j)}) + c^2, \end{aligned}$$

and in $n = 3$ dimensions, one possible feature map is:

$$\mathbf{\Phi}(\mathbf{x}) = [x^{(1)2}, x^{(1)}x^{(2)}, ..., x^{(3)2}, \sqrt{2c}x^{(1)}, \sqrt{2c}x^{(2)}, \sqrt{2c}x^{(3)}, c]$$

and c controls the relative weight of the linear and quadratic terms in the inner product.

Even more generally, if you wanted to, you could choose the kernel to be any higher power of the regular inner product.

Solving a Rank-Deficient System

If A is *m*-by-*n* with *m* > *n* and full rank *n*, each of the three statements

```
x = A\b
x = pinv(A)*b
x = inv(A'*A)*A'*b
```

theoretically computes the same least-squares solution x, although the backslash operator does it faster.

Nice slide, But why?

However, if A does not have full rank, the solution to the least-squares problem is not unique. There are many vectors x that minimize

```
norm(A*x -b)
```

The solution computed by x = A\b is a basic solution; it has at most *r* nonzero components, where *r* is the rank of A. The solution computed by x = pinv(A)*b is the minimal norm solution because it minimizes norm(x). An attempt to compute a solution with x = inv(A'*A)*A'*b fails because A'*A is singular.