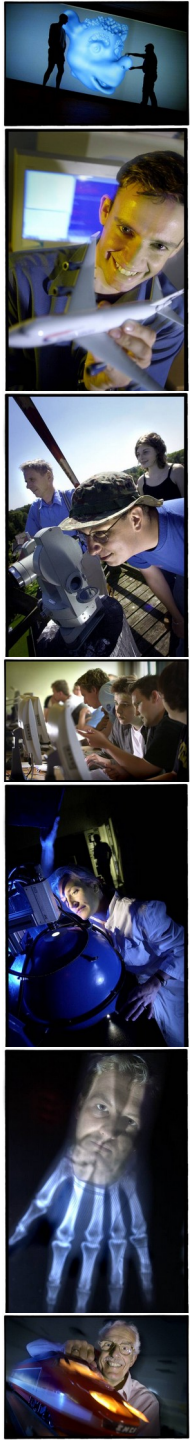
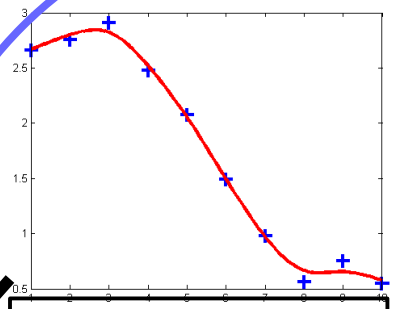


Neural Networks

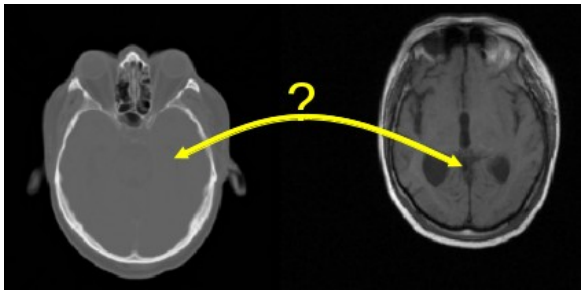
Course 22525

Koen Van Leemput
DTU HealthTech
Technical University of Denmark

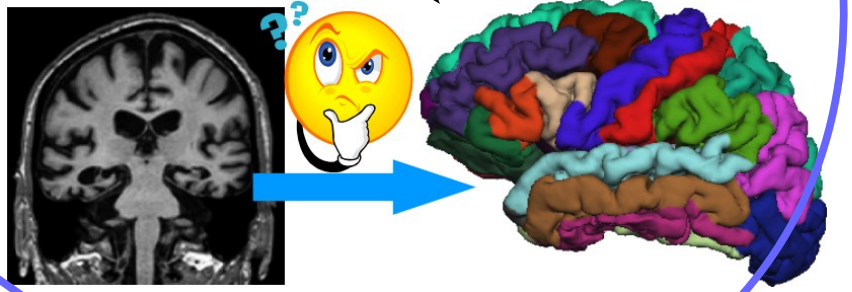
Course structure



Fitting functions



Registration



Segmentation

Remember regression?

- $f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x})$
- Training set $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$

  Input vector: $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^{\mathrm{T}}$

  Corresponding output: y_i

- Estimate parameters

$$\boldsymbol{\theta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$$

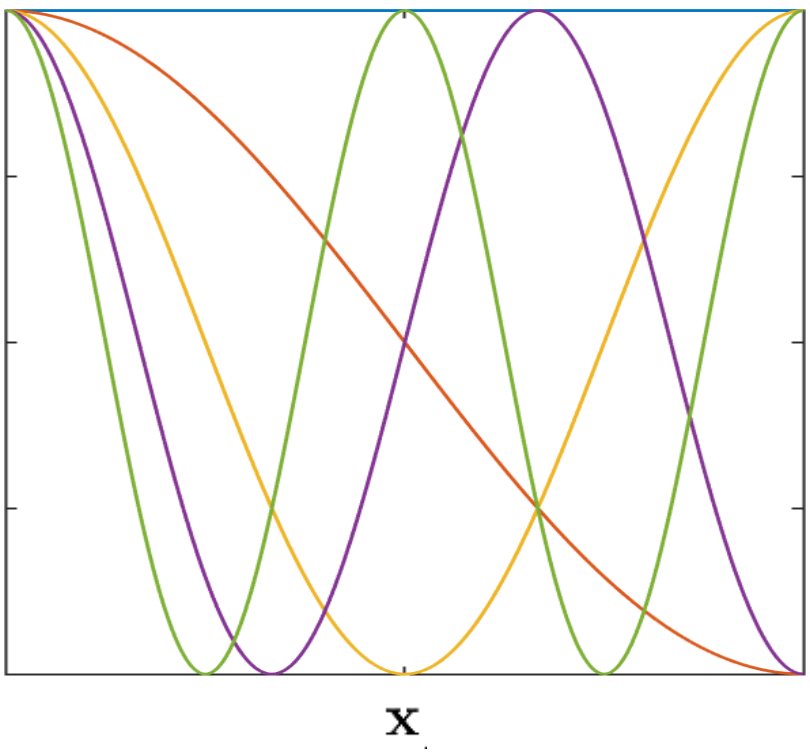
by minimizing the cost

$$\sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2$$

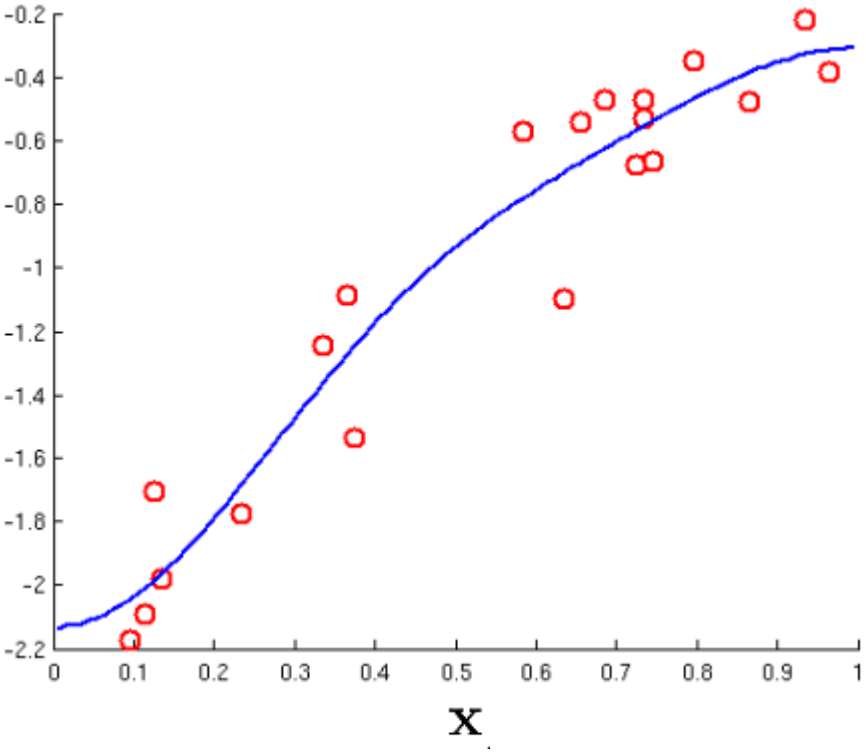
Remember regression?

Example: p=1 and M=5 cosines

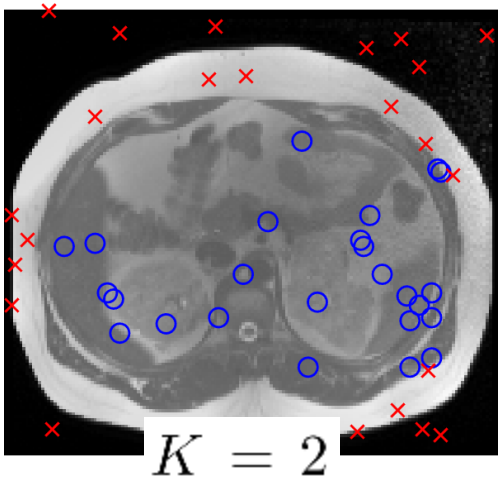
$$\phi_m(\mathbf{x})$$

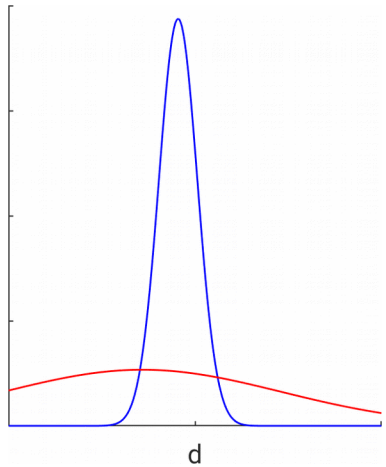


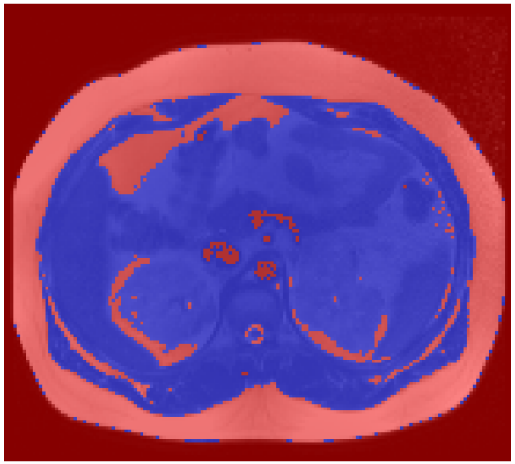
$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x})$$

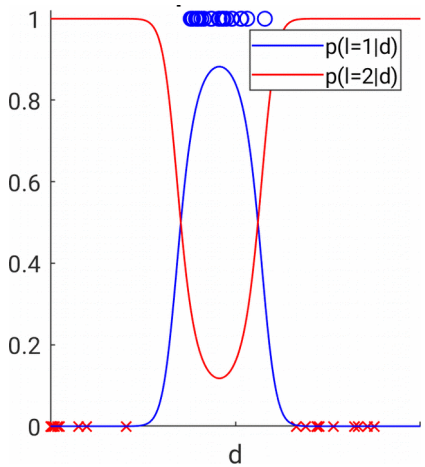


Remember Gaussian mixture model?



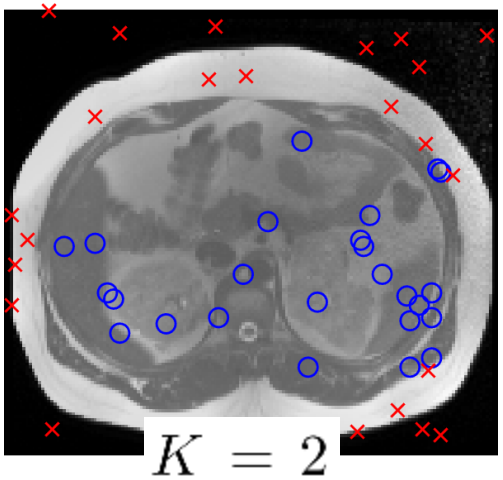


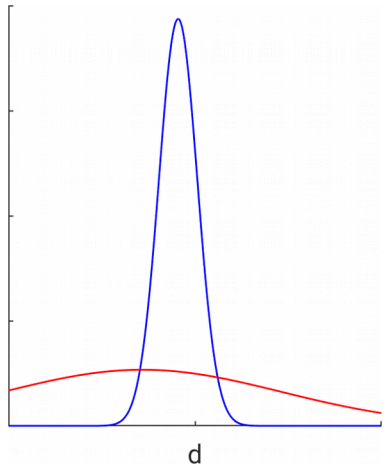


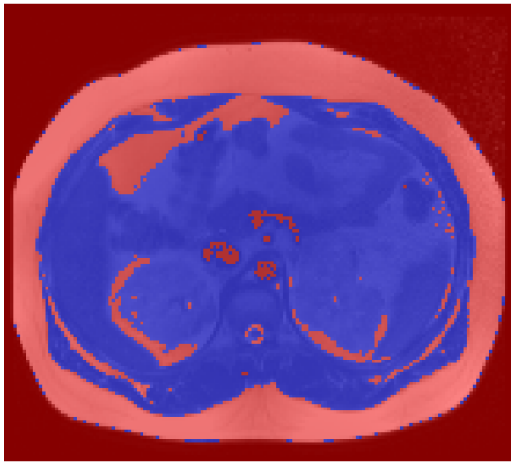


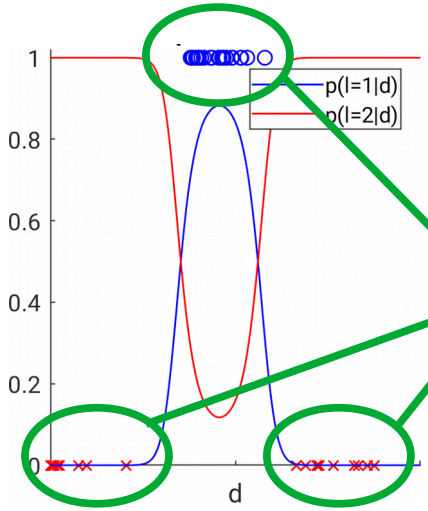
Posterior using Bayes' rule: $p(l = k|d, \boldsymbol{\theta}) = \dfrac{\mathcal{N}(d|\mu_k, \sigma_k^2)\pi_k}{\sum_{k'} \mathcal{N}(d|\mu_{k'}, \sigma_{k'}^2)\pi_{k'}}$

Remember Gaussian mixture model?







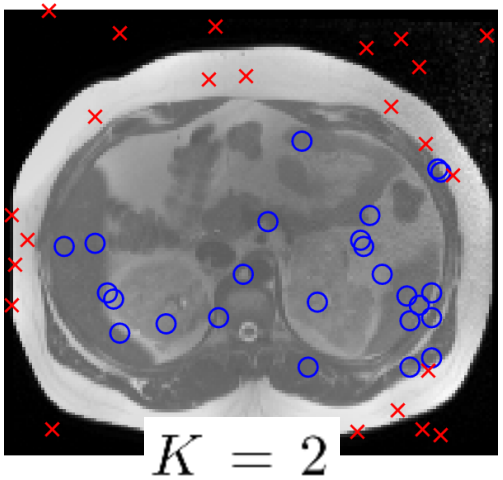


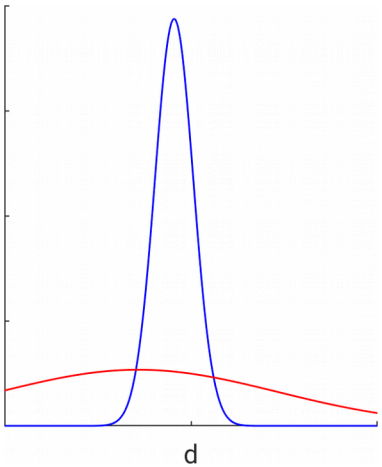
New notation/terminology:

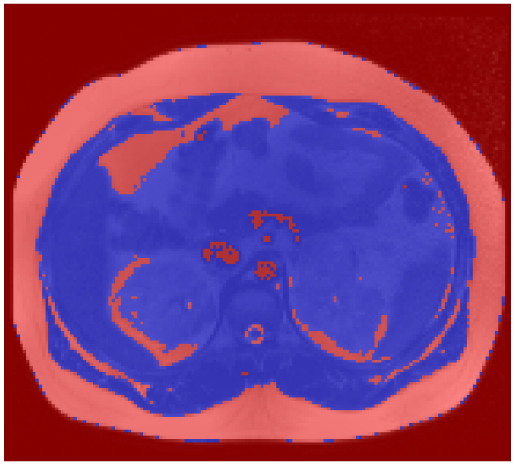
- “Training samples”
- “y=1” if l=1
- “y=0” if l=2

Posterior using Bayes’ rule: $p(l = k|d, \boldsymbol{\theta}) = \dfrac{\mathcal{N}(d|\mu_k, \sigma_k^2)\pi_k}{\sum_{k'} \mathcal{N}(d|\mu_{k'}, \sigma_{k'}^2)\pi_{k'}}$

Remember Gaussian mixture model?





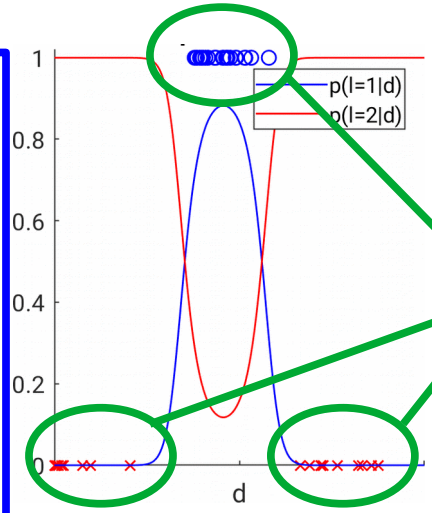


This lecture: can we get a "classifier"

$$p(l|d,\boldsymbol{\theta})$$

directly

<u>without a model</u>?



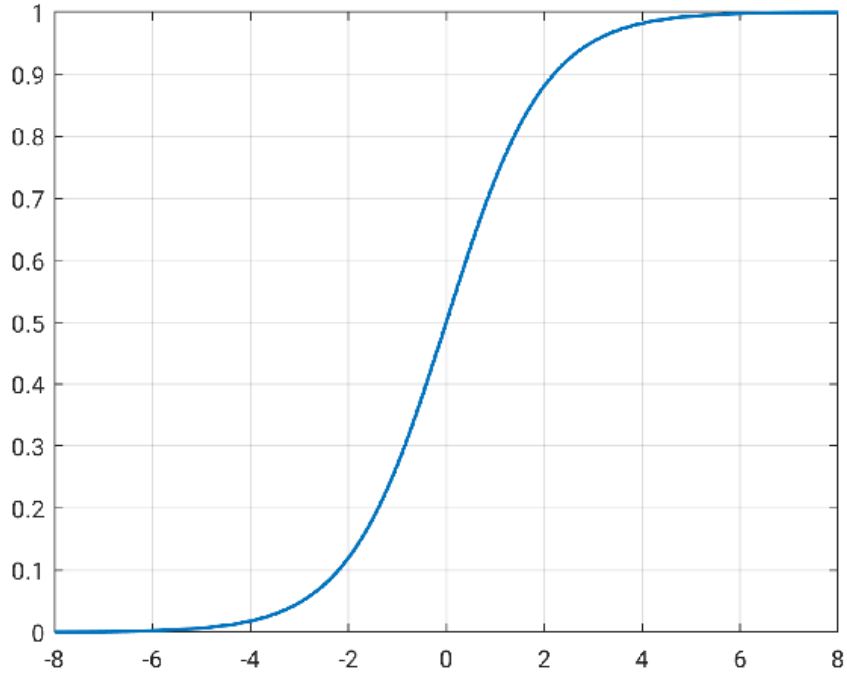
New notation/terminology:

- "Training samples"
- "y=1" if l=1
- "y=0" if l=2

Posterior using Bayes' rule: $p(l=k|d,\boldsymbol{\theta}) = \dfrac{\mathcal{N}(d|\mu_k,\sigma_k^2)\pi_k}{\sum_{k'}\mathcal{N}(d|\mu_{k'},\sigma_{k'}^2)\pi_{k'}}$

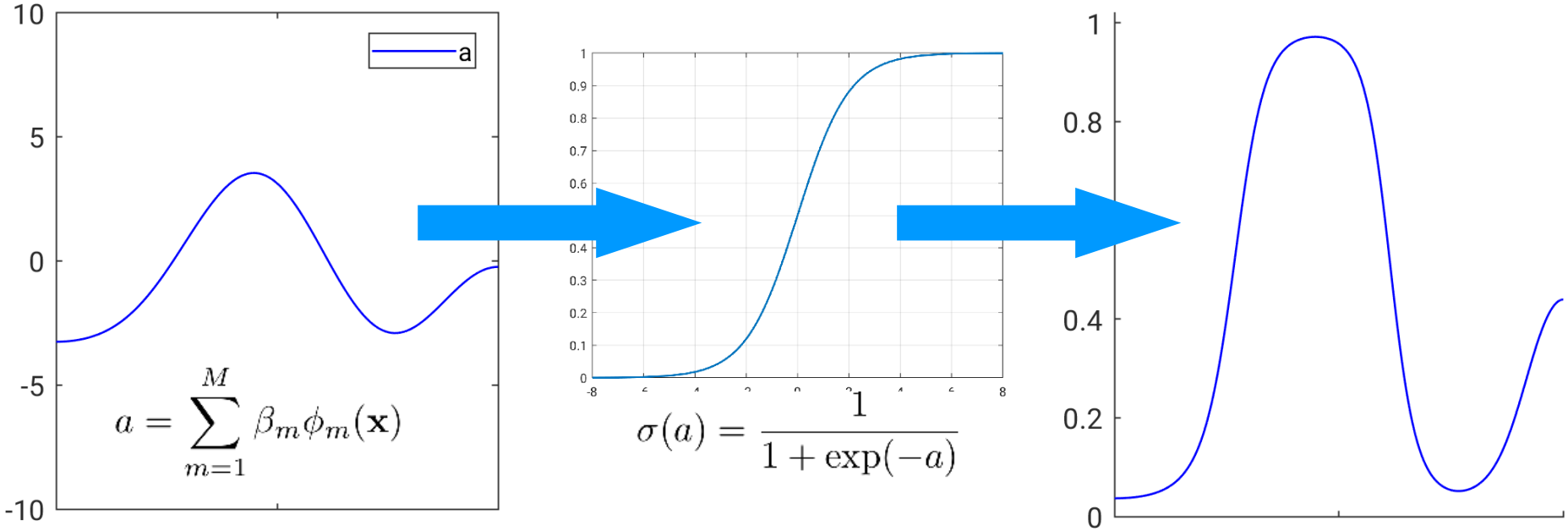
Logistic regression

- Logistic function as a “squashing” function



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

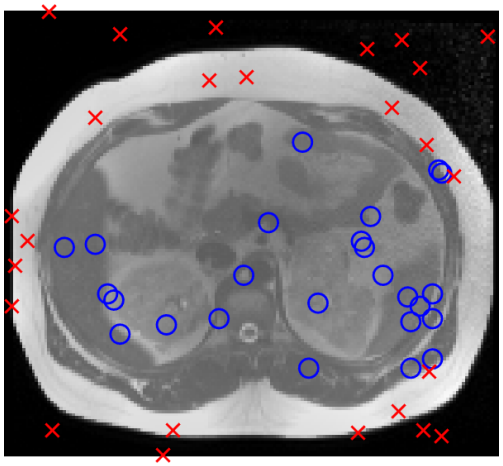
Logistic regression



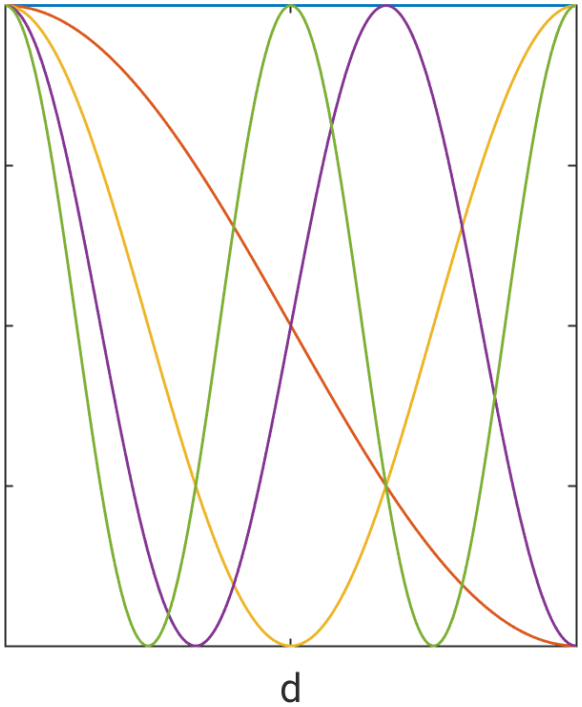
- $p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x})$ where $f(\mathbf{x}) = \sigma\left(\sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x})\right)$

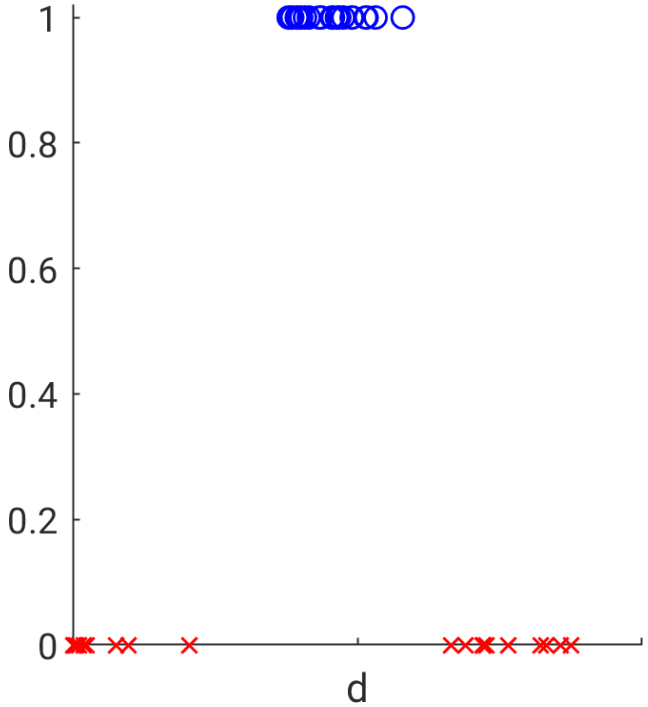
- Of course: $p(y = 0|\mathbf{x}, \boldsymbol{\theta}) = 1 - p(y = 1|\mathbf{x}, \boldsymbol{\theta}) = 1 - f(\mathbf{x})$

Voxel-based classifier

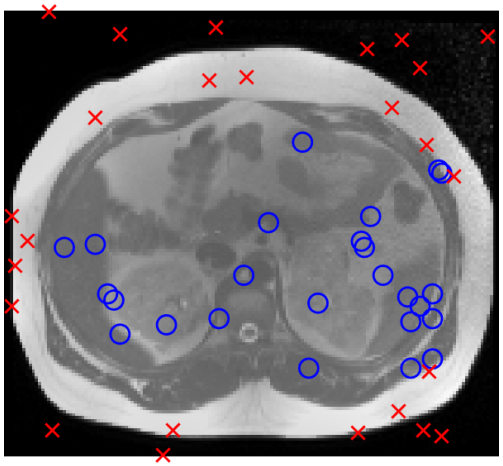


- Training data $\{\mathbf{x}_i, y_i\}_{i=1}^N$
  with $\mathbf{x}_i = d_i$ (i.e., $p = 1$) and $y_i \in \{0, 1\}$
- Estimate parameters $\boldsymbol{\theta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$
  by maximizing the likelihood function $\prod_{i=1}^{N} p(y_i | \mathbf{x}_i, \boldsymbol{\theta})$

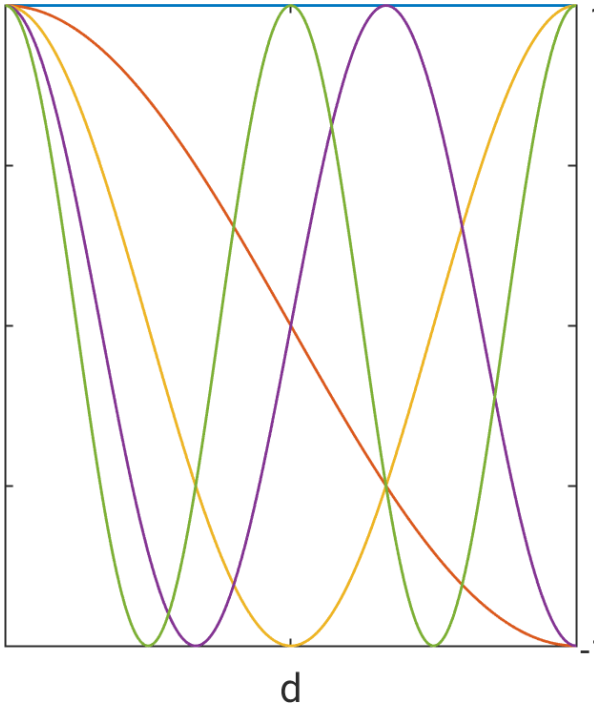


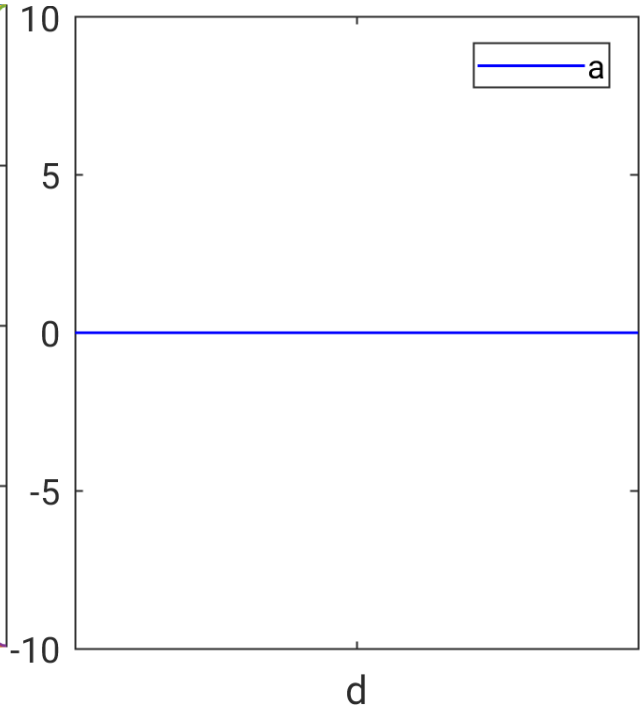


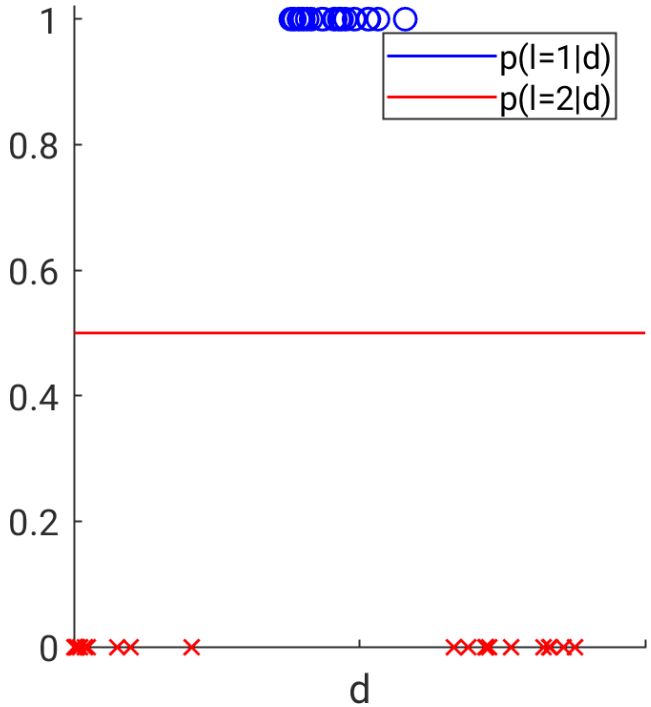
Voxel-based classifier



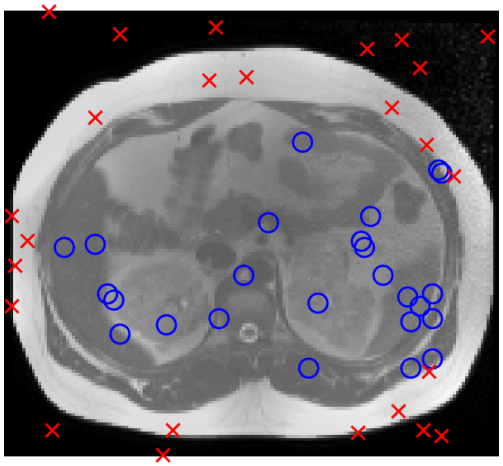
- Training data $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$
  with $\mathbf{x}_i = d_i$ (i.e., $p = 1$) and $y_i \in \{0, 1\}$
- Estimate parameters $\boldsymbol{\theta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$
  by maximizing the likelihood function $\prod_{i=1}^{N} p(y_i | \mathbf{x}_i, \boldsymbol{\theta})$



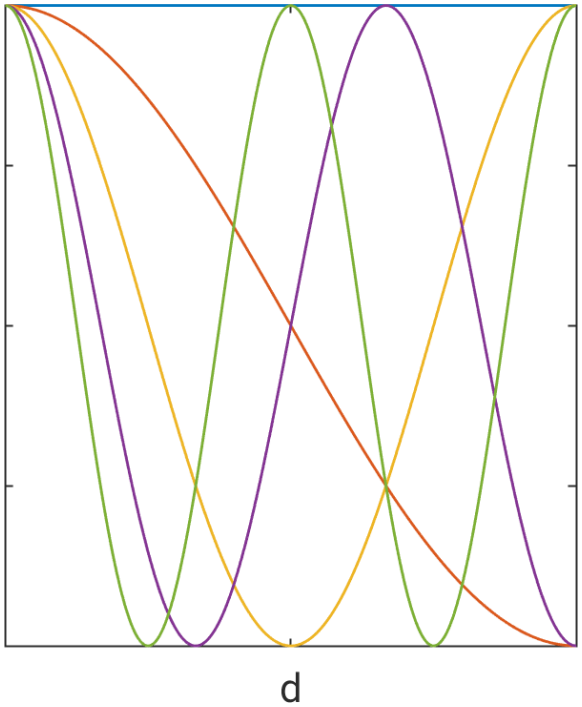


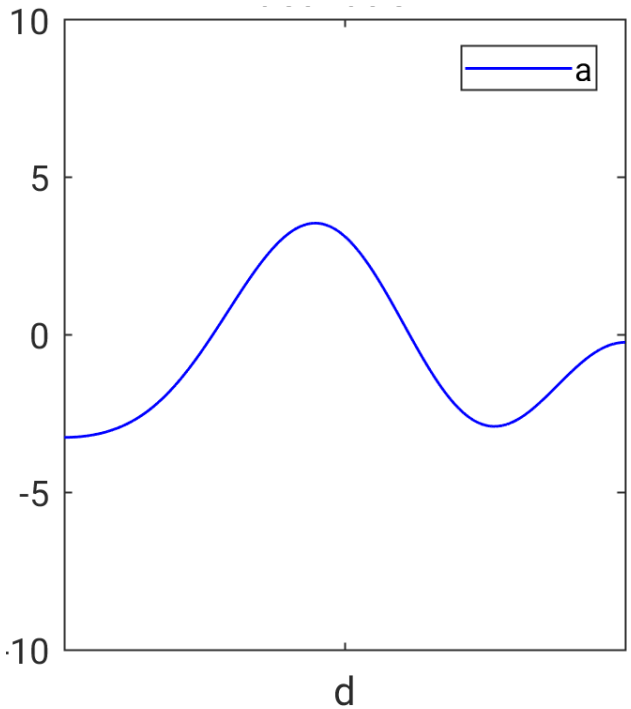


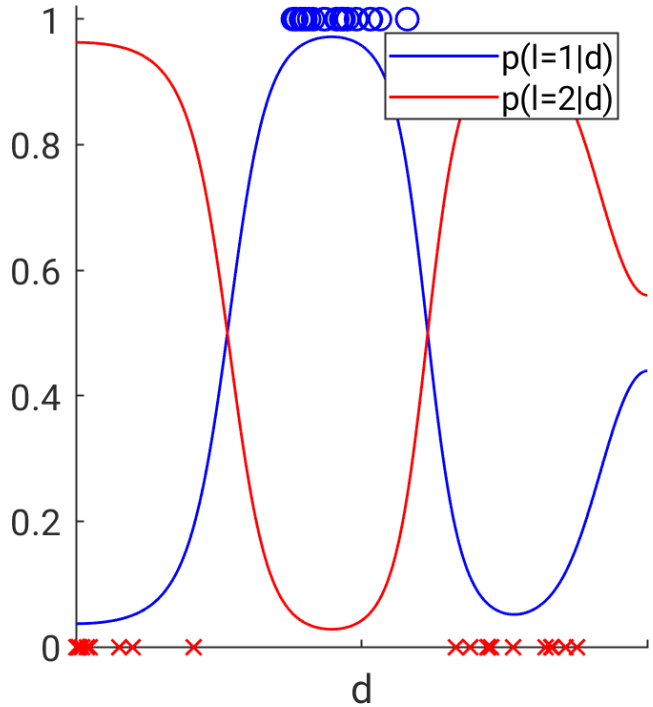
Voxel-based classifier



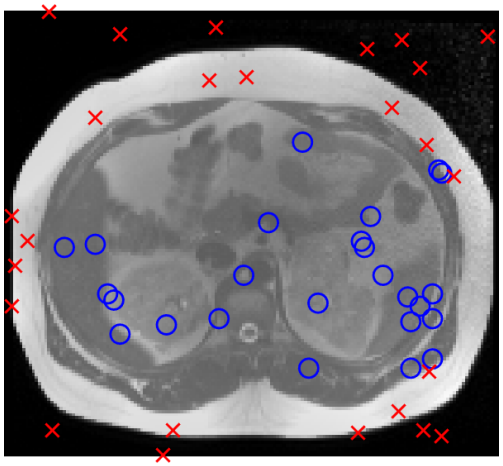
- Training data $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$
  with $\mathbf{x}_i = d_i$ (i.e., $p = 1$) and $y_i \in \{0, 1\}$
- Estimate parameters $\boldsymbol{\theta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$
  by maximizing the likelihood function $\prod_{i=1}^{N} p(y_i | \mathbf{x}_i, \boldsymbol{\theta})$



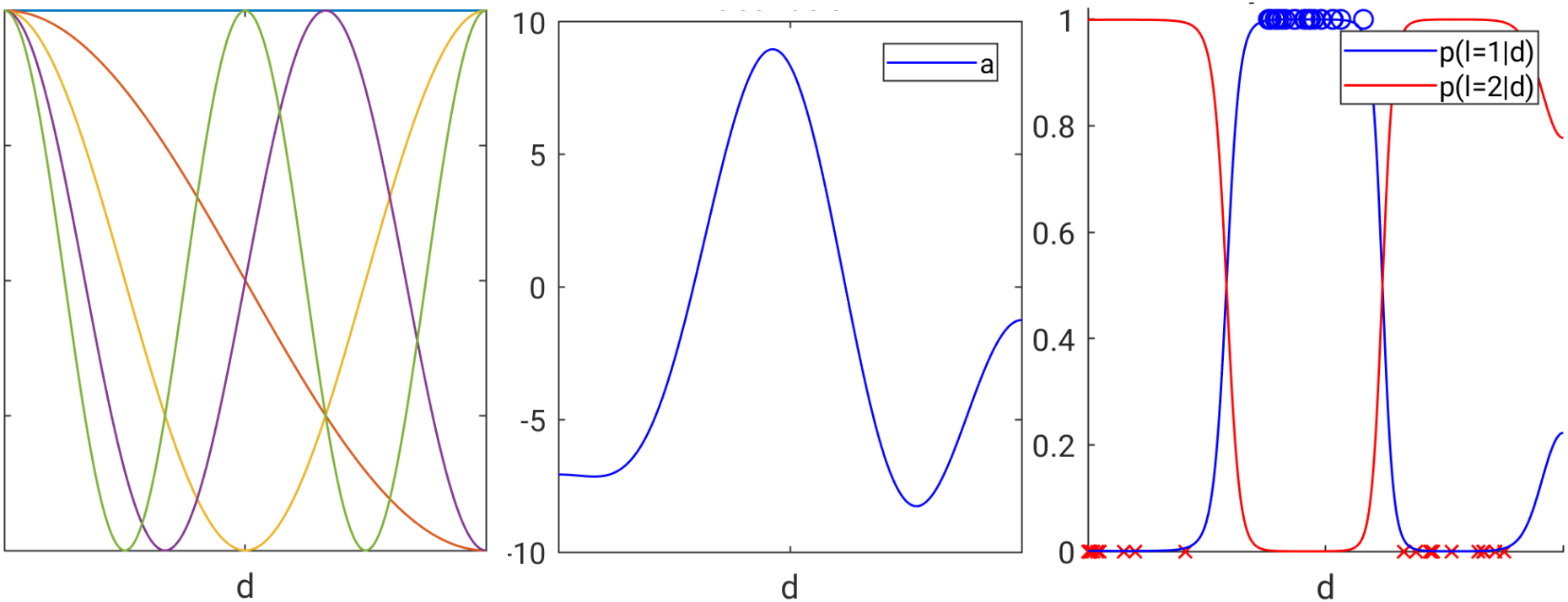




Voxel-based classifier



- Training data $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$
  with $\mathbf{x}_i = d_i$ (i.e., $p = 1$) and $y_i \in \{0, 1\}$
- Estimate parameters $\boldsymbol{\theta} = (\beta_1, \ldots, \beta_M)^{\mathrm{T}}$
  by maximizing the likelihood function $\prod_{i=1}^{N} p(y_i | \mathbf{x}_i, \boldsymbol{\theta})$

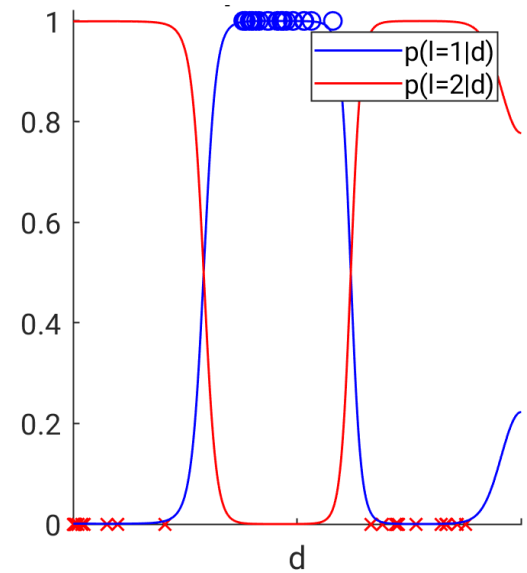


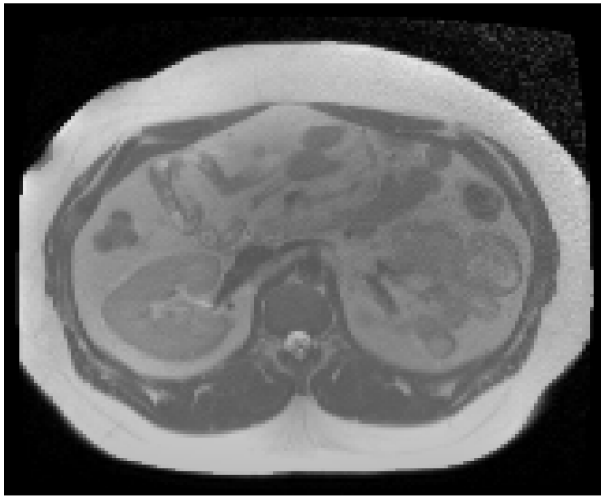
Voxel-based classifier

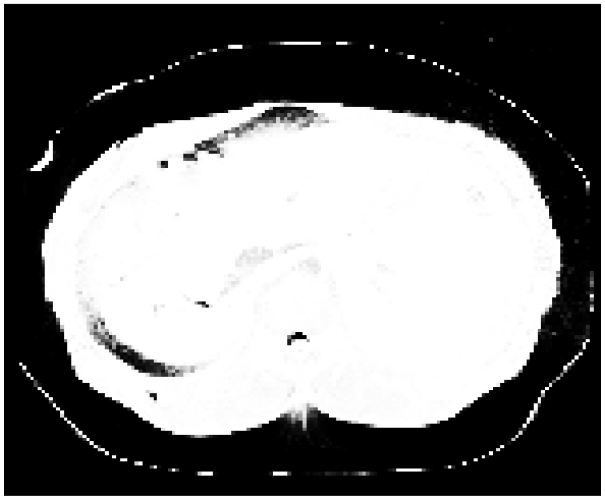
- Once trained keep the classifier

$$p(l = 1|d, \hat{\boldsymbol{\theta}})$$

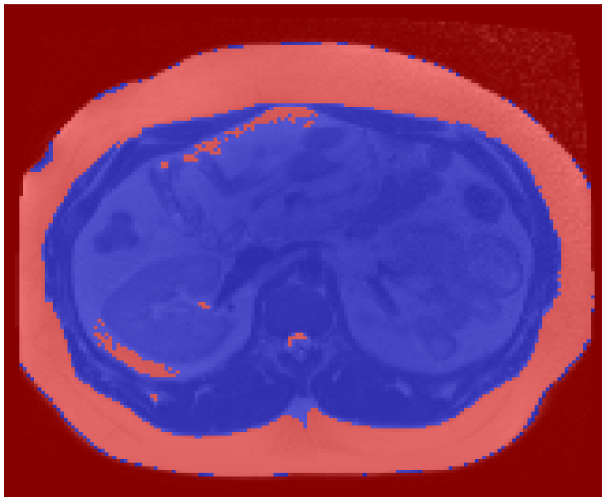
- Simply apply it to new data







$p(l = 1|d, \hat{\boldsymbol{\theta}})$



$p(l = 1|d, \hat{\boldsymbol{\theta}}) > 0.5$

Optimization algorithm for training

- Maximizing the likelihood function $\prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$ is equivalent to minimizing

$$E_N(\boldsymbol{\theta}) = -\log \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$$
$$= -\sum_{i=1}^{N} \{y_i \log f(\mathbf{x}_i) + (1 - y_i) \log [1 - f(\mathbf{x}_i)]\}$$

step size (user-specified)

- Gradient descent: $\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \nu \nabla E_N(\boldsymbol{\theta}^{(\tau)})$

  with gradient $\nabla E_N(\boldsymbol{\theta}) = \frac{\partial E_N}{\partial \boldsymbol{\theta}}$

- Stochastic gradient descent: use only $N' \ll N$ randomly sampled training points, and approximate:

$$\nabla E_N(\boldsymbol{\theta}) \simeq \frac{N}{N'} \nabla E_{N'}(\boldsymbol{\theta})$$

More fun: patch-based classifier

- Classify 3x3 image “patches”:

  intensity of the pixel to be classified + intensities of 8 neighboring pixels

- $\mathbf{x}$ is now a 9-dimensional vector ($p = 9$), but otherwise everything is the same:

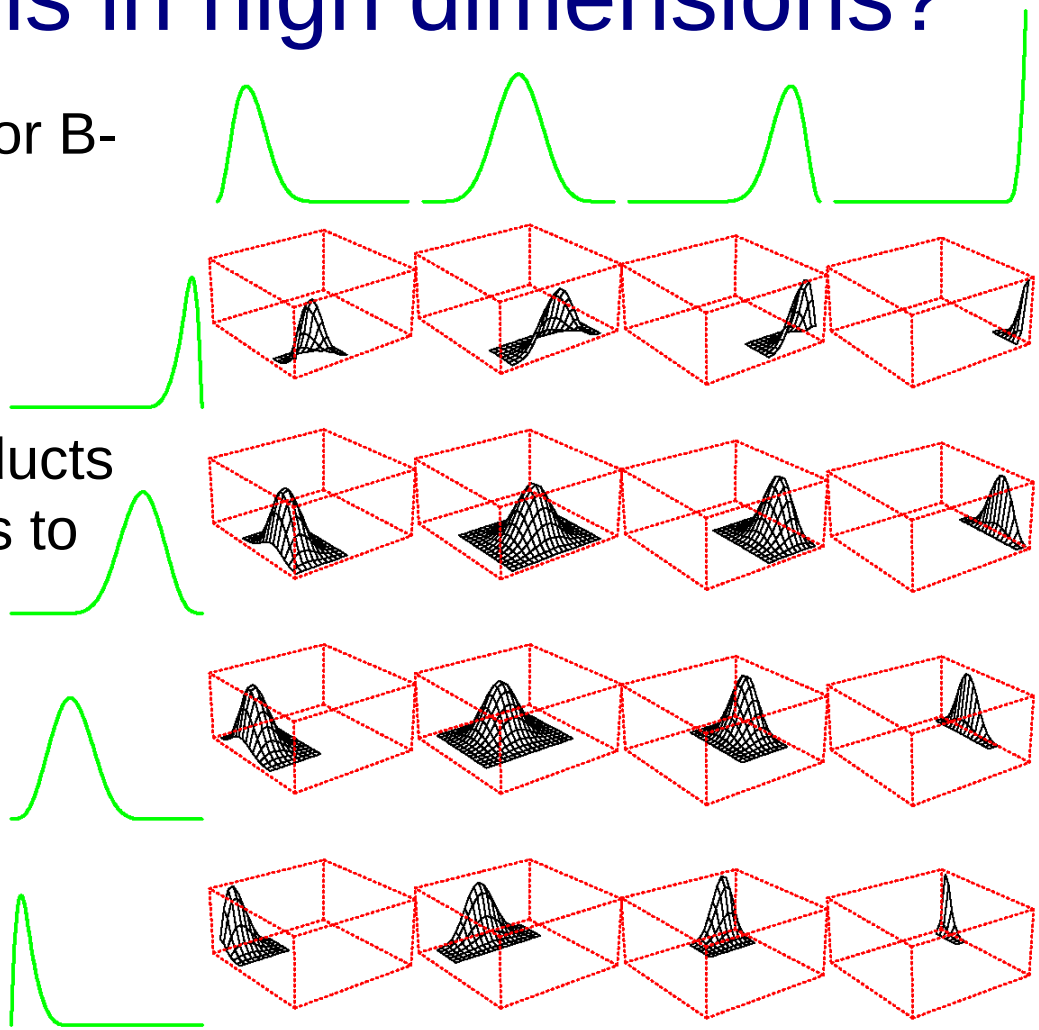
$$p(y = 1|\mathbf{x}, \hat{\boldsymbol{\theta}}) = \sigma\left(\sum_{m=1}^{M} \hat{\beta}_m \phi_m(\mathbf{x})\right)$$

- But how to choose basis functions $\phi_m(\mathbf{x})$ in a 9-dimensional space?

Basis functions in high dimensions?

- Idea: remember the tensor B-spline trick?

  Example: take outer products of four 1D basis functions to "make" sixteen 2D basis functions



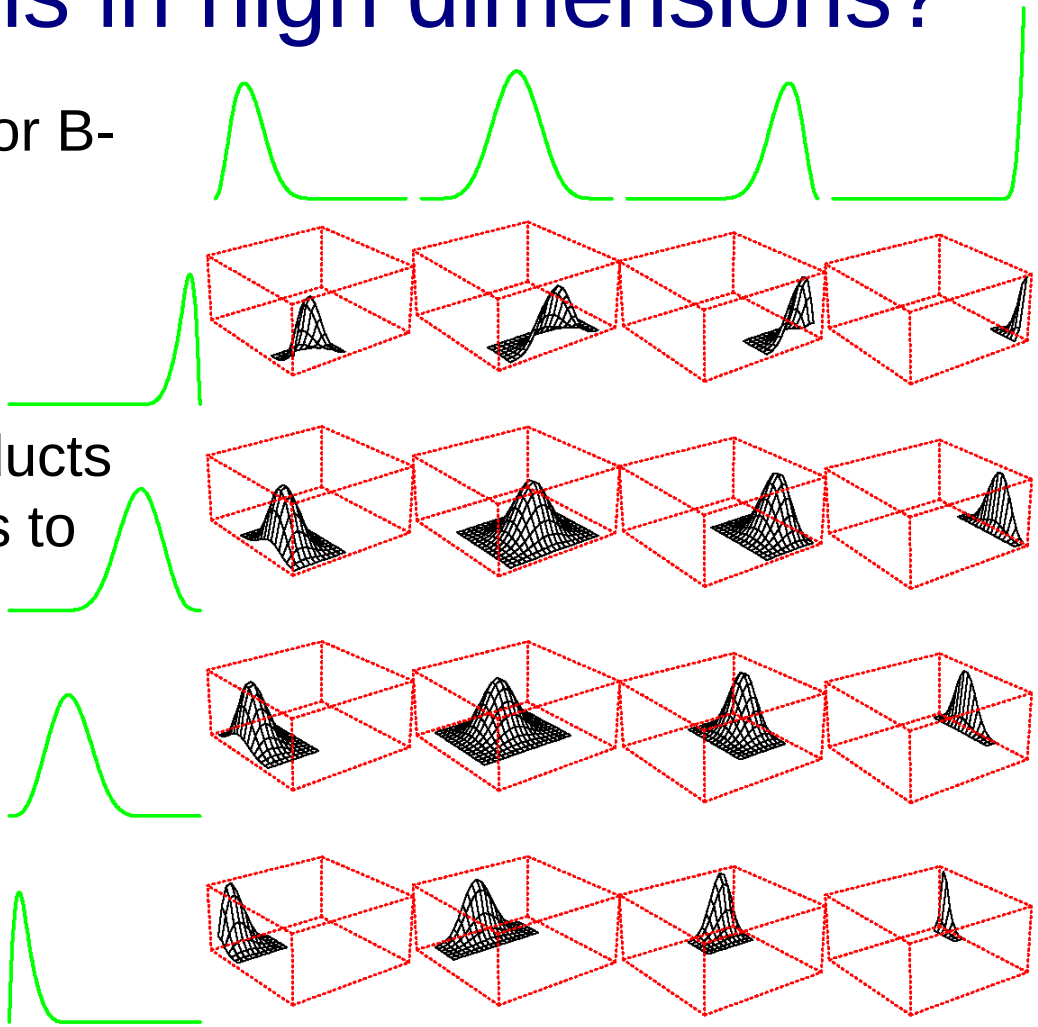
2-D: "tensor B-spline"

- Does this work in 9D?

Basis functions in high dimensions?

- Idea: remember the tensor B-spline trick?

Example: take outer products of four 1D basis functions to "make" sixteen 2D basis functions



2-D: "tensor B-spline"

- Does this work in 9D?

No! 4^9 = 262144 basis functions!

Adaptive basis functions

- Introduce extra parameters that alter the ***form*** of a limited set of basis functions

- Prototypical example:

$$\phi_m(\mathbf{x}) = \begin{cases} 1 & \text{if } m = 1, \\ \sigma\left(\sum_{j=1}^{p} w_{m,j} x_j + w_{m,0}\right) & \text{otherwise,} \end{cases}$$

- All parameters ($\{w_{m,j}\}$ and $\{\beta_m\}$) optimized together during training (stochastic gradient descent)

Adaptive basis functions

- Introduce extra parameters that alter the ***form*** of a limited set of basis functions

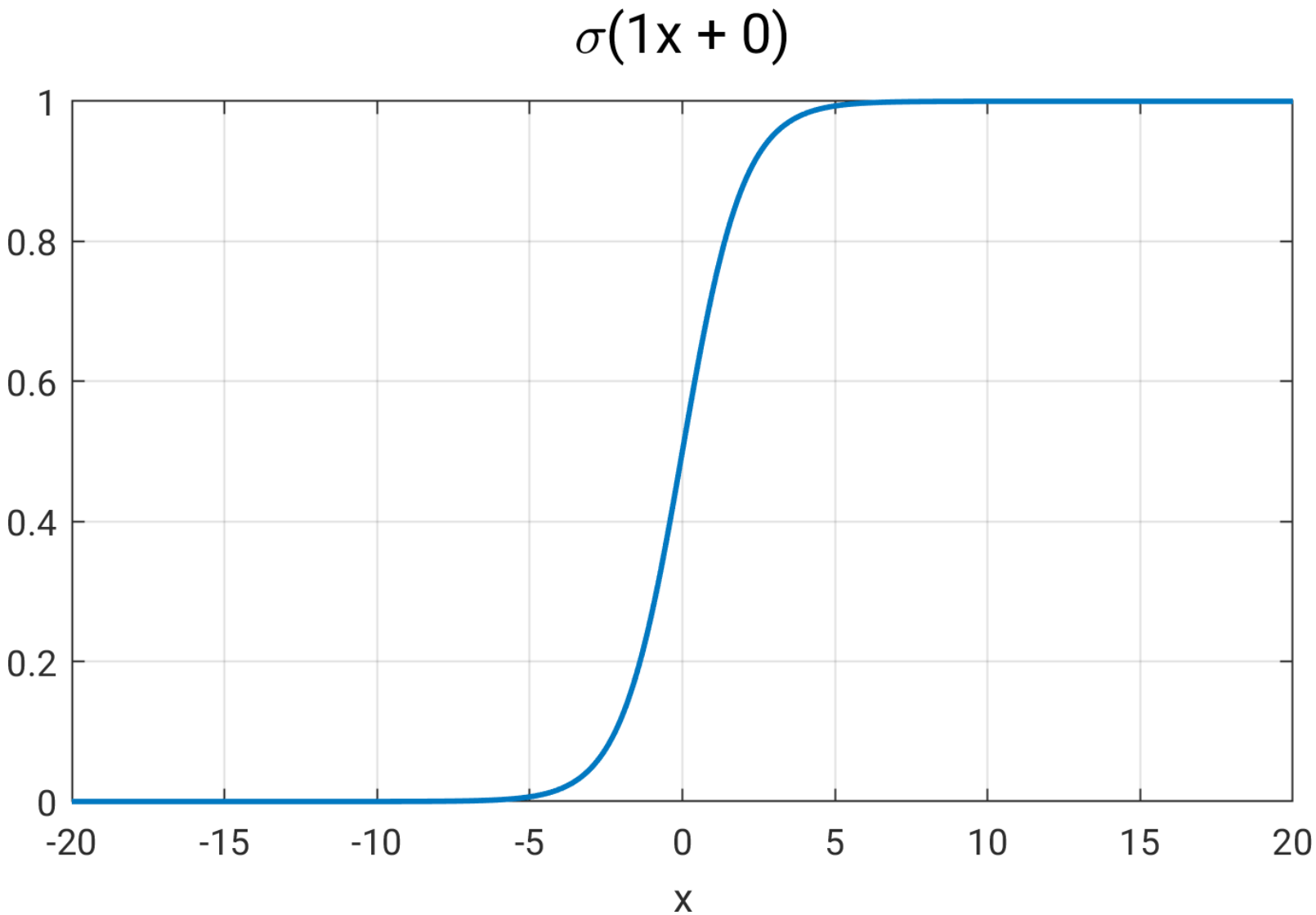
- Prototypical example:

$$\phi_m(\mathbf{x}) = \begin{cases} 1 & \text{if } m = 1, \\ \sigma\left(\sum_{j=1}^{p} w_{m,j} x_j + w_{m,0}\right) & \text{otherwise,} \end{cases}$$

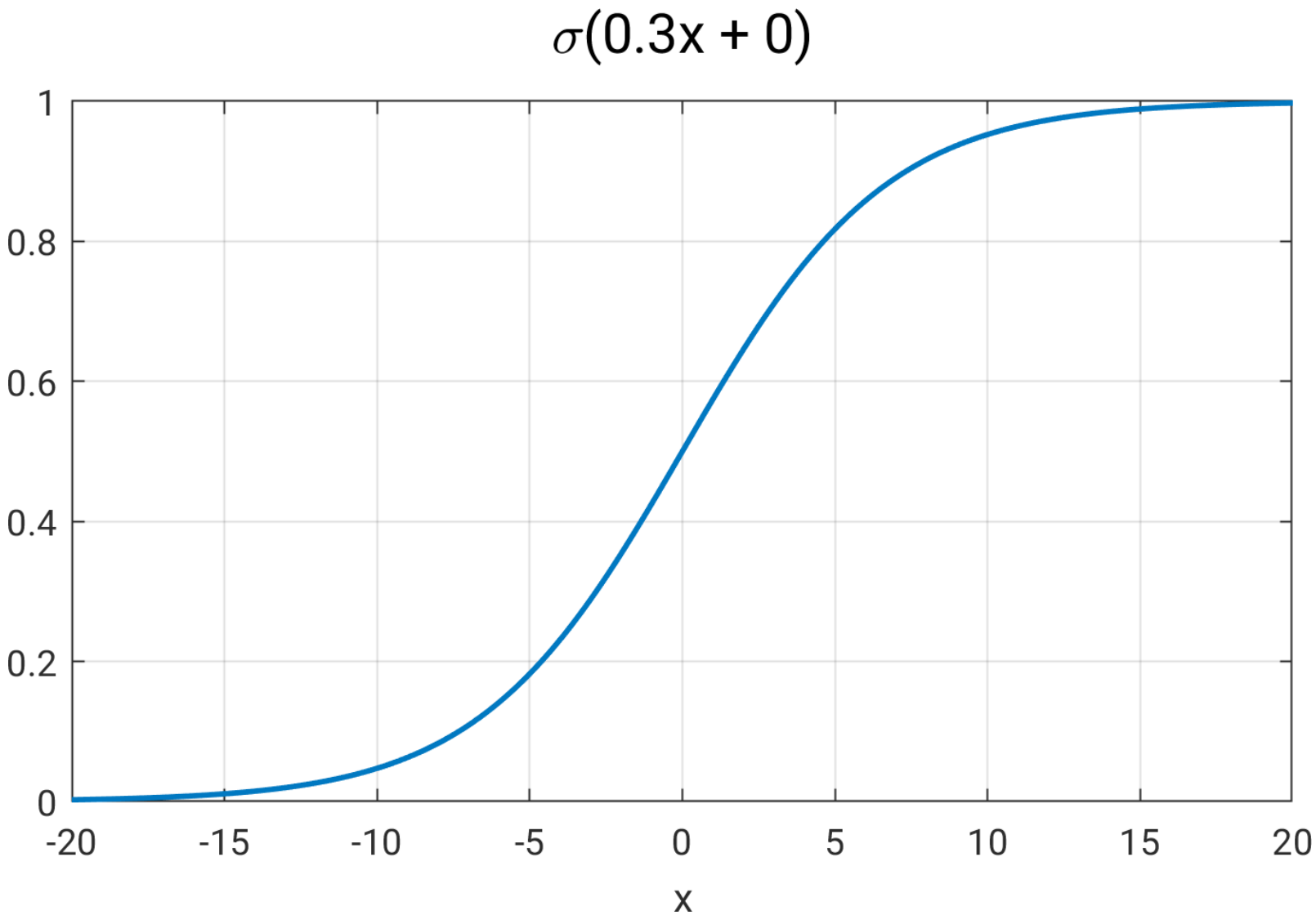
Extra parameters

- All parameters ($\{w_{m,j}\}$ and $\{\beta_m\}$) optimized together during training (stochastic gradient descent)

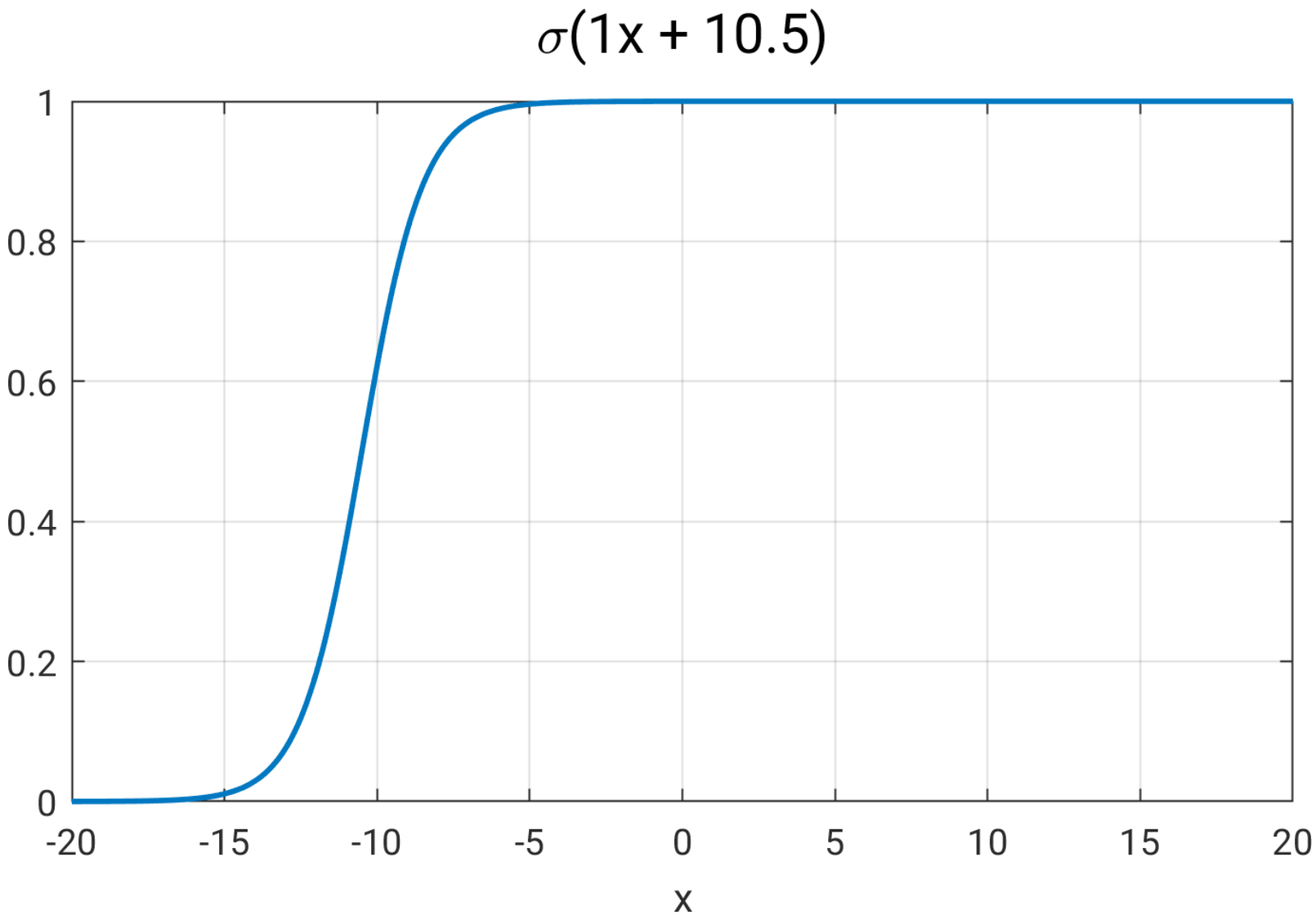
Adaptive basis functions (p=1)



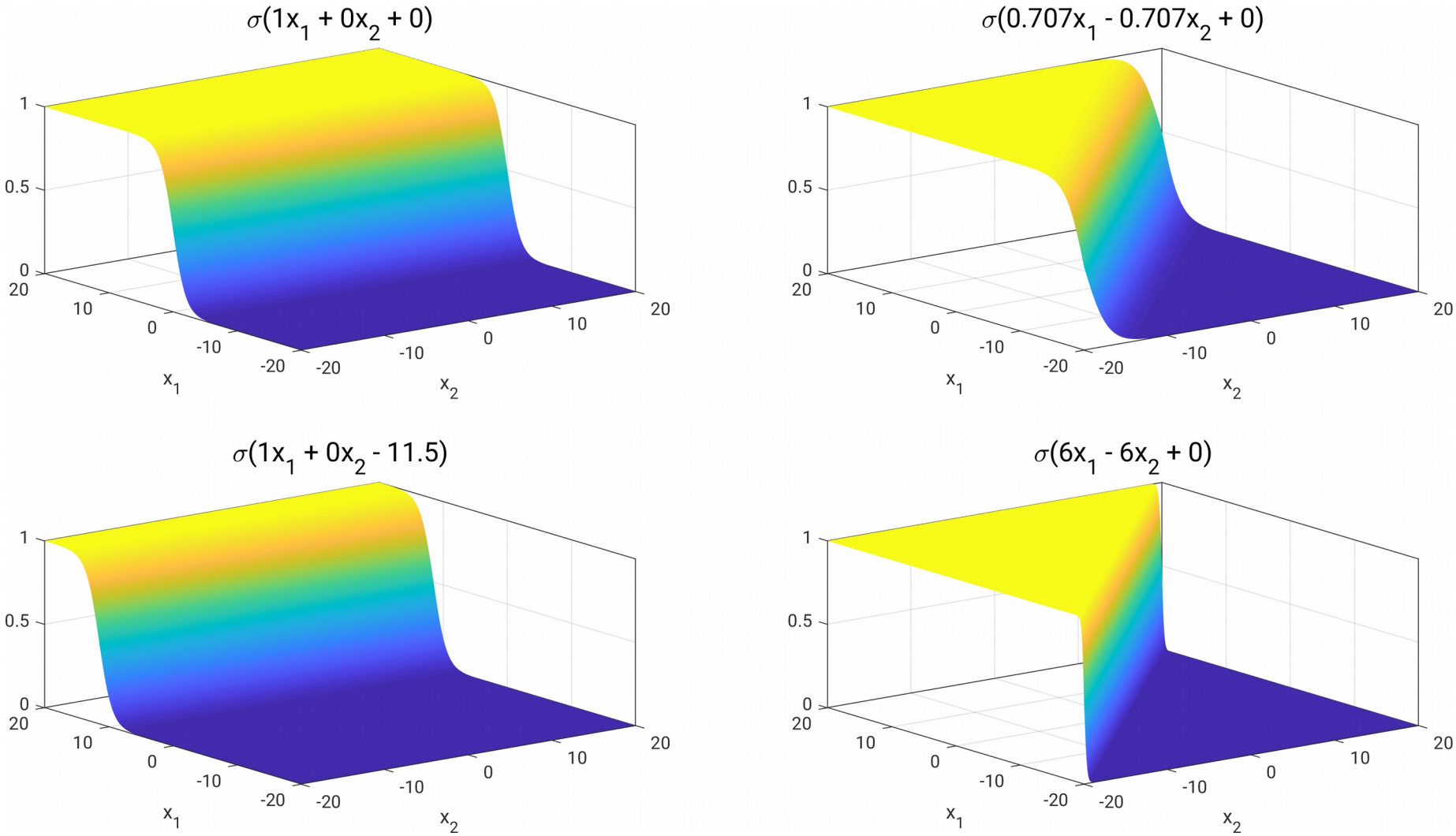
Adaptive basis functions (p=1)



Adaptive basis functions (p=1)



Adaptive basis functions (p=2)



Feed-forward neural network

So the model is $$p(y=1|\mathbf{x},\boldsymbol{\theta}) = \sigma\left(\sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x})\right)$$

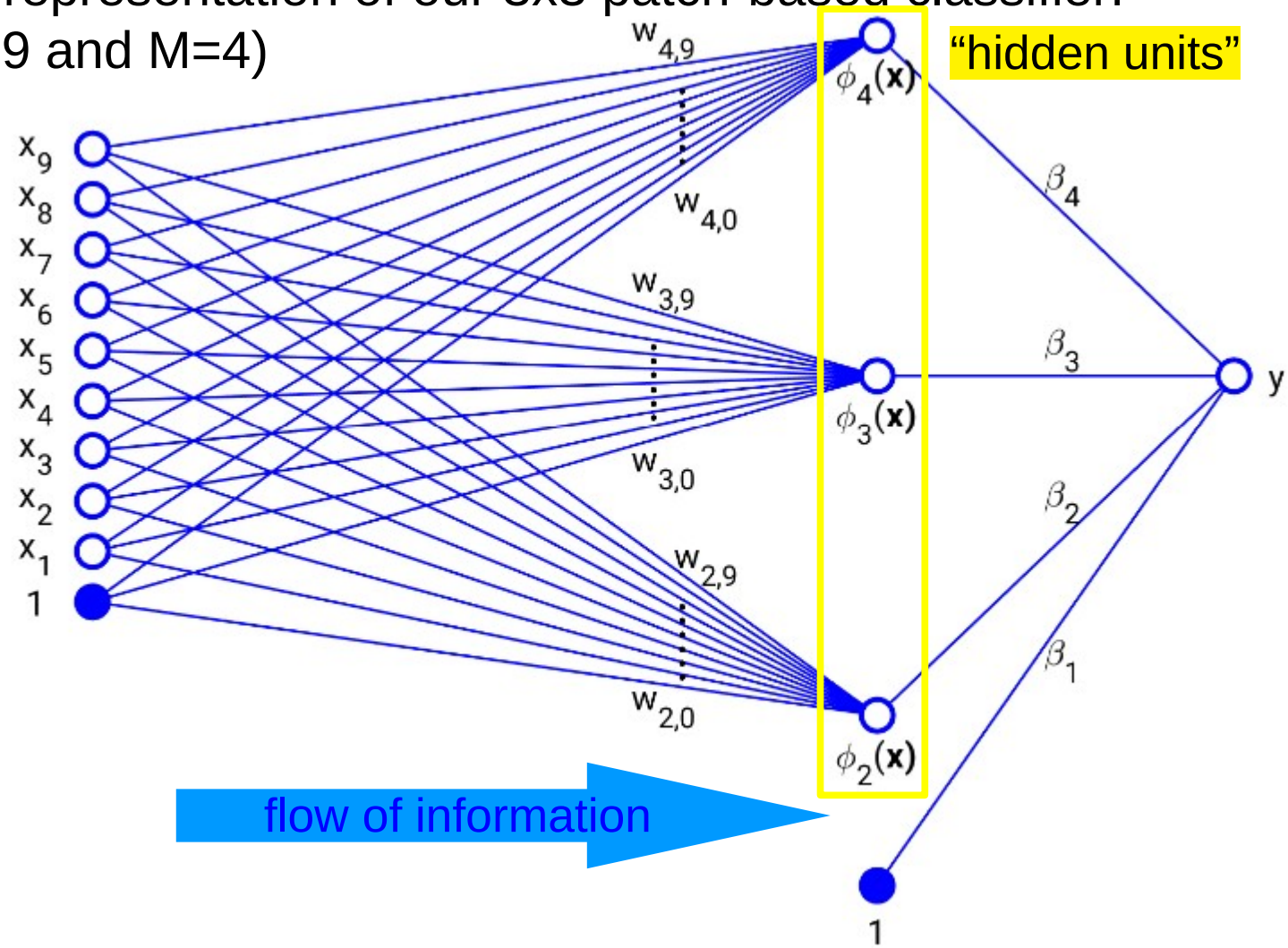
with basis functions

parameters

$$\phi_m(\mathbf{x}) = \begin{cases} 1 & \text{if } m=1, \\ \sigma\left(\sum_{j=1}^{p} w_{m,j} x_j + w_{m,0}\right) & \text{otherwise,} \end{cases}$$

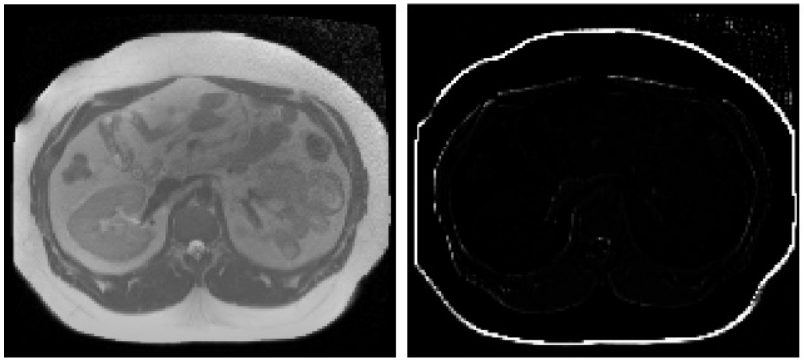
Feed-forward neural network

Graphical representation of our 3x3 patch-based classifier:
(p=9 and M=4)



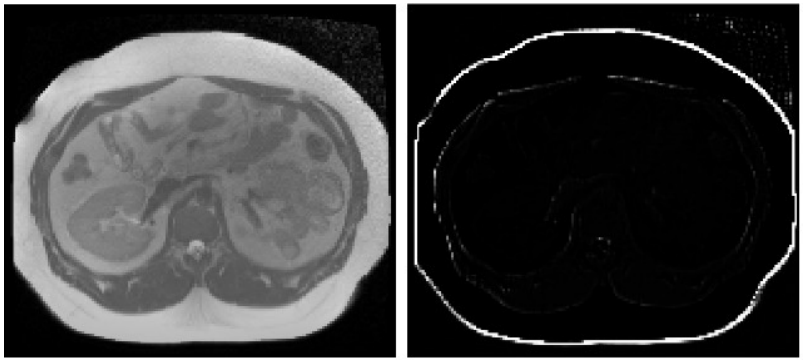
- Can insert more than one "hidden" layer ("deep learning")

Applying the trained classifier on new data:



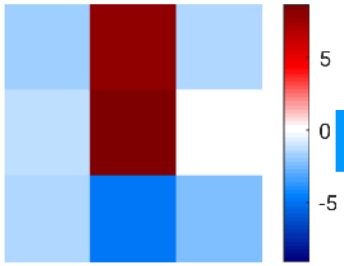
$p(y = 1|\mathbf{x}, \hat{\boldsymbol{\theta}})$

Applying the trained classifier on new data:



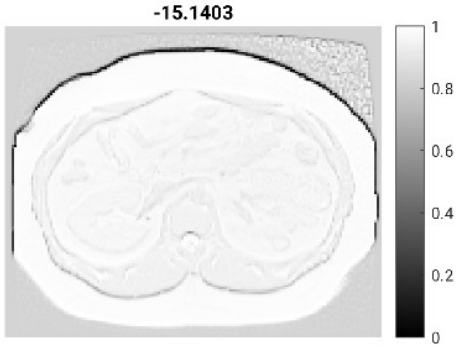
$p(y = 1|\mathbf{x}, \hat{\boldsymbol{\theta}})$

$\{w_{2,1}, \ldots, w_{2,9}\}$

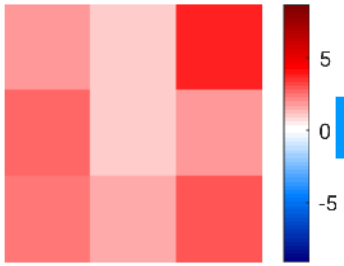


filter

$\phi_2(\mathbf{x})$

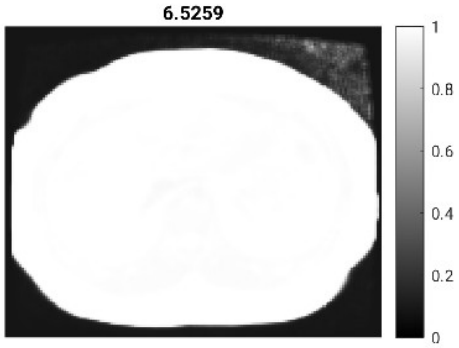


$\{w_{3,1}, \ldots, w_{3,9}\}$

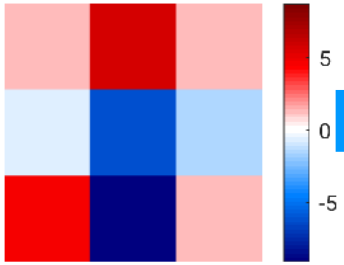


filter

$\phi_3(\mathbf{x})$

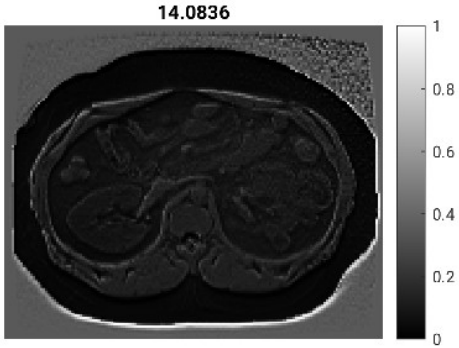


$\{w_{4,1}, \ldots, w_{4,9}\}$

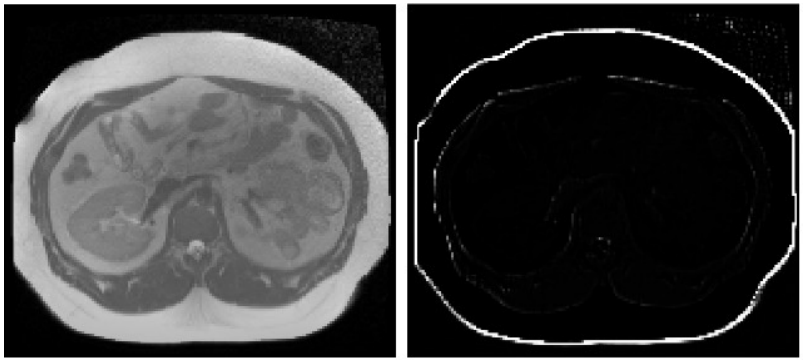


filter

$\phi_4(\mathbf{x})$



Applying the trained classifier on new data:

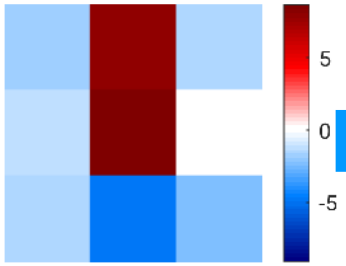


$p(y = 1|\mathbf{x}, \hat{\boldsymbol{\theta}})$

Filtering operations can be implemented using convolutions

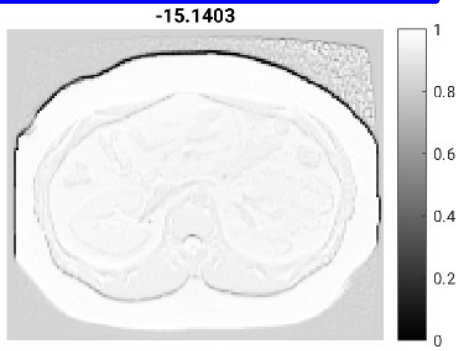
=> “convolutional neural network”

$\{w_{2,1}, \ldots, w_{2,9}\}$

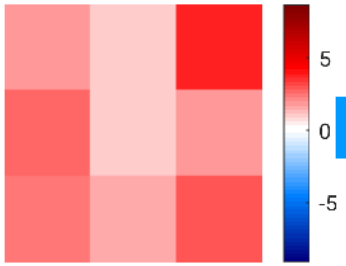


filter

$\phi_2(\mathbf{x})$

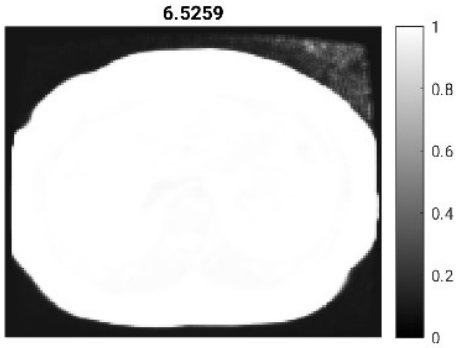


$\{w_{3,1}, \ldots, w_{3,9}\}$

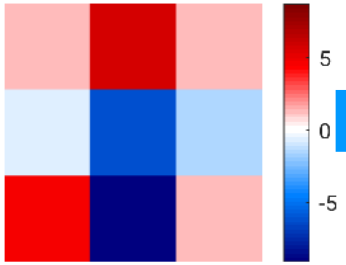


filter

$\phi_3(\mathbf{x})$

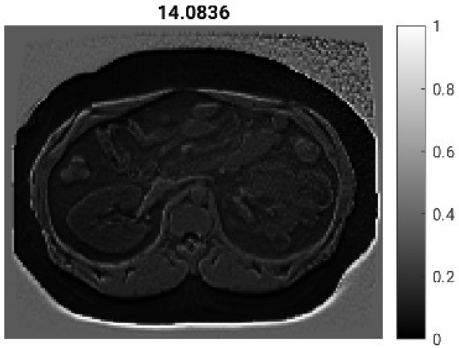


$\{w_{4,1}, \ldots, w_{4,9}\}$



filter

$\phi_4(\mathbf{x})$



Neural networks = ultimate solution?

<u>No model, only training data:</u>

- No domain expertise needed
- Very easy to train and deploy
- Super fast (GPUs)

- Training data often very hard to get in medical imaging!
- Scanning hardware/software/protocol changes routinely!