COMPUTER SCIENCE
INDIANA UNIVERSITY
School of Informatics and Computing
Bloomington

Nonlinear representations

Reminders

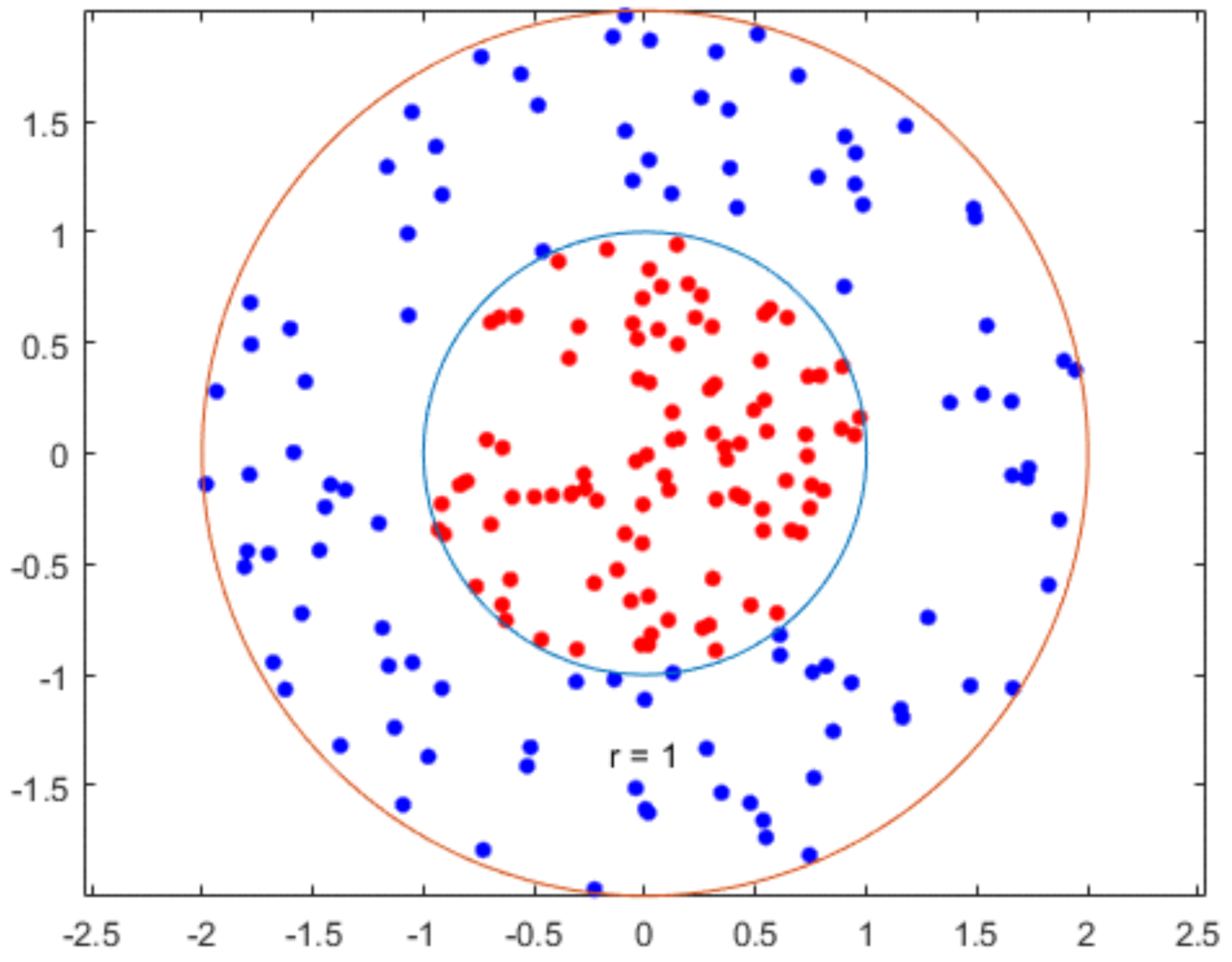
- Quiz 1 should be marked by Friday
- Assignment 2 marks released
- Hopefully you have started Assignment 3
  - any issues?
  - you should be able to get about 25% for all your methods on SUSY
  - naive Bayes might be bit worse (say about 26%)
  - for kernel question, accuracy should be about 22%

Representations for learning nonlinear functions

- Generalized linear models enabled many p(y | x) distributions
  - Still however learning a simple function for E[y | x], i.e., f(<x,w>)
- Approach we discussed earlier: augment current features x using polynomials
- There are many strategies to augmenting x
  - fixed representations, like polynomials, wavelets
  - learned representations, like neural networks and matrix factorization

What if classes are not linearly separable?

$$x_1^2 + x_2^2 = 1$$



How to learn f(x) such that f(x) > 0 predicts + and f(x) < 0 predicts negative?

Nonlinear transformation

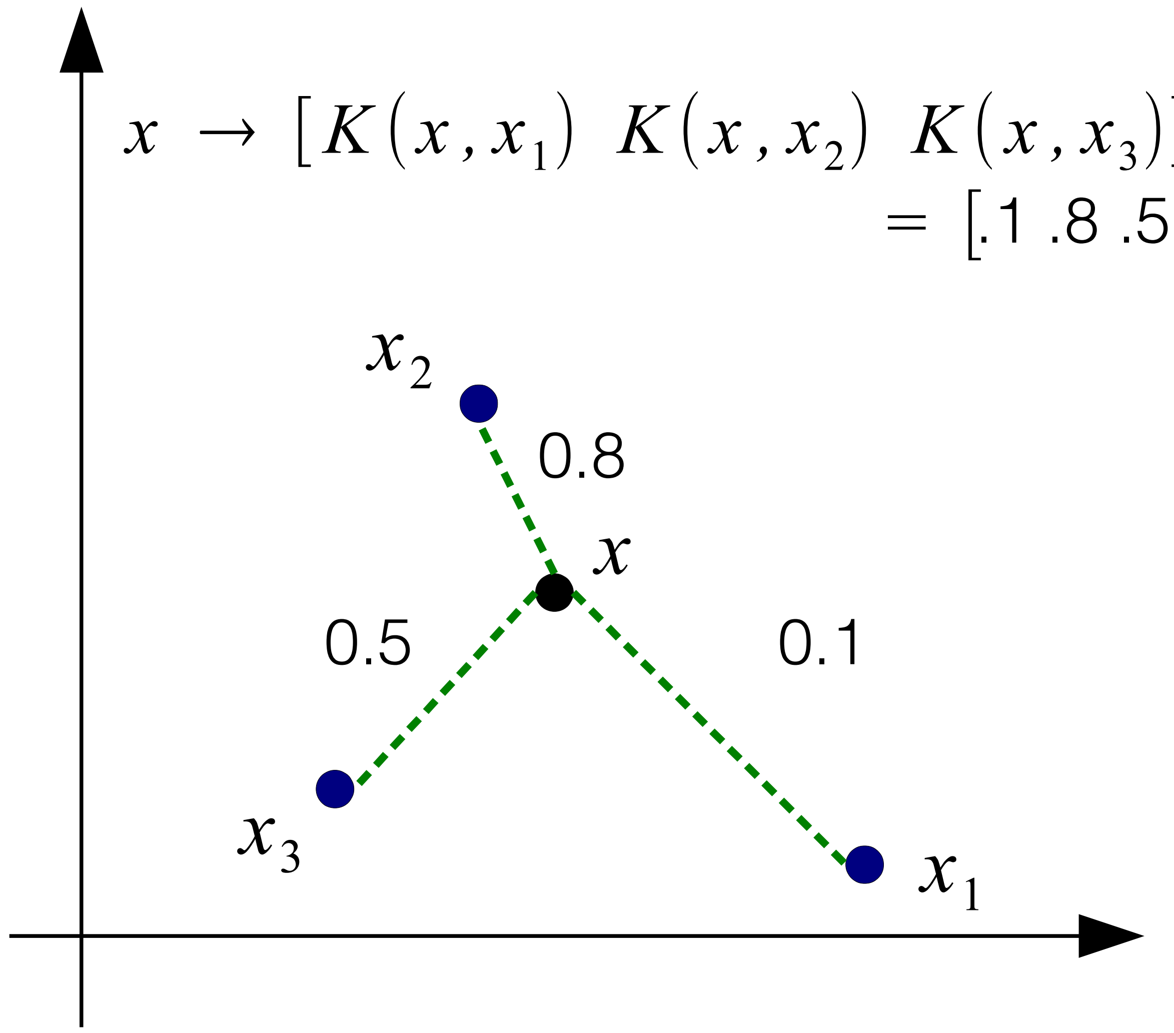
$$\mathbf{x} \to \boldsymbol{\phi}(\mathbf{x}) = \left(\begin{array}{c} \phi_1(\mathbf{x}) \\ \dots \\ \phi_p(\mathbf{x}) \end{array} \right)$$

$$\text{e.g., } \mathbf{x} = [x_1, x_2], \quad \boldsymbol{\phi}(\mathbf{x}) = \left(\begin{array}{c} x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ x_2^3 \end{array} \right)$$

Gaussian kernel / Gaussian radial basis function

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|_2^2}{\sigma^2}\right) \qquad f(\mathbf{x}) = \sum_{i=1}^{p} w_i k(\mathbf{x}, \mathbf{x}_i)$$

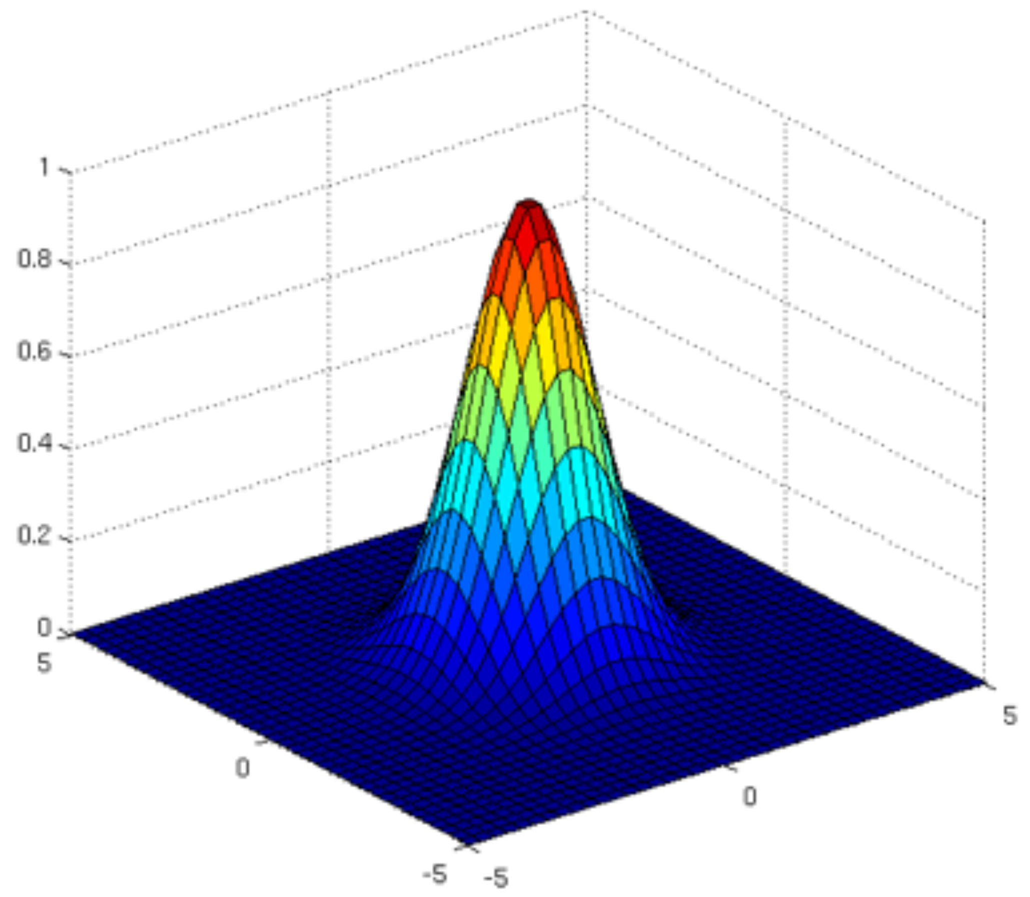
$$\mathbf{x} = \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] \rightarrow \left[\begin{array}{c} k(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ k(\mathbf{x}, \mathbf{x}_p) \end{array}\right]$$



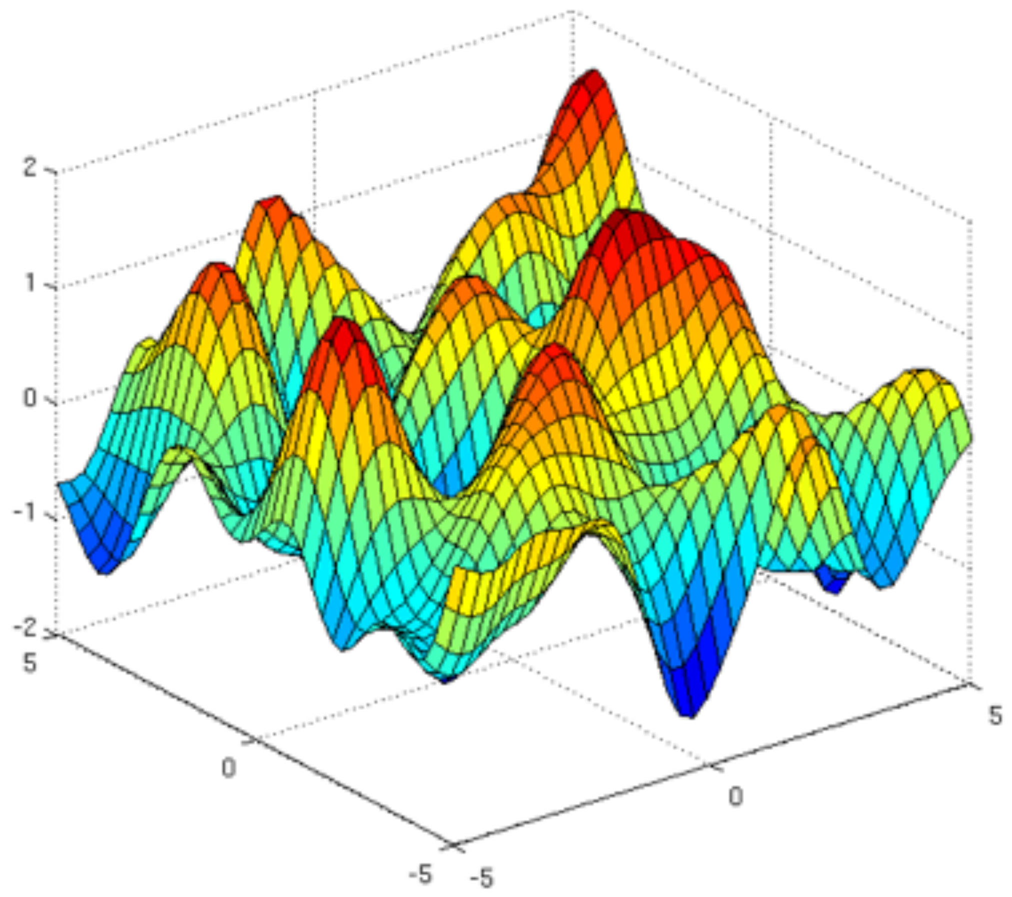
Gaussian kernel / Gaussian radial basis function

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|_2^2}{\sigma^2} \right) \qquad f(\mathbf{x}) = \sum_{i=1}^{p} w_i k(\mathbf{x}, \mathbf{x}_i)$$

Kernel



Possible function f with several centers



Selecting centers

- Many different strategies to decide on centers
  - many ML algorithms use kernels e.g., SVMs, Gaussian process regression
- For kernel representations, typical strategy is to select training data as centers
- Clustering techniques to find centers
- A grid of values to best (exhaustively) cover the space
- Many other strategies, e.g., using information gain, coherence criterion, informative vector machine

Exercise

- What would it mean to use an l1 regularizer with a kernel representation?
  - Recall that l1 prefers to zero elements in w

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{p} k(\mathbf{x},\mathbf{z}_j)\mathbf{w}_j - y_i\right)^2 + \lambda\|\mathbf{w}\|_1$$

Distinction with the kernel trick

- We have discussed using kernels to provide similarity features radial basis functions

- In some cases, they are used to compute inner products efficiently, for a nonlinear representation

Regression with new features

$$\min_{\mathbf{w}} \sum_{i=1}^{n} (\boldsymbol{\phi}(\mathbf{x}_i)^\top \mathbf{w} - y_i)^2 = \min_{\mathbf{w}} \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{p} \boldsymbol{\phi}_j(\mathbf{x}_i)\mathbf{w}_j \right) - y_i \right)^2$$

What if p is really big?

$$\min_{\mathbf{w}} \sum_{i=1}^{n} (\boldsymbol{\phi}(\mathbf{x}_i)^\top \mathbf{w} - y_i)^2 = \min_{\mathbf{a}} \sum_{i=1}^{n} \left(\left(\sum_{j=1}^{n} \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle \mathbf{a}_j \right) - y_i \right)^2$$

If can compute dot product efficiently, then can solve this regression problem efficiently

Example: polynomial kernel

$$\phi(\mathbf{x}) = \begin{bmatrix} \mathbf{x}_1^2 \\ \sqrt{2}\mathbf{x}_1\mathbf{x}_2 \\ \mathbf{x}_2^2 \end{bmatrix}$$

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle = \langle \mathbf{x}, \mathbf{x}' \rangle^2$$

In general, for order d polynomials, $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^d$

Gaussian kernel

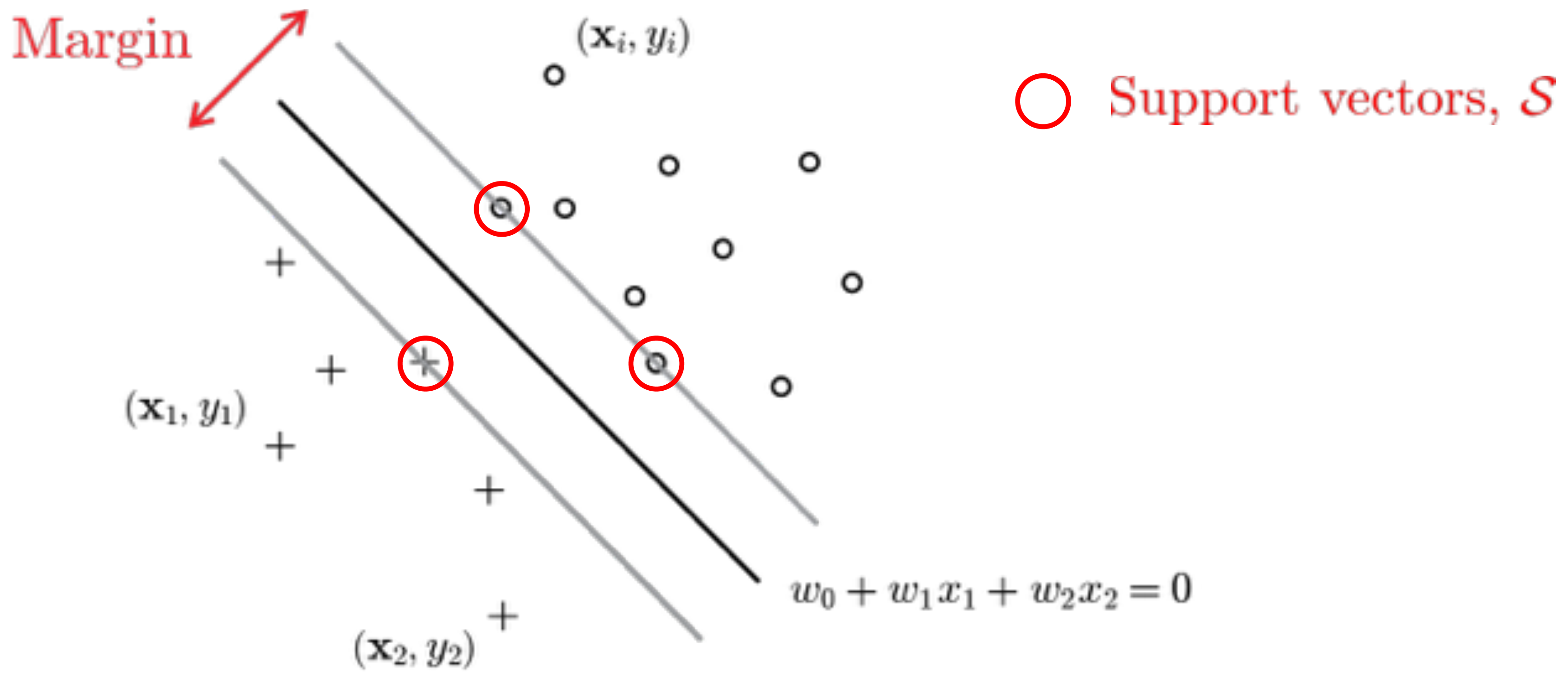
Infinite polynomial representation

$$\phi(x) = \exp(-\gamma x^2) \begin{bmatrix} 1 \\ \sqrt{\frac{2\gamma}{1!}}x \\ \sqrt{\frac{(2\gamma)^2}{2!}}x^2 \\ \vdots \end{bmatrix}$$

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

Another setting using the kernel trick: SVMs

- We formulated the linear classification problem for two classes with the additional constraint to maximize the margin



Dual Problem

$$\boxed{\sum_{i=1}^{n} \alpha_i y_i = 0 \qquad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i}$$

$$\begin{aligned}
L(\mathbf{w}, w_0, \boldsymbol{\alpha}) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{w}^T \mathbf{x}_i - \sum_{i=1}^{n} \alpha_i y_i w_0 + \sum_{i=1}^{n} \alpha_i \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^{n} \alpha_i y_i \Big(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j\Big)^T \mathbf{x}_i + \sum_{i=1}^{n} \alpha_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)
\end{aligned}$$

kernel property

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

Subject to:

$$\alpha_i \geq 0 \qquad \forall i \in \{1, 2, \ldots, n\}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

Prediction also only needs inner product

Use quadratic programming to solve for $\boldsymbol{\alpha}$

Then set

$$\Longrightarrow \qquad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

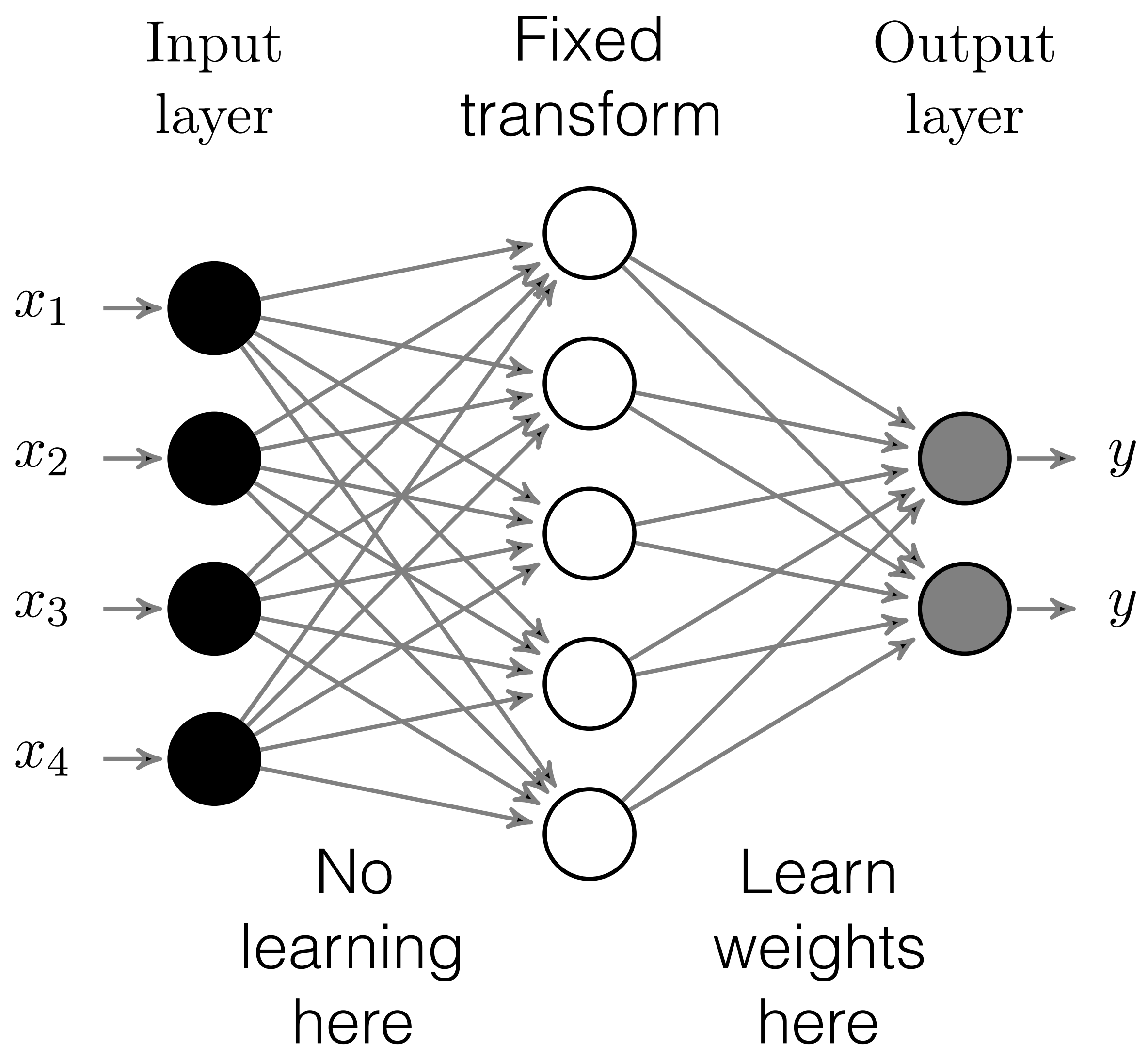
$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

$$\Longrightarrow \qquad \begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + w_0 \\ &= \frac{1}{2} \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + w_0 \end{aligned}$$

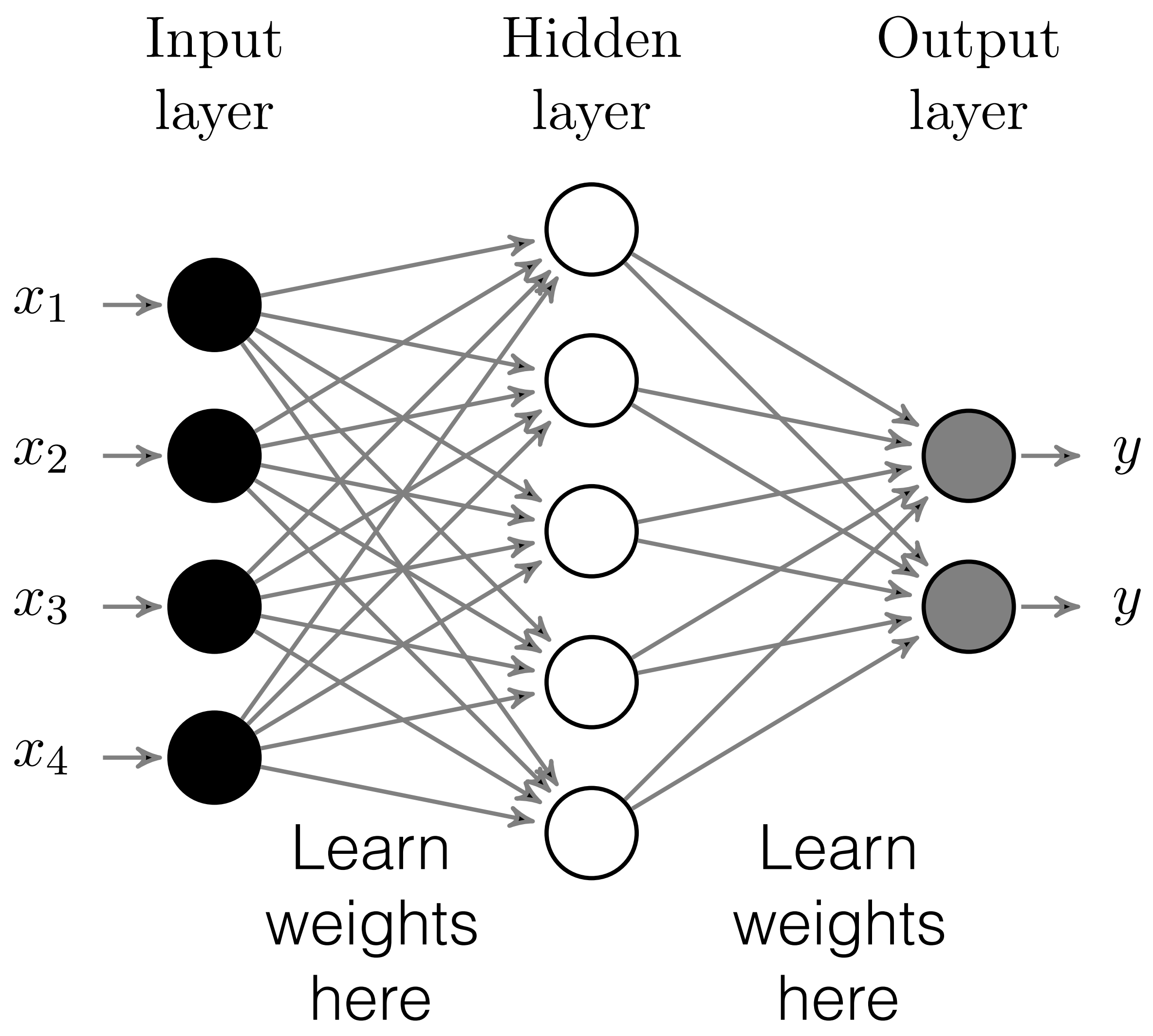
What about learning the representation?

- We have talked about fixed nonlinear transformations
  - polynomials
  - kernels
- Neural networks are one way to **learn** this transformation

GLM with fixed representation vs. neural network

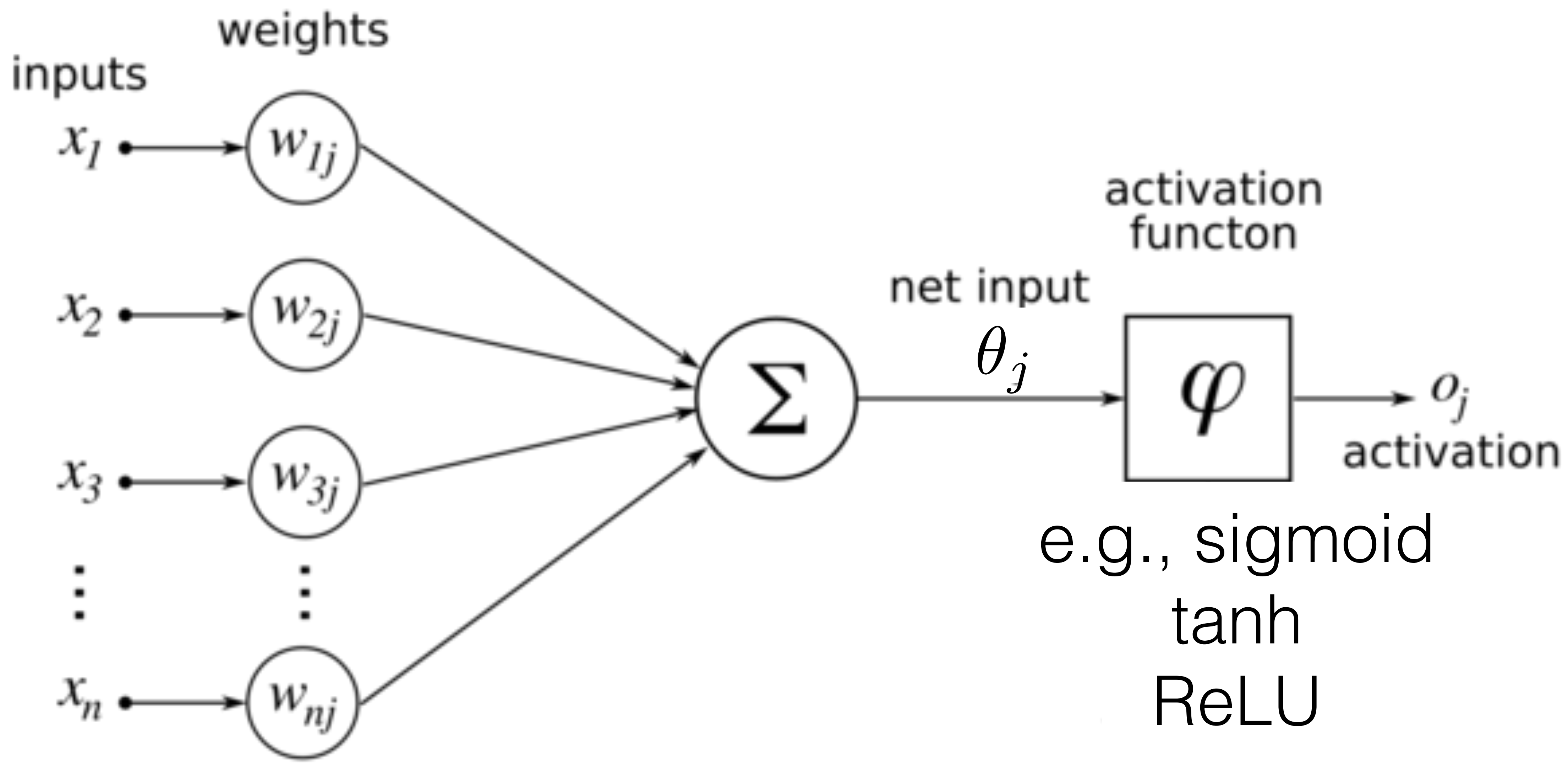


GLM with
augmented fix representation



Two-layer neural network

Explicit steps in visualizations



e.g., sigmoid
tanh
ReLU

Example: logistic regression versus neural network

- Both try to predict p(y = 1 | x)

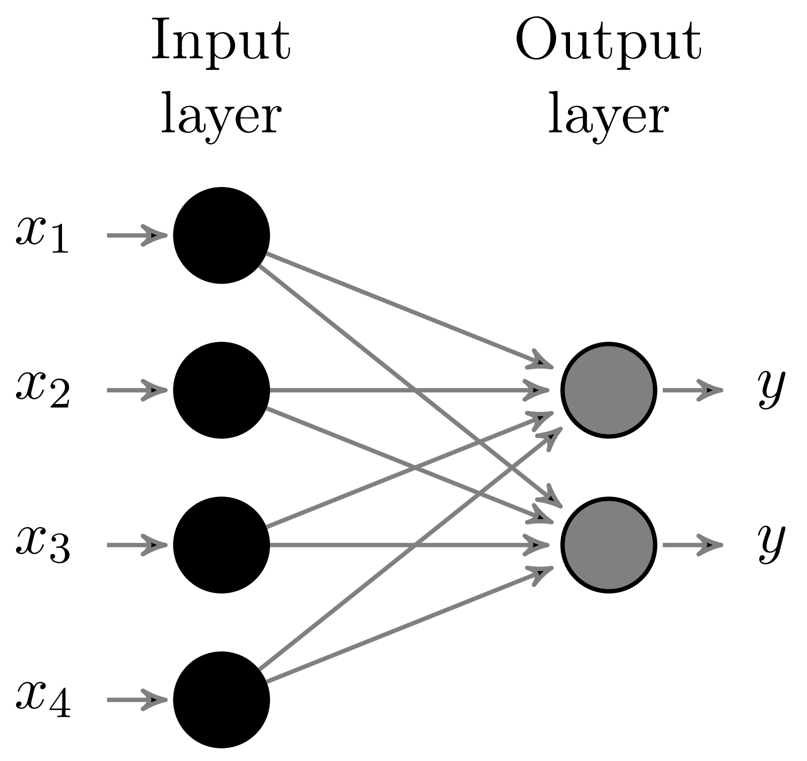
- Logistic regression learns W such that

$$f(\mathbf{xW}) = \sigma(\mathbf{xW}) = p(y = 1|\mathbf{x})$$

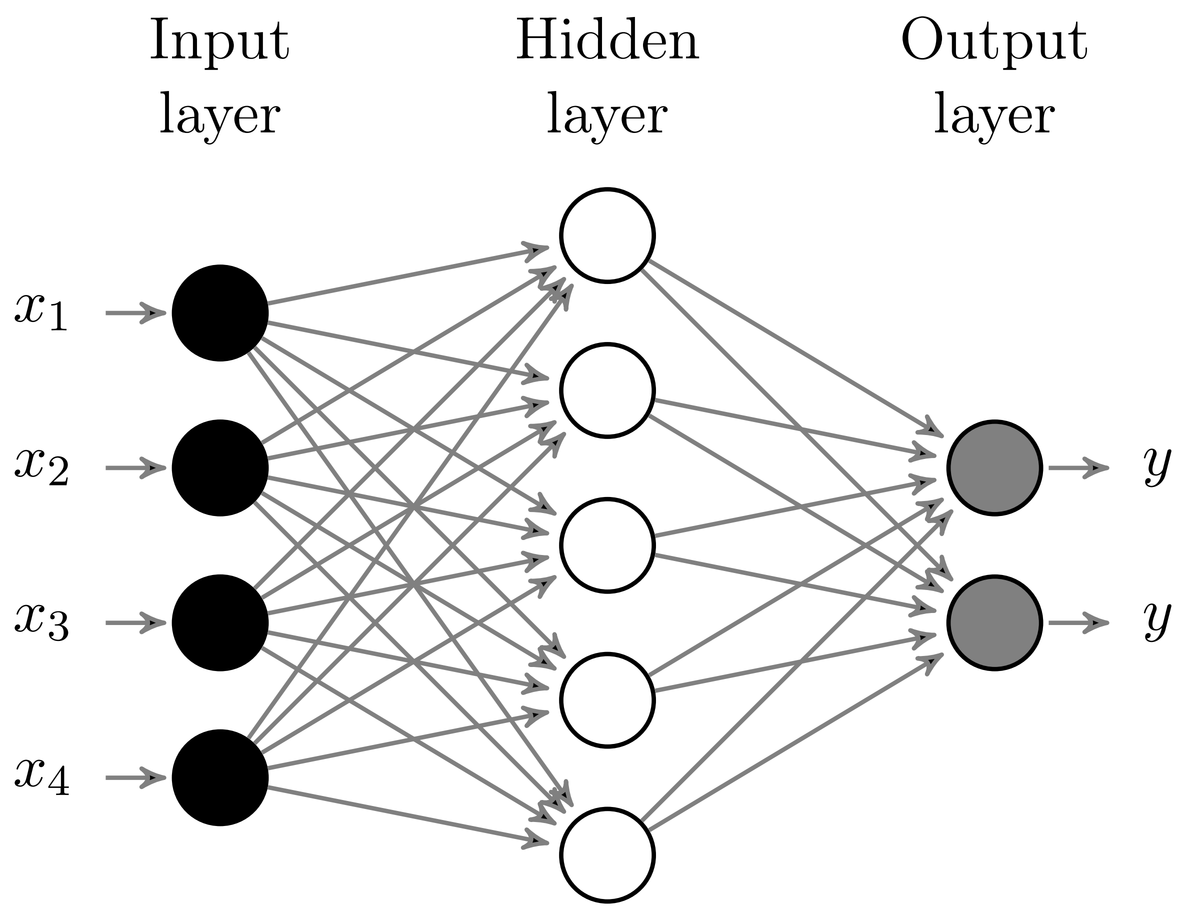
- Neural network learns W1 and W2 such that

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{hW}^{(1)}) = \sigma(\sigma(\mathbf{xW}^{(2)})\mathbf{W}^{(1)}).$$

Generalized linear model vs. neural network



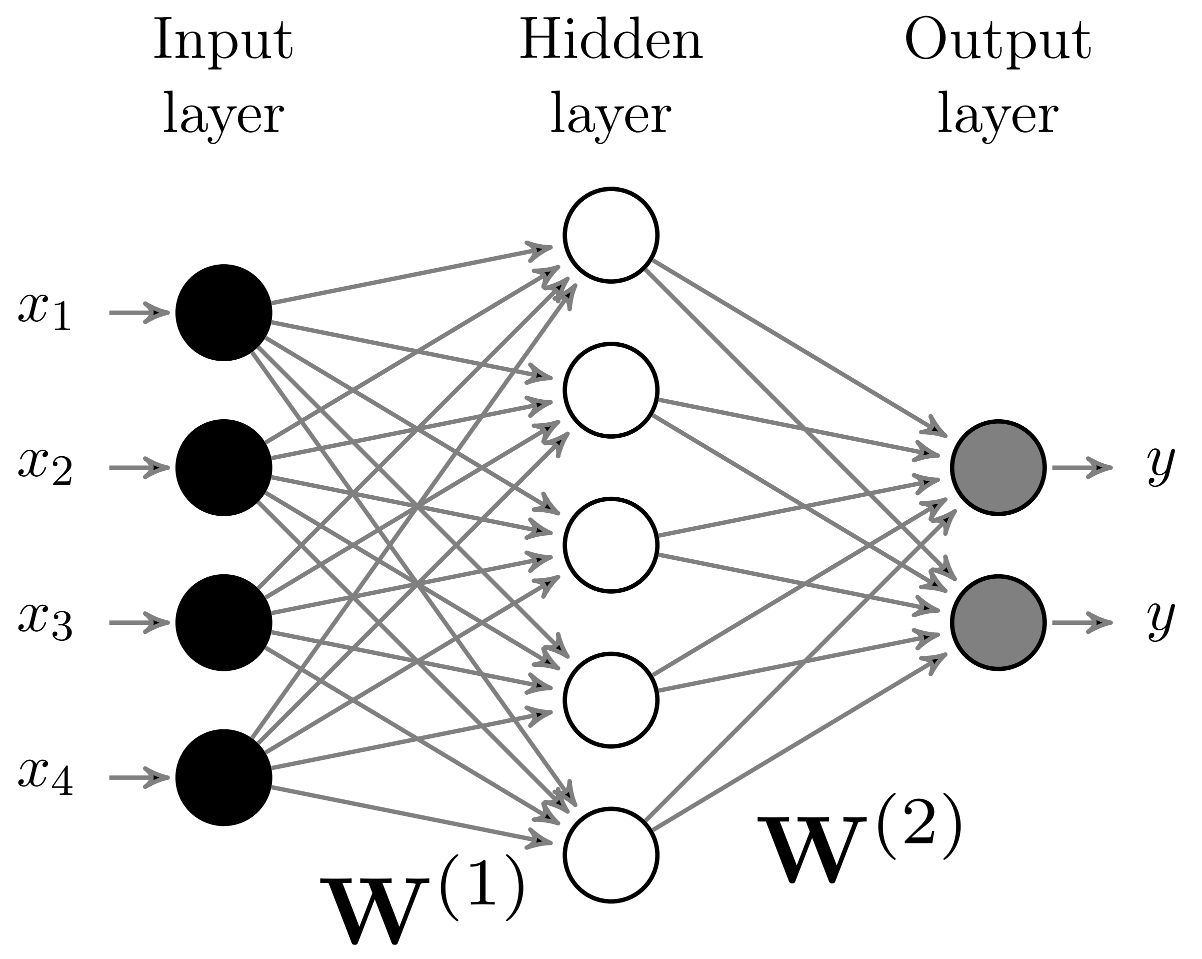
GLM
(e.g. logistic regression)



Two-layer neural network

What are the representational capabilities of neural nets?

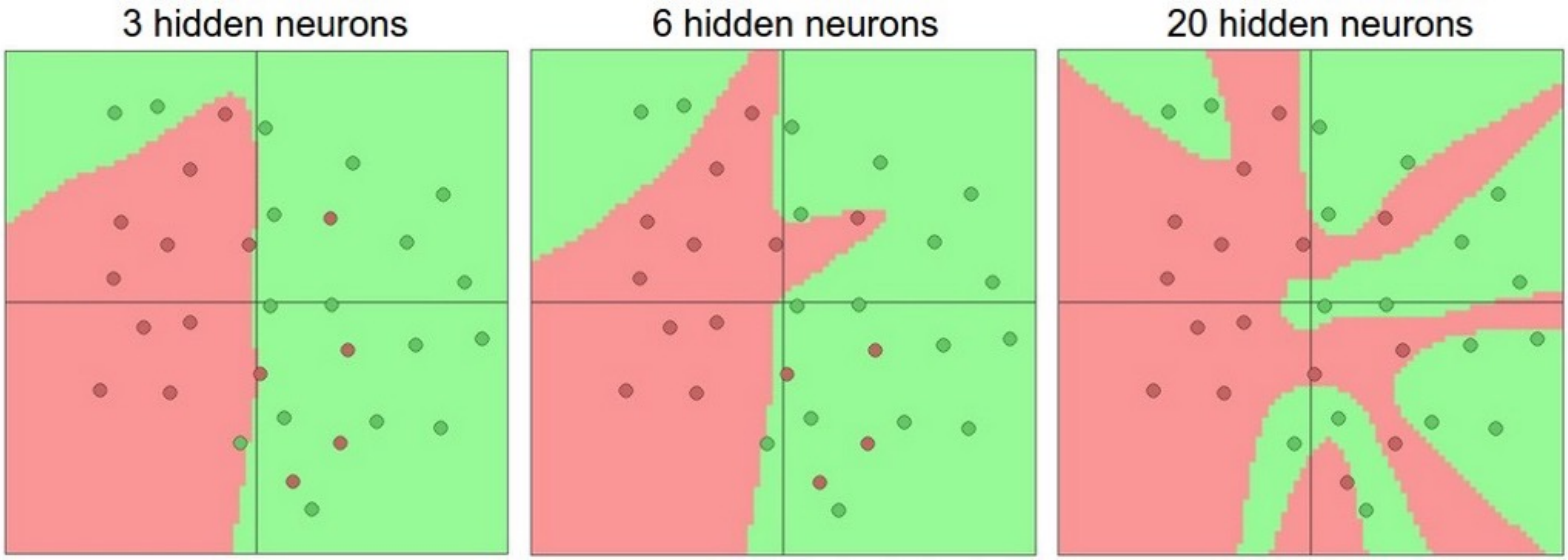
- Single hidden-layer neural networks with sigmoid transfer can represent any continuous function on a bounded space within epsilon accuracy, for a large enough number of hidden nodes
    - see Cybenko, 1989: “Approximation by Superpositions of a Sigmoidal Function”



$$\mathbf{h} = f_1(\mathbf{x}\mathbf{W}^{(1)})$$

$$\hat{y} = f_2(\mathbf{h}\mathbf{W}^{(2)})$$

Nonlinear decision surface



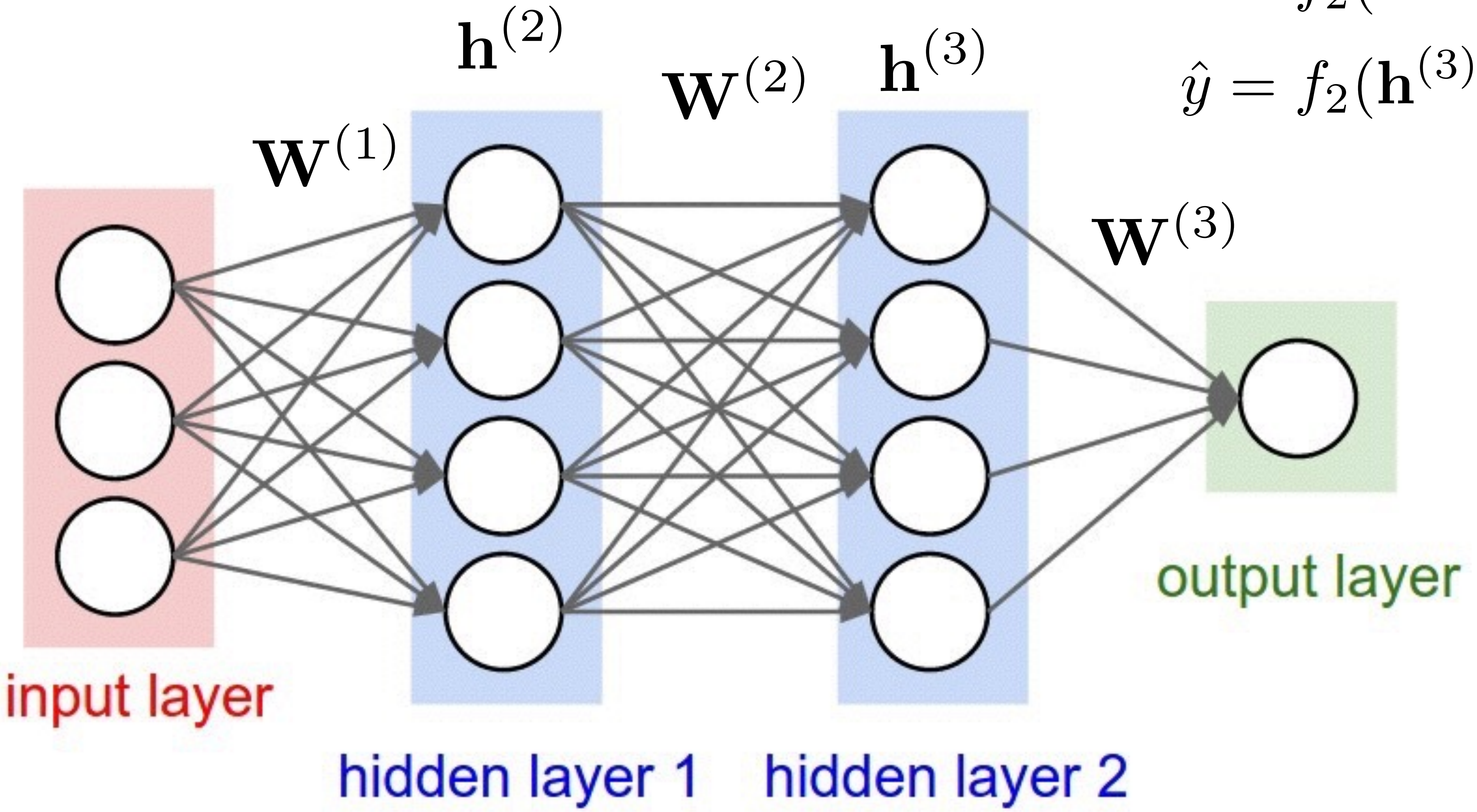
* from http://cs231n.github.io/neural-networks-1/; see that page for a nice discussion on neural nets

Thought question

- "It can be proved that with a sufficiently large number of radial bias functions we can accurately approximate any functions." This statement sounds really strong, but leads me to the (possibly naive or arrogant) question of why don't we just use these for everything? If stacking multiple linear models is better than a single one, why use a single one? (side note: I'm sure that there are lots of reasons, but I really want to know what the weaknesses are).
  - The theoretical result assumes a huge hidden layer
  - In practice, we need to pick a more compact layer

Multi-layer neural network

$$\mathbf{h}^{(2)} = f_1(\mathbf{x}\mathbf{W}^{(1)})$$
$$\mathbf{h}^{(3)} = f_2(\mathbf{h}^{(2)}\mathbf{W}^{(2)})$$
$$\hat{y} = f_2(\mathbf{h}^{(3)}\mathbf{W}^{(3)})$$



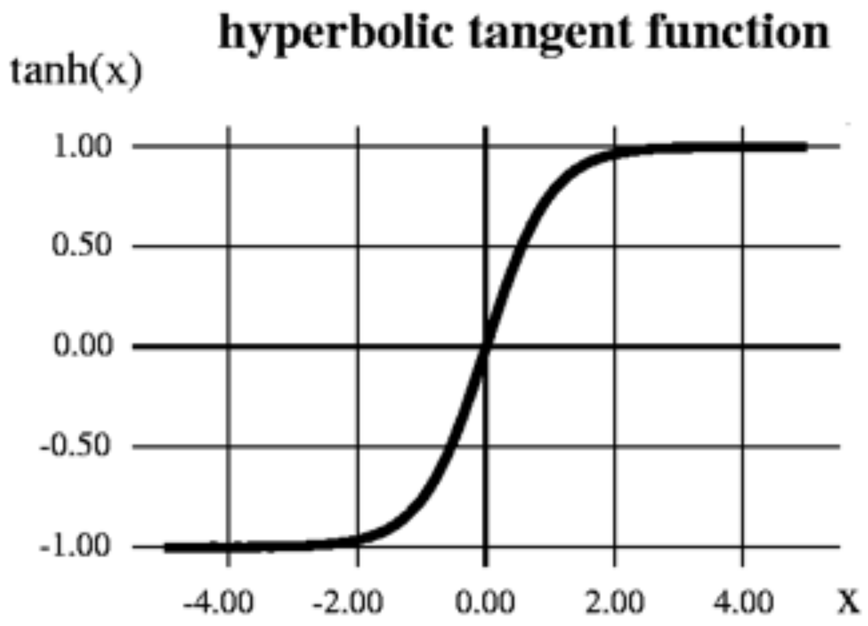
* from http://cs231n.github.io/neural-networks-1/; see that page for a nice discussion on neural nets

How do we learn the parameters to the neural network?

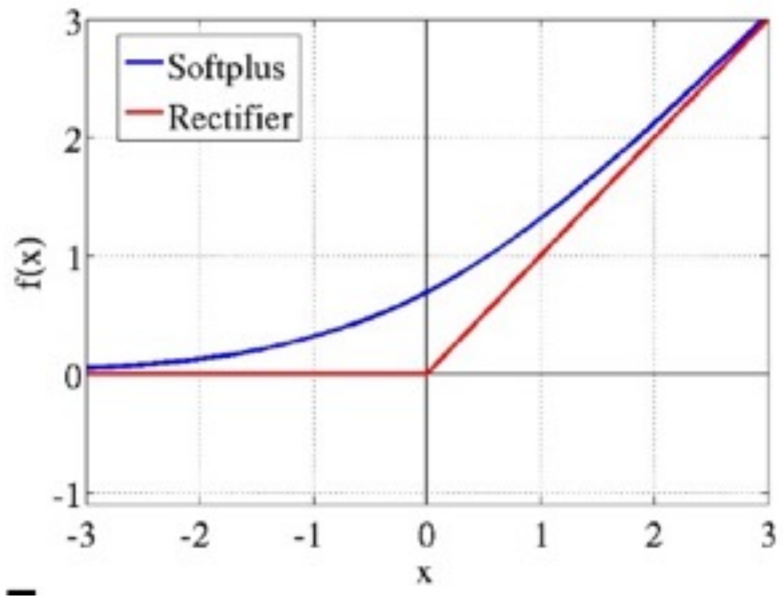
- In linear regression and logistic regression, learned parameters by specifying an objective and minimizing using gradient descent
- We do the exact same thing with neural networks; the only difference is that our function class is more complex
- Need to derive a gradient descent update for W1 and W2

Tanh and rectified linear

- Two more popular transfers are tanh and rectified linear
- Tanh is balanced around 0, which seems to help learning
  - usually preferred to sigmoid



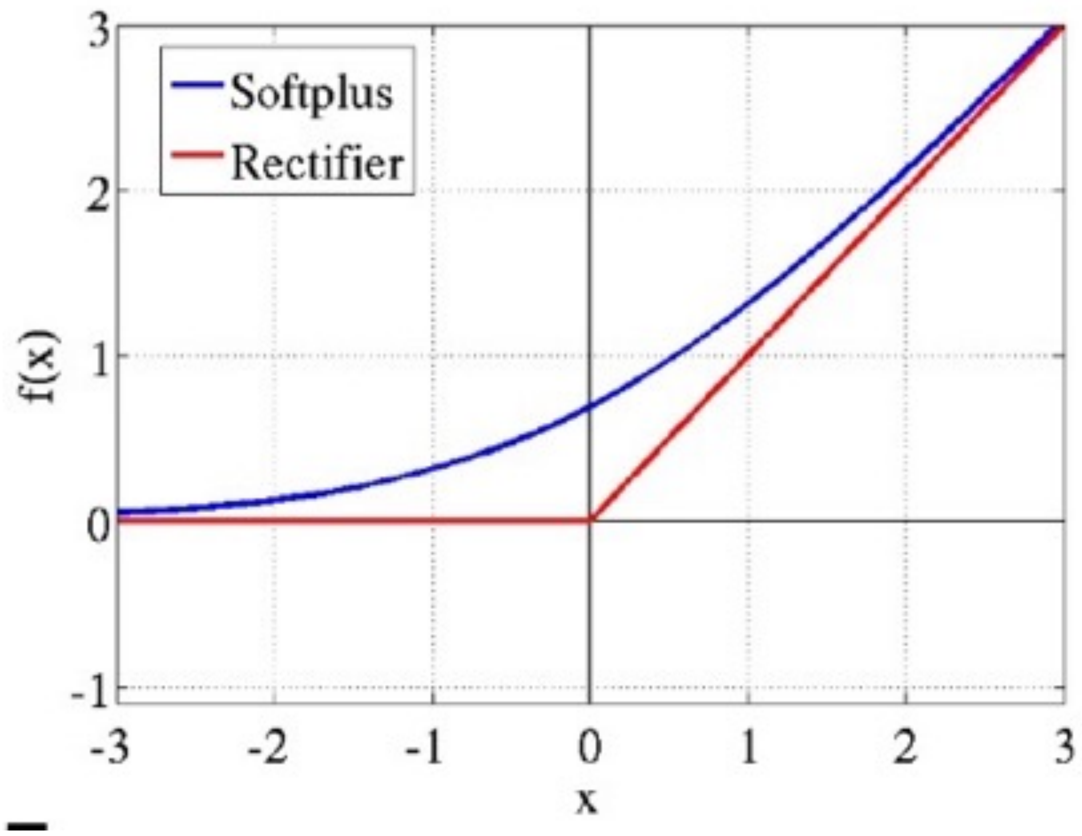
- Rectified linear



- Binary threshold function (perceptron): less used, some notes for this approach: http://www.cs.indiana.edu/~predrag/classes/2015springb555/9.pdf

Rectified linear unit (ReLU)

- Rectified(x) = max(0, x)
  - Non-differentiable point at 0
  - Commonly gradient is 0 for x <= 0, else 1
- Softplus(x) = ln(1+e^{x})
- Recall our variable is sum w_i x_i



- Common strategy: still use sigmoid (or tanh) with cross-entropy in the last output layer, and use rectified linear units in the interior