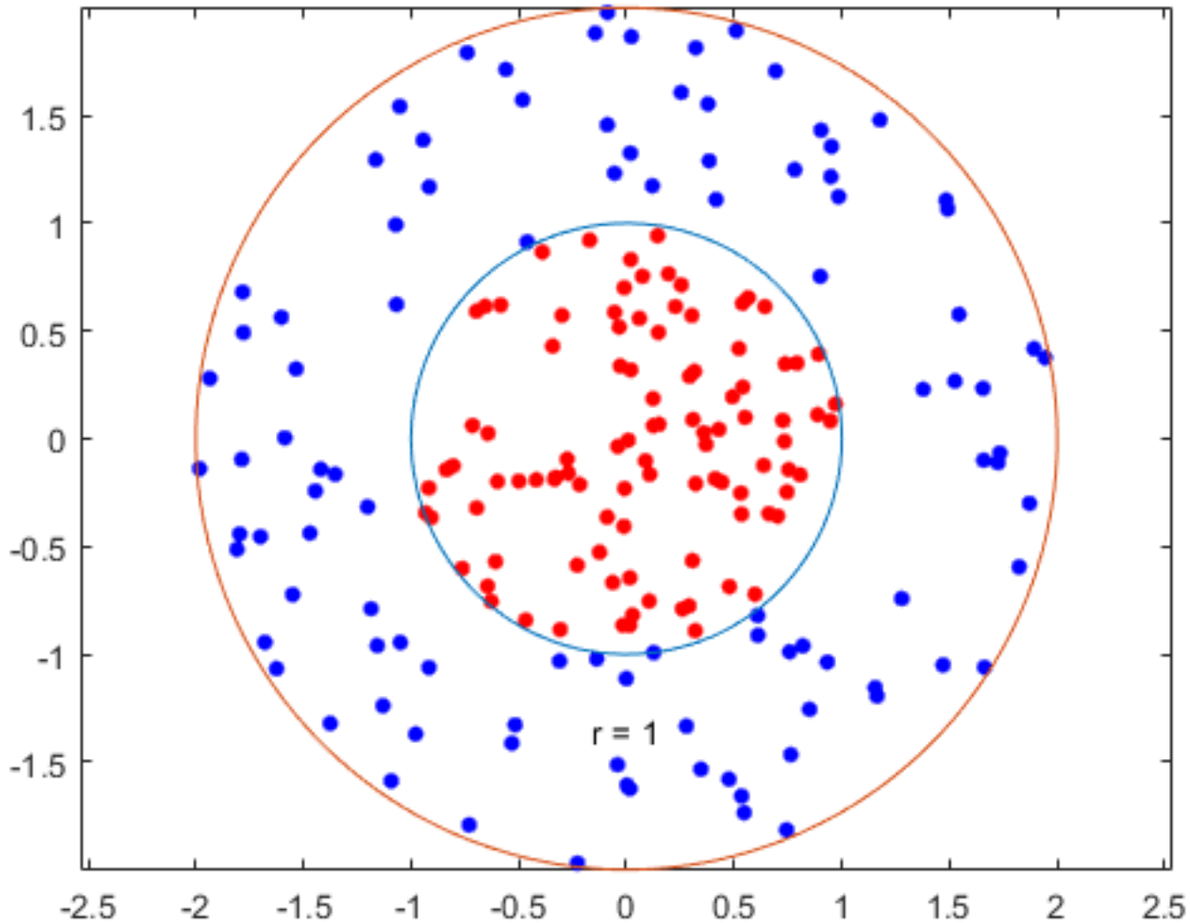
Nonlinear representations



Reminders: Oct. 31, 2019

- Moved deadline for Thought Questions 3 to Nov 14

- Any questions?

Representations for learning nonlinear functions

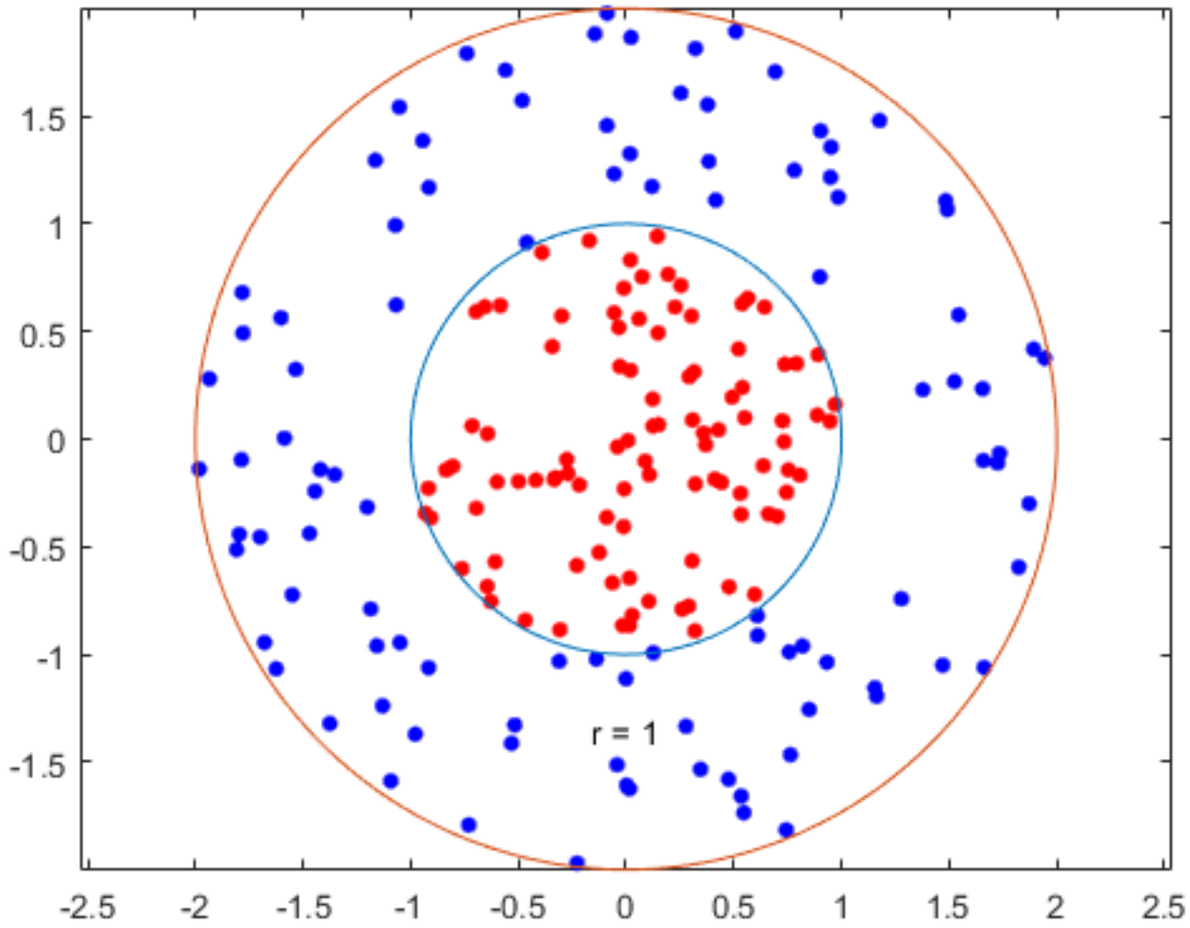
- Generalized linear models enabled many p(y | x) distributions
  - Still however learning a simple function for E[Y | x], i.e., f(<x,w>)
  - i.e. a linear function to predict f^{-1}(E[Y | x])

- 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?

$x_1^2 + x_2^2 = 1 \quad f(x) = x_1^2 + x_2^2 - 1$



$x_1 = x_2 = 0$

$\Longrightarrow \quad f(x) = -1 < 0$

$x_1 = 2, x_2 = -1$

$\Longrightarrow \quad f(x) = 4 + 1 - 1 = 4 > 0$

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

What if classes are not linearly separable? (cont…)

$x_1^2 + x_2^2 = 1 \quad f(x) = x_1^2 + x_2^2 - 1$

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ 1 \end{bmatrix} \qquad f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}$$

If use logistic regression, what is p(y=1 | x)?

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

What if classes are not linearly separable? (cont…)

$x_1^2 + x_2^2 = 1 \quad f(x) = x_1^2 + x_2^2 - 1$

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ 1 \end{bmatrix} \qquad f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}$$

Imagine learned w. How do we predict on a new x?

Nonlinear transformation

$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = \begin{pmatrix} \phi_1(\mathbf{x}) \\ \dots \\ \phi_p(\mathbf{x}) \end{pmatrix}$$

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

Predict class 1 if $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w} > 0$

Another common transformation

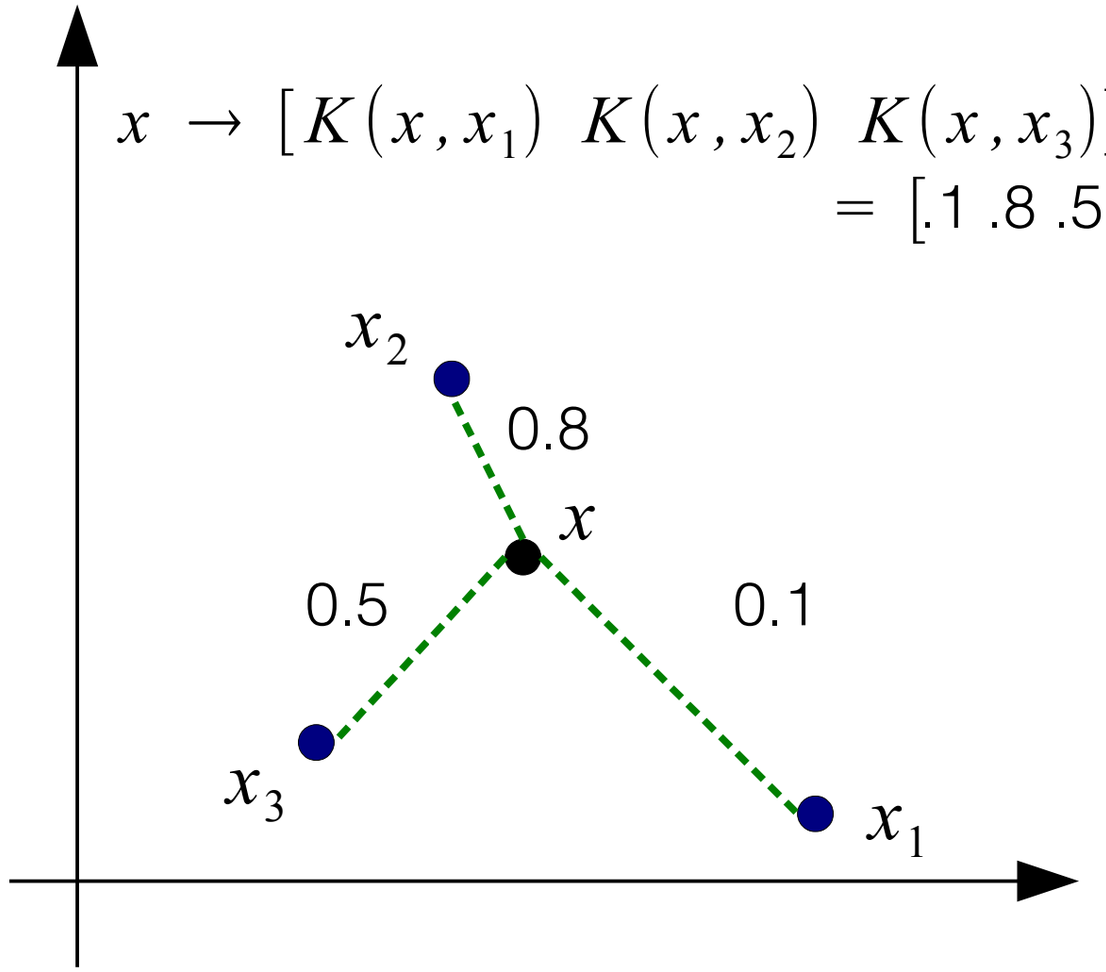
- Use similarity to a set of (representative) points or prototypes
- Intuitively, similarity features can be quite powerful: if similar to a previously observed point, should have similar predictions
- But, relies heavily on a
  - meaningful similarity measure (keywords when googling this: Radial basis functions, kernel functions)
  - picking a set of prototypes
- Let’s go through some examples!

Gaussian kernel / Gaussian radial basis function

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

$$f(\mathbf{x}) = \sum_{i=1}^{p} w_i k(\mathbf{x}, \mathbf{x}_i)$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} k(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ k(\mathbf{x}, \mathbf{x}_p) \end{bmatrix}$$



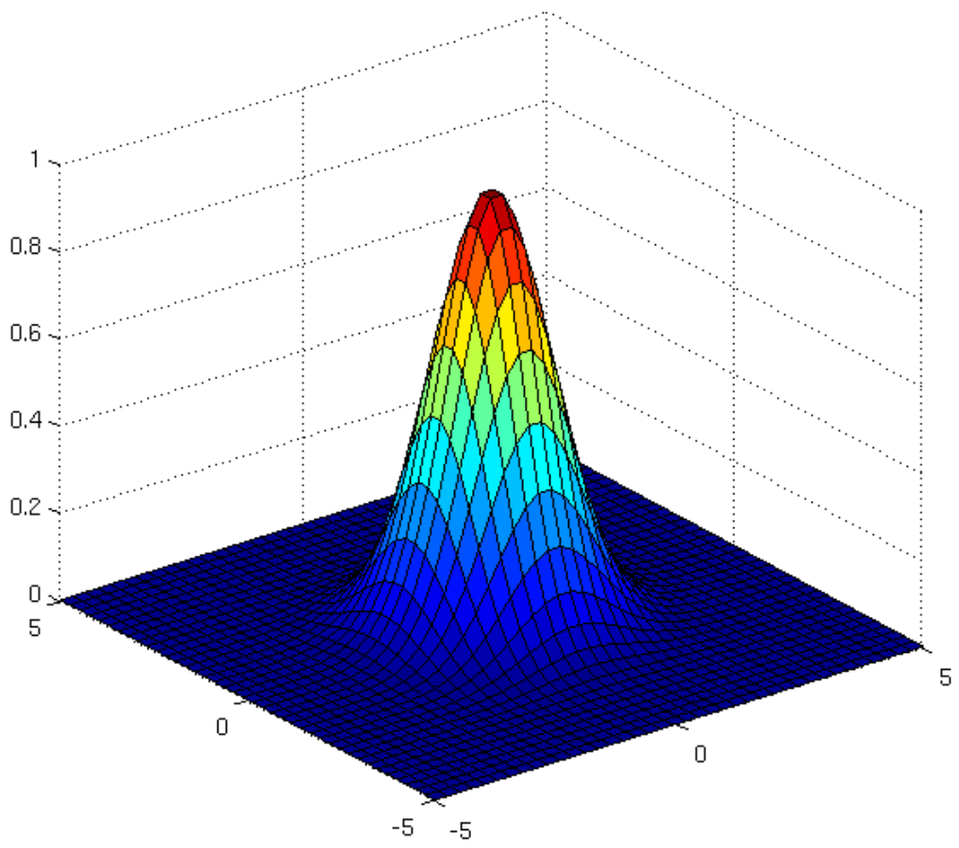
Gaussian kernel / Gaussian radial basis function

$\mathbf{x}_i$ is a prototype or center

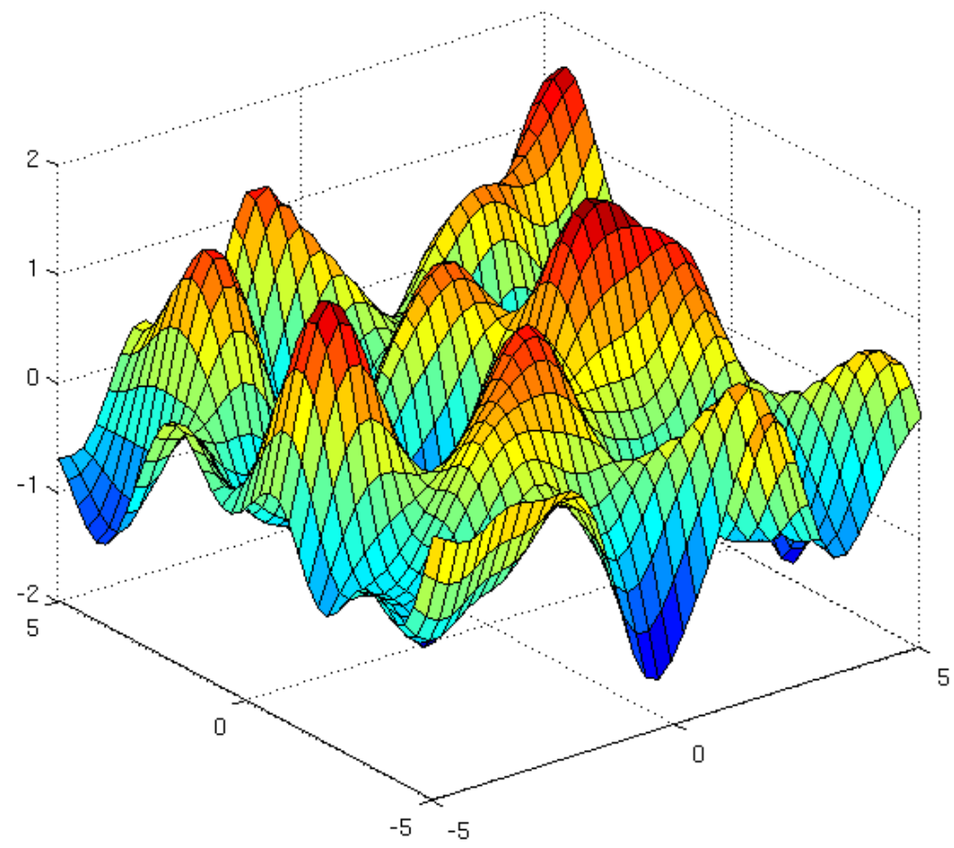
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}'\|_2^2}{\sigma^2}\right)$$

$$f(\mathbf{x}) = \sum_{i=1}^{p} w_i k(\mathbf{x}, \mathbf{x}_i)$$

Kernel



Possible function f with several centers



Can learn a highly nonlinear function!

Other similarity transforms

- Linear kernel: $k(\mathbf{x}, \mathbf{c}) = \mathbf{x}^\top \mathbf{c}$

- Laplace kernel (Laplace distribution instead of Gaussian)

$$k(\mathbf{x}, \mathbf{c}) = \exp(-b\|\mathbf{x} - \mathbf{c}\|_1)$$

- Binning transformation

$$s(\mathbf{x}, \mathbf{c}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ in box around } \mathbf{c} \\ 0 & \text{else} \end{cases}$$

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

Covering space uniformly with centres

- Imagine has 1-d space, from range [-10, 10]
- How would we pick p centers uniformly?
- What if we have a 5-d space, in ranges [-1,1]?
  - To cover entire 5-dimensional, need to consider all possible options
  - Split up 1-d into m values, then total number of centres is m^5
  - i.e., for first value of x1, can try all other m values for x2, …, x5

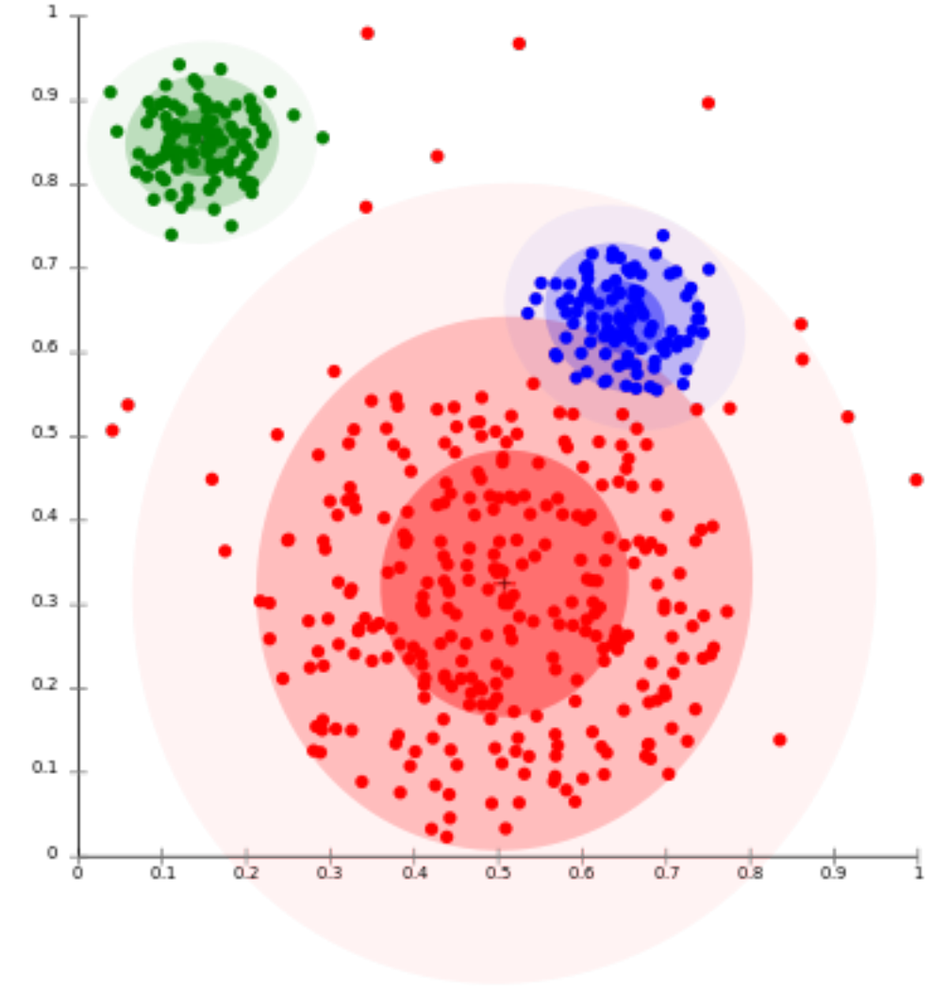
Why select training data as centers?

- Observed data indicates the part of the space that we actually need to model
  - can be much more compact than exhaustively covering the space
  - imagine only see narrow trajectories in world, even if data is d-dimensional, data you encounter may lie on a much lower-dimensional manifold (space)

- Any issues with using all training data for centres?
  - How can we subselect centres from training data?

How would we use clustering to select centres?

- Clustering is taking data and finding p groups
- What distance measure should we use?
  - If k(x,c) between 0 and 1 and k(x,x) = 1 for all x, then 1-k(x,c) gives a distance



What if we select centres randomly?

- Are there any issues with the linear regression with a kernel transformation, if we select centres randomly from the data?

- If so, any suggestions to remedy the problem?

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

Exercise

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

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

Exercise: How do we decide on the nonlinear transformation?

- We can pick a 5-th order polynomial or 6-th order, or… Which should we pick?
- We can pick p centres. How many should we pick?
- How can we avoid overparametrizing or underparameterizing?

Dealing with non-numeric data

- What if we have categorical features?
  - e.g., feature is the blood type of a person
- Even worse, what if we have strings describing the object?
  - e.g., feature is occupation, like “retail”

Some options

- Convert categorical feature into integers
  - e.g., {A, B, AB, O} —> {1, 2, 3, 4}
  - Any issues?
- Convert categorical feature into indicator vector
  - e.g., A —> [1 0 0 0], B —> [0 1 0 0], …
  - Any issues?

Using kernels for categorical or non-numeric data

- An alternative is to use kernels (or similarity transforms)
- If you know something about your data/domain, might have a good similarity measure for non-numeric data
- Some more generic kernel options as well
  - Matching kernel: similarity between two items is the proportion of features that are equal

Example: matching kernel

$\mathbf{x} =$

age	{15-24, 25-34, …, 65+}
gender	{F, M}
income	{Low, Medium, High}
education	{Bachelors, Trade-Sch, High-Sch, …}

Census dataset: Predict hours worked per week

Example: Matching similarity for categorical data

$\mathbf{x} =$

age	24-34	35-44
gender	F	F
income	Medium	Medium
education	Trade-Sch	Bachelors

$k(x_1, x_2) = k$ = 0.5

Representational properties of transformations

- Approximation properties: which transformations can approximate “any function”?
- Radial basis functions (a huge number of them)
- Polynomials and the Taylor series
- Fourier basis and wavelets

Distinction with the kernel trick

- When is the similarity actually called a “kernel”?

- Nice property of kernels: can always be written as a dot product in some feature space

$$k(\mathbf{x}, \mathbf{c}) = \boldsymbol{\psi}(\mathbf{x})^\top \boldsymbol{\psi}(\mathbf{c})$$

- In some cases, they are used to compute inner products efficiently, assuming one is actually learning with the feature expansion
  - This is called the kernel trick

- Implicitly learning with feature expansion $\boldsymbol{\psi}(\mathbf{x})$
  - not learning with expansion that is similarities to centres

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

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

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

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

Summary from last time

- Talked about using transformation (representation) that corresponds to similarities to prototypes or center

- Advantages:

- Intuitive representation
    - e.g., if feature high, then might be able to explain that the reason for a prediction is because the new point looked like that prototype point

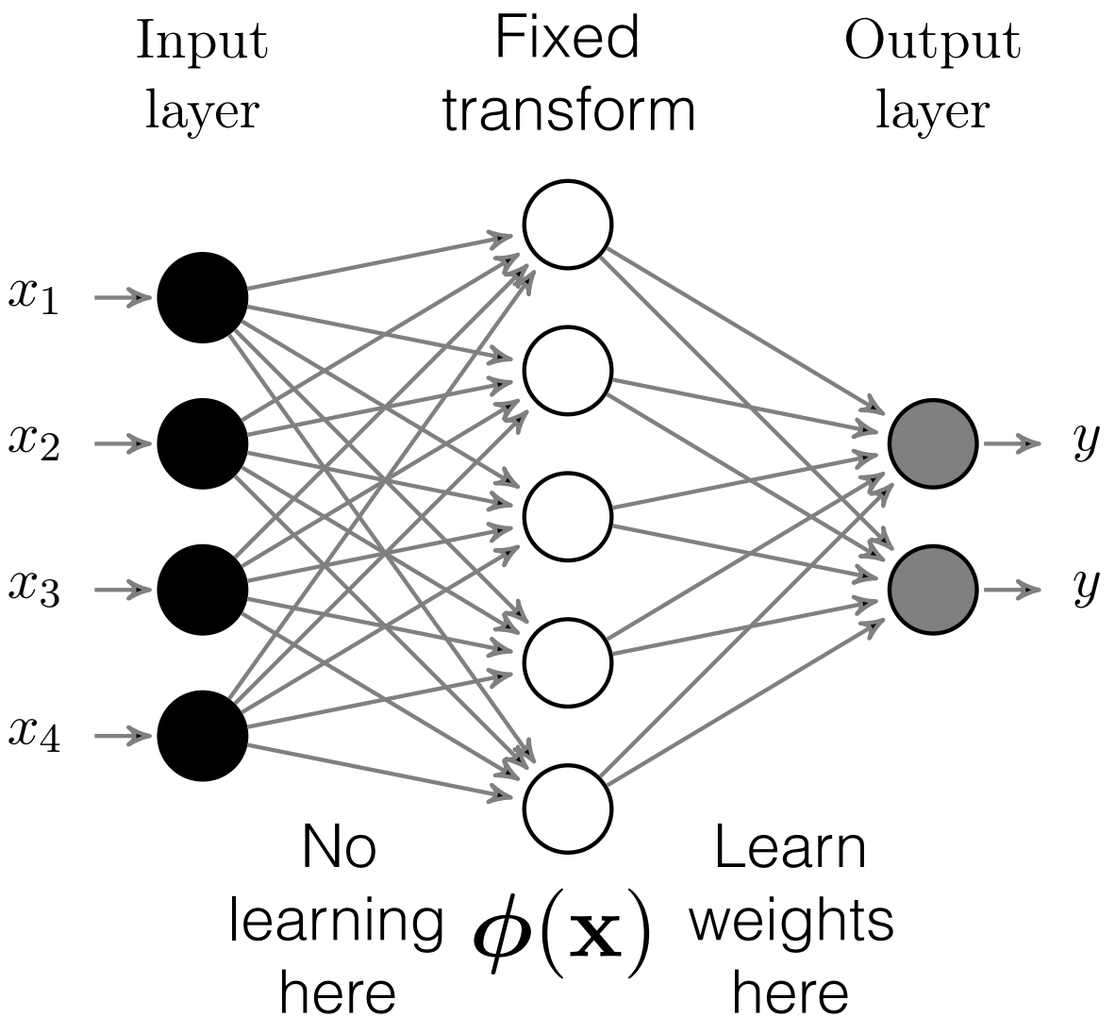
- Allows any types of inputs (categorical features, strings, etc)
    - e.g., word embeddings can provide similarity metrics —> we'll talk about embeddings later

- Nice theoretical properties, including representational capacity

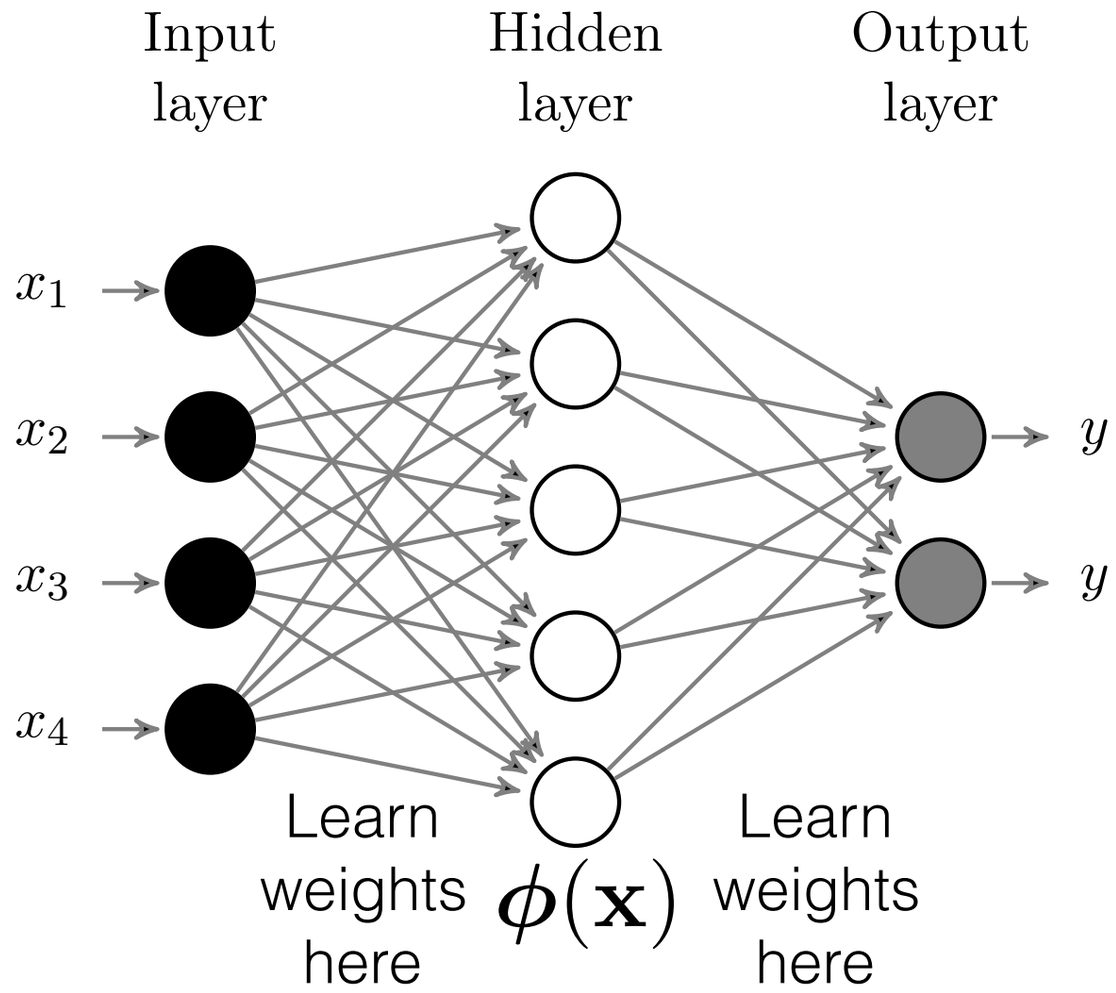
What about learning the representation?

- We have talked about fixed nonlinear transformations
  - polynomials
  - kernels
- How do we introduce learning?
  - could learn centers, for example
  - learning is quite constrained, since can only pick centres and widths
- Neural networks **learn** this transformation more from scratch
  - still some built-in structure

Fixed representation vs. NN

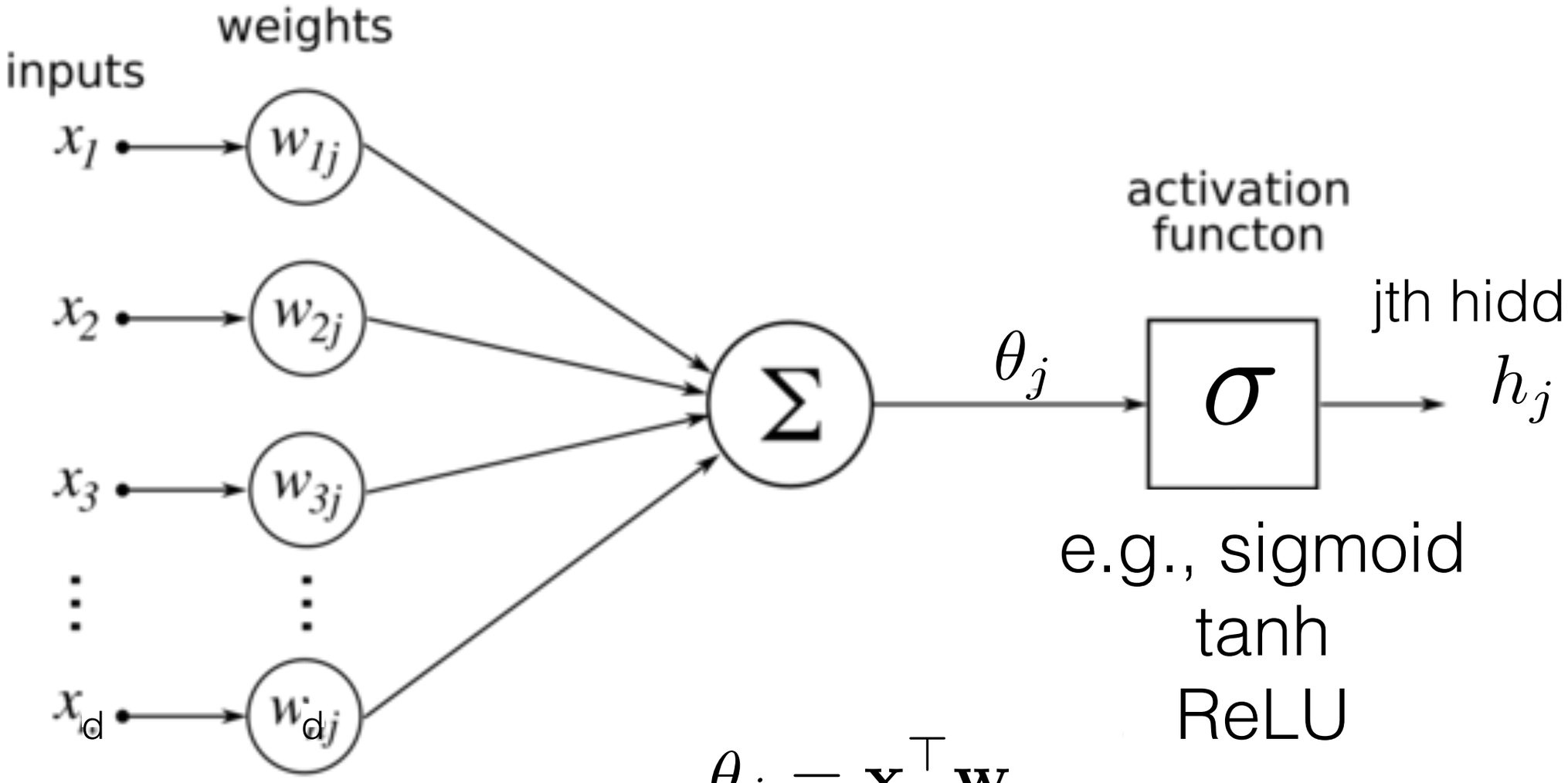


GLM with fixed representation (e.g., kernels)



Two-layer neural network

Explicit steps in one hidden node for an NN



$$\theta_j = \mathbf{x}^\top \mathbf{w}$$

$$h_j = \sigma(\theta_j)$$

Example: logistic regression and using a neural network

- The goal is still to predict p(y = 1 | x)
  - But now want this to be a more general nonlinear function of x

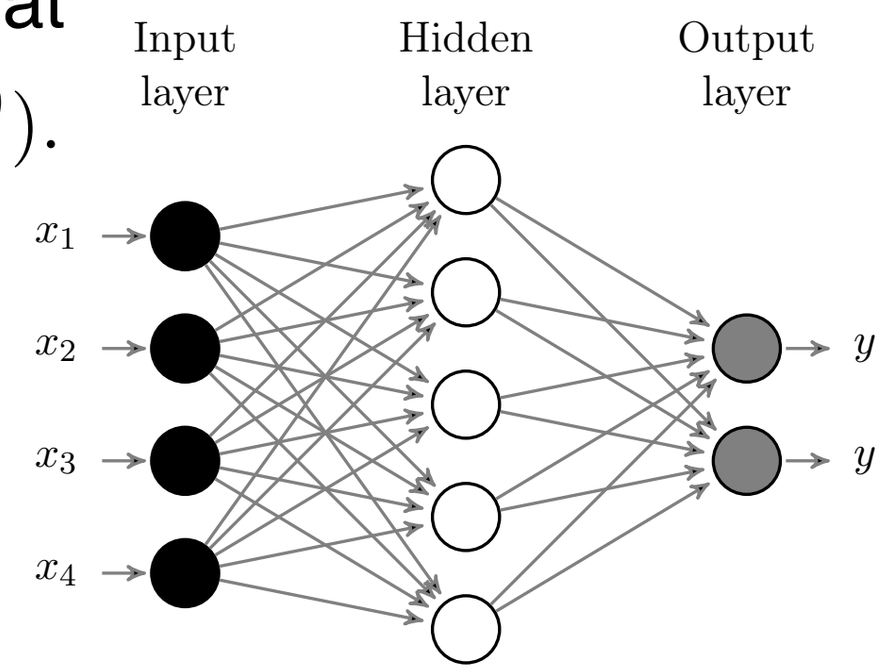
- Logistic regression learns W such that

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

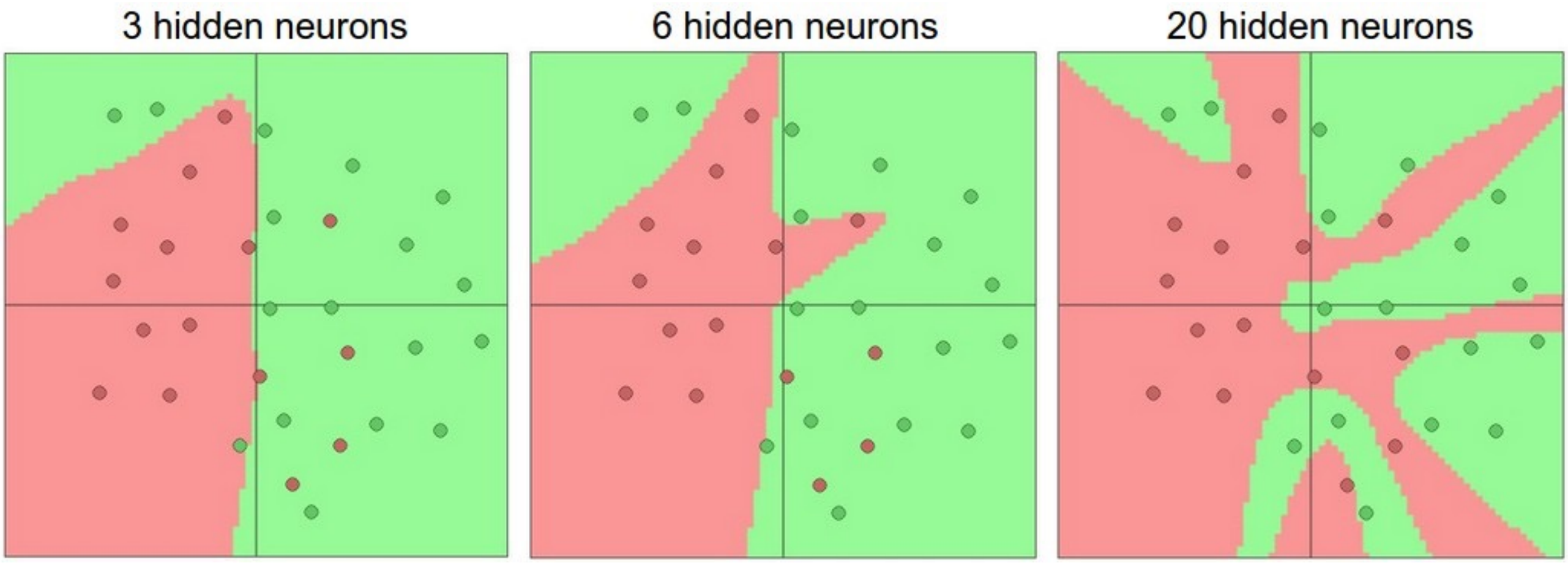
(Note: we will now start talking about x as a row vector)

- Neural network learns W1 and W2 such that

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



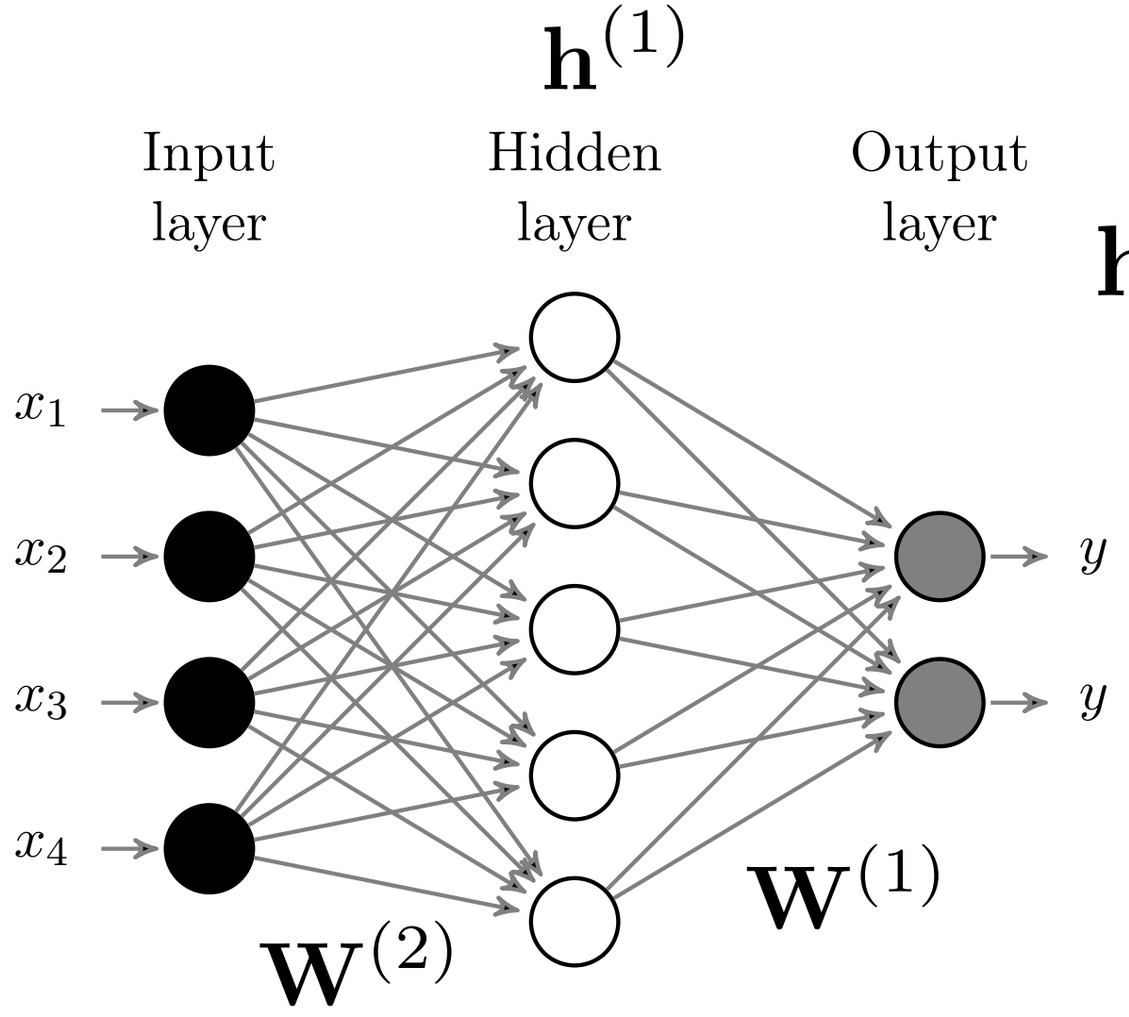
Nonlinear decision surface



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

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”



$$\mathbf{h}^{(1)} = f_2(\mathbf{x}\mathbf{W}^{(2)})$$

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

$$\text{e.g., } f_1 = \sigma$$

$$f_2 = \sigma$$

Exercise: naive Bayes and nonlinearity

- Is naive Bayes, with Gaussian p(x_j l y), a nonlinear classifier?
  - i.e., does it learn nonlinear decision surfaces, like the double circle at the beginning of these slides?

- How do we increase the modeling power of naive Bayes, i.e., how do we increase the size of the set of functions p(x l y) representable by our approximate model class?

- Can we use nonlinear transformations like kernels to do so? If so, how?

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
  - reasonably straightforward, indexing just a bit of a pain

Maximum likelihood problem

- The goal is to still to find parameters (i.e., all the weights in the network) that maximize the likelihood of the data

- What is p(y l x), for our NN?

$$E[Y|x] = NN(\mathbf{x}) = f_1(f_2(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)})$$

$$\text{e.g., mean of Gaussian, variance } \sigma^2 \text{ still a fixed value}$$

$$\text{e.g., Bernoulli parameter } p(y = 1|x) = E[Y|x]$$

$$p = NN(\mathbf{x}) = f_1(f_2(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)})$$

$$\text{Gaussian: } \sum_{i=1}^{n}(p_i - y_i)^2$$

$$\text{Bernoulli: } \sum_{i=1}^{n}\text{Cross-Entropy}(p_i, y_i)$$

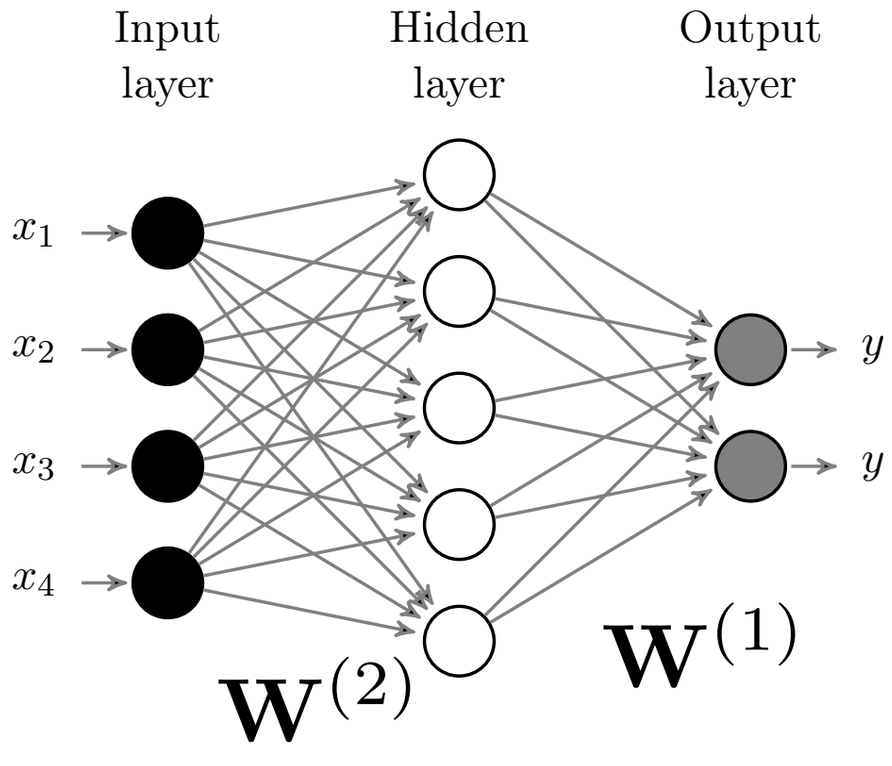
Example for p(ylx) Bernoulli

$$L(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

Cross-entropy loss

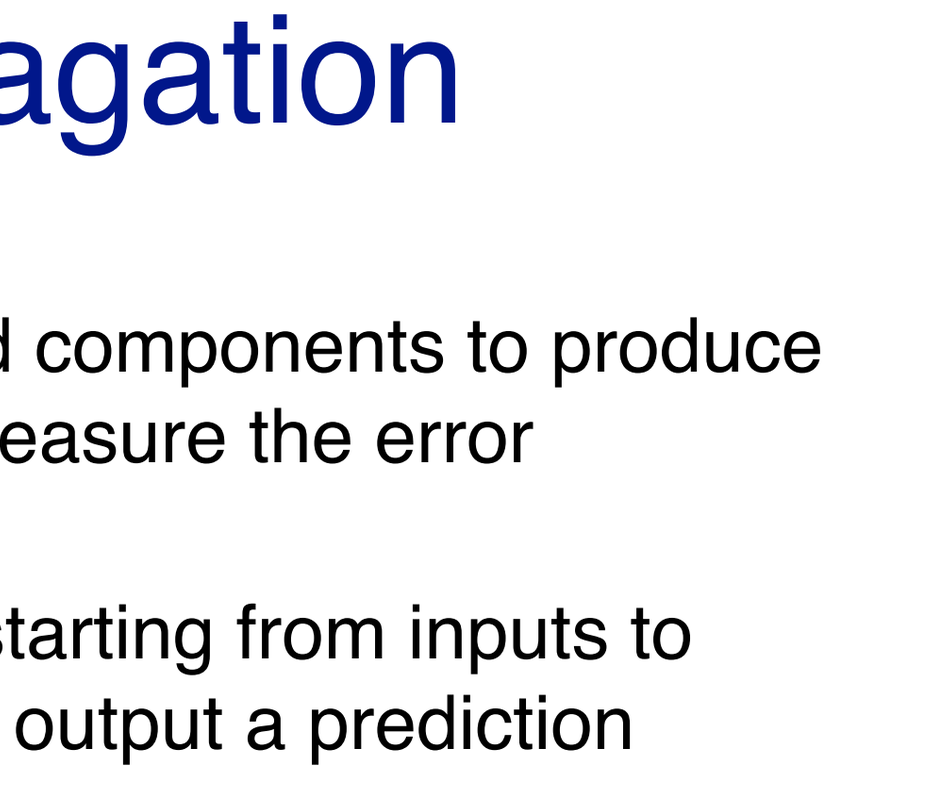
$$f_2(\mathbf{x}\mathbf{W}^{(2)}_{:j}) = \sigma(\mathbf{x}\mathbf{W}^{(2)}_{:j}) = \frac{1}{1 + \exp(-\mathbf{x}\mathbf{W}^{(2)}_{:j})}$$

$$f_1(\mathbf{h}\mathbf{W}^{(1)}_{:k}) = \sigma(\mathbf{h}\mathbf{W}^{(1)}_{:k}) = \frac{1}{1 + \exp(-\mathbf{h}\mathbf{W}^{(1)}_{:k})}$$



Forward propagation

- First have to compute all the required components to produce the prediction yhat, so that we can measure the error
- Forward propagation simply means starting from inputs to compute hidden layers to then finally output a prediction
  - i.e., simply means evaluating the function f(x) that is the NN



Backward propagation

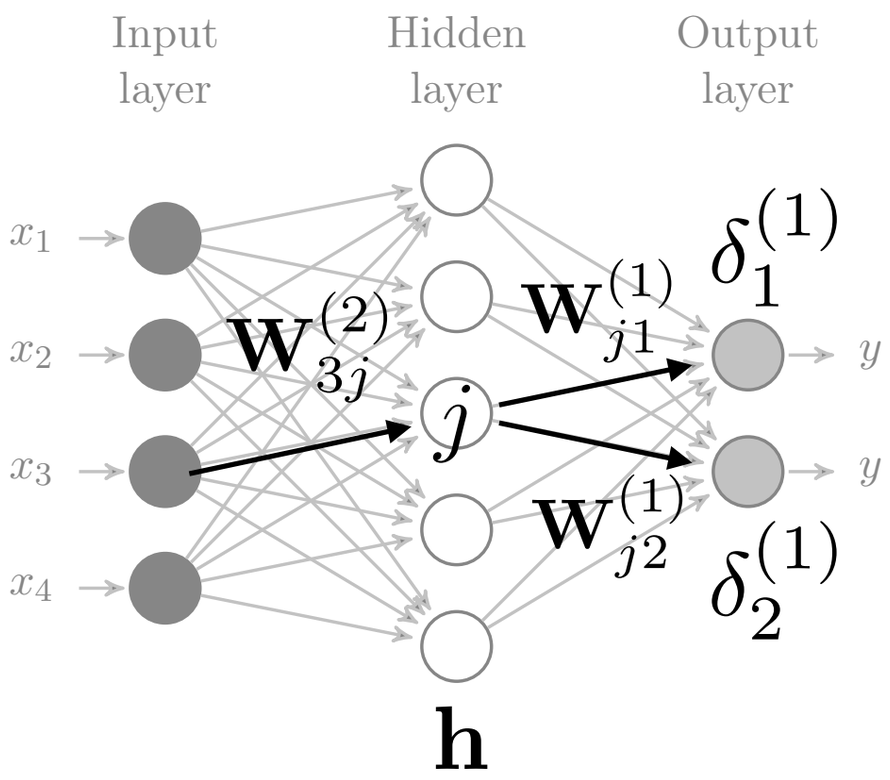
- Once have output prediction yhat (and all intermediate layers), can now compute the gradient

- The gradient computed for the weights on the output layer contains some shared components with the weights for the hidden layer

- This shared component is computed for output weights W1

- Instead of recomputing it for W2, that work is passed to the computation of the gradient of W2 (propagated backwards)

Example for Bernoulli (cont)



$$\boldsymbol{\delta}^{(1)}_k = \hat{y}_k - y_k$$

$$\frac{\partial}{\partial \mathbf{W}^{(1)}_{jk}} = \boldsymbol{\delta}^{(1)}_k \mathbf{h}_j$$

$$\boldsymbol{\delta}^{(2)}_j = \left(\mathbf{W}^{(1)}_{j:} \boldsymbol{\delta}^{(1)}\right) \mathbf{h}_j (1 - \mathbf{h}_j)$$

$$\frac{\partial}{\partial \mathbf{W}^{(2)}_{ij}} = \boldsymbol{\delta}^{(2)}_j \mathbf{x}_i$$

$$\longrightarrow$$

$$\mathbf{W}^{(1)}_{jk} \leftarrow \mathbf{W}^{(1)}_{jk} - \alpha \frac{\partial}{\mathbf{W}^{(1)}_{jk}}$$

$$\mathbf{W}^{(2)}_{ij} \leftarrow \mathbf{W}^{(2)}_{ij} - \alpha \frac{\partial}{\mathbf{W}^{(2)}_{ij}}$$