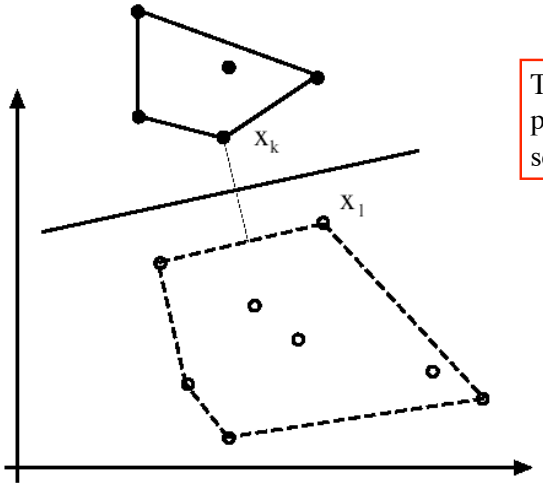
Support vector machines (SVM)

- A generic, simple, standard way to build a classifier is with a SVM
- The basic "plug-in classifier" (black box)
- Very convenient software is available to do this.
- We will cover the approach briefly

Support vector machines (SVM)



The SVM finds a plane that best separates the data.

Support vector machines

- If we have a *separating* hyperplane, then if you are on one side $\mathbf{w} \bullet \mathbf{x_i} + b \geq +1$
- If you are on the other side $\mathbf{w} \bullet \mathbf{x_i} + b \leq -1$
- Let y_i be +1 for one class, -1 for the other.

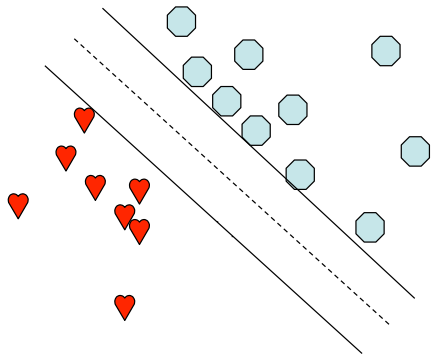
Support vector machines

- Linearly separable data means that we can chose

$$y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b \right) \geq 1$$

- Consider the best pair of parallel planes that push against points on the two groups.

Support vector machines



Support vector machines

- Consider the best pair of parallel planes that push against points on the two groups.
- The sum of the minimum distances from each group to the other plane can be shown to be:

$$\frac{2}{|\mathbf{w}|}$$

Support vector machines

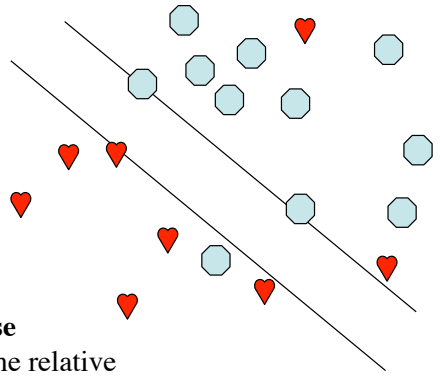
- Solved by

$$\text{minimize} \quad (1/2)\boldsymbol{w} \cdot \boldsymbol{w}$$

$$\text{subject to} \quad y_i \left(\boldsymbol{w} \cdot \boldsymbol{x}_i + b\right) \geq 1$$

- What if the data is not linearly separable
  - Find "best" plane (need to balance cost of misclassification)
  - The boundary is determined by a few points (the support vectors)

Support vector machines

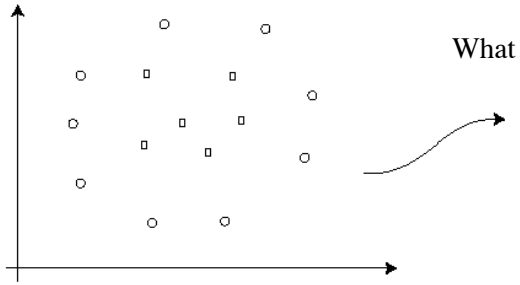


Non-separable case
Cost, C, specifies the relative desire to push the planes apart, verses the number of mistakes.

Support vector machines

- We have found the "best" plane from labeled training data
- How do we classify a new "test" point that has no label
  - Easy---the simple formula tells us which side of the plane we are on!
- Pseudo probabilities can be created from the distance to the plane
- This describes a binary classifier. For more than one class, there are a number of approaches
  - Multiple one against all
  - All against all, and a consensus measure
  - Train a multi-class classifier (Crammer JMLR 2001)

Support vector machines (kernel tricks)



Support vector machines (kernel tricks)

The SVM is completely a function of dot products between the vectors (this would be clear if we did it in more detail)

This means that we can get a non-linear SVM by using a different form of the dot product, K(**x**,**y**).

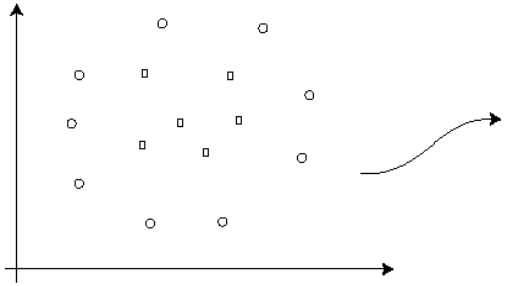
This is equivalent to a linear classification in a much higher dimensional space.

Support vector machines (kernel tricks)

For example, we can produce a higher dimensional space using polynomials = of the original points, e.g.,

$$(x,y) \rightarrow \left(x^2, xy, y^2, x, y\right) = \left(u_0, u_1, u_2, u_3, u_4\right)$$

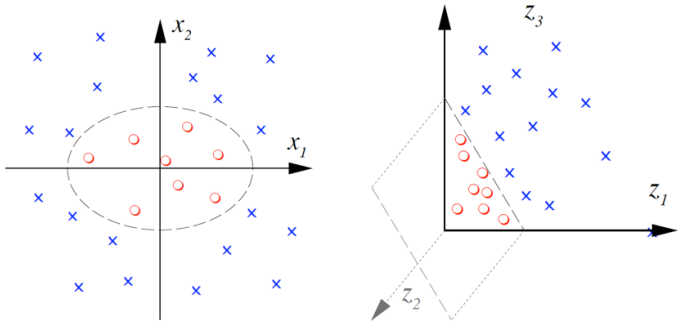
Support vector machines (kernel tricks)



Can you imagine a 3D space in which the obvious decision boundary is linear?

A similar example (Originally from Schölkopf and A. J. Smola)

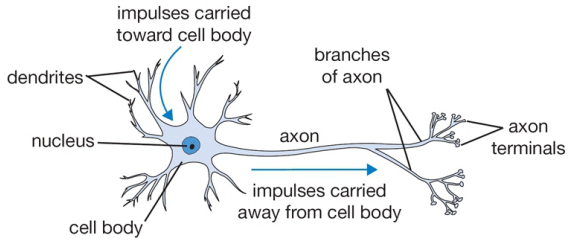
$$\Phi : R^2 \to R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)} x_1 x_2, x_2^2)$$

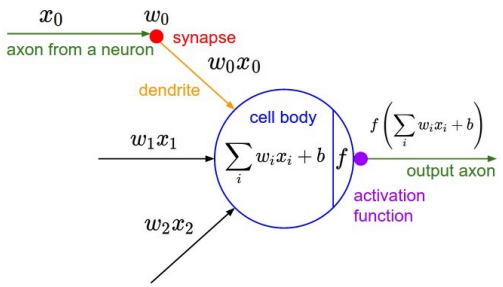


Artificial Neural Networks (ANN)

- Significant attention in the 80's
- Most researchers moved on to other things
- Other, often simpler, methods became popular
- Late 90's SVM became an easy way to get ANN performance
- Now, complex ("deep") neural networks usually outperform SVM, provided sufficient data and modern training ideas

Many slides adapted from Clay Morrison's machine learning class

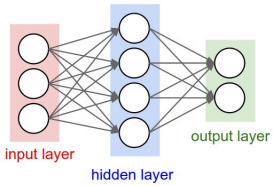




http://cs231n.github.io/neural-networks-1/

Multi-level networks

- A single artificial neuron has limited computing power
  - Interesting networks have at least three* layers



  - We will restrict our attention to feed-forward networks (no arrows from later stages going back to earlier ones)

*Some (e.g., Bishop) count the number of arrow blocks, so this would be a two layer network.

Neural network as function approximators

- Our three layer network is a function from input to output
- ANNs can approximate any "reasonable" function
  - This requires a nonlinear shaping function, f().
- Function approximators have been studied in the context of "no free lunch" theorems
  - (Worth reading about, not part of this course)

Layering and composition

- Our three layer network can be expressed as the composition $\mathbf{y} = f(\mathbf{x}) = f_2(f_1(\mathbf{x}))$
- Similarly "deep" networks are bigger compositions
- Note that we like the functions to be differentiable, and the chain rule is going to be useful.

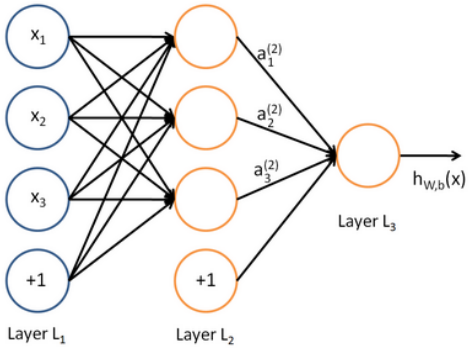
Notation

- Keeping track of everything (book keeping) is much of the heavy lifting in ANNs.
- Notation differs
  - Sometimes we have explicit "bias" sometimes we assume bias nodes frozen at "1", which means we can treat bias as a weights
    - We will sometimes treat bias as weights for simplicity
    - More advance use might require different regularizations for weights and bias terms, and then you want them separate
  - Bishop (PRML) reverses *x* and *a* from these slides. Also his *h()* is our *f()*.

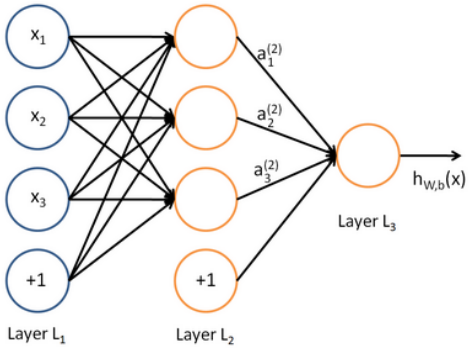
Notation*

- We will use super-scripts for network level. The level, l, for weights is between layer l and layer $l+1$.
  - Layer one is the input, and the first sets of weights are also indexed by one.
- We will also use super-scripts to index data points
- We use i,j for the weight between nodes j (current level) and i in the next layer.
  - The first index is where the information is going **to**.
- The activations, a are fed into the next layer inside linear function in which case they are sometimes called x.

*The notation (and images) in what follows is modified from http://ufldl.stanford.edu/wiki/index.php/Neural_Networks



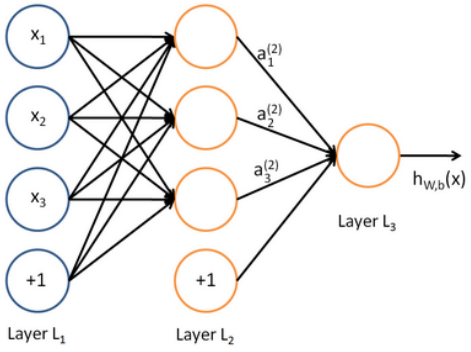
- This shows one output; often there are many
- The bottom nodes are locked in at +1. This represents the bias (offset).
- The following formulas use separate for biases (b's) and weights (w's).



$a_i^{(1)} = x_i$
(input layer)

For $l \geq 1$ $\begin{cases} z_i^{(l+1)} = \sum_{j=1}^{N^{(l)}} W_{ij}^{(l)} a_i^{(l)} + b_i^{(l)} \\ \text{followed by} \quad a_i^{(l+1)} = f\left(z_i^{(l+1)}\right) \end{cases}$

$$f(z) = \frac{1}{1+\exp(-z)} \quad \text{(for example)}$$



$$\mathbf{z}^{(l+1)} = W^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)}$$
$$\mathbf{a}^{(l+1)} = f\left(\mathbf{z}^{(l+1)}\right)$$

(Matrix-vector form, f(•) is applied element-wise)

Training

- Tweak the weights so that for each training instance, the output (e.g., label) for each input vector (e.g., image) is close to correct

Training

- Tweak the weights so that for each training instance, the output (e.g., label) for each input vector (e.g., image) is close to correct
- Optimize the objective function:

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}\left(\mathbf{x}^{(i)}\right) - \mathbf{y}^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

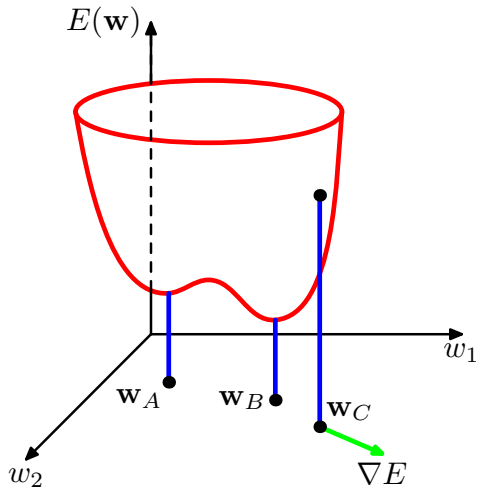
Training error, superscripts are training points

Neural networks are prone to overfitting, so regularize. This regularization term is just one example—there are many others.

Training

- Tweak the weights so that for each training instance, the output (e.g., label) for each input vector (e.g., image) is close to correct
- Gradient descent
  - Look at the effect of changing the weights on the error
  - Take a step in the reverse direction (to reduce it)
  - Stochastic version perturbs the step to mitigate going to the nearest local minimum
  - Conceptually, we estimate the gradient using each data point, and then take a small step based on each one
- In what follows we switch notation: $E(\mathbf{w}) \equiv J(W,b)$

Training



Training

- We consider computing the gradient, and what we do with it as separate tasks
- We generally compute the gradient in training
- We might compute it due to a specific data point (on-line) or for all points considered together
- Once we estimate the gradient we then might
  - Follow it (e.g., conjugate gradient descent)
  - Follow it stochastically
  - Pass it to some other fancy optimizer