Today's Lecture

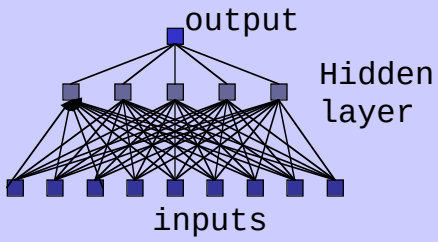
- Neural networks
  - Training
    - Backpropagation of error (backprop)
  - Example
  - Radial basis functions

Recall: training

For a single input-output layer, we could adjust the weights to get linear classification.

- The perceptron computed a **hyperplane** over the space defined by the inputs.
  - This is known as a **linear classifier.**

- By stacking layers, we can compute a wider range of functions.
- Compute error derivative with respect to weights.



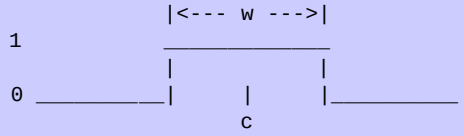
- "Train" the weights to correctly classify a set of examples (TS: the training set).
- Started with perceptron, which used summing and a step function, and binary inputs and outputs.
- Embellished by allowing continuous activations and a more complex "threshold" function.
  - In particular, we considered a **sigmoid** activation function, which is like a "blurred" threshold.

The Gaussian

- Another continuous, differentiable function that is commonly used is the Gaussian function.

$$\text{Gaussian}(x) = e^{\frac{x^2}{2\sigma}}$$

- where σ is the width of the Gaussian.
- The Gaussian is a continuous, differentiable version of the step function.



What is learning?

- For a fixed set of weights $w_1,...,w_n$

  $$f(x_1,...,x_n) = Sigma(x_1\ w_1 + ... + x_n\ w_n)$$

  represents a particular scalar function of n variables.
- If we allow the weights to vary, then we can represent a family of scalar function of n variables.

  $$F(x_1,...,x_n,w_1,...,w_n) = Sigma(x_1\ w_1 + ... + x_n\ w_n)$$
- If the weights are real-valued, then the family of functions is determined by an n-dimensional parameter space, R^n.
- Learning involves searching in this parameter space.

Basis functions

- Here is another family of functions. In this case, the family is defined by a linear combination of basis functions

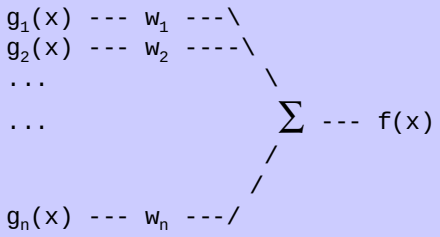
  $$\{g_1,g_2,...,g_n\}.$$

The input x could be scalar or vector valued.

$$F(x,w_1,...,w_n) = w_1\ g_1(x) + ... + w_n\ g_n(x)$$

Combining basis functions

We can build a network as follows:



E.g. From the basis $\{1,x,x^2\}$ we can build quadratics:

$$F(x, w_1, w_2, w_3) = w_1 + w_2 x + w_3 x^2$$

Receptive Field

- It can be generalized to an arbitrary vector space (e.g., R^n).
- Often used to model what are called "**localized receptive fields**" in biological learning theory.
  - Such receptive fields are specially designed to represent the output of a learned function on a small portion of the input space.
  - How would you approximate an arbitrary continuous function using a sum of gaussians or a sum of piecewise constant functions of the sort described above?

Backprop

- Consider sigmoid activation functions.
- We can examine the output of the net as a function of the weights.
  - How does the output change with changes in the weights?
  - Linear analysis: consider partial derivative of output with respect to weight(s).
    - We saw this last lecture.
  - If we have multiple layers, consider effect on each layer as a function of the **preceding** layer(s).
    - We propagate the error backwards through the net (using the chain rule for differentiation).
- Derivation on overheads [reference: DAA p. 212]

Backprop observations

- We can do gradient descent in weight space.
- What is the dimensionality of this space?
  - Very high: each weight is a free variable.
    - There are as many dimensions as weights.
    - A "typical" net might have hundreds of weights.
- Can we find the minimum?
  - It turns out that for multi-layer networks, the error space (often called the "**energy**" of the network) is **NOT CONVEX**. [so?]
  - Commonest approach: multiple restart gradient descent.
    - i.e. Try learning given various random initial weigh t distributions.

Success? Stopping?

- We have a training algorithm (backprop).

- We might like to ask:
  - 1. Have we done enough training (yet)?
  - 2. How good is our network at solving the problem?
  - 3. Should we try again to learn the problem (from the beginning)?
- The first 2 problems have standard answers:
  - Can't just look at energy. Why not?
    - Because we want to GENERALIZE across examples. "I understand multiplication: I know 3*6=18, 5*4=20."
      - What's 7*3? Hmmmm.
  - Must have additional examples to **validate** the training.
    - Separate input data into 2 classes: training and testing sets. Can also use **cross-validation.**

What can we learn?

- For any mapping from input to output units, we can learn it if we have enough hidden units with the right weights!
- In practice, many weights means difficulty.
- The right representation is critical!

- Generalization depends on bias.
  - The hidden units form an **internal representation** of the problem. make them learn something general.
    - Bad example: one hidden unit learns exactly one training example.
      - Want to avoid learning by table lookup.

Representation

- Much learning can be equated with selected a good problem representation.
  - If we have the right hidden layer, things become easy.
- Consider the problem of face recognition from photographs. Or fingerprints.
  - Digitized photos: a big array (256x256 or 512x512) of intensities.
  - How do we match one array to another? (Either manually or by computer.)
  - Key: measure important properties, use those as criteria for estimating similarity?

Faces (an example)

- What is an important property to measure for faces?
  - Eye distance?
  - Average intensity
    - BAD!
  - Nose width?
  - Forehead height?
- These measurements form the basis functions for describing faces.
  - BUT NOT NECESSARILY photographs!!!
    - We don't need to reconstruct the photo. Some information is not needed.

Radial basis functions

- Use “blobs” summed together to create an arbitrary function.
  - A good kind of blob is a Gaussian: circular, variable width, can be easily generalized to 2D, 3D, ….

Topology changes

- Can we get by with fewer connections?
- When every neuron from one layer is connected to every layer in the **next** layer, we call the network fully-connected.
- What if we allow signals to flow **backwards** to a preceding layer?

Recurrent networks