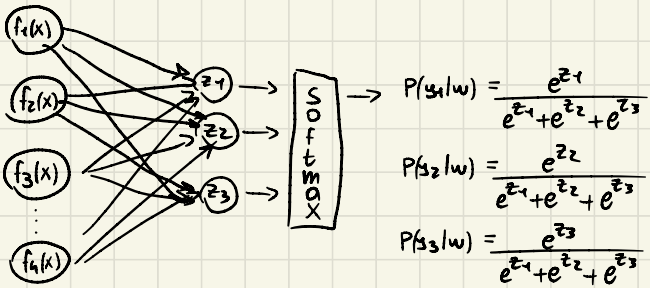
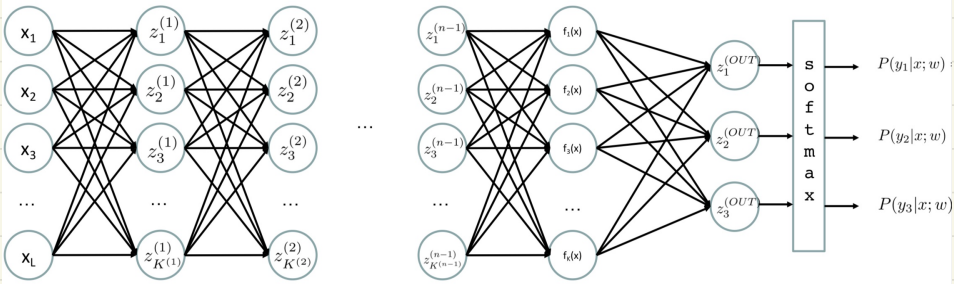
multiclass logistic regression is a special case of neural networks





$$z_i^{(k)} = g\left(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}\right)$$

g = nonlinear activation function

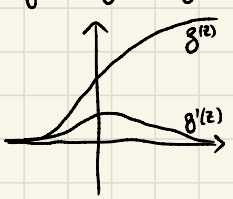
non linear activation function used in deep neural networks

some common are:

Sigmoid function

$g(z) = \frac{1}{1+e^{-z}}$

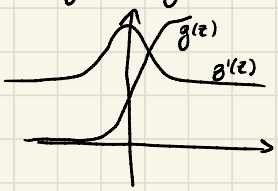
$g'(z) = g(z)(1-g(z))$



Hyperbolic tangent

$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

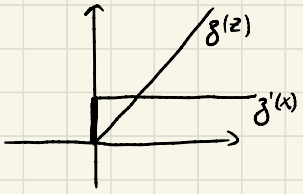
$g'(z) = 1 - g(z)^2$



Rectified linear unit (ReLu)

$g(z) = \max(0, z)$

$g'(z) = \begin{cases} 1 & z>0 \\ 0 & \text{otherwise} \end{cases}$



training is just like logistic regression, but larger so run gradient ascent

Theorem of universal function approximators

a two layer neural network with a sufficient # of neurons can approximate any continuos function to any desired accuracy.

This can be seen as learning the features
with a large number of neurons there is the danger of overfitting -> early stopping

Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure μ, standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

Hornik theorem 2: Whenever the activation function is *continuous, bounded and nonconstant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

- In words: Given any continuous function f(x), if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate f(x).

Summary

optimize probability of label given input $\max_w ll(w) = \max_w \sum_i \log P(y^i | x^i; w)$

continuos optimization

gradient Ascent:

- compute steepest uphill direction = gradient (= just vector of partial derivatives)
- take step in the gradient direction
- repeat (until held-out data accuracy starts to drop)

Deep neural nets

- last layer = logistic regression
- layers in between
  - compute features (learned)
- universal approximation theorem
  neural net large enough -> can represent continuos mapping with arbitrary accuracy
- automatic differentiation gives derivatives efficiently

Inductive learning

simplest form → learn a function from examples

target function: g

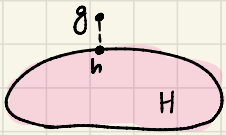
examples: input-output pairs $(x, g(x))$

problem:

given a hypothesis space H

given training set of examples x_i

find hyp. $h(x) \mid h \sim g$



classification gives labels

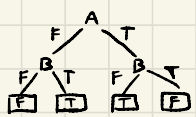
regression gives real numbers

Decision tree

compact representation of a function

can express any function of the features

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F



$P(C \mid A, B)$

difference with perceptrons relative evidence weighting (NB), complex evidence interaction (DTs)

How many DTs with n bool values?

$2^{(2^n)}$

How many trees of depht 1?

$4n$

Learning

Aim: find a small tree consistent with the training examples

idea: (recursively) choose "most significant" attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Information

answers questions

more incertain -> more info in the answer

probabilities p tipical of

uniform dist. of size $1/p$

code of lenght $\log 1/p$

Entropy

general answer: if prior is $\langle p_1, \dots p_n \rangle$

$$H(\langle p_1, \dots, p_n \rangle) = E_p \log_2 1/p_i = \sum_{i=1}^{n} -p_i \log_2 p_i$$

In DT check entropy for each split before and after