MACHINE LEARNING IN BIOINFORMATICS

Part 8: Neural Networks

František Mráz
KSVI MFF UK

Adapted slides of Jianlin Cheng
Department of Computer Science University of Missouri, 2012

Neural networks

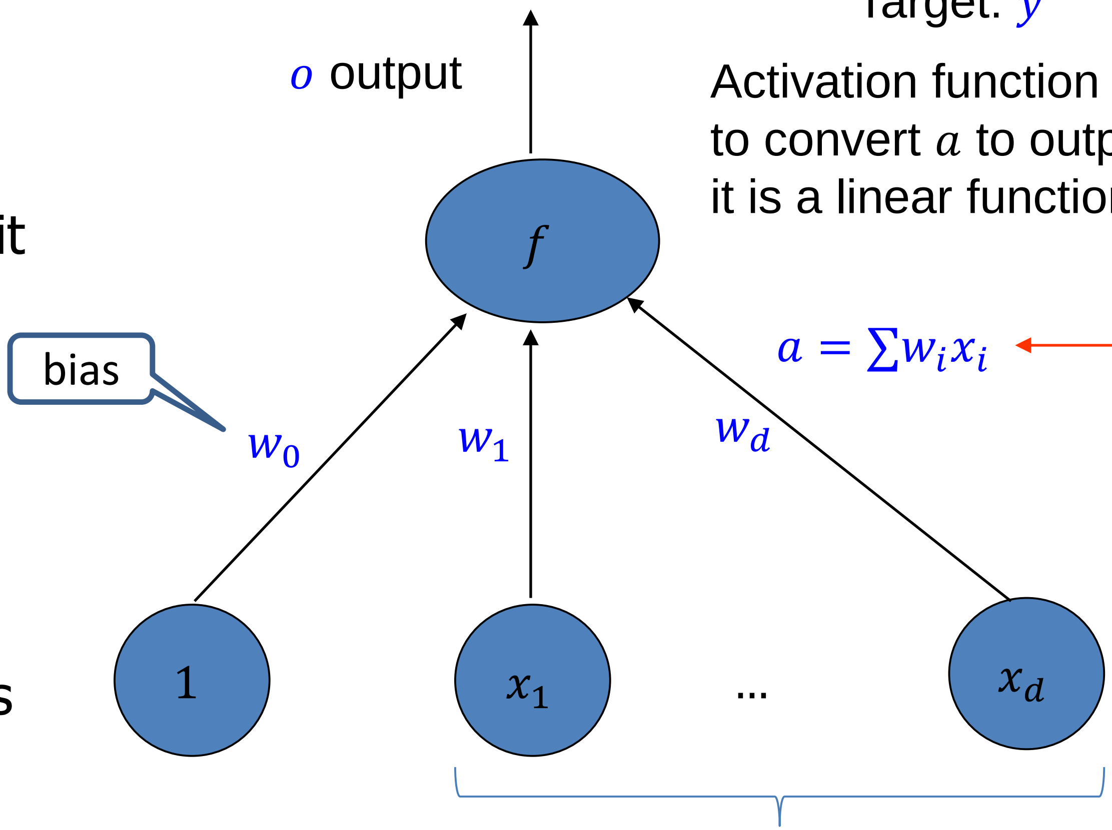
- Both supervised and unsupervised learning
- Both regression (a real-value output) and classification (discrete output)
- Background:
  1. Neurology – artificial intelligence would like to utilize it
  2. **Statistics** – linear regression, generalized linear regression, discriminant analysis

Neuron

Target: y

Activation function f is used to convert a to output. Here it is a linear function: $o = f(a)$.

- Output unit
- Input units



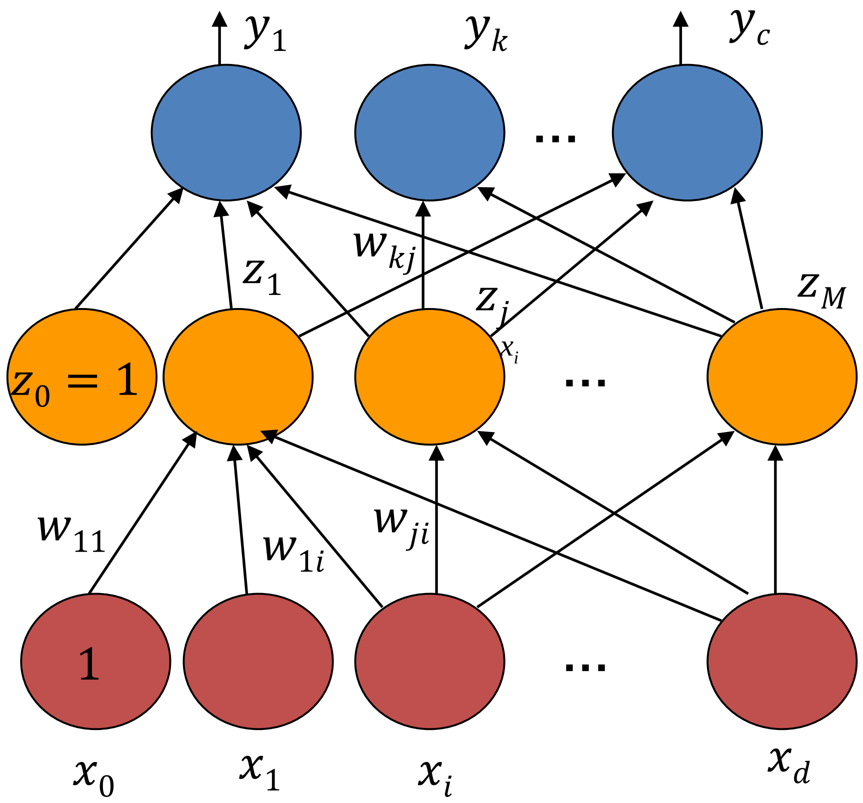
$a = \sum w_i x_i$ ← Activation

Input vector $(x_1, \dots, x_d)$

Activation functions

- Linear $f(a) = a$
- Sigmoid $f(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent $f(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- Rectified linear unit $f(a) = \begin{cases} 0 & \text{for } a < 0 \\ a & \text{for } a \geq 0 \end{cases}$
- ...

Multi-Layer Neural Network



- **Output**: Activation function: f (linear, sigmoid, softmax)
- **Hidden layers**: Activation function: g (linear, tanh, sigmoid)
- **Input**: no activation function

Multi-Layer Perceptron

- Two-layer neural network (one hidden and one output) with non-linear activation function is a **universal function approximator**
  - it can approximate any numeric function with arbitrary precision given a set of appropriate weights and hidden units.
- In early days, usually two-layer (or three-layer if you count the input as one layer) neural network. Increasing the number of layers was occasionally helpful.
- Later expanded into deep learning with many layers

Adjust Weights by Training

- How to adjust weights?
- Adjust weights using known examples (training data)

$$\left\{\left(x_1^{(1)}, x_2^{(1)}, \dots, x_d^{(1)}, t^{(1)}\right), \dots, \left(x_1^{(n)}, x_2^{(n)}, \dots, x_d^{(n)}, t^{(n)}\right)\right\}$$

  where $t^{(i)}$ are the target (desired) outputs
- Try to adjust weights so that the difference between the output of the neural network y and t (target) becomes smaller and smaller.
- Goal is to minimize error function

$$E = \sum_{i=1}^{n} \left(y^{(i)} - t^{(i)}\right)^2$$

  where $y^{(i)}$ is the actual output of the network
- **Idea:** gradient descent – update weight according

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}}$$

t is time

Learning rate

Algorithm

- **Initialize** weights w
- **Repeat**

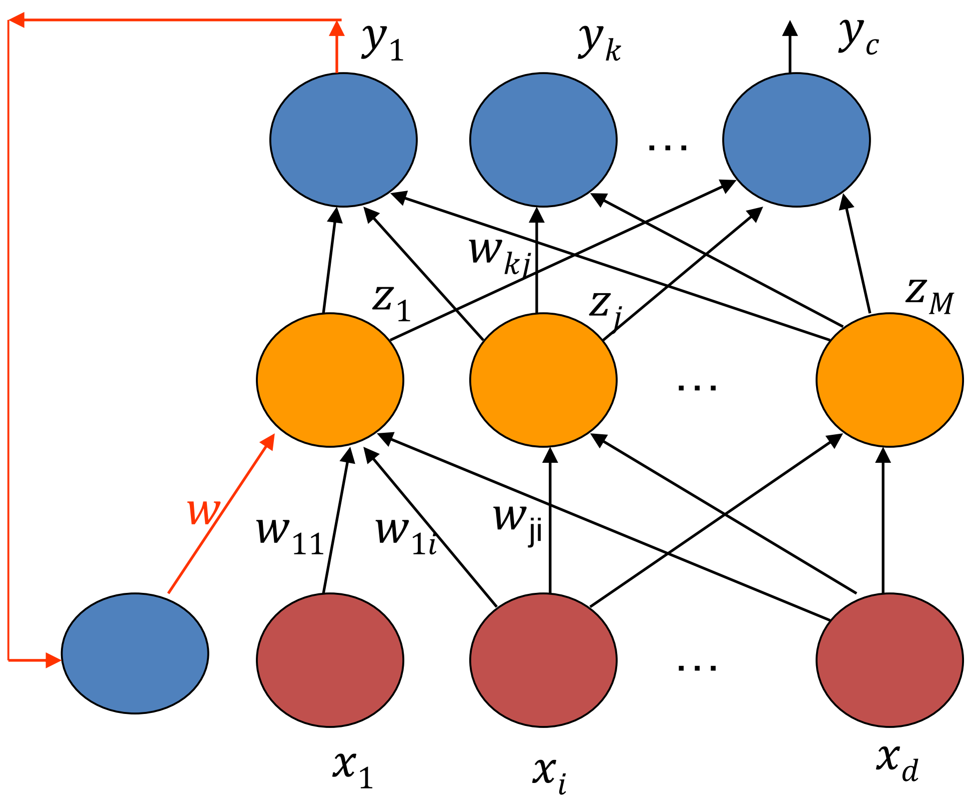
  For each data point x, do the following:

  Forward propagation: compute outputs and activations

  Backward propagation: compute errors for each output units and hidden units. Compute gradient for each weight.

  Update weight $w = w - \eta \frac{\partial E}{\partial w}$
- **Until** a given number of iterations or errors drops below a threshold.

Recurrent Network



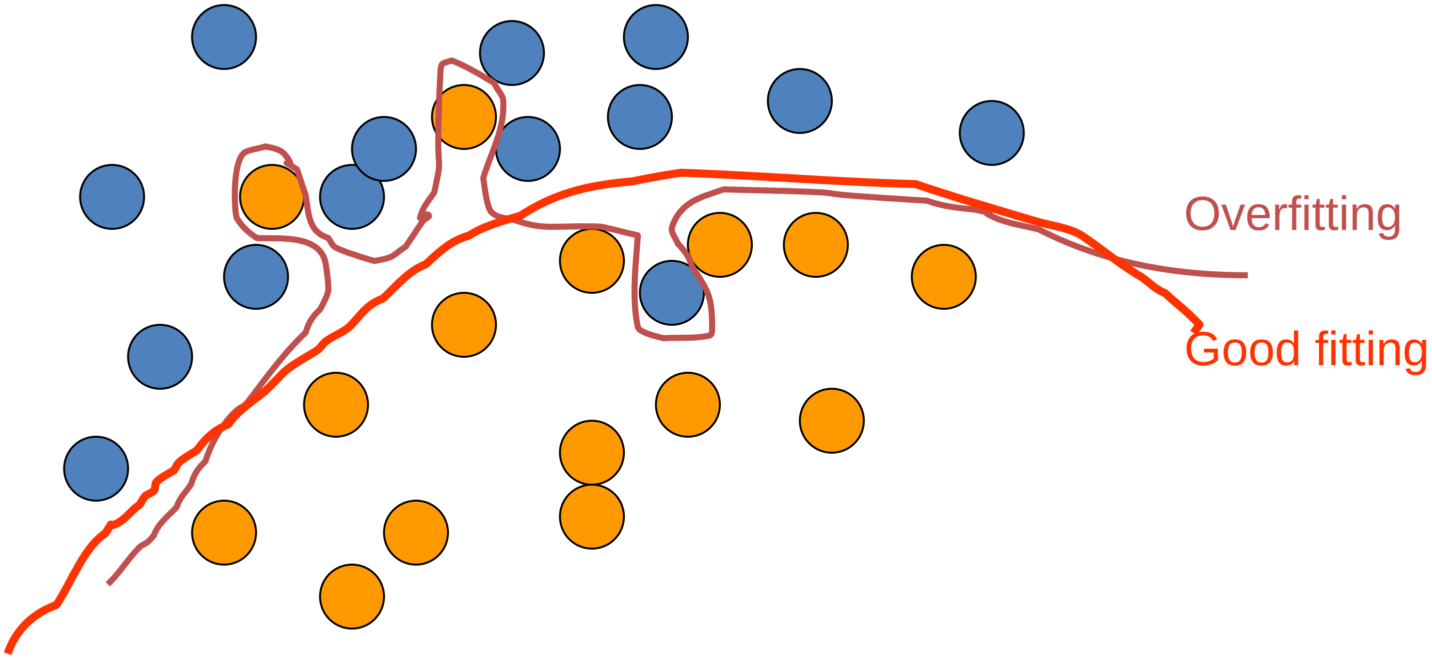
Forward:
At time 1: present $x_1, 0$
At time 2: present x_2, y_1
……

Backward:
Time t: back-propagate
Time $t-1$: back-propagate with
Output errors and errors from previous step

Recurrent Neural Network

1. Recurrent network is essentially a series of feed-forward neural networks sharing the same weights
2. Recurrent network is good for time series data and sequence data such as biological sequences and stock series

Overfitting and Good Fitting



- very close modeling of training data with accidental regularities caused by sampling
- Overfitting function can not generalize well to unseen data.

Preventing Overfitting

- Use a model that has the right capacity:
  - enough to model the true regularities
  - not enough to also model the spurious regularities (assuming they are weaker).
- Standard ways to limit the capacity of a neural net:
  - Limit the number of hidden units.
  - Limit the size of the weights – weigh decay
  - Stop the learning before it has time to overfit
    - Divide the total dataset into three subsets:
      1. Training data – for learning the parameters of the model.
      2. Validation data – for deciding what type of model and what amount of regularization works best.
      3. Test data – to estimate of how well the network works. We expect this estimate to be worse than on the validation data.

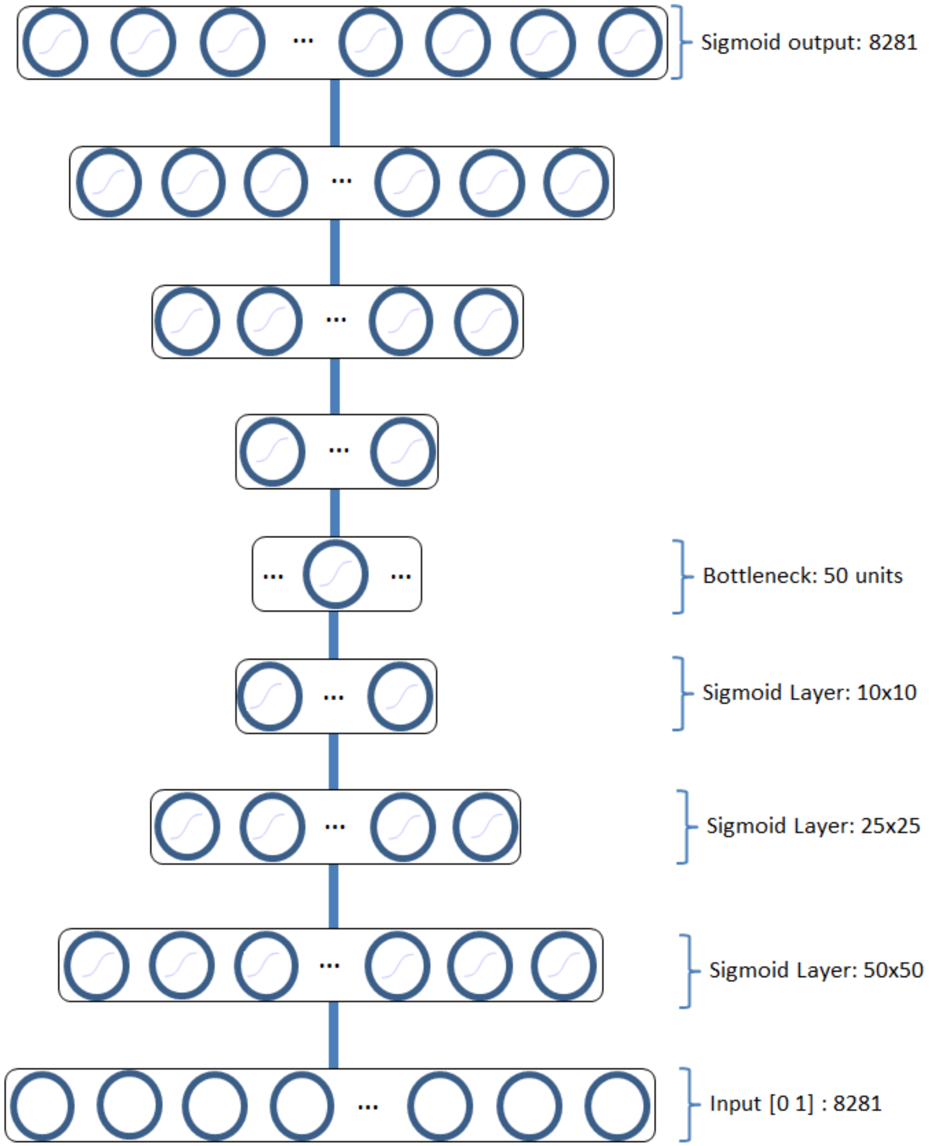
Four Ways to Speed up Learning

1. Use an adaptive global learning rate
   - Increase the rate slowly if its not diverging
   - Decrease the rate quickly if it starts diverging
2. Use separate adaptive learning rate on each connection
   - Adjust using consistency of gradient on that weight axis
3. Use momentum
   - Instead of using the gradient to change the position of the weight "particle", use it to change the velocity.
4. Use a stochastic estimate of the gradient from a few cases
   - This works very well on large, redundant datasets.

Problems of Neural Networks

- Vanishing gradients
- Cannot use unlabeled data
- Hard to understand the relationship between input and output
- Cannot generate data

Deep AutoEncoder



Deep Convolution Neural Network