Supervised adapting (learning) neural network

Computational Intelligence

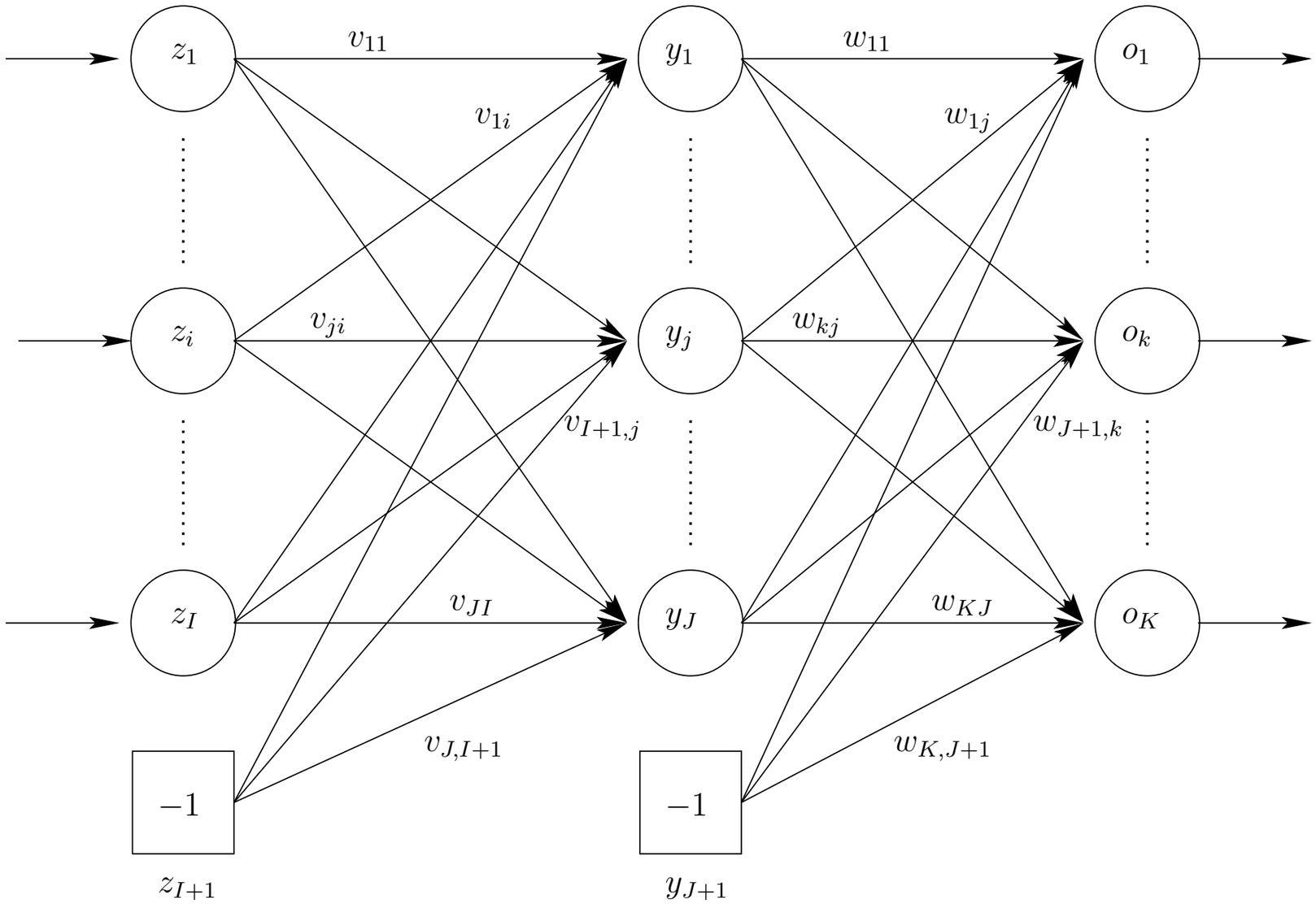
Supervised adapting (learning) neural networks

- Supervised learning requires a training set.
- The NN learner uses the target vector to determine how well it has adapted, and to guide adjustments to weight values to reduce its overall error.
- There are several NN types that adapt under supervision.
  - These network types include standard multilayer NNs,
  - functional link NNs,
  - simple recurrent NNs,
  - time-delay NNs, product unit NNs, and
  - cascade networks.

Feedforward neural networks (FFNN)

- It has been proved that FFNNs with monotonically increasing differentiable functions can approximate any continuous function with one hidden layer, provided that the hidden layer has enough hidden neurons (Gudise and Venayagamoorthy, 2003)

FFNN


z_1
v_{11}
v_{1i}
z_i
v_{ji}
$v_{I+1,j}$
z_I
v_{JI}
$v_{J,I+1}$
-1
z_{I+1}
y_1
w_{11}
w_{1j}
y_j
w_{kj}
$w_{J+1,k}$
y_J
w_{KJ}
$w_{K,J+1}$
-1
y_{J+1}
o_1
o_k
o_K

The output of a FFNN

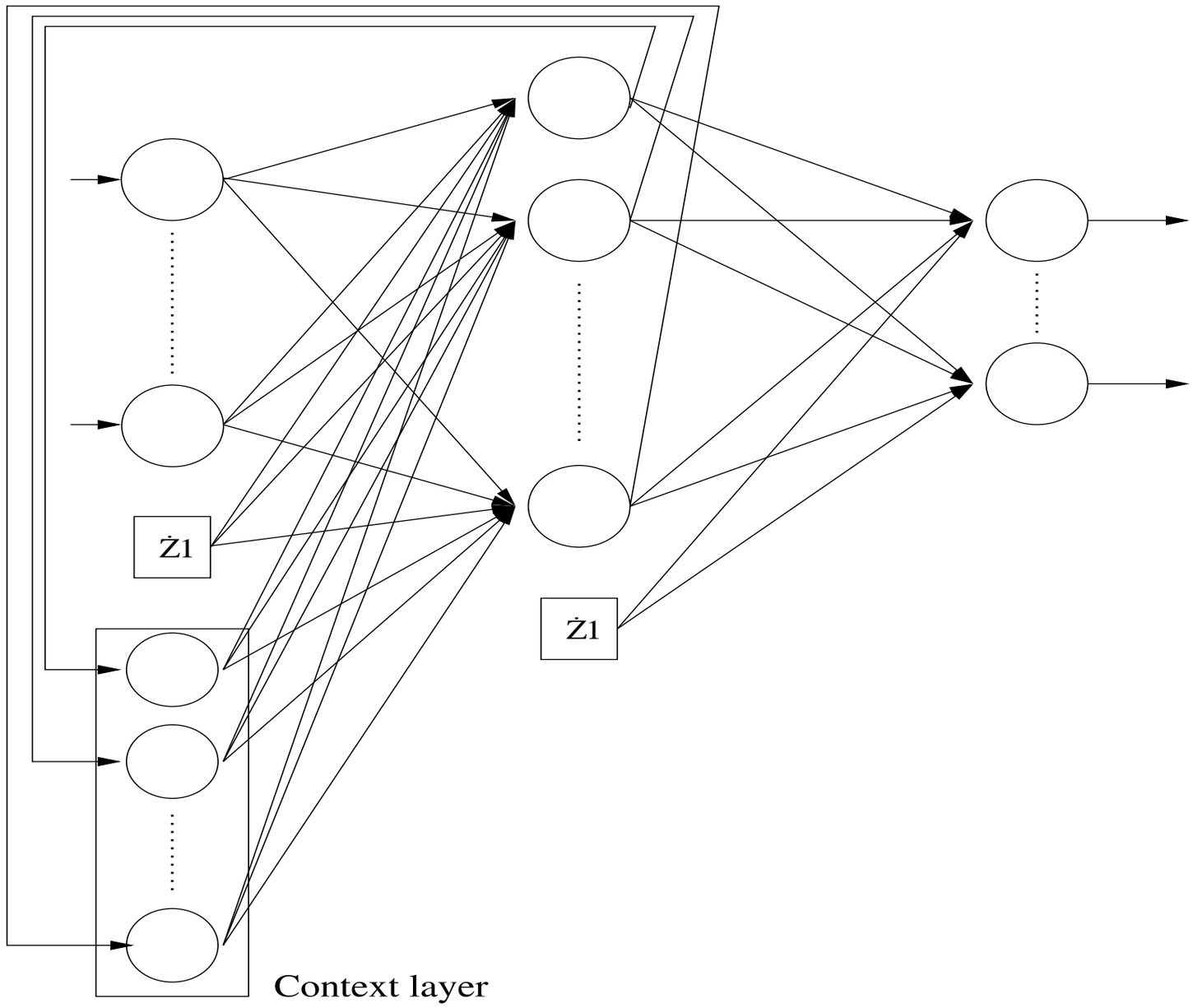
$$
\begin{aligned}
o_{k,p} &= f_{o_k}(net_{o_{k,p}}) \\
&= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}(net_{y_{j,p}})\right) \\
&= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I+1} v_{ji} z_{i,p}\right)\right)
\end{aligned}
$$

where f_{o_k} and f_{y_j} are respectively the activation function for output unit o_k and hidden unit y_j; w_{kj} is the weight between output unit o_k and hidden unit y_j; $z_{i,p}$ is the value of input unit z_i of input pattern $\mathbf{z}_p$; the $(I+1)$-th input unit and the $(J+1)$-th hidden unit are bias units representing the threshold values of neurons in the next layer.

Product unit neural networks (PUNN)

$$
\begin{aligned}
net_{y_{j,p}} &= \prod_{i=1}^{I} z_{i,p}^{v_{ji}} \\
&= \prod_{i=1}^{I} e^{v_{ji} \ln(z_{i,p})} \\
&= e^{\sum_i v_{ji} \ln(z_{i,p})}
\end{aligned}
$$

Simple recurrent neural network



SRNN

Simple recurrent neural networks (SRNN) have feedback connections which add the ability to also learn the temporal characteristics of the data set.

It is possible to have weights not equal to 1, in which case the influence of previous states is weighted. Determining such weights adds additional complexity to the training step.

Each output unit's activation is then calculated as

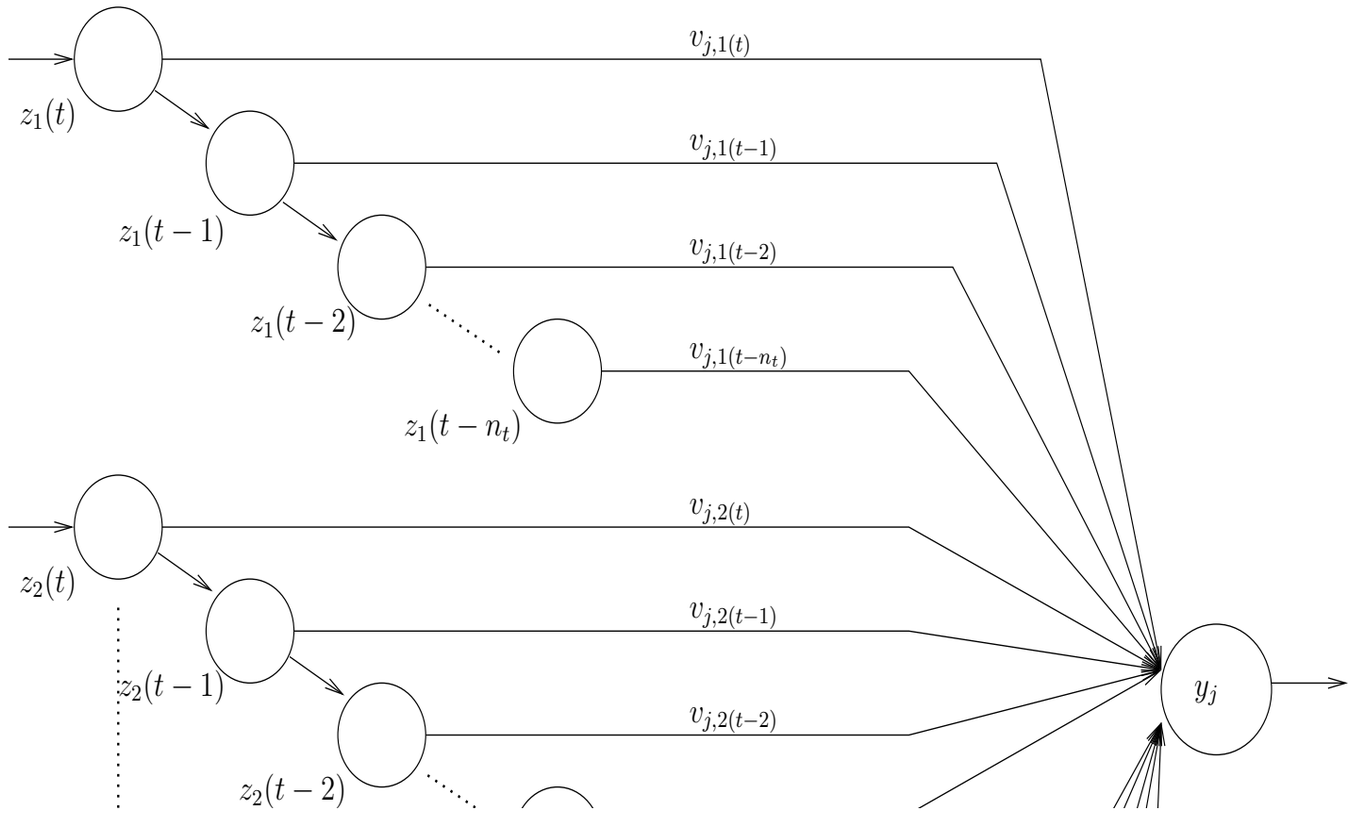
$$o_{k,p} = f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I+1+J} v_{ji} z_{i,p}\right)\right)$$

where $(z_{I+2,p}, \cdots, z_{I+1+J,p}) = (y_{1,p}(t-1), \cdots, y_{J,p}(t-1))$.

Time delay neural network (TDNN)

The output of a TDNN is calculated as

$$o_{k,p} = f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I}\sum_{t=0}^{n_t} v_{j,i(t)} z_{i,p}(t) + z_{I+1} v_{j,I+1}\right)\right)$$



Cascade neural network (CNN)

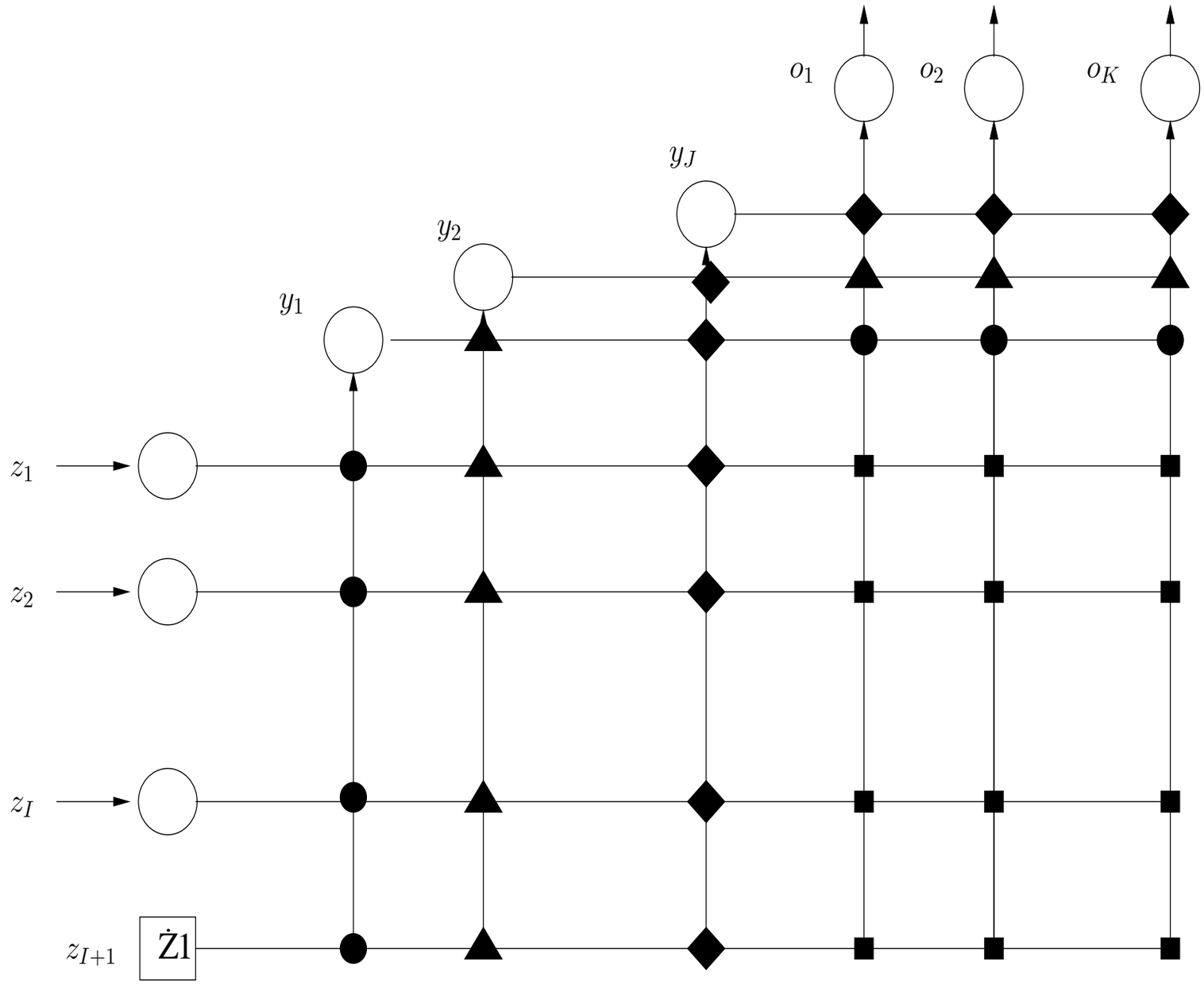
- It is a multilayer FFNN where all input units have direct connections to all hidden units and to all output units. Furthermore, the hidden units are cascaded. That is, each hidden unit's output serves as an input to all succeeding hidden units and all output units.

The output of a CNN is calculated is

$$o_{k,p} = f_{o_k}\left(\sum_{i=1}^{I+1} u_{ki} z_i + \sum_{j=1}^{J} w_{kj} f_{y_j}\left(\sum_{i=1}^{I+1} v_{ji} z_i + \sum_{l=1}^{j-1} s_{jl} y_l\right)\right)$$

where u_{ki} represents a weight between output unit k and input unit i, s_{jl} is a weight between hidden units j and l, and y_l is the activation of hidden unit l.

Cascade neural network



Supervised Learning

- During learning, the function $f_{NN} : \mathbb{R}^I \longrightarrow \mathbb{R}^K$ is found which minimizes the empirical error

$$\mathcal{E}_T(D_T; \mathbf{W}) = \frac{1}{P_T} \sum_{p=1}^{P_T} (F_{NN}(\mathbf{z}_p, \mathbf{W}) - \mathbf{t}_p)^2$$

Optimization algorithms for training NNs

- Optimization algorithms are grouped into two classes:
  - Local optimization, where the algorithm may get stuck in a local optimum without finding a global optimum. Gradient descent and scaled conjugate gradient are examples of local optimizers.
  - Global optimization, where the algorithm searches for the global optimum by employing mechanisms to search larger parts of the search space.
- Local and global optimization techniques can be combined to form hybrid training algorithms.

Supervised learning methods

- Learning consists of adjusting weights until an acceptable empirical error has been reached. Two types of supervised learning algorithms exist, based on when weights are updated:
  - Stochastic/online learning, where weights are adjusted after each pattern presentation. In this case the next input pattern is selected randomly from the training set, to prevent any bias that may occur due to the order in which patterns occur in the training set.
  - Batch/offline learning, where weight changes are accumulated and used to adjust weights only after all training patterns have been presented.

Gradient descent algorithm

- Gradient descent (GD) optimization has led to one of the most popular learning algorithms, namely backpropagation, popularized by Werbos. Learning iterations (one learning iteration is referred to as an epoch) consists of two phases:

1. Feedforward pass, which simply calculates the output value(s) of the NN for each training pattern.
2. Backward propagation, which propagates an error signal back from the output layer toward the input layer. Weights are adjusted as functions of the backpropagated error signal.

Updating of the weights

$$
\begin{aligned}
w_{kj}(t) &+= \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1) \\
v_{ji}(t) &+= \Delta v_{ji}(t) + \alpha \Delta v_{ji}(t-1)
\end{aligned}
$$

$$
\begin{aligned}
\Delta w_{kj} &= \eta \left(-\frac{\partial E}{\partial w_{kj}} \right) & \Delta v_{ji} &= \eta \left(-\frac{\partial E}{\partial v_{ji}} \right) \\
&= -\eta \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial w_{kj}} & &= -\eta \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial v_{ji}} \\
&= -\eta \delta_{o_k} y_j & &= -\eta \delta_{y_j} z_i
\end{aligned}
$$

$$
\begin{aligned}
\delta_{o_k} &= -(t_k - o_k)(1 - o_k) o_k = -(t_k - o_k) f_{o_k}^{'} \\
\delta_{y_j} &= \sum_{k=1}^{K} \delta_{o_k} w_{kj} f_{y_j}^{'}
\end{aligned}
$$

- Weight change = constant x error x input activation
- For an output node the error is
  - error = (target - output) x output x (1- output)
- For a hidden node, the error is
  - error = weighted sum of to-node errors x hidden x 1- hidden)

Weight change and momentum

- Backpropagation algorithm often takes a long time to learn
- So, the learning rule is often augmented with a so called momentum term
- This consist in adding a fraction of the old weight change
- The learning rule then looks like
- weight change = constant x error per input + momentum constant x old weight change

Stochastic Gradient Descent Learning Algorithm

Initialize weights, η, α, and the number of epochs $t = 0$;
while *stopping condition(s) not true* **do**
    Let $\mathcal{E}_T = 0$;
    for *each training pattern* p **do**
        Do the feedforward phase to calculate $y_{j,p}$ $(\forall\ \ j = 1, \cdots, J)$ and $o_{k,p}$ $(\forall\ \ k = 1, \cdots, K)$;
        Compute output error signals $\delta_{o_{k,p}}$ and hidden layer error signals $\delta_{y_{j,p}}$;
        Adjust weights w_{kj} and v_{ji} (backpropagation of errors);
        $\mathcal{E}_T += [\mathcal{E}_p = \sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2]$;
    end
    $t = t + 1$;
end

Fletcher-Reeves Conjugate gradient algorithm

Initialize the weight vector, $\mathbf{w}(0)$;
Calculate the gradient, $\mathcal{E}^{'}(\mathbf{w}(0))$;
Compute the first direction vector as $\mathbf{p}(0) = -\mathcal{E}^{'}(\mathbf{w}(0))$;
while *stopping conditions(s) not true* **do**
  for $t = 0, \ldots, n_w - 1$ **do**
    Calculate the step size,

$$\eta(t) = \min_{\eta \geq 0} \mathcal{E}(\mathbf{w}(t) + \eta \mathbf{p}(t))$$

Calculate a new weight vector,

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(t)\mathbf{p}(t)$$

Calculate scale factors,

$$\beta(t) = \frac{\mathcal{E}^{'}(\mathbf{w}(t+1))^T \mathcal{E}^{'}(\mathbf{w}(t+1)}{\mathcal{E}^{'}(\mathbf{w}(t))^T \mathcal{E}^{'}(\mathbf{w}(t))}$$

Calculate a new direction vector,

$$\mathbf{p}(t+1) = -\mathcal{E}^{'}(\mathbf{w}(t+1)) + \beta(t)\mathbf{p}(t)$$

  end
  if *stopping condition(s) not true* **then**

$$\mathbf{w}(0) = \mathbf{w}(n_w)$$

  end
end
Return $\mathbf{w}(n_w)$ as the solution;

Stopping criteria usually includes:

- Stop when a maximum number of epochs has been exceeded.
- Stop when the mean squared error (MSE) on the training set, is small enough.

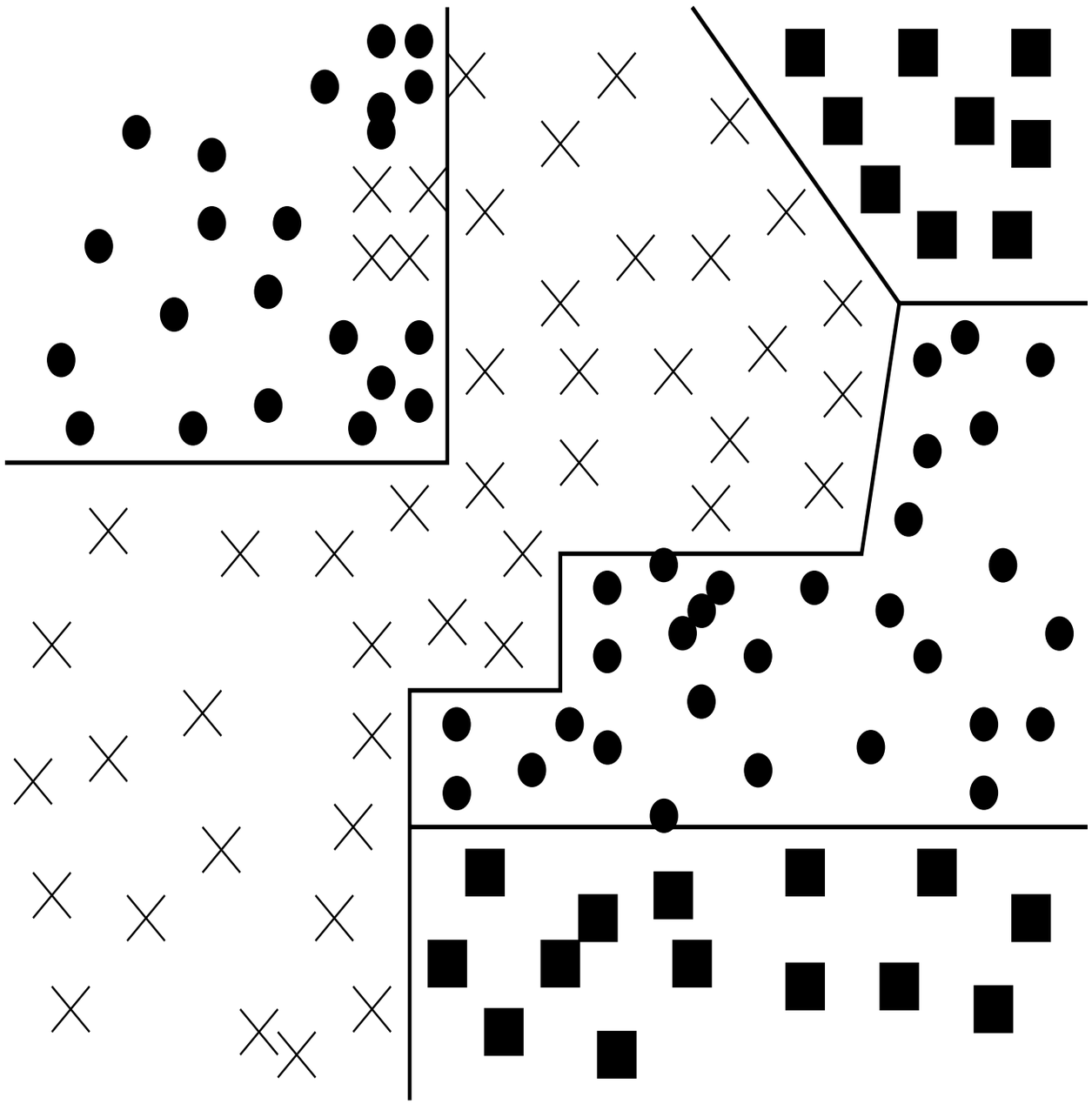
$$\mathcal{E}_T = \frac{\sum_{p=1}^{P_T} \sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2}{P_T K}$$

- Stop when overfitting is observed, i.e. when training data is being memorized. An indication of overfitting is when *EV* > *EV* + σ*EV* , where *EV* is the average validation error over the previous epochs, and σEV is the standard deviation in validation error.

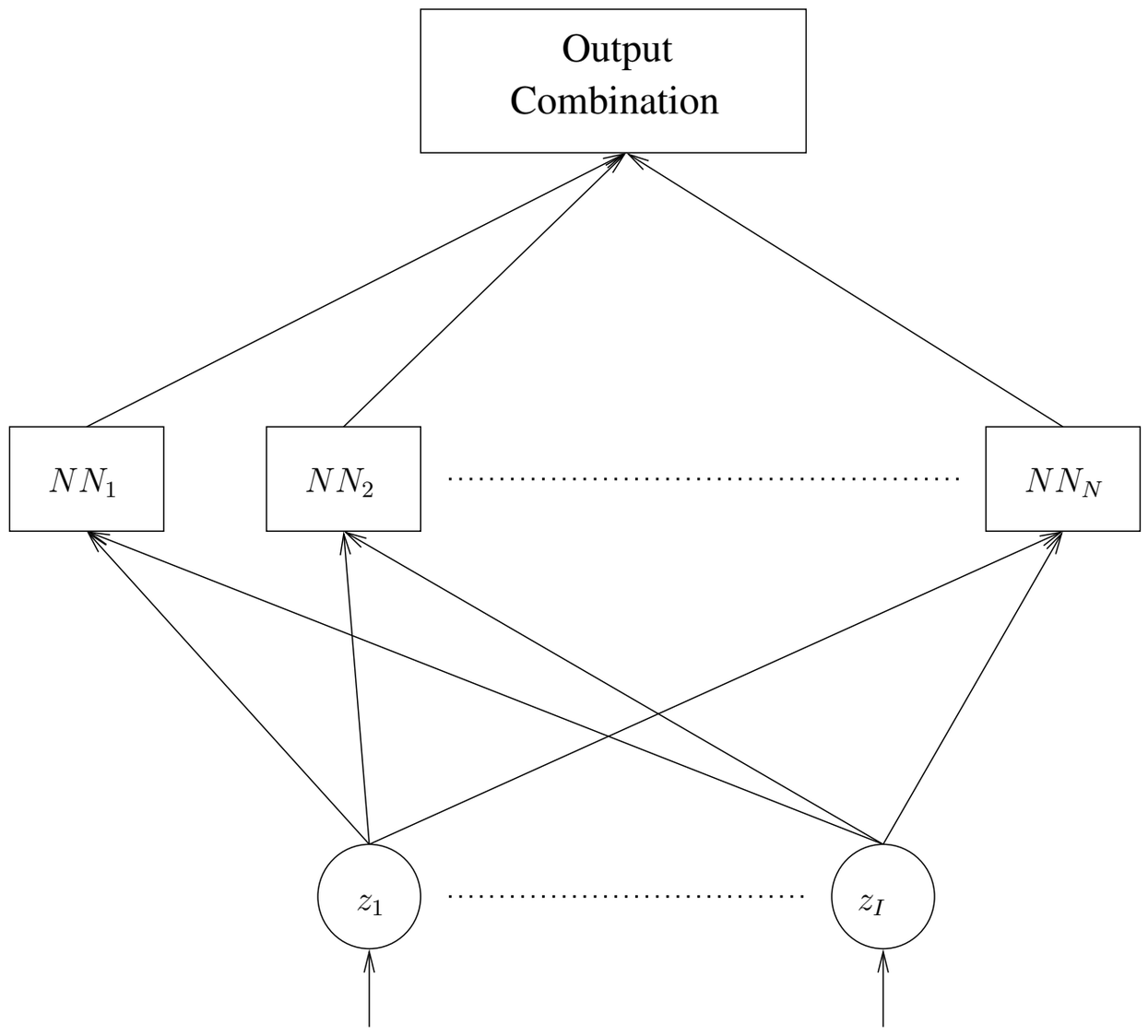
Global CI training methods

- PSO
- DE
- GA

Feedforward Neural Network Classification Boundary Illustration



Ensemble neural networks



- http://www.rueckstiess.net/snippets/show/17a86039