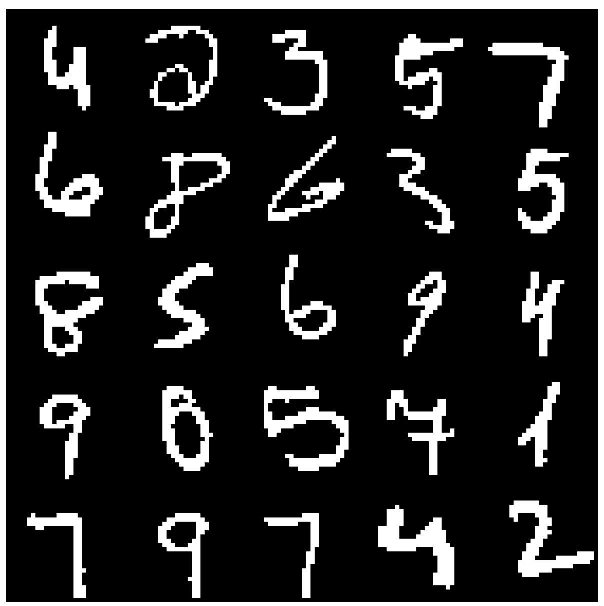
Gradient-based deep learning

gradient descent and batch updating

```
function err_g=gradient_check2(obj,x,y)
    %Calculate gradient of output with respect to w by Richardson
    %Extrapolation by flow chart 4.
    %x conbtains batch data

    L=obj.layers;
    M=size(obj.w{L-1},2);
    z=0.01;
    err_g=0;
       for k=1:L-1
          W_k=obj.w{k};
          RE_gW{k} = zeros(size(W_k));
           for i=1:size(W_k,1)
             for j=1:size(W_k,2)
               %calculate f1 f2 f3 f4
                obj.w{k}(i,j)=W_k(i,j)+z;
                obj =obj.ff(x);obj=obj.cal_se(y);f1=obj.se;
                obj.w{k}(i,j)=W_k(i,j)-z;
                obj =obj.ff(x);obj=obj.cal_se(y);f2=obj.se;
                obj.w{k}(i,j)=W_k(i,j)+z/2;
                obj=obj.ff(x);obj=obj.cal_se(y);f3=obj.se;
                obj.w{k}(i,j)=W_k(i,j)-z/2;
                obj=obj.ff(x);obj=obj.cal_se(y);f4=obj.se;
                g1=(f1-f2)/(2*z);g2=(f3-f4)/z;
                RE_gW{k}(i,j)=g2+(g2-g1)/3;
                obj.w{k}(i,j)=W_k(i,j);
             end
          end

          err_g=err_g+sum(sum(abs(obj.E_gW{k}-RE_gW{k}')));
    end
 end % gradient check 2
```

```
function show_digits(J,imax,jmax)
I3=[];
for i=1:imax
    I2=[];
    for j=1:jmax
        I=J((i-1)*jmax+j,:);
        I=(I-min(I));
        I=I/max(abs(I))*255;
        I2=[I2 reshape(I',28,28)];
    end
    I3=[I3 ; I2];
end
imshow(I3')
```



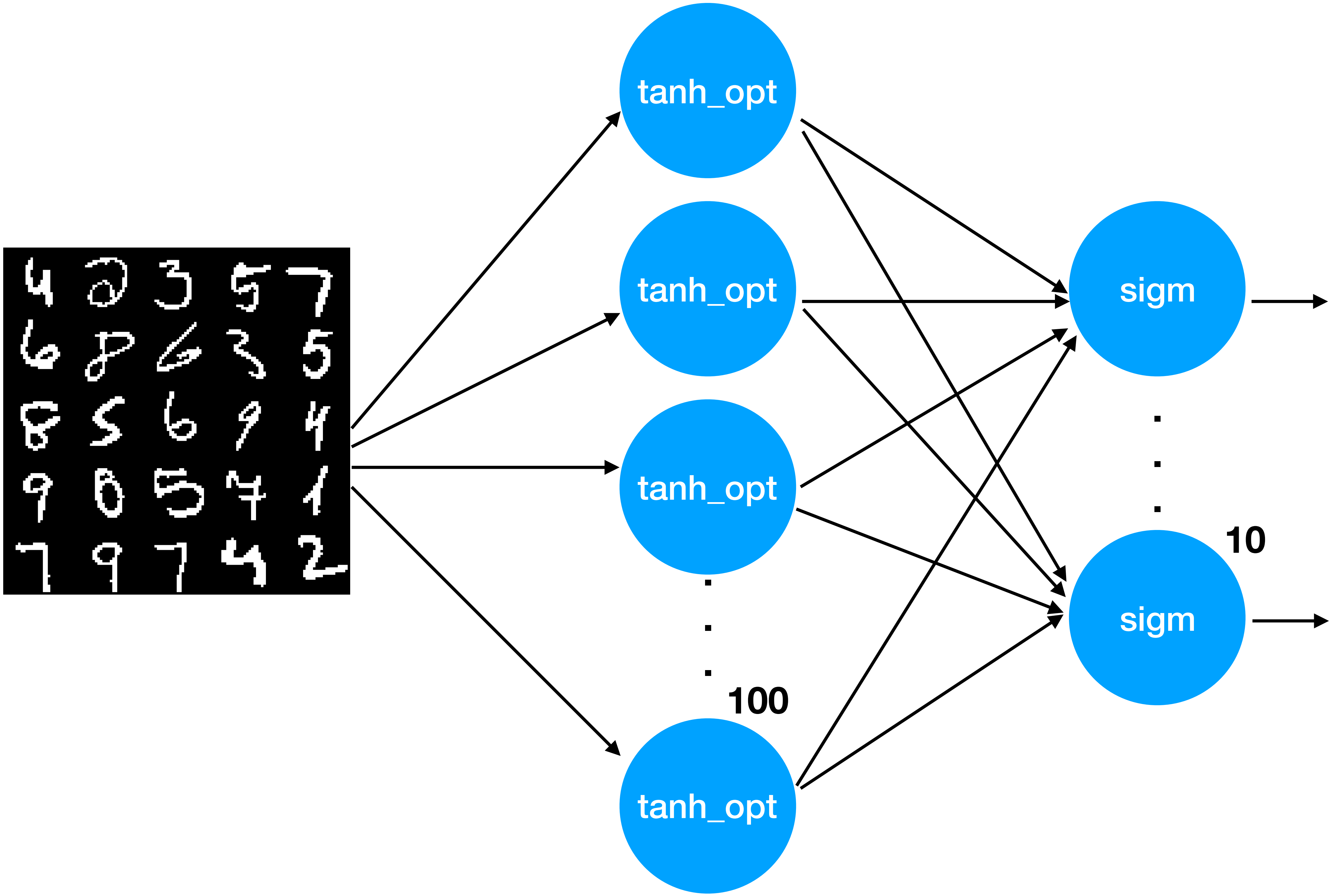
```
>> load mnist_uint8;
>> show_digits(train_x(1:25,:),5,5)
```

Example

```
load mnist_uint8;

train_x = double(train_x) / 255;
test_x  = double(test_x)  / 255;
train_y = double(train_y);
test_y  = double(test_y);

% normalize
[train_x, mu, sigma] = zscore(train_x);
test_x = normalize(test_x, mu, sigma);

%% ex1 vanilla neural net
rand('state',0)
nn = nnsetup([784 100 10]);
opts.numepochs =  100;   %  Number of full sweeps through data
opts.batchsize = 100;  %  Take a mean gradient step over this many samples
[nn, L] = nntrain(nn, train_x, train_y, opts);

[er, bad] = nntest(nn, test_x, test_y);

assert(er < 0.08, 'Too big error');
```

preprocess

training & testing




```
epoch 94/100. Took 2.1078 seconds. Mini-batch mean squared error on training set is 0.0084362; Full-batch train err = 0.008431
epoch 95/100. Took 2.1063 seconds. Mini-batch mean squared error on training set is 0.0084341; Full-batch train err = 0.008428
epoch 96/100. Took 2.2176 seconds. Mini-batch mean squared error on training set is 0.0084317; Full-batch train err = 0.008419
epoch 97/100. Took 1.9804 seconds. Mini-batch mean squared error on training set is 0.0084266; Full-batch train err = 0.008414
epoch 98/100. Took 1.8918 seconds. Mini-batch mean squared error on training set is 0.0084164; Full-batch train err = 0.008402
epoch 99/100. Took 1.7038 seconds. Mini-batch mean squared error on training set is 0.0084094; Full-batch train err = 0.008402
epoch 100/100. Took 1.897 seconds. Mini-batch mean squared error on training set is 0.0084034; Full-batch train err = 0.008397
K>> er
```

modules

- Set up: specify architecture of a neural network and how to train weight matrices
- Train: batch updating, feedforward translation and back-propagation of gradients of outputs with respect to neural stimuli and activations.
- Test: verify effectiveness of a neural network subject to testing data

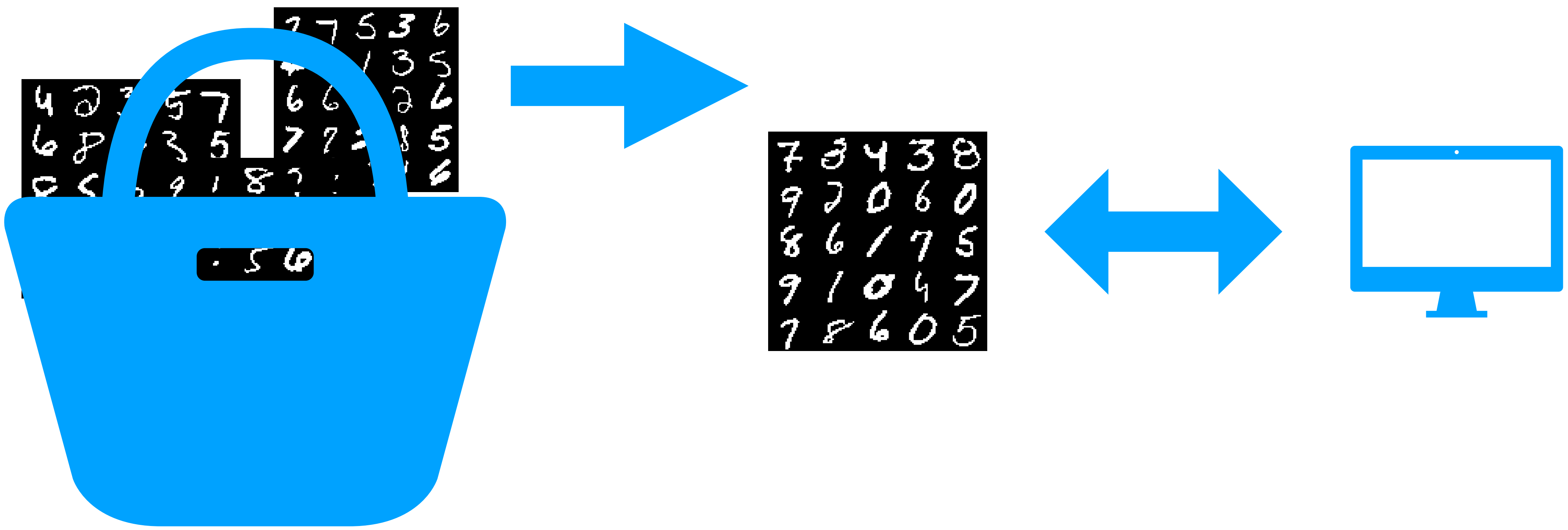
```
function nn = nnsetup(architecture)
%NNSETUP creates a Feedforward Backpropagate Neural Network
% nn = nnsetup(architecture) returns an neural network structure with n=numel(architecture)
% layers, architecture being a n x 1 vector of layer sizes e.g. [784 100 10]

    nn.size   = architecture;
    nn.n      = numel(nn.size);

    nn.activation_function              = 'tanh_opt';   %  Activation functions of hidden layers: 'sigm' (sigmoid) or 'tanh_opt' (optimal tanh).
    nn.learningRate                     = 2;            %  learning rate Note: typically needs to be lower when using 'sigm' activation function and non-
normalized inputs.
    nn.momentum                         = 0.5;          %  Momentum
    nn.scaling_learningRate             = 1;            %  Scaling factor for the learning rate (each epoch)
    nn.weightPenaltyL2                  = 0;            %  L2 regularization
    nn.nonSparsityPenalty               = 0;            %  Non sparsity penalty
    nn.sparsityTarget                   = 0.05;         %  Sparsity target
    nn.inputZeroMaskedFraction          = 0;            %  Used for Denoising AutoEncoders
    nn.dropoutFraction                  = 0;            %  Dropout level (http://www.cs.toronto.edu/~hinton/absps/dropout.pdf)
    nn.testing                          = 0;            %  Internal variable. nntest sets this to one.
    nn.output                           = 'sigm';       %  output unit 'sigm' (=logistic), 'softmax' and 'linear'

    for i = 2 : nn.n
        % weights and weight momentum
        nn.W{i - 1} = (rand(nn.size(i), nn.size(i - 1)+1) - 0.5) * 2 * 4 * sqrt(6 / (nn.size(i) + nn.size(i - 1)));
        nn.vW{i - 1} = zeros(size(nn.W{i - 1}));

        % average activations (for use with sparsity)
        nn.p{i}     = zeros(1, nn.size(i));
    end
end
```

batch updating

- Large data size, such as 60000 handwritten digits, 1000,000 color images

- Random partition train_x to many batches

- Training data in a batch are employed to calculate gradients of square errors with respect to weight matrices



nntrain

```
for i = 1 : numepochs
    tic;

    kk = randperm(m);
    for l = 1 : numbatches
        batch_x = train_x(kk((l - 1) * batchsize + 1 : l * batchsize), :);

        %Add noise to input (for use in denoising autoencoder)
        if(nn.inputZeroMaskedFraction ~= 0)
            batch_x = batch_x.*(rand(size(batch_x))>nn.inputZeroMaskedFraction);
        end

        batch_y = train_y(kk((l - 1) * batchsize + 1 : l * batchsize), :);

        nn = nnff(nn, batch_x, batch_y);
        nn = nnbp(nn);
        nn = nnapplygrads(nn);

        n = n + 1;
    end

    t = toc;

    nn.learningRate = nn.learningRate * nn.scaling_learningRate;
end
end
```

1. From the first layer to the output layer, calculate stimuli, activations and outputs

batch updating

2.a From the output layer to the first layer, determine gradients of mse with respect to stimuli and activations
2b. Determine gradients of mse with respect to weight matrices

Exercise

- Try to complete methods of nn_train and nn_test
- Apply codes based on your class perceptrons to classification of hand-written digits of MNIST dataset