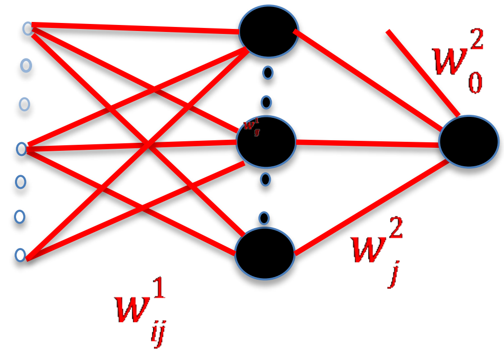
Two-layer Neural Networks

Week 4

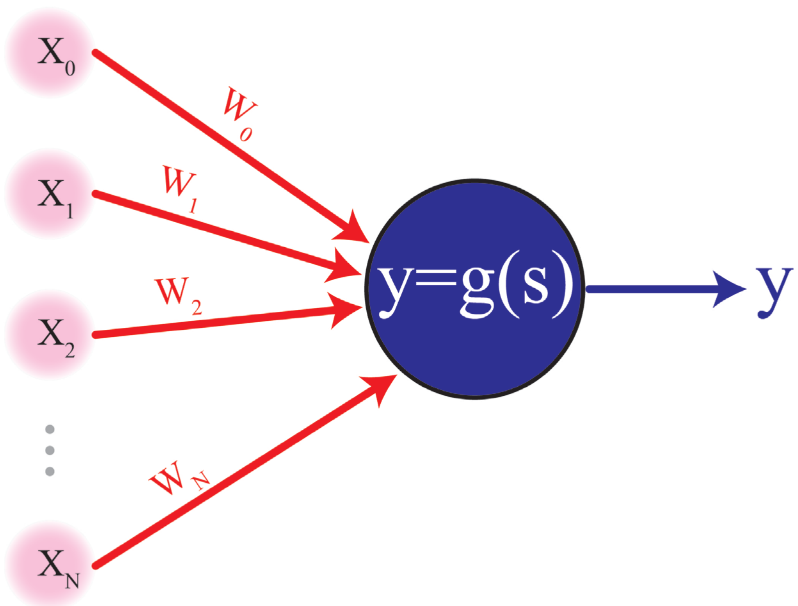
Single output



Outline

- Cost functions: MSE vs cross entropy
- Two layer decision surface
- Error back propagation
- ReLu vs sigmoid
- Cross entropy vs MSE
- Generalization; number of HU’s
- Local minima; stochastic decent
- Transfer learning
- Shift, rotation, scale invariances

One neuron



$$s = \sum_{i}^{N} w_i x_i$$

$$y = g\left(\sum_{i}^{N} w_i x_i\right)$$

$$Labels : y = [0,\ 1]$$

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

The index i determines the input.
N is the total number of the inputs.

One neuron: Mean squared cost function

Mean squared cost function for one neuron is: $C = \sum_{m}^{M} (y^{(m)} - y)^2$

The index m is used for the training samples. M is the total number of the samples.
$y^{(m)}$ denotes the label and y denotes the predicted output by the neuron.
The weight w_i should be reduced in the direction of $-\frac{\partial C}{\partial w_i}$.
$\frac{\partial C}{\partial w_i}$ calculation is as follows:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial s}\frac{\partial s}{\partial w_i}$$

$$s = \sum_{i}^{N} w_i x_i \rightarrow \frac{\partial s}{\partial w_i} = x_i$$

$$y = g(s) = \frac{1}{1+e^{-s}} \rightarrow \frac{\partial y}{\partial s} = \frac{e^{-s}}{1+e^{-s}} = y(1-y)$$

$$C = \sum_{m}^{M} (y^{(m)} - y)^2 \rightarrow \frac{\partial C}{\partial y} = -\sum_{m}^{M} 2(y^{(m)} - y)$$

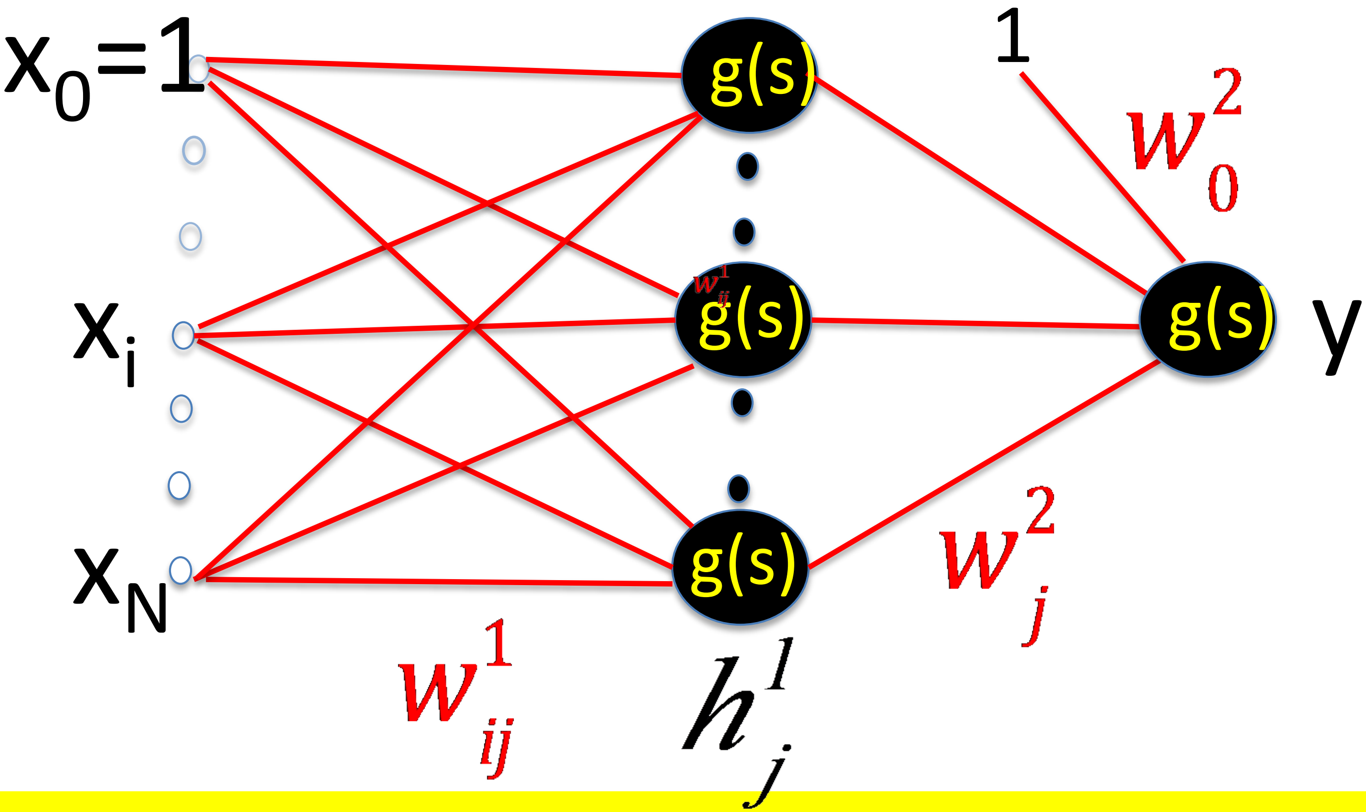
$$\Rightarrow \frac{\partial C}{\partial w_i} = -\sum_{m}^{M} 2(y^{(m)} - y) y (1-y) x_i$$

The weight w_i can be updated using the following equation:

$$w_i^{new} = w_i^{old} - \alpha \frac{\partial C}{\partial w_i}$$

α is the learning rate

Two layer network



$$y = g\left(\sum_{j=0}^{N_1} w_j^2 h_j^1\right) \qquad h_j^1 = g\left(\sum_{i=0}^{N} w_{ij}^1 x_i\right)$$

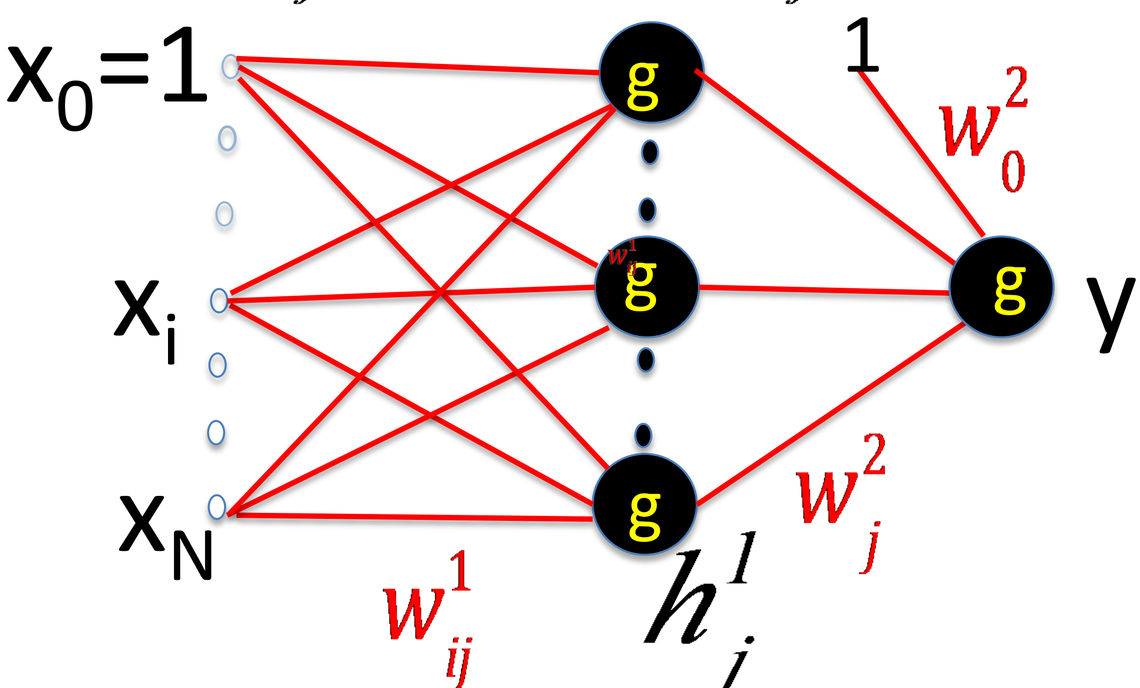
Training set: $\left\{x^{(m)}, y^{(m)}\right\} \quad m = 1, M \quad y^{(m)} = label\,(0\ or\ 1, \pm 1)$

Error Back Propagation

$$C(w_{ij}^1) = \sum_{m}^{M} (y^{(m)} - y)^2$$

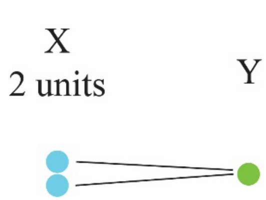
$$\Delta w_{ij}^1 = -\alpha \frac{\partial C(w_{ij}^1)}{\partial w_{ij}^1} = -\alpha \sum_{m=1}^{M} 2(y - y^{(m)}) \frac{\partial y}{\partial w_{ij}^1}$$

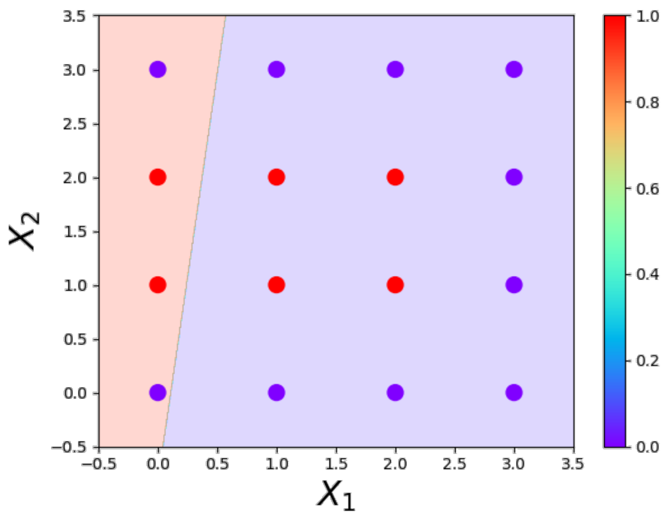
$$\frac{\partial y}{\partial w_{ij}^1} = \frac{\partial g}{\partial s_2} \frac{\partial s_2}{\partial w_{ij}^1} = \frac{\partial g}{\partial s_2} w_j^2 \frac{\partial h_j}{\partial w_{ij}^1} = \frac{\partial g}{\partial s_2} w_j^2 \frac{\partial g}{\partial s_1} x_i^{(m)}$$



Decision boundary

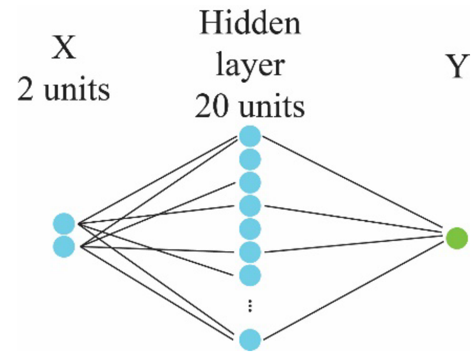
One layer network
ADALINE

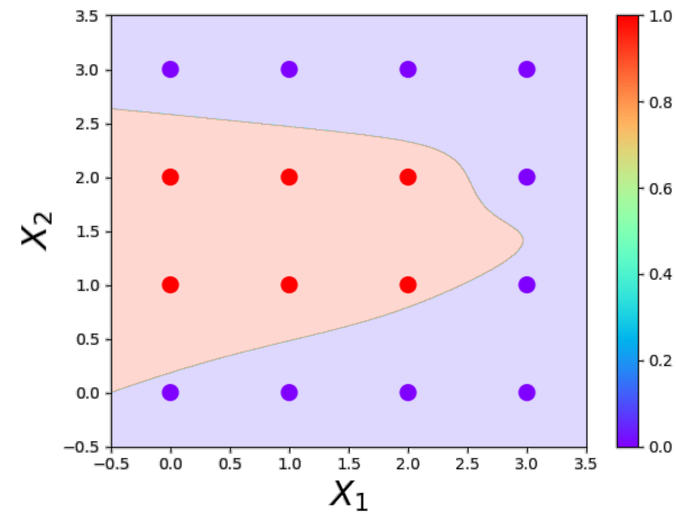




Test accuracy	62.50%

Two layer network



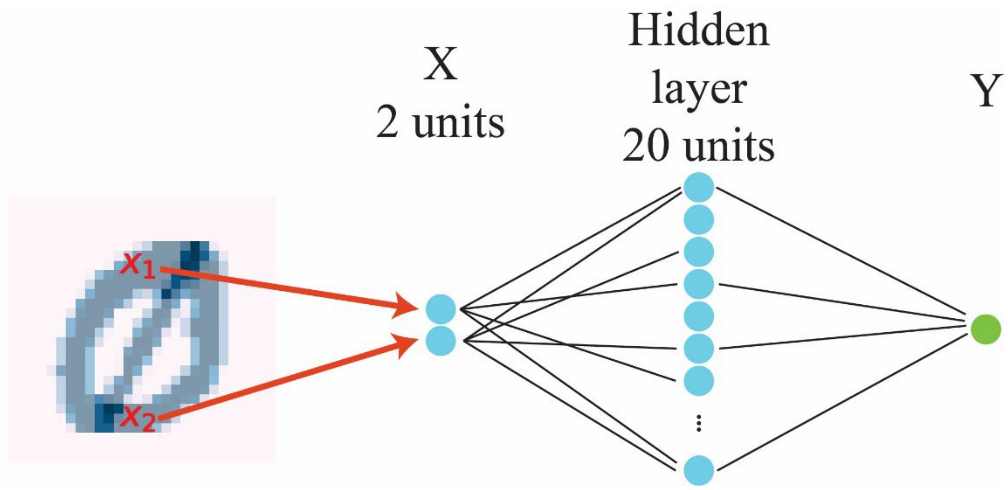


Test accuracy	100%

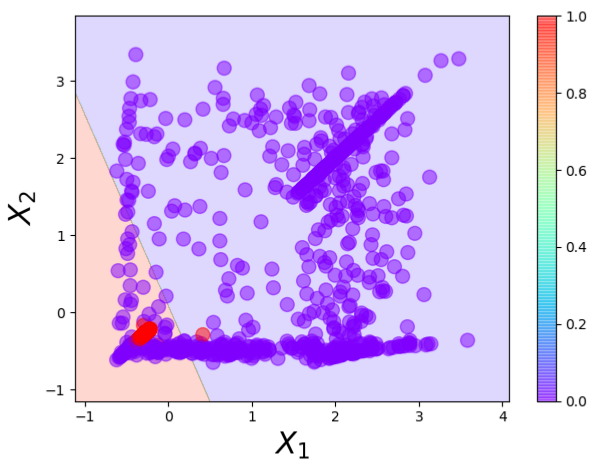
20 neurons in hidden layer

Decision boundary

Classification of 0 and 1 images based on two pixels



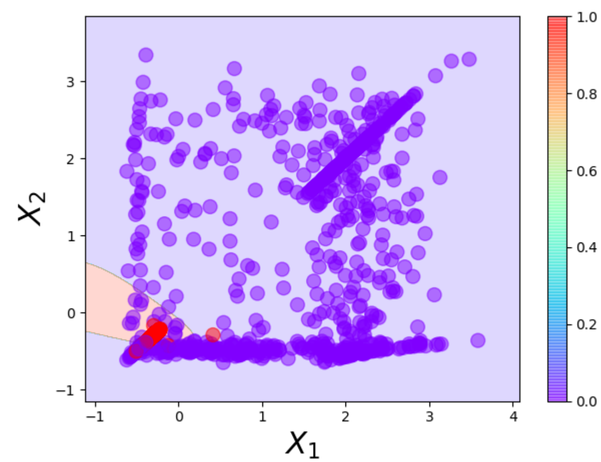
One layer network



Training accuracy	83.08%
Test accuracy	83.55%

12665 training samples
2115 test samples

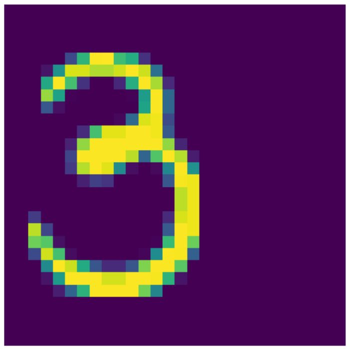
Two layers network

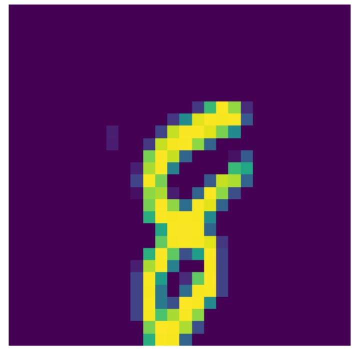


Training accuracy	95.20%
Test accuracy	95.41%

20 neurons in hidden layer

Two layer network: shifted images of 3 and 8





11,982 images of 3 and 8 are shifted by 10 pixels in horizontal and vertical direction
1984 images in the test set

2 layers network has 20 neurons in the hidden layer

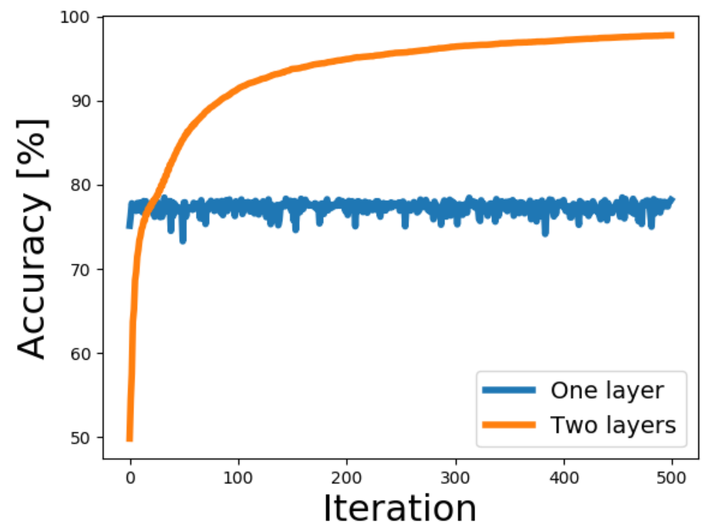
C=Mean square error

One layer network
Adaline

Training accuracy	78.20%
Test accuracy	77.87%

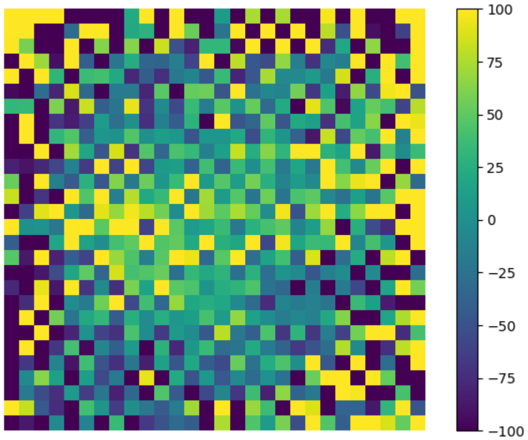
Two layers network

Training accuracy	97.89%
Test accuracy	95.97%

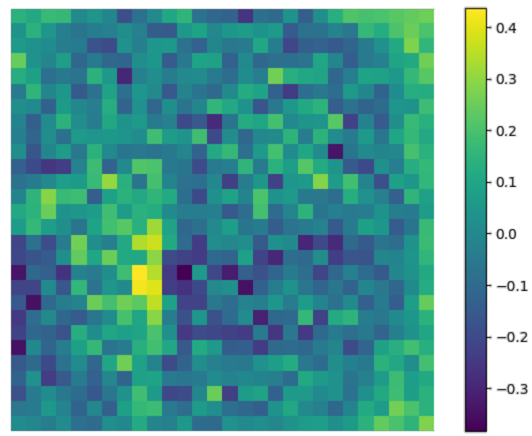


Receptive fields: 3 versus 8-shifted

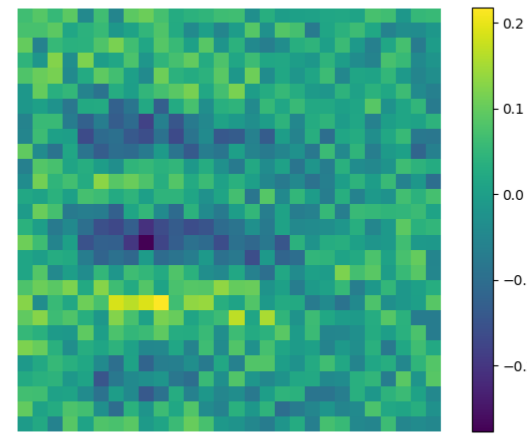
Single layer weights
(receptive field)



Examples receptive fields
First layer of a 2-layer network
2 hidden units

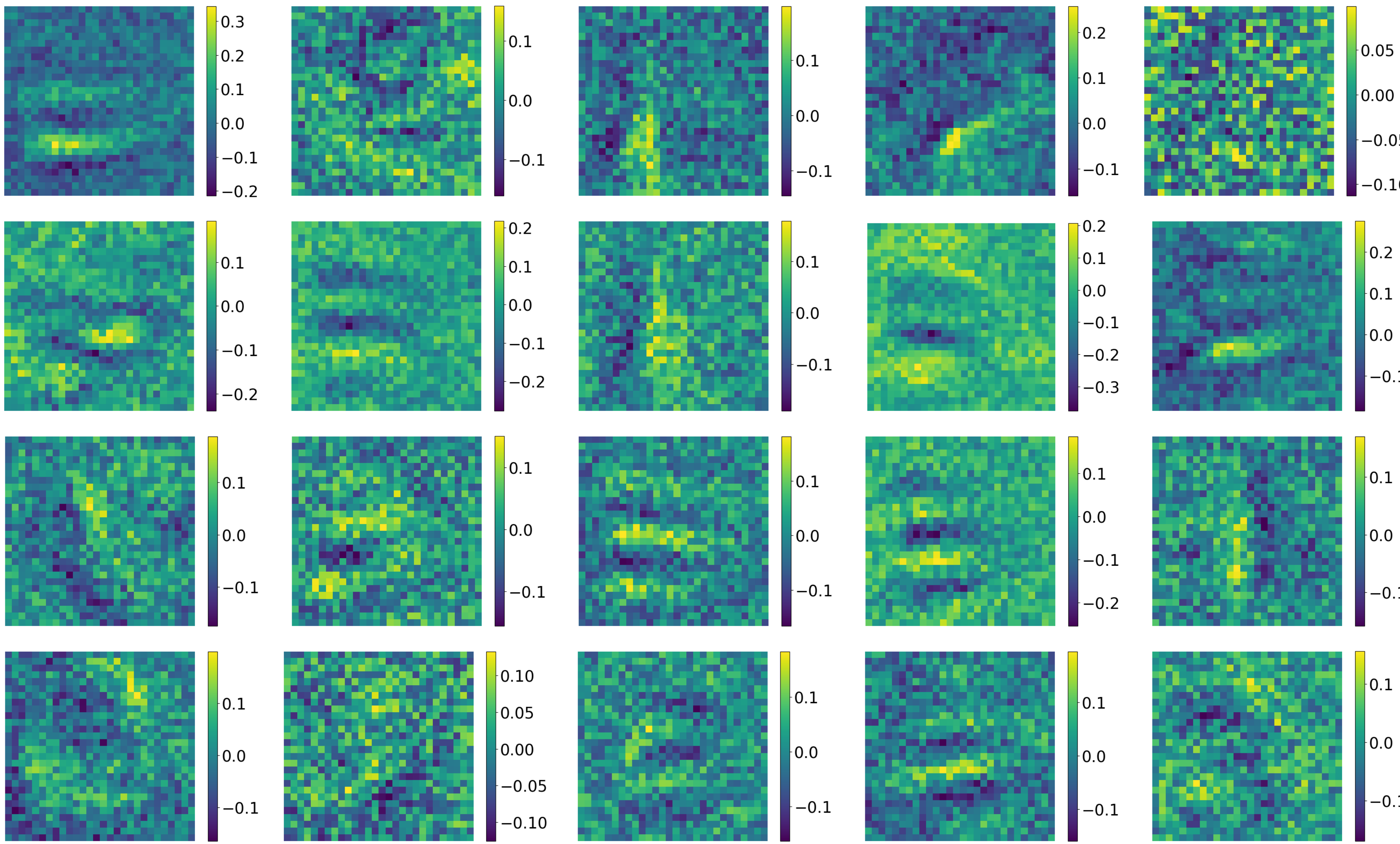


Examples receptive fields
First layer of a 2-layer network
20 hidden units



# of HU's	1	2	20
Training set	78 ->	87 ->	97
Test set	77 ->	81 ->	95

Two layers network: all 20 weights-3 versus 8 shifted



Quiz #4

- Example: Dense layer with 10 hidden units and sigmoid activation usage

```
dense_layer = Dense(10, activation='sigmoid')

output = dens_layer(input)
```

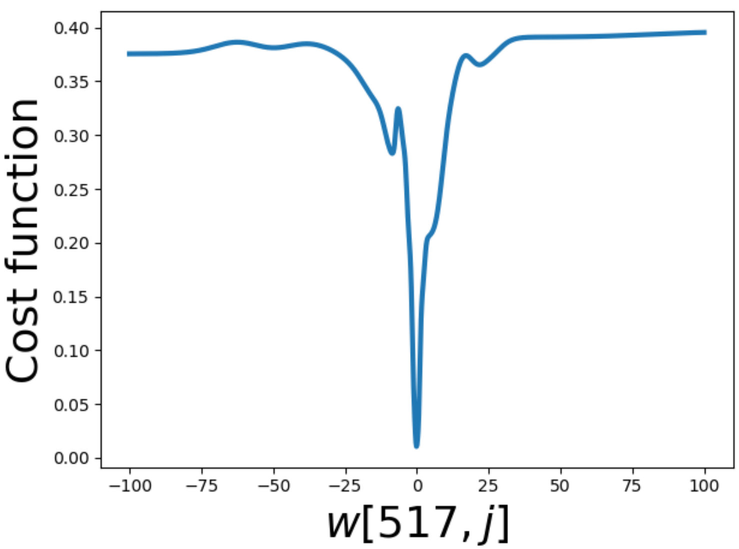
Type “Keras dense” in google

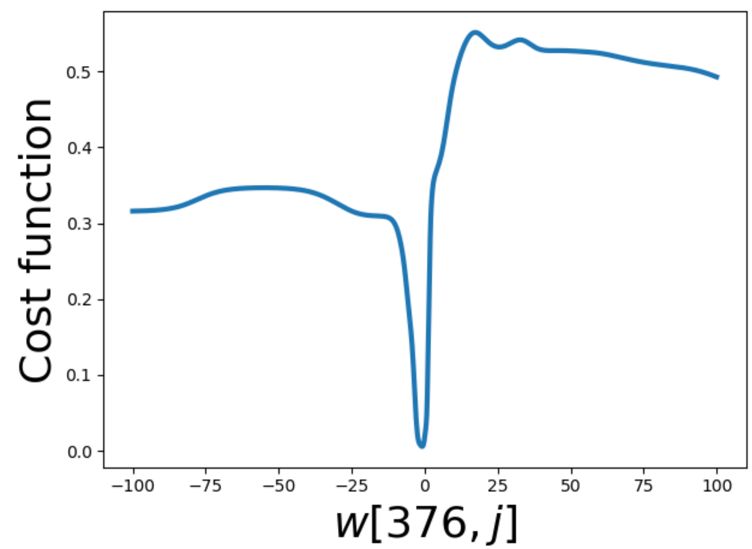
https://keras.io/api/layers/core_layers/dense/

Two layer network: shifted images of 3 and 8

$$C(w) = \sum_{m}^{M} (y^{(m)} - y)^2$$

Trained weights





20 hidden units; 20 weights changed at a time

Cross entropy and Mean squared error

Classification of shifted images of 3 and 8

- 11982 images for training
- 1984 images for test
- One hidden layer with 20 neurons
- Learning rate = 1
- Sigmoid activation function

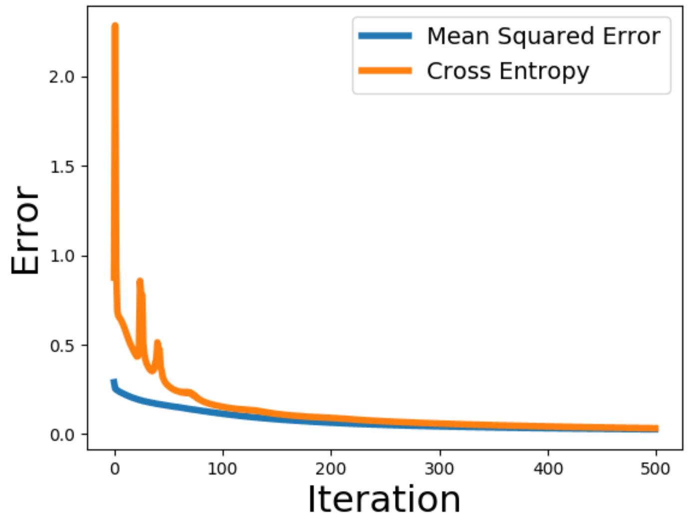
Mean squared error

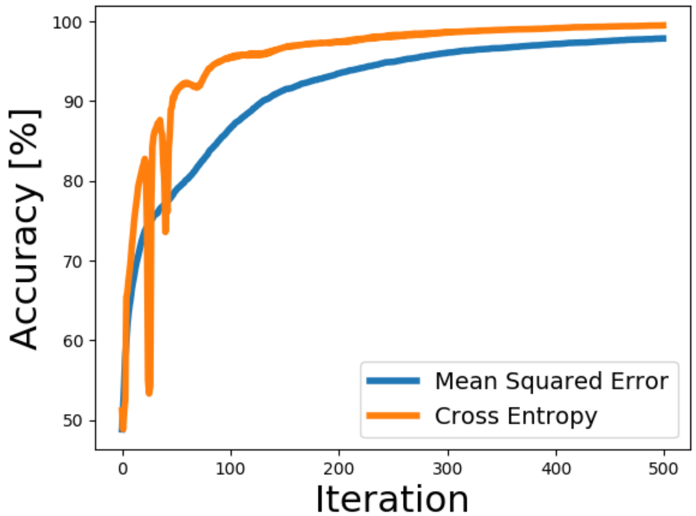
Training accuracy	97.87%
Test accuracy	96.07%

Cross entropy

$$C = -\sum_{m}\left[y^{(m)} \ln(y) + (1-y^{(m)}) \ln(1-y) \right]$$

Training accuracy	99.52%
Test accuracy	96.67%





Batch size

Classification of shifted images of 3 and 8

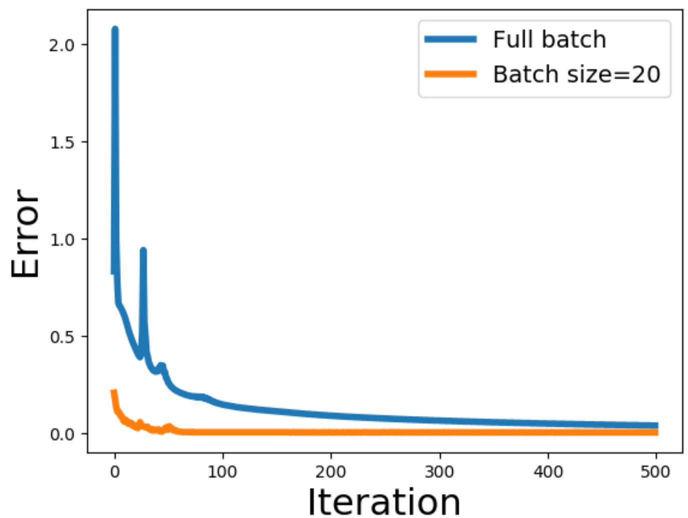
- 11982 images for training
- 1984 images for test
- One hidden layer with 20 neurons
- Learning rate = 1
- Sigmoid activation function
- Cross entropy cost function

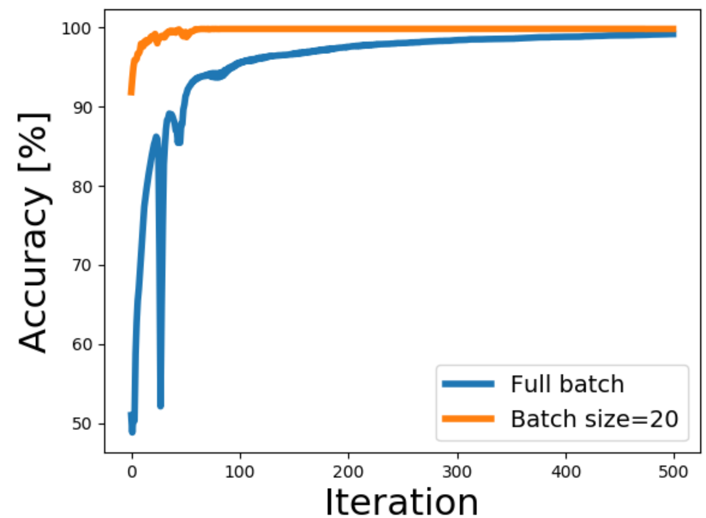
Full batch

- All images are fed to the network simultaneously

Small batch

- At each epoch, images are randomly distributed into batches of 20 images and then batches are fed one by one to the network.





Sigmoid and Rectified linear (ReLu) with MSE

Classification of shifted images of 3 and 8

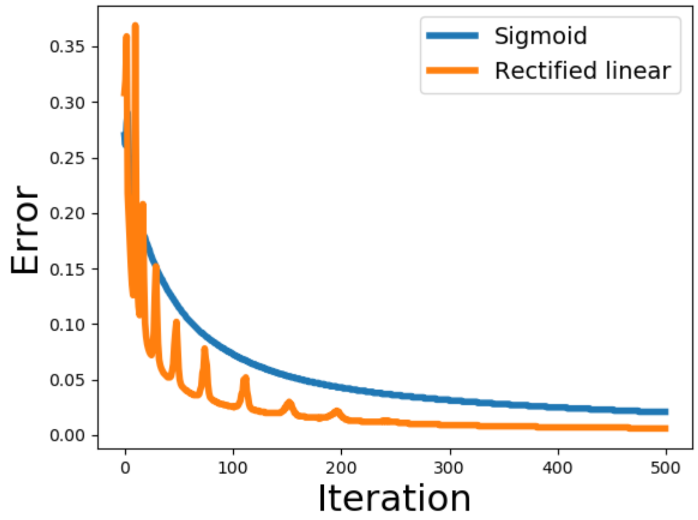
- 11982 images for training
- 1984 images for test
- One hidden layer with 20 neurons
- Learning rate = 1
- Mean squared cost function

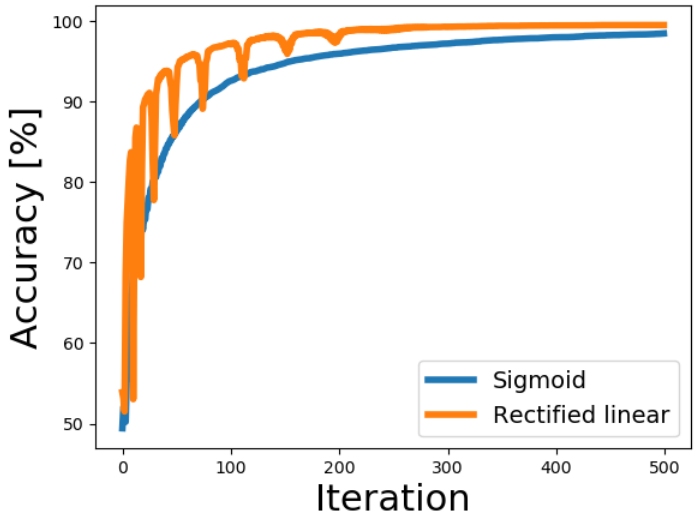
Sigmoid

Training accuracy	98.46%
Test accuracy	96.37%

Rectified linear

Training accuracy	99.51%
Test accuracy	96.47%





Sigmoid and Rectified linear (ReLu) with cross entropy

ReLu does worse in this case

Classification of shifted images of 3 and 8

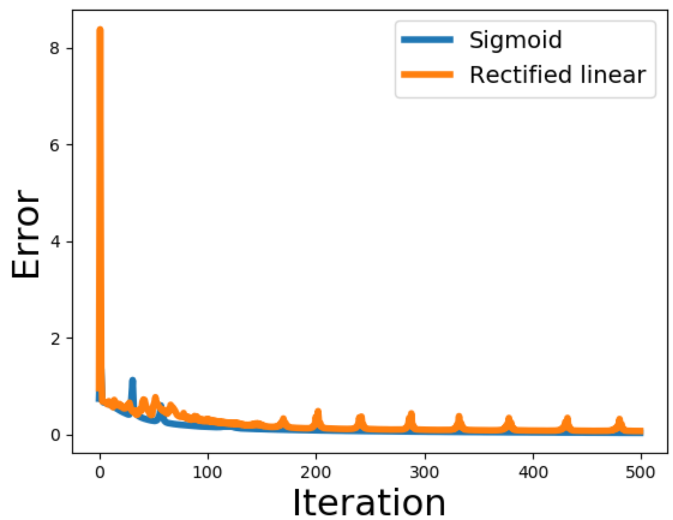
- 11982 images for training
- 1984 images for test
- One hidden layer with 20 neurons
- Learning rate = 1
- Cross entropy cost function

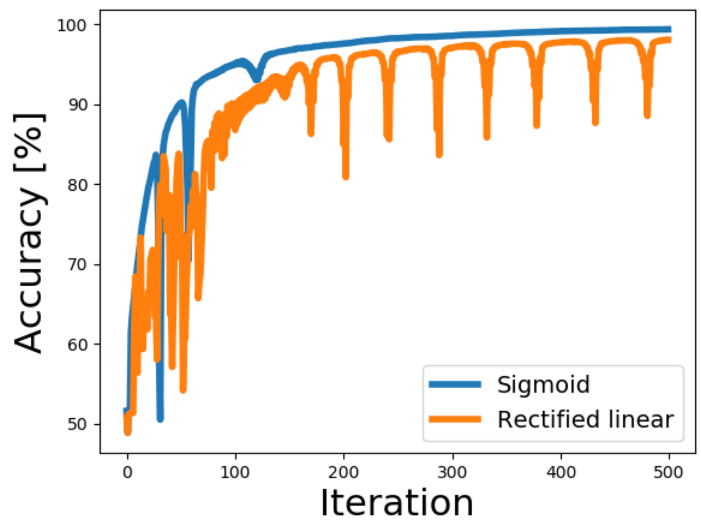
Sigmoid

Training accuracy	99.35%
Test accuracy	96.32%

Rectified linear

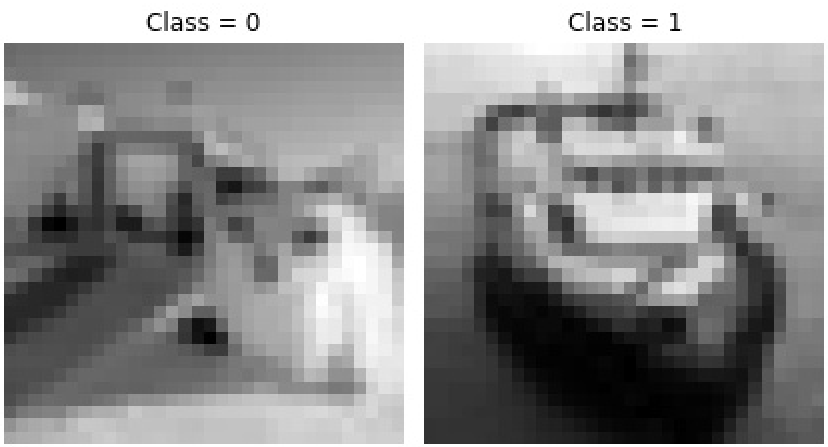
Training accuracy	98.03%
Test accuracy	95.06%

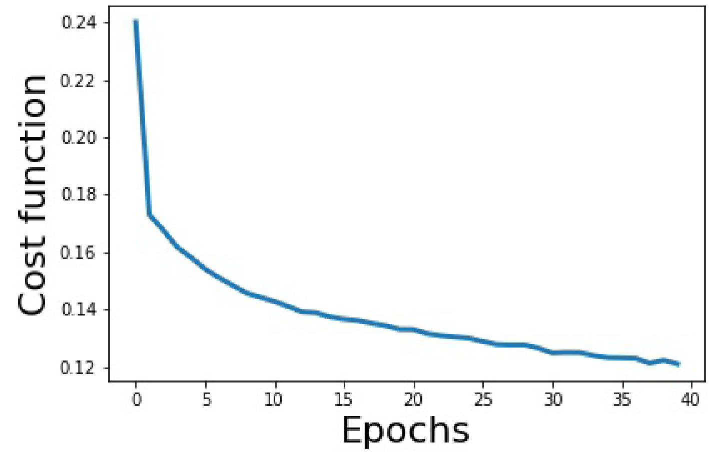


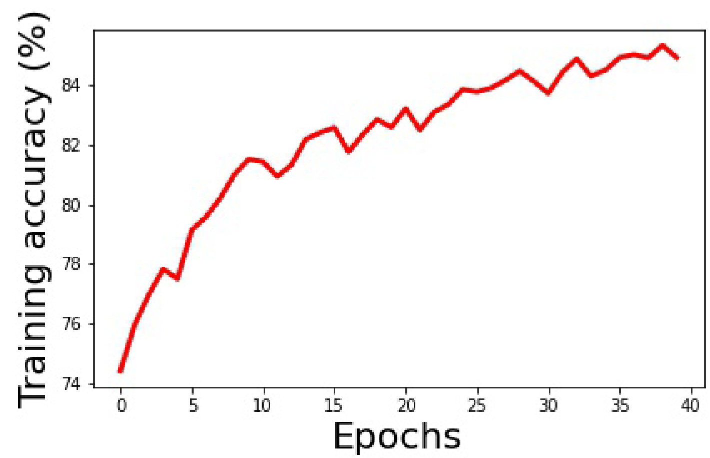


Two-layer network: Cats versus Boats

CIFAR 10: 2 classes





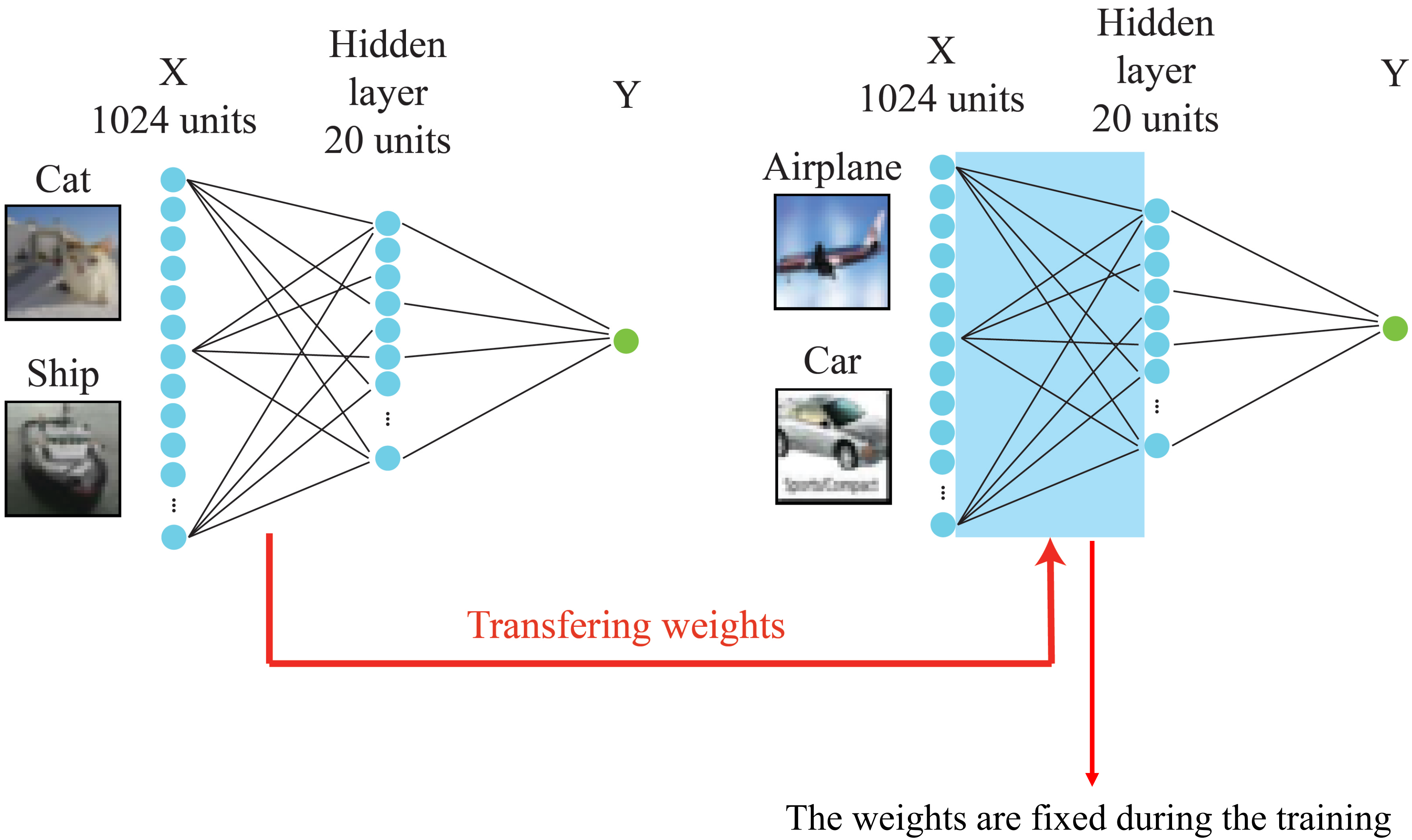


Training: 10000 samples
Test: 2000 samples
Pixel size: 32x32

Network parameters:
Optimizer: Stochastic gradient descent
Learning rate: 0.01
Number of epochs: 40
Number of hidden units: 40
Hidden layer activation function: relu
Cost function: Mean squared error
Final layer does not have any activation function

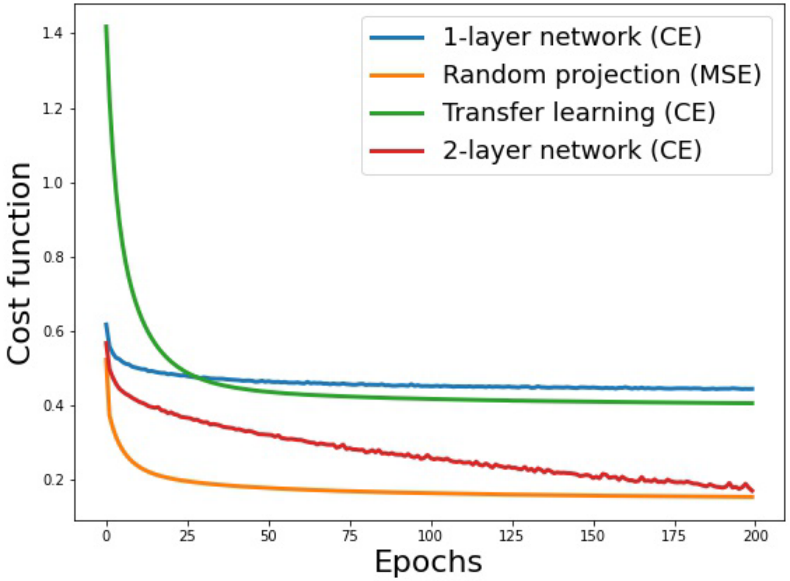
Train accuracy	84.92 %
Test accuracy	83.15 %

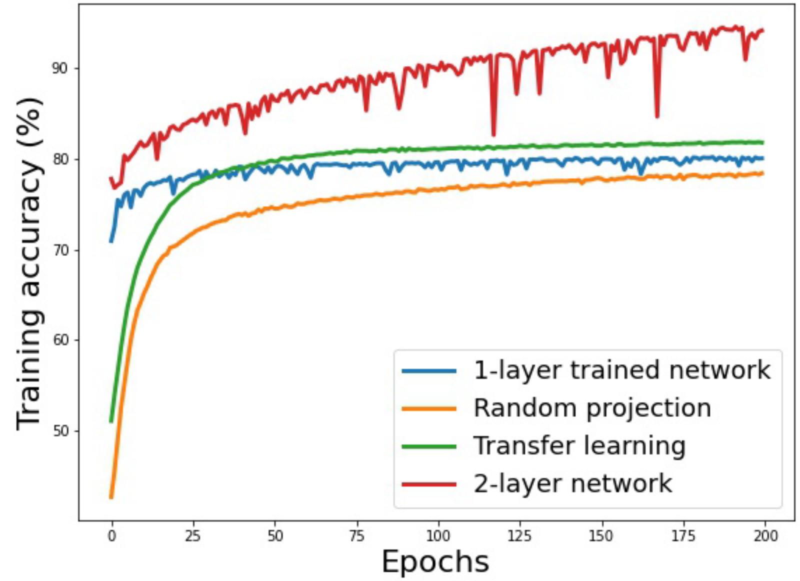
Transfer learning



Transfer learning

CIFAR 10 dataset: Cars versus Airplanes
Number of pixels: 32x32
10000 training samples
2000 test samples





One layer network:
Sigmoid activation function
Learning rate: 0.01
Stochastic gradient descent
Cost function: cross entropy
200 epochs

Random projection:
Hidden layer: Sigmoid activation function
Hidden layer: 200 neurons
Learning rate: 0.001
Stochastic gradient descent
Cost function: mean squared error
200 epochs

Transfer learning:
Hidden layer: Sigmoid activation function
Hidden layer: 200 neurons
Hidden layer trained with Cats and ships
Learning rate: 0.01
Stochastic gradient descent
Cost function: cross entropy
200 epochs

	1-layer network	Random projection	Transfer learning	2-layer network
Training accuracy	80.06	78.41	81.80	94.20
Test accuracy	79.25	78.35	81.35	88.25