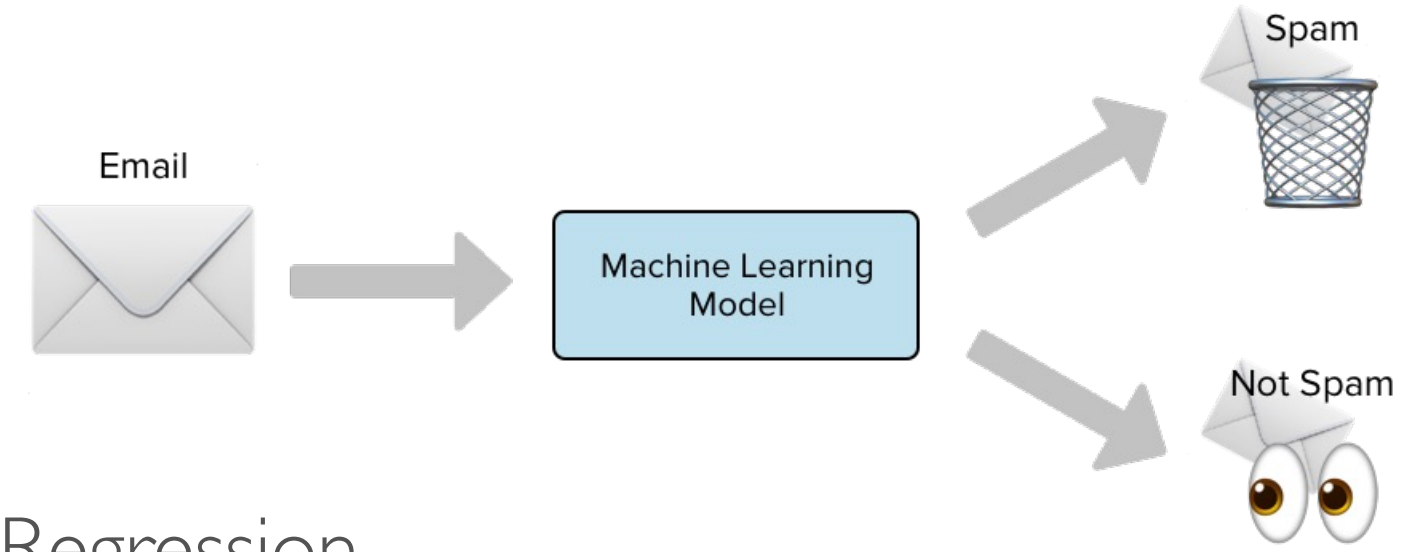
REMINDER: PROJECT

- Proposal due 10/23: 1-2 pages of problem, dataset, what you plan to do
- Spotlight slides due 10/30
- Spotlight: 11/1 in class
- Presentation: 11/29 and 12/4
- Report and deliverable due 12/13

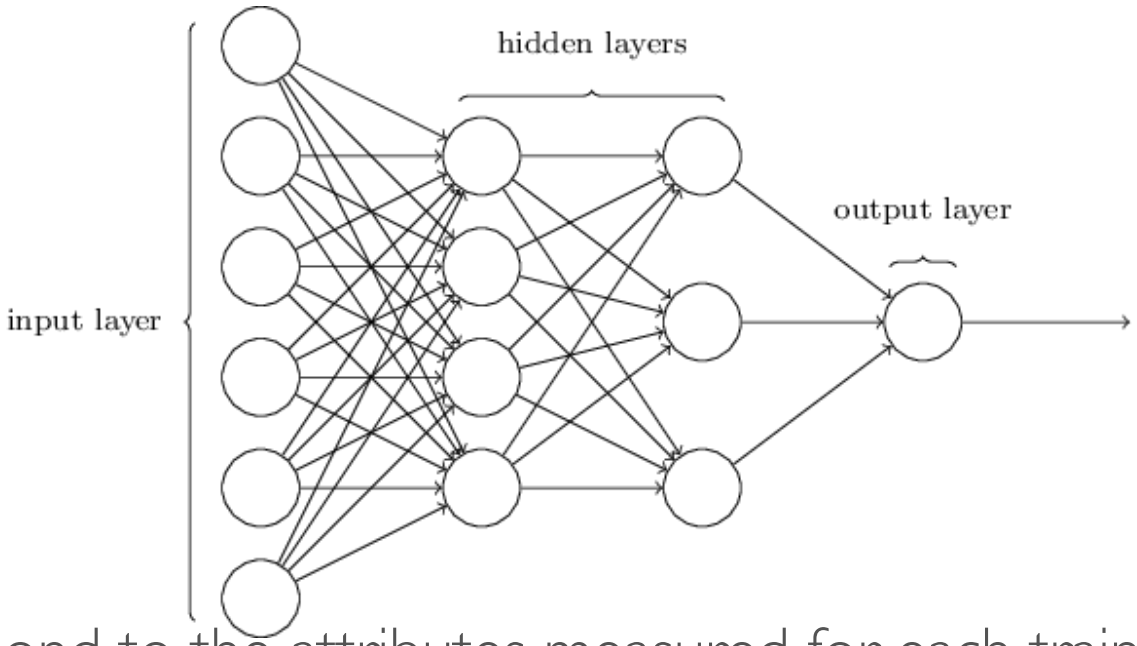
Samples posted on Piazza

HOMEWORK #4

- Out 10/13 and due **10/27 @ 11:59 PM ET** on Gradescope
- 3 questions
  - Feature extraction
  - Perceptron
  - Naïve Bayes and Logistic Regression



REVIEW: NEURAL NETWORK



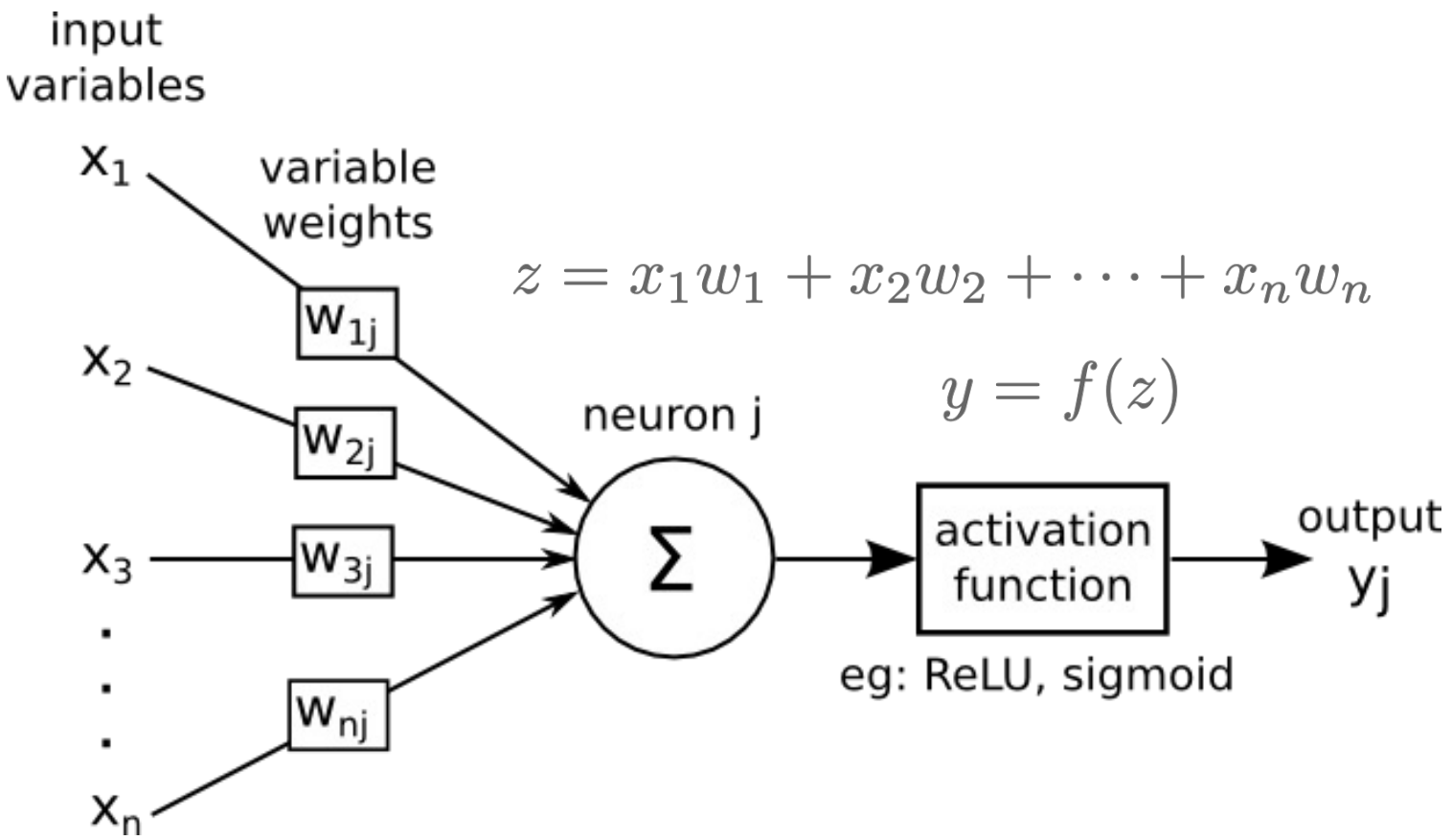
The **input layer** correspond to the attributes measured for each training tuple

They are then weighted and fed simultaneously to **hidden layers**

The weighted outputs of the last hidden layer are input to **output layer**, which emits the network's prediction

Perform **nonlinear regression**: Given enough hidden units and training samples, can closely approximate any function

REVIEW: NEURON



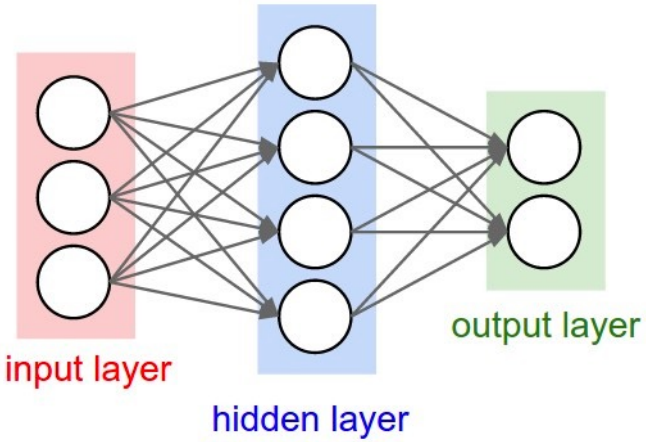
$$z = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

$$y = f(z)$$

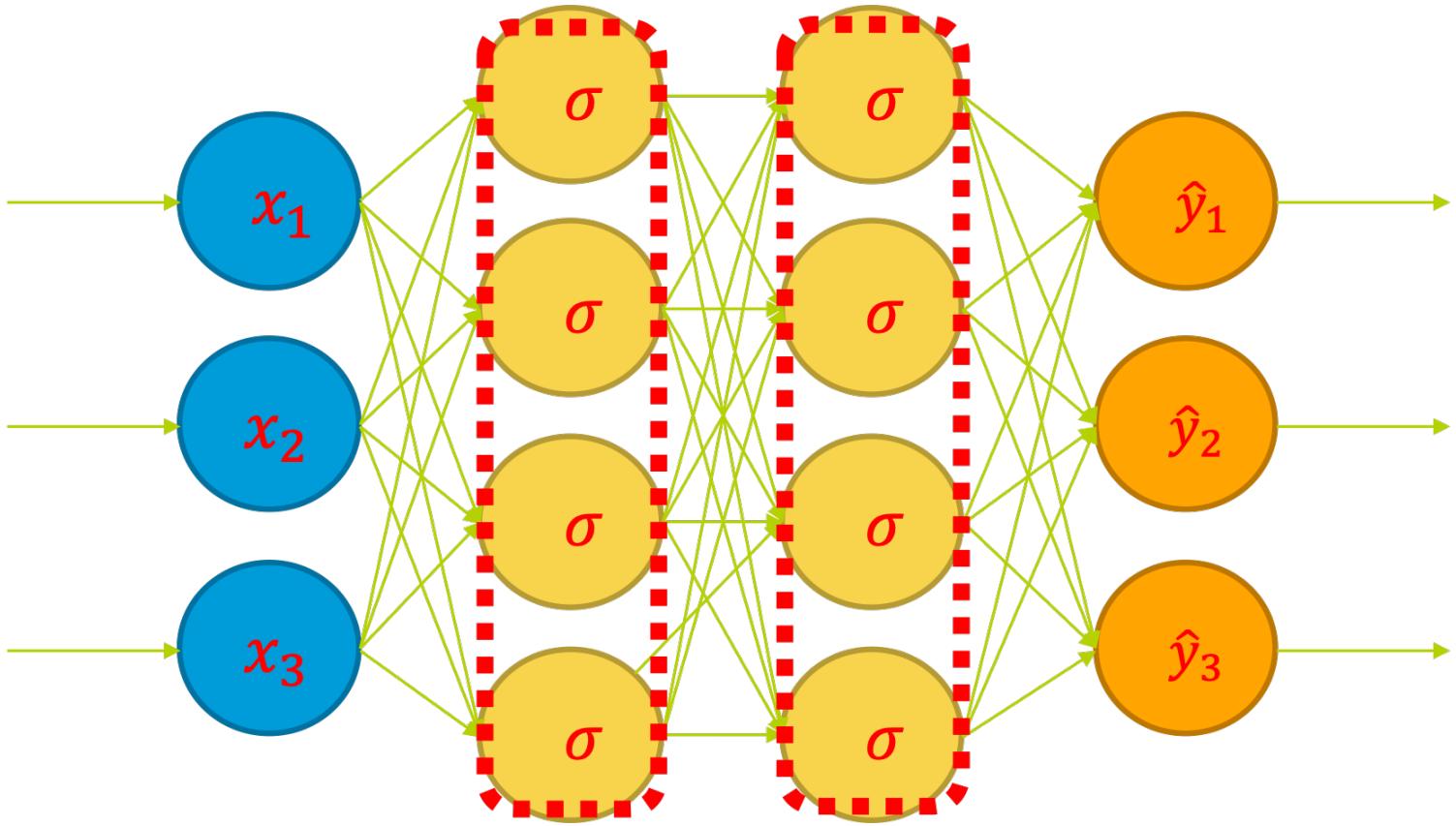
http://dataminingtheworld.blogspot.com/

REVIEW: PREDICTION AND TRAINING

- Prediction: Feed-forward computation
- Training: Learning the weights by backpropogation



WHAT DOES A NEURAL NETWORK LEARN?



GROUP ACTIVITY

MNIST DATASET

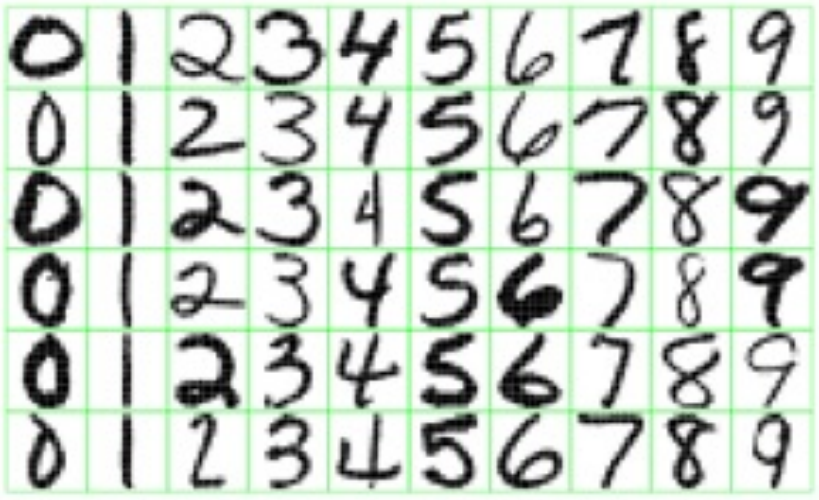
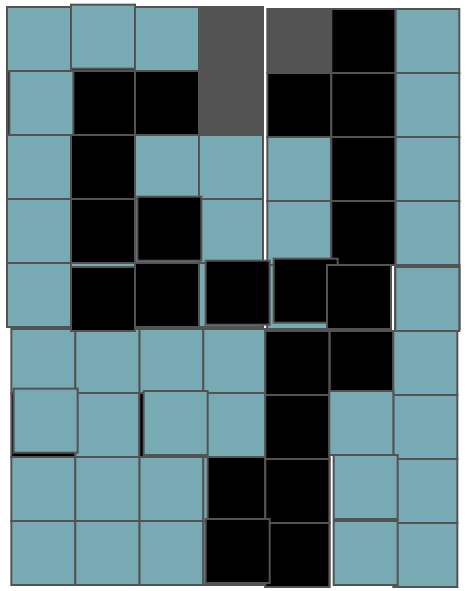
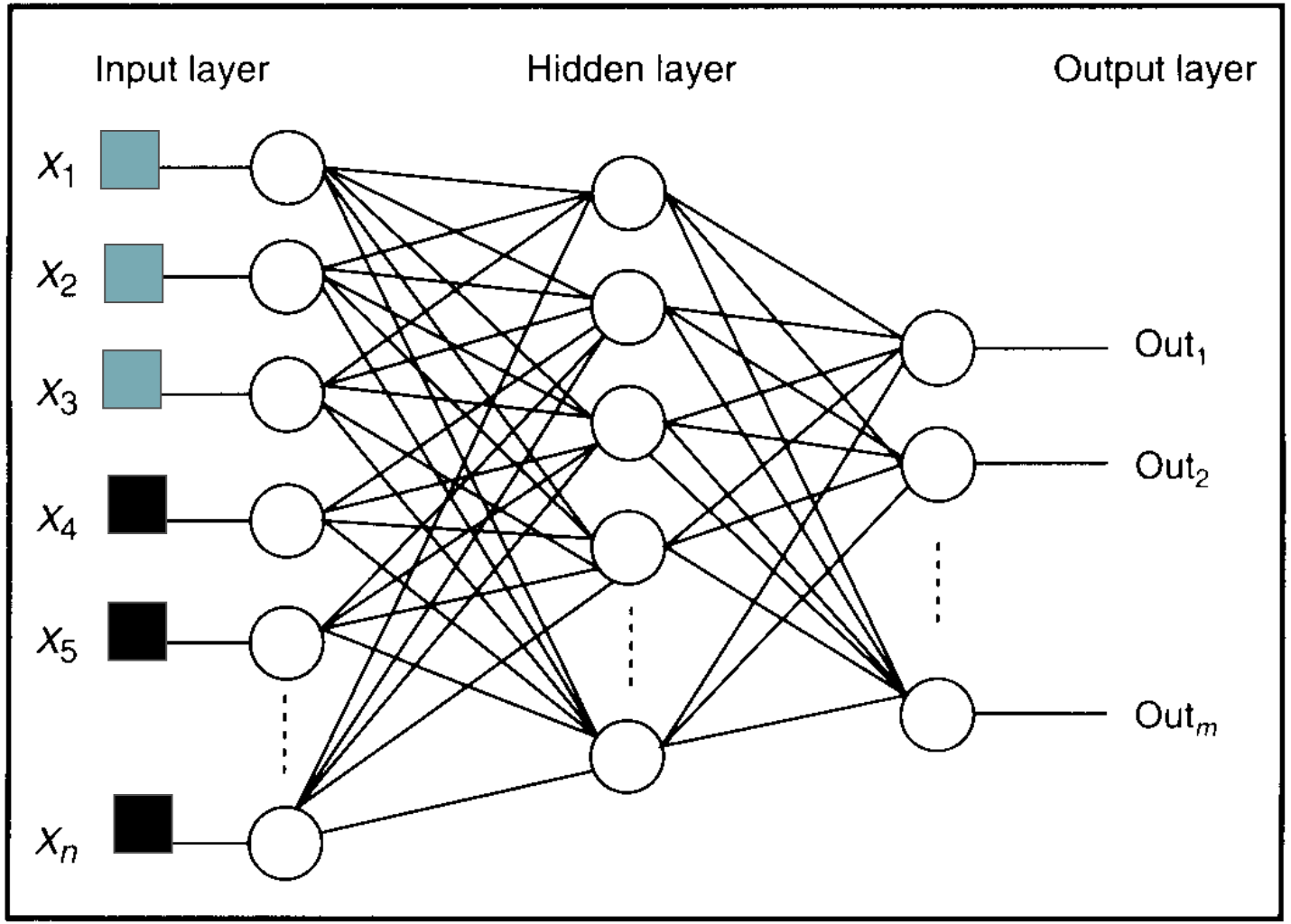


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*





What features might you expect a neural network to learn at different layers?

1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

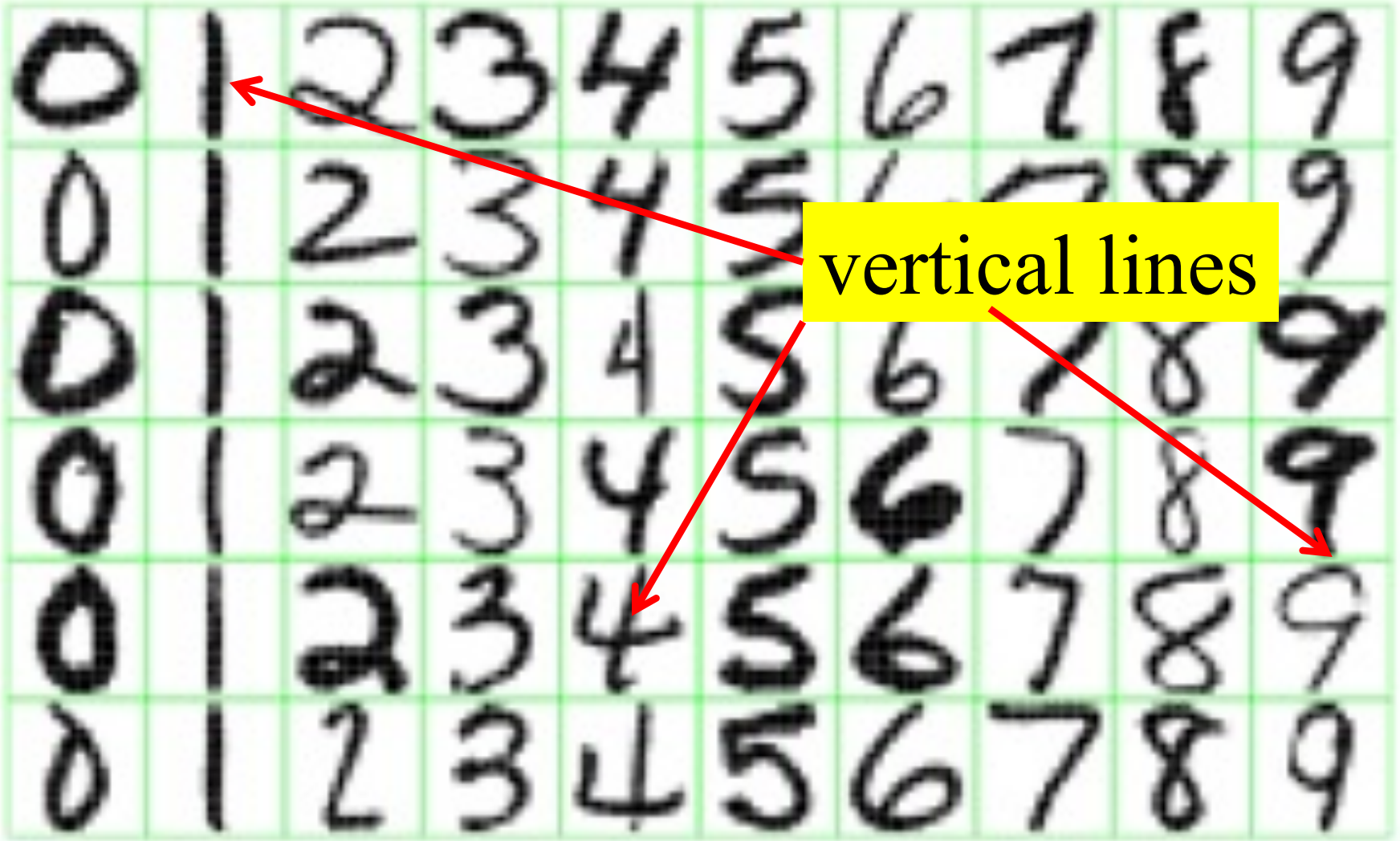


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

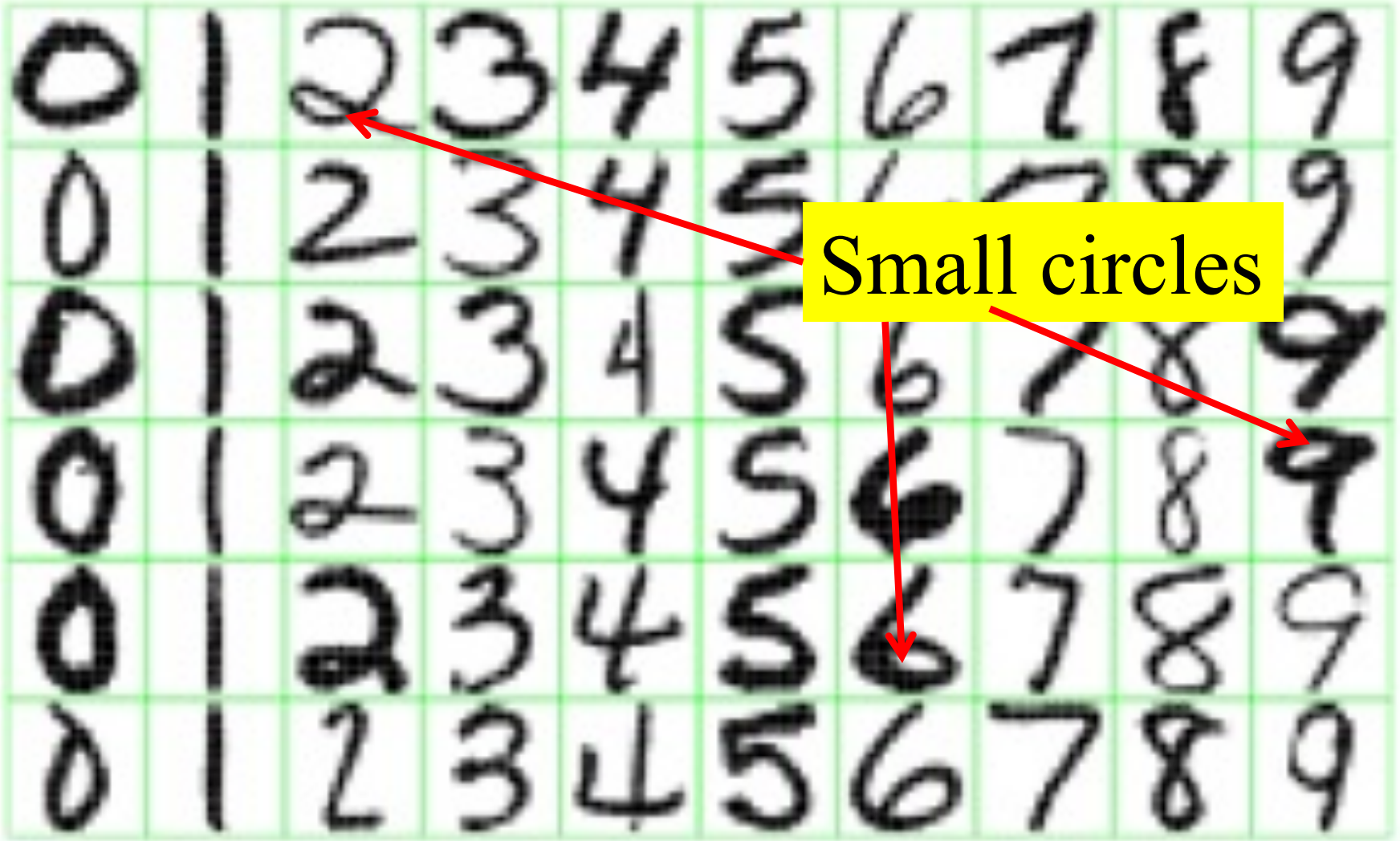
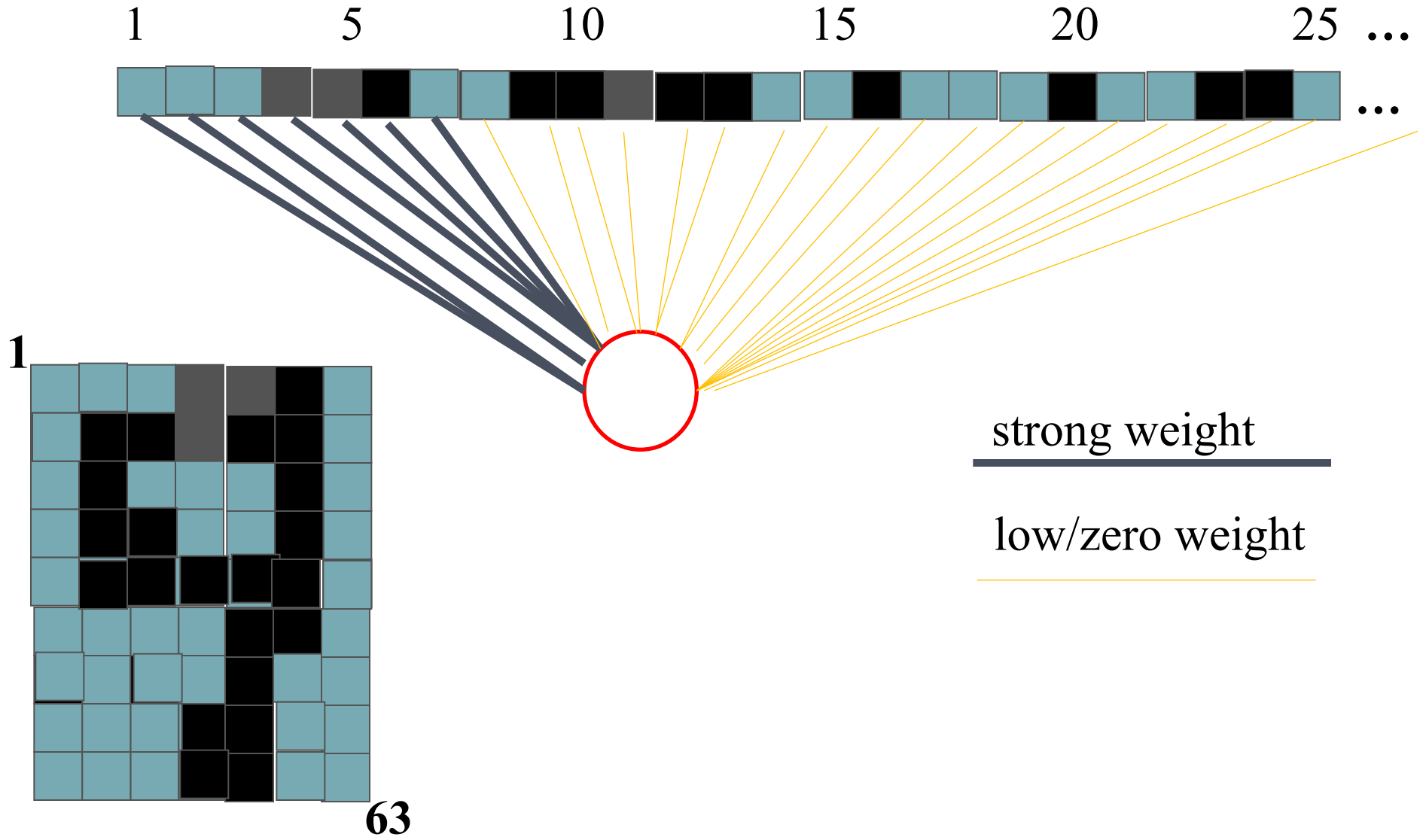
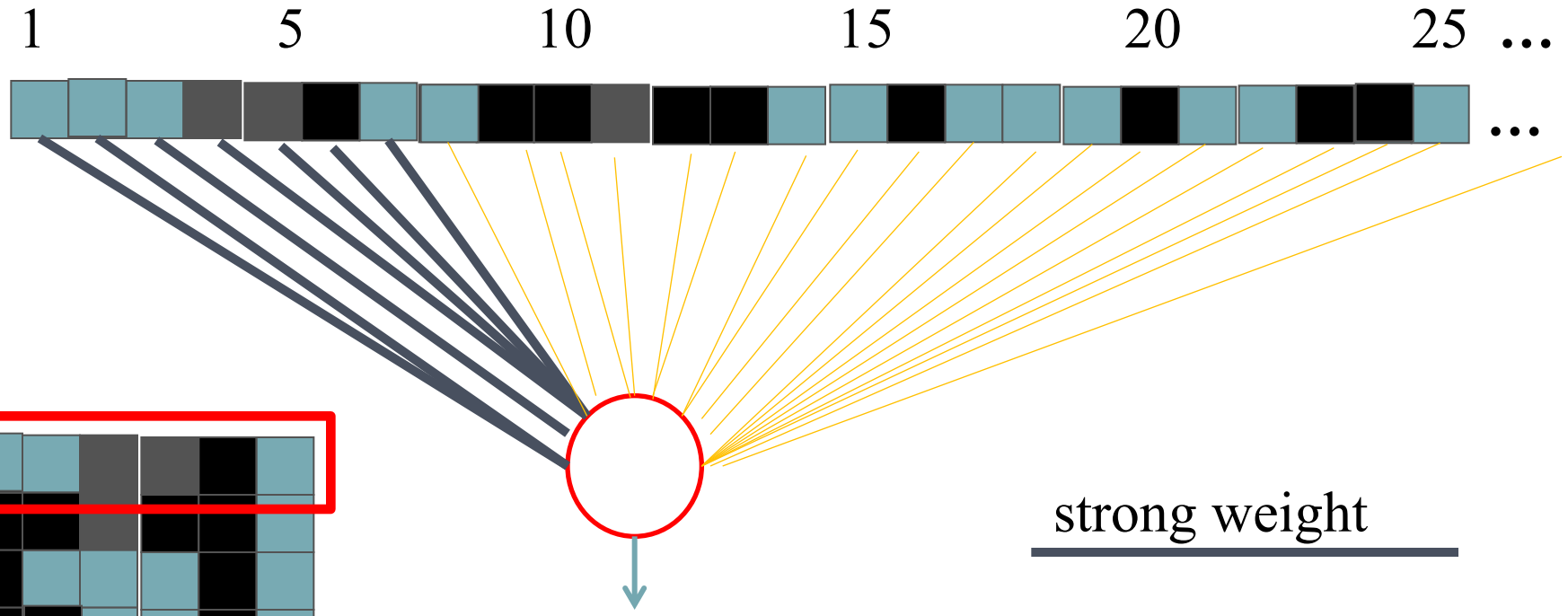


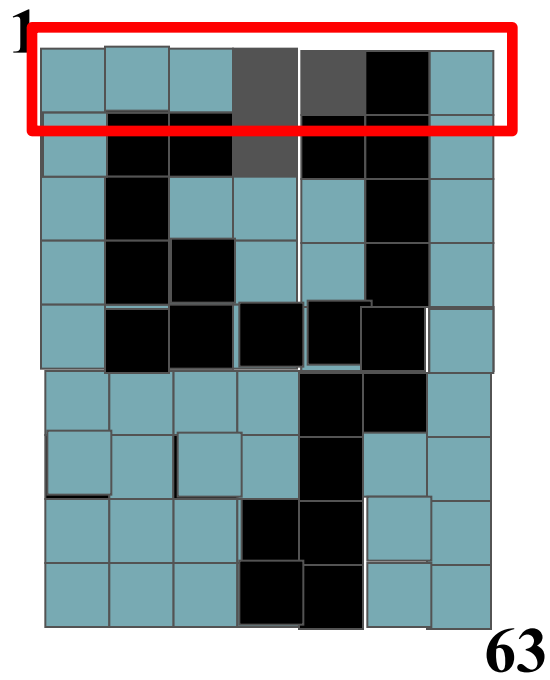
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

WHAT DOES THIS UNIT DETECT?



WHAT DOES THIS UNIT DETECT?



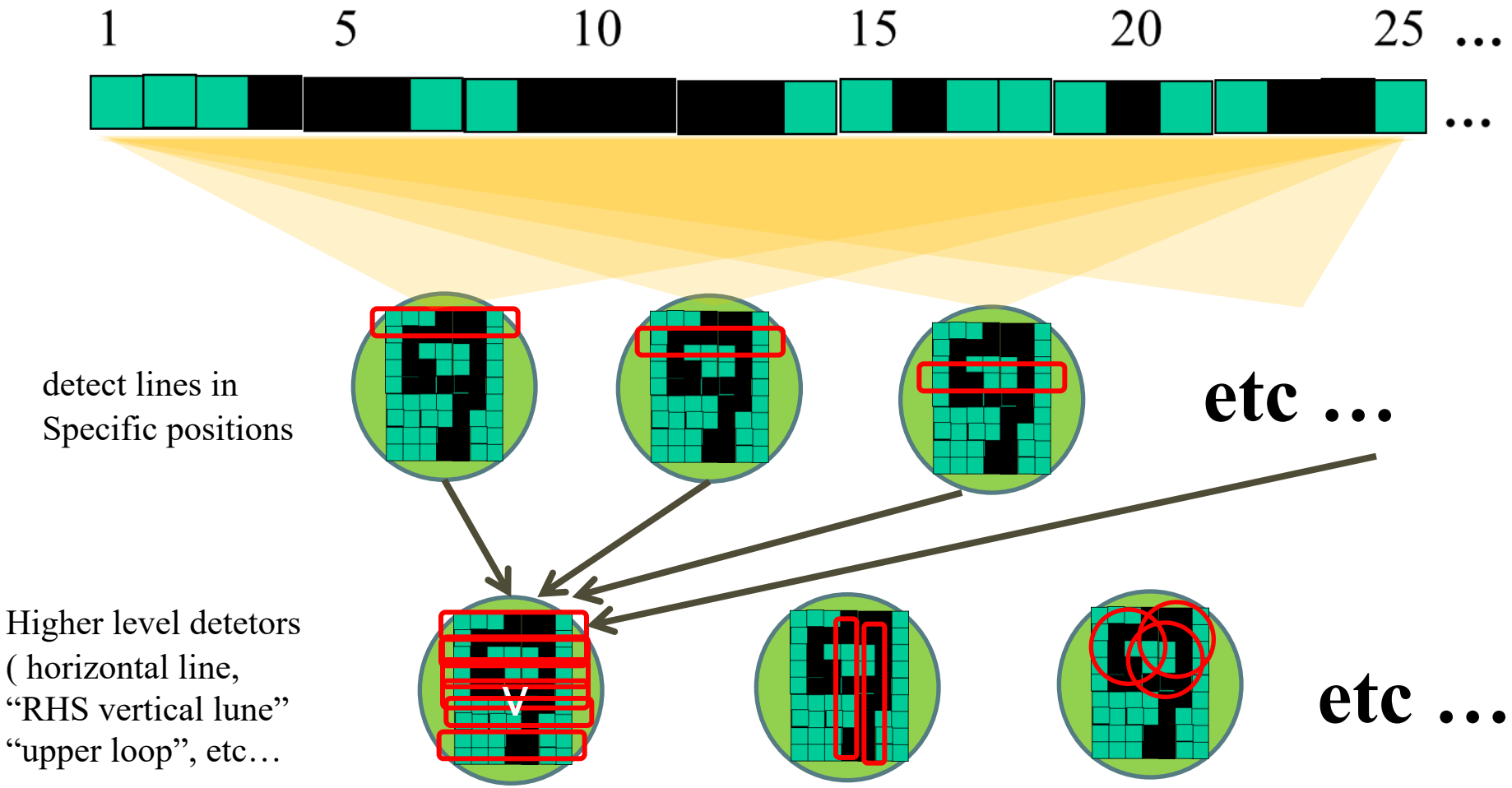


strong weight

low/zero weight

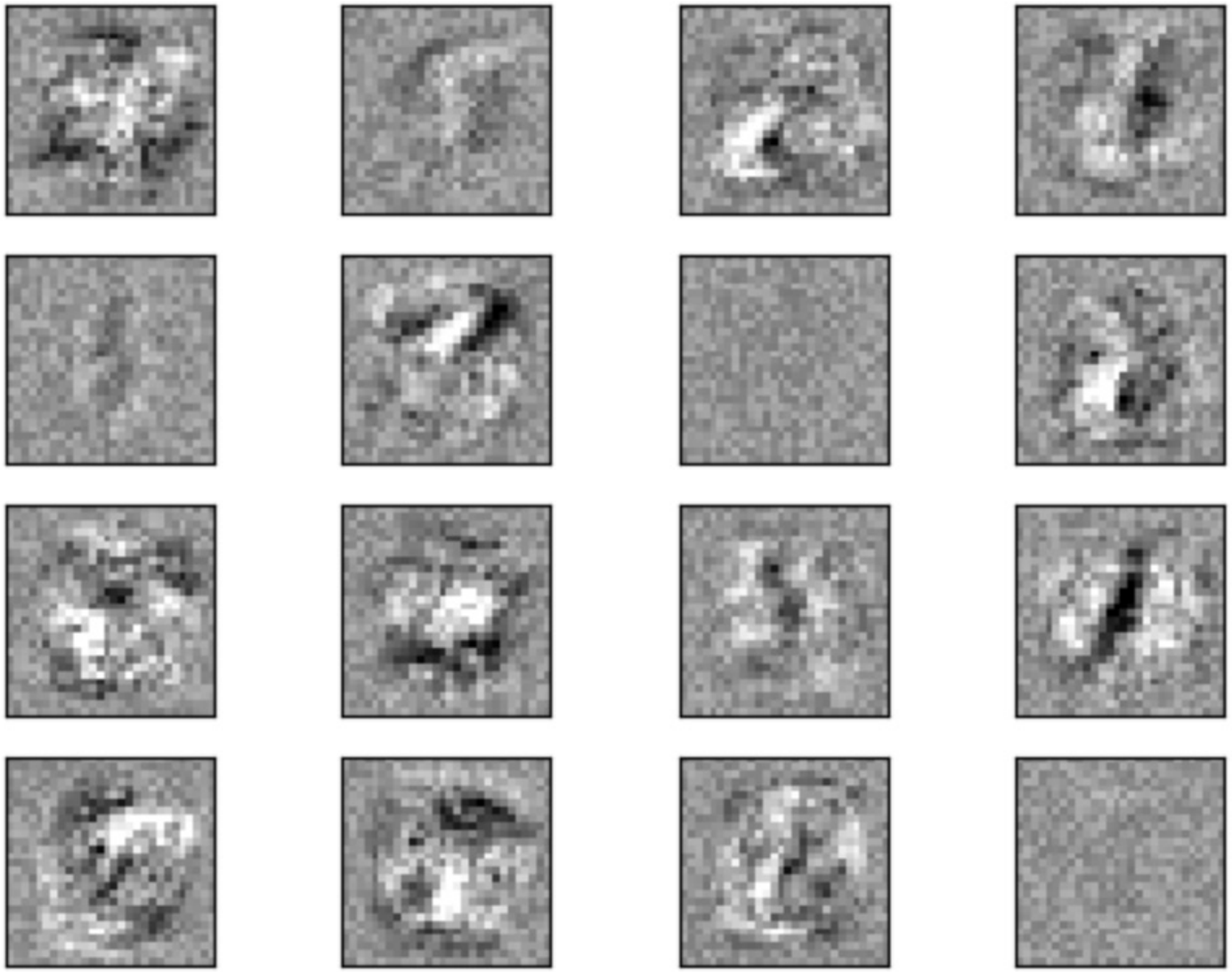
it will send strong signal for a horizontal line in the top row, ignoring everywhere else

HIERARCHICAL FEATURE LEARNING



VISUALIZATION OF WEIGHTS ON MNIST

- First layer



https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py

BACKPROPAGATION: PRACTICAL CONSIDERATIONS

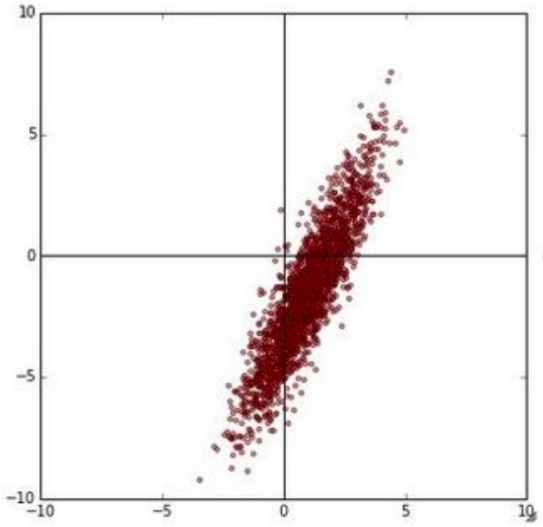
- Do we need to pre-process the training data? If so, how?
- How do we choose the initial weights?
- How do we choose an appropriate learning rate?
- Are some activation functions better than others?
- How do we choose the network structure?
- How to address overfitting?

PRE-PROCESSING DATA

- In principle, can use any raw input-output data
- Pre-process can help learning
  - Zero-centering: normalize to zero mean
  - Standardization: normalize to zero mean and standard deviation of 1
  - De-correlate data: remove correlated features and transformed data with diagonal covariance matrix
  - Whiten data: convert diagonal covariance matrix to identity matrix so all eigenvalues are the same

PRE-PROCESSING DATA

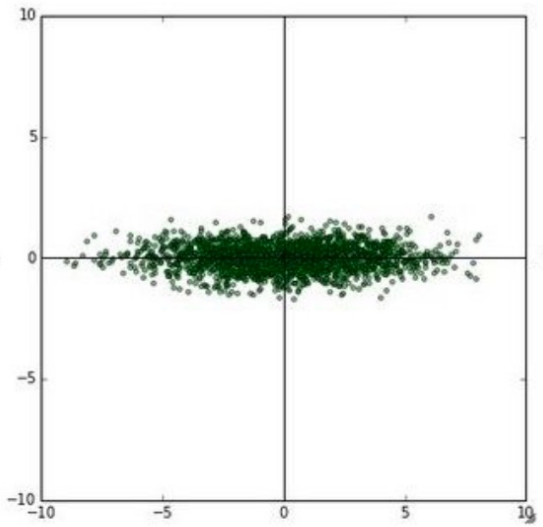
original data



Covariance matrix is not diagonal

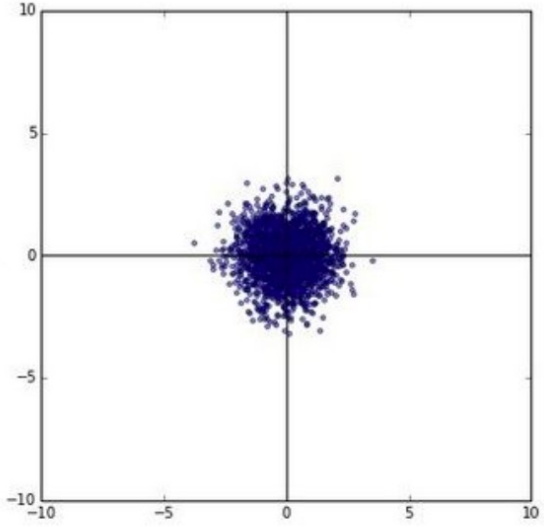
$$\mathbf{\Sigma} = E(\mathbf{x}\mathbf{x}^T)$$

decorrelated data



(data has diagonal covariance matrix)

whitened data



(covariance matrix is the identity matrix)

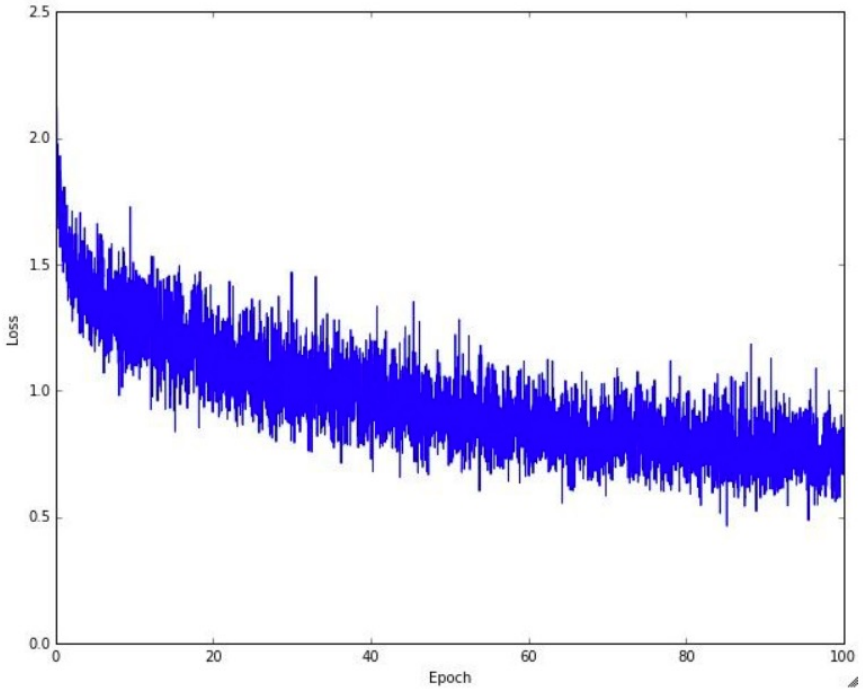
CHOOSING INITIAL WEIGHTS

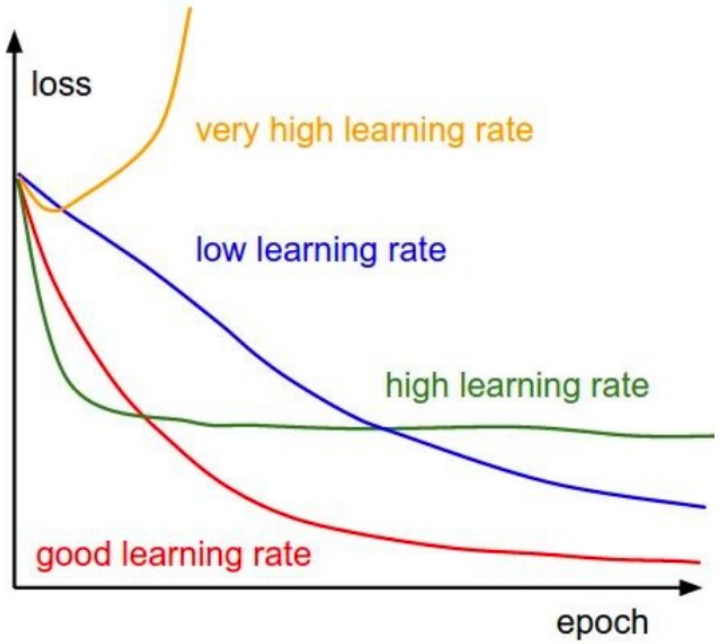
- Neural networks with hidden layers have non-convex loss function -> different initialization can lead to different accuracy
- All weights are treated the same way using gradient descent —> do not initialize with the same values
- Generally start off weights with small random values that do not cause saturation
- Proper initialization is an active area of research

CHOOSING LEARNING RATE

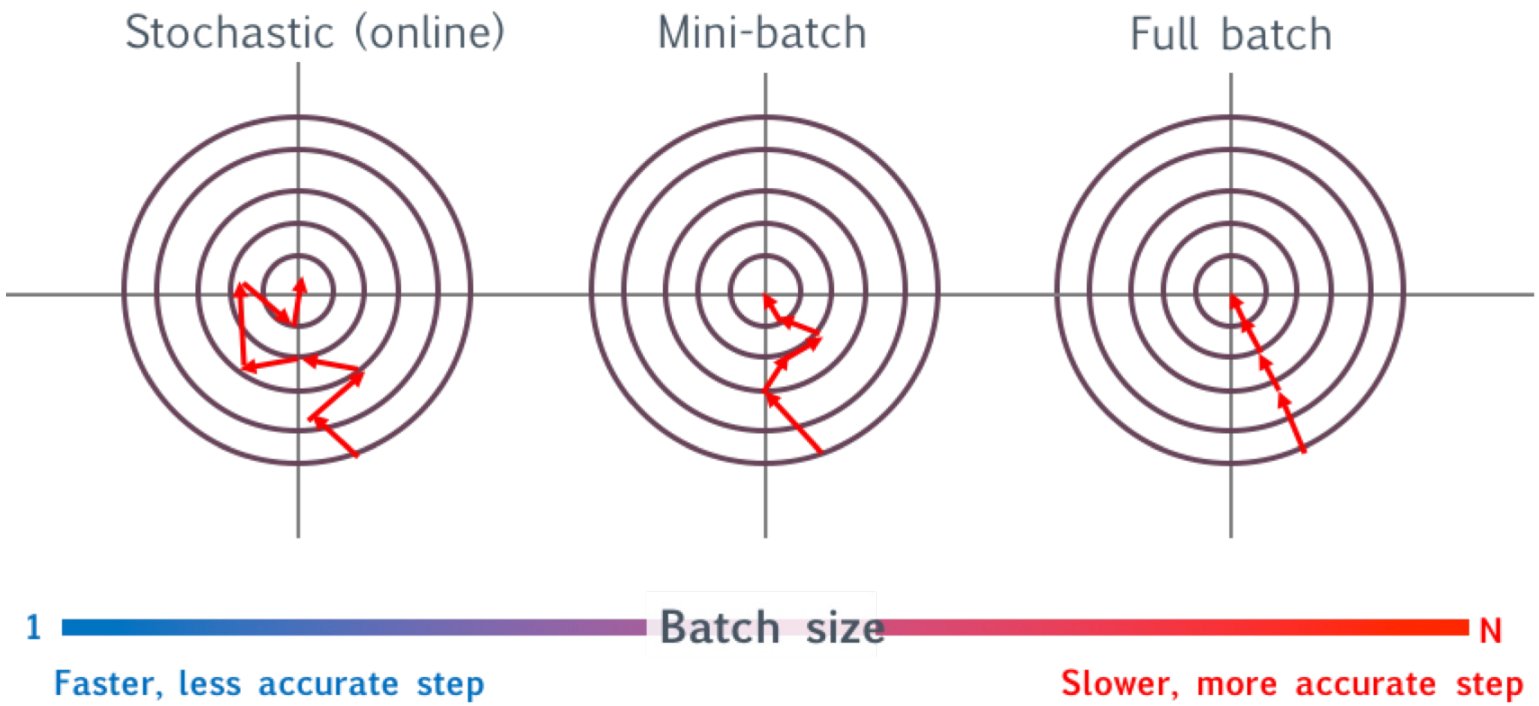
- If learning rate is too small, it will take a long time to get anywhere near the minimum of the error function
- If learning rate is too large, the weight updates will overshoot the error minimum and weights will oscillate or even diverge
- Solution: Babysit the learning process at the beginning for small portion of training data

CHOOSING LEARNING RATE





BATCHING APPROACHES

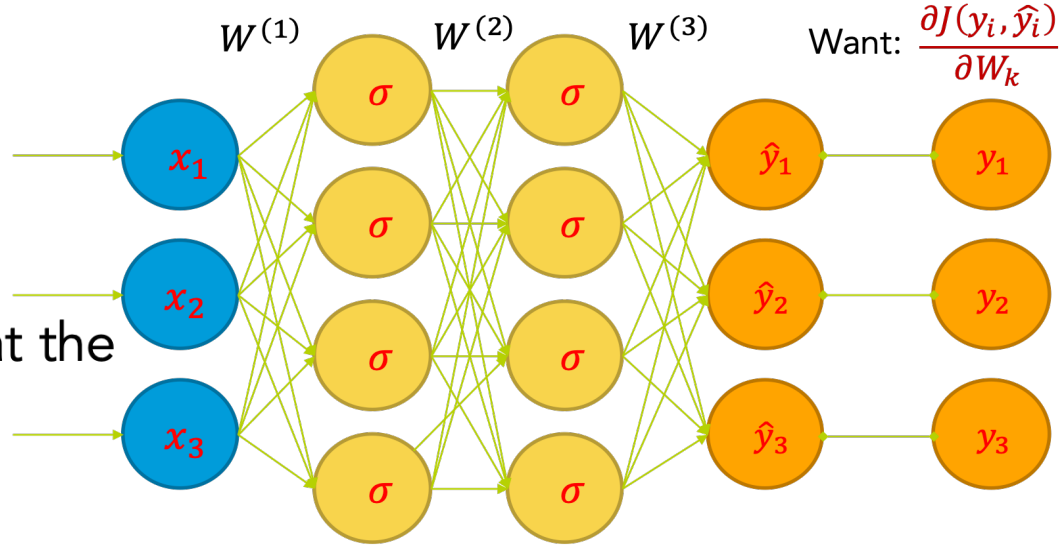


VANISHING GRADIENTS

Recall that:

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot X$$

- Remember: $\sigma'(z) = \sigma(z)\left(1 - \sigma(z)\right) \leq .25$
- As we have more layers, the gradient gets very small at the early layers.
- This is known as the "vanishing gradient" problem.
- For this reason, other activations (such as ReLU) have become more common.



POPULAR ACTIVATION FUNCTIONS

- Sigmoid function

$$\sigma(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{x})}$$

- Hyperbolic tangent function

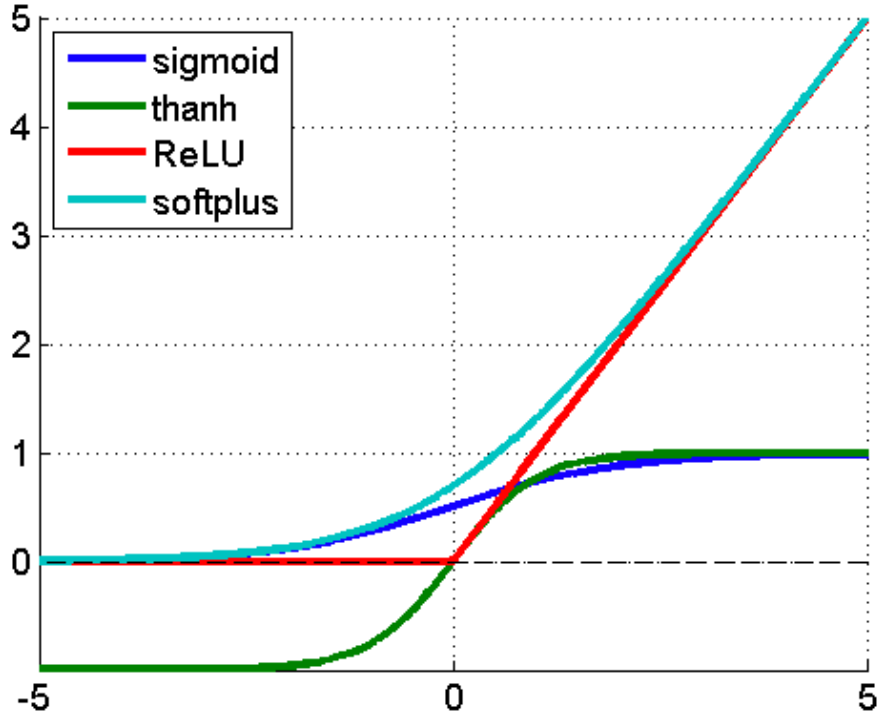
$$f(\mathbf{x}) = \sinh(\mathbf{x})/\cosh(\mathbf{x})$$

- Rectified linear unit (ReLU)

$$f(\mathbf{x}) = \max(0, \mathbf{x})$$

- Softplus

$$f(\mathbf{x}) = \log(1 + \exp(\mathbf{x}))$$

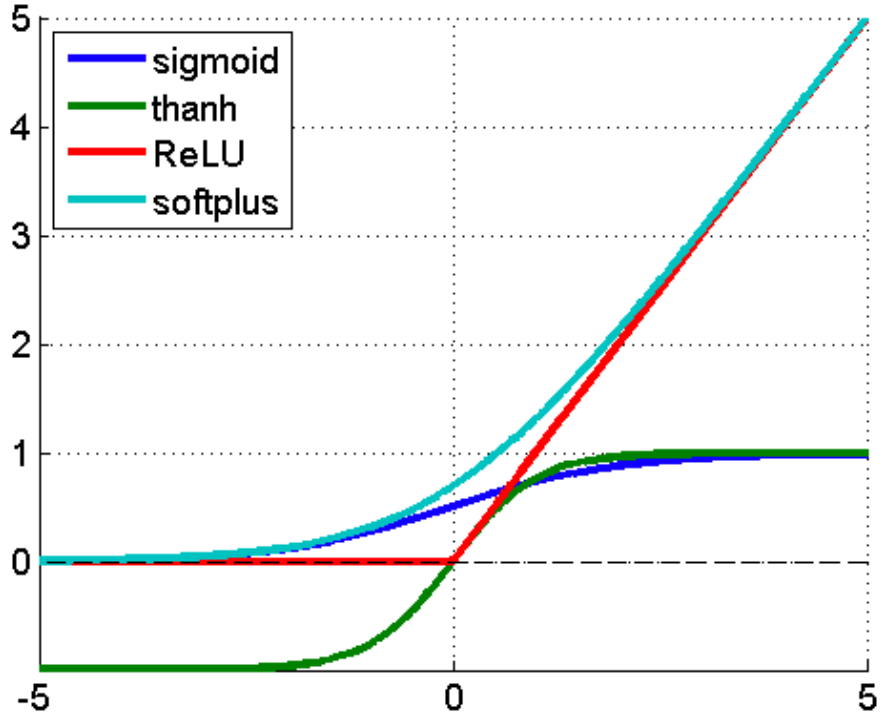


https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearities/

Can we use a linear activation function?

ACTIVATION FUNCTIONS: IN PRACTICE

- Use ReLU and be careful with the learning rates
  - Avoids vanishing gradient problem
  - Sparse representation
  - Simple and fast
- Try out tanh but don't expect too much
- Don't use sigmoid



NETWORK STRUCTURE

- In theory a single hidden layer with a large number of units has the ability to approximate most functions
- The learning task of discovering a good solution is made much easier with multiple layers each of modest size
- Deep learning (later)

ADDRESSING OVERFITTING

- A 3 layer network for MNIST
  - 784 (28*28) units for the input layer
  - 256 and 128 units for the hidden layers
  - 10 units for the output layer
- Has 235,146 parameters!
- Recall we only have 60,000 training images

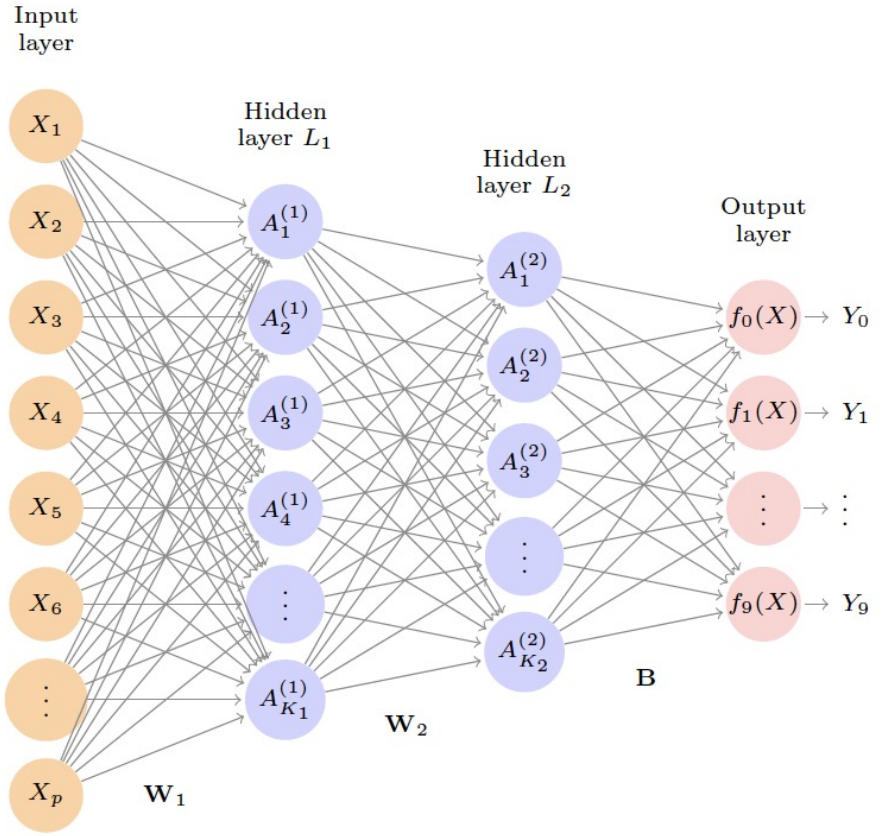


FIGURE 10.4. *Neural network diagram with two hidden layers and multiple outputs, suitable for the* MNIST *handwritten-digit problem. The input layer has* $p = 784$ *units, the two hidden layers* $K_1 = 256$ *and* $K_2 = 128$ *units respectively, and the output layer* 10 *units. Along with intercepts (referred to as* biases *in the deep-learning community) this network has 235,146 parameters (referred to as weights).*

OVERFITTING

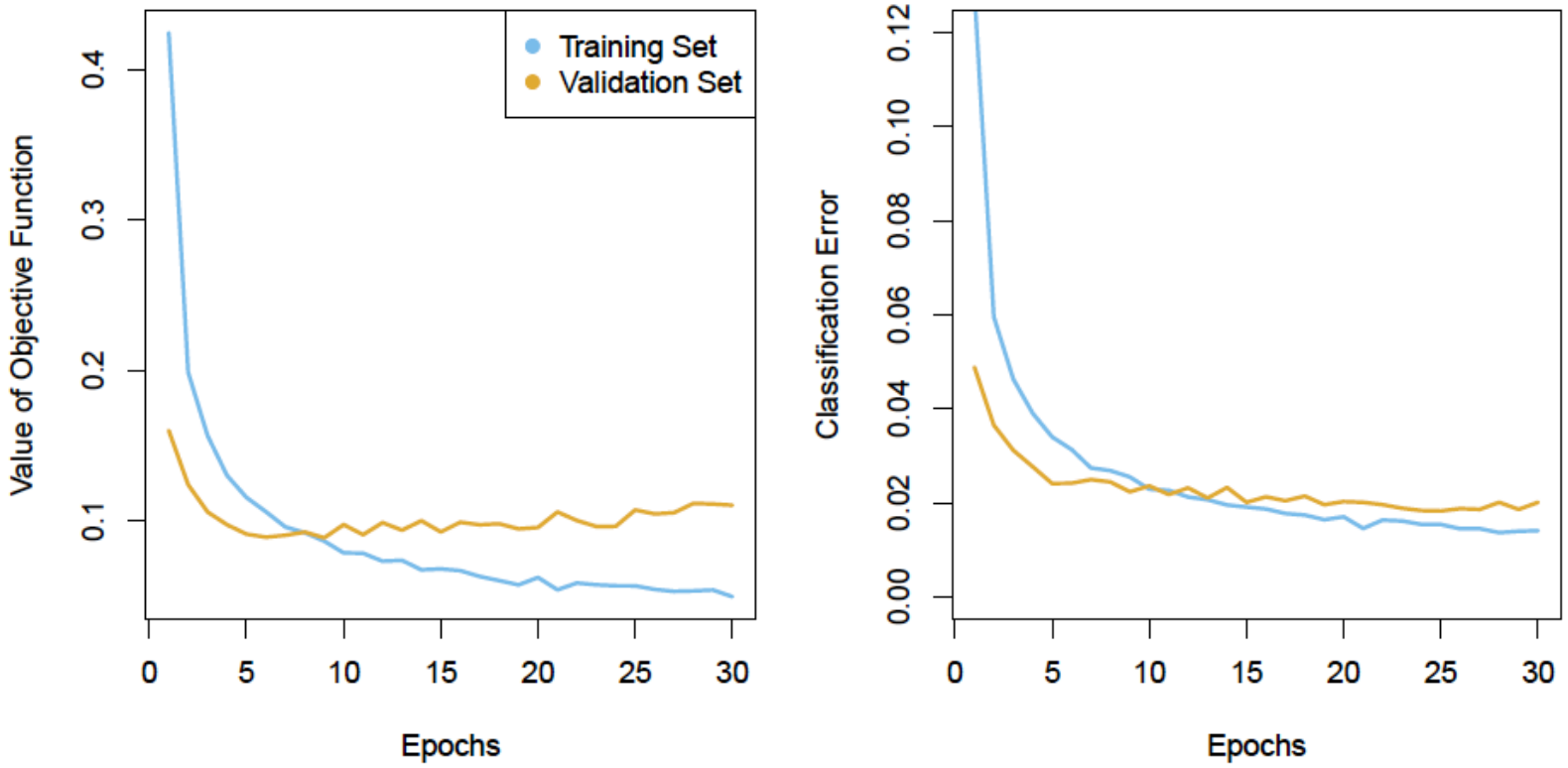


FIGURE 10.18. *Evolution of training and validation errors for the* `MNIST` *neural network depicted in Figure 10.4, as a function of training epochs. The objective refers to the log-likelihood (10.14).*

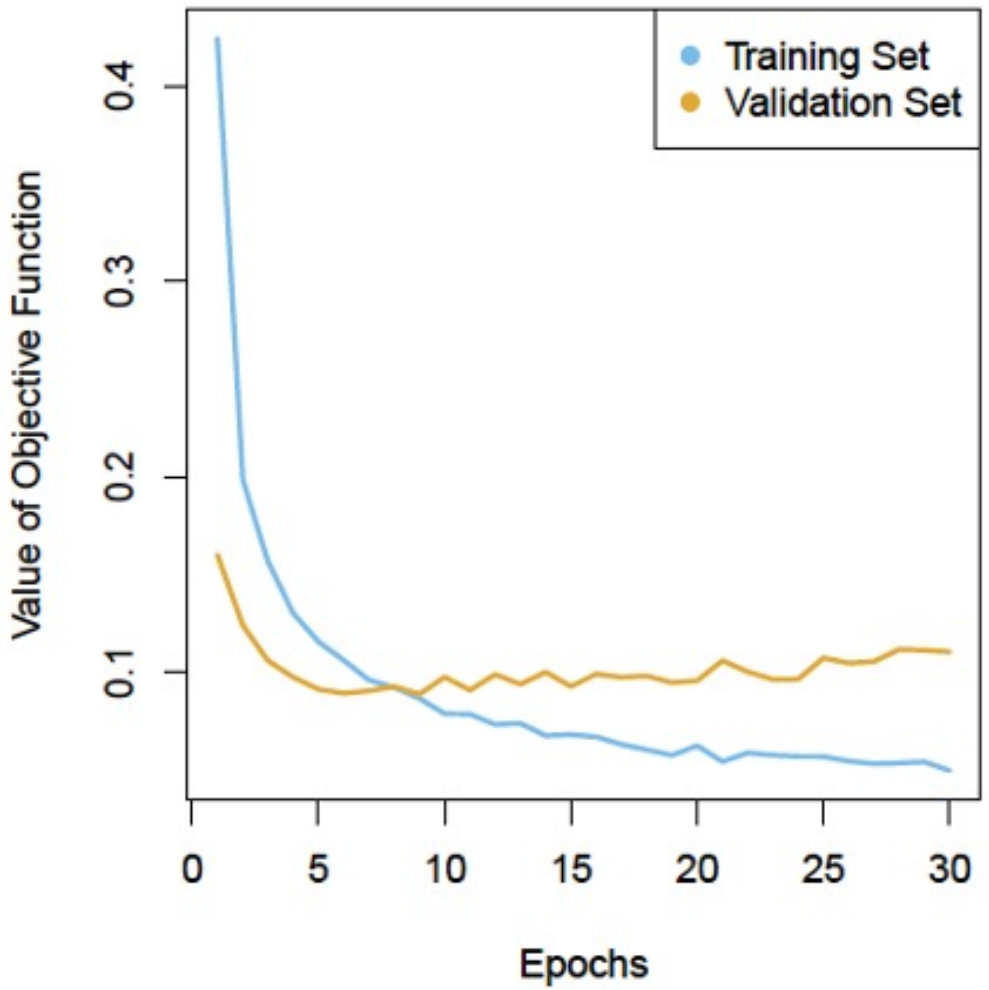
How to avoid overfitting?

ADDRESSING OVERFITTING

- Several different methods
  - Early stopping
  - Regularization penalty in cost function
  - Dropout
  - Stochastic / Mini-batch GD (to some degree)

EARLY STOPPING

- Choosing some rules after which to stop training
- Example:
  - Check validation log-loss every 10 epochs
  - If it is higher than it was last time, stop and use the previous model (i.e., from 10 epochs previous)



PENALIZED COST FUNCTION

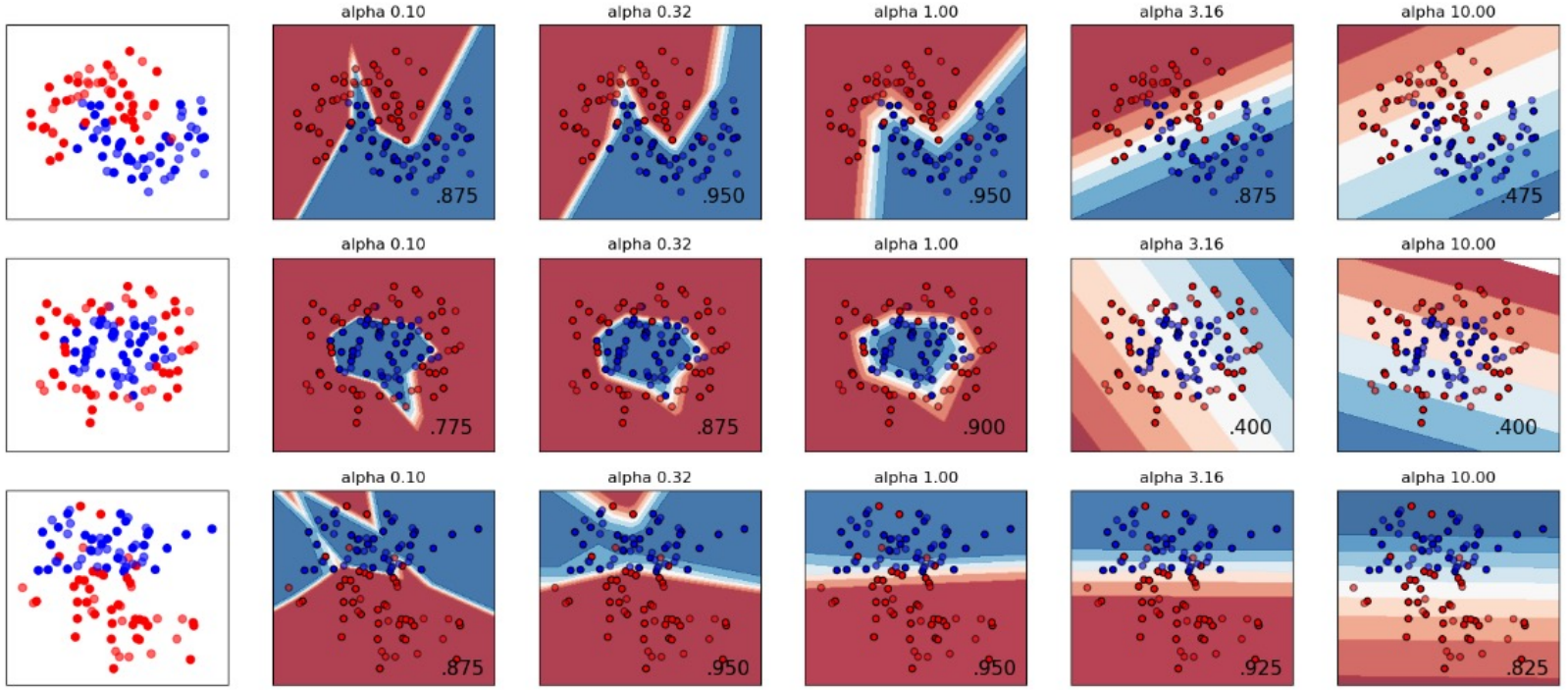
- Add a penalty to the loss function for high weights
- L2 regularization: similar to ridge regression

$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda\sum_{j=1}^{m}W_i^2$$

- Can also use L1 regularization
- Can have different penalty weight for different layers
- Can have analogous expression for categorical cross entropy (classification)

EFFECT OF REGULARIZATION IN MLP

- Higher penalty term encourages smaller weights, resulting in less complex decision boundaries



https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html#sphx-glr-auto-examples-neural-networks-plot-mlp-alpha-py

DROPOUT

- Mechanism where at each training iteration (batch) we randomly remove a subset of neurons
- Prevents NN from relying too much on individual pathways or co-adapt too much — making it more "robust"
- At test time, "rescale" the weight of the neuron to reflect the percentage of time it was active

DROPOUT: PICTORIALLY

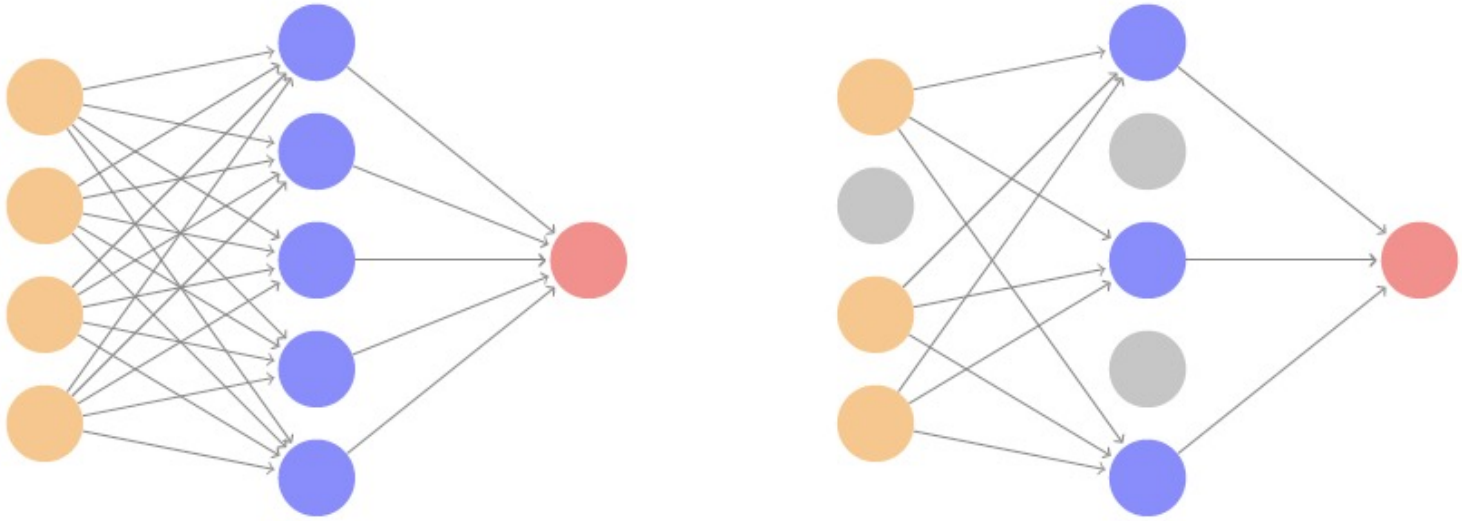
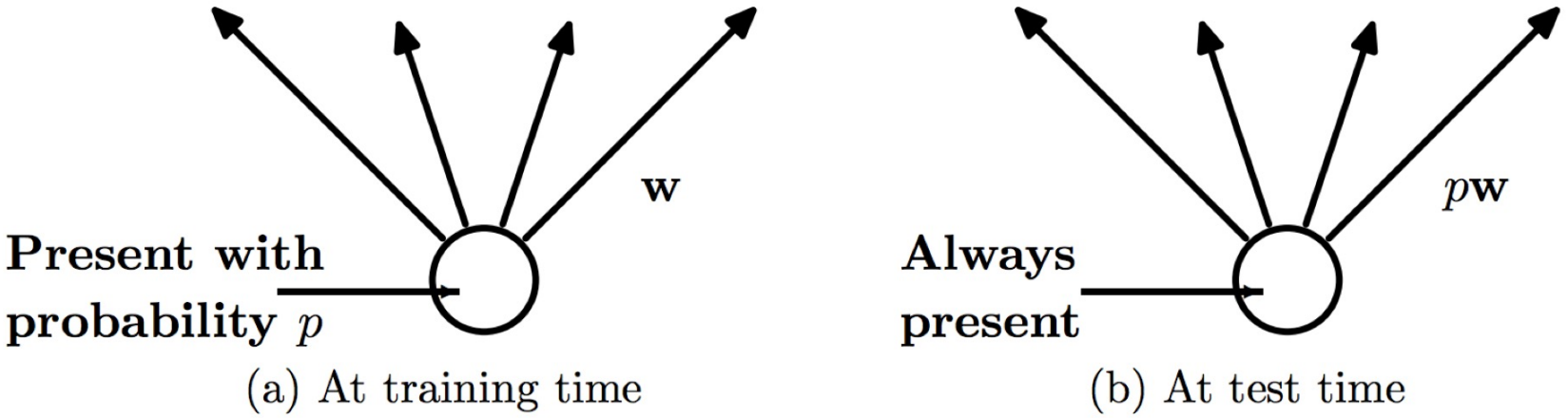


FIGURE 10.19. *Dropout Learning. Left: a fully connected network. Right: network with dropout in the input and hidden layer. The nodes in grey are selected at random, and ignored in an instance of training.*

What happens to the number of iterations and training time for each epoch?

DROPOUT: TEST TIME

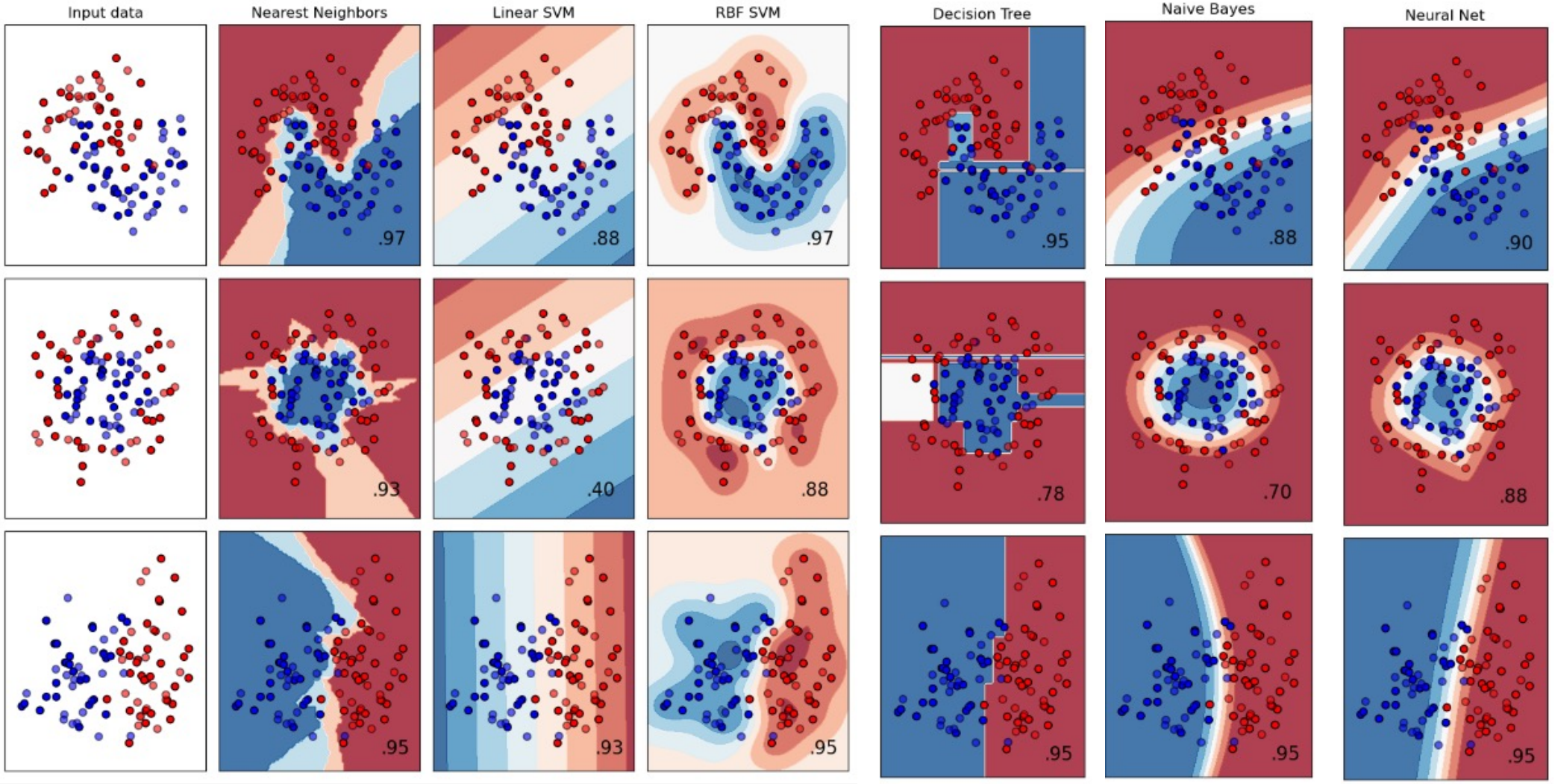


(a) At training time (b) At test time

NEURAL NETWORKS: HYPER PARAMETER TUNING

- Network structure: number of hidden layers, number of units per layer
- Regularization: dropout rate, penalty strength
- Stochastic gradient descent: batch size, learning rate, number of epochs

WHEN TO USE NEURAL NETWORK: DECISION BOUNDARY COMPARISON



WHEN TO USE NEURAL NETWORK

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

TABLE 10.1. *Test error rate on the* `MNIST` *data, for neural networks with two forms of regularization, as well as multinomial logistic regression and linear discriminant analysis. In this example, the extra complexity of the neural network leads to a marked improvement in test error.*

WHEN TO USE NEURAL NETWORK

- Predict the Salary of a baseball player in 1987 using his performance statistics from 1986

- 9 variables, training set of 176 players, test set of 87 players

Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network	1345	257.4	0.54

TABLE 10.2. *Prediction results on the* `Hitters` *test data for linear models fit by ordinary least squares and lasso, compared to a neural network fit by stochastic gradient descent with dropout regularization.*

SKLEARN: NEURAL NETWORKS

- MLPClassifier
  - Cross-entropy loss
  - solver: lbfgs, sgd, adam
  - alpha: L2 regularization strength

```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
```

https://scikit-learn.org/stable/modules/neural_networks_supervised.html