Neural networks (NN)[1]

Hedibert F. Lopes
INSPER Institute of Education and Research
São Paulo, Brazil

[1]Slides based on Chapter 11 of Hastie, Tibshirani and Friedman's book *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd Edition)*.

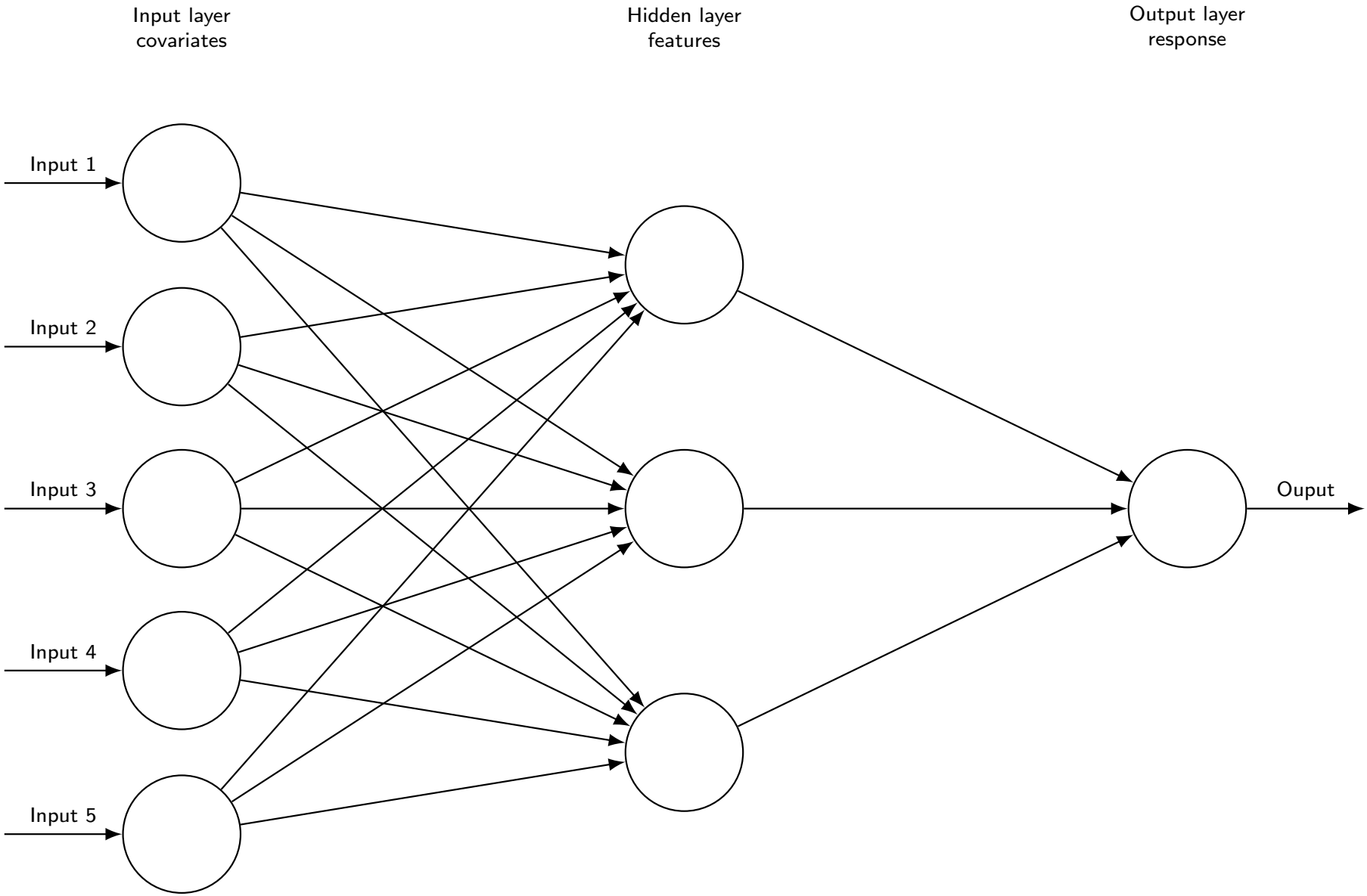
Chapter's Introduction

In this chapter we describe a class of learning methods that was developed separately in different field – statistics and artificial intelligence – based on essentially identical models.

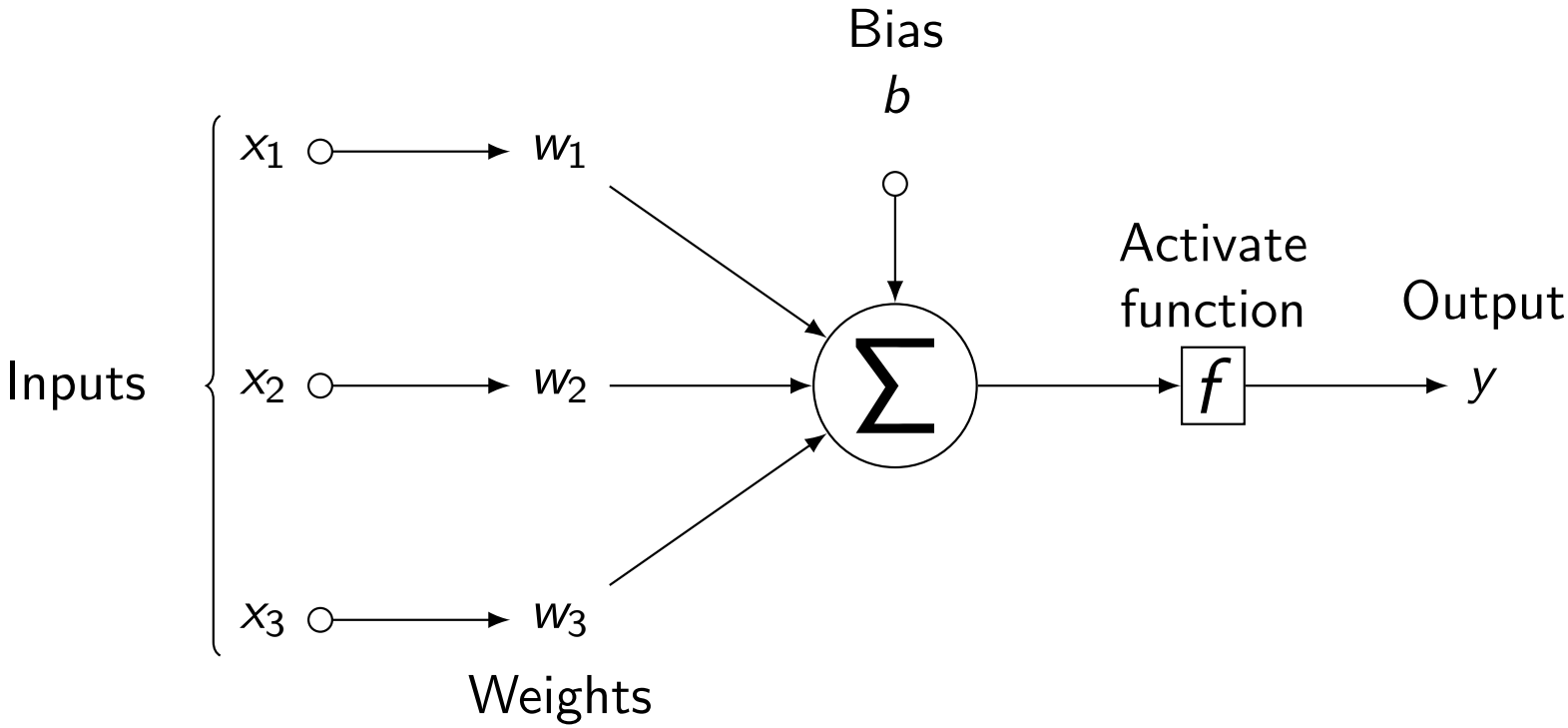
The central idea is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features.

The result is a powerful learning method, with widespread applications in many fields.

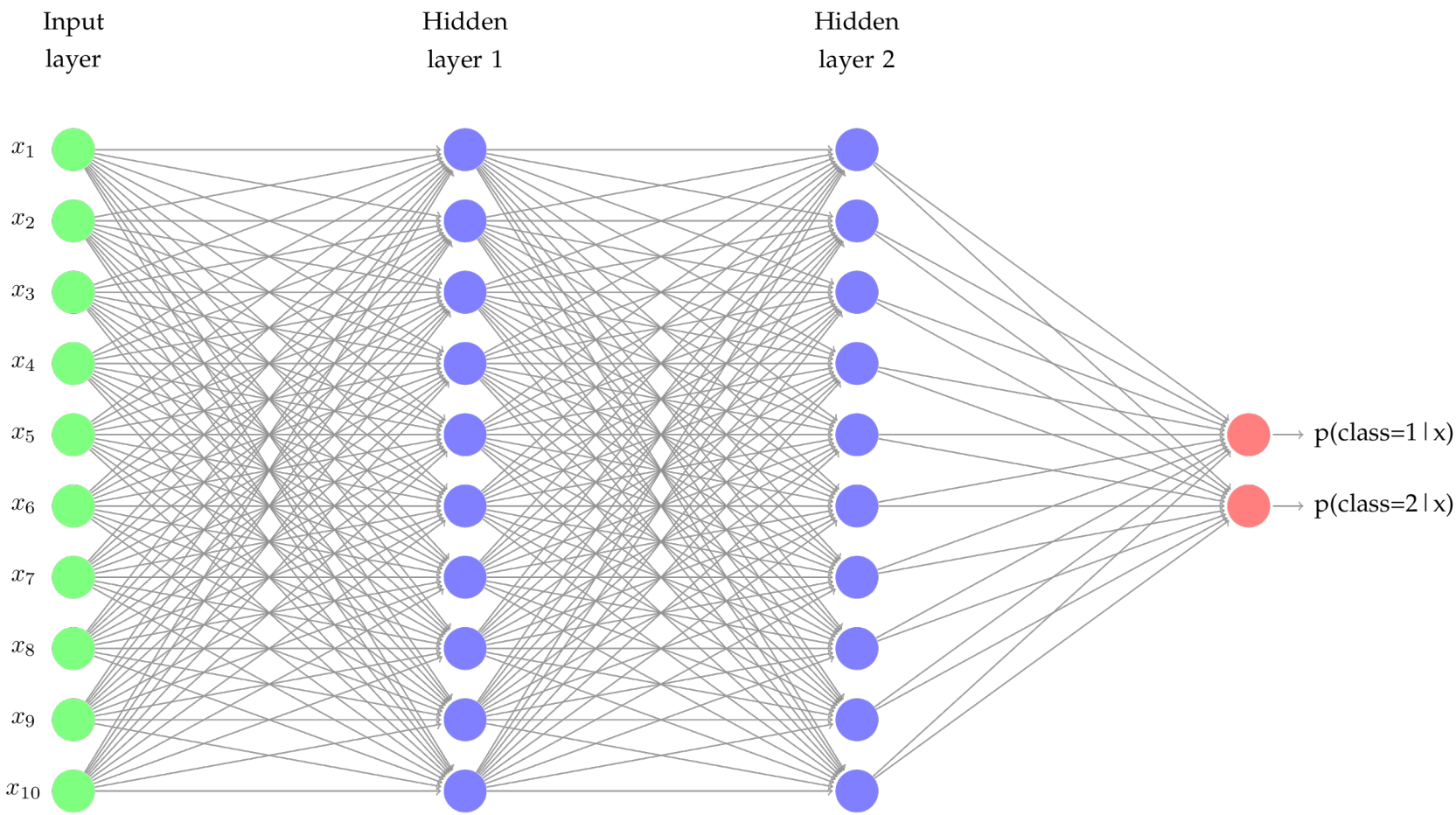
Single hidden layer, feed-forward neural network



Neural network



Neural network[2]



[2] From `http://mdtux89.github.io/2015/12/11/torch-tutorial.html`

Vanilla neural net

Single hidden layer back-propagation network or single layer perceptron.

A neural network is a two-stage regression or classification model.

Derived features z_m are created from linear combinations of the inputs, and then the target y (with K categories or K-dimensional responses) is modeled as a function of linear combinations of the z_m,

$$\begin{aligned} z_m &= \sigma(\alpha_{0m} + \alpha'_m x), && \text{for } m = 1, \ldots, M, \\ y_k &= \beta_{0k} + \beta'_k z, && \text{for } k = 1, \ldots, K, \\ f_k(x) &= g_k(y), \end{aligned}$$

where $x = (x_1, \ldots, x_p)$, $z = (z_1, \ldots, z_M)'$ and $y = (y_1, \ldots, y_K)$.

Activation function

The activation function $\sigma(v)$ is usually chosen to be the sigmoid

$$\sigma(v) = \frac{1}{1 + e^{-v}}.$$

Sometimes Gaussian radial basis functions are used for the $\sigma(v)$, producing what is known as a radial basis function network.

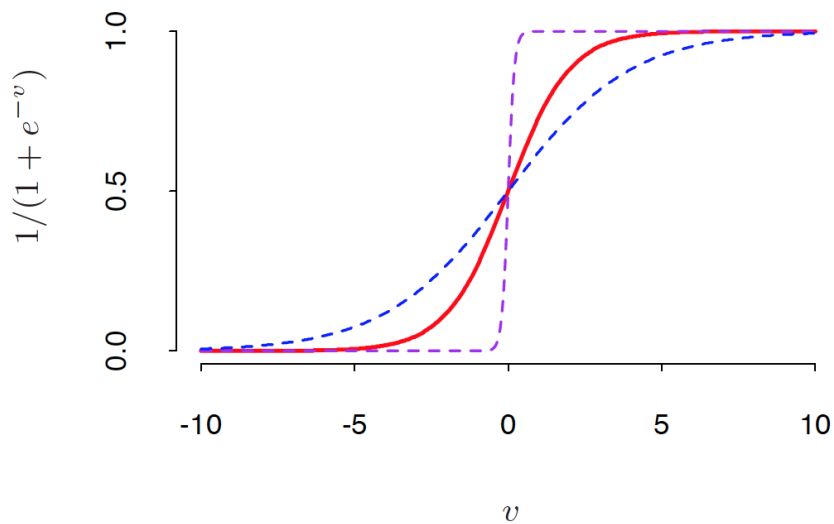


FIGURE 11.3. *Plot of the sigmoid function* $\sigma(v) = 1/(1+\exp(-v))$ *(red curve), commonly used in the hidden layer of a neural network. Included are* $\sigma(sv)$ *for* $s = \frac{1}{2}$ *(blue curve) and* $s = 10$ *(purple curve). The scale parameter* s *controls the activation rate, and we can see that large* s *amounts to a hard activation at* $v = 0$*. Note that* $\sigma(s(v - v_0))$ *shifts the activation threshold from* 0 *to* v_0*.*

Output function

The output function $g_k(y)$ allows a transformation of the vector of outputs y.

- For regression we typically choose the identity function $g_k(y) = y_k$.
- Early work in K-class classification also used the identity function, but this was later abandoned in favor of the softmax function:

$$g_k(y) = \frac{e^{y_k}}{\sum_{l=1}^{K} e^{y_l}}$$

Linear regression and classification:
Notice that if σ is the identity function, then the entire model collapses to a linear model in the inputs. Hence a neural network can be thought of as a nonlinear generalization of the linear model, both for regression and classification.

Weights

The neural network model has unknown parameters, often called weights, and we seek values for them that make the model fit the training data well.

We denote the complete set of weights by θ, which consists of

$$\{\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M\} \quad \text{and} \quad \{\beta_{0k}, \beta_k; k = 1, 2, \ldots, K\},$$

or $M(p+1)$ plus $K(M+1)$ weights!

Measures of fit

Regression - sum-of-squared errors

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Classification - cross-entropy

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i),$$

and the corresponding classifier is $G(x) = \text{argmax}_k f_k(x)$.

Back-propagation

With the softmax activation function and the cross-entropy error function, the neural network model is exactly a linear logistic regression model in the hidden units, and all the parameters are estimated by maximum likelihood.

The generic approach to minimizing $R(\theta)$ is by gradient descent, called back-propagation in this setting.

Because of the compositional form of the model, the gradient can be easily derived using the chain rule for differentiation.

This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit.

Back-propagation (part 1)

Here is back-propagation in detail for squared error loss. Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$, from (11.5) and let $z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})$. Then we have

$$\begin{aligned} R(\theta) &\equiv \sum_{i=1}^{N} R_i \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} - f_k(x_i))^2, \end{aligned} \tag{11.11}$$

with derivatives

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{m\ell}} &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{i\ell}. \end{aligned} \tag{11.12}$$

Back-propagation (part 2)

Given these derivatives, a gradient descent update at the $(r+1)$st iteration has the form

$$\begin{aligned}\beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\ \alpha_{m\ell}^{(r+1)} &= \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}},\end{aligned} \tag{11.13}$$

where γ_r is the *learning rate*, discussed below.

Now write (11.12) as

$$\begin{aligned}\frac{\partial R_i}{\partial \beta_{km}} &= \delta_{ki} z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{m\ell}} &= s_{mi} x_{i\ell}.\end{aligned} \tag{11.14}$$

Back-propagation (part 3)

The quantities δ_{ki} and s_{mi} are "errors" from the current model at the output and hidden layer units, respectively. From their definitions, these errors satisfy

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}, \tag{11.15}$$

known as the *back-propagation equations*. Using this, the updates in (11.13) can be implemented with a two-pass algorithm. In the *forward pass*, the current weights are fixed and the predicted values $\hat{f}_k(x_i)$ are computed from formula (11.5). In the *backward pass*, the errors δ_{ki} are computed, and then back-propagated via (11.15) to give the errors s_{mi}. Both sets of errors are then used to compute the gradients for the updates in (11.13), via (11.14).

The advantages of back-propagation are its simple, local nature. In the back propagation algorithm, each hidden unit passes and receives information only to and from units that share a connection. Hence it can be implemented efficiently on a parallel architecture computer.

Zip Code Data

This example is a character recognition task: classification of handwritten numerals scanned from envelopes by the U.S. Postal Service.

Input: 256 pixel values (16×16)
Output: digits $\{0, 1, 2, \ldots, 8, 9\}$
Training set: 320 digits
Test set: 160 digits

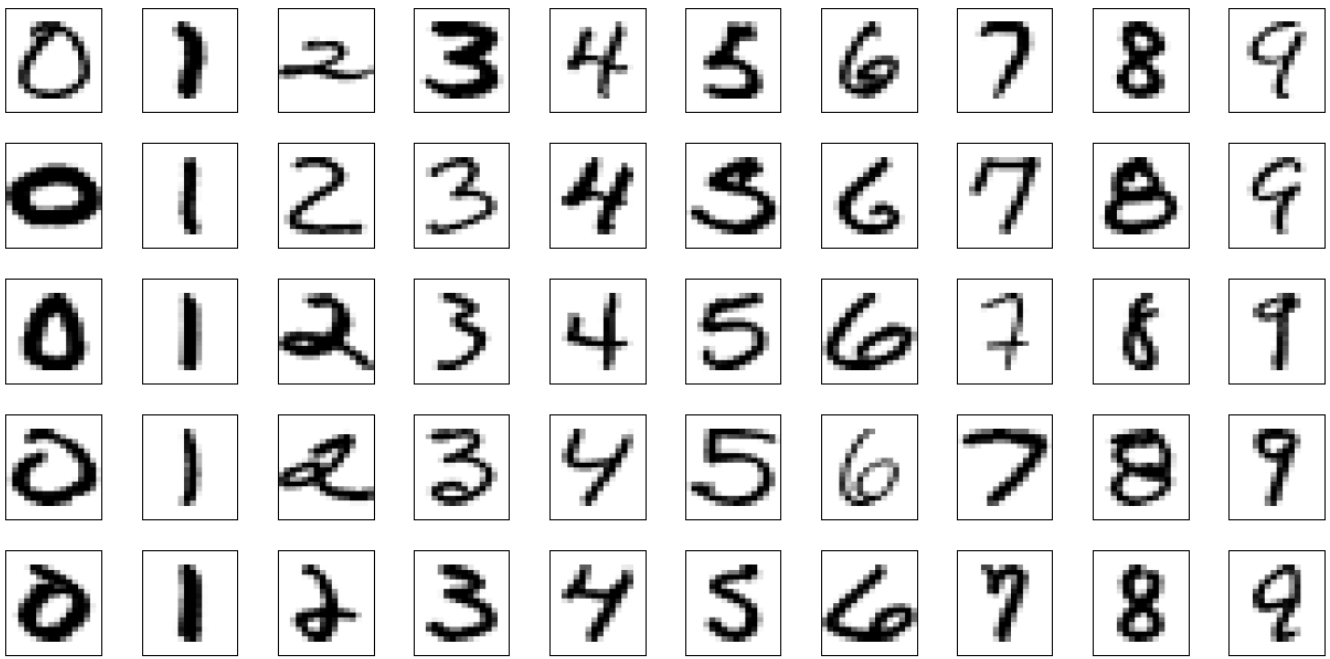


FIGURE 11.9. *Examples of training cases from ZIP code data. Each image is a* 16×16 *8-bit grayscale representation of a handwritten digit.*

Five different networks

Five different networks were fit to the data:

Net-1: No hidden layer, equivalent to multinomial logistic regression.

Net-2: One hidden layer, 12 hidden units fully connected.

Net-3: Two hidden layers locally connected.

Net-4: Two hidden layers, locally connected with weight sharing.

Net-5: Two hidden layers, locally connected, two levels of weight sharing.

Five different networks

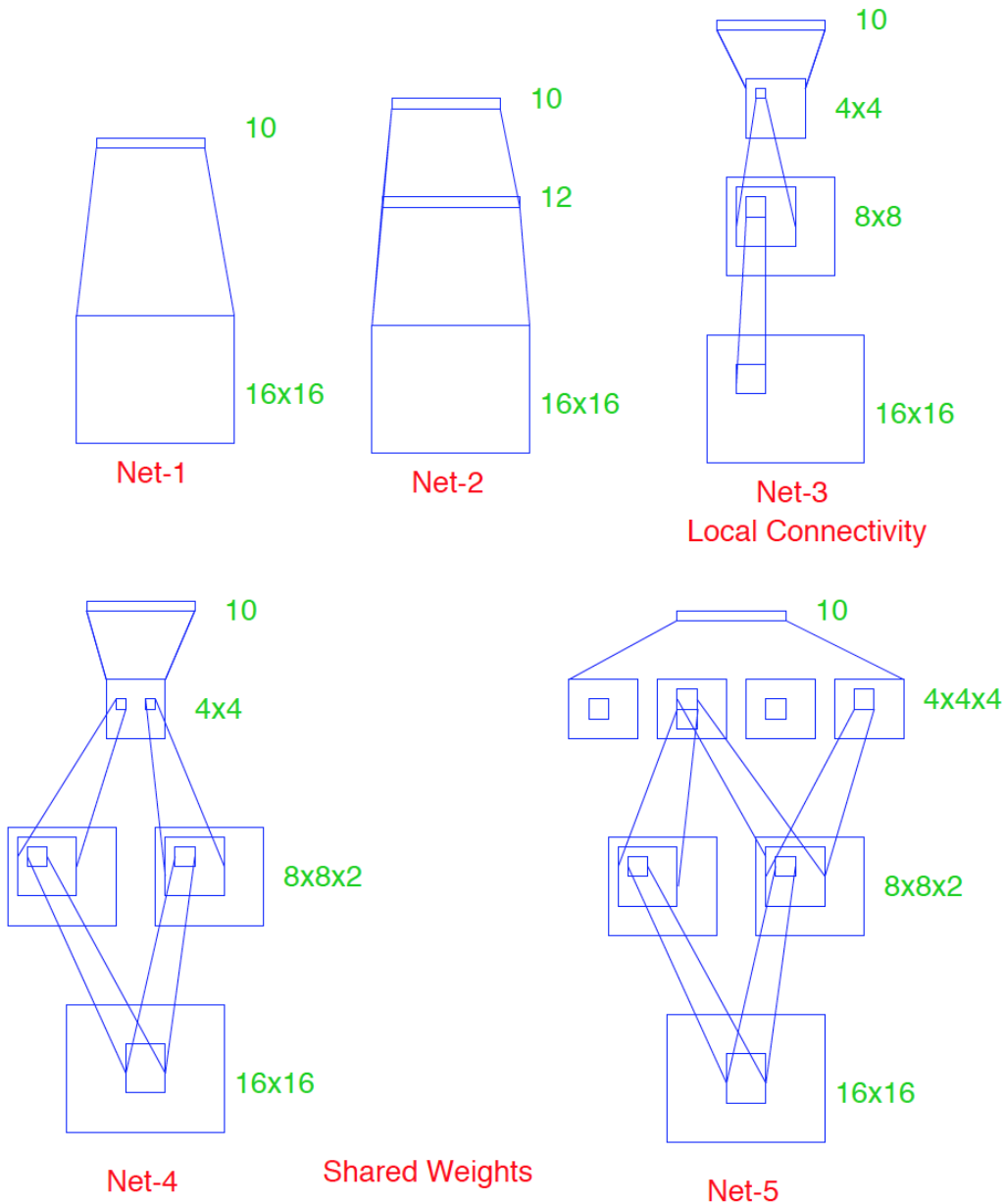


FIGURE 11.10. *Architecture of the five networks used in the ZIP code example.*

Test performance curves

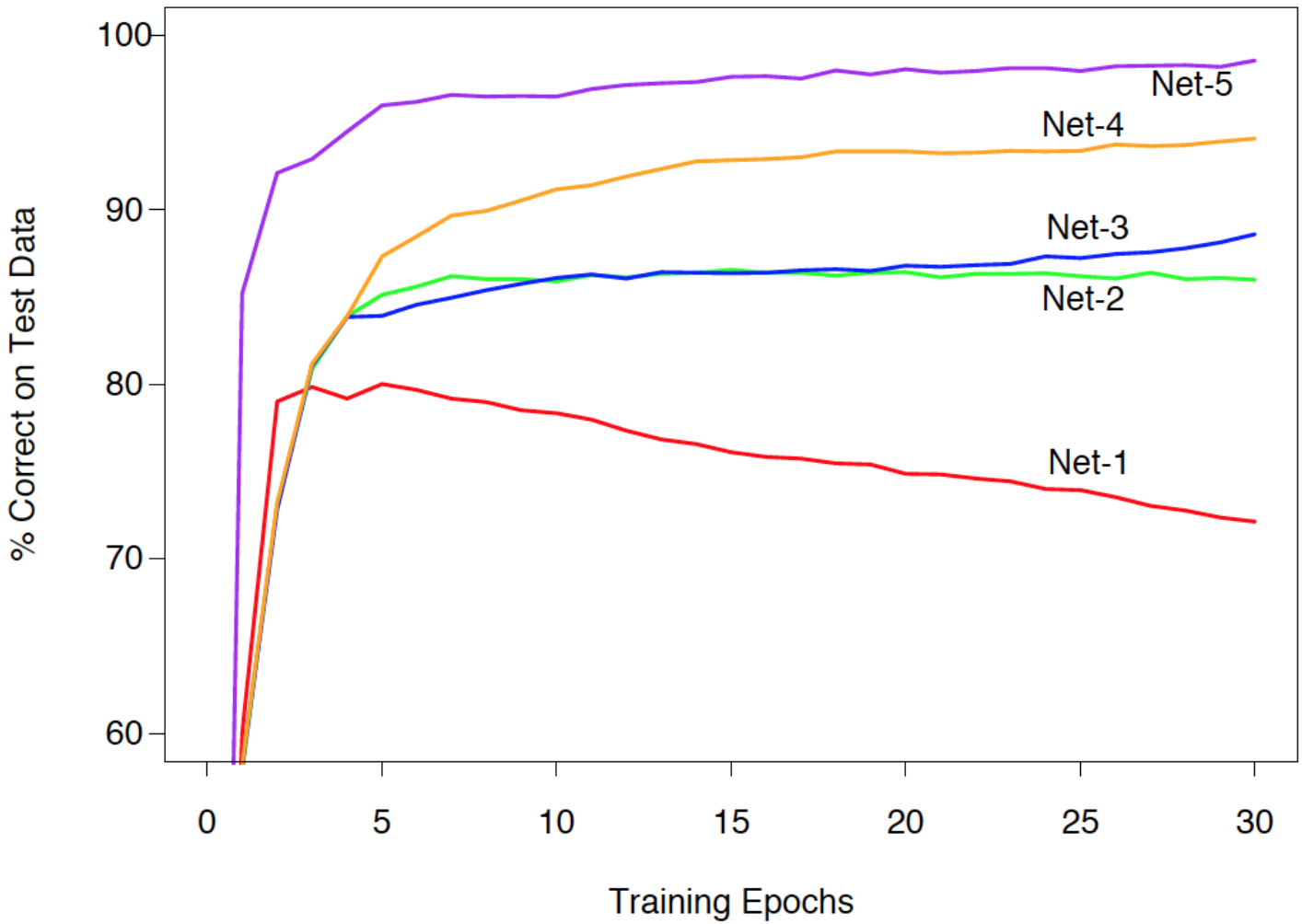


FIGURE 11.11. *Test performance curves, as a function of the number of training epochs, for the five networks of Table 11.1 applied to the ZIP code data. (Le Cun, 1989)*

Test performance

TABLE 11.1. *Test set performance of five different neural networks on a handwritten digit classification example (Le Cun, 1989).*

	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

Keras and RStudio

https://keras.rstudio.com
Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

dataset_mnist
From keras v2.1.6, by JJ Allaire
MNIST Database Of Handwritten Digits

Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images. Lists of training and test data: train$x, train$y, test$x, test$y, where x is an array of grayscale image data with shape (num_samples, 28, 28) and y is an array of digit labels (integers in range 0-9) with shape (num_samples).

R script

```
devtools::install_github("rstudio/keras")
library(keras)
install_keras()

mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# reshape
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
# rescale
x_train <- x_train / 255
x_test <- x_test / 255
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
summary(model)
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
history <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
plot(history)
model %>% evaluate(x_test, y_test)
model %>% predict_classes(x_test)
```