ST720 Data Science

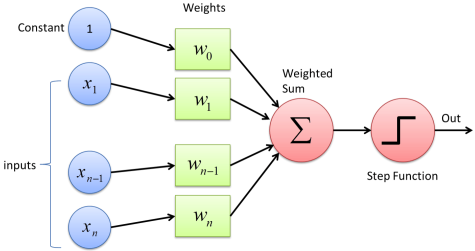
Neural Network and CNN

Seung Jun Shin (sjshin@krea.ac.kr)

Department of Statistics, Korea University

Perceptron

- Predict binary output (y) based on three binary inputs $\mathbf{x} = (x_1, \ldots, x_p)^T$.



- Perceptron predicts y:

$$\hat{y} = \begin{cases} 0 & \sum_j w_j x_j \leq t \\ 1 & \sum_j w_j x_j > t \end{cases}$$

where t denotes a threshold constant.

Perceptron

- Introducing a weight vector $\mathbf{w} = (w_1, \cdots, w_p)^T$, we have

$$\hat{y} = \begin{cases} 0 & \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1 & \mathbf{w}^T\mathbf{x} + b > 0 \end{cases} = \mathbb{1}\{\mathbf{w}^T\mathbf{x} + b > 0\}$$

- $\mathbf{w}$ and b can be estimated as

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \hat{y}_i)$$

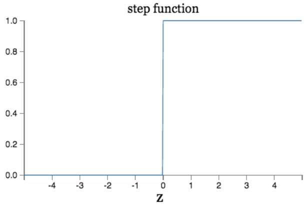
- Drawback: Small change in either $\mathbf{w}$ or b may result completely different prediction.

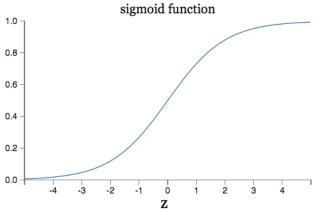
Sigmoid Function

- Instead of step function, consider

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

where $\sigma(z) = 1/(1 + e^{-z})$.





- Small change in either $\mathbf{w}$ or b always results little changes in prediction.

Perceptron cannot solve the XOR Problem

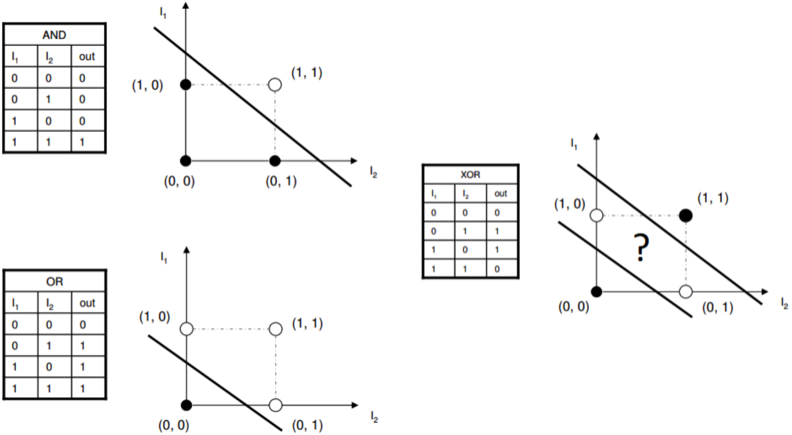


Figure 1: Perceptron cannot solve the XOR Problem.

Hiden Layer

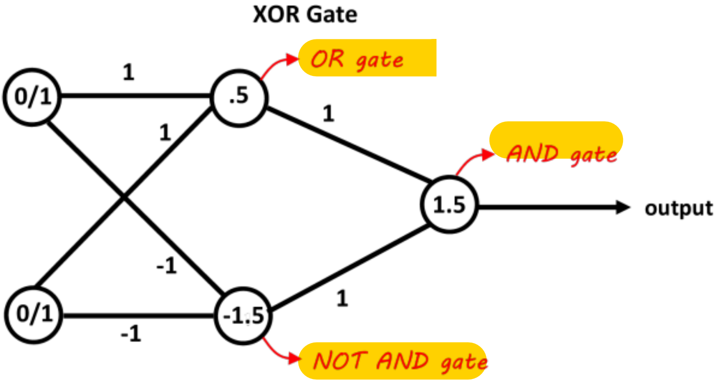
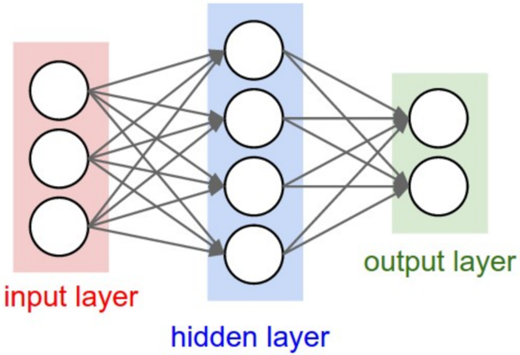


Figure 2: A Multi-layer Perceptron Solution for XOR problem.

Neural Network

- Neural Network (a.k.a Feedforward/Artificial Nerual Network) is a Multi Layered Perceptron.



Neural Network

- Consider K-class classification with one hot encoding for y_i.
  - $\mathbf{x} = (x_1, \cdots, x_p)$ (Input Layer)
  - $\mathbf{z}_1 = \{z_{1,m} : m = 1, \cdots, M\}$ (Hidden Layer)

    where
    $$z_{1,m} = \sigma(b_{m,1} + \mathbf{w}_{m,1}^T \mathbf{x})$$
  - $\mathbf{z}_2 = \{z_{2,k} : m = 1, \cdots, K\}$ (Output Layer)

    where
    $$z_{2,k} = b_{2,k} + \mathbf{w}_{2,k}^T \mathbf{z}_1$$
- Prediction Rule: (softmax function)
  $$\hat{P}(y = k) = \hat{P}(y_k = 1) = f_k(\mathbf{z}_2) = \frac{e^{z_{2,l}}}{\sum_{l=1}^K e^{z_{2,l}}}$$

Training NN

- We have to minimize the empirical risk:

$$R(\theta) = \sum_{i=1}^{n} R_i(\theta) = \sum_{i=1}^{n} \sum_{k=1}^{K} L(y_{ik}, f_k(\mathbf{x}_i))$$

  - Squared Loss: $L(y_{ik} - f_k(\mathbf{x}_i)) = (y_{ik} - f_k(\mathbf{x}_i))^2$
  - Deviance Loss: $L(y_{ik} - f_k(\mathbf{x}_i)) = y_{ik} \log f_k(\mathbf{x}_i)$

Training NN

- We have a plenty of parameters, $\{(p+1) \times M\} + (M+1) \times K\}$ in total as follows:

$$
\begin{aligned}
\mathbf{b}_1 &= \{b_{1,m} : m = 1, \cdots, M\}, \\
\mathbf{b}_2 &= \{b_{2,k} : k = 1, \cdots, K\}, \\
\mathbf{W}_{1,m} &= \{w_{1,ml} : m = 1, \cdots, M,\ \ l = 1, \cdots, p\} \in \mathbb{R}^{M \times p} \\
\mathbf{W}_{2,k} &= \{w_{2,km} : k = 1, \cdots, K,\ \ m = 1, \cdots, M\} \in \mathbb{R}^{K \times M}
\end{aligned}
$$

- This explains why neural networks even with moderately large number of layers are quite challenging.
- Then how to estimate these parameters?

Gradient Decent Algorithm

- Consider

$$R(\boldsymbol{\theta}) = \sum_{i=1}^{n} R_i(\boldsymbol{\theta}) = \sum_{i=1}^{n} \sum_{k=1}^{K} L(y_{ik}, f_k(\mathbf{x}_i))$$

  where $\boldsymbol{\theta} = (\mathbf{b}, \mathbf{W})^T$.

- Gradient Decent Algorithm updates the parameters as

$$\text{(Hidden)} \quad w_{1,ml}^{(t+1)} = w_{1,ml}^{(t)} - \gamma \sum_{i=1}^{n} \frac{\partial R_i(\boldsymbol{\theta})}{\partial w_{1,ml}} \bigg|_{w_{1,ml} = w_{1,ml}^{(t)}} \quad (1)$$

$$\text{(Ouput)} \quad w_{2,km}^{(t+1)} = w_{2,km}^{(t)} - \gamma \sum_{i=1}^{n} \frac{\partial R_i(\boldsymbol{\theta})}{\partial w_{2,km}} \bigg|_{w_{2,km} = w_{2,km}^{(t)}} \quad (2)$$

  where γ is learning rate.

- Stochastic GDA: Instead of picking all observations from $i = 1$ to n, randomly select observations at each iterations:

Back-Propagation: Compute Gradient

- Suppressing i, we have

$$\frac{\partial R}{\partial w_{2,km}} = \frac{\partial R}{\partial z_{2,k}} \times \frac{\partial z_{2,k}}{\partial w_{2,km}} = \delta_k \cdot z_{1,m}$$

$$\begin{aligned}
\frac{\partial R}{\partial w_{1,ml}} &= \frac{\partial R}{\partial \mathbf{z}_2} \times \frac{\partial \mathbf{z}_2}{\partial z_{1,m}} \times \frac{\partial z_{1,m}}{\partial w_{ml,1}} \\
&= \sum_{k=1}^{K} \left[\frac{\partial R}{\partial z_{2,k}} \times \frac{\partial z_{2,k}}{\partial z_{1,m}} \right] \times \frac{\partial z_{1,m}}{\partial w_{ml,1}} \\
&= \sum_{k=1}^{K} [\delta_k \cdot w_{2,km}] \times \sigma'(b_{1,0m} + \mathbf{w}_{1,m}^T \mathbf{x}) \cdot x_l = s_m \cdot x_l
\end{aligned}$$

Back-propagation equation

- By the definition of δ_k:

$$\delta_k = L'(y, f(\mathbf{x})) \times g'(b_{2,k} + \mathbf{w}_{2,k}^T \mathbf{z}) \tag{3}$$

- Back-propagation equation:

$$s_m = \sum_{k=1}^{K} [\delta_k \cdot w_{km}] \times \sigma'(b_{2,m} + \mathbf{w}_{2,m}^T \mathbf{x}) \tag{4}$$

GDA via Back-Propagation

1. Initialize $\boldsymbol{\theta}^{(0)}$.
2. For a given i, repeat steps 1–3 until convergence.
   2.1 (Forward pass) For a given $\boldsymbol{\theta}^{(t)} = (\mathbf{b}^{(t)}, \mathbf{W}^{(t)})^T$, compute $f(\mathbf{x}_i)$.
   2.2 (Backward pass) Compute δ_k from (3) and s_m from (4).
   2.3 Update $\boldsymbol{\theta}^{(t+1)}$ via GDA equations (1) and (2) with learning rate γ.

Cautions

- NN have too many weights and will overfit the data.
- Early stopping rule is a reasonable option, but returns (nearly) linear model.
- Add penalty to the risk function

$$R(\boldsymbol{\theta}) + \lambda J(\theta)$$

  - Weight Decay (Ridge) penalty

$$J(\boldsymbol{\theta}) = \sum_{km} w_{2,km}^2 + \sum_{ml} w_{1,ml}^2$$

  - Weight Elimination penalty

$$J(\boldsymbol{\theta}) = \sum_{km} \frac{w_{2,km}^2}{1 + w_{2,km}^2} + \sum_{ml} \frac{w_{1,ml}^2}{1 + w_{1,ml}^2}$$

    Shrink smaller weights toward 0 than the ridge penalty does.

Cautions

- Scaling of Input Variables
  - A large effect on the quality of the output.
  - Standardize the input variables.
- Initial Weights
  - Sensitive to the choice of initial value of $\boldsymbol{\theta}^{(0)}$.
  - $\boldsymbol{\theta}^{(0)} \sim \mathit{Uniform}(-0.7, 0.7)$.

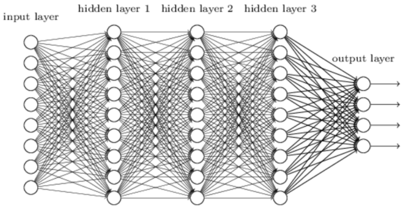
Cautions

- Number of Hidden Units and Layers.
  - One layer is good enough.
  - For the number of hidden units M, set large number and employ penalty.
  - $5 \leq M \leq 100$

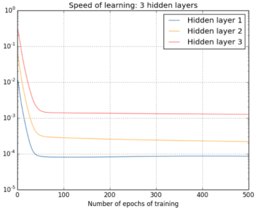
- Multiple minima
  - $R(\boldsymbol{\theta})$ is not convex, possessing multiple local minima.
  - Final solutions are heavily dependent on the choice of starting values.
  - One must try multiple NN with different initial values and choose the solution giving lowest error.
  - Bagging would be one choice.

Problems on Training Deep Network

- The unstable gradient problem
  - Vanishing gradient
  - Exploding gradient problem



(a) Deep network



(b) learning speed

Convolutional Neural Network

- Proposed by Lecun (1998).
- Most popular Deep learning method.
- Key idea is the use convolution operator.

Convolution

- Suppose we have two functions $f(t)$ and $w(t)$.
- The convolution of $f(t)$ and $w(t)$ denoted by $(f * w)(t)$ is given by

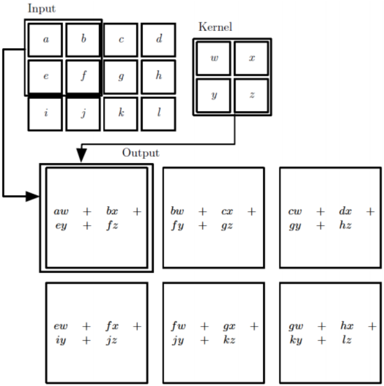
$$(f * w)(t) = \int f(s)w(t-s)ds = \int f(t-s)w(s)ds$$

- Convolution can be viwed as a weighted average of the function values of $f(s)$ at $s \in (t-s, t+s)$ with weight g.

Convolution as a Filter/Kernel of Image

- Extension to the descrete and mutidimensional functionss is straightforward:

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(i,j)w(i-m, j-n)$$

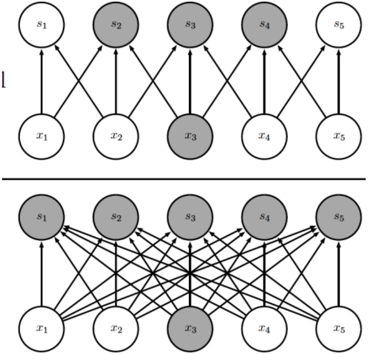


Convolutional Neural Network

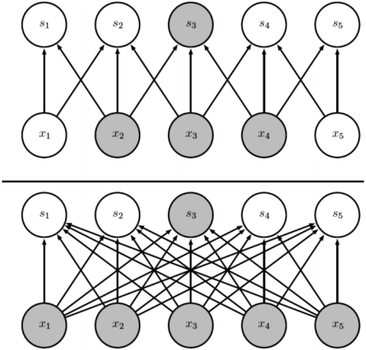
- Convolution leverages three important ideas:
  - Sparse interactions
  - parameter sharing
  - equiariant representation

Effect of Convolution: Sparse Interactions

- In the illustrative previous example, only a part of input is used to compute the value of each node of the following layer.



(c) From below



(d) From above

Effect of Convolution: Sparse Interactions

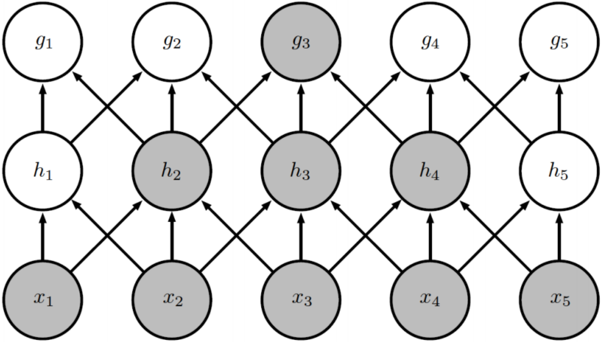


Figure 3: The receptive field of the units in the deepper layers of a convolutionl network is larger than the receptive field of the units in the shallow layers.

Effect of Convolution: Parameter Sharing

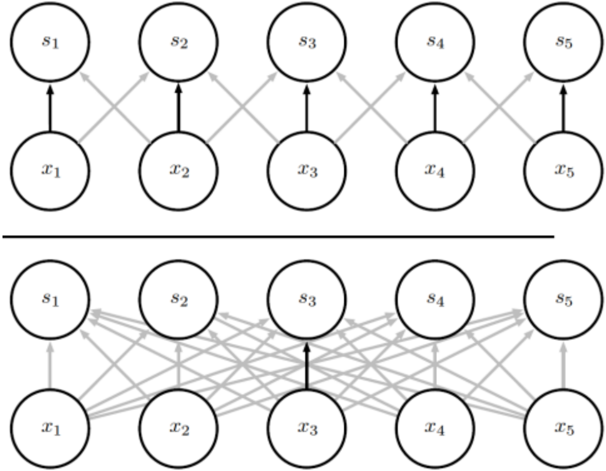
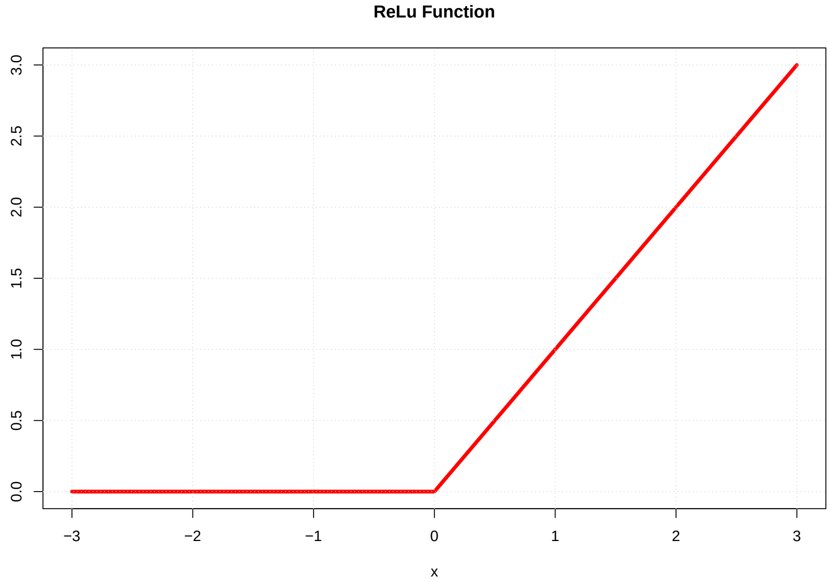


Figure 4: Connections with same color share parameters.

Effect of Convolution: Equiariant Representation

- Convolution operator is equivariant to the translation
  - i.e., shifting the input and applying convolution is equivalent to applying convolution to the input and shifting it.
  - If we move the object in the input, its representation will move the same amount in the output

ReLu Activation function



Pooling

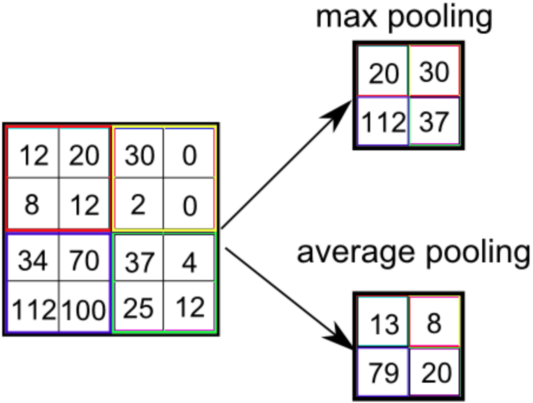


Figure 5: Pooling with stride 2.

Convolutional Neural Network

- Convolution, ReRu Activation, Pooling forms a one CNN layer.

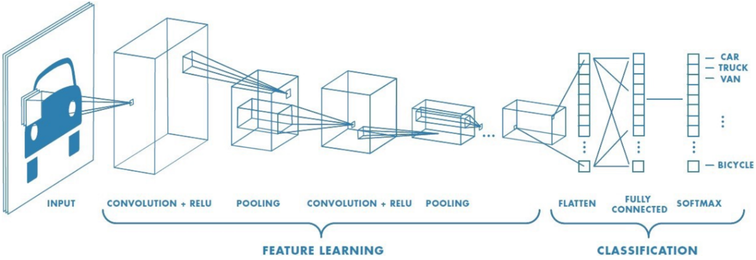


Figure 6: Simplified Illustration of the convolutional neural network

Additional Concepts You should Know

- Batch Normalization
- Dropout

CNN using R

- The R interface to TensorFlow using Keras is avaiable.

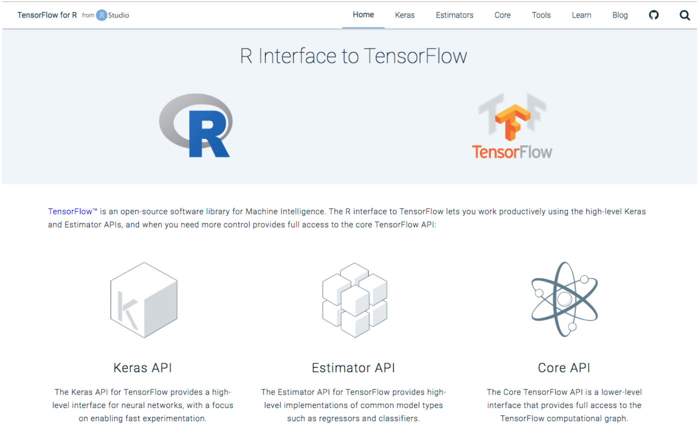


Figure 7: https://tensorflow.rstudio.com/

CNN using R

- Installation

```
library(keras)
install_keras()
```

```
##
## Installation complete.
```

- Download MNIST data

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

```
##  int [1:60000, 1:28, 1:28] 0 0 0 0 0 0 0 0 0 0 ...
```

```
str(y_train)
```

```
##  int [1:60000(1d)] 5 0 4 1 9 2 1 3 1 4 ...
```

CNN using R: Data Transformation

- 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
```

- one-hot-encoded y.

```
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

CNN using R: Model

- Set models

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu', input_shape = c(
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
summary(model)
```

CNN using R: Model

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                            Output Shape
## ================================================================================
## dense (Dense)                           (None, 256)
## ________________________________________________________________________________
## dropout (Dropout)                       (None, 256)
## ________________________________________________________________________________
## dense_1 (Dense)                         (None, 128)
## ________________________________________________________________________________
## dropout_1 (Dropout)                     (None, 128)
## ________________________________________________________________________________
## dense_2 (Dense)                         (None, 10)
## ================================================================================
## Total params: 235,146
## Trainable params: 235,146
## Non-trainable params: 0
## ________________________________________________________________________________
```

CNN using R: Model

- Compile the model with appropriate loss function, optimizer, and metrics:

```
model %>% compile(
loss = 'categorical_crossentropy',
optimizer = optimizer_rmsprop(),
metrics = c('accuracy')
)
```

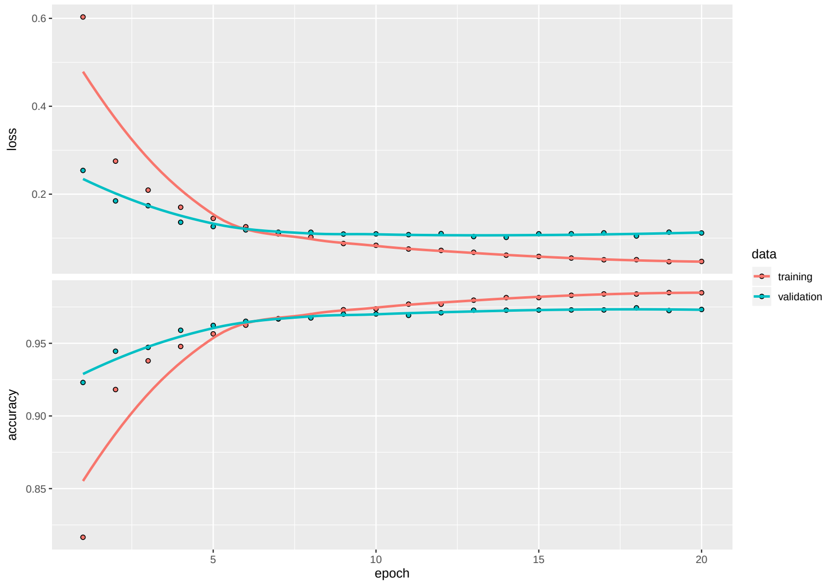
CNN using R: Training and Evaluation

- Use the `fit()` function to train the model for 30 epochs using batches of 128 images:

```
history <- model %>% fit(
x_train, y_train,
epochs = 20, batch_size = 256,
validation_split = 0.5
)
```

CNN using R: Training and Evaluation

```
plot(history)
```



CNN using R: Training and Evaluation

```r
model %>% evaluate(x_test, y_test)
```

```
## $loss
## [1] 0.09299823
##
## $accuracy
## [1] 0.9755
```