Deep Learning Lecture 1 - Basic components

MA8701 General Statistical Methods

Thiago G. Martins, Department of Mathematical Sciences, NTNU

Spring 2019

Tensor: Data representation for neural networks

Currently, most (if not all) current ML systems use tensors as their basic data structure.

Tensors are a generalisation of vectors and matrices to an arbitrary number of dimensions.

- In the context of tensor, a dimension is often called an axis or rank

A tensor is defined by 3 key attributes:

- Number of axis (rank or dimension)
- Shape (how many dimensions along each axis)
- Data type

Abstract examples

Scalars (0D tensors)

- R doesn't have a data type to represent scalars. All numeric objects are vectors, matrices, or arrays.
- But an R vector that is always length 1 is conceptually similar to a scalar.

```
# scalar
tensor <- as.array(1)
```

```
# rank
length(dim(tensor))
```

```
## [1] 1
```

```
# shape
dim(tensor)
```

```
## [1] 1
```

```
# data type
typeof(tensor)
```

```
## [1] "double"
```

Vectors (1D tensors)

- Don't confuse a 5D vector with a 5D tensor!

```
# vector
tensor <- as.array(c(12, 3, 6, 14, 10))
```

```
# rank
length(dim(tensor))
```

```
## [1] 1
```

```
# shape
dim(tensor)
```

```
## [1] 5
```

```
# data type
typeof(tensor)
```

```
## [1] "double"
```

Matrices (2D tensors)

- A matrix has two axes (often referred to as rows and columns)

```
tensor <- matrix(rep(0, 3*5), nrow = 3, ncol = 5)
```

```
# rank
length(dim(tensor))
```

```
## [1] 2
```

```
# shape
dim(tensor)
```

```
## [1] 3 5
```

```
# data type
typeof(tensor)
```

```
## [1] "double"
```

3D tensors and higher-dimensional tensors

```
tensor <- array(rep(0, 2*3*2), dim = c(2,3,2))
```

```
# rank
length(dim(tensor))
```

```
## [1] 3
```

```
# shape
dim(tensor)
```

```
## [1] 2 3 2
```

```
# data type
typeof(tensor)
```

```
## [1] "double"
```

Concrete data examples

Vector data

- 2D tensor of shape `(samples, features)`
- Data structure most commonly used by statistical packages
- Usually, this is the expected data structure when fitting fully connected neural networks
- Example: The Boston housing price dataset

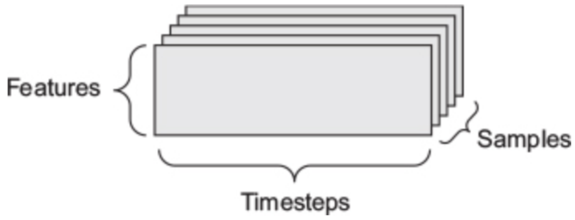
```
str(boston_train_data)
```

```
##  num [1:404, 1:13] 1.2325 0.0218 4.8982 0.0396 3.6931 ...
```

- More complex data types can be mapped to vector data format.
  - Image data: Usually represented in 3D tensors can be flatened into a 2D tensor

Timeseries or sequence data

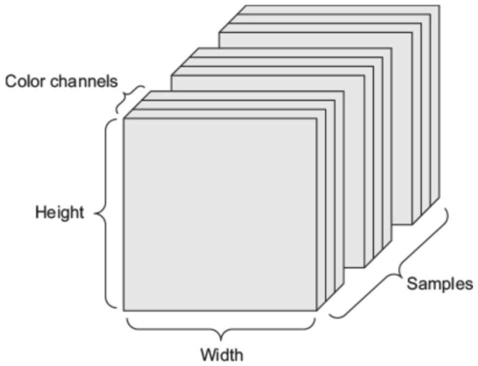
- 3D tensor of shape `(samples, timesteps, features)`



- Stock price dataset
  - Each minute we record the current price, last minute lowest price and highest price
  - A trading day has 390 minutes and a trading year has 250 days
  - One year of data can be store in a 3D tensor `(250, 390, 3)`
- A statistician would usually store in a 2D tensor with a date column.

Image data

- 4D tensor of shape `(samples, height, width, channels)`



- Grayscale images (like our MNIST digits) have only a single color channel, but are still stored in 4D tensors by convention.

Note: Unlike TensorFlow, Theano uses the convention
`(samples, channels, height, width)`

Video data

- 5D tensor of shape `(samples, frames, height, width, channels)`
- A video can be understood as a sequence of frames, each frame being a color image.

Tensor operations

- All transformations learned by deep neural networks can be reduced to a handful of tensor operations applied to tensors of numeric data
- Example: Dense layer

```
output = relu(dot(W, input) + b)
```

- Element-wise operation: `relu(x) = max(x,0)`
- Addition between a 2D tensor and a 1D tensor
- Dot product between `W` and `input`

Element-wise operations

- Operations that are applied independently to each entry in the tensors being considered
- Massively parallel implementations available (vectorised implementations)

Operations involving tensors of different dimensions

- The R `sweep()` function enables you to perform operations between higher-dimension tensors and lower-dimension tensors

```
# Add vector y for each column of matrix x
sweep(x, 2, y, `+`)
```

Tensor dot

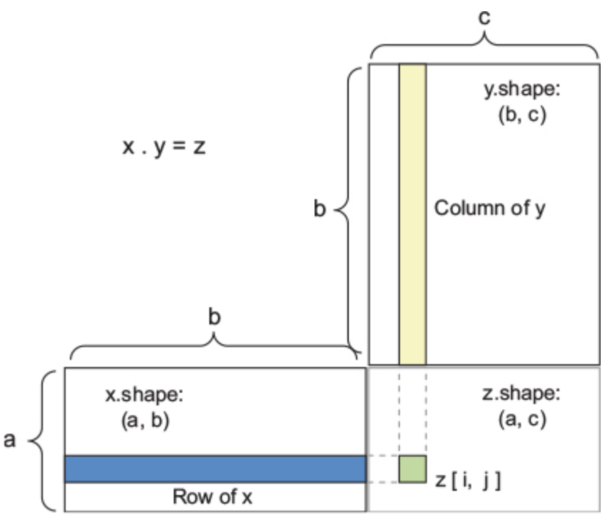
- Most common and useful tensor operation
- An element-wise product is done with the `*` operator in R, whereas dot products use the `%*%` operator
- Mathematically, what does the dot operation do?

```
# dot product between two vector of the same length
naive_vector_dot <- function(x, y) {
  z <- 0
  for (i in 1:length(x))
```

```
    z <- z + x[[i]] * y[[i]]
  z
}
```

```
# dot product between a matrix x and a vector y
naive_matrix_vector_dot <- function(x, y) {
  z <- rep(0, nrow(x))
  for (i in 1:nrow(x))
    row_x <- x[i,]
    z[[i]] <- naive_vector_dot(row_x, y)
  z
}
```

```
# dot product between a matrix x and a matrix y
naive_matrix_dot <- function(x, y) {                  1
  z <- matrix(0, nrow = nrow(x), ncol = ncol(y))
  for (i in 1:nrow(x))
    for (j in 1:ncol(y)) {
      row_x <- x[i,]
      column_y <- y[,j]
      z[i, j] <- naive_vector_dot(row_x, column_y)
    }
  z
}
```



- The dot product generalizes for higher dimensions, given the apropriate shape compatibility observed before:
  - (a, b, c, d) . (d) -> (a, b, c)
  - (a, b, c, d) . (d, e) -> (a, b, c, e)
  - And so on.

Tensor reshaping

- Always use the `array_reshape()` function when reshaping R arrays that will be passed to Keras.
  - Uses row-major semantics (as opposed to R's default column-major semantics)
  - Compatible with the way the numerical libraries called by Keras (NumPy, TensorFlow, and so on) interpret array dimensions.

```r
x <- matrix(c(0, 1,
              2, 3,
              4, 5),
            nrow = 3, ncol = 2, byrow = TRUE)
x
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    2    3
## [3,]    4    5
```

```r
# R uses column-major semantics by default
dim(x) <- c(6,1)
x
```

```
##      [,1]
## [1,]    0
## [2,]    2
## [3,]    4
## [4,]    1
## [5,]    3
## [6,]    5
```

```r
x <- matrix(c(0, 1,
              2, 3,
              4, 5),
            nrow = 3, ncol = 2, byrow = TRUE)
x
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]    2    3
## [3,]    4    5
```

```r
# array_reshape uses row-major semantics
x <- array_reshape(x, dim = c(6, 1))
x
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    2
## [4,]    3
## [5,]    4
## [6,]    5
```

Reference material

This lecture note is based on (Chollet and Allaire 2018).

References

Chollet, F., and J. Allaire. 2018. *Deep Learning with R*. Manning Publications. https://books.google.no/books?id=xnIRtAEACAAJ.