Module 11: Deep Learning and Neural Networks

TMA4268 Statistical Learning V2023

Stefanie Muff, Department of Mathematical Sciences, NTNU

April 24 and 27, 2023

Acknowledgements

- Some of this material was (in a modified version) created by Mette Langaas who has put a lot of effort in creating this module in its original version. Thanks to Mette for the permission to use the material!
- Some of the figures and slides in this presentation are taken (or are inspired) from James et al. (2021).

Learning material for this module

- James et al (2021): An Introduction to Statistical Learning. Chapter 10.
- All the material presented on these module slides and in class.
- Videos on neural networsk and back propagation
  - Video 1
  - Video 2
  - Video 3
  - Video 4

Secondary material (not compulsory):

- Background material: Chapters 6-8 Goodfellow, Bengio, and Courville (2016) https://www.deeplearningbook.org

See also *References and further reading* (last slide), for further reading material.

What will you learn?

Todo: Update

- Deep learning: The timeline
- Single and multilayer neural networks
- Convolutional neural networks
- Recurrent neural networks
- Interpolation and double descent

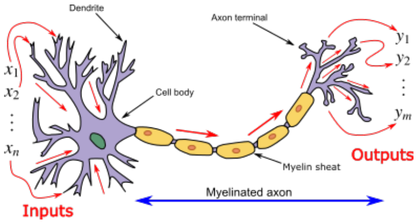
- Neural network parts: model – method – algorithm – recent developents

Introduction: Time line

- 1950's: First neural networks (NN) in "toy form".
- 1980s: the backpropagation algorithm was rediscovered.
- 1989: (Bell Labs, Yann LeCun) used convolutional neural networks to classifying handwritten digits.
- 2000s: After the first hype, NNs were pushed aside by boosting and support vector machines in the 2000s.
- Since 2010: Revival! The emergence of *Deep learning* as a consequence of improved computer resources, some innovations, and applications to image and video classification, and speech and text processing

- Shift from statistics to computer science and machine learning, as they are highly parameterized.

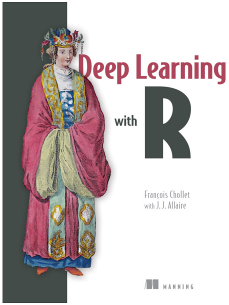
- Statisticians were skeptical: “It’s just a nonlinear model”.

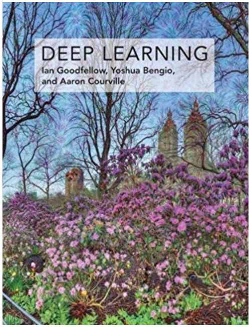


Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals.

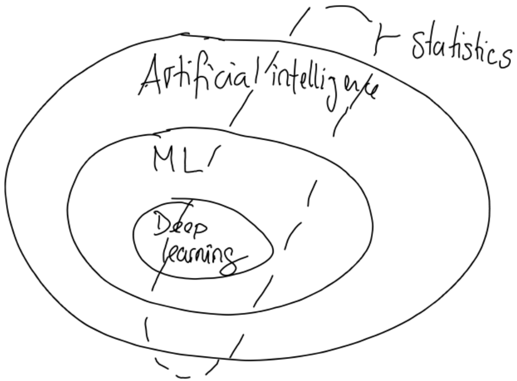
Image credits: By Egm4313.s12 (Prof. Loc Vu-Quoc)
https://commons.wikimedia.org/w/index.php?curid=72816083

- There are several learning resources (some listed under 'further references') that you my turn to for further knowledge into deep learning.
- There is a new IT3030 deep learning course at NTNU.





AI, machine learning and statistics



AI

- Artificial intelligence (AI) dates back to the 1950s, and can be seen as *the effort to automate intellectual tasks normally performed by humans* (page 4, Chollet and Allaire (2018)).

- AI was first based on hardcoded rules (like in chess programs), but turned out to be intractable for solving more complex, fuzzy problems.

Machine learning

- With the field of *machine learning* the shift is that a system is *trained* rather than explicitly programmed.
- Machine learning is related to mathematical statistics, but differs in many ways.
- ML deals with much larger and more complex data sets than what is usually done in statistics.
- The focus in ML is oriented towards *engineering*, and ideas are proven *empirically* rather than theoretically (which is the case in mathematical statistics).

According to Chollet and Allaire (2018) (page 19):

Machine learning isn't mathematics or physics, where major advancements can be done with a pen and a piece of paper. It's an engineering science.

Deep learning

Deep Learning is an algorithm which has no theoretical limitations of what it can learn; the more data you give and the more computational time you provide, the better it is. Geoffrey Hinton (Google)

- *Deep* does not refer to a deeper understanding.
- Rather, deep referes to the *layers of representation*, for example in a neural network.

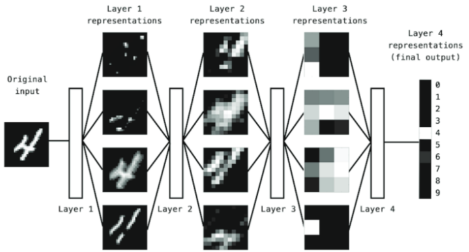


Figure 1.6 Deep representations learned by a digit classification model

- In 2011 neural networks with many layers (and trained with GPUs) were performing well on image classification tasks.

- The *ImageNet* classification challenge (classify high resolution colour images into 1k different categories after training on 1.4M images) was won by solutions with deep convolutional neural networks (convnets). In 2011 the accuracy was 74.3%, in 2012 83.6% and in 2015 96.4%.

- From 2012, convnets is the general solution for computer vision tasks. Other application areas are natural language processing.

Deep?

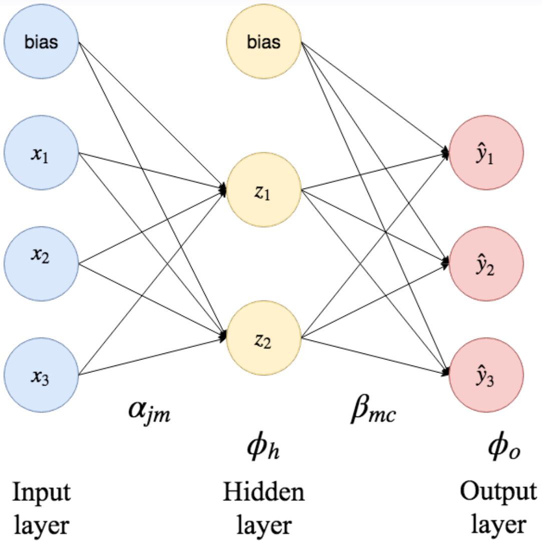
- Deep learning does not mean a deeper understanding, but refers to sucessive layers of representations - where the number of layers gives the *depth* of the model. Often tens to hundreds of layers are used.
- Deep neural networks are not seen as models of the brain, and are not related to neurobiology.
- A deep network can be seen as many stages of *information-destillation*, where each stage performes a simple data transformation. These transformations are not curated by the data analyst, but is estimated in the network.
- In contrast, in statistics we first select a set of inputs, then look at how these inputs should be transformed, before we apply some statistical methods (*feature engineering*).

- The success of deep learning is dependent upon the breakthroughts
  - in *hardware* development, expecially with faster CPUs and massively parallell graphical processing units (GPUs).
  - *datasets* and benchmarks (internet/tech data).
  - improvemets of the *algorithms*.

- Achievements of deep learning includes high quality (near-human to super human) image classification, speech recognition, handwriting transcription, machine translation, digital assistants, autonomous driving, advertise targeting, web searches, playing Go and chess.

Feedforward networks

- Connections are only forward in the network, but no feedback connections that sends the output of the model back into the network.
- Examples: Linear, logistic and multinomial regression with or without any *hidden layers* (between the input and output layers).
- We may have between zero and very many hidden layers.
- Adding *hidden layers* with *non-linear activation functions* between the input and output layer will make nonlinear statistical models.
- The number of hidden layers is called the *depth* of the network, and the number of nodes in a layer is called the *width* of the layer.


bias
x_1
x_2
x_3
bias
z_1
z_2
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
α_{jm}
β_{mc}
ϕ_h
ϕ_o
Input layer
Hidden layer
Output layer

The single hidden layer feedforward network

The nodes are also called *neurons*.

Notation

1. Inputs: p input layer nodes $x^\top = (x_1, x_2, \dots, x_p)$.
2. The nodes z_m in the hidden layer, $m = 1, \dots, M$; as vector $z^\top = (z_1, \dots, z_M)$, and the hidden layer activation function $g()$.

$$z_m(x) = g(\alpha_{0m} + \sum_{j=1}^{p} \alpha_{jm} x_j)$$

where α_{jm} is the weight[1] from input j to hidden node m, and α_{0m} is the bias term for the mth hidden node.

[1] We stick with greek letters α and β for parameters, but call them weights.

3. The node(s) in the output layer, $c = 1, \dots C$: $y_1, y_2, \dots, y_C$, or as vector y, and output layer activation function $f()$.

$$\hat{y}_c(x) = f(\beta_{0c} + \sum_{m=1}^{M} \beta_{mc} z_m(x))$$

where β_{mc} is from hidden neuron m to ouput node c, and β_{0c} is the bias term for the cth output node.

4. Taken together

$$\hat{y}_c(x) = f(\beta_{0c} + \sum_{m=1}^{M} \beta_{mc} z_m) = f(\beta_{0c} + \sum_{m=1}^{M} \beta_{mc} g(\alpha_{0m} + \sum_{j=1}^{p} \alpha_{jm} x_j))$$

Hands on:

- Identify p, M, C in the network figure above, and relate that to the $y_c(x)$ equation.
- How many parameters (the α and βs) need to be estimated for this network?
- What determines the values of p and C?
- How is M determined?

Special case: linear activation function for the hidden layer

If we assume that $g(z) = z$ (linear or identity activiation):

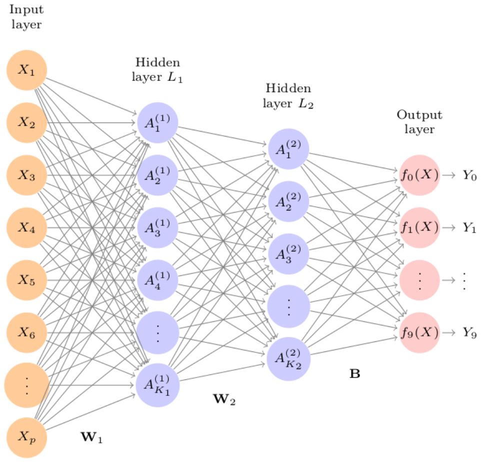
$$\hat{y}_c(x) = f(\beta_{0c} + \sum_{m=1}^{M} \beta_{mc}(\alpha_{0m} + \sum_{j=1}^{p} \alpha_{jm} x_j))$$

Q: Does this look like something you have seen before?

A:

Multilayer neural networks

Alternative: networks with more than one hidden layer, but fewer total number of nodes but more layers. A network with *many hidden layers* is called a *deep network*.



(Fig 10.4 James et al. (2021))

- The idea of the multilayer NN is exactly the same as for the single-layer version.

- $f_m(X)$ is a (transformation of) the linear combination of the last layer.

Outcome encoding

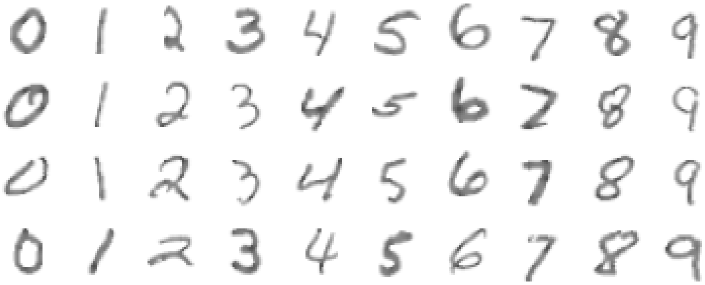
- *Continuous* and *binary* may only have one output node ($y_i =$ observed value).

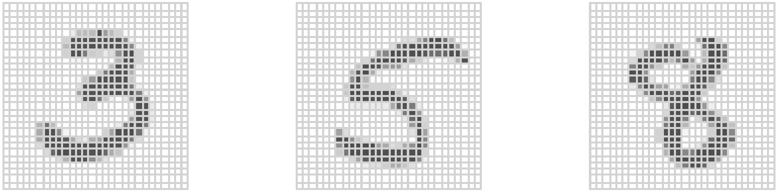
- For C *categories*, we have C output nodes, where we encode the output as $Y = (Y_1, Y_2, \dots, Y_C)$, where $y_i = (0, 0, \dots, 0, 1, 0, \dots, 0)$ with a value of 1 in the c^{th} element of y_i if the class is c. This is called *one-hot encoding* or *dummy encoding*.

Example: MNIST dataset

- Aim: Classification of handwritten digits.
- Categorical outcome $C = 0, 1, \ldots, 9$.
- This data is based on https://www.math.ntnu.no/emner/TMA4268/2018v/11NN/8-neural_networks_mnist.html and the `R` `keras` cheat sheet.

Objective: classify the digit contained in an image (128 $\times$ 128 greyscale).





Neural network parts

We now focus on the different elements of neural networks.

1) Output layer activation
2) Hidden layer activation
3) Network architecture
4) Loss function
5) Optimizers

1) Output layer activation

These choices have been guided by solutions in statistics (multiple linear regression, logistic regression, multiclass regression)

- *Linear activation*: for *continuous outcome* (regression problems)

$$f(X) = X \ .$$

- *Sigmoid activation*: for *binary outcome* (two-class classification problems)

$$f(X) = \frac{1}{1 + \exp(-X)} \ .$$

- *Softmax*: for *multinomial/categorical outcome* (multi-class classification problems)

$$f_m(X) = \Pr(Y = m|X) = \frac{\exp(Z_m)}{\sum_{s=1}^{C} \exp(Z_s)} \ .$$

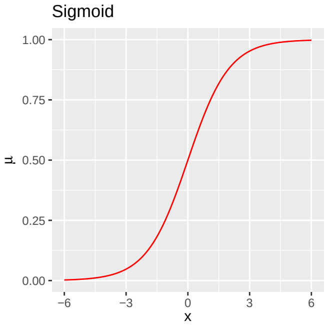
Note that we denote by Z_m the value in the output node m *before* the output layer activation.

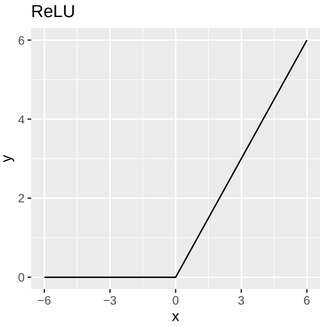
2) Hidden layer activation

(See chapter 6.3 in Goodfellow, Bengio, and Courville (2016))

Very common:

- The **sigmoid** $g = \sigma(x) = 1/(1 + \exp(-x))$ (logistic) activation functions.
- The **rectified linear unit (ReLU)** $g(x) = \max(0, x)$ activation functions.





Less common:

- Radial basis functions: as we looked at in Module 9.
- Softplus: $g(x) = \ln(1 + \exp(x))$
- Hard tanh: $g(x) = \max(-1, \min(1, x))$

Among all the possibilities, ReLU is nowadays the most popular one. Why?

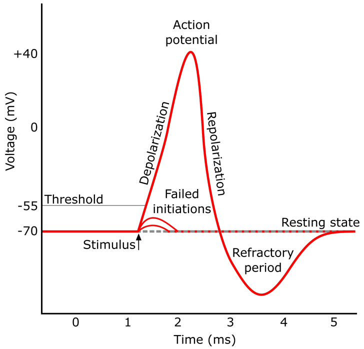
- The function is piecewise linear, but *in total non-linear*.
- Replacing sigmoid with ReLU is reported to be one of the major changes that have improved the performance of the feedforward networks[2].

[2]Goodfellow et al, Section 6.6

ReLU can also be motivated from biology.

- For some inputs a biological neuron can be completely inactive
- For some inputs a biological neuron output can be proportional to the input
- But, most of the time a biological neuron is inactive.

According to Goodfellow, Bengio, and Courville (2016) (Section 6.3), hidden unit design is an *active area of research.*



https://commons.wikimedia.org/wiki/File:Action_potential.svg

Q: Why can we not just use linear activation function in all hidden layers?

Q: Why can we not just use linear activation function in all hidden layers?

A:

- Then each layer would only be able to do linear transformations of the input data and a deep stack of linear layers would still implement a linear operation.
- The universial approximation property is dependent on a squashing type activation function.

Universal approximation property

- Think of the goal of a feedforward network to approximate some function f, mapping our input vector x to an output value y.
- What type of mathematical function can a feedforward neural network with one hidden layer and linear output activation represent?

[3] Goodfellow et al 2016, Section 6.4.1, https://www.deeplearningbook.org

Universal approximation property

- Think of the goal of a feedforward network to approximate some function f, mapping our input vector x to an output value y.

- What type of mathematical function can a feedforward neural network with one hidden layer and linear output activation represent?

The *universal approximation theorem*[3] says that a feedforward network with

- a *linear output layer*
- at least one hidden layer with a "squashing" activation function (e.g., ReLU or sigmoid) and "enough" hidden units

can approximate any (Borel measurable) function from one finite-dimensional space (our input layer) to another (our output layer) with any desired non-zero amount of error.

[3]Goodfellow et al 2016, Section 6.4.1, https://www.deeplearningbook.org

3) Network architecture

Network architecture contains three components:

- *Width*: How many nodes are in each layer of the network?
- *Depth*: How deep is the network (how many hidden layers)?
- *Connectivity*: How are the nodes connected to each other?

Especially the connectivity depends on the problem, and here experience is important.

- We will consider *feedforward networks*, *convolutional neural networks (CNNs)* and *recursive neural networks (RNNs).*

However, the recent practice is to

- choose a too large network, train it until convergence (optimum), which results in overfitting,
- then use other means to avoid this (various variants of regularization and hyperparameter optimization).

This simplifies the choice of network architecture to *choose a large enough network*.

See e.g. Chollet and Allaire (2018), Section 4.5.6/7 and Goodfellow, Bengio, and Courville (2016), Section 7

4) Loss function (“Method”)

- The choice of the loss function is closely related to the output layer activation function.
- Most popular problem types, output activation and loss functions:

Problem	Output nodes	Output activation	Loss function
Regression	1	`linear`	`mse`
Classification (C=2)	1	`sigmoid`	`binary_crossentropy`
Classification (C>2)	C	`softmax`	`categorical_crossentropy`

- Regression: *MSE*

$$\sum_{i=1}^{n}(y_i - f(x_i))^2$$

- Classification: *Cross-entropy*

$$-\sum_{i=1}^{n}\sum_{m=1}^{C} y_i \log f_m(x_i) \ ,$$

with special case for $C = 2$ (binomial cross-entropy loss):

$$-\sum_{i=1}^{n} y_i \log f_m(x_i) + (1 - y_i) \log(1 - f_m(x_i)) \ .$$

5) Optimizors

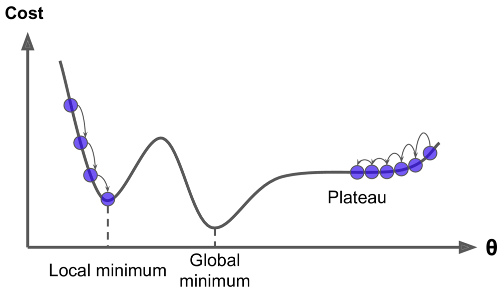
Let the unknown parameters be denoted θ (what we have previously denotes as αs and βs), and the loss function to be minimized $J(\theta)$.

- Gradient descent
- Mini-batch stochastic gradient descent (SGD) and true SGD
- Backpropagation

Here comes more thoery on the optimization

But use the structure in the book (Chapter 10.7)

Start with a general overview slide here



(https://github.com/SoojungHong/MachineLearning/wiki/Gradient-Descent)

Backpropagation

Regularization

Stochastic gradient descent (SGD)

Another form of regularization ($\rightarrow$ protection from over-fitting).

Crucial idea:
The expectation can be approximated by the average gradient over just a *mini-batch* (random sample) of the observations.

Advantages:

- The optimizer will converge much faster if it can rapidly compute approximate estimates of the gradient, instead of slowly computing the exact gradient (using all training data).
- Mini-batches may be processed *in parallel*, and the batch size is often a power of 2 (32 or 256).
- Small batches also bring in a *regularization effect*, maybe due to the variability they bring to the optimization process.

Dropout

Tuning the network's architechture

How to fit those models?

- We will use both the rather simple `nnet` R package by Brian Ripley and the currently very popular `keras` package for deep learning (the `keras` package will be presented later).
- `nnet` fits *one hidden layer* with *sigmoid activiation function*. The implementation is not gradient descent, but instead BFGS using `optim`.
- Type `?nnet()` into your R-console to see the arguments of `nnet()`.
- If the response in formula is a factor, an appropriate classification network is constructed; this has one output, sigmoid activation and binary entropy loss for a binary response, and a number of outputs equal to the number of classes, softmax activation and categorical cross-entropy loss for more levels.

An example

Boston house prices

Objective: To predict the median price of owner-occupied homes in a given Boston suburb in the mid-1970s using 10 input variables. This data set is both available in the `MASS` and `keras` R package.

Preparing the data

- Only 506, split between 404 training samples and 102 test samples.
- Each feature in the input data (for example, the crime rate) has a different scale, some values are proportions, which take values between 0 and 1; others take values between 1 and 12, others between 0 and 100, and so on.

Read and check the data file:

```
library(MASS)
data(Boston)
dataset <- Boston
head(dataset)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

Split into training and test data

```
set.seed(123)
tt.train <- sort(sample(1:506, 404, replace = FALSE))
train_data <- dataset[tt.train, 1:13]
train_targets <- dataset[tt.train, 14]

test_data <- dataset[-tt.train, 1:13]
test_targets <- dataset[-tt.train, 14]
```

- To make the optimization easier with gradient based methods do *feature-wise normalization.*

```r
org_train = train_data
mean <- apply(train_data, 2, mean)
std <- apply(train_data, 2, sd)
train_data <- scale(train_data, center = mean, scale = std)
test_data <- scale(test_data, center = mean, scale = std)
```

- **Note**: the quantities used for normalizing the test data are computed using the training data. You should never use in your workflow any quantity computed on the test data, even for something as simple as data normalization.

Just checking out one hidden layer with 5 units to get going.

```r
library(nnet)
fit5 <- nnet(train_targets ~ ., data = train_data, size = 5, linout = TRUE,
    maxit = 1000, trace = F)
```

Calculate the MSE and the mean absolute error:

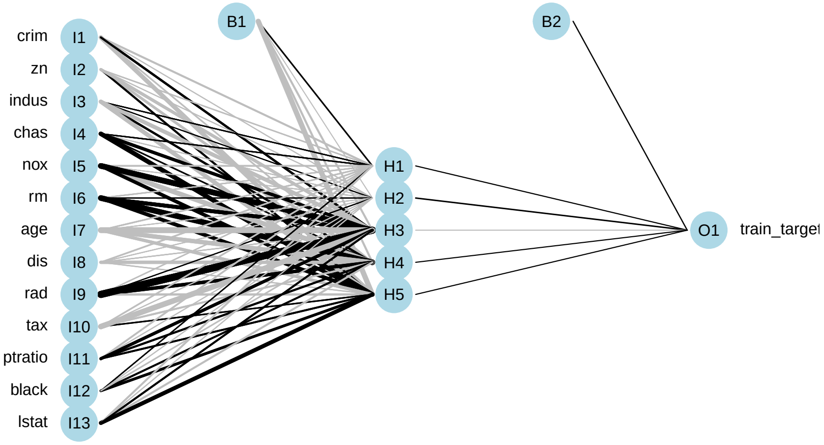
```r
pred = predict(fit5, newdata = test_data, type = "raw")
mean((pred[, 1] - test_targets)^2)
```

```
## [1] 17.65425
```

```r
mean(abs(pred[, 1] - test_targets))
```

```
## [1] 3.239263
```

```
library(NeuralNetTools)
plotnet(fit5)
```



Boston example using `keras`

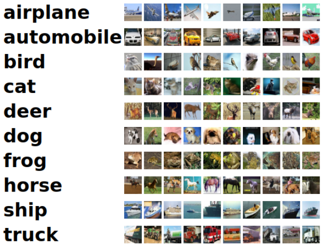
See recommended exercise.

It can serve as an excellent example to illustrate that *simple* linear regression can do a pretty good job in short time.

Convolutional neural networks (CNNs)

- Motivated by image classification.

- Example: the CIFAR-100 dataset (https://www.cs.toronto.edu/~kriz/cifar.html): Images of 100 categories with 600 images each.



(Example from the CIFAR-10 data set with only 10 classes).

- Idea of CNNs: recognize features and patterns.
- The network identifies *low-level features* (edges, color patched etc).
- These low-level features are then combined into *higher-level features.*
- Two types of layers: *Convolution layers* and *pooling layers*.

Convolution layers

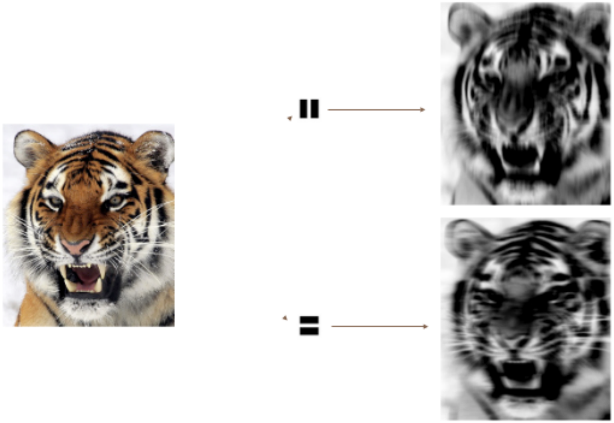
- Composed of *filters*.
- Example:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix} \qquad \text{Convolved with} \qquad \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

$\rightarrow$ Convolved image:

The filter highlights regions in the image that are similar to the filter itself.

Filtering for vertical or horizontal stripes:



(Figure 10.7)

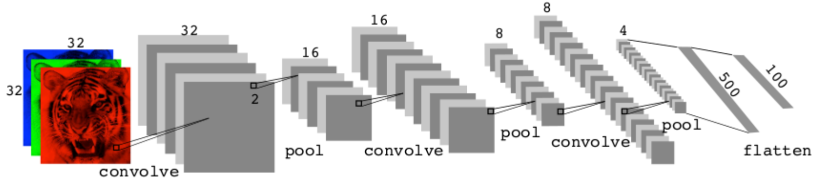
- In *image processing* we would use predefined (fixed) filters.
- In CNNs, the idea is that the filters are *learned*.
- One filter is applied to each color (red, green, blue), so three convolutions are happening in parallel and then immediately summed up.
- In addition, we can use K different filters in a convolution step. This produced 3D feature maps (of depth K).
- The convolved image is then also processed with the ReLU activation function.

Pooling layers

- Idea: consense/summarize information about the image.
- *Max pool*: Use the maximum value in each 2×2 block.

$$\begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \qquad \rightarrow \qquad \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

In a CNN, we now combine convolution and pooling steps interatively:



- The number of channels after a convolution step is the number of filters (K) that is used in this iteration.
- The dimension of the 2D images after a pooling step is reduced, depending on the dimension of the filter (e.g., 2×2 reduces each dimension by a factor of 2).
- In the end, all the dimensions are *flattened* (pixels become ordered in 2D).
- The output layer has a *softmax* activation function since the aim is classification.

Data augmentation

- Very simple idea: Make the analysis more robust by including replicated, but slightly modified pictures of the original data.
- Example:

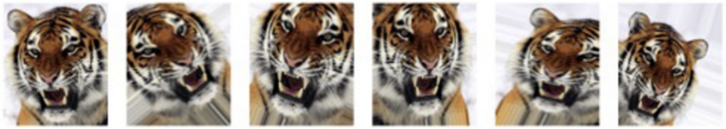


Figure 10.9 of James et al. (2021)

Examples

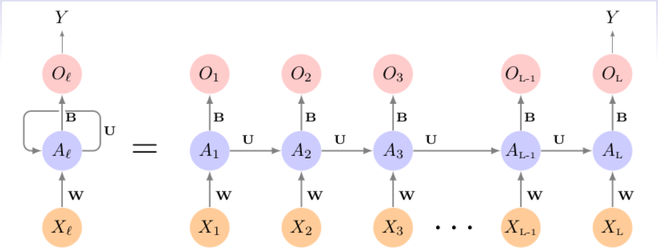
See

- Section 10.3.5 in the book,
- Examples in the recommended exercise 11.

Recurrent neural networks (RNNs)

- Suitable for data with sequential character.
- Examples: Text documents, time series (temperature, stock prices, music, speech,...)
- The input object X is a sequence.
- In the most simple case, the output Y is a single value (continuous, binary or a category).
- More advanced RNNs are able to map sequences to sequences (*Seq2Seq*)[4] in language modeling, and much more!

[4]Google Translate uses this technique, for example



Y
O_ℓ
B
U
A_ℓ
W
X_ℓ
$=$
O_1 O_2 O_3 O_{L-1} O_L
A_1 A_2 A_3 A_{L-1} A_L
X_1 X_2 X_3 $\cdots$ X_{L-1} X_L

Figure 10.12 of James et al. (2021)

- Observed sequence $X = \{X_1, \ldots, X_L\}$, where each $X_l^\top = (X_{l1}, \ldots, X_{lp})$ is an input vector at point l in the sequence.
- Sequence of hidden layers $\{A_1, \ldots, A_L\}$, where each A_l is a layer of K units $A_l^\top = (A_{l1}, \ldots, A_{lK})$.
- A_{lk} is determined as

$$A_{lk} = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{lj} + \sum_{s=1}^{K} u_{ks} A_{l-1,s}) \ , \qquad (1)$$

with hidden layer activation function $g()$ (e.g., ReLU).

- The output is determined as

$$O_l = \beta_0 + \sum_{k=1}^{K} \beta_k A_{lk} \ ,$$

  potentially with a sigmoid or softmax output activation for binary or categorical outcome.

- Note: The weights W, U and B are the *same* at each point in the sequence. This is called *weight sharing*.

Fitting the weights in an RNN

- Minimize a *loss function*. In regression problems:

$$\text{Loss} = (Y - O_L)^2 \ .$$

- Only the *last observation* is relevant. How can this be meaningful?

- Reason: each element X_l contributes to O_L via equation (1).

- For input sequences (x_i, y_i), we minimize $\sum_{i=1}^{n} (y_i - o_{iL})^2$.

- Note: $x_i = \{x_{i1}, \ldots, x_{iL}\}$ is a sequence of *vectors*.

Why are the outputs $O_1, \ldots, O_{L-1}$ there at all?

Why are the outputs $O_1, \dots, O_{L-1}$ there at all?

A:

- They come for free (same weights B).
- Sometimes, the output is a whole sequence.

Example of an RNN: Time series forecasting

Trading statistics from New York Stock exchange:

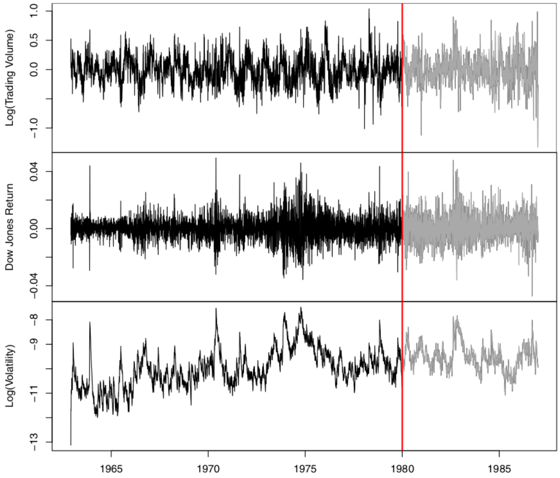


Figure 10.14 of James et al. (2021)

Observations:

- Every day ($t = 1, \dots, 6051$) we measure three things, denoted as (v_t, r_t, z_t).
- All three series have high *autoc-orrelation*.

Aim:

- Predict v_t from
  - v_{t-1}, v_{t-2}, ...,
  - r_{t-1}, r_{t-2}, ..., and
  - z_{t-1}, z_{t-2}, ...

But, how do we represent this problem in terms of Figure 10.12?

The idea is to extract shorter series up to a *lag* of length L:

$$X_1 = \begin{pmatrix} v_{t-L} \\ r_{t-L} \\ z_{t-L} \end{pmatrix}, \quad X_1 = \begin{pmatrix} v_{t-L+1} \\ r_{t-L+1} \\ z_{t-L+1} \end{pmatrix}, \quad X_1 = \begin{pmatrix} v_{t-1} \\ r_{t-1} \\ z_{t-1} \end{pmatrix}, \quad Y = v_t$$

- And then continue to formulate the model as indicated in Figure 10.12.

Interpolation and double descent

References and further reading

- https://youtu.be/aircAruvnKk from 3BLUE1BROWN - 4 videos - using the MNIST-data set as the running example
- Look at how the hidden layer behave: https://playground.tensorflow.org
- Friedman, Hastie, and Tibshirani (2001),Chapter 11: Neural Networks
- Efron and Hastie (2016), Chapter 18: Neural Networks and Deep Learning
- Chollet and Allaire (2018)
- Goodfellow, Bengio, and Courville (2016) (used in IT3030) https://www.deeplearningbook.org/
- Explaining backpropagation http://neuralnetworksanddeeplearning.com/chap2.html
- Slides from MA8701 (Thiago Martins) https://www.math.ntnu.no/emner/MA8701/2019v/DeepLearning/

Acknowledgements

Chollet, François, and J. J. Allaire. 2018. *Deep Learning with r*. Manning Press. https://www.manning.com/books/deep-learning-with-r.

Efron, Bradley, and Trevor Hastie. 2016. *Computer Age Statistical Inference - Algorithms, Evidence, and Data Science*. Cambridge University Press.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics New York.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2021. *An Introduction to Statistical Learning*. 2nd ed. Vol. 112. Springer.