Deloitte.

Convolutional Neural Networks in insurance

Patrik Lipták

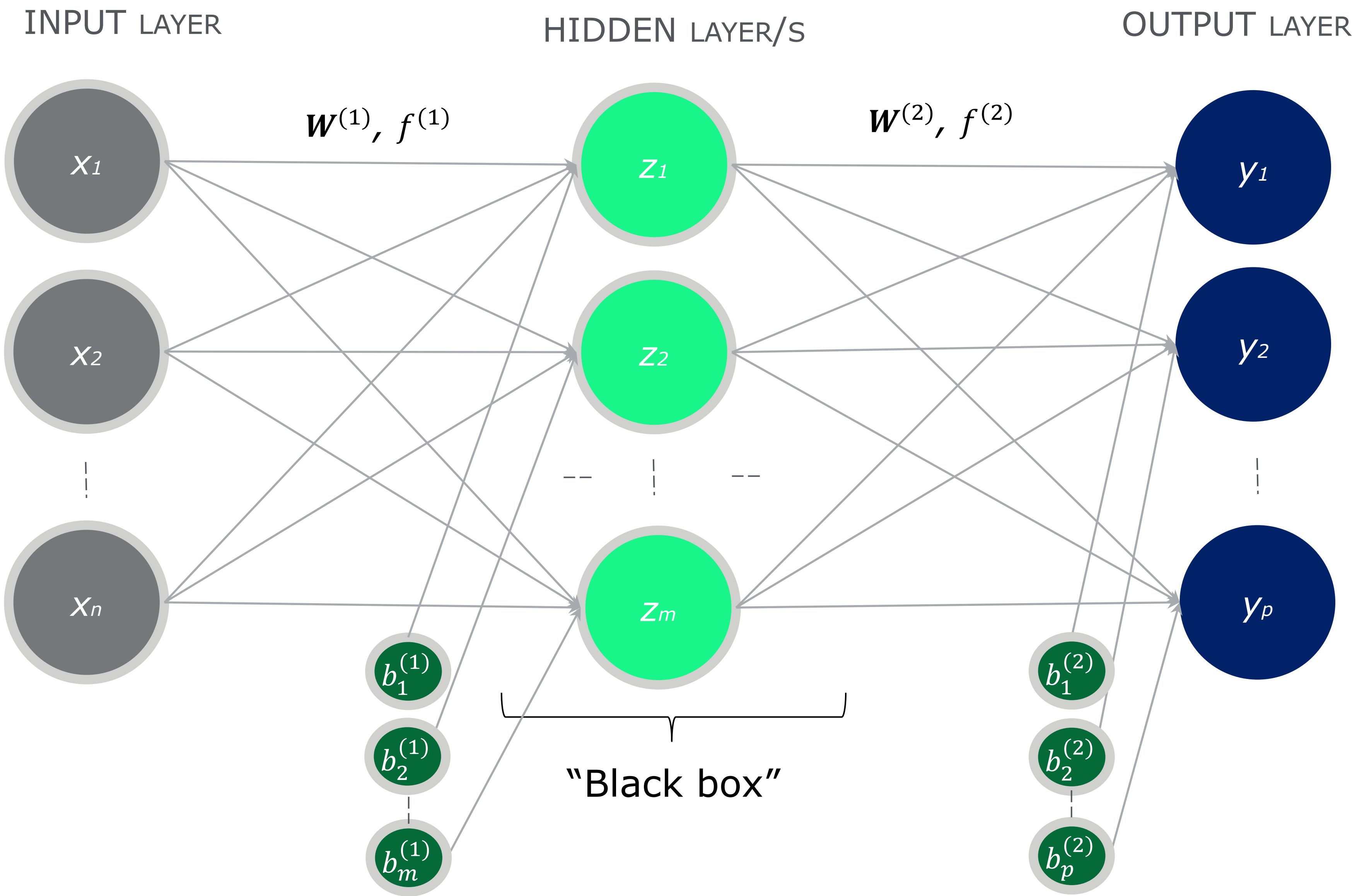
01.11.2019

Agenda

- Deep feed-forward neural networks
- Convolutional neural networks
- R session – Parkinson disease spiral drawings
- Practical application – Online underwriting and Fraud detection in MTPL insurance

Introduction to Deep Neural Networks

Deep Neural Network
Architecture



$\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ - matrices of weights

$f^{(1)}, f^{(2)}$ - activation functions

$\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ - vectors of biases

Deep Neural Network
Notation

- m hidden neurons z$_1$, …, z$_m$ with:

$$z_i = f^{(1)}\left(\sum\nolimits_j \boldsymbol{W}_{i,j}^{(1)}\, x_j + b_i^{(1)}\right), \qquad i = 1, \dots, m$$

$$\boldsymbol{z} = f^{(1)}\big(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\big), \qquad \boldsymbol{z} \in \mathbb{R}^m, \boldsymbol{W}^{(1)} \in \mathbb{R}^{m\times n}, \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{b}^{(1)} \in \mathbb{R}^m$$

- Linear combination of derived features $\boldsymbol{z}$*:*

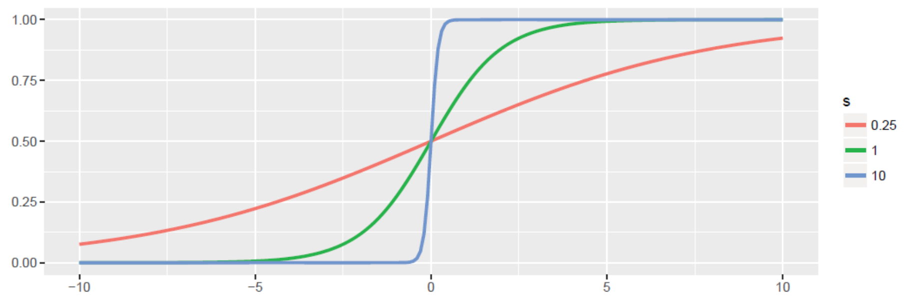
$$\boldsymbol{y} = f^{(2)}\big(\boldsymbol{W}^{(2)}\boldsymbol{z} + \boldsymbol{b}^{(2)}\big), \qquad \boldsymbol{y} \in \mathbb{R}^p, \boldsymbol{W}^{(2)} \in \mathbb{R}^{p\times m}, \boldsymbol{b}^{(2)} \in \mathbb{R}^p$$

$$\boldsymbol{y} = f^{(2)}\big(\boldsymbol{W}^{(2)}\boldsymbol{z} + \boldsymbol{b}^{(2)}\big) = f^{(2)}\left(\boldsymbol{W}^{(2)}\left(f^{(1)}\big(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\big)\right) + \boldsymbol{b}^{(2)}\right)$$

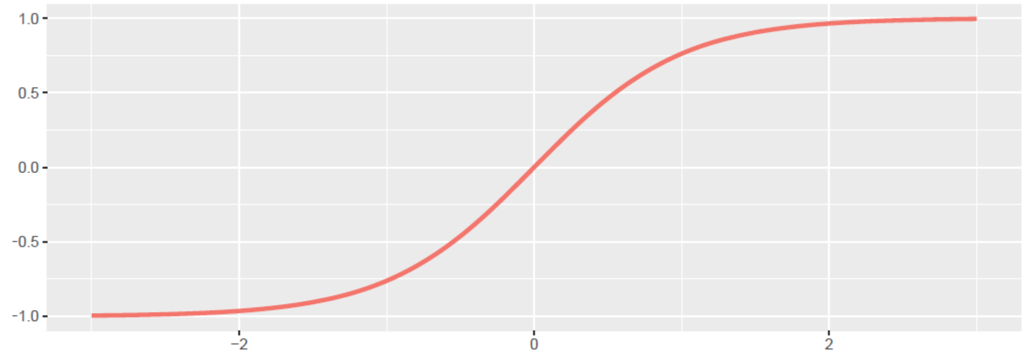
- $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ are called matrices of weights (with not necessarily positive elements).
- $\boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}$ are so called vectors of biases (intercepts).
- $f^{(1)}()$ is called activation function and $f^{(2)}$ output activation function.

Deep Neural Network
Activation function

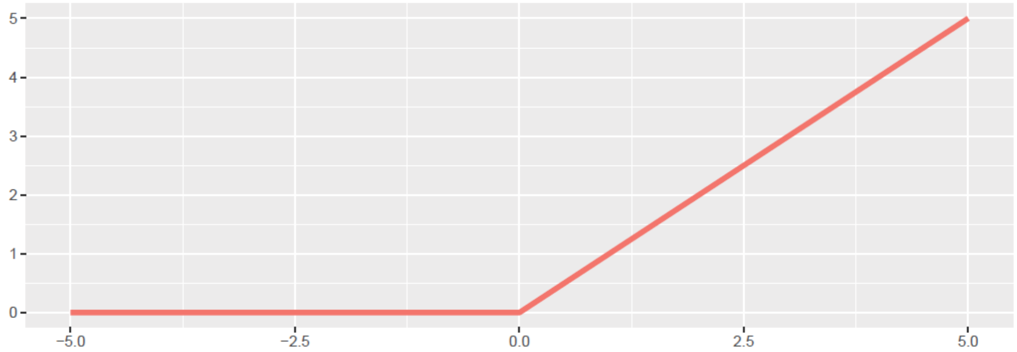
- Converting an input signal of node into an output signal respectively “activation” of a neuron.
- In general non-linear, monotonically increasing and bounded.
- $f^{(2)}$ depends on the nature of the problem – classification: softmax, regression: identity…



$$f^{(1)}(x) = \frac{1}{(1+\exp(-sx))} \rightarrow \text{logistic regression}$$



$$f^{(1)}(x) = \tanh(x)$$



$$\text{ReLU: } f^{(1)}(x) = \max(0, x)$$

Deep Neural Network
Training

- We usually have a **training set**, which is assumed to consist of examples generated independently from a data generating distribution.
- The goal of optimization is to match the training set as well as possible.
- However, the main goal of machine learning is to perform well on previously unseen data and to minimize so called **generalization error** or **test error**. We typically estimate the generalization error using a **test set** of examples independent of the training set, but generated by the same data generating distribution.

Training of neural nets is composed of two iterative steps:

1. **Forward pass**: the information of the inputs flows through the model to produce a prediction.

Based on that, we compute a loss which is sometimes called the cost.

2. **Backward pass**: the information of the error flows backwards through the model. Thereby we use the error values to calculate the gradient of the loss with respect to each weight.

In a final step we update the weights (i.e. “move” them in the direction of the steepest descent of the loss).

Deep Neural Network

Loss function

- Our objective is to minimize the loss for a neural network by optimizing its parameters - weights.
- A common principle used to design loss functions is the *maximum likelihood principle*.
- For the sake of simplicity, let's denote $\boldsymbol{y} = f(\boldsymbol{x})$. In the case of binary output, the loss function is the negative log-likelihood of the bernoulli distribution, which is commonly just called the cross entropy:

$$\mathrm{L}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log f(\boldsymbol{x}) + (1 - y_i)\log(1 - f(\boldsymbol{x}))]$$

- Another choice for linear regression is the mean squared error:

$$\mathrm{L}\big(\boldsymbol{y}, f(\boldsymbol{x})\big) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}))^2$$

- The term cross-entropy is widely used for the negative log-likelihood of a bernoulli or softmax distribution, but that is a misnomer.
  - Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution of the training data and the probability distribution defined by model!
  - For example, the mean squared error is the cross-entropy between the empirical distribution and a Gaussian model.

Deep Neural Network
Gradient descent method

- Let $f(x)$ be an arbitrary, differentiable, unrestricted target function, which we want to minimize.
  - We can calculate the gradient $\nabla f(x)$, which always points in the direction of the **steepest ascent**.
  - Thus $-\nabla f(x)$ points in the direction of the **steepest descent**!
- Standing at a point x_k during minimization, we can improve this point by doing the following step:

$$f(x_{k+1}) = f(x_k) - \alpha \, \nabla f(x_k)$$

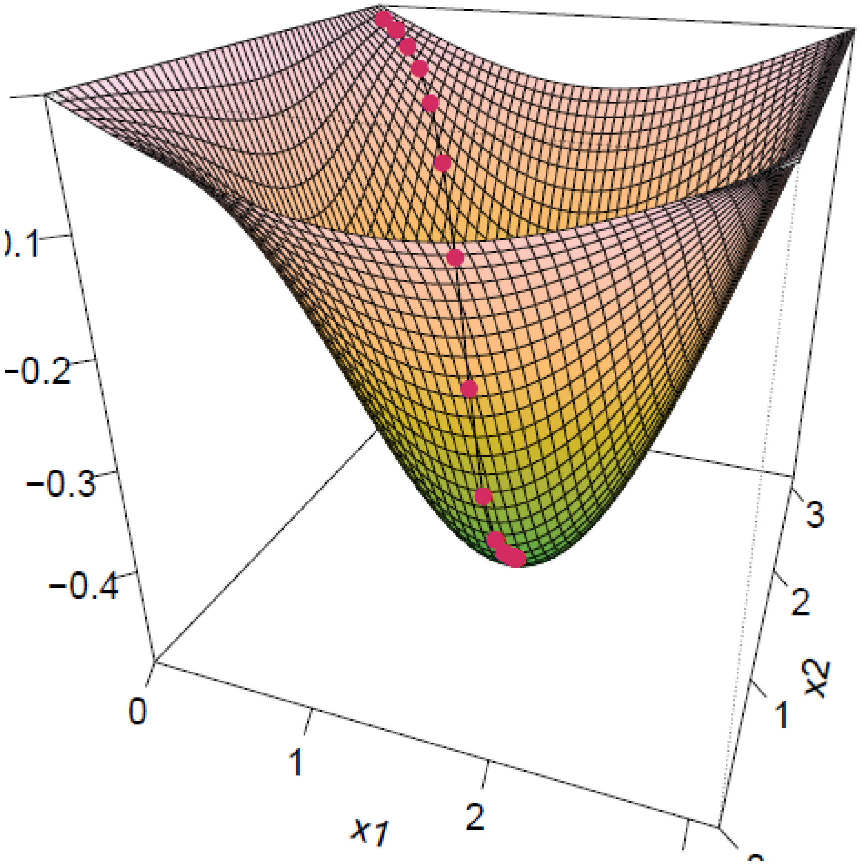
"Walking down the hill, towards the valley."

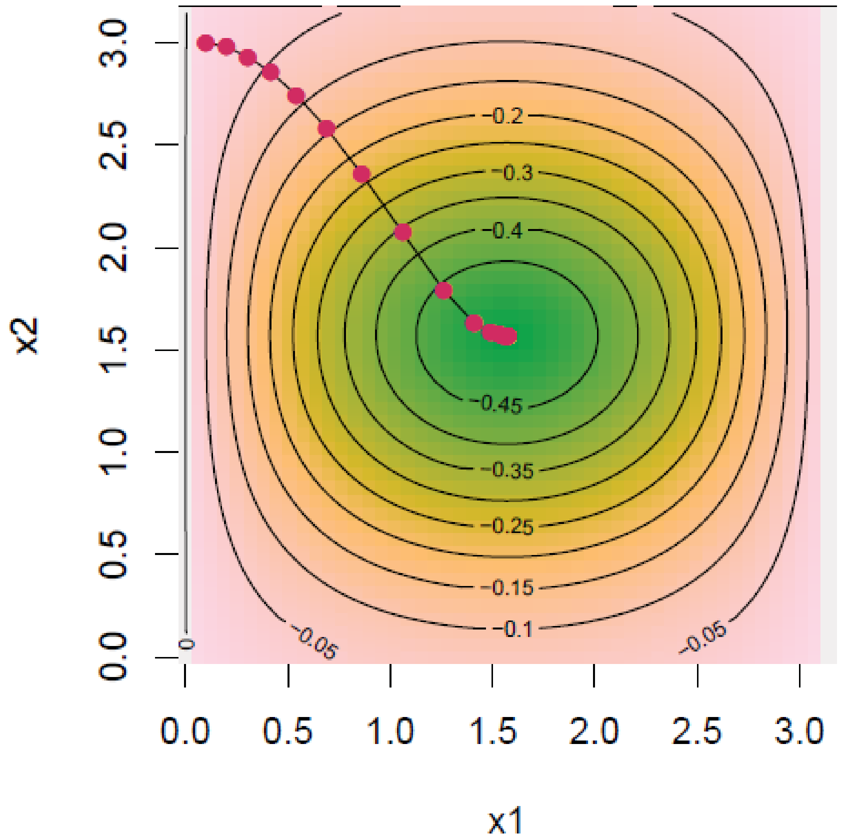
- α determines the length of the step and is called step size or in terms of neural networks **learning rate**. To find the optimal α we need to look at **minimal** value.

Deep Neural Network

Gradient descent method

$$f(x_1, x_2) = -\sin(x_1) \cdot \frac{1}{2\pi} \exp\left((x_2 - \pi/2)^2\right)$$



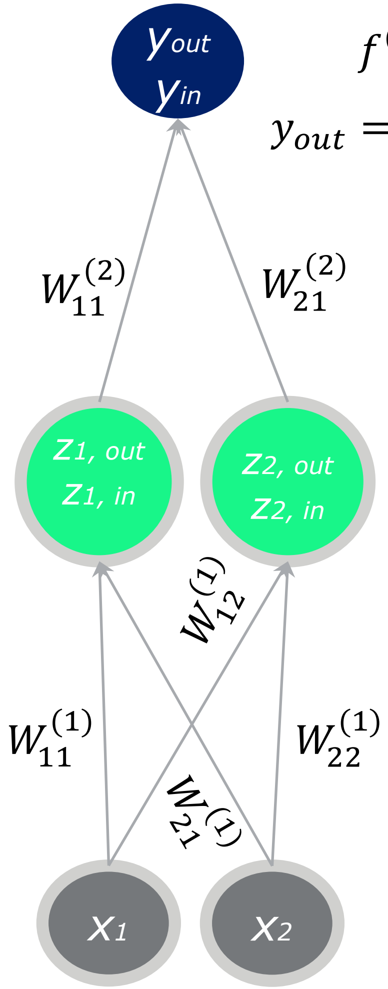


Deep neural network

Backpropagation algorithm example – binary classification

Forward feeding: calculation

$$L(y, y_{out}) = -(y \log y_{out} + (1-y)\log(1-y_{out}))$$



$f^{(2)} \rightarrow$ sigmoid

$$y_{out} = \frac{1}{1+\exp(-\boldsymbol{W}^{(2)}\boldsymbol{z})}$$

$f^{(1)} \rightarrow$ ReLU

$$\boldsymbol{z}_{out} = MAX(\boldsymbol{0}, \boldsymbol{W}^{(1)}\boldsymbol{x})$$

Backpropagation: training of weights

$$f(\boldsymbol{x}) = f^{(2)}\left(\boldsymbol{W}^{(2)}\left(f^{(1)}(\boldsymbol{W}^{(1)}\boldsymbol{x})\right)\right)$$

$$W_{11}^{(2)}: \frac{\partial L(y,f(\boldsymbol{x}))}{\partial W_{11}^{(2)}} = \frac{\partial L(y,f(\boldsymbol{x}))}{\partial y_{out}} \frac{\partial y_{out}}{\partial y_{in}} \frac{\partial y_{in}}{\partial W_{11}^{(2)}}$$

Updating parameter: $W_{11}^{(2)} \leftarrow W_{11}^{(2)} - \alpha \times \frac{\partial L(y,f(\boldsymbol{x}))}{\partial W_{11}^{(2)}}$

$$W_{11}^{(1)}: \frac{\partial L(y,f(\boldsymbol{x}))}{\partial W_{11}^{(1)}} = \frac{\partial L(y,f(\boldsymbol{x}))}{\partial y_{out}} \frac{\partial y_{out}}{\partial y_{in}} \frac{\partial y_{in}}{\partial z_{1,out}} \frac{\partial z_{1,out}}{\partial z_{1,in}} \frac{\partial z_{1,in}}{\partial W_{11}^{(1)}}$$

Updating parameter: $W_{11}^{(1)} \leftarrow W_{11}^{(1)} - \alpha \times \frac{\partial L(y,f(\boldsymbol{x}))}{\partial W_{11}^{(1)}}$

*Biases are usually initialized to a constant value, usually 0. Weights are usually initialized to small random values, either with uniform or normal distribution.

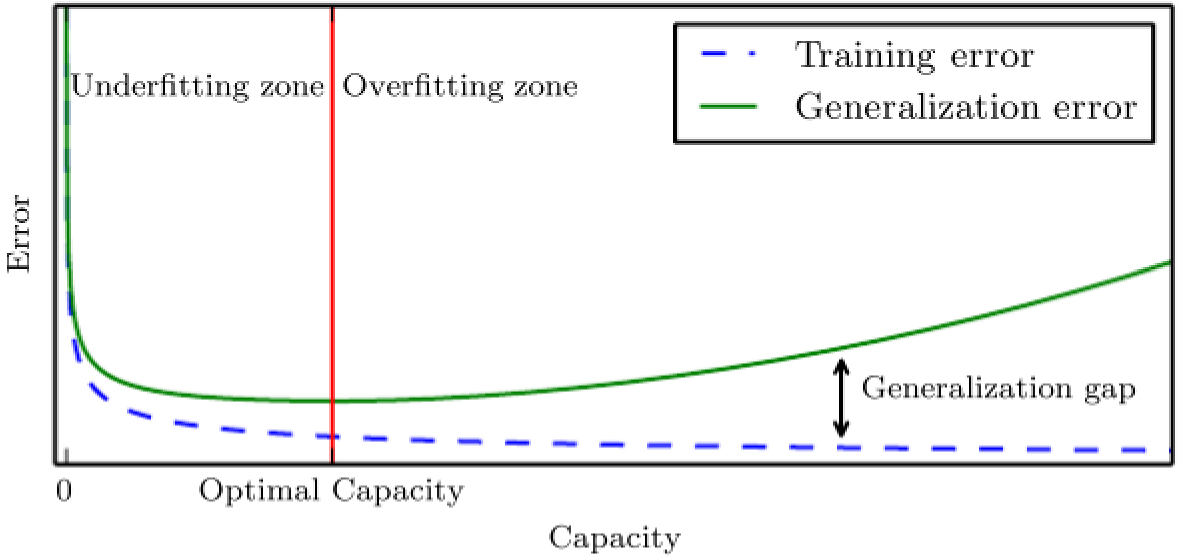
Overfitting
Regularization

Regularization is any change in the machine learning algorithm that is designed to reduce generalization error but not necessarily its training error.

Regularization is usually needed only if training error and generalization error are different.

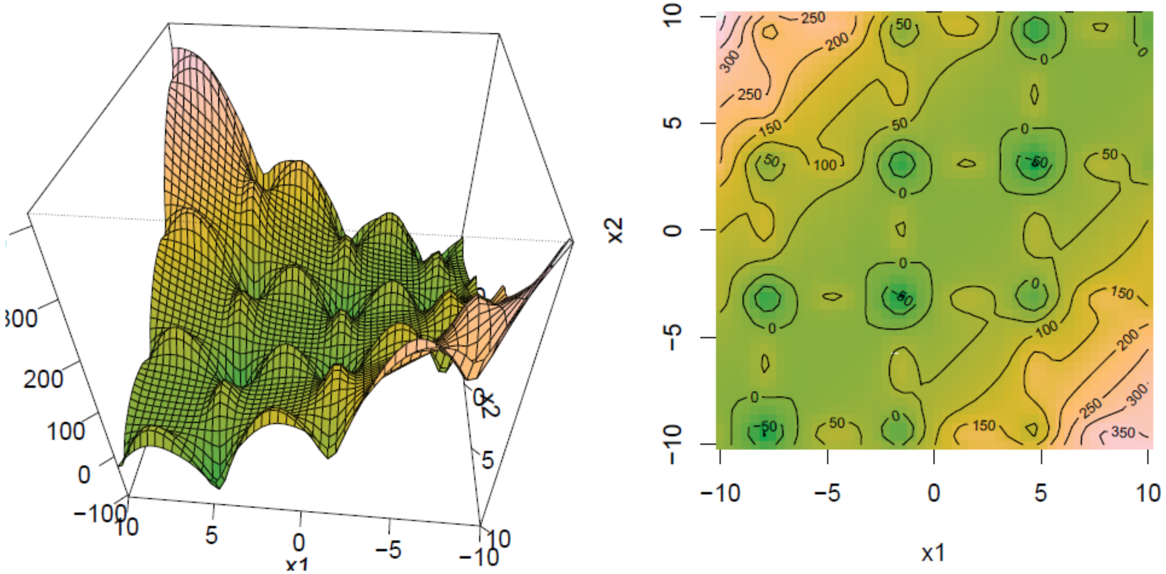
Methods:

- Early stopping
- L2, L1 regularization
- Dataset augmentation
- Ensembling
- Dropout
- …

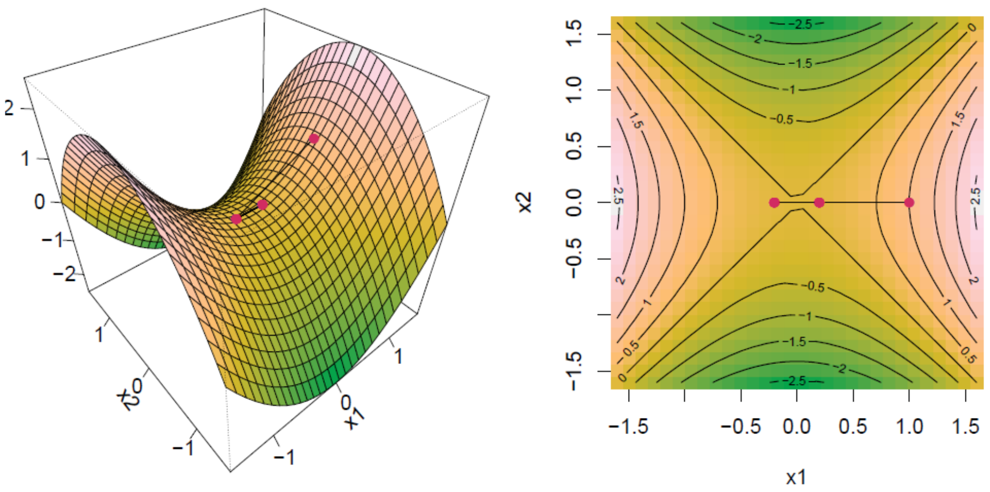


Deep Neural Network
Optimization problem

- Convex optimization problems can be reduced to "finding a local minimum".
- In neural networks we have to deal with non-convex problems or flat regions such as saddle points:



$$f(x_1, x_2) = x_1^2 - x_2^2$$



Tenth step

Deep Neural Network
Stochastic gradient descent

However, serious consequences can be easily avoided using a technique called **gradient clipping**.

- The gradient does not specify the optimal step size, but only the optimal direction within an infinitesimal region.
- Gradient clipping simply caps the step size to be small enough that it is less likely to go outside the region where the gradient indicates the direction of steepest descent.
- SGD and its modifications are the most used optimization algorithms for machine learning in general and for deep learning in particular.
- **Algorithm 1** SGD parameter update at training iteration k

  require learning rate α_k
  require initial parameter θ
  while stopping criterion not met **do**
      Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$
      Compute gradient estimate: $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(y^{(i)}, f(x^{(i)}, \theta))$
      Apply update: $\theta \leftarrow \theta - \alpha \hat{g}$
  end while

  In practice, a common strategy is to decay the learning rate α_k linearly over time.

  To choose α : monitor learning curves that plot the objective function as a function of time.

- Other techniques: momentum, algorithms with adaptive learning rates (Adagrad, RMSProp, Adam…)

Practical use

Use in insurance

- In principle, neural networks bring massive improvements in structuring of unstructured data.
- On the other side, „traditional“ structured insurance data is still not „big“ enough, that NN´s would outperform complex tree of regression models in predicting on structured data.
- Possible usage:
  - Pricing of non-life insurance (Noll, Salzmann and Wüthrich 2018; Wüthrich and Buser 2018),
  - Analysis of telematics data (Gao, Meng and Wüthrich 2018; Gao and Wüthrich 2017),
  - IBNR Reserving (Kuo 2018),
  - Automation,
  - Classification problems,
  - Mortality forecasting (Hainaut 2018).

Quiz time!

Which of these elements must be specified at the beginning of the backpropagation algorithm?

a) Activation functions

b) Number of hidden layers

c) Weights

d) Loss function

Convolutional Neural Networks

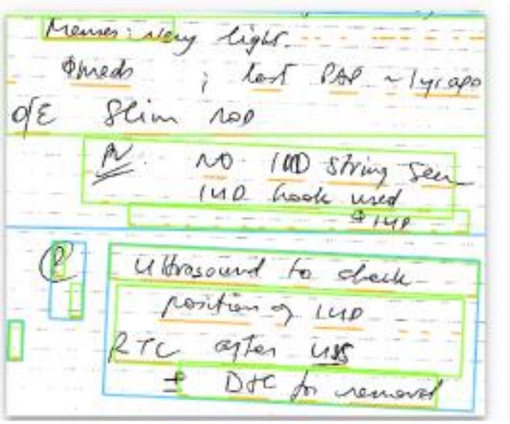
Motivation

“New” insurance data


Constat amiable d'accident automobile


CONSTAT AMIABLE D'ACCIDENT AUTOMOBILE


REPORT OF INVESTIGATION BY COUNTY MEDICAL EXAMINER

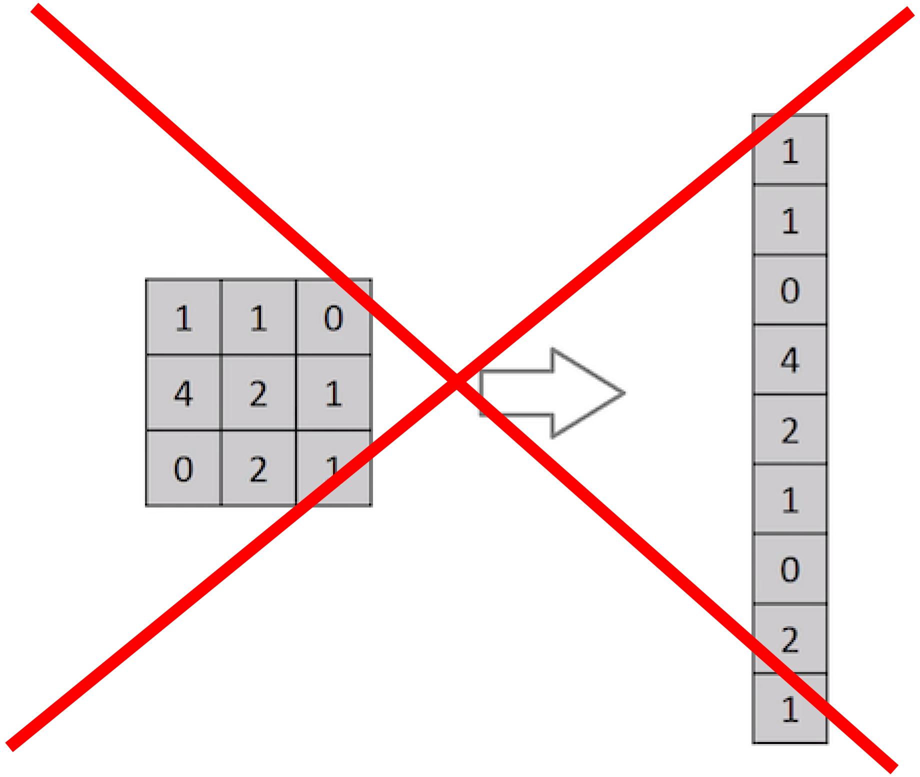

Menses: very light.
Meds ; last PAP ~ 1yr ago
O/E Slim NAD
NO IUD string seen
IUD hook used
Ultrasound to check position of IUD
RTC after USS
± D&C for removal

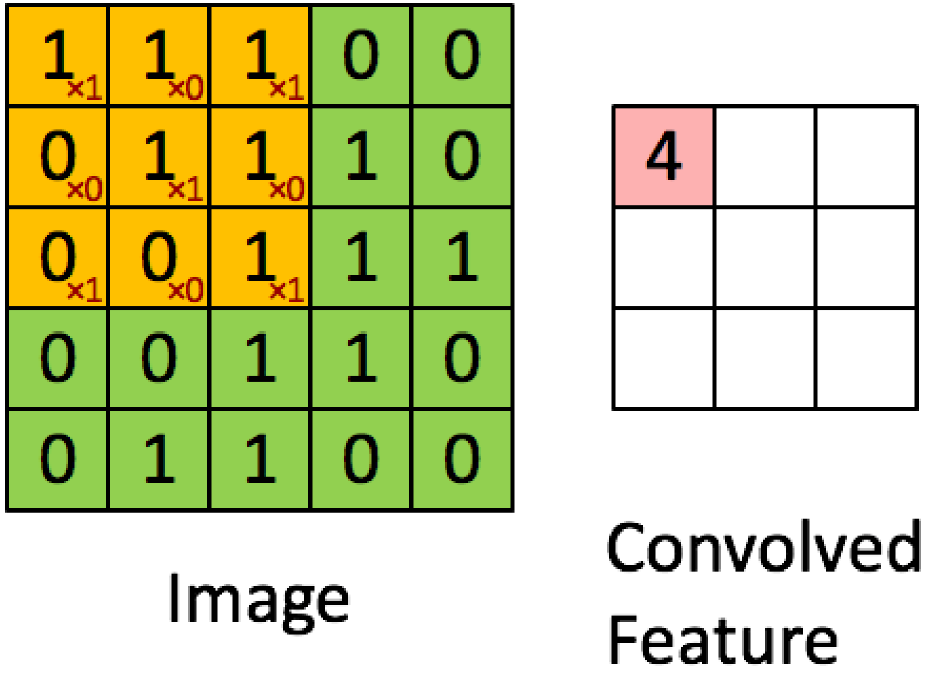
Convolutional neural network

Concept

Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

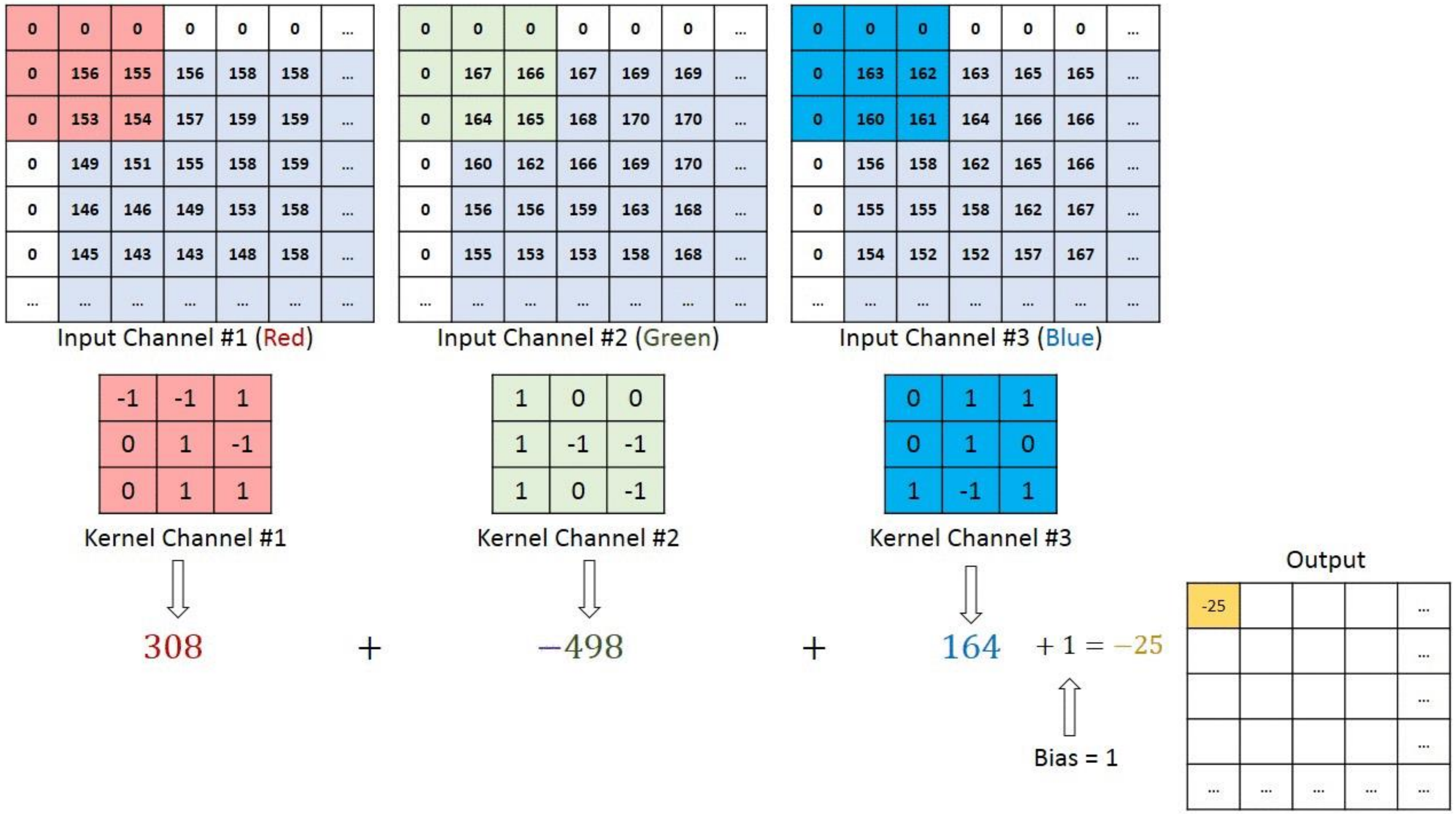
How to make it work? Ideas?





Convolutional neural network
Convolution

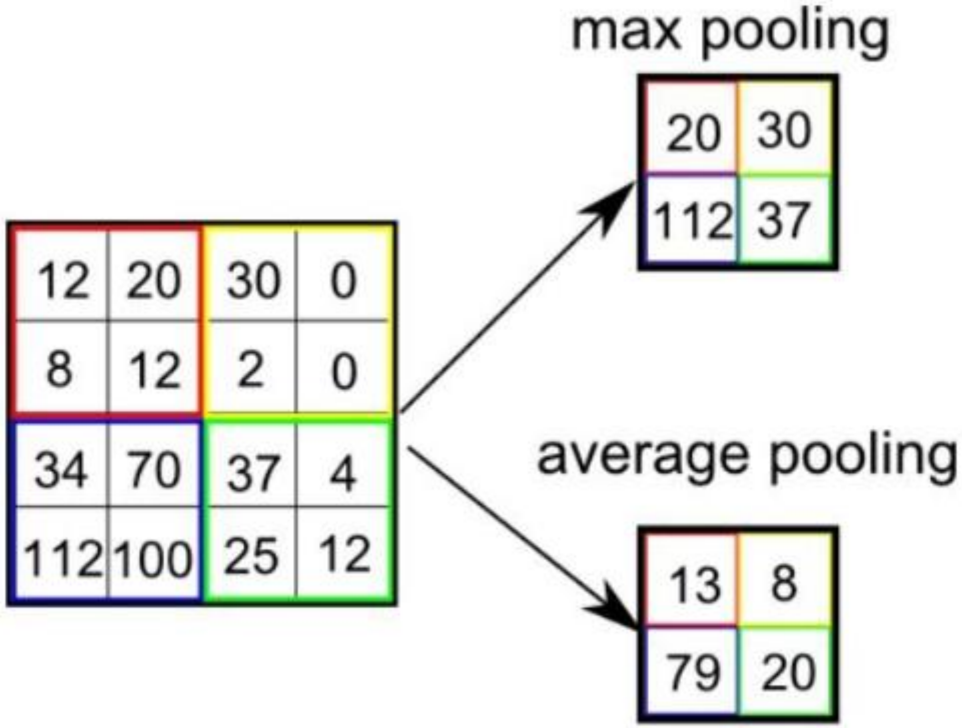
Extracting the high-level features/patterns such as edges, colour, gradient orientation from the input image:



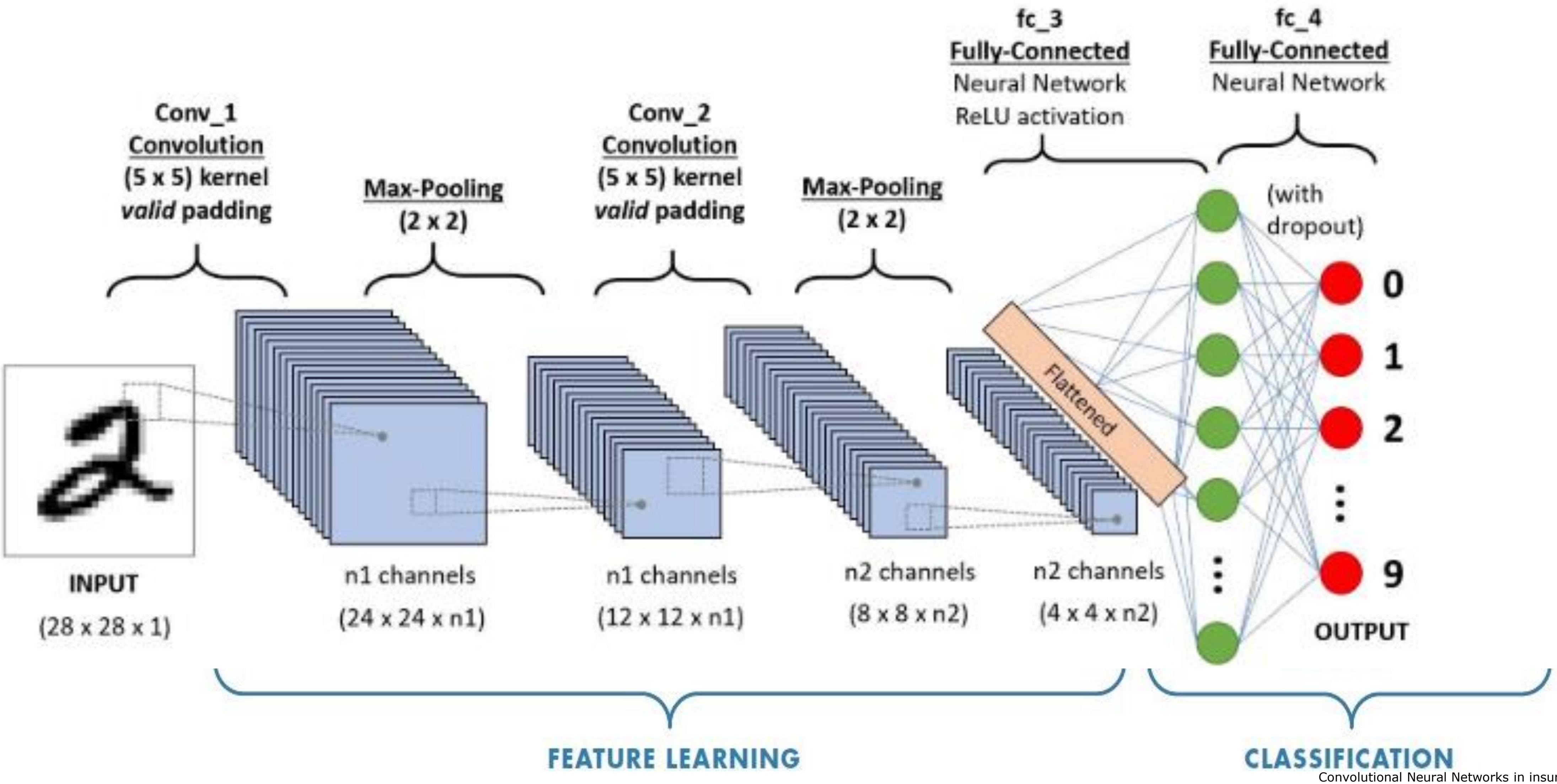
Convolutional neural network
Pooling

- **Decrease the computational power** required to process the data.
- **Extracting dominant features** which are rotational and positional invariant.

Two types of Pooling: Max Pooling (noise suppressant) and Average Pooling:



Convolutional neural network
Practical example



http://scs.ryerson.ca/~aharley/vis/conv/flat.html

Quiz time!

Which stage of CNN extract the high-level patterns such as edges… from the input image?

a) Pooling

b) Convolution

R session

Parkinson disease spiral drawings

Coding

R as a programming language

- Tensorflow
- Keras (include installing ‘reticulate’ package for interface of Python in R)

List of different types of neural network models that can be built in R using Keras:

- Feed-forward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Use pre-trained models like VGG16, RESNET etc.
- Fine-tune the pre-trained models.

**’devtools' package f*or installing packages from github needed

R session
Syntax of keras package – binary classification

```
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 8,
                      kernel_size = c(5,5),
                      activation = 'relu',
                      input_shape = input_shape) %>%
  layer_max_pooling_2d(pool_size = c(3, 3)) %>%
  layer_dropout(rate = 0.25) %>%
  ### next possible convolutional and pooling layers
  layer_flatten() %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 2, activation = 'sigmoid')


model %>% compile(
  loss = "binary_crossentropy",
  optimizer = "adam",
  metrics = c('accuracy')
)

batch_size <- 72
epochs <- 50


# Train model
model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_split = 0.2,
  shuffle = TRUE
)

# Predictions
model %>% evaluate(x_train, y_train)
model %>% evaluate(x_test, y_test)
```

Practical application

Online underwriting in MTPL insurance

Practical application

Fraud detection

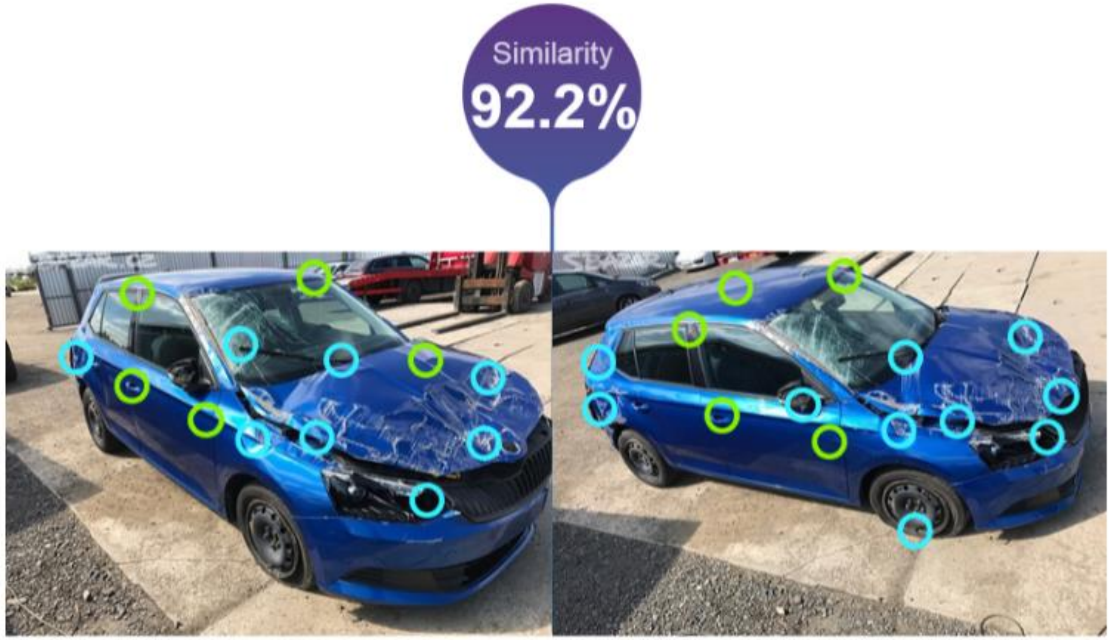
Convolutional neural network

Insurance analytics – fraud detection

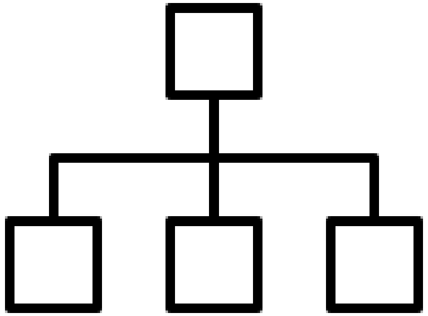
Costly **claim processing** with large involvement of personal stuff. **Frauds** amounting on average to 10% of all claim expenditure every year (€4bn in DE).

Searpent (with Deloitte’s cooperation):

- Visual car accidents and flooded flats fraud detection system for insurance market.
- Identifies repeated claims and modified images across the market.



Going further…



Claims classification (identification of straight-through processing cases vs requiring special treatment) and **improved fraud detection** using client's **structured data**.

Improving claims classification (identification of straight-through processing cases vs requiring special treatment) and **fraud detection** using client's **unstructured textual data** processed by NLP.

Monitoring and producing a **fraud risk score** for **voice interactions** based on speech, behavioral and emotional tendencies detected during **customer service calls** by **voice analytics**.

Providing and setting up simple and scalable all-in-one **cloud environment** that will enable clients to **integrate all relevant internal and external data sources** and utilize them for making **claims processing more effective and efficient**.

Conclusion and discussion

Conclusion
and discussion…

- Deep learning can enhance the predictive power of models built by actuaries.
- Very useful for high-frequency and high-dimensional data.
- Application of deep learning techniques to actuarial problems seems to be rapidly emerging field within actuarial science => appears reasonable to predict more advances in the near-term.
- Deep learning is not a panacea for all modelling issues - applied to the wrong domain, deep learning will not produce better or more useful results than other techniques.
- Winter might be coming – if actuaries do not take the lead in applying deep learning, someone else will.

Thank you for your attention!

Contact

Patrik Lipták

Senior Consultant | Actuarial & Insurance Solutions

Deloitte Advisory s.r.o.

Churchill I, Italská 2581/67, 120 00, Prague 2 – Vinohrady, Czech Republic

Reception: +420 246 042 500 | M: +420 778 820 777

pliptak@deloitteCE.com | www.deloitte.cz

Deloitte.

References

- Gao, G., S. Meng and M. Wüthrich. 2018. Claims Frequency Modeling Using Telematics Car Driving Data. SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3102371. Accessed: 14 October 2019.
- Gao, G. and M. Wüthrich. 2017. Feature Extraction from Telematics Car Driving Heatmaps. SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3070069. Accessed: 14 October 2019.
- Goodfellow, I., Bengio Y. and A. Courville. 2016. Deep Learning. http: // www. deeplearningbook. Org. Accessed: 14 October 2019
- Hainaut, D. 2018. "A neural-network analyzer for mortality forecast", Astin Bulletin 48(2): 481-508.
- Kuo, K. 2018. "DeepTriangle: A Deep Learning Approach to Loss Reserving"
- Noll, A., R. Salzmann and M. Wüthrich. 2018. Case Study: French Motor Third-Party Liability Claims. SSRN. https://ssrn.com/abstract=3164764 Accessed: 14 October 2019.
- Wüthrich, M. and C. Buser. 2018. Data analytics for non-life insurance pricing. Swiss Finance Institute Research Paper. https://ssrn.com/abstract=2870308. Accessed: 14 October 2019.
- EAA seminar: Mitevski, M. and Thomas J. Neural Networks and Deep Learning in Insurance – Theory and Practice