# F. Chollet / J.J. Allaire
# Deep Learning with R
# Chapter 9: Conclusions

---

Konstantin Strauch

Machine Learning Club @ IMBEI

22.03.2021

# What should we take home?

- Deep learning ≠ AI or machine learning
- AI: "all attempts to automate cognitive processes"
  – in other words, the automation of thought
- Machine learning:

  subfield of AI focused on automatically developing programs (called *models*) purely from exposure to training data. This process is called *learning*.
- Deep learning:

  branch of machine learning, where the models are long chains of geometric functions, applied one after the other. These operations are structured into modules called layers: deep-learning models are typically stacks of layers – or, more generally, graphs of layers. These layers are parameterized by weights, which are the parameters learned during training. The knowledge of a model is stored in its weights, and the process of learning consists of finding good values for these weights.
- Deep learning is a breakout success

# What makes deep learning special?

- Deep learning: tremendous breakthroughs across wide range of tasks historically perceived as extremely difficult for computers, especially in the area of machine perception: extracting useful information from images, videos, sound, and more. Given sufficient training data (labeled by humans), it's possible to extract from perceptual data almost anything that a human could extract.
- Currently the third and by far the largest *AI summer*
- Human-level speech recognition, smart assistants, image classification, lang. translation
- Hype: may recede, but sustained economic & technological impact of deep learning will remain – analogous to the internet
- Deploying existing algorithms to every applicable problem:
  game changer for most industries, revolution, fast progress
- Exponential investment in resources and headcount
- Deploying deep learning to the full extent of its potential will take over a decade

# Concept of deep learning (1)

- Conceptually simple: all you need is sufficiently large parametric models trained with gradient descent on sufficiently many examples.
- Feynman once said about the universe, “It’s not complicated, it’s just a lot of it.”
- Model inputs (text, images, …) and targets are first vectorized: turned into an initial input vector space and target vector space (vector: point in a geometric space).
- Each layer: one simple geometric transformation on the data that goes through it
- The chain of layers in the model forms one complex geometric transformation, broken down into a series of simple ones.
- This complex transformation attempts to map the input space to the target space, one point at a time. This transformation is parameterized by the weights of the layers, which are iteratively updated based on how well the model is currently performing.
- Geometric transformation must be differentiable, required to learn parameters via gradient descent. Inputs → outputs: smooth and continuous – a significant constraint

# Concept of deep learning (2)

- Concept visualized in 3D by imagining a person trying to uncrumple a paper ball: the crumpled paper ball is the manifold of the input data that the model starts with
- Each movement operated by the person on the paper ball is similar to a simple geometric transformation operated by one layer
- Full uncrumpling gesture sequence is the complex transformation of the entire model
- Deep-learning models are mathematical machines for uncrumpling complicated manifolds of high-dimensional data

# Concept of deep learning (3)

- Core idea: *meaning is derived from the pairwise relationships between things* (between words in a language, between pixels in an image, and so on) and that *these relationships can be captured by a distance function.*
- Entirely separate question whether the brain implements meaning via geometric spaces
- Vector spaces: efficient to work with from a computational standpoint, but different data structures for intelligence can easily be envisioned – in particular, graphs.
- Neural networks initially emerged from the idea of using graphs as a way to encode meaning, which is why they're named neural networks
- Name "neural network" exists purely for historical reasons –
  extremely misleading name because they're neither neural nor networks.
- More appropriate name would have been layered representations learning or hierarchical representations learning, or maybe even deep differentiable models or chained geometric transforms, to emphasize the fact that continuous geometric space manipulation is at their core.

# Key enabling technologies

- Incremental algorithmic innovations, first spreading over two decades (starting with backpropagation) and then happening increasingly faster as more research effort was poured into deep learning after 2012.
- The availability of large amounts of perceptual data, which is a requirement in order to realize that sufficiently large models trained on sufficiently large data are all we need. This is in turn a byproduct of the rise of the consumer internet and Moore's law applied to storage media.
- The availability of fast, highly parallel computation hardware at a low price, especially the GPUs produced by NVIDIA – first gaming GPUs and then chips designed from the ground up for deep learning. Early on, NVIDIA CEO Jensen Huang took note of the deep-learning boom and decided to bet the company's future on it.
- Complex stack of software layers that makes this computational power available to humans: the CUDA language, frameworks like TensorFlow that do automatic differentiation, and Keras, which makes deep learning accessible to most people.

# Perspective

- In the future, deep learning will not only be used by specialists, but will also be a tool in the toolbox of every developer, much like web technology today.
- This requires us to build tools that make deep learning radically easy to use and accessible to anyone with basic coding abilities.
- Keras is the first major step in that direction.

# Universal machine-learning workflow (1)

Difficult part of ML workflow: everything before designing and training such models – and [maybe not only?] for production models, what comes after, as well.

1. Define the problem: What data is available, and what are you trying to predict?
2. Identify a way to reliably measure success on your goal. For simple tasks, this may be prediction accuracy, but often more sophisticated domain-specific metrics are required.
3. Prepare the validation process that you'll use to evaluate your models.
   In particular, you should define a training set, a validation set, and a test set.
4. Vectorize the data by turning it into vectors and preprocessing it in a way that makes it more easily approachable by a neural network (normalization, and so on.
5. Develop a first model that beats a trivial common-sense baseline, thus demonstrating that machine learning can work on your problem. This may not always be the case!

# Universal machine-learning workflow (2)

6. Gradually refine your model architecture by tuning hyperparameters and adding regularization. Make changes based on performance on the validation data only, not the test data or the training data. Remember that you should get your model to overfit (thus identifying a model capacity level that's greater than you need) and only then begin to add regularization or downsize your model.
7. Be aware of validation-set overfitting when tuning hyperparameters: the fact that your hyperparameters may end up being overspecialized to the validation set. Avoiding this is the purpose of having a separate test set!

# Key network architectures

- Three important families of network architectures:
  densely connected networks, convolutional networks, recurrent networks
- Each type addresses a specific input modality and encodes assumptions about the data
- Whether a given architecture will work on a given problem depends entirely on the match between the structure of the data and the assumptions of the network architecture.
- Different network types can be easily combined to achieve larger multi-modal networks (like LEGO bricks).

# Input modalities & network architectures

- *Vector data* – Densely connected network (dense layers)
- *Image data* – 2D convnets
- *Sound data (for example, waveform)* – Either 1D convnets (preferred) or RNNs
- *Text data* – Either 1D convnets (preferred) or RNNs
- *Timeseries data* – Either RNNs (preferred) or 1D convnets
- *Other types of sequence data* – Either RNNs or 1D convnets. Prefer RNNs if data ordering is strongly meaningful (for example, for timeseries, but not for text)
- *Video data* – Either 3D convnets (if you need to capture motion effects) or a combination of a frame-level 2D convnet for feature extraction followed by either an RNN or a 1D convnet to process the resulting sequences
- *Volumetric data* – 3D convnets

# Densely connected networks (1)

- Stack of dense layers, meant to process vector data (batches of vectors)
- Densely connected networks assume no specific structure in the input features: they're called *densely connected* because the units of a dense layer are connected to every other unit.
- The layer attempts to map relationships between any two input features; this is unlike a 2D convolution layer, for instance, which only looks at local relationships
- Most commonly used for categorical data
- Also used as final classification or regression stage of most networks, e.g. convnets (Chapter 5) or recurrent networks (Chapter 6).

# Densely connected networks (2)

- To perform binary classification, end your stack of layers with a dense layer with a single unit and a sigmoid activation, and use binary_crossentropy as the loss. Targets: 0 or 1
- To perform single-label categorical classification (where each sample has exactly one class), end your stack of layers with a dense layer with a number of units equal to the number of classes, and a softmax activation. If the targets are one-hot encoded, use categorical_crossentropy as loss; with integers, use sparse_categorical_crossentropy.
- To perform multilabel categorical classification (where each sample can have several classes), end your stack of layers with a dense layer with a number of units equal to the number of classes and a sigmoid activation, and use binary_crossentropy as the loss. The targets should be one-hot encoded.
- To perform regression toward a vector of continuous values, end your stack of layers with a dense layer with a number of units equal to the number of values you're trying to predict (often a single one), and no activation. Several losses can be used for regression, most commonly mean_squared_error (MSE) and mean_absolute_error (MAE).

# Convolutional networks (Convnets)

- Convolutional layers look at spatially local patterns by applying the same geometric transformation to different spatial locations (*patches*) in an input tensor.
- Representations are *translation invariant*, highly data efficient and modular.
- Idea applicable to spaces of any dimensionality: 1D (sequences), 2D (images), 3D (volumes), and so on.
- Convnets consist of stacks of convolution and maxpooling layers. The pooling layers let you spatially downsample the data, which is required to keep feature maps to a reasonable size as the number of features grows, and to allow subsequent convolution layers to “see” a greater spatial extent of the inputs.
- Convnets are often ended with either a layer_flatten layer or a global pooling layer, turning spatial feature maps into vectors, followed by dense layers to achieve classification or regression.

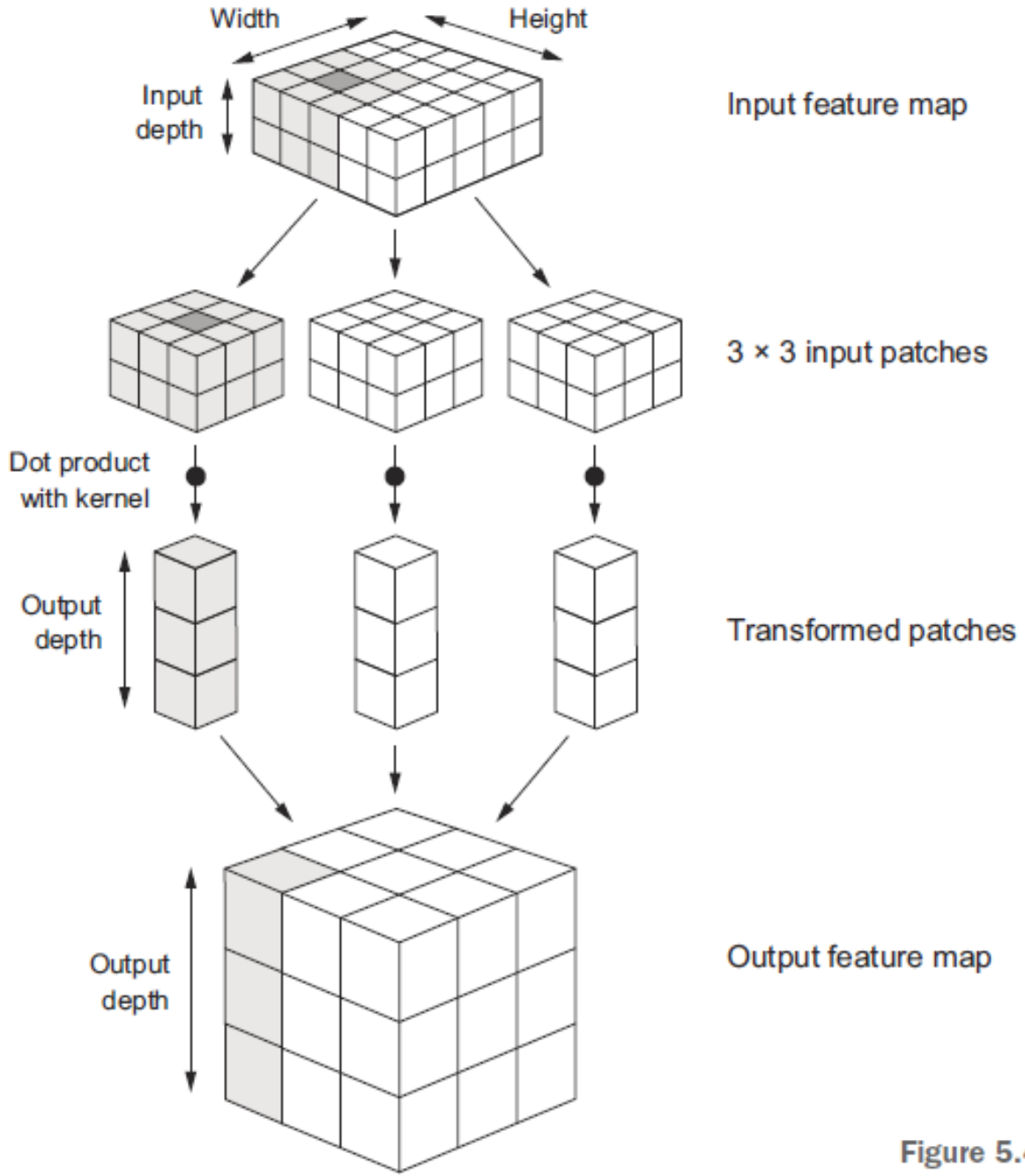



Figure 5.4 How convolution works

# Recurrent neural networks (RNNs)

- RNNs work by processing sequences of inputs one timestep at a time and maintaining a state throughout (a state is typically a vector or set of vectors: a point in a geometric space of states).
- They should be used preferentially over 1D convnets in the case of sequences where patterns of interest are not invariant by temporal translation (for instance, timeseries data where the recent past is more important than the distant past).

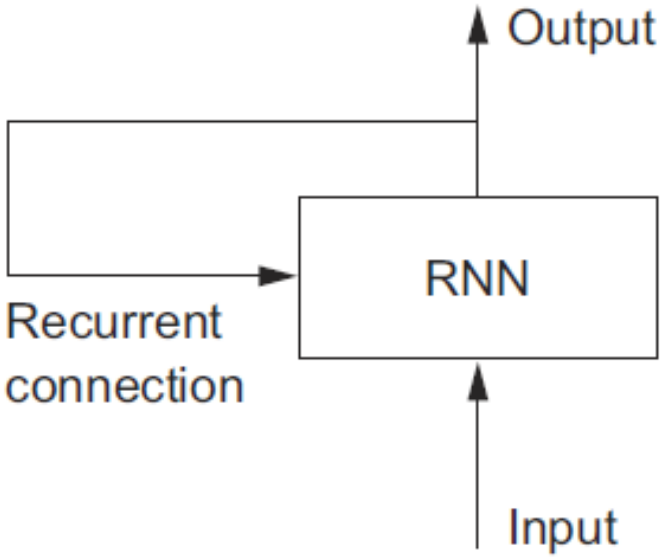



**Figure 6.7 A recurrent network: a network with a loop**

# The space of possibilities

- Mapping vector data to vector data:
  Predictive healthcare, behavioral targeting, product quality control
- Mapping image data to vector data:
  Medical assistant, self-driving car, board-game AI (go/chess), diet helper, age prediction
- Mapping time-series data to vector data:
  Weather prediction, brain-computer interfaces, behavioral targeting
- Mapping text to text: smart reply, answering questions, summarization
- Mapping images to text: captioning
- Mapping text to images: conditional image generation, logo generation/selection
- Mapping images to images: super-resolution, visual depth sensing
- Mapping images and text to text: visual question answering
- Mapping video and text to text: video question answering

# Limitations of deep learning (1)

- Developing source code to meet certain requirements
  – impossible, no matter how big the training data set!
- In general, anything that requires reasoning, long-term planning, and algorithmic data manipulation is out of reach for deep-learning models.
- Even learning a sorting algorithm with a deep neural network is tremendously difficult.
- A deep-learning model is just a chain of simple, continuous geometric transformations mapping one vector space into another.
- A deep-learning model can be interpreted as kind of program; but, inversely, most programs can’t be expressed as deep-learning models. The corresponding geometric transform may be far too complex, or there may not be appropriate data available to learn it.
- Scaling up current deep-learning techniques by stacking more layers and using more training data can only superficially palliate some of these issues & won’t solve the more fundamental problems that deep-learning models are limited.

# Limitations of deep learning (2)

- Risk of anthropomorphizing machine-learning models

“The boy is holding a baseball bat”

# Limitations of deep learning (3)

- Adversarial example:

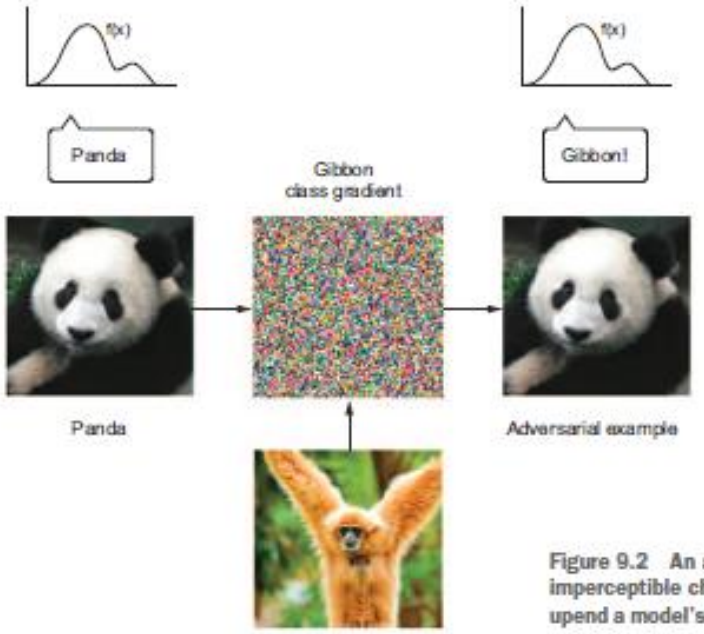
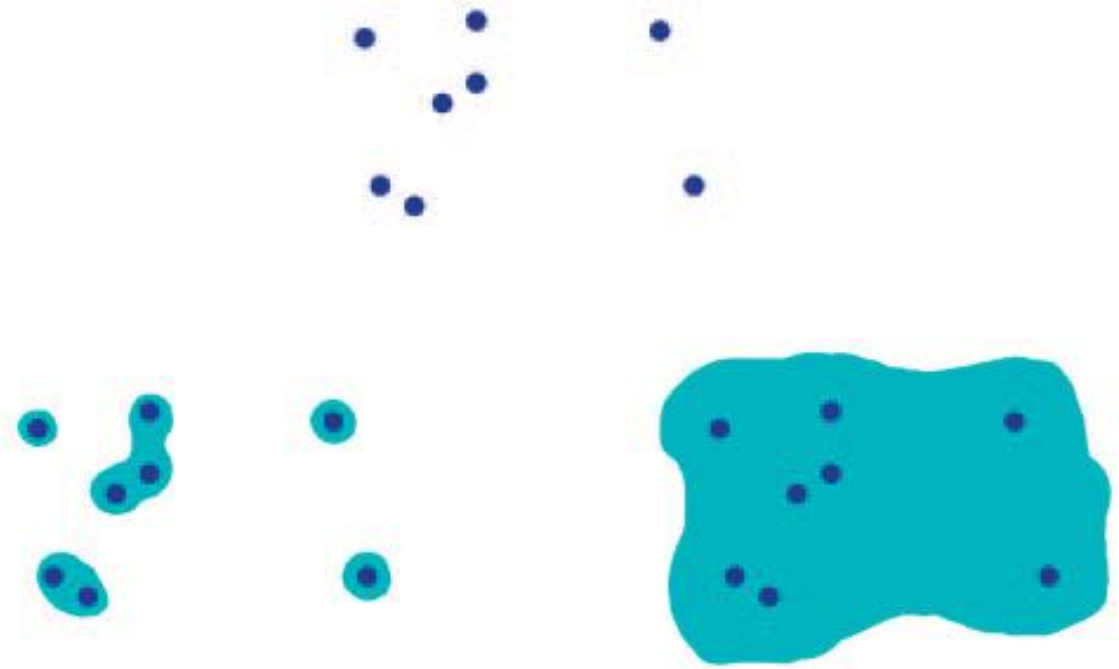



Figure 9.2 An adversarial example: imperceptible changes in an image can upend a model's classification of the image.

- Deep-learning models don't have any understanding of their input – at least, not in a human sense.
- Lack of abstraction!

# Local vs. extreme generalization



Local generalization: generalization power of pattern recognition

Extreme generalization: generalization power achieved via abstraction and reasoning

- Only real success of deep learning: the ability to map space X to space Y using a continuous geometric transform, given large amounts of human-annotated data. This allows (only) for local generalization, as depicted above.
- Doing this well is a game changer for essentially every industry, but it's still a long way from human-level AI, i.e., from human intelligence.

# The future of deep learning (1)

- Models closer to general purpose computer programs
- New forms of learning that make the previous point possible, allowing models to move away from differentiable transforms
- Models that require less involvement from human engineers (no tuning of knobs by hand)
- Greater, systematic reuse of previously learned features and architectures, such as meta-learning systems based on reusable and modular program subroutines

→ Move toward models capable of abstraction and reasoning that can achieve extreme generalization

- Stacking of coding elements: *for* loops, if / then statements, genetic search / programming
- Blend of formal algorithmic modules that provide reasoning and abstraction capabilities, and (classical deep-learning) geometric modules that provide informal intuition and pattern-recognition capabilities.
- Emergence of a crossover subfield between deep learning and program synthesis

# The future of deep learning (2)

- Non-differentiable functions
- Introduction of modularity and hierarchy
- Automated machine learning
- Lifelong learning and modular subroutine reuse
  – just as re-use of software packages today
- Long-term vision:
  – Models will be more like programs, with capabilities far beyond currently available continuous geometric transformations of the input data – closer to human abstraction
  – Models will blend algorithmic modules providing formal reasoning, search & abstraction with geometric modules providing informal intuition & pattern-recognition capabilities
  – Models will be grown automatically, using modular parts stored in a global library
  – Global library & model-growing system will be able to achieve some form of human-like extreme generalization – can be interpreted as *artificial general intelligence (AGI)*
- Staying up to date: Kaggle, arXiv, Keras, …