E.T.S.E.T.

Universidade deVigo

Deep Learning (Intro)

Author: Juan Carlos Burguillo Rial

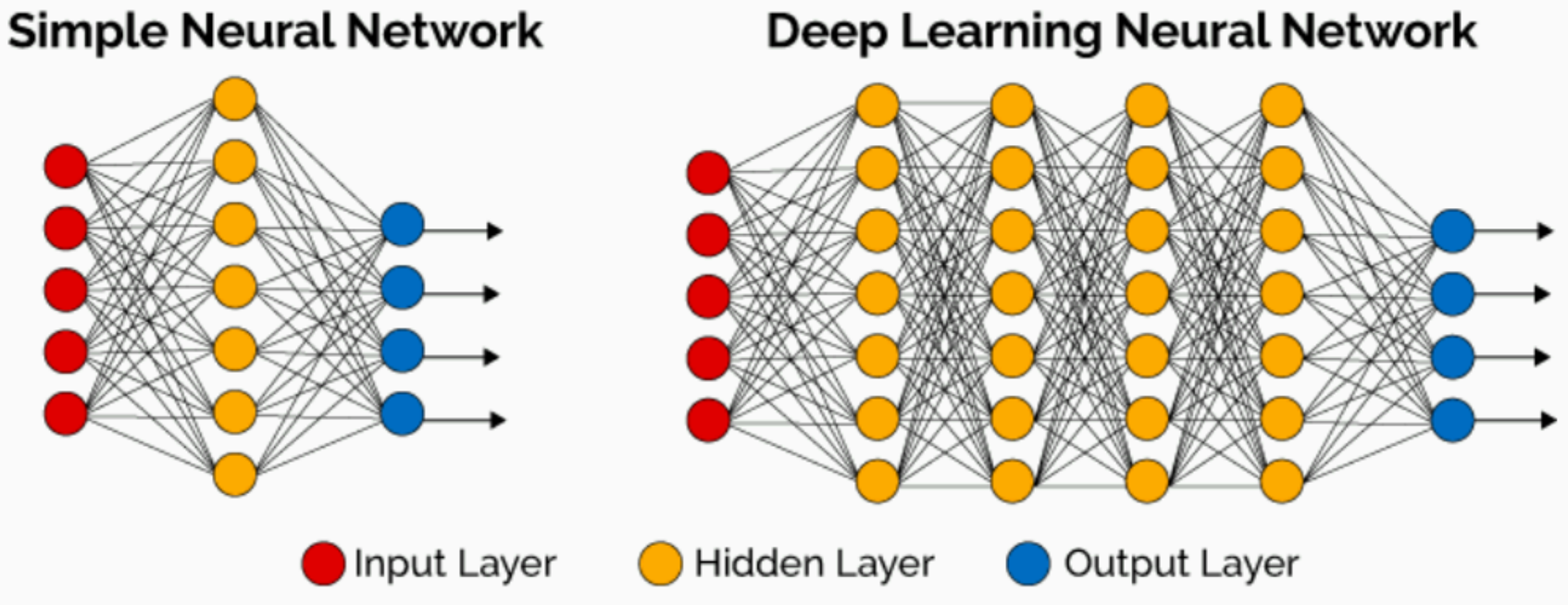
Departament Telematic Engineering

Universidade de Vigo

J.C.Burguillo@det.uvigo.es

http://www.det.uvigo.es/~jrial

1. Deep Learning Neural Networks



Training deep learning networks produces de **vanishing gradient problem**, resulting in a very slow process; so GPUs are needed for parallelizing.

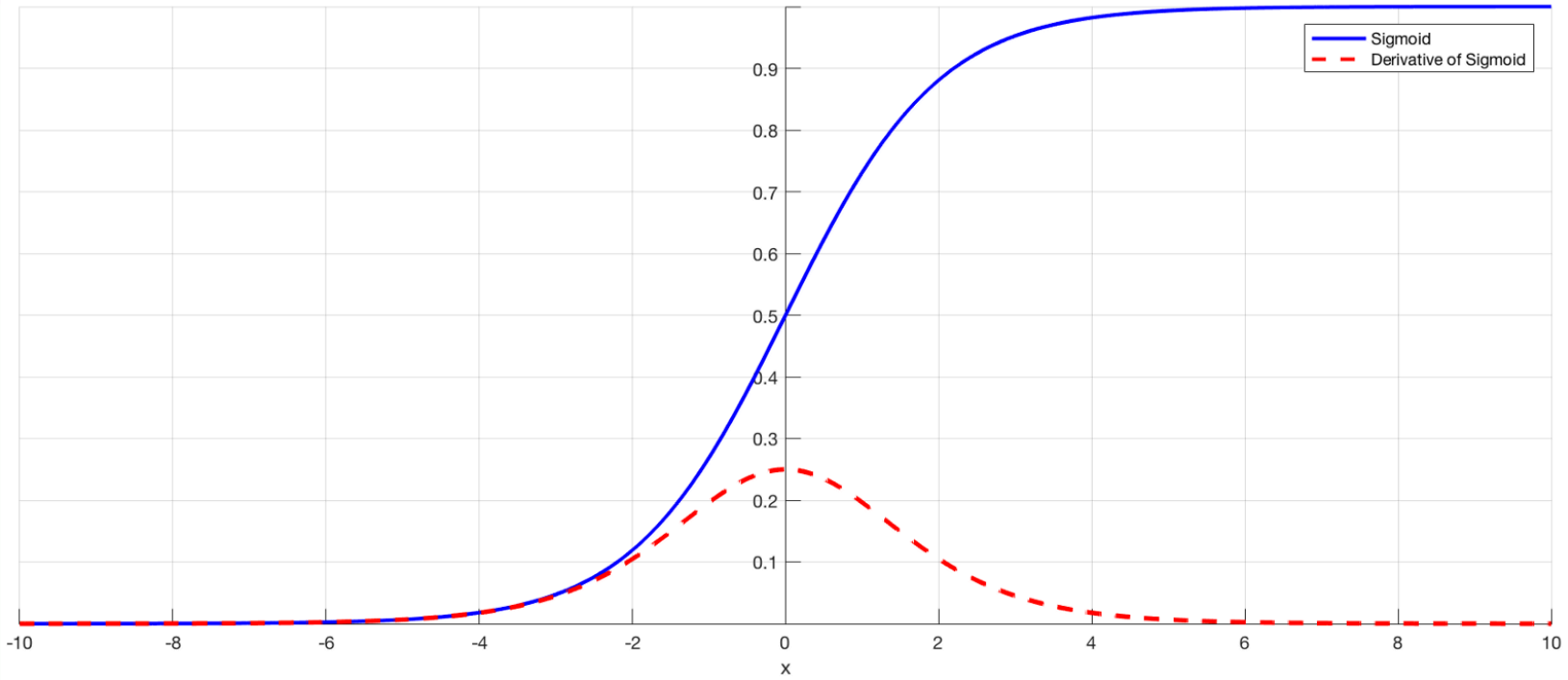
1. The Vanishing Gradient Problem

- The vanishing gradient problem is a difficulty found in training networks with **gradient-based learning methods and backpropagation**.

- In such methods, each of the neural network's weights receives an update proportional to the gradient of the error function with respect to the current weight in each iteration of training.

- Traditional activation functions such as the hyperbolic tangent function have gradients in the range (−1, 1), and **backpropagation computes gradients by the chain rule**.

1. The Vanishing Gradient Problem



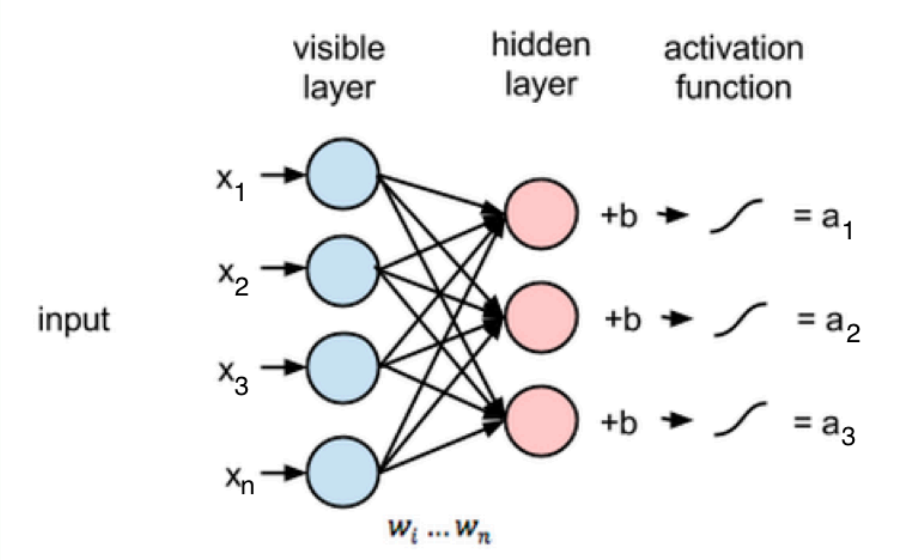
This has the effect of multiplying *k* of these small numbers in the interval [-1, 1] to compute gradients of the "front" layers in an *k*-layer network, meaning that the gradient (error signal) decreases exponentially with *k* so **the front layers change very slowly**.

2. Restricted Boltzmann Machine (RBM)

- A Restricted Boltzmann Machine is a method to automatically find patterns in the data by reconstructing the input.

- An RBM is a shallow, two-layer net. The first layer is known as the visible layer and the second is called the hidden layer.

- Each node in the visible layer is connected to every node in the hidden layer. An RBM is “restricted” because nodes don’t share connections in the same layer.



2. RBM as a Translator

- An RBM is the mathematical equivalent of a **two-way translator**:
  - In the **forward pass**, an RBM takes the inputs and translates them into a set of numbers that encode the inputs.
  - In the **backward pass**, it takes this set of numbers and translates them back to form the re-constructed inputs. A well-trained net will be able to perform the reconstruction with a high degree of accuracy.

- In both steps, the weights and biases have a very important role. They allow the **RBM to decipher the interrelationships** among the input features, and they also help the RBM to decide which input features are the most important when **detecting patterns**.

2. RBM: Training

- Through several forward and backward passes, an RBM is trained to reconstruct the input data. Three steps are repeated over and over through the training process:

  1. With a forward pass, every input is combined with an individual weight and one overall bias, and the result is passed to the hidden layer which may or may not activate.
  2. Next, in a backward pass, each activation is combined with an individual weight and an overall bias, and the result is passed to the visible layer for reconstruction.
  3. At the visible layer, the reconstruction is compared (using the KL Divergence metric) with the original input to determine the quality of the result.

- Steps 1) to 3) are repeated changing weights and biases until the input and the re-construction are as close as possible.

3. RBMs and Autoencoders

- An interesting aspect of an RBM is that the data does not need to be labeled. This turns out to be very important for real-world data sets like photos, videos, voices, and sensor data – as all are usually unlabeled.

- RBM is part of a family of feature extractor neural nets, which are all designed to recognize inherent patterns in data. These nets are also called autoencoders, because they encode data into their own structure.

- **Autoencoders** are an important family of neural networks that are well-suited to figure out the underlying structure of a data set.

- The RBM is the most popular one, but there are other types of autoencoders like: denoising or contractive (just to name a few).

3. RBMs and Autoencoders

- An autoencoder takes a set of unlabeled inputs, and after encoding them, is able to reconstruct them (almost) accurately.

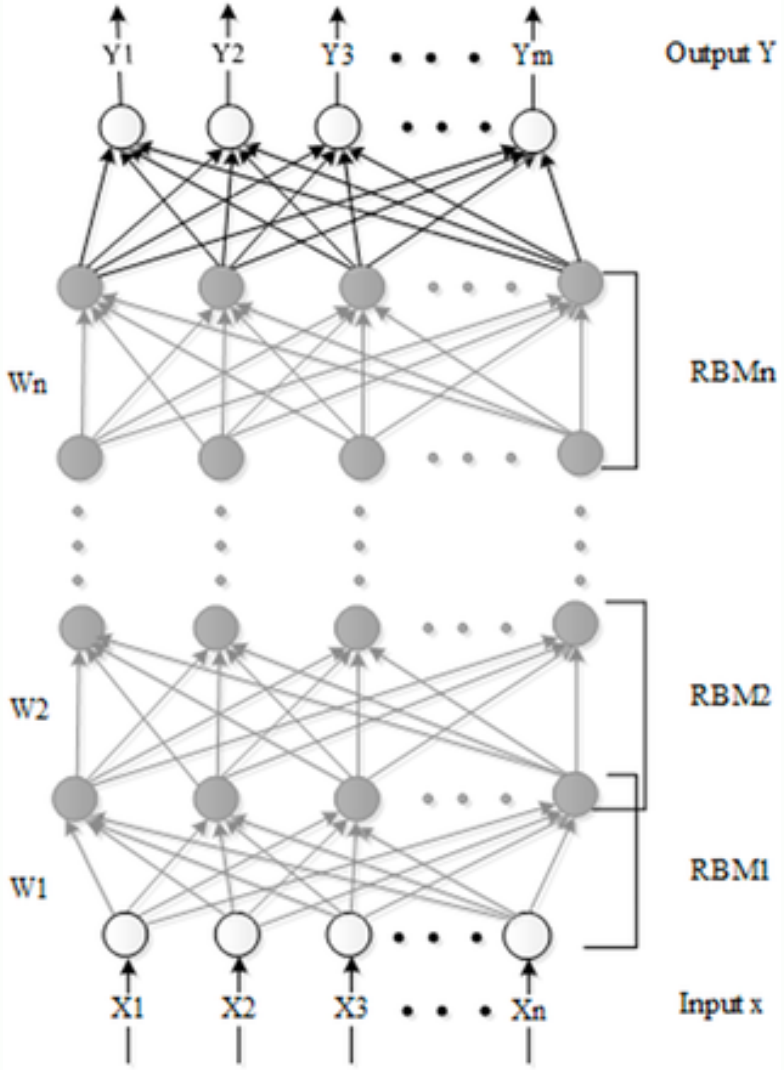
- As a result, the net must decide which of the data features are the most important, essentially acting as a **feature extraction engine**.

- **Autoencoders are typically very shallow**, and are usually comprised of an input layer, an output layer and a hidden layer.

- **Typically, the same weights that are used to encode a feature in the hidden layer are used to reconstruct an image in the output layer**.

3. Deep Belief Networks (DBN)

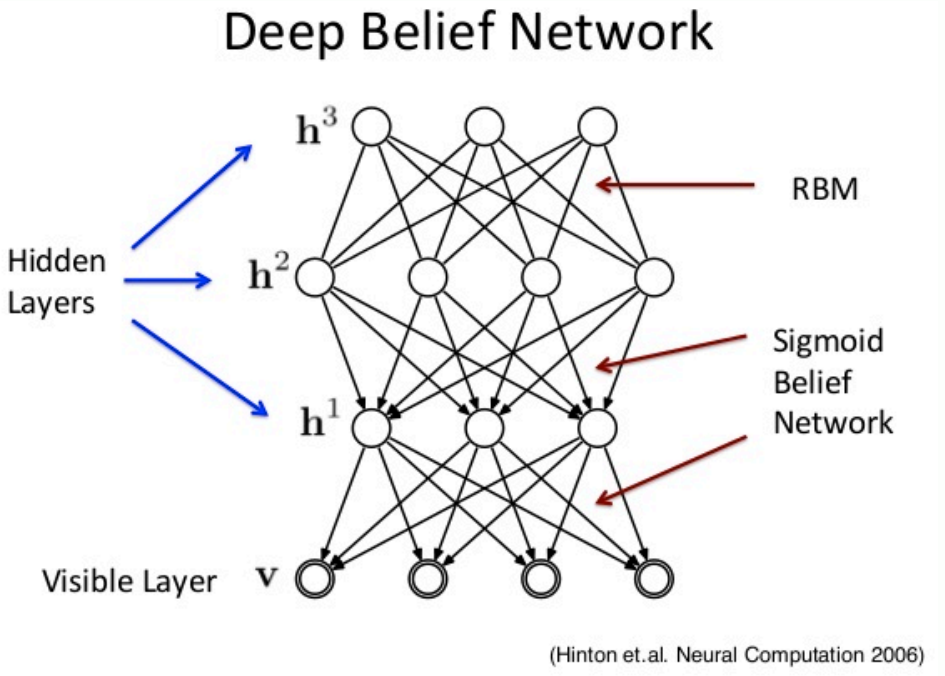
- An RBM can extract features and reconstruct inputs, but how exactly does that help with the vanishing gradient?
- By combining RBMs together and introducing a clever training method, we obtain a powerful new model to solve such problem.
- A deep belief network can be viewed as a stack of RBMs, where the hidden layer of one is the visible layer of the one "above" it.
- DBN are also denoted as Deep Boltzmann Machines.



3. Deep Belief Networks (DBN)

- In terms of network structure, a DBN is identical to an MLP. But when it comes to training, they are entirely different.

- In fact, the difference in training methods is the key factor that enables DBNs to outperform their shallow counterparts.



3. DBN: Training 1/2

- A DBN is trained as follows:

1. The first RBM is trained to re-construct its input as accurately as possible.

2. The hidden layer of the first RBM is treated as the visible layer for the second and the second RBM is trained using the outputs from the first RBM.

3. This process is repeated until every layer in the network is trained.

3. DBN: Training 2/2

- **To finish training**, we need to **introduce labels** to a small set of samples and fine-tune the net with supervised learning. Using this approach patterns can be associated with a name.

- Weights and biases are altered slightly, resulting in small changes in the pattern perception, and often a small increase in the total accuracy.

- As stated, **the set of labeled data is usually smaller** than the original dataset, and this is extremely helpful in real-world apps.

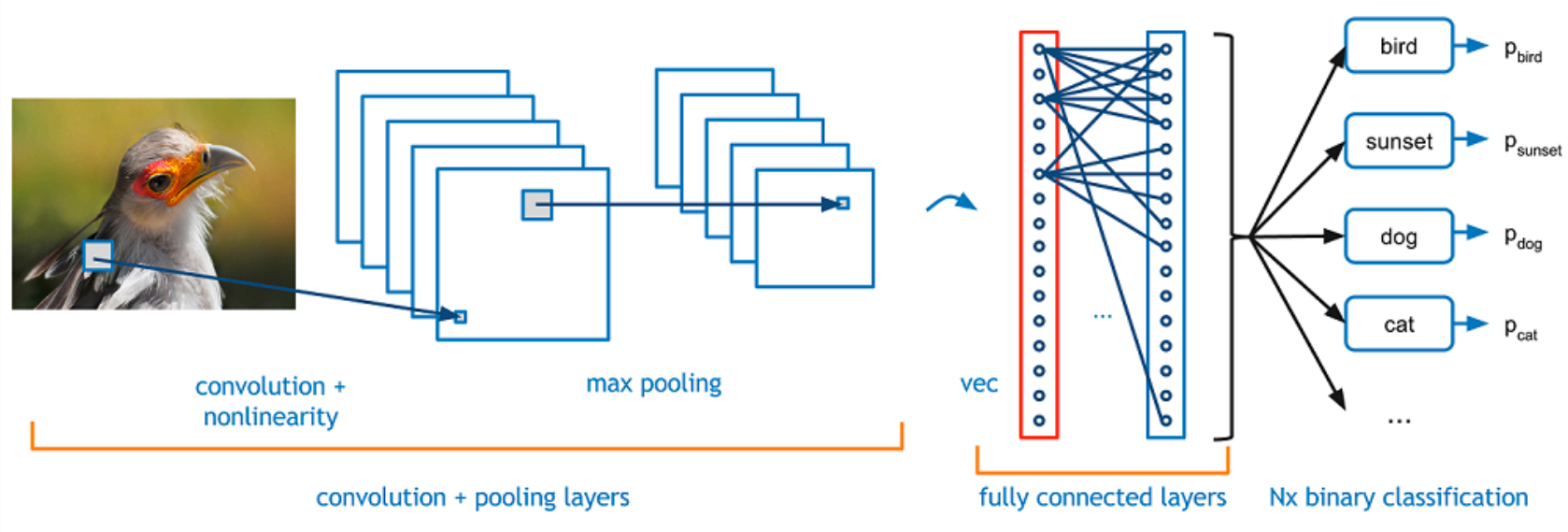
- After full training, the DBN has created a model that can detect inherent patterns in the data.

3. DBN: Performance

- A DBN, being a stack of RBMs, outperforms a single RBM – just like a MLP outperforms a single perceptron.
- The training process can be completed in a reasonable amount of time through the use of GPUs.
- A DBN works globally by fine tuning the entire input in succession as the model slowly improves –like a camera lens slowly focusing a picture.
- In other models – like convolutional nets – early layers detect simple patterns and later layers recombine them. For instance, in facial recognition, early layers would detect edges in the image, and later layers would use these results to form facial features.

4. Convolutional Neural Nets (CNN)

- CNNs have completely dominated machine vision in recent years.
- In early 2015, a CNN was able to beat a human at an object recognition challenge for the first time in the history of AI.



4. CNN: Convolutional Layer

- The CNN performs a **convolution** to search for a particular pattern, and this is done by neurons at the convolutional layer.

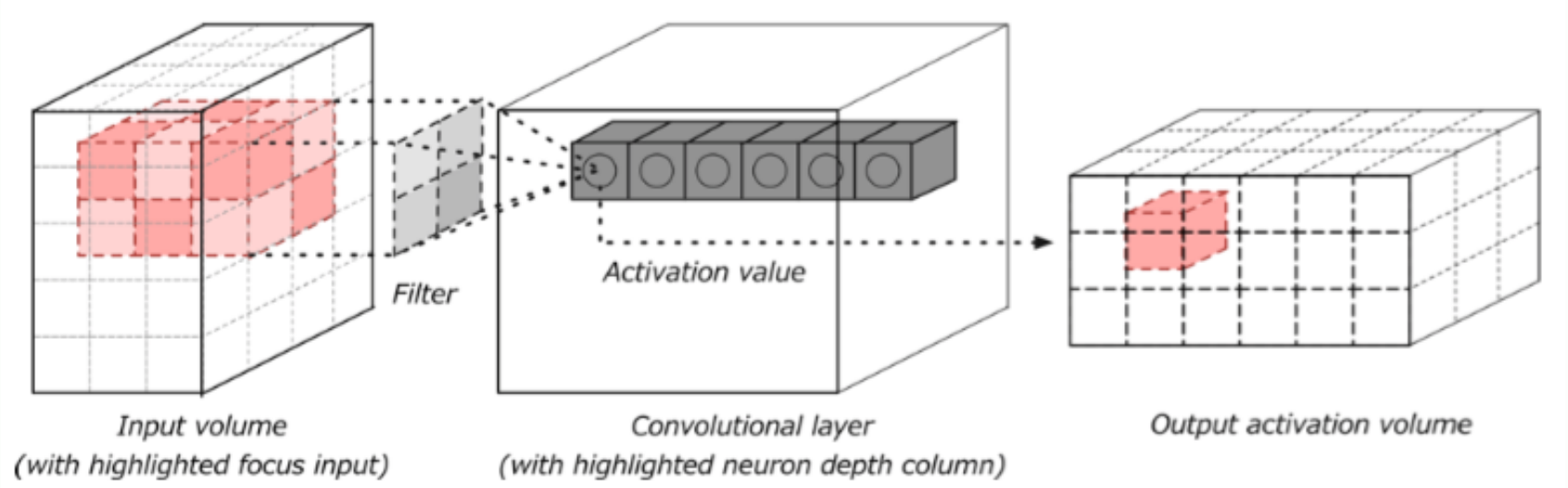
- Each neuron is only connected to some neurons in the next layer, and the whole structure works as a **spatial filter**.

- Neurons in a given filter share the same weight and bias parameters. This means that, anywhere on the filter, a given neuron is connected to the same number of input neurons and has the same weights and biases.

- This allows the filter to look for the same pattern in different sections of the image. By arranging these neurons in a grid structure we ensure that the entire image is scanned.

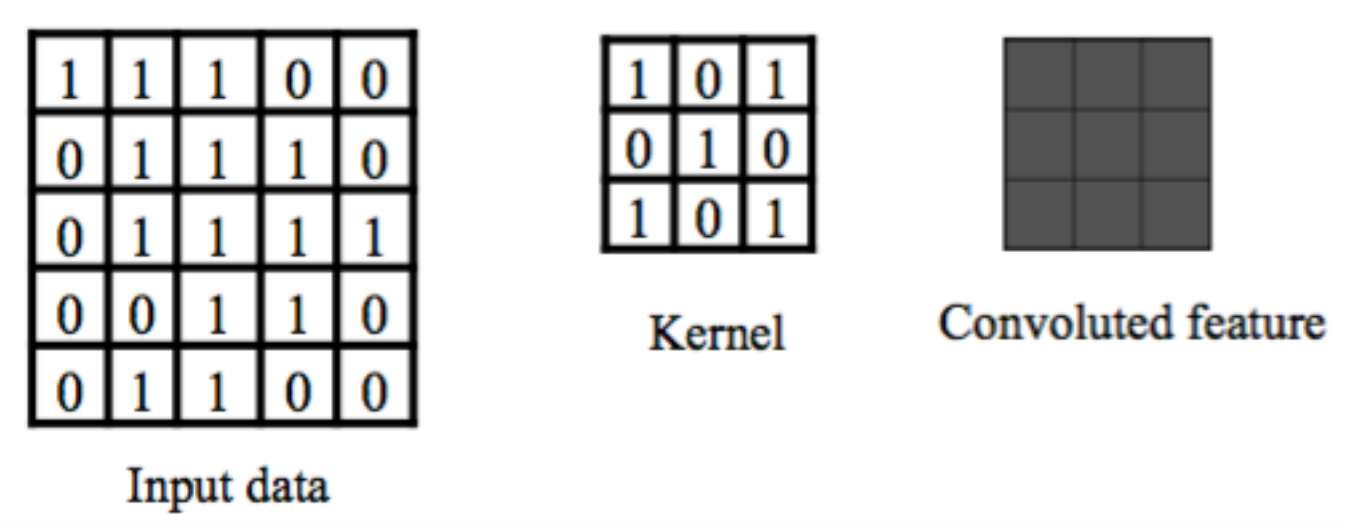
4. CNN: Convolutional Layer

- Input layers store the raw input data of the image that specifies the width, height, and number of channels (p.e., 3 for the RGB pixel values).
- Convolutional layers transform the input data by using a set of neurons.
- The CNN layer computes the dot product between the region of the neurons in the input layer and their weights to the output layer.



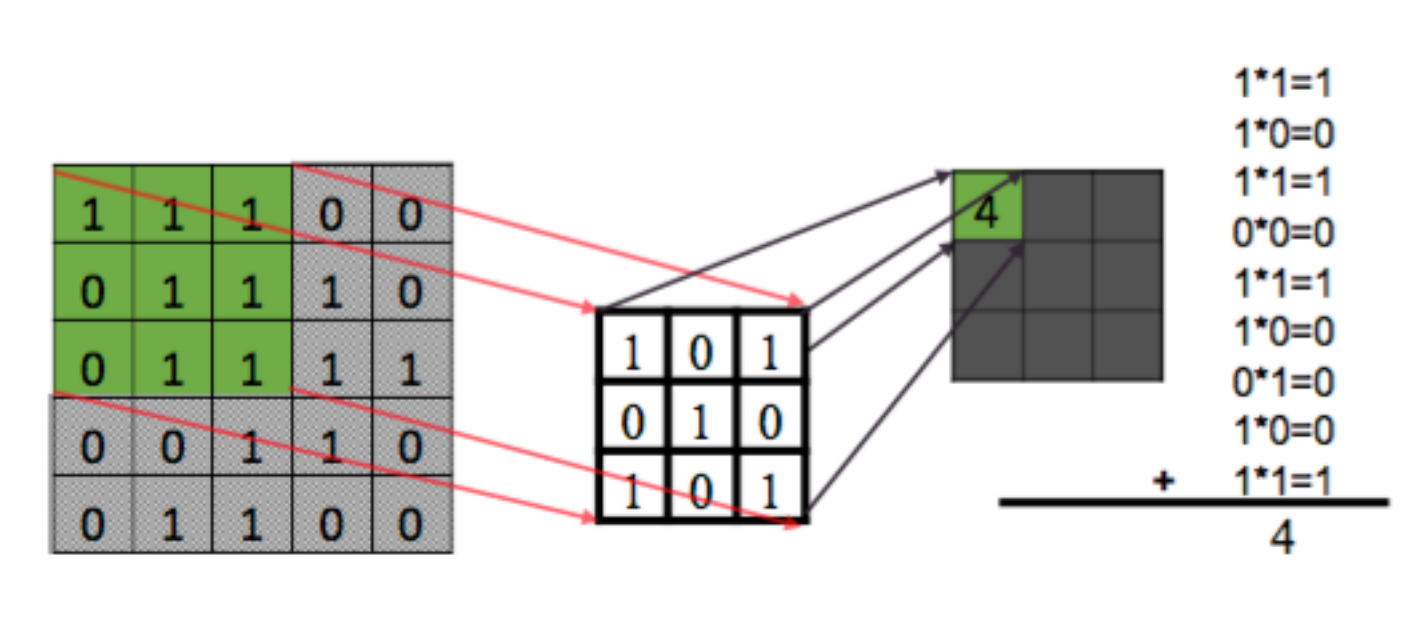
4. CNN: Convolutional Layer – an example

- The kernel is slid across the input data to produce the convoluted feature.
- At each step, the kernel is multiplied by the input data values within its bounds, creating a single entry in the output feature map.
- The set of weights in a convolutional layer is a filter (or kernel), which is convolved with the input and the result is a feature/activation map.



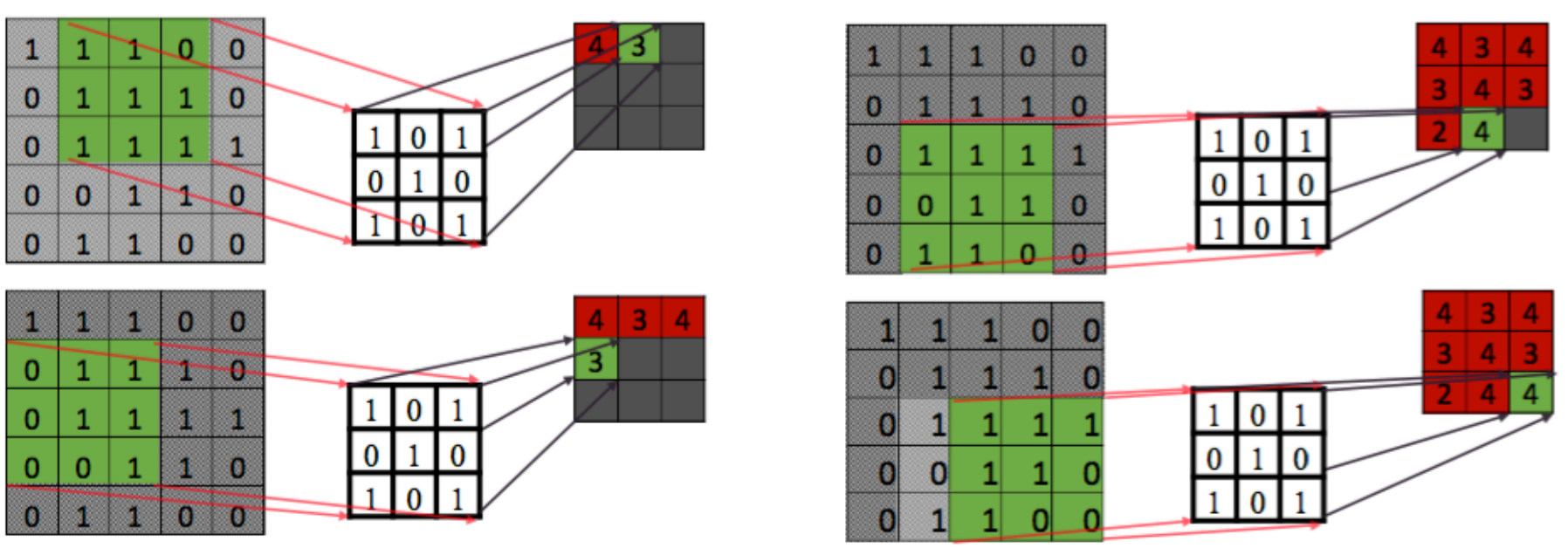
4. CNN: Convolutional Layer – an example

- Lets take it step by step.
- First, apply the kernel to the first patch that matches the kernel size.



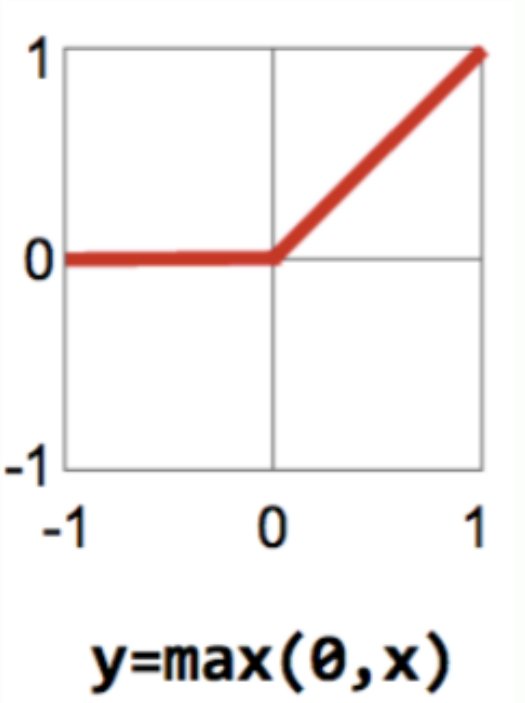
4. CNN: Convolutional Layer – an example

- Now lets slide the kernel over the input image.



4. CNN: ReLU Layer

- Each node in the convolutional layer is connected to a node that fires, using an activation function called ReLU (**Rectified Linear Unit**).
- CNNs are trained using back-propagation, so the vanishing gradient is a potential issue, but given the mathematical definition of the ReLU, the gradient is held more or less constant at every layer of the net.
- The ReLU layer changes the pixel values, but not the spatial dimensions from the input to output.
- Thus the RELU activation allows the net to be properly trained, without harmful slowdowns in the crucial early layers.

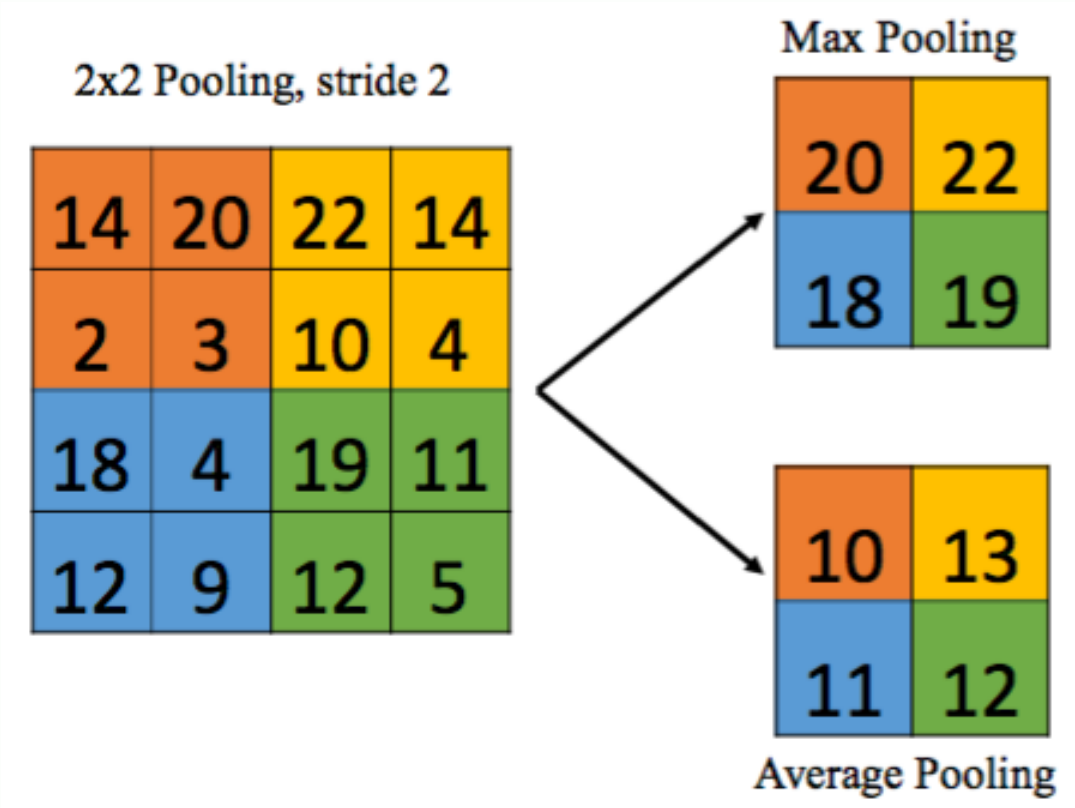


4. CNN: Pooling Layer

- CNNs tile multiple instances of convolutional layers and RELU layers together in sequence, in order to build more and more complex patterns.
- The problem with this is that the number of possible patterns becomes exceedingly large.
- By introducing pooling layers, we ensure that the net focuses on only the most relevant patterns discovered by convolution and RELU.
- **The pooling layer is used for dimensionality reduction**.
- This helps to limit both the memory and the processing requirements for running a CNN.

4. CNN: Pooling Layer

- With a 2 × 2 filter size, the max() operation is taking the largest of four numbers in the filter area, while the average one takes the mean value.



4. CNN: Fully Connected Layers

- These three layers in a CNN can discover complex patterns, but the net will have no understanding of what these patterns mean.

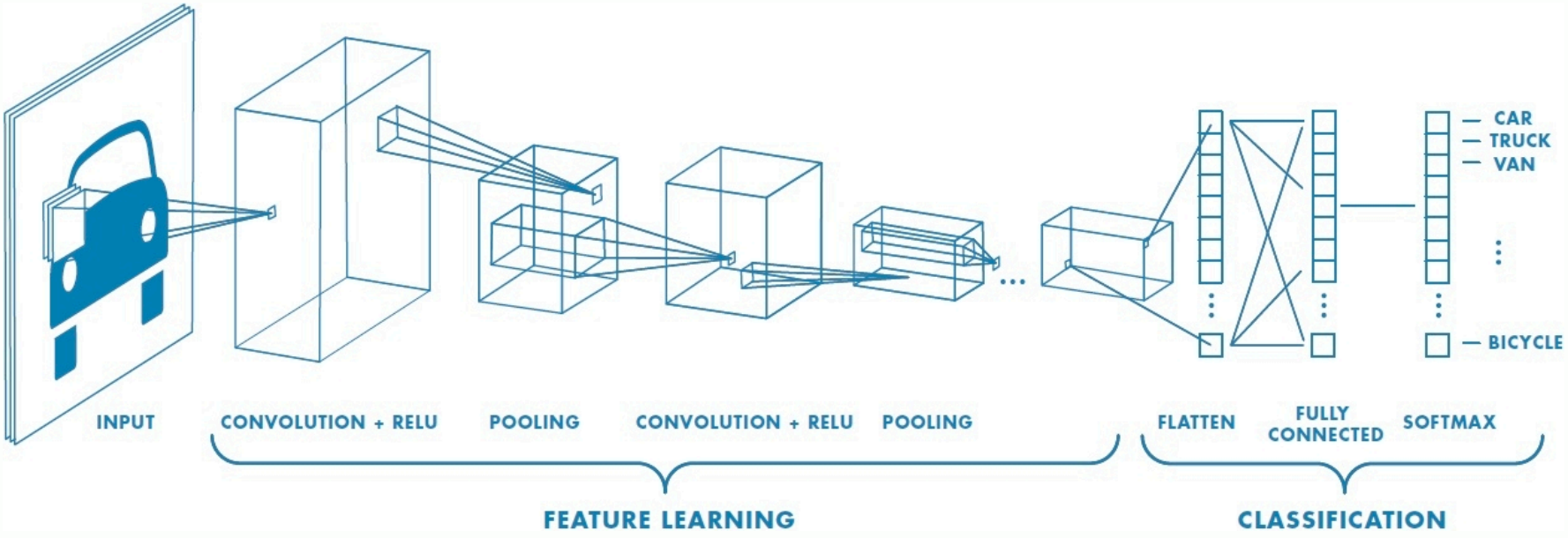
- A **fully connected layer** is attached to the end of the global net to provide the ability to classify data samples.

- Despite the power of CNNs, these nets have one drawback. Since they are a supervised learning method, they require a large set of labeled data for training, which can be challenging to obtain in a real-world application.

4. CNN: Summary

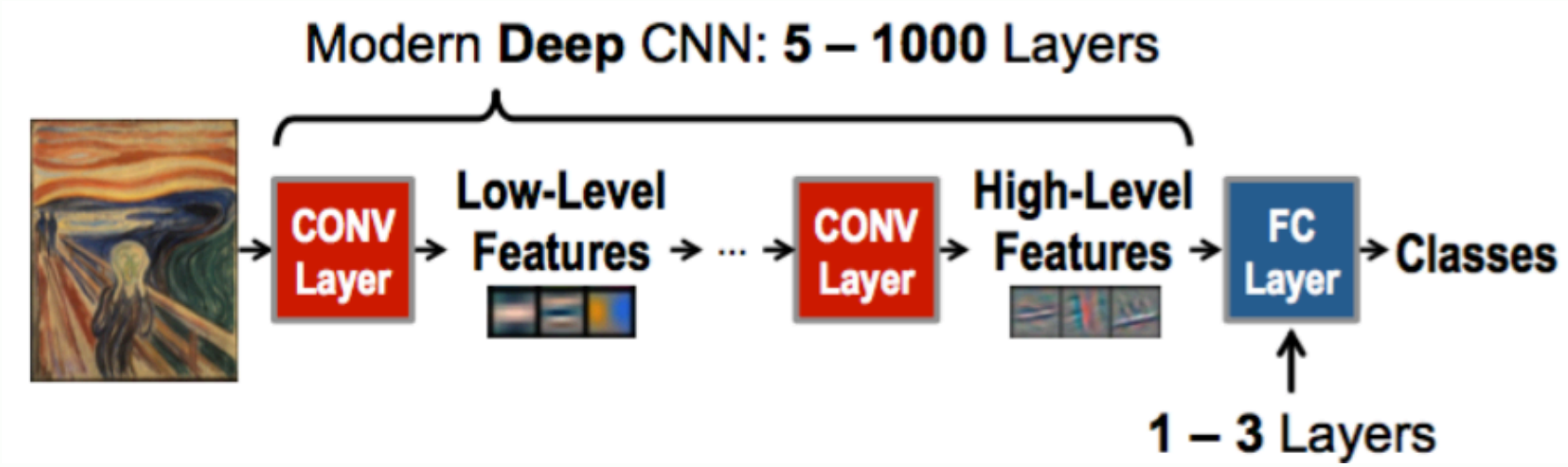
- To sum up, a typical deep CNN has three sets of layers: i) a convolutional layer, ii) a RELU, and iii) a pooling one; which are repeated several times.

- These layers are followed at the end by a few fully connected layers in order to support classification.



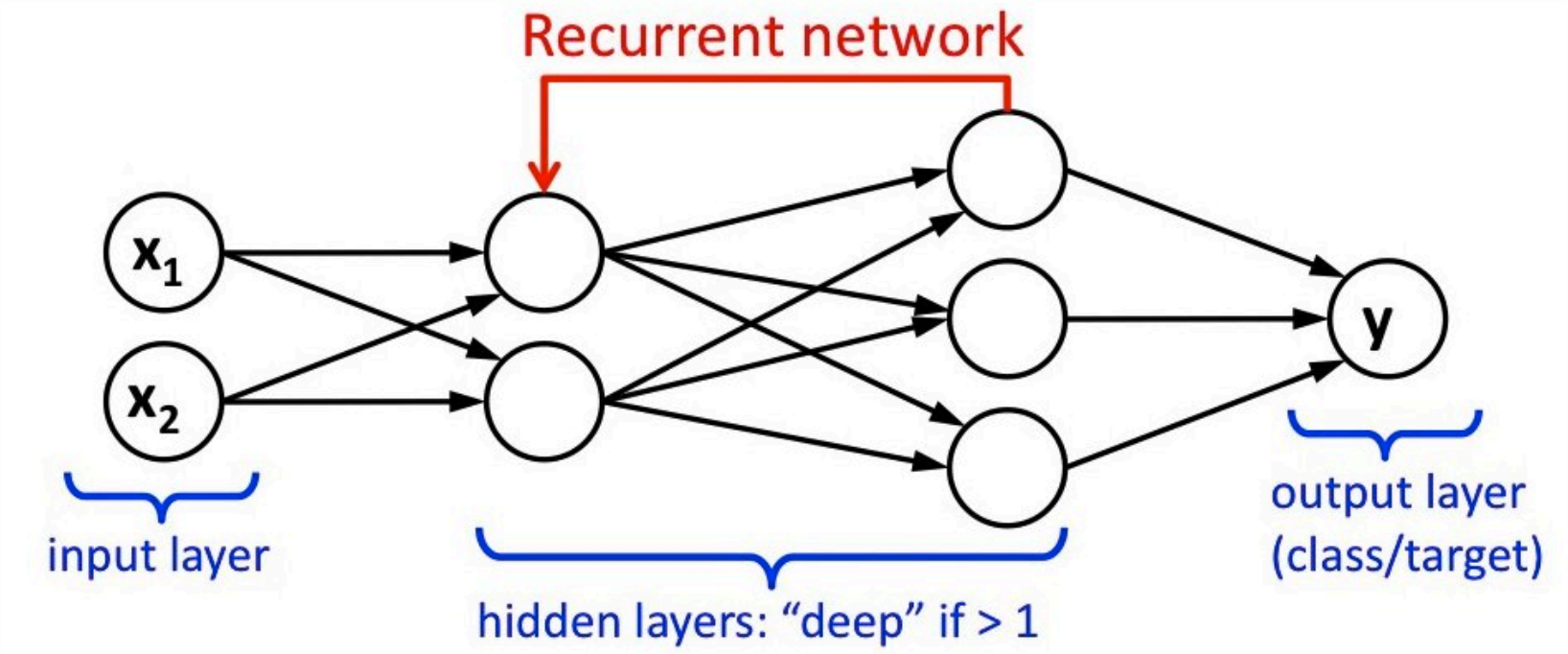
4. CNN: Practical example

- Convolutions take more than 90% of overall computation, dominating runtime and energy consumption.



5. Recurrent Neural Nets (RNN)

- Recurrent neural networks are appropriate **if data patterns change** with time. This deep learning model has a simple structure with a built-in feedback loop, allowing it to act as a **forecasting engine**.



5. RNNs and feedback

- In a feedforward neural network, signals flow in only one direction from input to output, one layer at a time.

- In a recurrent net, the output of a layer is added to the next input and **feedback** into the same layer.

- By **stacking RNNs** on top of each other, you can form a net capable of more complex output than a single RNN working alone.

- A few application examples:
  - If the input is singular and the output a sequence: image captioning.
  - A sequence of inputs with a single output: document classification.
  - When input and output are sequences: video classification (frame by frame).
  - If a time delay is introduced: statistically forecast time series.

5. RNN: Training

- A RNN is an extremely difficult net to train, as these nets use backpropagation, so we have **the problem of the vanishing gradient, which is exponentially worse for an RNN**.

- The reason for this is that each time step is the equivalent of an entire layer in a feedforward network. So training an RNN for 100 time steps is like training a 100-layer feedforward net – this leads to exponentially small gradients and a decay of information through time.

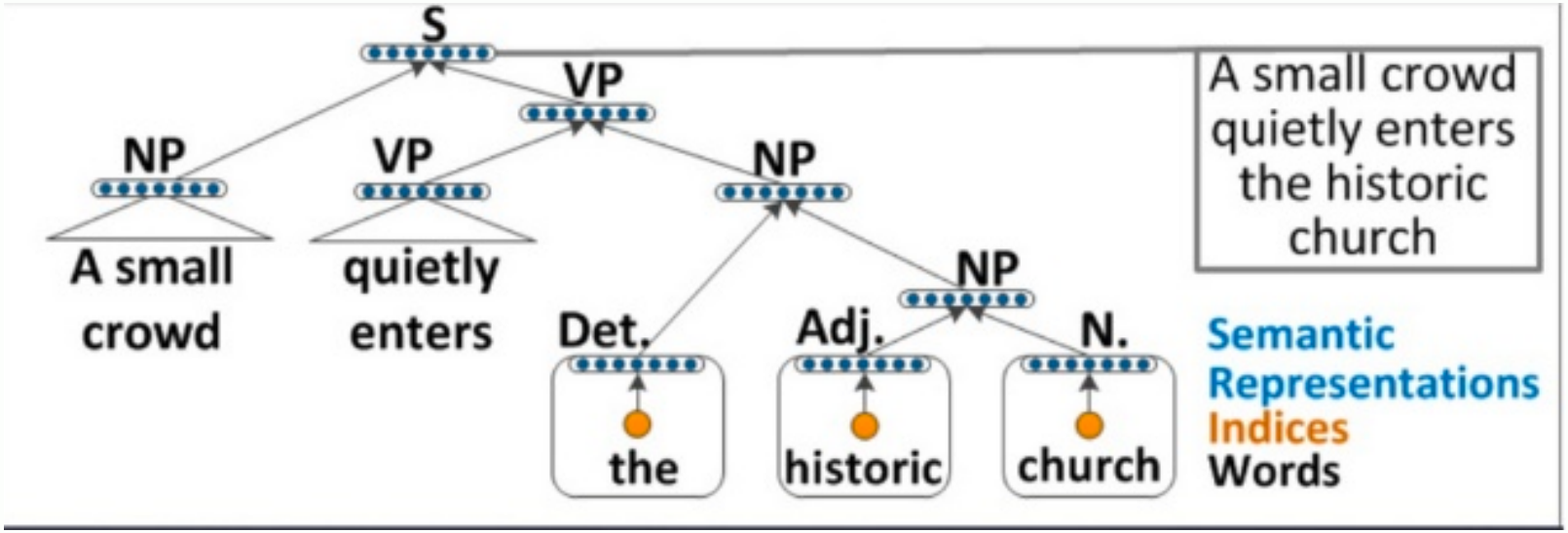
- There are **several ways to address this problem**: gating (the most popular), gradient clipping, steeper gates, and better optimizers.

- **Gating** is a technique that helps the net decide when to forget the current input, and when to remember it for future time steps. The most popular gating types today are **GRU** and **LSTM**.

6. Recurrent Neural Tensor Nets (RNTN)

- RNTNs are better than feedforward or recurrent nets to discover the hierarchical structure of a set of data, such as parsing trees of a group of sentences, i.e., Natural Language Processing (NLP).

- These nets analyze hierarchical structures of data. They were originally designed for sentiment analysis, where the sentiment of a sentence depends on the syntactic order in which the words appear.



6. Recurrent Neural Tensor Nets (RNTN)

- An RNTN has a parent group, which we'll call the **root**, and the child groups, which we'll call the **leaves**. The root is connected to leaves, but the leaves are not connected to each other.

- These components form what's called a **binary tree**. In general, the leaf groups receive input, and the root group uses a classifier to fire out a class and a score.

- Each group is simply a collection of neurons, where the number of neurons depends on the complexity of the input data.

- An RNTN's structure may seem simple, but just like a recurrent net, the complexity comes from the way in which data moves throughout the network. In the case of an RNTN, this process is recursive.

6. RNTN: Training

- **RNTNs use backpropagation** by comparing the predicted sentence structure with the proper one obtained from labeled training data.

- Once trained, the net will give a higher score to structures that are more similar to the parse trees that it saw during training.

- Applications:

  - RNTNs are used in natural language processing for both syntactic parsing and sentiment analysis.

  - They are also used to parse images, typically when an image contains a scene with many different components.

7. Summarizing some DLNs & Applications

- For **unsupervised learning or non-labeled** data we can use Self-organizing Maps (SOM), Restricted Boltzmann Machines (RBM) or Deep Belief Networks (DBN), i.e., auto-encoders.

- For **labeled data and classifying** we have:
  - **Images**: Deep Belief Networks (DBN) and Convolutional Neural Nets (CNN).
  - **Object recognition**: CNNs and Recurrent Neural Tensor Nets (RNTN)
  - **Speech recognition**: Recurrent Neural Networks (RNN).
  - **Text & NLP**: RNNs or RNTNs.

8. Concepts: Loss vs. Cost Functions

- A **loss function** is usually a function defined on a data point, prediction or label, and measures the penalty over such point.

Loss (over sample i) = $|Y_i - \hat{Y}_i|$

- A **cost function** is usually more general, and it might be a sum of loss functions over your training set plus some extra penalty.

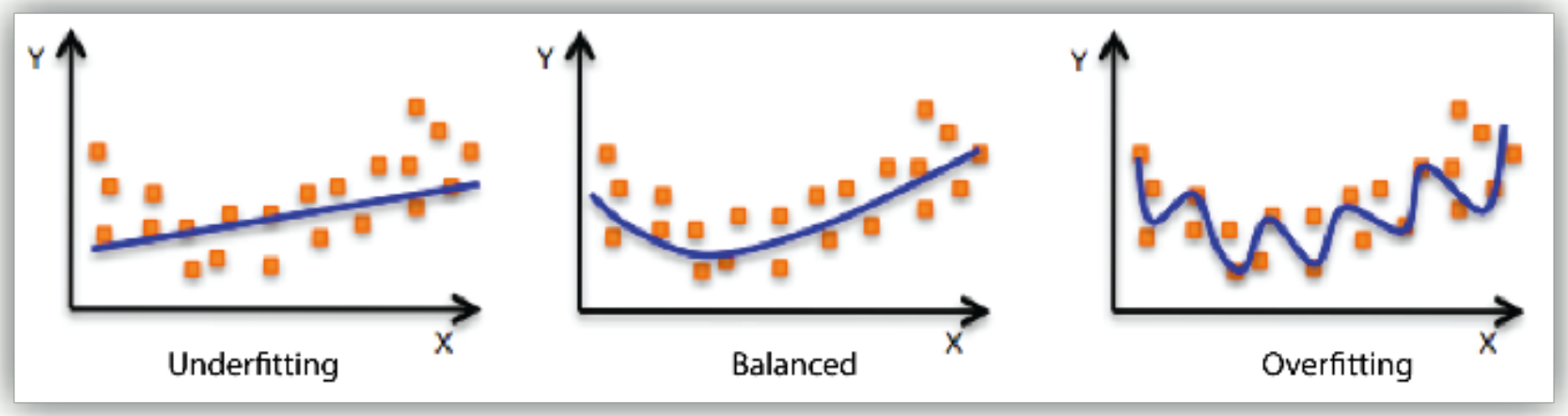
$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

8. Concepts: Underfitting vs. Overfitting

- Understanding how a model fits is important to know the reasons for a poor model accuracy.

- To understand if a model is underfitting or overfitting the training data by analyzing the prediction error on the training and the evaluation data.



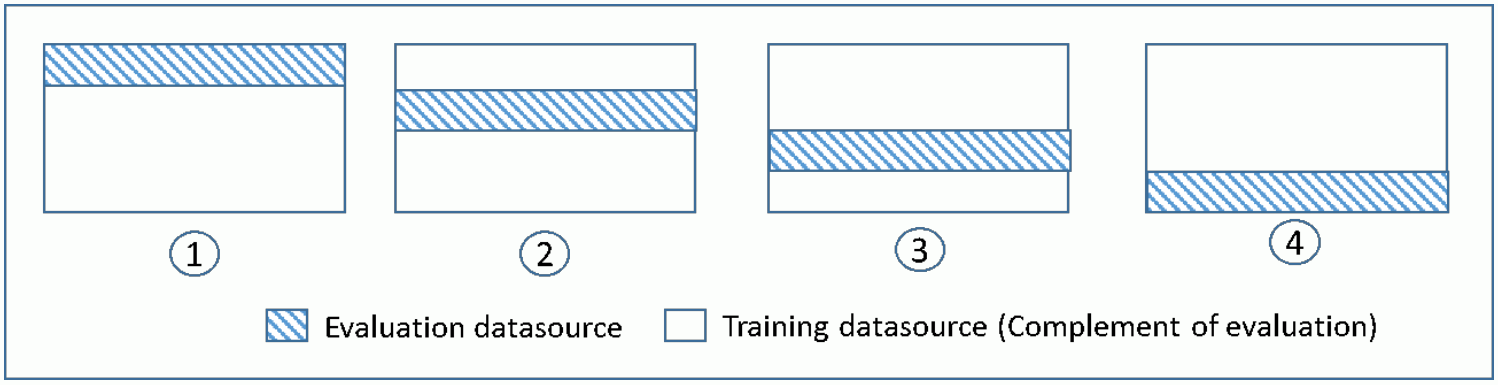
8. Concepts: Underfitting vs. Overfitting

- **Underfitting**: the model performs poorly on the training data, probably because the model is too simple. Performance enhances increasing model flexibility:
  - adding new domain-specific features,
  - modifying the feature processing, or
  - increasing the number of epochs on the existing training data.

- **Overfitting**: the model performs well on the training data, but doesn't do it on the evaluation data; probably because the model is memorizing the data, and it is unable to generalize it to unseen samples. Performance enhances reducing model flexibility:
  - reducing domain-specific features,
  - modifying the feature processing used, or
  - increasing the amount of training data examples.

8. Concepts: Cross-Validation

- Cross-validation is a technique used to evaluate the results of a statistical analysis, and ensure that they are independent of the partition performed for training and testing the data.
- It consists of repeating and calculating the arithmetic mean obtained from the evaluation measures on different partitions.
- Cross-validation is used to detect overfitting, i.e., failing to generalize a certain pattern.



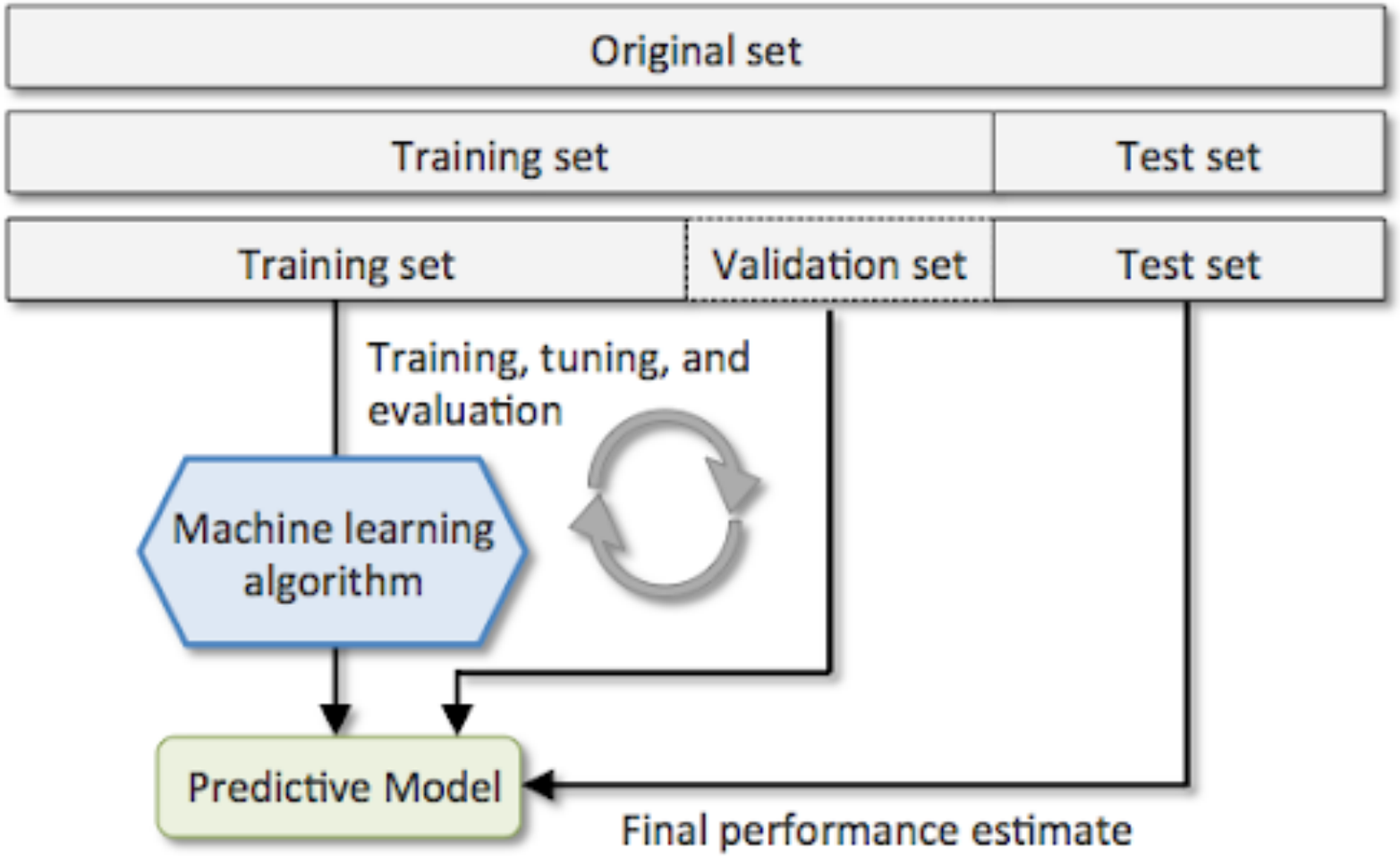
8. Training vs. Validation vs. Testing

- A **training** dataset is a sample dataset used during the learning process to fit the parameters (p.e., neuron weights).
- A **validation** dataset is a sample dataset used to tune the hyperparameters (p.e., the network architecture or the layer depth).
- A **testing** dataset is a sample dataset independent from the training one, but following the same probability distribution.

Procedure:

1) Divide the available data into training, validation and test set.
2) Select an algorithm and training parameters.
3) Train the model using the training set.
4) Evaluate the model using the validation set.
5) Repeat steps 2 through 4 using different algorithms and training parameters.
6) Select the best model and train it using the training and validation sets.
7) Assess this final model using the test set.

8. Training vs. Validation vs. Testing



8. Normalization and Standardization

- Normalization is the technique to bound your data typically between ranges [0,1], but you can also define your range [*a*,*b*] where *a* and *b* are real numbers:

$$x' = a + \frac{(x - \min(x))(b - a)}{\max(x) - \min(x)}$$

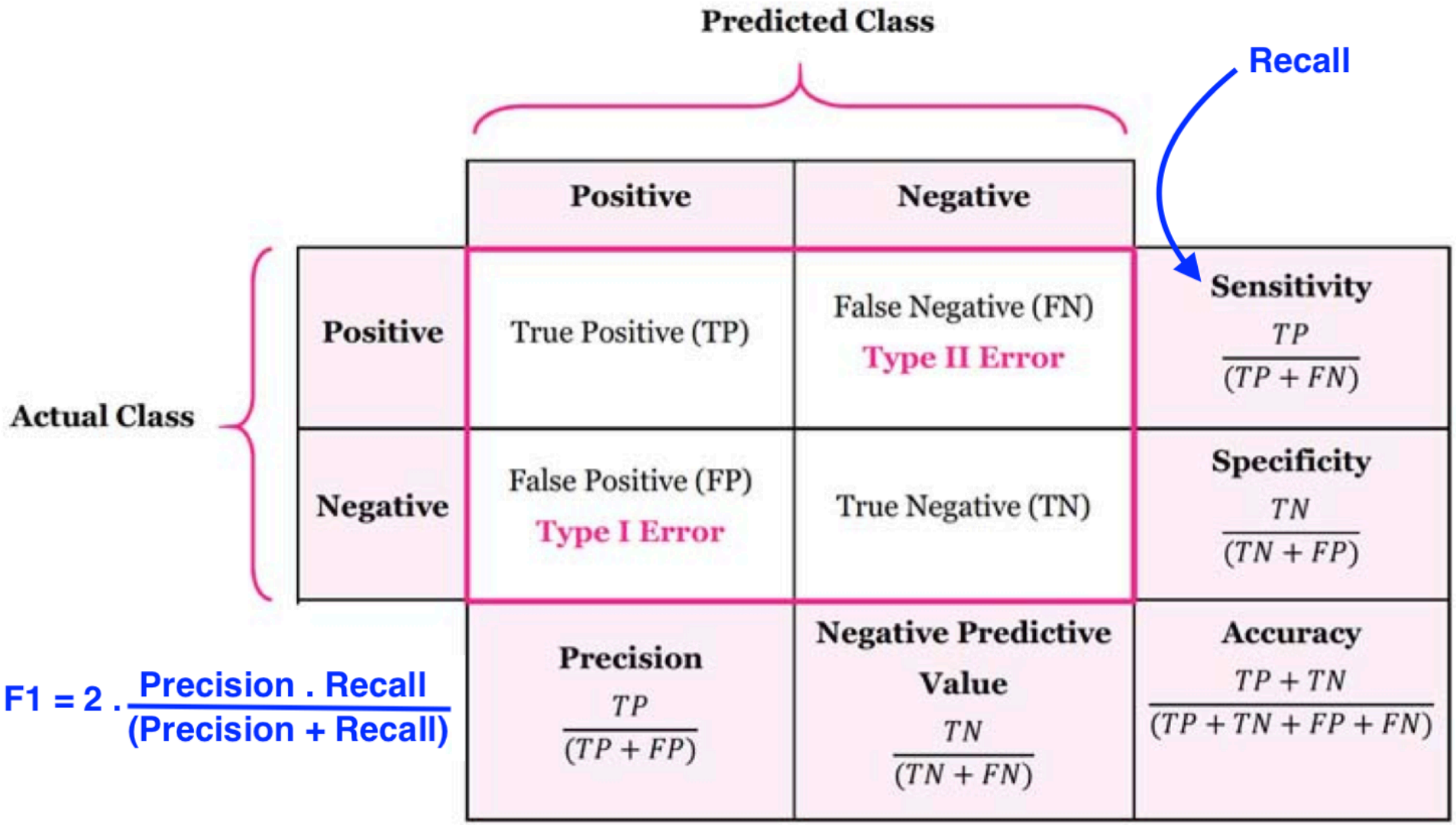
- Standardization transforms data to have a mean of 0 and a variance of 1. First calculate the standard deviation and mean of the feature, then calculate the new values using this formula:

$$x' = \frac{x - \bar{x}}{\sigma}$$

8. Confusion Matrix

- A **confusion** matrix (error matrix or matching matrix) is a specific table to visualize the performance of a certain technique.
- The matrix makes it easy to see if the system is confusing two types of samples, i.e., commonly mislabeling one as another.
- Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class (or vice versa).
- Using a confusion matrix it is easy to see the **performance** of the model concerning: false positives, false negatives, model accuracy, model precision, etc.

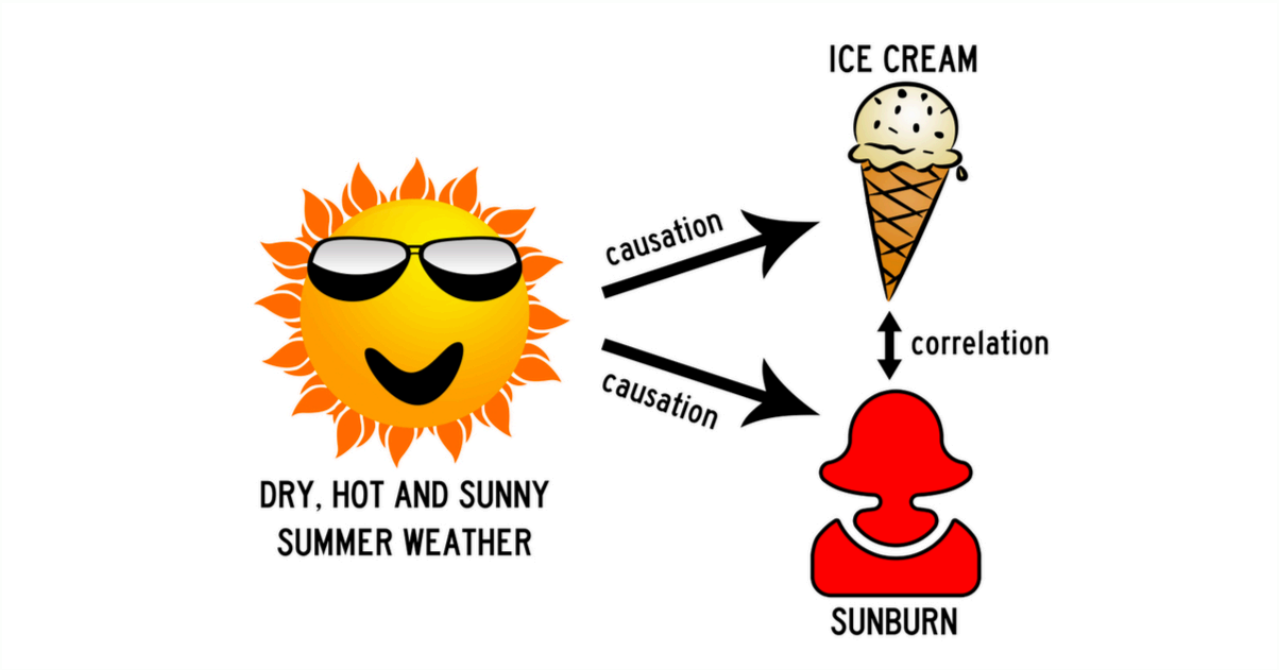
8. Confusion Matrix



8. Correlation vs. Causation

- Correlation checks a relationship between two variables, but having two variables closely related does not necessarily mean that one of them causes the other.

"**Correlation does not imply Causation**"



9. Platforms vs. Libraries

- A **software platform** is an out-of-the-box application that lets you configure a deep net hyper-parameters through an intuitive UI.
- With a platform, you do not need to know anything about coding in order to use the tools. The downside is that you are constrained by the platform's selection of deep nets as well as the configuration options.
- Examples: H2O.ai and GraphLab
- A **software library** is a set of functions and modules that you can call through your own code in order to perform certain tasks.
- Deep net libraries give you a lot of extra flexibility with net selection and hyper-parameter configuration, but you need to code to get full access to its potential.
- Examples: DeepLearning4j, Theano, Caffe, PyTorch, TensorFlow and Keras.

9. Platforms: H2O.ai

- H2O started out as an open-source machine learning platform, with deep nets being a recent addition.

- The platform has sophisticated data managing capabilities as well as an **intuitive model** and management UI, and you also can access the tools through familiar programming environments.

- Training with backpropagation is performed by means of the L-BFGS algorithm. Thankfully, H2O comes with built-in integration tools for platforms like HDFS, Amazon S3, SQL, and NoSQL.

- H2O is provided as a downloadable software package, which you'll need to deploy and manage on your own hardware infrastructure.

- The platform provides an in-memory map-reduce capability, distributed parallel processing, and columnar compression.

9. Platforms: GraphLab (turi.com)

- GraphLab is a software platform that offers deep nets, and host multiple machine learning and graph algorithms.

- In addition to deep nets, the platform has several built-in machine learning algorithms including text analytics, recommenders, classification, regression, clustering, and also Graph Analytic tools.

- GraphLab provides built-in integration for SQL databases, Hadoop, Spark, Amazon S3, Pandas data frames, and many others.

- GraphLab offers three different types of open source built-in storage, all of which are open source: Sarray (columnar representation), Sframe (tabular storage), and Sgraph (the graph model). These tools are designed to handle **massive amounts of data** at interactive speeds.

9. Libraries: DeepLearning4J

- Eclipse Deeplearning4j is a deep learning programming library to be used in Java as a framework with wide support for deep learning algorithms.
- Deeplearning4j includes implementations of the restricted Boltzmann machine, deep belief net, deep autoencoder, stacked denoising autoencoder and recursive neural tensor network, word2vec, doc2vec, and GloVe.
- These algorithms all include distributed parallel versions that integrate with Apache Hadoop and Spark.
- Deeplearning4j is open-source and released under Apache License 2.0.

9. Libraries: DeepLearning4J

- Deeplearning4j is compatible with Clojure and includes a Scala API.
- It is powered by its own open-source numerical computing library (ND4J), and works with both CPUs and GPUs.
- The framework allows to combine shallow neural nets, and recurrent nets can be added to one another to create deep nets of varying types.
- DL4J also has extensive visualization tools and a computation graph.

9. Libraries: Theano

- **Theano is a Python library** that lets you define and **evaluate mathematical expressions with vectors and matrices as rectangular arrays of numbers**.

- Both neural nets and input data can be represented as matrices, and all the standard net operations can be redefined as matrix calculations.

- Keep in mind that if you use Theano, you will have to build a deep net from the down to the top.

- The good news is that Theano allows you to build your implementation atop of a set of vectorized functions providing a highly efficient solution.

9. Libraries: Caffe

- Caffe was originally designed for machine vision tasks, but recent versions of the library provide support for speech and text, reinforcement learning, and recurrent nets for sequence processing.

- Since the library is written in C++ with CUDA, applications can easily switch between a CPU and a GPU as needed.

- You can create a net with many different types of layers, such as a vision layer, a loss layer, an activation layer, and a few others. This flexibility allows you to develop extremely complex deep nets for your application.

- In April 2017, Facebook announced Caffe2, which included new features such as Recurrent Neural Networks. At the end of March 2018, Caffe2 was merged into PyTorch.

9. Libraries: PyTorch

- PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing.

- It has been developed by Facebook's artificial intelligence research group as a free and open-source software released under the Modified BSD license.

- PyTorch provides two high-level features:

  - Tensor computing (like Numpy) with strong acceleration via graphics processing units (GPU).
  - Deep neural networks.

- PyTorch has become recently a trending library for scientific research.

9. Libraries: TensorFlow

- TensorFlow grew out of an earlier Google library called “DistBelief”, which is a proprietary library developed under the Google Brain project.
- The project team's vision was to build a system that simplified the deployment of large-scale machine learning models onto a variety of different hardware setups – from a smart phone, to single servers, or systems consisting of hundreds of machines with thousands of GPUs.
- TensorFlow is currently the most popular Machine Learning Library on GitHub, and is based on the concept of a computational graph.
- In a computational graph, nodes represent either persistent data or a math operation, and edges represent the flow of data between nodes.
- The data that flows through these edges is a multi-dimensional array known as a **tensor**, hence the library's name: “TensorFlow”.

9. Libraries: TensorFlow

- The output from an operation or set of operations is then fed as an input into the next.
- TensorFlow can support any domain where computation can be modeled as a data flow graph.
- TensorFlow and Theano are very similar in many ways, but there are a few key differences. A nice feature is TensorBoard, a visualization tool for network architecture and performance.
- TensorFlow also adopts several useful features from Theano such as auto-differentiation, shared and symbolic variables, and common sub-expression elimination.

9. Metalibrary: Keras

- **Keras was conceived to be a DL interface written in Python**, rather than an end-to-end machine-learning framework.

- Therefore, it is an **open source neural network** library capable of running on top of Theano, Caffe, Tensorflow or DeepLearning4J, among others.

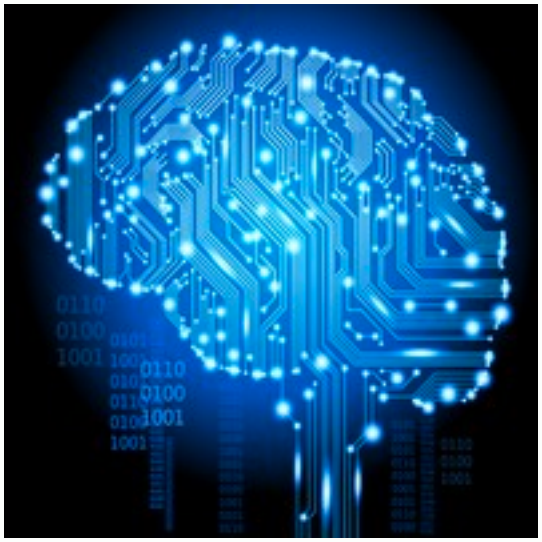
- Designed to enable fast experimentation with deep neural networks, it focuses on being minimal, modular and extensible.

- In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library, and Microsoft did so recently.

E.T.S.E.T.

Deep Learning

Universidade deVigo



Author: Juan Carlos Burguillo Rial

Departament Telematic Engineering

Universidade de Vigo

J.C.Burguillo@det.uvigo.es

http://www.det.uvigo.es/~jrial