Delivering Data Science
In Resources & Energy

Machine Learning II

Day 9

Deep Learning

Dr Debora Correa and Dr Thomas Stemler

Lecturers

Department of Computer Science

and

Department of Mathematical and Statistical sciences, UWA

Thomas

leonardo.portesdossantos@uwa.edu.au

A CORE partnership

Program Timeline

1 day Leading Data Scientists Executive Program	1/2-1 Day*	Enabling your people's data science capability in context of the 15 day program
Data Science Springboard Technical Preparatory	Day 1	Introduction to the program tools & set up
	Day 2	Introduction to the program tools & set up
Introduction to Data Projects	Day 3	Zero to Data Science in a day
Data Analysis	Day 4	Getting to know the program tools - data munging and exploratory data analysis
	Day 5	Simple predictions - regression and statistical model building
	Day 6	Multivariate analysis and model building
Data & Communication Sandbox	Day7	Effective data storytelling - communicating results to non-technical audiences
Data Fusion and Machine Learning	Day 8	Pros and cons of commonly used statistical and machine learning techniques I
Data Fusion and Machine Learning	Day 9	Pros and cons of commonly used statistical and machine learning techniques II
Data Fusion Sandbox	Day 10	Sandbox - Consolidate approaches covered and test on datasets
Special Data Types - Time-series Data	Day 11	The 4th dimension and predictions
Special Data Types - Natural Language Processing and Text Mining	Day 12	Finding needles in wordstacks
Special Data Types - Spatial Data	Day 13	Spatial analytics and predictions
Capstone Project Development & Presentation	Day 14	Pitching Capstone Projects
Capstone Propeller	Day 15	Project Review Day

Plan of the day

AWST	AEST	Agenda	Educator
08:00	**10:00**	**Q&A, Open JupyterHub**	
08:00	10:15	Fundamentals and Multilayer Perceptrons	Debora
09:30	*11:30*	*Morning tea*	
09:45	11:45	Training Artificial Neural Networks	Thomas
11:15	*13:15*	*Lunch*	
12:00	14:00	Convolutional Neural Networks	Debora
13:30	15:30	*Afternoon tea*	
13:45	*15:45*	Recurrent Neural Networks	Thomas
15:15	17:15	Closeout – Reflections, Takeaways, Menti Feedback	Tamryn
15:30	17:30	**Close**	

Aims & Learning Outcomes – Day 9

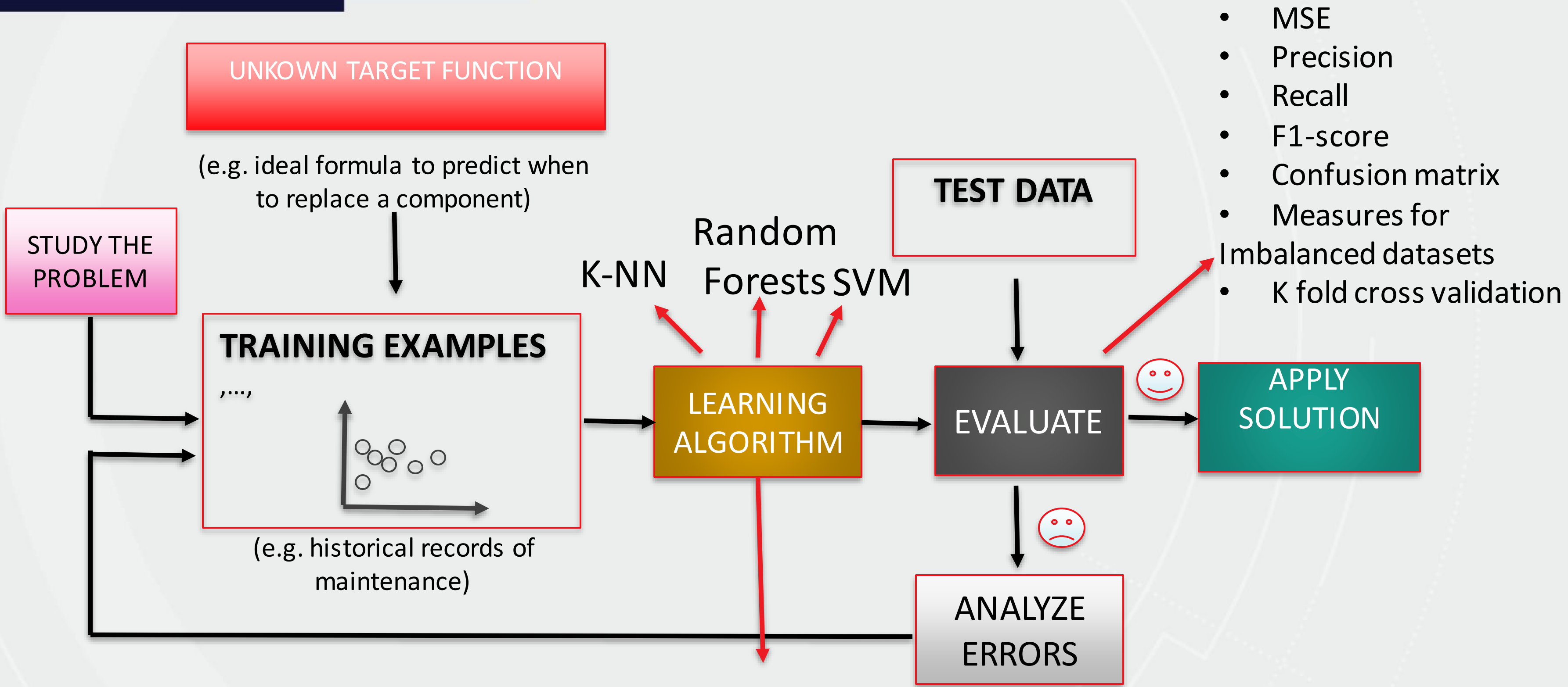
Aims

1. Introduce artificial neural networks and discuss architectures specially designed for images, sequence data and structured data.
2. Introduce the main concepts of deep learning and provide everyday industry examples.
3. Present the practical strategies to train neural networks.

Learning Outcomes

1. Appreciate the powerful framework given by neural networks and deep learning techniques.
2. Recognise what types of data are required by such algorithms.
3. Have the general idea about how to train a neural network.
4. Know how to develop a project for classification/forecast with deep learning.

Day 09 Recap.



Today: ANNs and deep learning.

Picture adapted from: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap

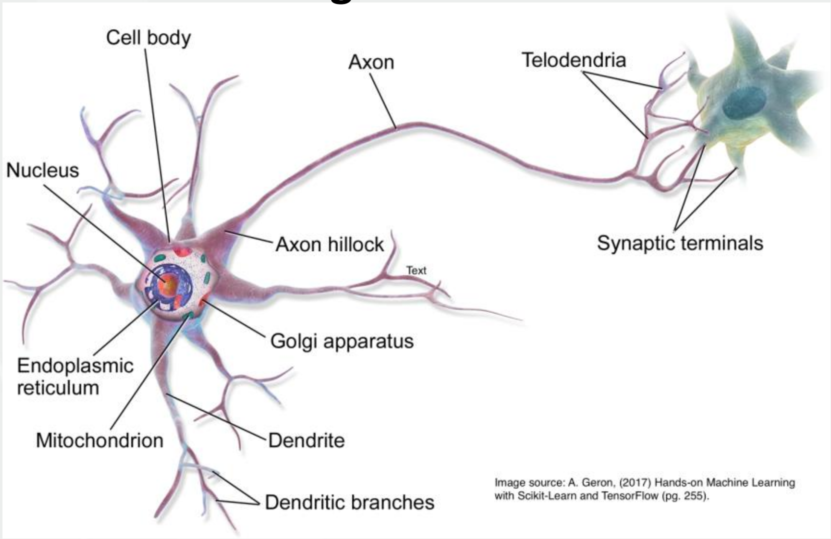
Day 09 Fundamentals and Multilayer Perceptrons

Day 09 ANNs Neuro-inspired computation

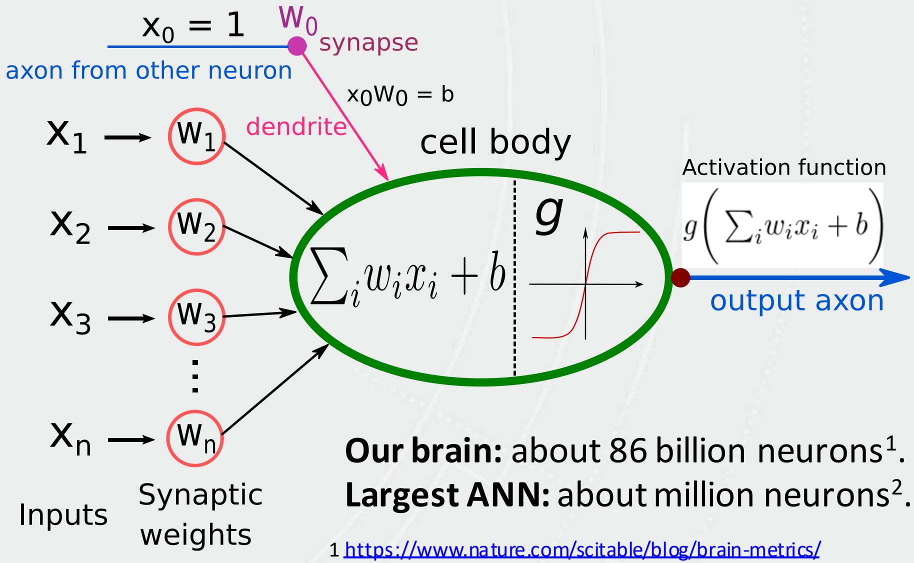
Artificial neural networks (ANNs): family of models with a gross inspiration from the brain.

- Excellent pattern recognition models.
- Excellent on unstructured data.

Biological neuron



Artificial neuron



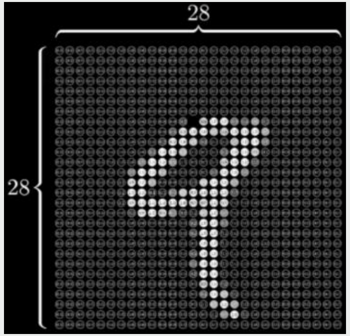
Our brain: about 86 billion neurons[1].
Largest ANN: about million neurons[2].

1 https://www.nature.com/scitable/blog/brain-metrics/
2 https://phys.org/2018-06-ai-method-power-artificial-neural.html

Day 09 ANNs Layers

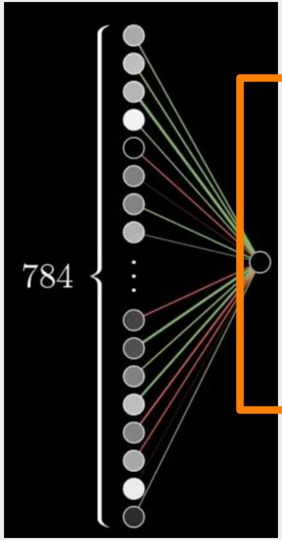
data

instance



28 x 28 = 784

Input layer



"Hidden Layers"

?

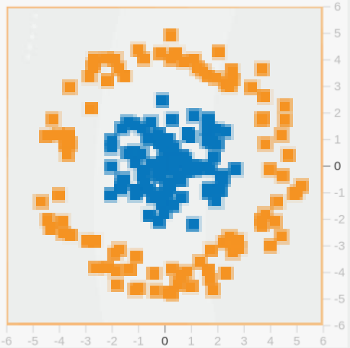
Output

0.86
probability
of being a
9

Some figures from 3Blue1Brown
https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Day 09 ANNs Layers

data

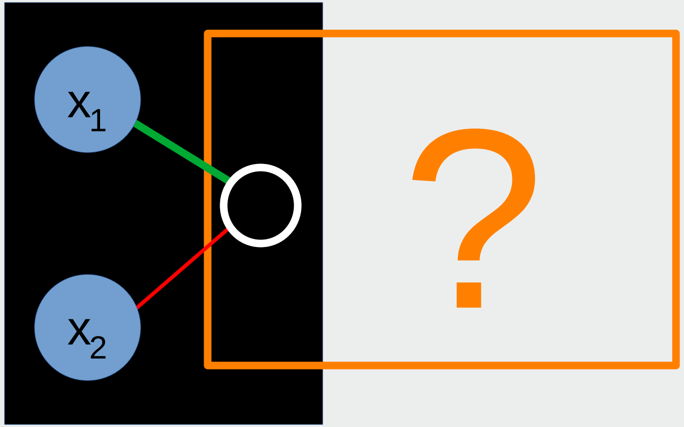


Feature x_2

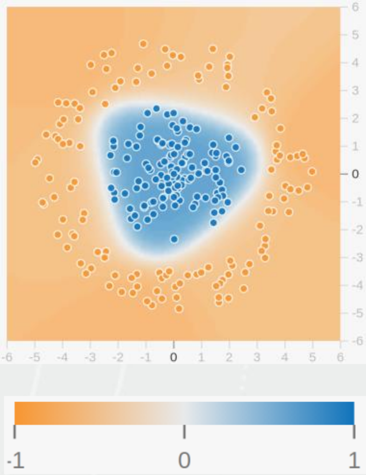
Feature x_1

Input layer

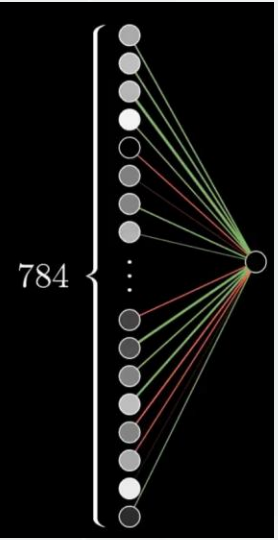
"Hidden Layers"



Output



Day 09 ANNs fundamental computational unity ("neuron")



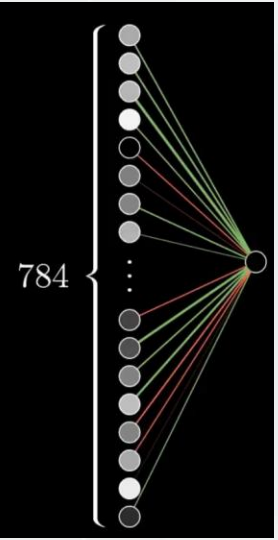
<u>Input for our **unity** ("neuron"):</u>

- Gray scale levels of all pixels x_i, for i = 1, …, 784
- With a given **weight** w_i

$$x^{unity} = w_1 x_1 + w_2 x_2 + \ldots + w_{784} x_{784}$$

Looks like a linear regression!

Day 09 ANNs fundamental computational unity (“neuron”)



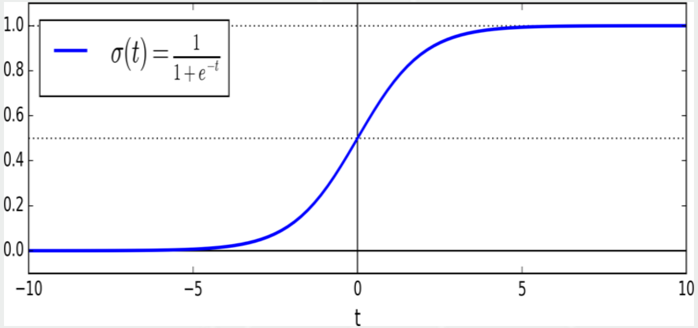
<u>Input for our **unity** (“neuron”):</u>

- Gray scale levels of all pixels x_i, for i = 1, …, 784
- With a given **weight** w_i

<u>State of our unity (it’s output):</u>

- A number (called **activation**)
- $0 < x^{unity} < 1$
- We need a transformation (activation function)

g = sigmoid



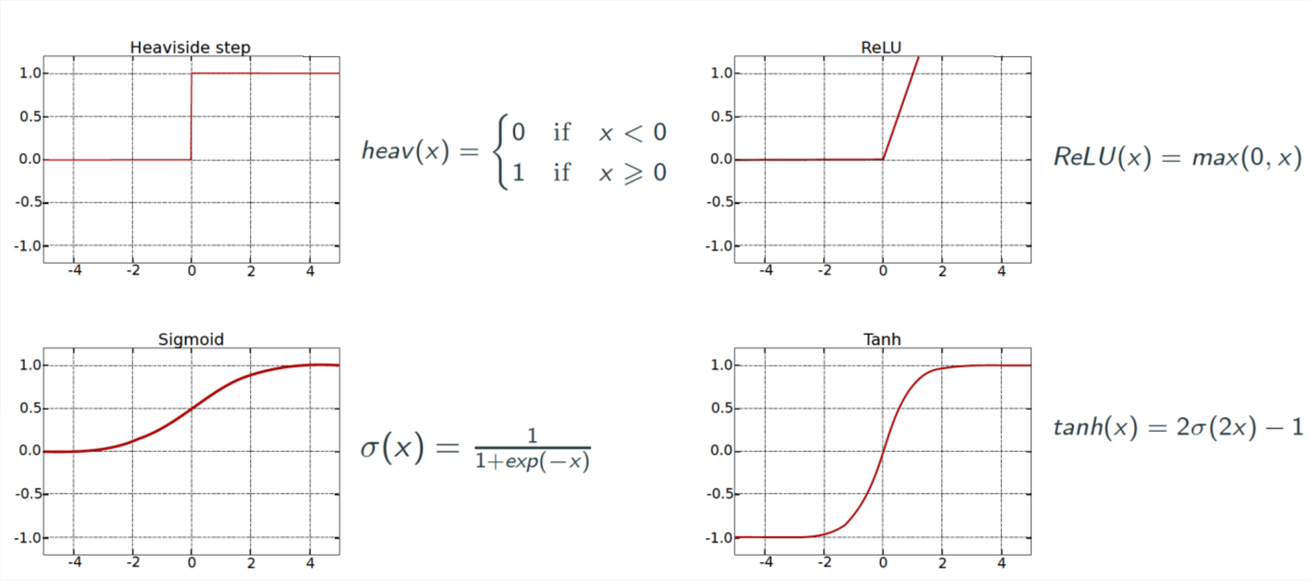
$$x^{unity} = g\left(w_1 x_1 + w_2 x_2 + \dots + w_{784} x_{784} \right)$$

Now it looks like a <u>logistic</u> regression!

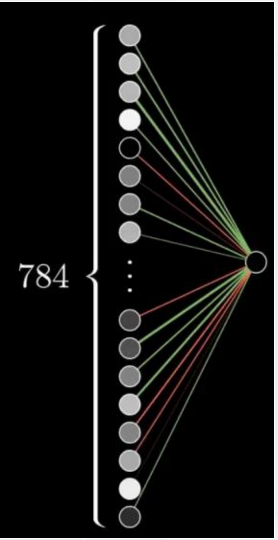
Day 09 Activation functions

Idea: Transformation the output of a neuron by a nonlinear function.

- Normalization of the output (prevent outputs to explode due to the cascading effect).
- This allows neural networks to learn complex patterns and deal with variability in the data.
- Decision making at the output of a neuron.



Day 09 ANNs fundamental computational unity (“neuron”)



Input for our **unity** (“neuron”):

- Gray scale levels of all pixels x_i, for i = 1, …, 784
- With a given **weight** w_i

State of our unity (it’s output):

- A number (called **activation**)
- $0 < x^{unity} < 1$
- We need a transformation (activation function)

Last ingredient:

- Only activate meaningfully if weighted sum is greater than threshold b (bias!)

$$x^{unity} = g\left(w_1 x_1 + w_2 x_2 + \ldots + w_{784} x_{784} + b \right)$$

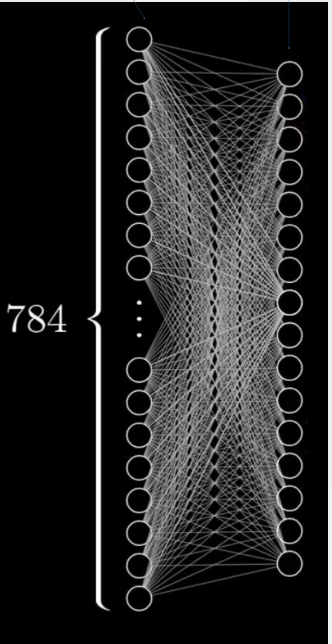
$$\mathbf{x}^{unity} = g\left(\mathbf{w}^t \mathbf{x} + b \right)$$

Bold letters → vectors!

Day 09 ANNs connecting the unities

Layer 1
(input)

Layer 2
(hidden)



$$x^{\text{unity } k} = g\left(w_{k,1} x_1 + w_{k,2} x_{k,2} + \dots + w_{k,n} x_{k,n} + b_k \right)$$

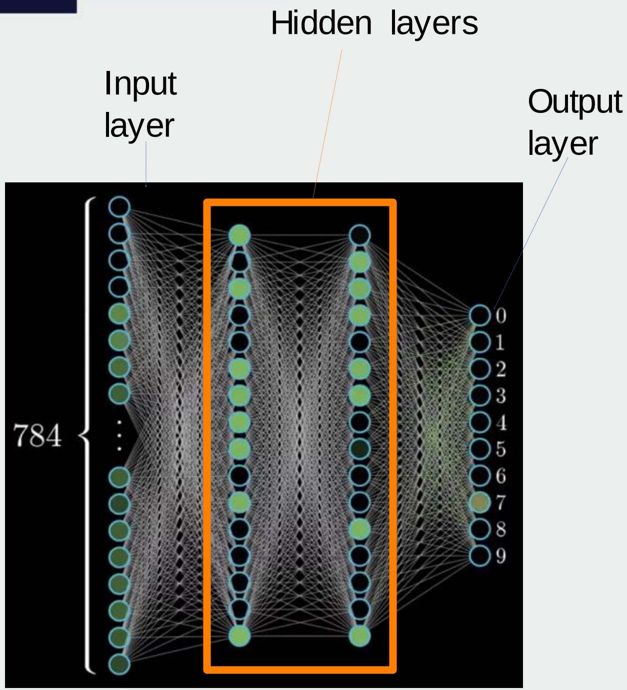
$$g\left(\begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,n} \\ W_{2,1} & W_{2,2} & \dots & W_{2,n} \\ & & & \\ W_{k,1} & W_{k,2} & \dots & W_{k,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \\ b_n \end{bmatrix} \right)$$

N = 784 (inputs)
k (lower case) unity ("neuron") index
K (upper case) number of unities ("neurons")

$$\mathbf{x}^{(\text{layer } 2)} = g\left(\mathbf{W}\, \mathbf{x}^{(\text{layer } 1)} + \mathbf{b} \right)$$

Bold letters → vectors!
Bold upper case → matrices!

Instanc



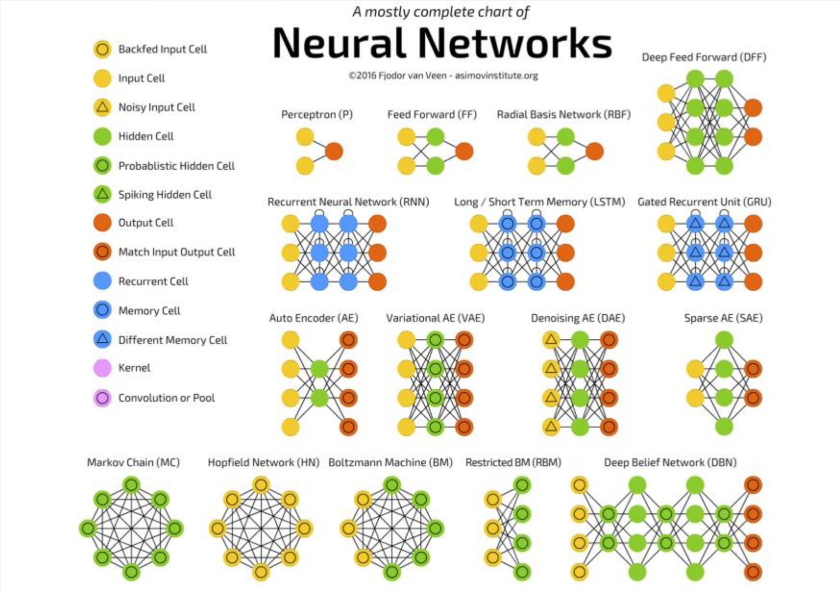
$$x^{(L+1)} = g\left(w^{(L)} x^{(L)} + b^{(L)} \right)$$

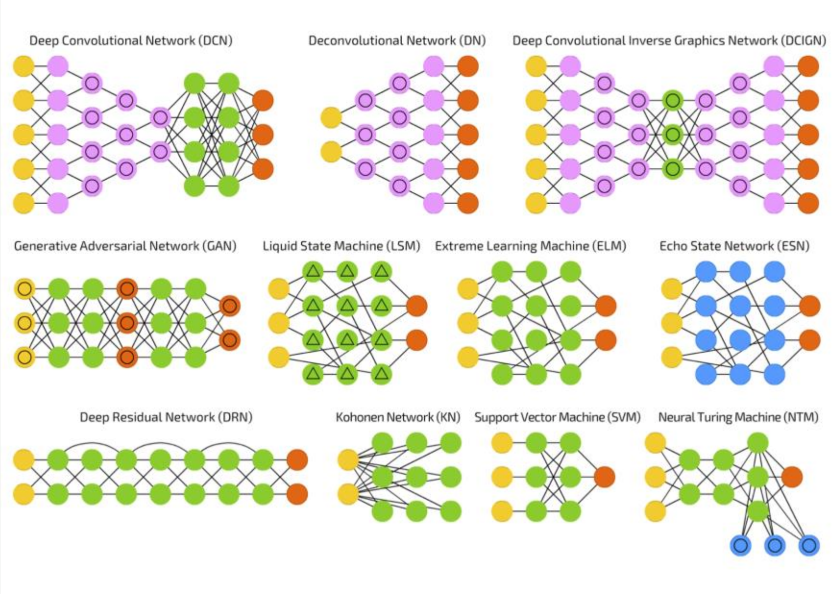
ANN
- One hidden layer

Deep ANN
- Two or more hidden layers

Learning what?
- Weights and bias are the parameters to be **learned**

Day 09 ANNs and their flavours





- ANNs are general function approximators and perform well when learning a complex mapping from the **input** to the **output** space.
- What the network should capture in order to perform such mapping? What should be the input and the output?

Chart available at: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

Day 09 ANNs and applications

Structured data	Multilayer Perceptrons	Input: features, data matrix (samples x features)	Classification, regression.
Image data	Convolutional Neural Networks	Input: images	Classification (objects, face, scene, action, etc.)
Sequence data	Recurrent Neural Networks	Input: time series, audio, text	Classification, speech recognition, prediction.
Different types of data	Autoencoders	Input: original data representation	Dimensionality reduction, pre-training, data reconstruction.

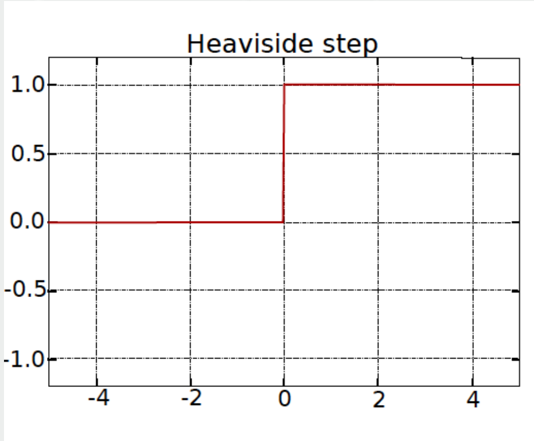
Combination of different ANNs is also very common.

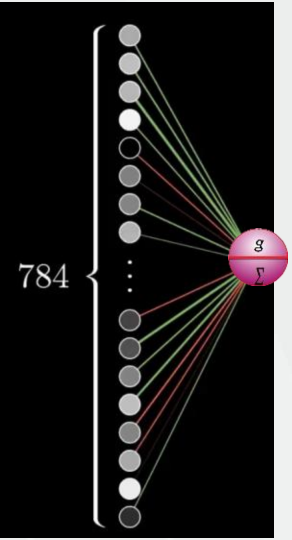
Day 09 Perceptron

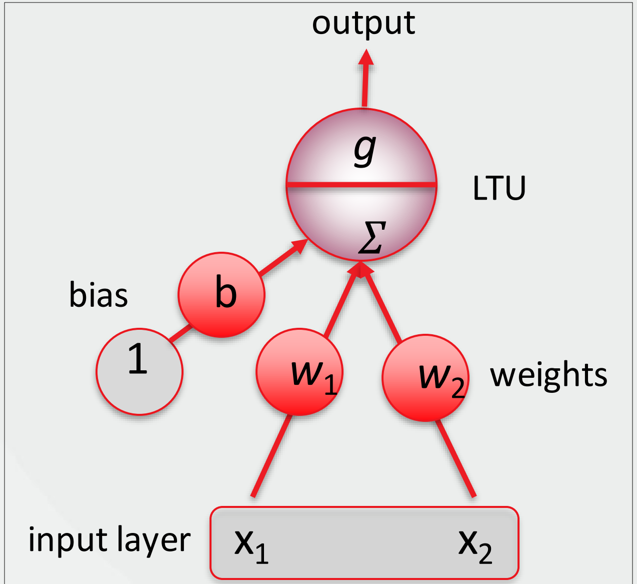
Linear Threshold Units (LTUs) – Proposed by Frank Rosenblatt in 1957

$$h_{\mathbf{w}}(\mathbf{x}) = g(b + w_1 x_1 + w_2 x_2) = g(\mathbf{w}^t.\mathbf{x} + b)$$

$$g(z) = \begin{cases} 0 if z < 0 \\ 1 if z \geq 0 \end{cases}$$



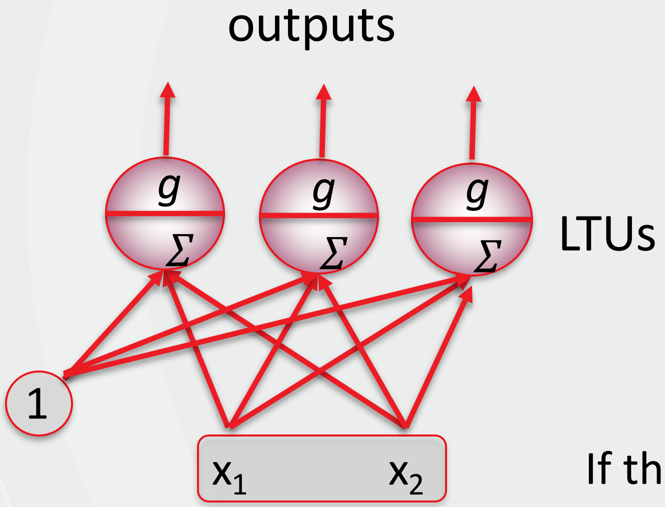




- The weights and bias are the parameters to be **learned**
- Single LTU: simple linear binary classification

Day 09 Perceptron (Cont.)

The perceptron can do multiclass classification



Perceptron learning rule (**weight update**)

$$w_{ij}^{t+1} = w_{ij}^{t} + \eta\left(\hat{y}_j - y_j\right)x_i$$

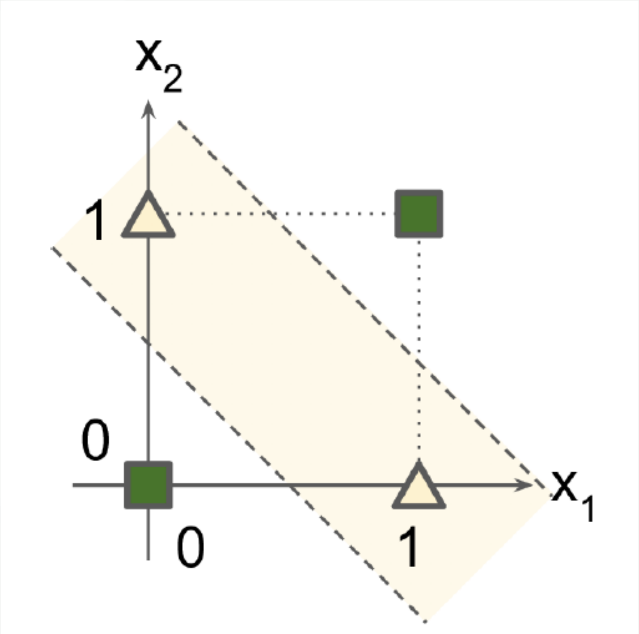
learning rate

If the classes are linearly separable, Perceptron will converge to a solution:
Perceptron convergence theorem

Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 10

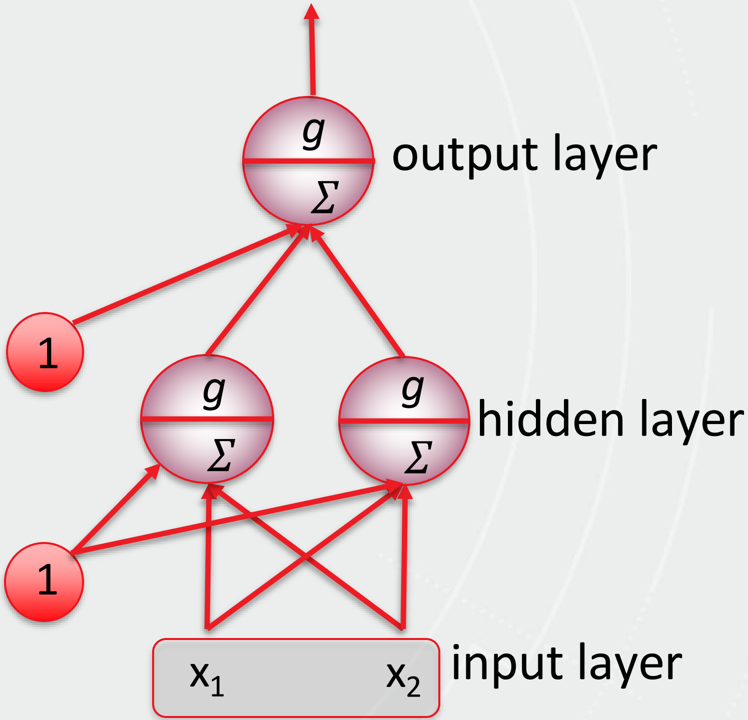
Day 09 From Perceptrons to Multilayer Perceptrons (MLPs)

XOR classification problem



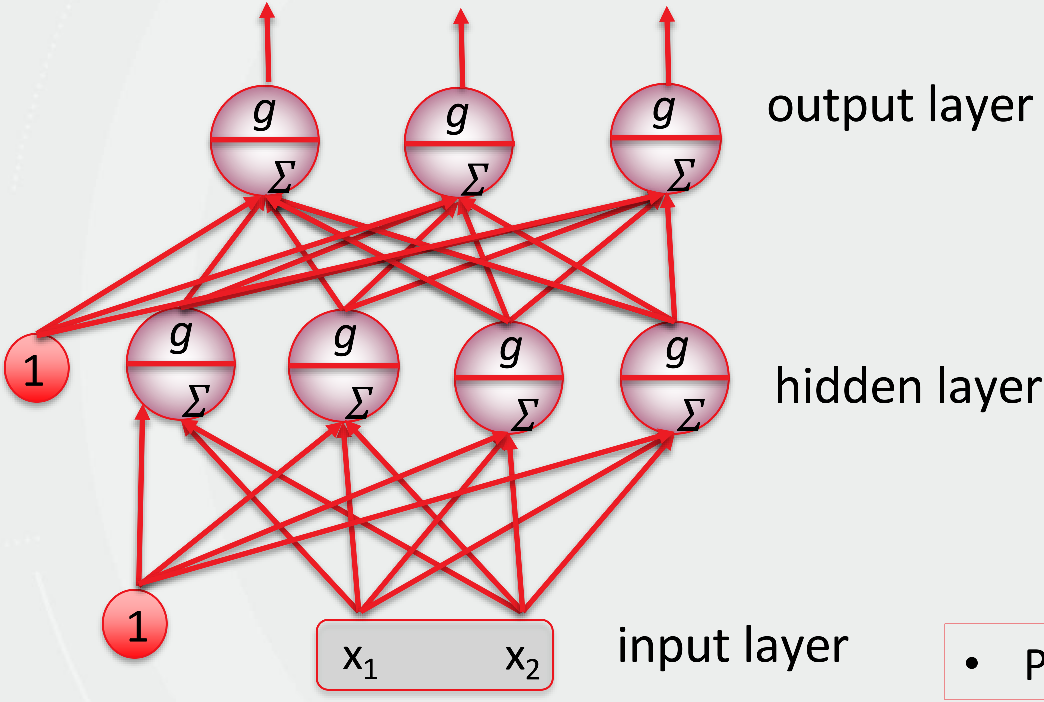
Perceptron can not solve this!

MultiLayer Perceptron



Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 10

Day 09 Multilayer Perceptrons



- Used for the predictions.
- For regression, linear activationfunctions can be used, or no activation function at all

- Compute complex function by cascading simpler functions

- Provides input data to the network

Deep Learning: when an ANN has two or more hidden layers.

Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 10

Day 09 Practice

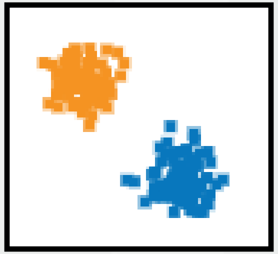
Interactive Google platform for playing with ANNs

playground.tensorflow.org

Let’s create simple feedforward ANNs and see some effects of training in real time. ☺

Day 09 Practice 1/3

DATA



Before running the playground app

1) Visual inspection of the data shows that the classes are linearly separable. To solve that classification problem, what are the minimum number of: (i) Hidden layers? (ii) "neurons"? Now, go to the app and test your answers! Call that minimal model by M1.

2) Using only the information displayed in the app, show evidence that the trained model (M1) is performing well.

3) With model M1, try using only one cell in the input layer (eg, only x_1). Is that model still good? What is the evidence for your answer?

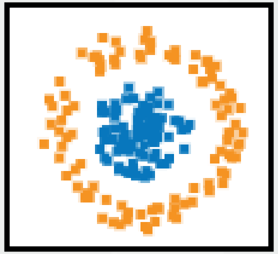
4) Call Md the default model (2 hidden layers: the first one with 4 "neurons" and the second with 2 "neurons"). How the performance of Md compares to M1 in question 2 and 3?

5) If you include noise (eg, level 20), is there any difference in performance between M1 and Md, for both 1 and 2 input cells?

Day 09 Practice 2/3

Start with the default model Md (2 hidden layers: the first one with 4 “neurons” and the second with 2 “neurons”)

DATA



1) Do you need “deep learning” to solve that classification problem? Explore that by changing the number of layers.

2) Train the ANN with only one hidden layer. Can you see any “neuron” in that layer doing “nothing”, or providing redundant information? Maybe you could remove it. If so, what happens with the model performance?

Consider these two models for questions 3 and 4:

(i) Hidden layer1 with 4 neurons and hidden layer 2 with 1 neuron.

(ii) Only one hidden layer with 4 neurons.

3) Which model is more complex (more degrees of freedom)?

4) Train both models. Which one performs better?

5) Could you train a successful model with just one neuron and one hidden layer? Explore the data structure (by visual inspection) and test if some features could help you.

Day 09 Practice 3/3

DATA

Try to solve this classification problem by exploring different:

- Configurations of hidden layers
- Number of neurons
- Combinations of input features

Try to use the information provided by the test loss curve and training loss curve to guide your search (hint: last week, you learned how to use them to infer over/under fitting issues).

What is the simplest configuration you were able to find that solve that problem?

Suggested activities (homework)

The next two slides contain exercises for you to study at home. If possible, try to discuss them with your colleagues.

Day 09 Suggested activity (homework) 1/2

Part 1: No hidden layers

1. Select the third dataset.
   a) Train the network using X_1 and X_2.
2. Select the first dataset.
   a) Do the same: train the network using X_1 and X_2. What happened?
   b) Explore the data structure. It has circular characteristics.
   c) Include the square values of the features, X_1^2 and X_2^2, and train the network again. What happened?
3. Check if these new features can do a good job in the second and fourth datasets.
4. Explore the data structure of the second dataset. What combination of feature can we use?
   a) Try to use only X_1 and X_2 with a nonlinear activation function.
   b) Include a product of the features, X_1X_2 and do the training again.
5. Check if you get any good classification performance on the fourth dataset.

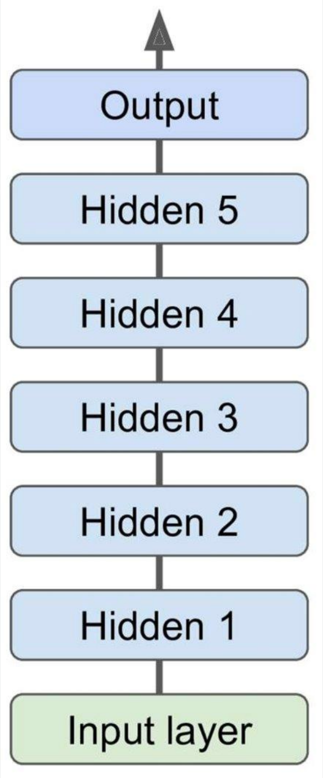
Day 09 Suggested activity (homework) 2/2

Part 2: Adding hidden layers

1. Include one hidden layer with one neuron, use only features X_1 and X_2.
   a) Select the second dataset. Do the training using a linear activation function and with a sigmoid or tanh activation function.
   b) Include one neuron in the hidden layer. Do step a) again.
   c) Keep the nonlinear activation function and include one more hidden layers. Does it improve the performance? Does it take long to train?
   d) Repeat some of these steps for a few times to see the effect of the random initialization of the weights.
2. Select the first dataset.
   a) Do the training using one hidden layer with two neurons (you can use the sigmoid activation function).
   b) Now, include one more neuron in the hidden layer. What happened?
3. Explore different configurations of hidden layers, number of hidden neuros and activation functions on the datasets.

Day 09 Training ANNs

Multi-layer Perceptrons



One output layer

One or more hidden layers

One input layer

*Deep neural network (DNN), when an ANN has two or more hidden layers

Day 09 Gradient-based learning algorithm

Gradient Descent:

- Iterative optimization approach.
- Idea: tweak parameters of the model to minimize a loss function over the training set.
- It computes the error derivatives with respect to each parameter of the model.

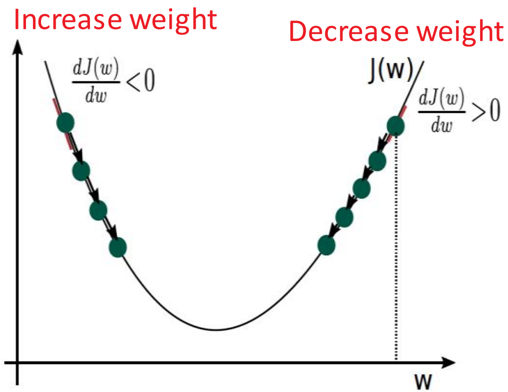
Given a loss function, $J(w,b) = \mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_x} \mathcal{L}^t(\hat{y}^t, y^t)$, the GD tries to find (w, b) that minimizes $J(w, b)$.

Update rule:

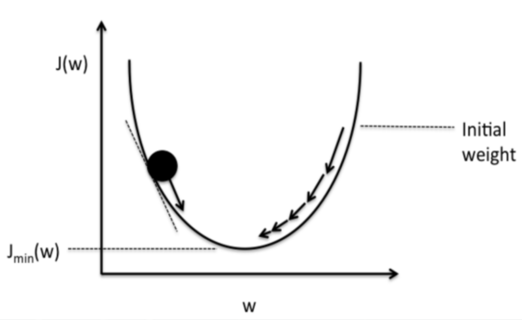
$$w := w - \eta \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \eta \frac{\partial J(w,b)}{\partial b}$$

η is the learning rate.



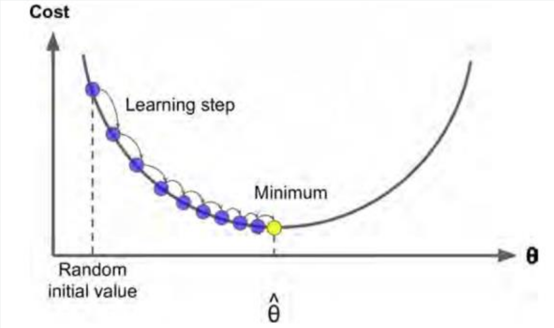
$$w := w - \eta \frac{dJ(w)}{dw}$$

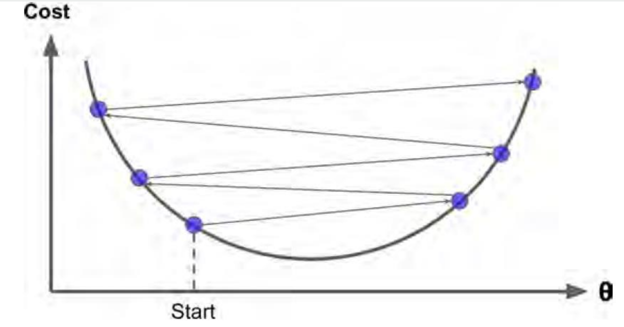


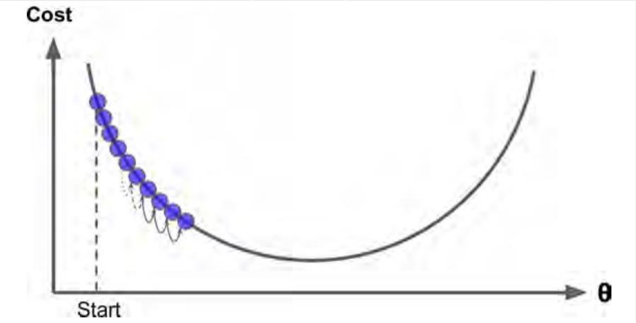
https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

Gradient Descent:

- Iterative optimization approach.
- Idea: tweak parameters of the model to minimize a loss function over the training set.
- It computes the error derivatives with respect to each parameter of the model.







Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 4

Stochastic Gradient Descent

- Gradient Descend (GD) uses the whole training set to compute the errors (gradients) at every step.

- GD is slow for training large data sets.

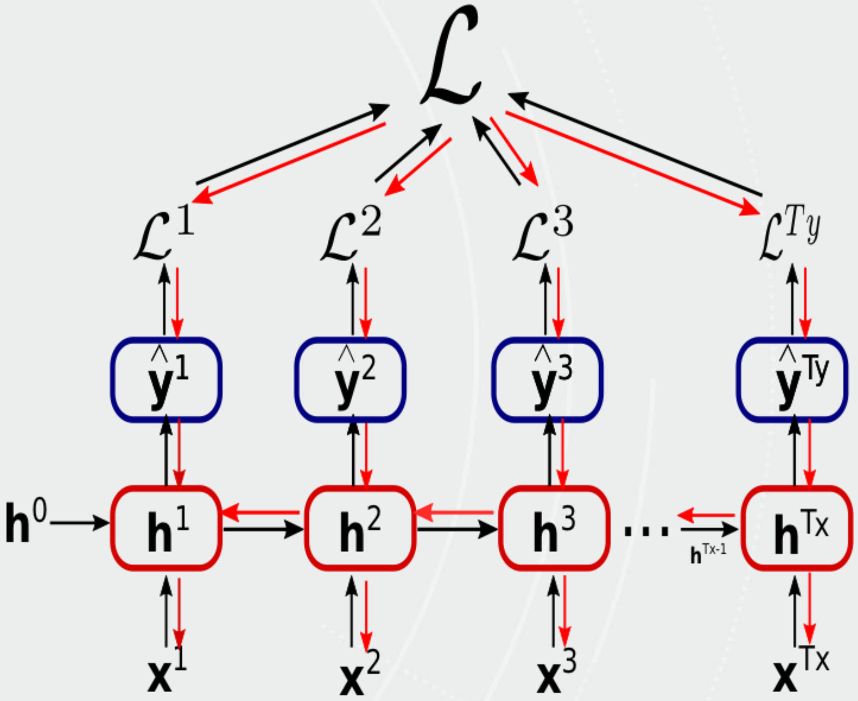
- SGD picks random instances of the training set at every step and computes the errors based only on that instances.
  - Less regular than GD, but it is feasible for large data sets.

Back Propagation Training Algorithm

- For each instance fed to the network, the backpropagation algorithm first makes a prediction **(forward pass)**

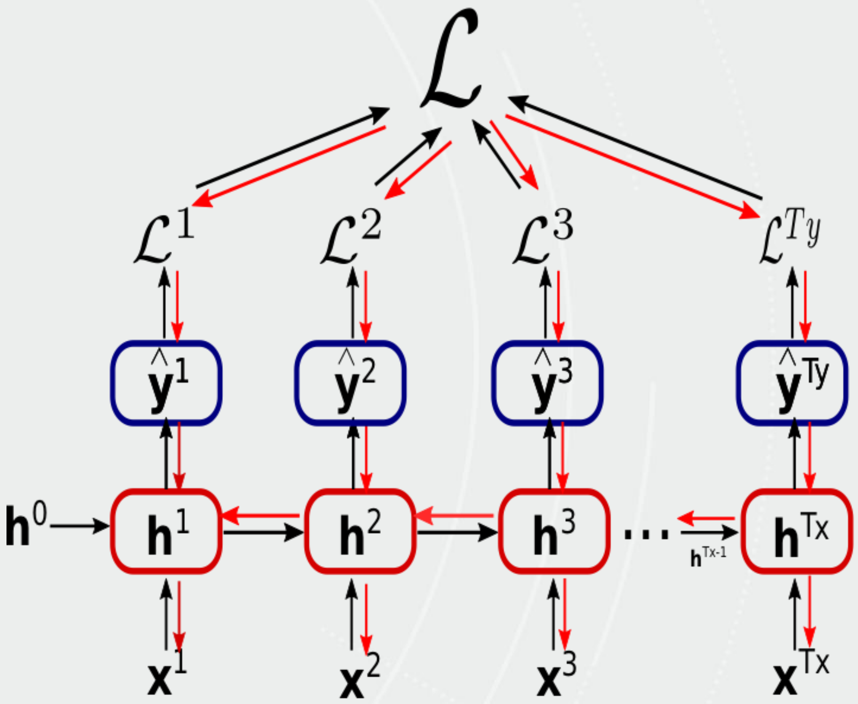
- Then measures the error and goes through each layer in reverse to measure the contribution from each neuron **(backward pass)**

- Finally, the algorithm tunes the neuron weights to reduce the error **(gradient descent step)**



Back Propagation Training Algorithm

- For each instance fed to the network, the backpropagation algorithm first makes a prediction **(forward pass)**
- Then measures the error and goes through each layer in reverse to measure the contribution from each neuron **(backward pass)**
- Finally, the algorithm tunes the neuron weights to reduce the error **(gradient descent step)**

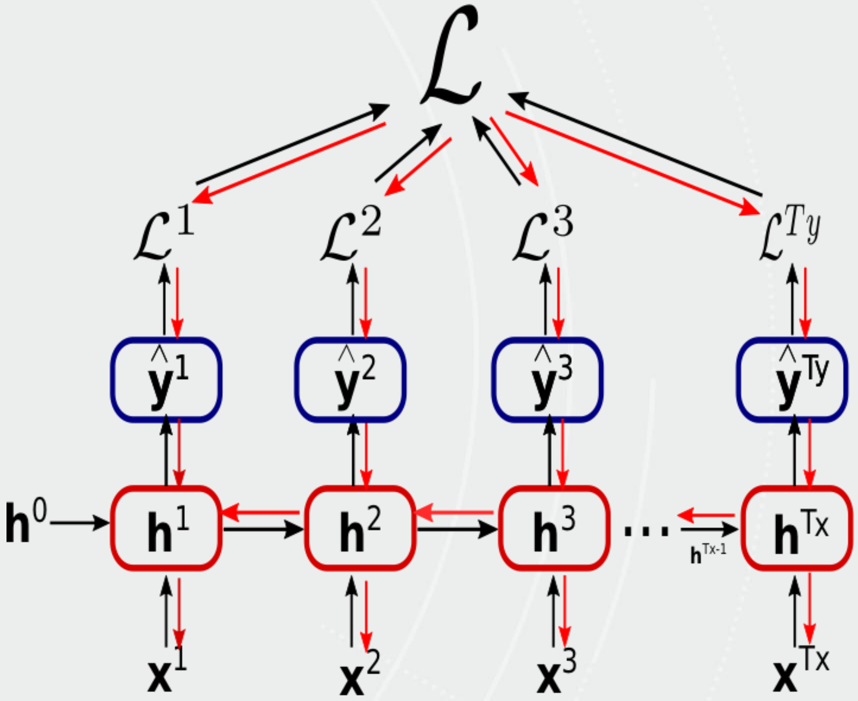


Back Propagation Training Algorithm

- For each instance fed to the network, the backpropagation algorithm first makes a prediction **(forward pass)**

- Then measures the error and goes through each layer in reverse to measure the contribution from each neuron **(backward pass)**

- Finally, the algorithm tunes the neuron weights to reduce the error **(gradient descent step)**



Must see videos on the topic

- What is a neural network? https://youtu.be/aircAruvnKk
- Gradient descent, https://youtu.be/IHZwWFHWa-w
- What is backpropagation really doing? https://youtu.be/Ilg3gGewQ5U
- Backpropagation calculus, https://youtu.be/tIeHLnjs5U8

3Blue1Brown

Must see videos on the topic

- What is a neural network? https://youtu.be/aircAruvnKk
- Gradient descent, https://youtu.be/IHZwWFHWa-w
- What is backpropagation really doing? https://youtu.be/Ilg3gGewQ5U
- Backpropagation calculus, https://youtu.be/tIeHLnjs5U8

MLPs Hyperparameters

Challenges in training MLPs – Many hypermeters to tune:

MLPs Hyperparameters

Challenges in training MLPs – Many hypermeters to tune:

- ***Number of hidden layers***: for many problems, one or two hidden layers are sufficient. For more complex problems, we can gradually increase the number of hidden layers until we start overfitting the training set. Very complex problems (large image classification or speech recognition), require dozens of layers and they need huge amount of training set. (Hierarchical Structure)
- ***Number of neurons per hidden layer***: number of neurons in the input and output layers is determined by your task. A common practice is to size the number of neurons in hidden layers to form a funnel (fewer and fewer neurons at each higher layer). A simpler approach is to pick a model with more layers and neurons than actually need, then use regularization (e.g. dropout) to prevent overfitting.
- ***Activation functions***: ReLU in hidden layers is a bit faster to compute than other activation functions, and Gradient Descent does not get stuck as much on local minima. For output layer: Softmax is generally a good choice for classification, for regression you can use no activation function at all.
- ***Weight initialization logic***

MLPs Hyperparameters

Challenges in training MLPs – Many hypermeters to tune:

- ***Number of hidden layers***: for many problems, one or two hidden layers are sufficient. For more complex problems, we can gradually increase the number of hidden layers until we start overfitting the training set. Very complex problems (large image classification or speech recognition), require dozens of layers and they need huge amount of training set. (Hierarchical Structure)
- ***Number of neurons per hidden layer***: number of neurons in the input and output layers is determined by your task. A common practice is to size the number of neurons in hidden layers to form a funnel (fewer and fewer neurons at each higher layer). A simpler approach is to pick a model with more layers and neurons than actually need, then use regularization (e.g. dropout) to prevent overfitting.
- ***Activation functions***: ReLU in hidden layers is a bit faster to compute than other activation functions, and Gradient Descent does not get stuck as much on local minima. For output layer: Softmax is generally a good choice for classification, for regression you can use no activation function at all.
- ***Weight initialization logic***

MLPs Hyperparameters

Challenges in training MLPs – Many hypermeters to tune:

- ***Number of hidden layers***: for many problems, one or two hidden layers are sufficient. For more complex problems, we can gradually increase the number of hidden layers until we start overfitting the training set. Very complex problems (large image classification or speech recognition), require dozens of layers and they need huge amount of training set. (Hierarchical Structure)
- ***Number of neurons per hidden layer***: number of neurons in the input and output layers is determined by your task. A common practice is to size the number of neurons in hidden layers to form a funnel (fewer and fewer neurons at each higher layer). A simpler approach is to pick a model with more layers and neurons than actually need, then use regularization (e.g. dropout) to prevent overfitting.
- ***Activation functions***: ReLU in hidden layers is a bit faster to compute than other activation functions, and Gradient Descent does not get stuck as much on local minima. For output layer: Softmax is generally a good choice for classification, for regression you can use no activation function at all.
- ***Weight initialization logic***

MLPs Hyperparameters

Challenges in training MLPs – Many hypermeters to tune:

- ***Number of hidden layers***: for many problems, one or two hidden layers are sufficient. For more complex problems, we can gradually increase the number of hidden layers until we start overfitting the training set. Very complex problems (large image classification or speech recognition), require dozens of layers and they need huge amount of training set. (Hierarchical Structure)
- ***Number of neurons per hidden layer***: number of neurons in the input and output layers is determined by your task. A common practice is to size the number of neurons in hidden layers to form a funnel (fewer and fewer neurons at each higher layer). A simpler approach is to pick a model with more layers and neurons than actually need, then use regularization (e.g. dropout) to prevent overfitting.
- ***Activation functions***: ReLU in hidden layers is a bit faster to compute than other activation functions, and Gradient Descent does not get stuck as much on local minima. For output layer: Softmax is generally a good choice for classification, for regression you can use no activation function at all.
- ***Weight initialization logic***

MLPs Hyperparameters

Challenges in training MLPs – Many hypermeters to tune:

- ***Number of hidden layers***: for many problems, one or two hidden layers are sufficient. For more complex problems, we can gradually increase the number of hidden layers until we start overfitting the training set. Very complex problems (large image classification or speech recognition), require dozens of layers and they need huge amount of training set. (Hierarchical Structure)
- ***Number of neurons per hidden layer***: number of neurons in the input and output layers is determined by your task. A common practice is to size the number of neurons in hidden layers to form a funnel (fewer and fewer neurons at each higher layer). A simpler approach is to pick a model with more layers and neurons than actually need, then use regularization (e.g. dropout) to prevent overfitting.
- ***Activation functions***: ReLU in hidden layers is a bit faster to compute than other activation functions, and Gradient Descent does not get stuck as much on local minima. For output layer: Softmax is generally a good choice for classification, for regression you can use no activation function at all.
- ***Weight initialization logic***

* In theory, we could use cross-validation but it costs a lot of time and computations

Tools and strategies for training MLPs/DNN

Problems faced through training a DNN:

Tools and strategies for training MLPs/DNN

Problems faced through training a DNN:

- **Vanishing gradients problem (or the related, exploding gradients problem) that makes lower layers very hard to train**: *backpropagation works by going from the output layer to the input layer. It gets smaller and smaller as the algorithm progresses down, as a result the gradient descent update leaves the lower layer connection weights unchanged and training never converges to a good solution.*

- Training DNNs would be very slow with large networks

- A DNN with many parameters would severely risk overfitting the training set

Tools and strategies for training MLPs/DNN

Problems faced through training a DNN:

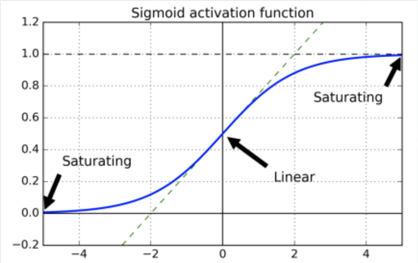
- **Vanishing gradients problem (or the related, exploding gradients problem) that makes lower layers very hard to train**: *backpropagation works by going from the output layer to the input layer. It gets smaller and smaller as the algorithm progresses down, as a result the gradient descent update leaves the lower layer connection weights unchanged and training never converges to a good solution.*

Popular strategies to solve vanishing/exploding gradients problem

- Training DNNs would be very slow with large networks

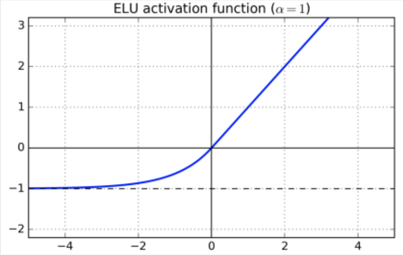
- A DNN with many parameters would severely risk overfitting the training set

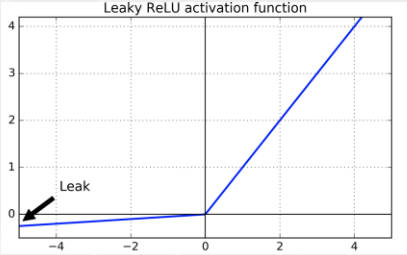
Tools and strategies for training MLPs/DNN



Strategies to solve vanishing/exploding gradients problems:

- **Input normalization (zero mean and unit std)**
- **Xavier/Glorot weight initialization**
- **Nonsaturating activation functions:**
  Exponential Linear Unit (ELU) Leaky ReLU





- **Batch Normalization:**
  Zero-centre and normalization of the inputs in each layer before activation function is computed
- **Gradient clipping:**
  Used for exploding gradients
  Rescale gradients to prevent they exceed a maximum threshold

Tools and strategies for training MLPs/DNN

Problems faced through training a DNN:

- **Vanishing gradients problem (or the related, exploding gradients problem) that makes lower layers very hard to train**: *backpropagation works by going from the output layer to the input layer. The gets smaller and smaller as the algorithm progresses down, as a result the gradient descent update leaves the lower layer connection weights unchanged and training never converges to a good solution.*

Popular strategies to solve vanishing/exploding gradients problem

- **Training DNNs would be very slow with large networks**

- **A DNN with many parameters would severely risk overfitting the training set**

Tools and strategies for training MLPs/DNN

Problems faced through training a DNN:

- **Vanishing gradients problem (or the related, exploding gradients problem) that makes lower layers very hard to train**: *backpropagation works by going from the output layer to the input layer. The gets smaller and smaller as the algorithm progresses down, as a result the gradient descent update leaves the lower layer connection weights unchanged and training never converges to a good solution.*

Popular strategies to solve vanishing/exploding gradients problem

- **Training DNNs would be very slow with large networks**

Faster optimizers that can speed up the training

- **A DNN with many parameters would severely risk overfitting the training set**

Tools and strategies for training MLPs/DNN

Faster optimizers (variations of the Stochastic Gradient Descent):

- Momentum optimization
- Nesterov Accelerated Gradient
- AdaGrad
- RMSProp
- Adam Optimization
- Learning Rate Scheduling

Tools and strategies for training MLPs/DNN

Problems faced through training a DNN:

- **Vanishing gradients problem (or the related, exploding gradients problem) that makes lower layers very hard to train**: *backpropagation works by going from the output layer to the input layer. The gets smaller and smaller as the algorithm progresses down, as a result the gradient descent update leaves the lower layer connection weights unchanged and training never converges to a good solution.*

Popular strategies to solve vanishing/exploding gradients problem

- **Training DNNs would be very slow with large networks**

Faster optimizers that can speed up the training

- **A DNN with many parameters would severely risk overfitting the training set**

Tools and strategies for training MLPs/DNN

Problems faced through training a DNN:

- **Vanishing gradients problem (or the related, exploding gradients problem) that makes lower layers very hard to train**: *backpropagation works by going from the output layer to the input layer. The gets smaller and smaller as the algorithm progresses down, as a result the gradient descent update leaves the lower layer connection weights unchanged and training never converges to a good solution.*

Popular strategies to solve vanishing/exploding gradients problem

- **Training DNNs would be very slow with large networks**

Faster optimizers that can speed up the training

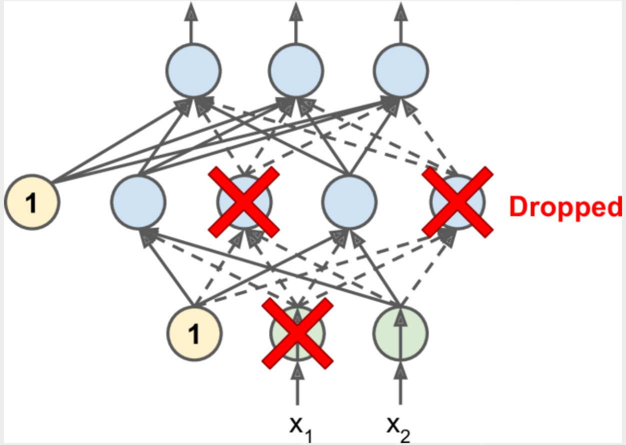
- **A DNN with many parameters would severely risk overfitting the training set**

Popular regularization techniques to avoid overfitting

Tools and strategies for training MLPs/DNN

Avoid overfitting through regularization:

- Early Stopping
- L1 and L2 Regularization
- Dropout
- Max-norm Regularization
- Data Augmentation



Tools and strategies for training MLPs/DNN

Avoid overfitting through regularization:

- Early Stopping
- L1 and L2 Regularization
- Dropout
- Max-norm Regularization
- Data Augmentation

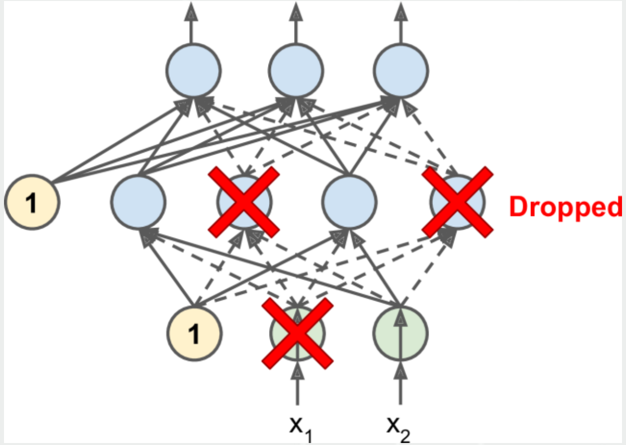


Table 11-2. Default DNN configuration

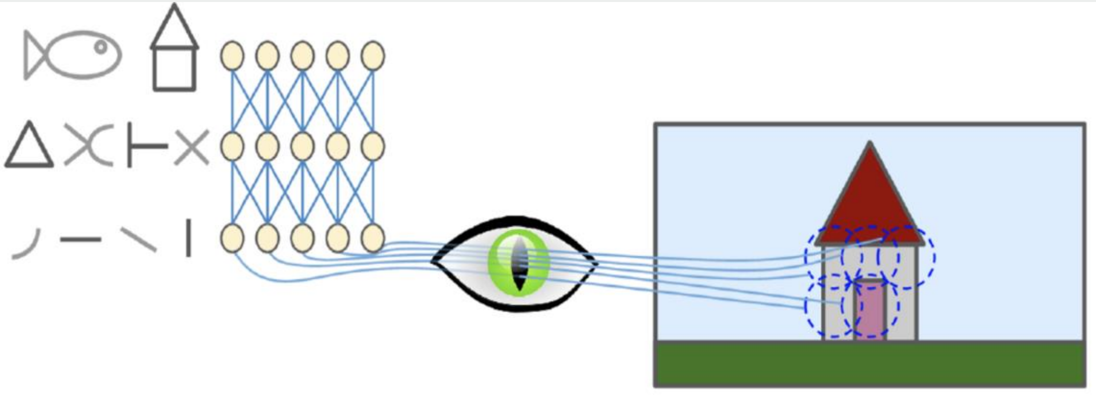
Initialization	He initialization
Activation function	ELU
Normalization	Batch Normalization
Regularization	Dropout
Optimizer	Adam
Learning rate schedule	None

Day 09 Convolutional Neural Networks (CNNs)

Day 09 CNN Motivation

Insights on the structure of the visual cortex[1,2,3]

- Neurons respond to a *small local receptive field*: they react to a visual *stimuli* placed in a small region of the visual field.
- Their receptive fields can overlap, and the whole visual field arises from the combination of individual local fields.
- Idea that higher-level neurons use the neighboring of lower-level neurons, forming more complex patterns by combining lower-level patterns.

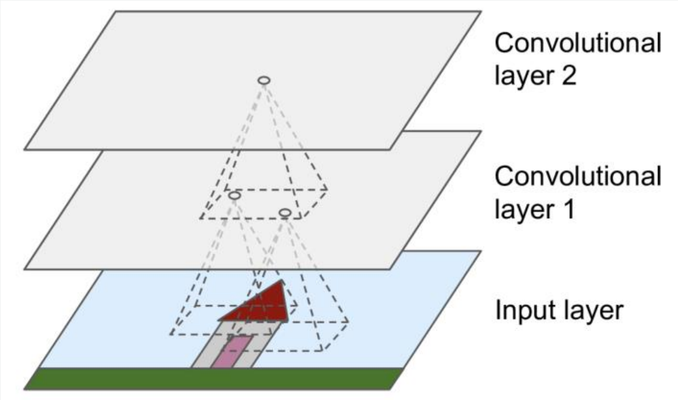


1 "Single Unit Activity in Striate Cortex of Unrestrained Cats," D. Hubel and T. Wiesel (1958).
2 "Receptive Fields of Single Neurones in the Cat's Striate Cortex," D. Hubel and T. Wiesel (1959).
3 "Receptive Fields and Functional Architecture of Monkey Striate Cortex," D. Hubel and T. Wiesel (1968).

Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 13.

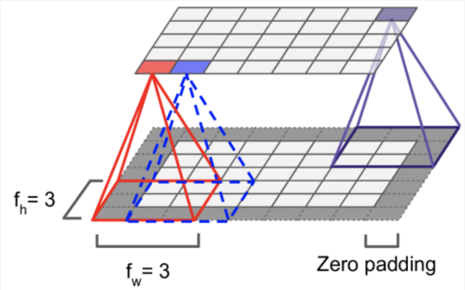
Day 09 Convolutional Layers

Each neuron layer is now 2D



- Neurons in the first convolutional layer are connected only to a small rectangle of pixels representing their receptive fields.
- Each neuron in subsequent layers is connected to neurons located in their receptive field.

f$_h$ and **f**$_w$ define the height and width of the receptive field.



Day 09 Convolution

Important operation in image processing.

- A (small) kernel matrix is applied to the input image in order to get a transformed image according to a specific task.

Input matrix

95	101	98	102	97
96	100	99	101	114
99	98	99	99	87
94	98	100	102	94
89	91	95	98	89
100	99	87	89	91
104	101	88	93	96

Kernel matrix

0	-1	0
-1	5	-1
0	-1	0

Output matrix

	106			

Multiplication of each element of the kernel with the corresponding element of the image matrix.

(95 * 0) + (101 * -1) + (98 * 0)
+ (96 * -1) + (100 * 5) + (99 * -1)
+ (99 * 0) + (98 * -1) + (99 * 0) = 106

Day 09 Convolution (Cont.)

- E.g., edge detection (horizontal, vertical), sharpen, blurring, etc.

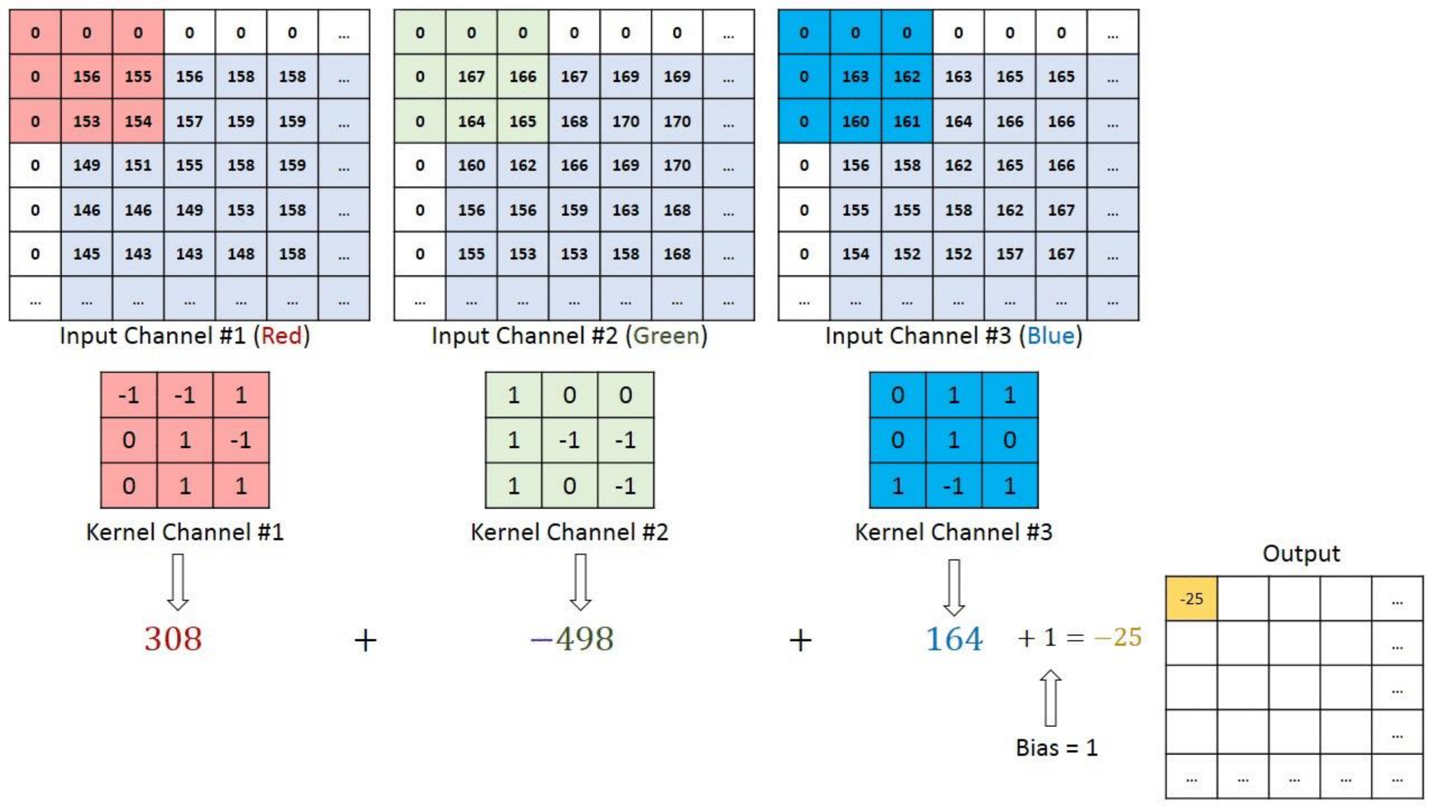
0	0	0
-1	2	-1
0	0	0

0	-1	0
0	2	0
0	-1	0

0	-1	0
-1	5	-1
0	-1	0

1	2	1
0	0	0
-1	-2	-1

Day 09 RGB Images



Source: http://machinelearningguru.com

Day 09 Stacking Multiple Feature Maps

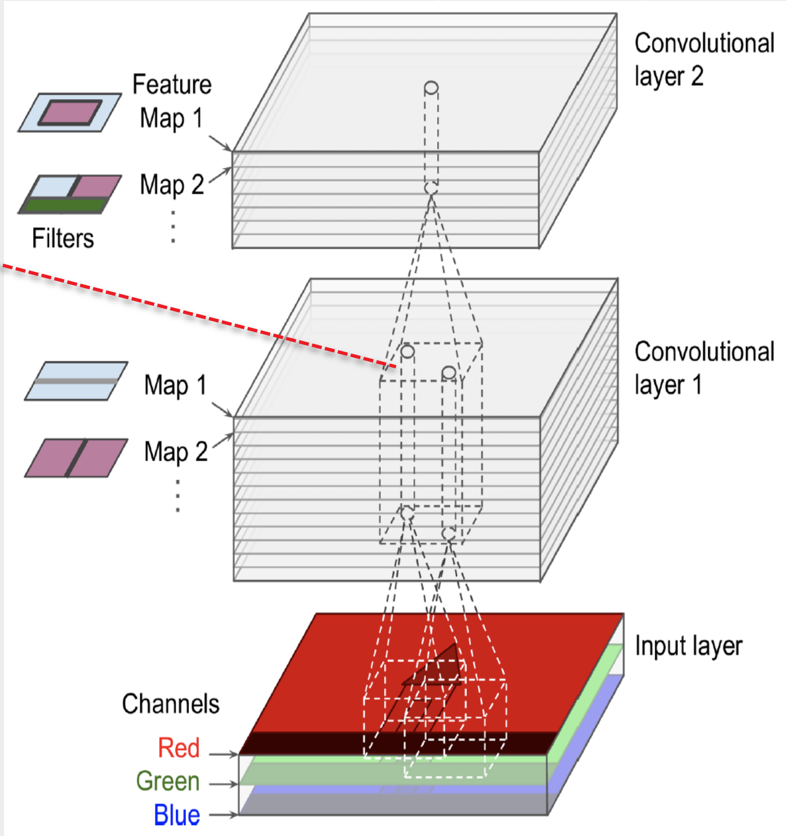
Kernel matrix

0	-1	0
-1	5	-1
0	-1	0

The **weights** of the kernel and the **bias** of each feature map are trained with the back-propagation algorithm.

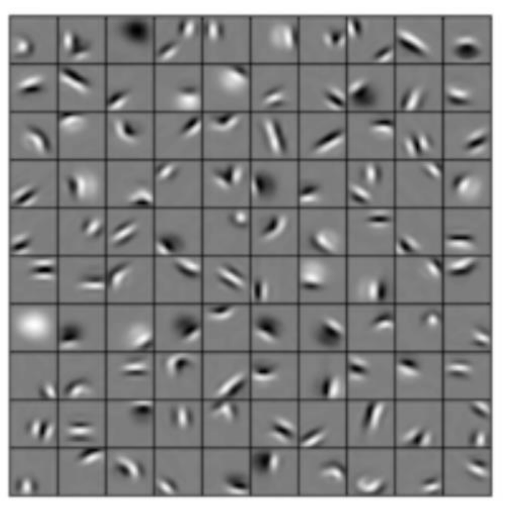
A feature map share the same weights and bias.

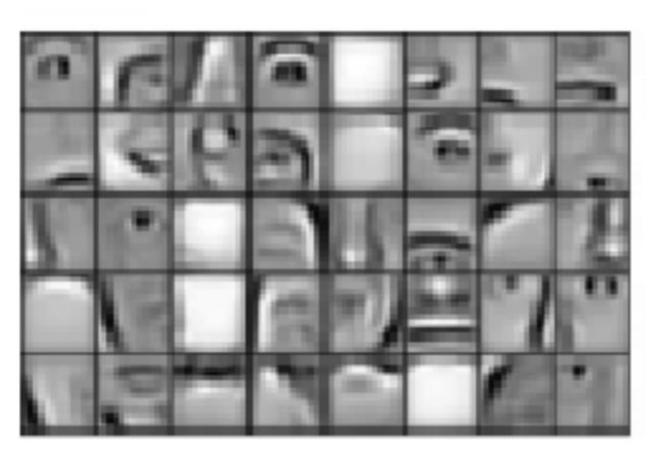
An activation function is applied after each convolutional layer.

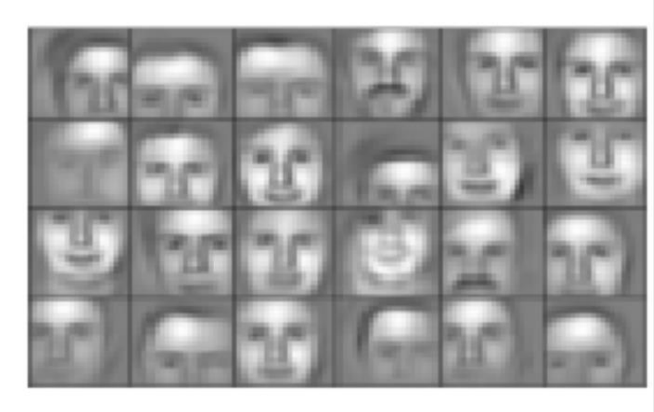


Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 13

CNN - Intuition



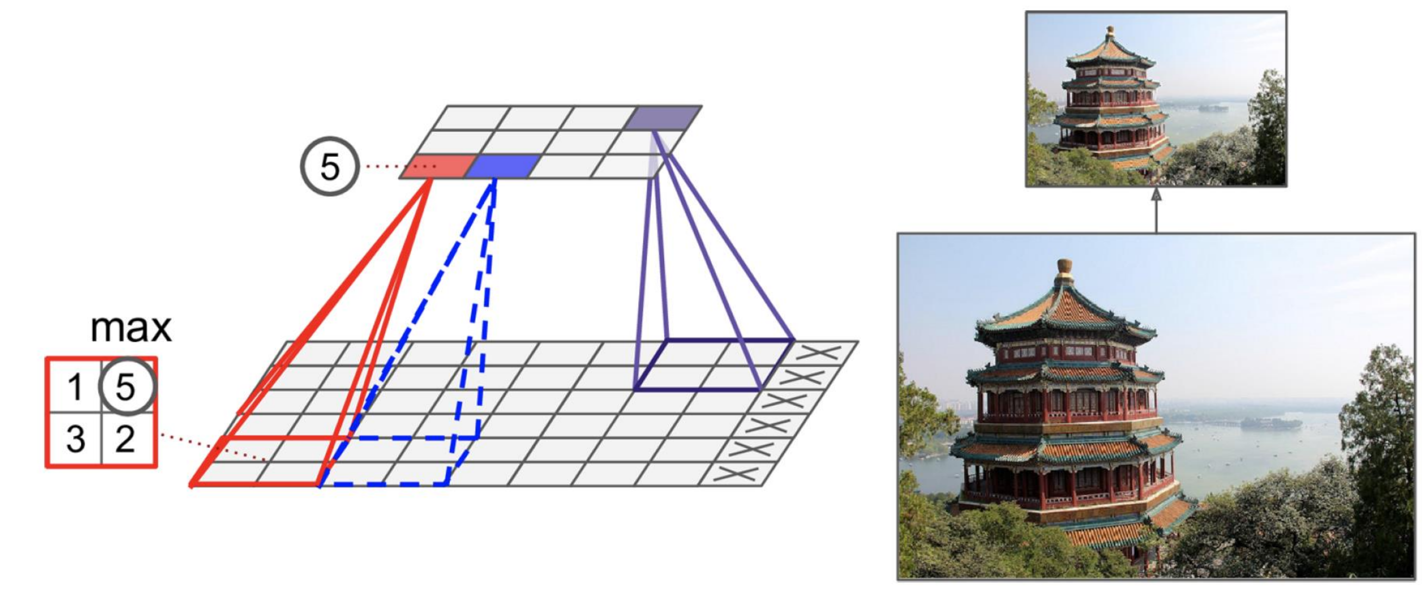




Source: https://devblogs.nvidia.com/deep-learning-nutshell-core-concepts/

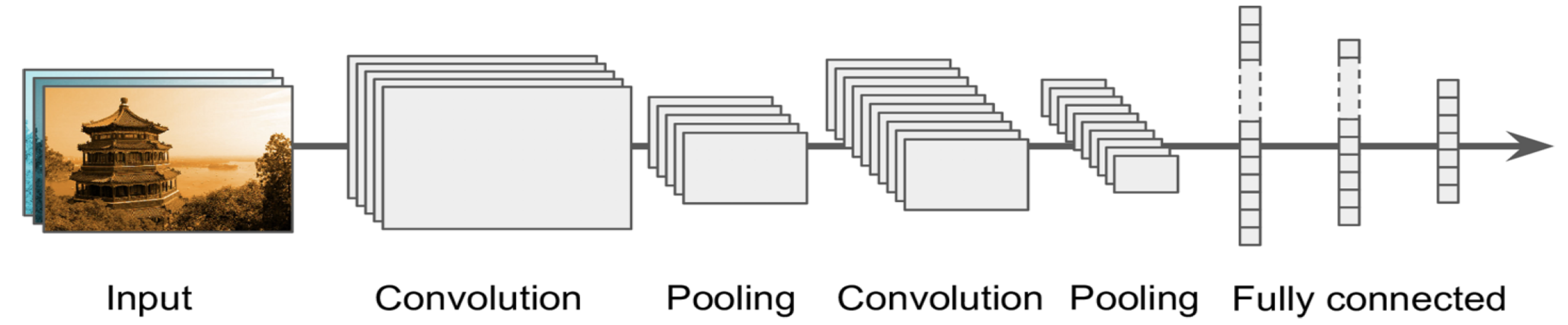
Idea: subsample the output of a layers.

- Aggregation function, e.g., mean or max.
- No weights, but requires the definition of size, stride, padding.



Day 09 CNN Architecture

The conventional convolutional neural network is formed by convolutional layers, pooling layers and fully connected layers (with activation functions).



Idea: Generate feature maps that contain relevant (discriminative) and compacted high-level features about the input image.

Final layer outputs the prediction of the class.

CNN vs Deep MLPs

CNNs propose partially connected layers

- As all neurons in a feature map share the same weights and bias, CNN models significantly reduces the number of parameters of the model.
- E.g., consider 100 x 100 images -> 10,000 pixels. A first MLP layer with 1,000 neurons (quite limited) requires 10 million parameters (just the first layer!).

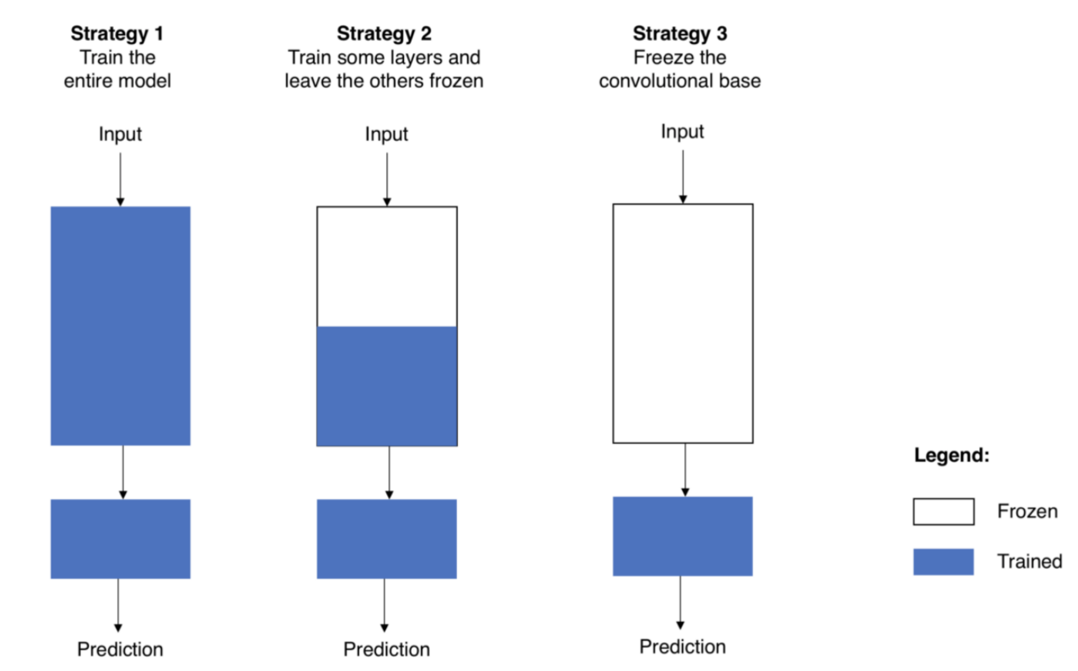
Location and rotation invariance

- CNNs learns a pattern independently of its location and 2D rotation.
- Transfer learning.

Day 09 Transfer Learning

Time efficient design of accurate deep learning models.

- **Idea: use of pre-trained models** on tasks similar to what you want to solve.



Source: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Day 09 Powerful architectures

Variants of the fundamental architecture used for transfer learning.

InceptionV4
https://arxiv.org/pdf/1602.07261.pdf

VGG
http://www.robots.ox.ac.uk/~vgg/practicals/cnn/index.html

GoogLeNet
https://ai.google/research/pubs/pub43022

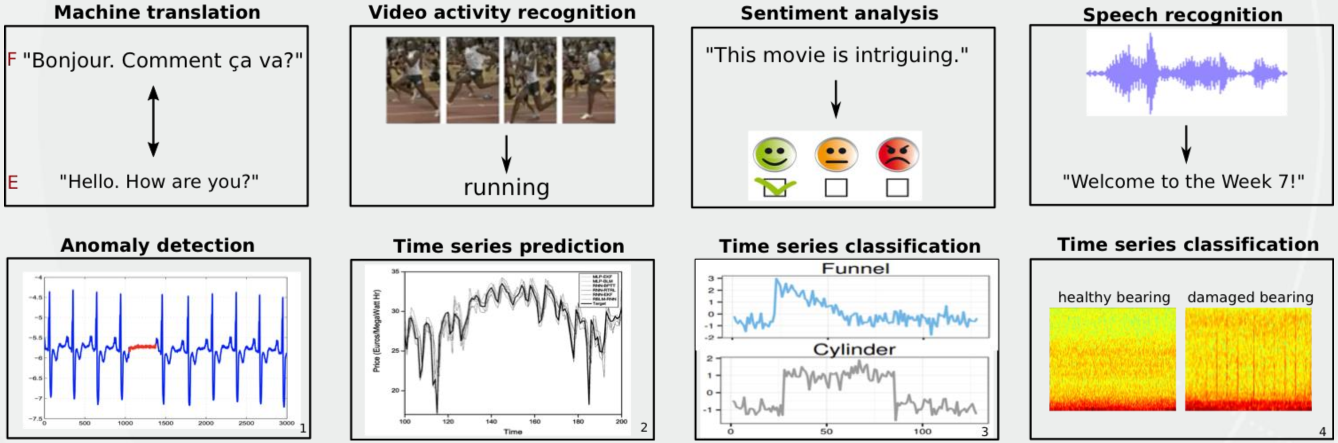
ResNet
https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

Day 9 Recurrent Neural Networks (RNNs)

Sequence models

When to use sequence models?

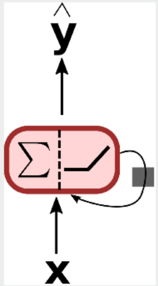
- Temporal dynamics connecting data samples are important.
- Context is important: past samples or decisions influence current predictions.
- Share features across different positions of the data is important.

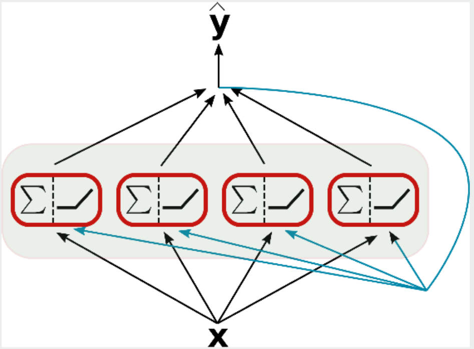


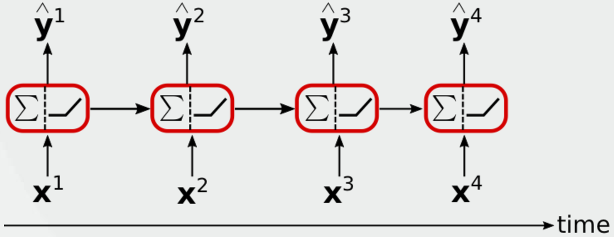
1 - ChandolaV, Banerjee A, Kumar V. Anomaly detection: A survey. ACM computing surveys (CSUR). 2009 Jul 1;41(3):15.
2 - MirikitaniDT, NikolaevN. Recursive bayesianrecurrent neural networks for time-series modeling. IEEE Transactions on Neural Networks. 2010 Feb;21(2):262-74.
3 - https://cs.gmu.edu/~xwang24/Projects/RPM.html
4 - Zak, G, et al. Application of ARMA modelling and alpha-stable distribution for local damage detection in bearings, Diagnostyka,2014.

The Recurrent Neural Network Model

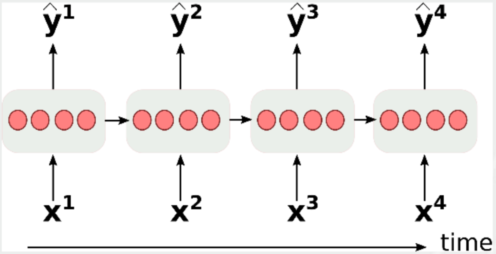
- Similar to their feedforward counterpart, but including loop connections, allowing information to persist.







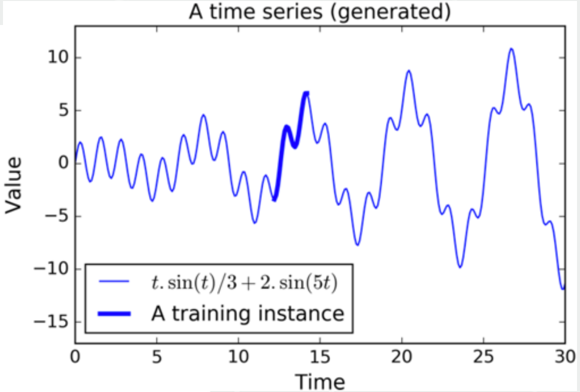
Unrolled recurrent neuron

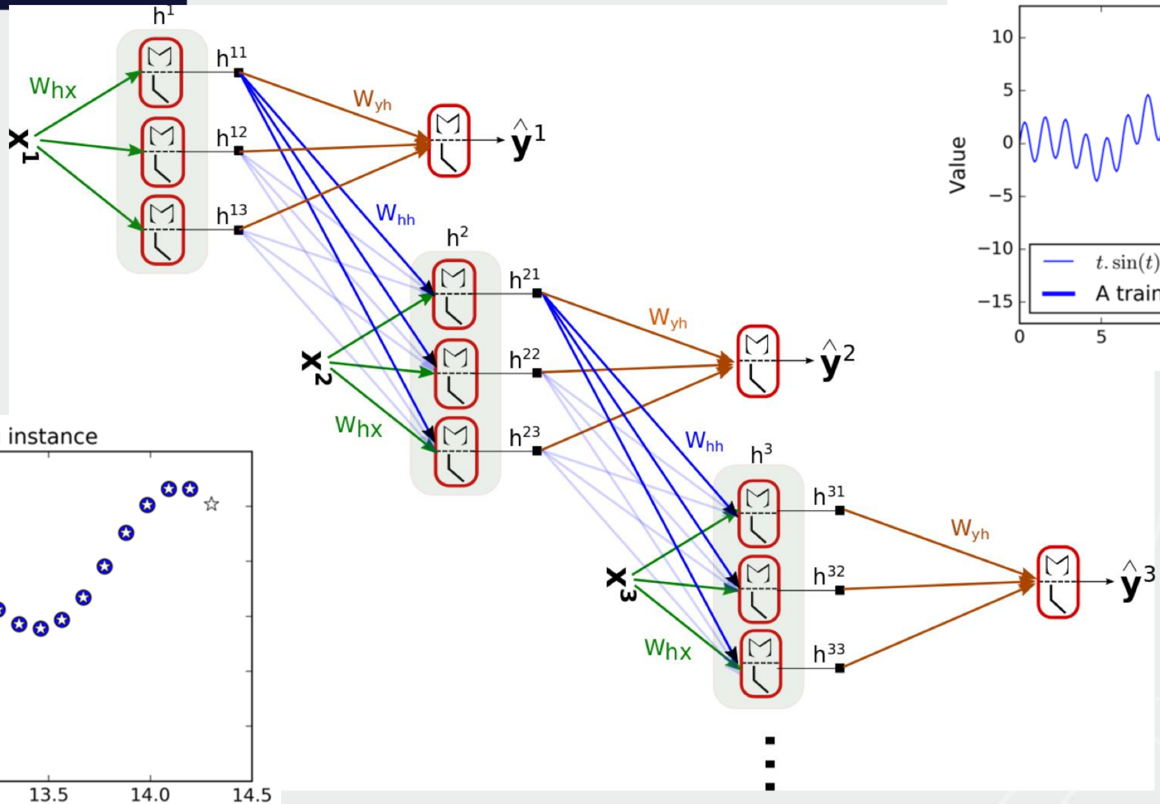


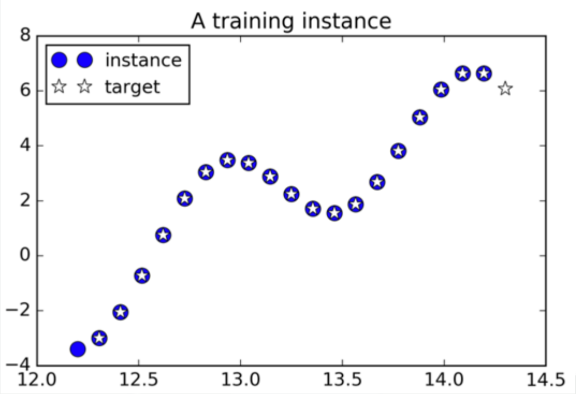
Layer of unrolled recurrent neurons

Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 14.

The RNN computations



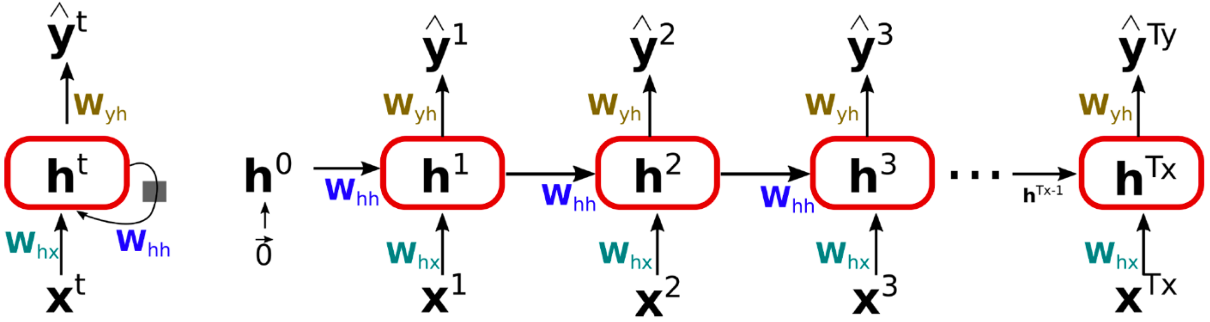




Débora/Thomas (UWA)

Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 14.

The RNN computations



$$\mathbf{h}^1 = \phi(\mathbf{W}_{hh} \cdot \mathbf{h}^0 + \mathbf{W}_{hx} \cdot \mathbf{x}^1 + \mathbf{b}_h)$$
$$\hat{\mathbf{y}}^1 = \phi(\mathbf{W}_{yh} \cdot \mathbf{h}^1 + \mathbf{b}_y)$$

More general:

$$\mathbf{h}^t = \phi(\mathbf{W}_{hh} \cdot \mathbf{h}^{t-1} + \mathbf{W}_{hx} \cdot \mathbf{x}^t + \mathbf{b}_h)$$
$$\hat{\mathbf{y}}^t = \phi(\mathbf{W}_{yh} \cdot \mathbf{h}^t + \mathbf{b}_y)$$

(or no activation function for $\hat{\mathbf{y}}^t$)

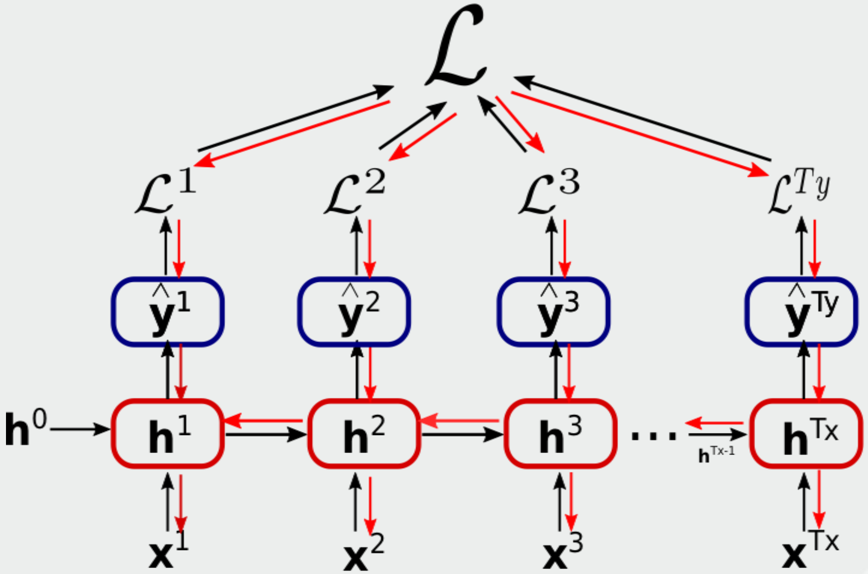
$$[\mathbf{W}_{hh}\ \mathbf{W}_{hx}] = \mathbf{W}_h$$
$$[\mathbf{h}^{t-1}, \mathbf{x}^t] = \begin{bmatrix} \mathbf{h}^{t-1} \\ \mathbf{x}^t \end{bmatrix}$$

$$\mathbf{h}^t = \phi(\mathbf{W}_h \cdot [\mathbf{h}^{t-1}, \mathbf{x}^t] + \mathbf{b}_h)$$

Source: Hands on Machine Learning with Scikit-Learn and TensorFlow, Cap. 14.

Backpropagation through time

- **Forward pass:** Training instances are fed to the network and output are computed.
- **Backward pass:** Measures the error gradients (in all time steps) with respect to all parameters in the network by (back)propagating the error gradient.

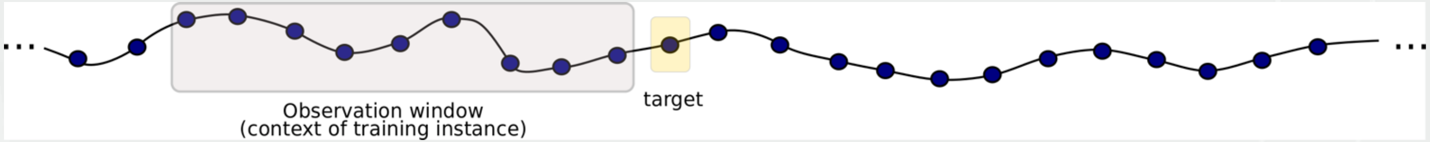


Source: P. J. Werbos, 1990. Backpropagation through time: what it does and how to do it

Vanishing Gradients

A basic RNN algorithm very often runs into vanishing gradient problems.

- To train it on long sequences, many time steps make the unrolled RNN similar to a very deep network.
- The gradients get smaller and smaller as GD goes down to the lower layers - for which connection weights remain practically unchanged.



Sentence 1

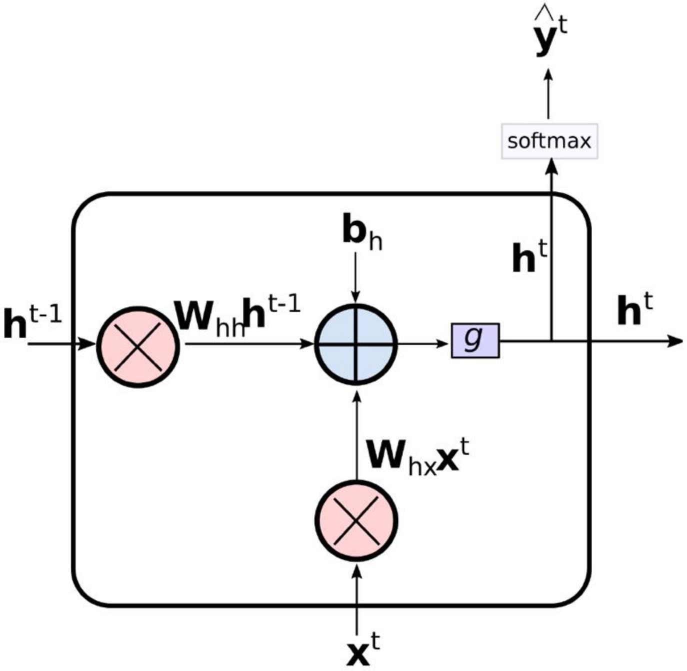
"Jane walked into the room. John walked in too. Jane said hi to ___"

Sentence 2

"Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to ___"

Pascanu et al., 2013. On the difficulty of training recurrent neural networks.
Sentence examples from https://cs224d.stanford.edu/lecture_notes/LectureNotes4.pd

Advanced RNNs



Cell state:

$$\mathbf{h}^t = g(\mathbf{W}_{hh} \cdot \mathbf{h}^{t-1} + \mathbf{W}_{hx} \cdot \mathbf{x}^t + \mathbf{b}_h)$$

Cell output:

$$\hat{\mathbf{y}}^t = softmax(\mathbf{W}_{yh} \cdot \mathbf{h}^t + \mathbf{b}_y)$$

Softmax:

Outputs a vector with the probability of each class.

For regression/prediction:

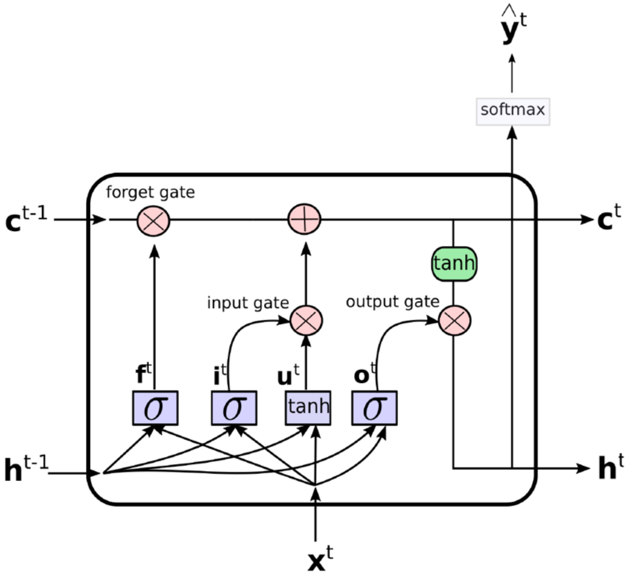
$$\hat{\mathbf{y}}^t = g(\mathbf{W}_{yh} \cdot \mathbf{h}^t + \mathbf{b}_y)$$

or no activation function at all.

Popular activations functions for $g(\cdot)$ are: sigmoid, tanh and ReLU.

Advanced RNNs – Long Short-term Memory (LSTM)

- The LSTM cell learns long-term dependencies in the data, and training usually converge faster than with the basic RNNs.



Forget gate:

$$\mathbf{f}^t = \sigma(\mathbf{W}_{fx} \cdot \mathbf{x}^t + \mathbf{W}_{fh} \cdot \mathbf{h}^{t-1} + \mathbf{b}_f)$$

Input/update gate :

$$\mathbf{i}^t = \sigma(\mathbf{W}_{ix} \cdot \mathbf{x}^t + \mathbf{W}_{ih} \cdot \mathbf{h}^{t-1} + \mathbf{b}_i)$$

Cell actual state

$$\mathbf{u}^t = \tanh(\mathbf{W}_{ux} \cdot \mathbf{x}^t + \mathbf{W}_{uh} \cdot \mathbf{h}^{t-1} + \mathbf{b}_u)$$

Output gate:

$$\mathbf{o}^t = \sigma(\mathbf{W}^{ox} \cdot \mathbf{x}^t + \mathbf{W}_{oh} \cdot \mathbf{h}^{t-1} + \mathbf{b}_o)$$

Cell new state:

$$\mathbf{c}^t = \mathbf{f}^t \otimes \mathbf{c}^{t-1} + \mathbf{i}^t \otimes \mathbf{u}^t$$

Cell output:

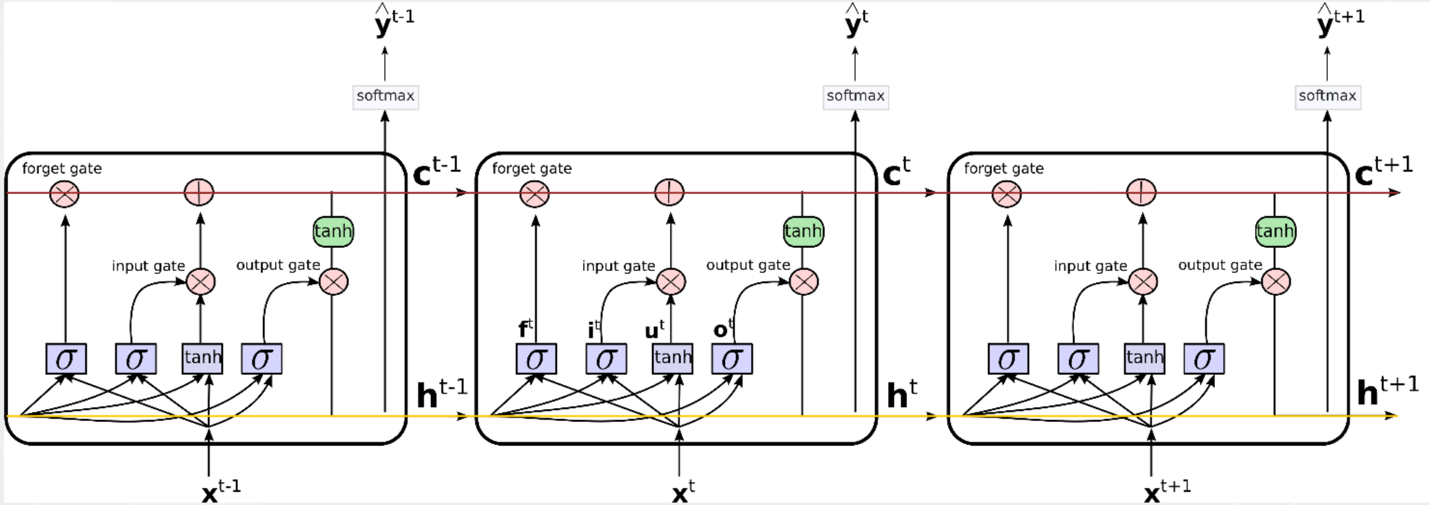
$$\hat{\mathbf{y}}^t = \mathbf{h}^t = \mathbf{o}^t \otimes \tanh(\mathbf{c}^t)$$

S. Hochreiterand J. Schmidhuber, 1997. Long Short-term Memory.
Gers et al., 1999. Learning to forget: continual prediction with LSTM.
http://colah.github.io/posts/2015-08-Understanding-LSTMs

Advanced RNNs – LSTM forward pass

- The cell state $\mathbf{c}^t$ is the important aspect of the LSTM: it flows the information related to long-term memories.

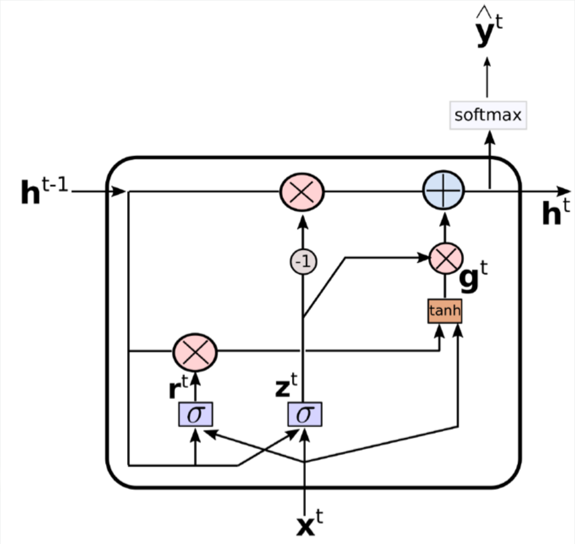
- The short-term state $\mathbf{h}^t$ works similar as the basic RNN cell, and its output goes directly to $\mathbf{y}^t$.



Variants and improvements:
F.A. Gers and J. Schmidhuber, 2000. Recurrent nets that time and count.
Gers et al., 2002, Learning Precise Timing with LSTM Recurrent Networks.
Zaremba et al., 2015, Recurrent Neural Network Regularization

Advanced RNNs – Gated Recurrent Unit (GRU) cell

- The GRU cell is a simpler (but effective) variant of the LSTM cell.
- GRU and LSTM cells have showed similar results for many tasks.



Gates:

$\mathbf{r}^t = \sigma(\mathbf{W}_{rx} \cdot \mathbf{x}^t + \mathbf{W}_{rh} \cdot \mathbf{h}^{t-1})$

It controls which parts from the previous state will be presented to $\mathbf{g^t}$.

$\mathbf{z}^t = \sigma(\mathbf{W}_{zx} \cdot \mathbf{x}^t + \mathbf{W}_{zh} \cdot \mathbf{h}^{t-1})$

It works as the input and forget gates.

$\mathbf{g}^t = \tanh(\mathbf{W}_{gx} \cdot \mathbf{x}^t + \mathbf{W}_{gh} \cdot (\mathbf{r}^t \otimes \mathbf{h}^{t-1}))$

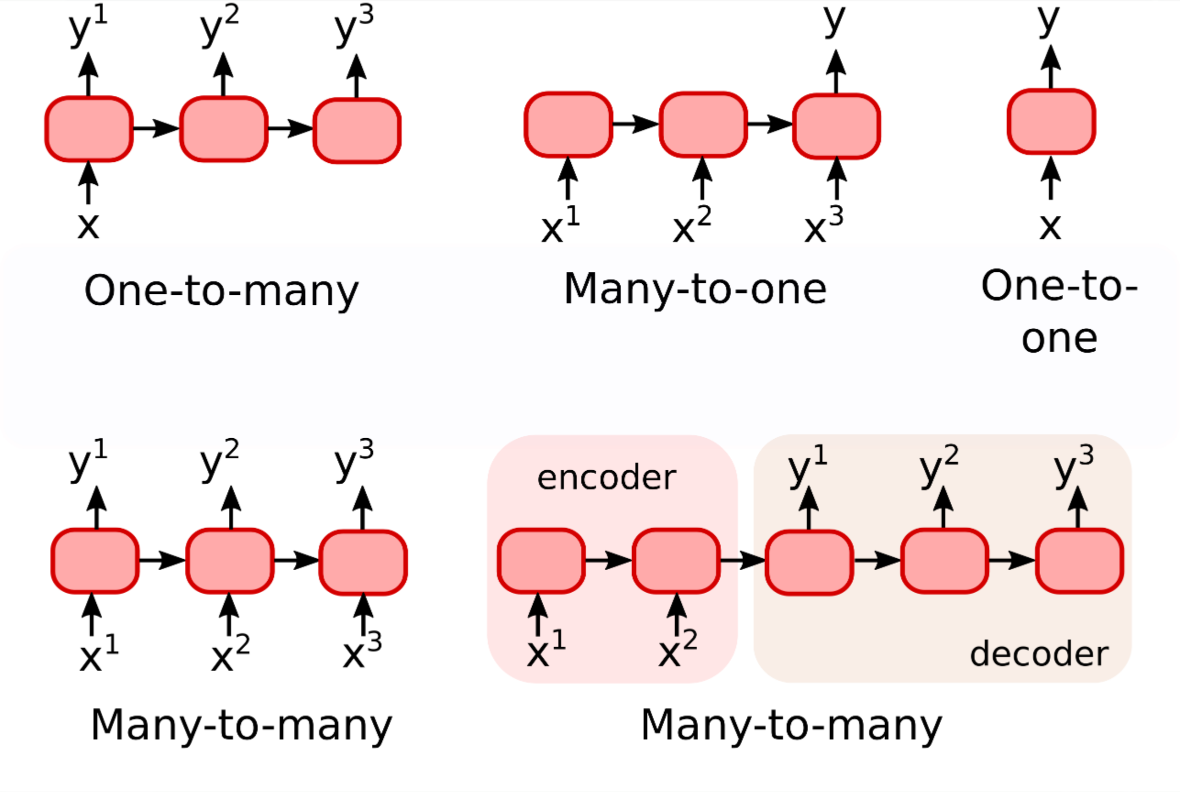
The new candidate state vector.

State vectors are merged:

$\mathbf{h}^t = (1 - \mathbf{z}^t) \otimes \mathbf{h}^{t-1} + \mathbf{z}^t \otimes \mathbf{g}^t$

Cho et al., 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.
Cho et al., 2014. On the Properties of Neural Machine Translation: Encoder - Decoder Approaches.
Chung et al., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling.
Greff et al., 2015. LSTM: A Search Space Odyssey.
Jozefowicz et al., 2015. An Empirical Exploration of Recurrent Network Architectures.

Types of RNN



y^1 y^2 y^3
x
One-to-many
y
x^1 x^2 x^3
Many-to-one
y
x
One-to-one
y^1 y^2 y^3
x^1 x^2 x^3
Many-to-many
encoder
x^1 x^2
y^1 y^2 y^3
decoder
Many-to-many

Discussion

What are ANNs possible applications in industry?

Discussion

What are ANNs possible applications in industry?

- Use of satellite images (remote sensing) to map the impacts to the environments of mining actives. Aim: identify locations which require attention in order to prevent extreme impact and fine from the government.

- Real-time analysis of components in a haul truck for predictive analysis, for instance, to identify whether this component is getting extreme heating when in operation.

- Given images from minerals, identify the quality of such materials.

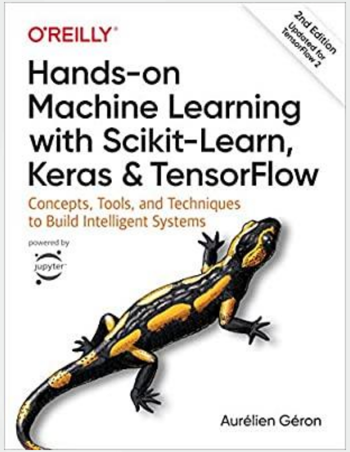
- Identification of high risk areas for slips, trips and falls in order to assist with workers' security.

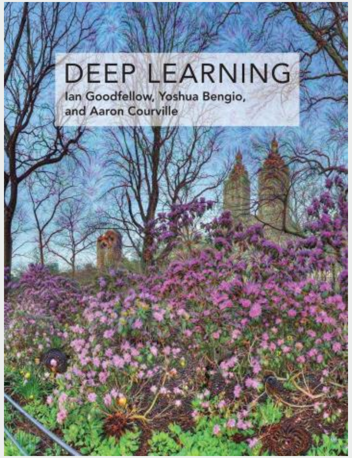
- Preventive analysis of fires in underground mines.

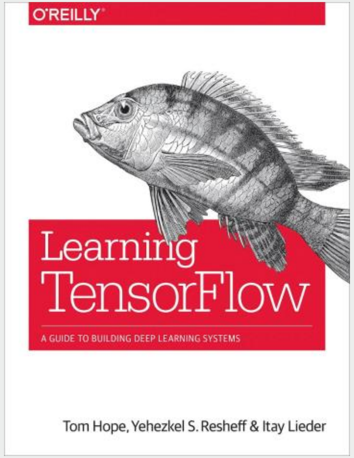
- Prediction of sky clearness and solar radiations for solar energy applications.

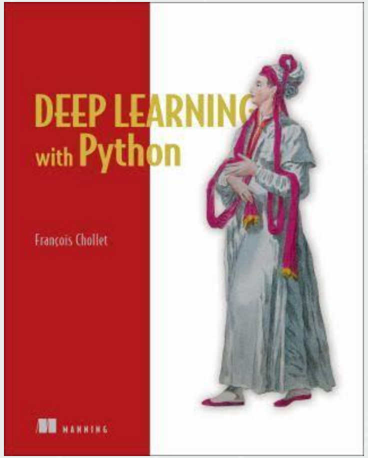
- Forecast of electrical energy consumption

Day 09 Suggested references









Scikit-learn: http://scikit-learn.org
TensorFlow: https://www.tensorflow.org
Andrew Ng deep learning page: https://www.deeplearning.ai

CORE Skills

A CORE partnership

COREHUB.COM.AU/SKILLS