

데이터 마이닝 특강

Practice session

[Classification]

Junseok Park

2018-08-23



Clustering review

- **Practice of clustering algorithms**
 - K-mean
 - Hierarchical clustering (H-Clustering)r
 - Self organizing map (SOM)
- **Realword data practice**
 - Data preprocessing and transformation
 - K-mean
 - H-Clustering

Overview

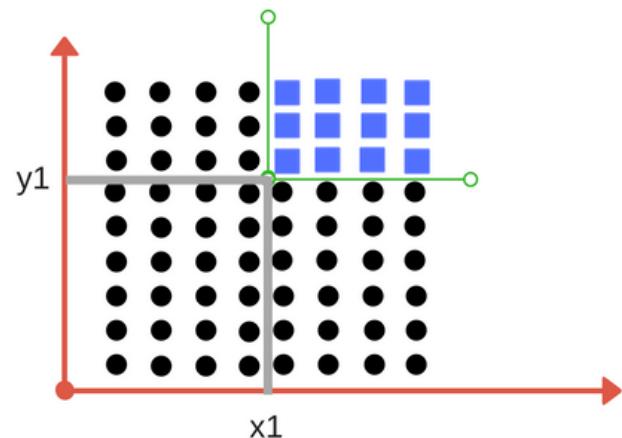
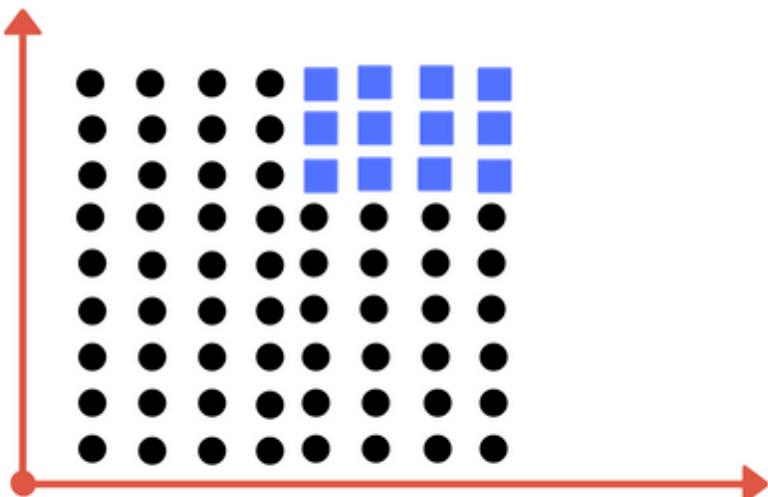
- **Decision Tree**
 - Introduction
 - Practice
- **Multi Layer Perceptron (MLP)**
 - Introduction
 - Practice with toy data
 - Practice with MNIST data
 - Other playground on google

Decision Tree

Decision Tree

- Motivation

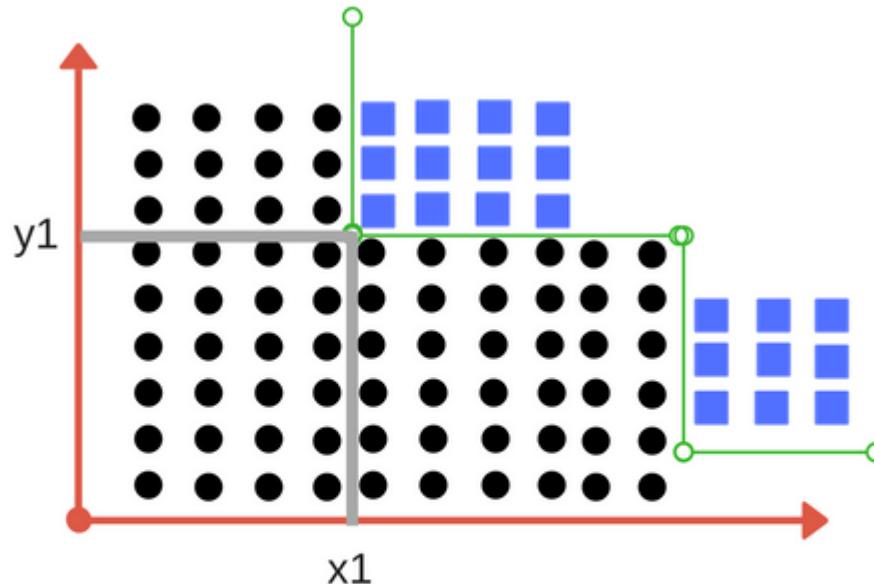
- Is it possible to draw a single separation line?



Decision Tree

- **Definition**

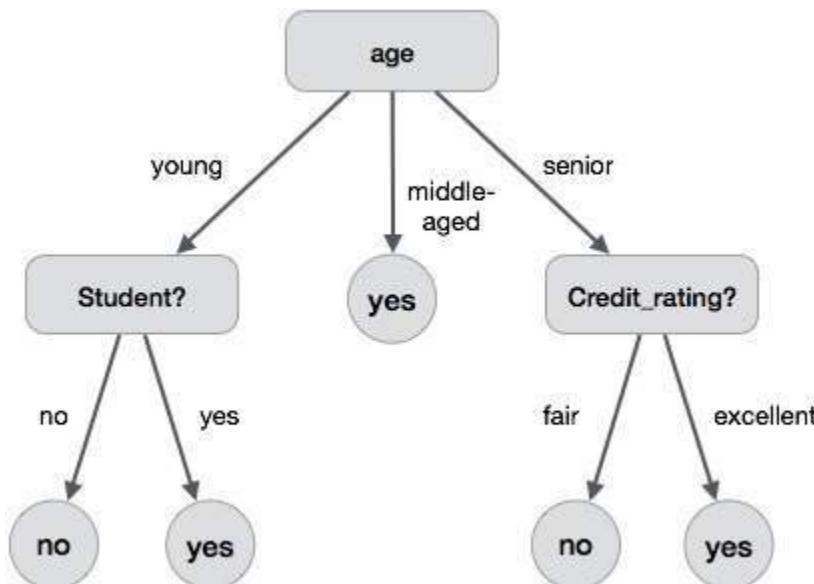
- Decision Tree classifier, repetitively divides the working area(plot) into sub part by identifying lines (repetitively because there may be two distant regions of same class divided by other as show in image below)



Decision Tree

- **Impurity**

- Impurity is when we have traces of one class division into other. This can arise due to following reason
 - We run out of available features to divide the class upon.
 - We tolerate some percentage of impurity (we stop further division) for faster performance. (There is always trade off between accuracy and performance).

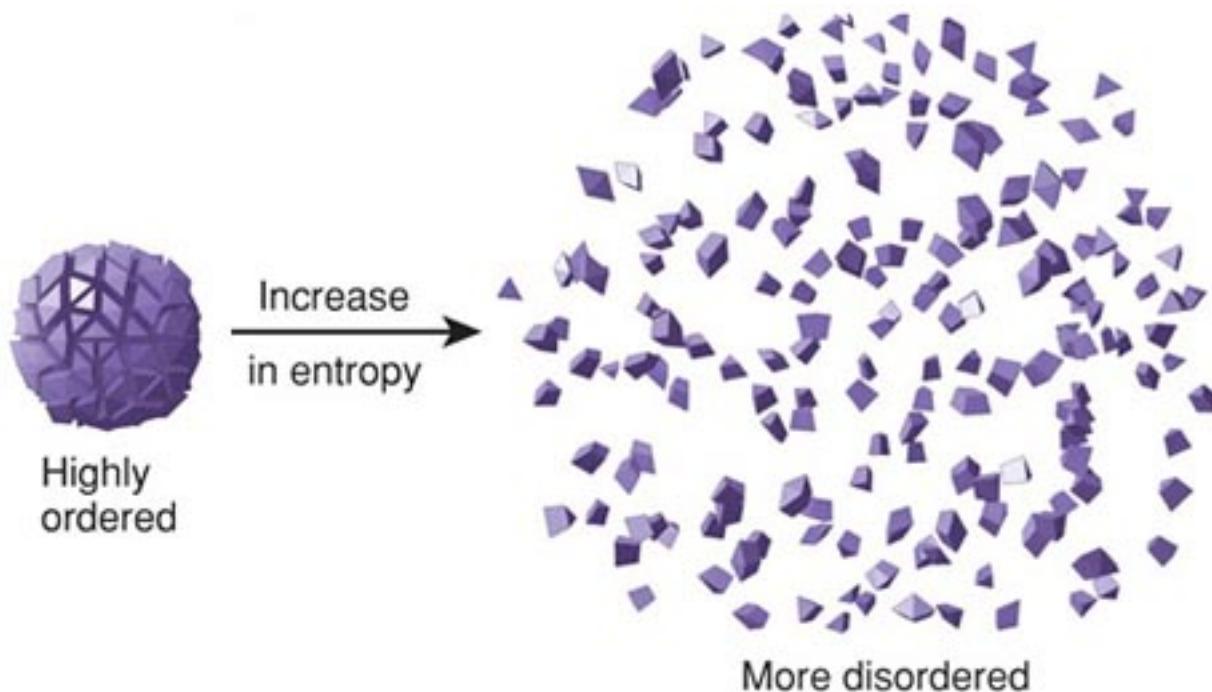


"Gini Impurity"

Decision Tree

- **Entropy**

- Entropy is degree of randomness of elements or in other words it is *measure of impurity*. Mathematically, it can be calculated with the help of probability of the items as:



<https://socratic.org/questions/what-does-entropy-measure>

Decision Tree

- **Entropy**

- It is negative summation of probability times the log of probability of item

$$H = - \sum p(x) \log p(x)$$

For example

if we have items as number of dice face occurrence in a throw event as 1123,

the entropy is

$$p(1) = 0.5$$

$$p(2) = 0.25$$

$$p(3) = 0.25$$

$$\begin{aligned} \text{entropy} &= -(0.5 * \log(0.5)) - (0.25 * \log(0.25)) - (0.25 * \log(0.25)) \\ &= 0.45 \end{aligned}$$

Decision Tree

- **Information Gain**

- a measure of changes in entropy

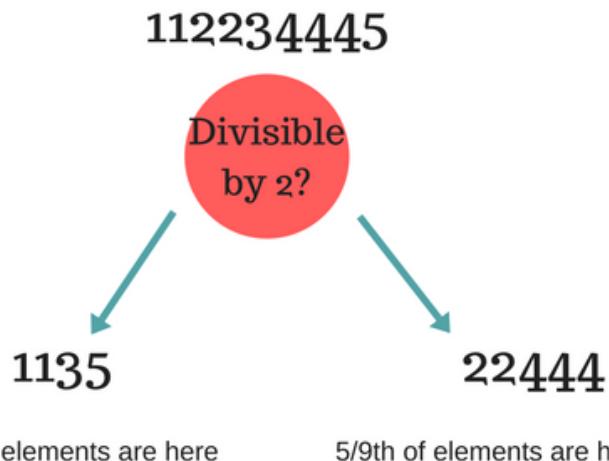
$$G(x, a) = H(x) - H(x|a)$$

Information Gain(n) =

Entropy(x) - ([weighted average] * entropy(children for features))

Decision Tree

- Information Gain



Entropy at root level : 0.66

Entropy of left child : 0.45 , weighted value = $(4/9) * 0.45 = 0.2$

Entropy of right child: 0.29 , weighted value = $(5/9) * 0.29 = 0.16$

Information Gain = $0.66 - [0.2 + 0.16] = 0.3$

Dataset (Cont'd)

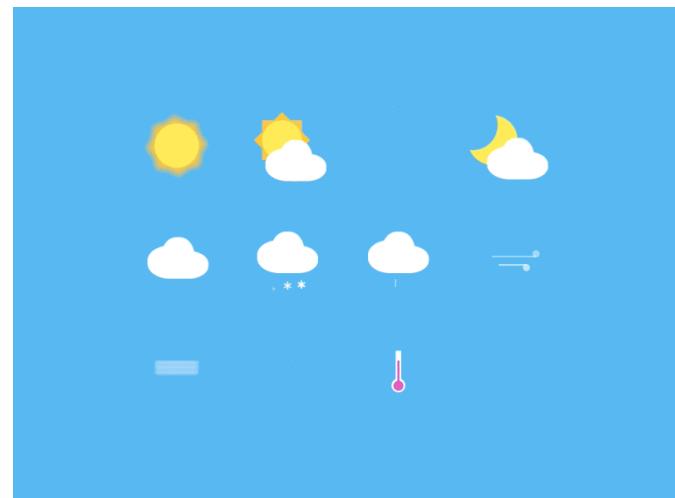
- Story of the data

- Sara likes weekend sailing. Though, not under any condition. Past twenty Wednesdays I have asked her if she will have any company, what kind of boat she can rent, and I have checked the weather forecast. Then, on Saturday, I wrote down if she actually went to the Sea.



Sailing boat

Images source : <https://www.bassproboatingcenters.com/brands.html>



Weathers

Images source : <https://www.pinterest.co.kr/pin/459367230724170802/>

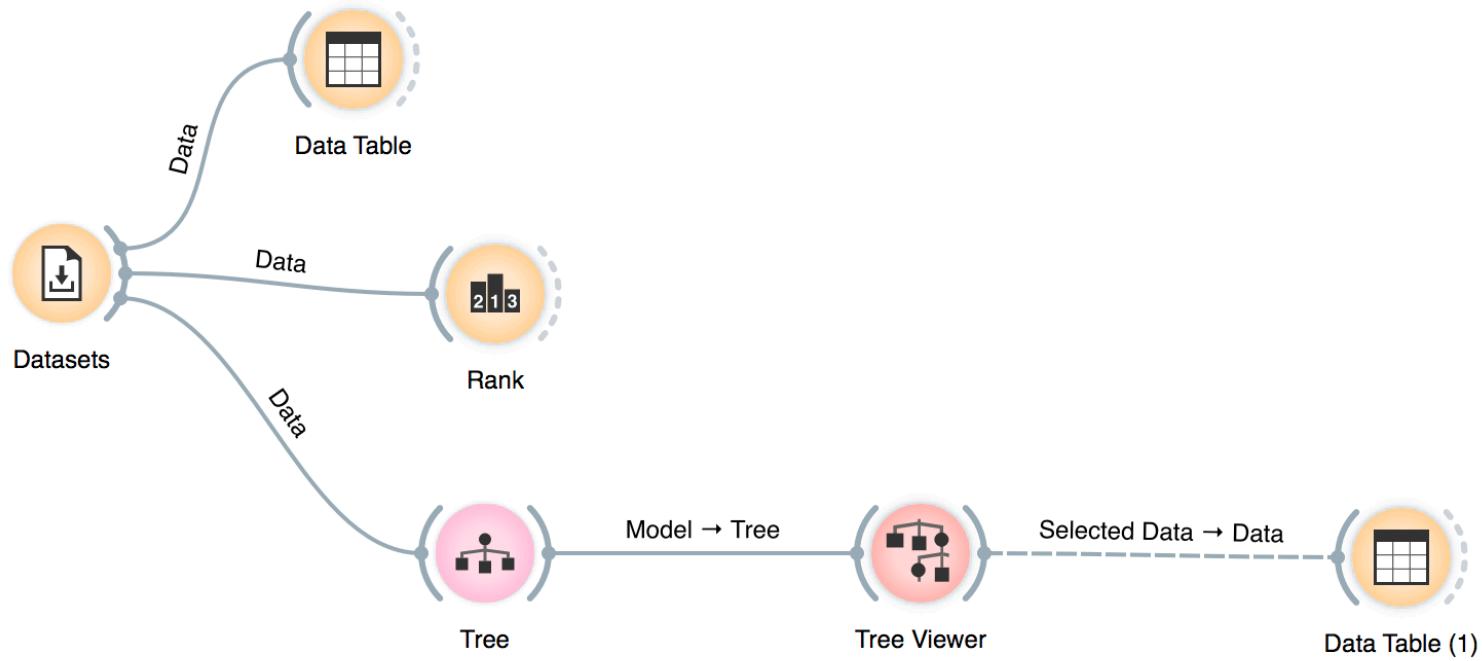
Dataset

- Sailing data
 - data that contains three attributes

The screenshot shows the Weka interface with the 'Datasets' window open. The 'Sailing' dataset is selected, showing details like size (455 bytes), instances (20), and variables (4 categorical). The 'Data Table' window displays the dataset's four columns: Sail, Outlook, Company, and Sailboat, with 20 rows of data. The 'Info' panel provides a summary: 20 instances (no missing values), 3 features (no missing values), Discrete class with 2 values (no missing values), and No meta attributes. The 'Variables' section includes checkboxes for 'Show variable labels (if present)', 'Visualize numeric values', and 'Color by instance classes', with the latter being checked. The 'Selection' section has a checkbox for 'Select full rows', which is also checked. At the bottom, there are buttons for 'Restore Original Order' and 'Send Automatically'.

	Sail	Outlook	Company	Sailboat
1	yes	rainy	big	big
2	yes	rainy	big	small
3	no	rainy	med	big
4	no	rainy	med	small
5	yes	sunny	big	big
6	yes	sunny	big	small
7	yes	sunny	med	big
8	yes	sunny	med	big
9	yes	sunny	med	small
10	yes	sunny	no	small
11	no	sunny	no	big
12	no	rainy	med	big
13	no	rainy	no	big
14	no	rainy	no	big
15	no	rainy	no	small
16	no	rainy	no	small
17	yes	sunny	big	big
18	no	sunny	big	small
19	no	sunny	med	big
20	no	sunny	med	big

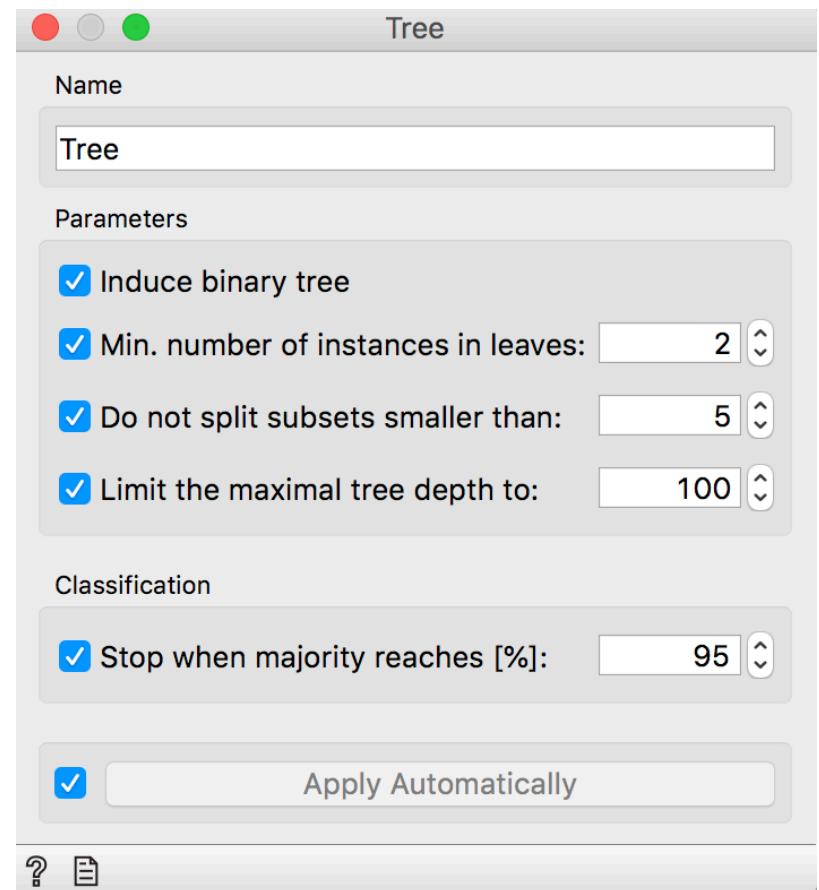
Build Schema



Hyperparameters and configuration

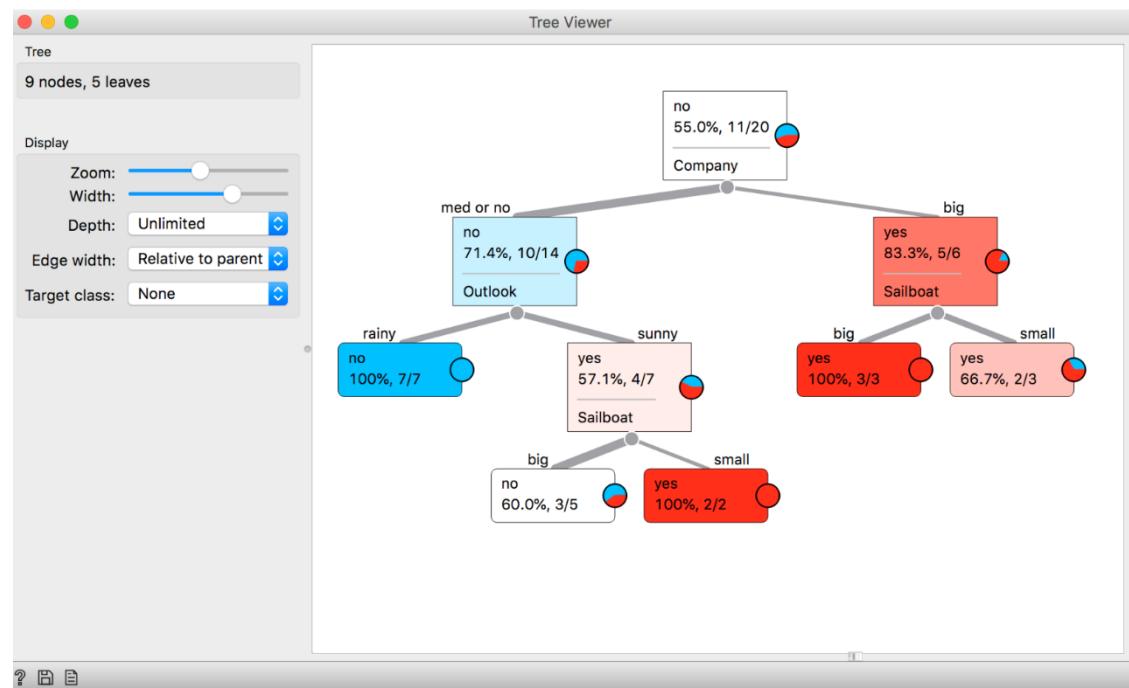
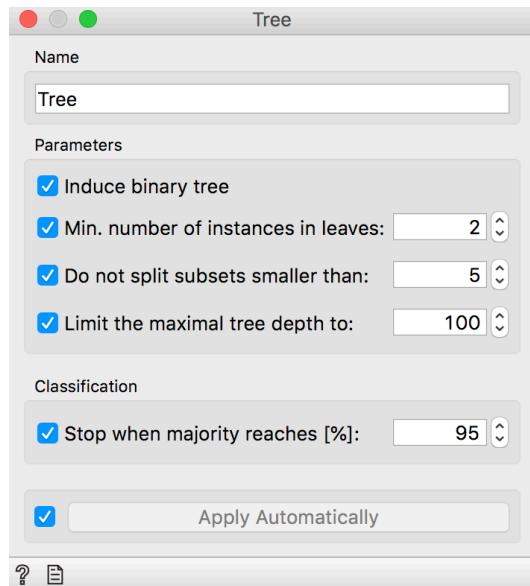
- **Parameters**

- Induce binary tree
 - build a binary tree (split into two child nodes)
- Min. number of instances in leaves
 - if checked, the algorithm will never construct a split which would put less than the specified number of training examples into any of the branches
- Do not split subsets smaller than
 - forbids the algorithm to split the nodes with less than the given number of instances
- Stop when majority reaches[%]
 - stop splitting the nodes after a specified majority threshold is reached



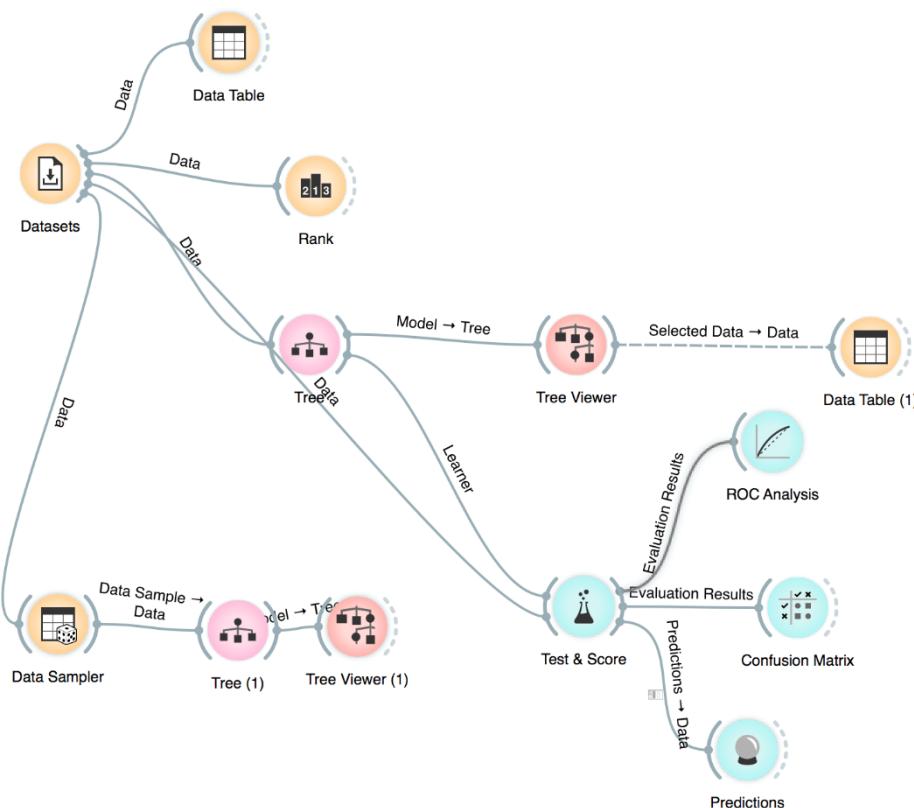
Observation of decision tree

- Self practices (5 min)



Build schema

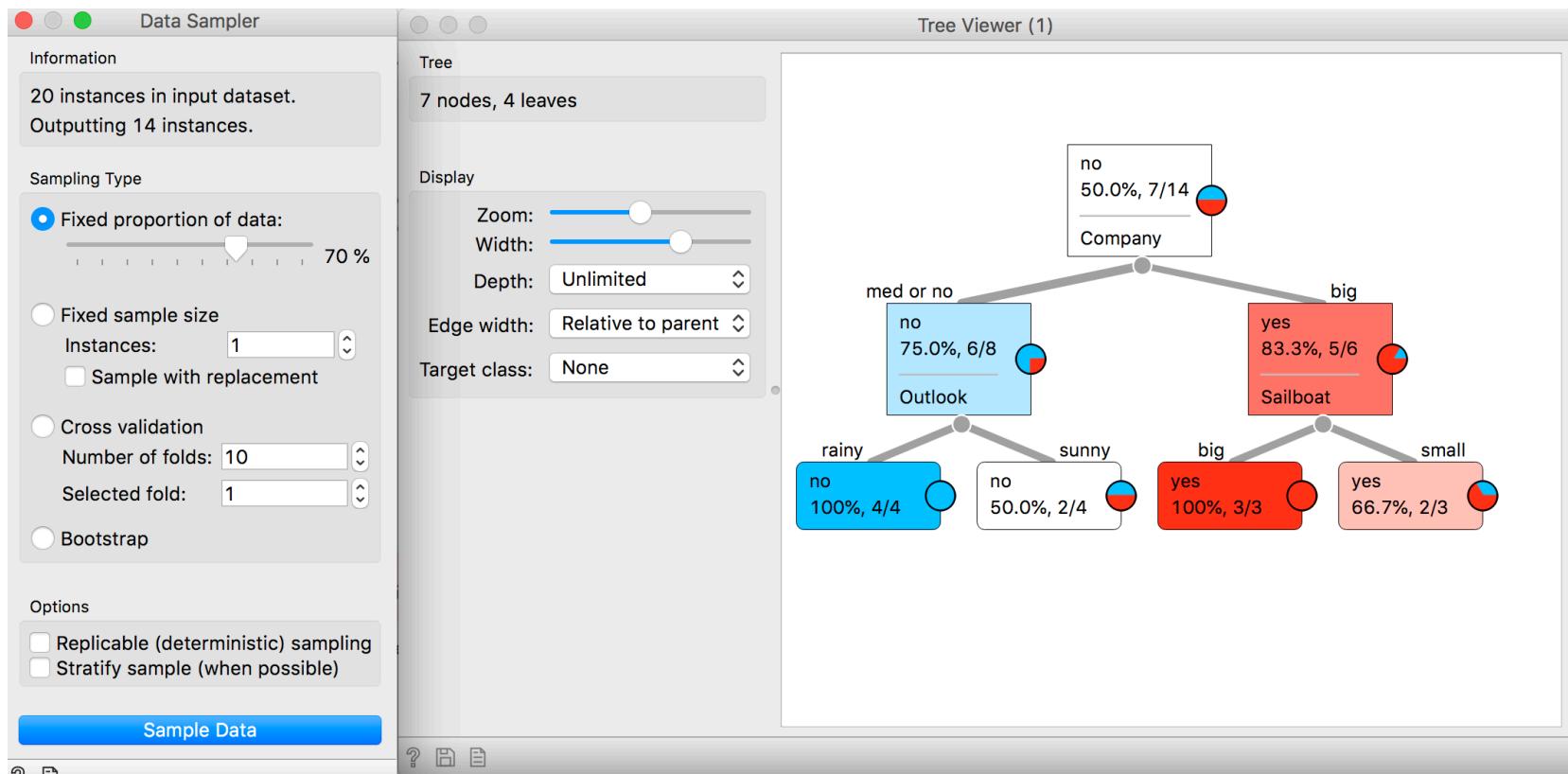
- Self Practices (10 min)
 - Comparison tests
 - Demonstration for instability of decision tree



Characteristic of decision tree

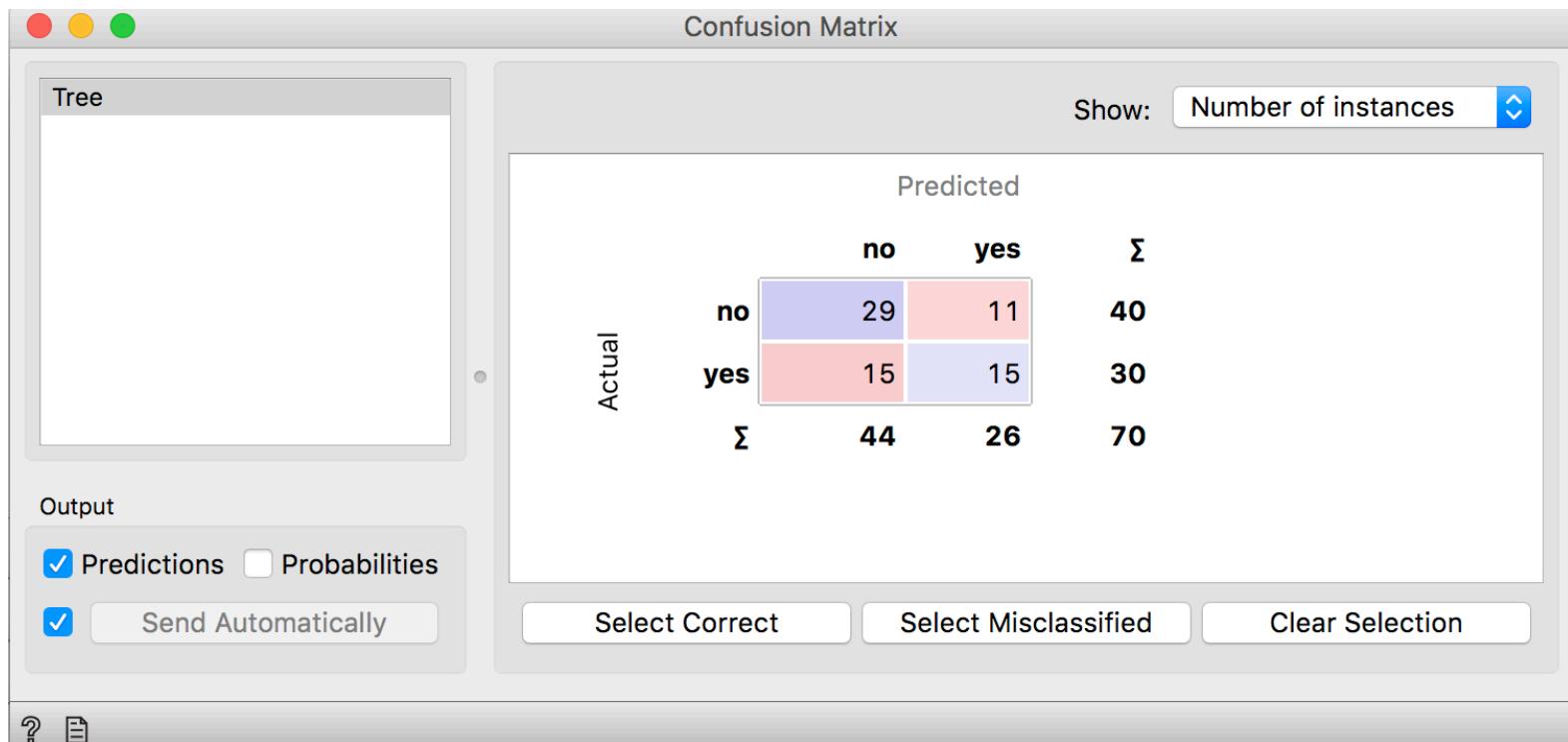
- **Instability**

- Decision tree is instable



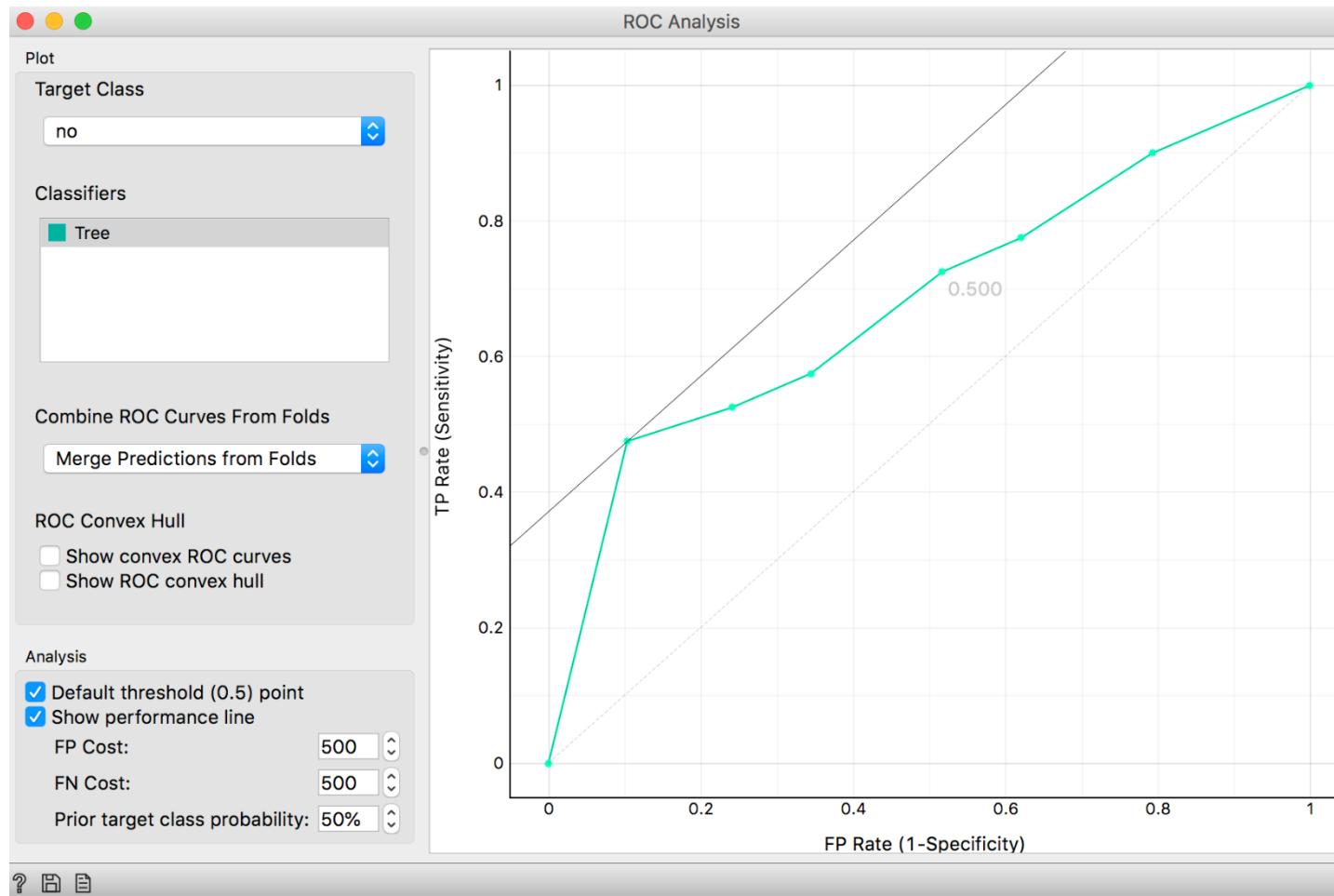
Performance test (Cont'd)

- Confusion matrix



Performance test (Cont'd)

- Area under curve(AUC) and



Performance test (Cont'd)

- Comparison tests

The screenshot shows the Weka Test & Score interface. On the left, under 'Sampling', 'Random sampling' is selected with 'Repeat train/test: 10' and 'Training set size: 66 %'. Under 'Target Class', '(Average over classes)' is chosen. On the right, the 'Evaluation Results' table displays the following data:

Method	AUC	CA	F1	Precision	Recall
Tree	0.729	0.629	0.624	0.624	0.629

Performance Test

- **Classification Accuracy (CA) Score**
 - In multilabel classification, this function computes subset accuracy:
- **Precision**
 - the ratio $tp / (tp + fp)$ where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
- **Recall**
 - The recall is the ratio $tp / (tp + fn)$ where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.
- **F1**
 - a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:
 - $$F1 = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

Results

Info
Data: 70 instances.
Predictors: N/A
Task: N/A
[Restore Original Order](#)

Data View
 Show full dataset

Output
 Original data
 Predictions
 Probabilities

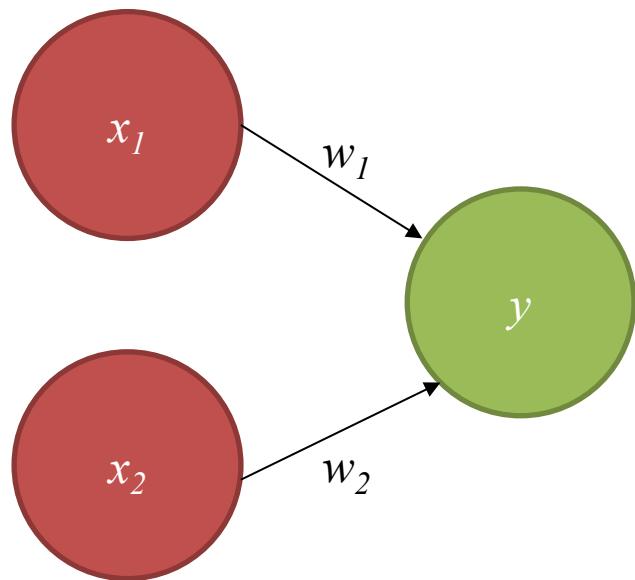
Predictions					
Sail	Tree	Tree (no)	Tree (yes)	Fold	Class
no	no	1.000	0.000	1	rainy
yes	no	0.600	0.400	1	sunny
no	no	1.000	0.000	1	rainy
yes	no	0.500	0.500	1	sunny
yes	no	0.600	0.400	1	sunny
no	no	1.000	0.000	1	rainy
no	no	1.000	0.000	1	rainy
yes	yes	0.250	0.750	2	sunny
no	no	1.000	0.000	2	rainy
yes	no	0.667	0.333	2	rainy
no	no	0.667	0.333	2	rainy
no	yes	0.250	0.750	2	sunny
no	no	1.000	0.000	2	rainy
yes	no	0.500	0.500	2	sunny
no	no	1.000	0.000	3	rainy
no	no	1.000	0.000	3	rainy
yes	no	0.500	0.500	3	sunny
no	yes	0.333	0.667	3	sunny

Multi Layer Perceptron

Perceptron

- Perceptron

- Assign inherent weight per input signal
- Circle : Neuron or Node
- Line : weight
- Theta : Threshold

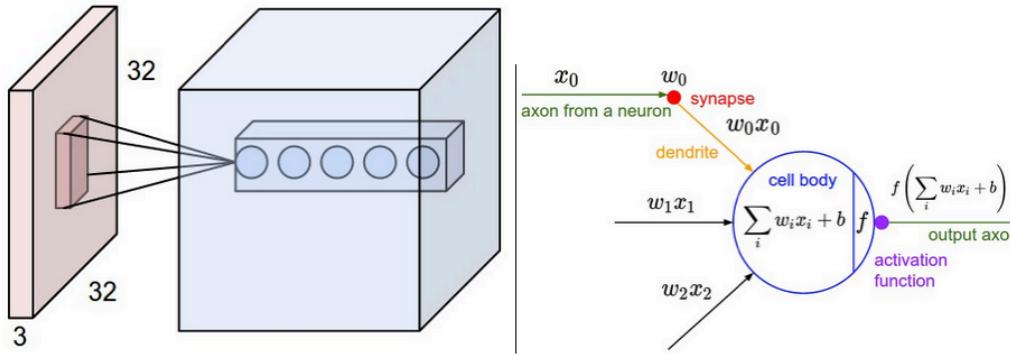


$$y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

Neural Network Overview

- Neural Network

- (before) Linear score function : $f = Wx$
- (now) 2-layer Neural Network : $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network : $f = W_3 \max(0, W_2 \max(0, W_1 x))$



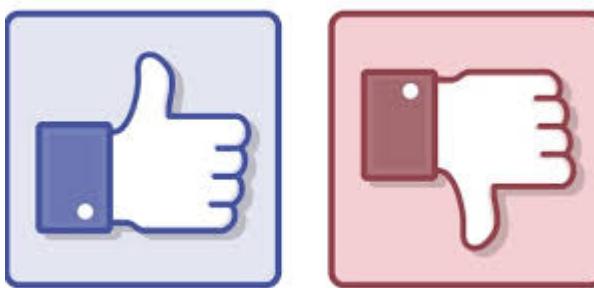
Left: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels). Note, there are multiple neurons (5 in this example) along the depth, all looking at the same region in the input - see discussion of depth columns in text below. **Right:** The neurons from the Neural Network chapter remain unchanged: They still compute a dot product of their weights with the input followed by a non-linearity, but their connectivity is now restricted to be local spatially.

- Learn about how we can interpret these computations from the neuron/network perspective

Modeling one neuron

- Single neuron as a linear classifier

neuron's capacity

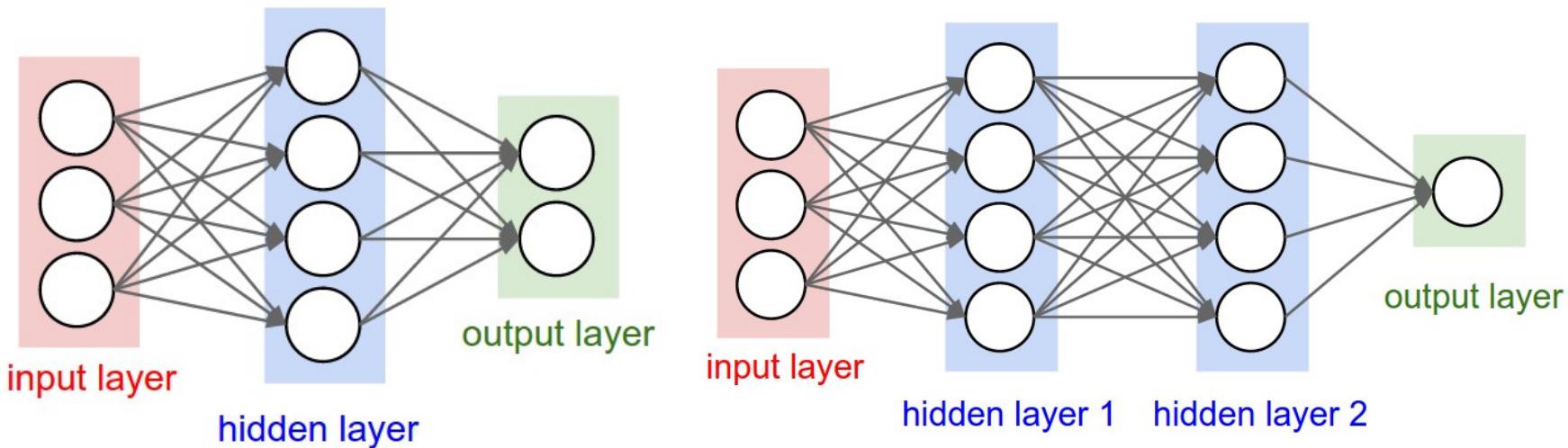


- Single neuron can be turned into a linear classifier with an appropriate loss function

Binary Softmax Classifier <i>Interpret $\sigma(\sum_i w_i x_i + b)$ as $P(y_i = 1 x_i; w)$</i>	Binary SVM Classifier <i>Max-margin hinge loss</i>
Regularization interpretation <i>Gradual forgetting</i>	

Neural Network Architecture

- Layer-wise organization
 - Neural Networks as neurons in graphs
 - The most common layer type is fully-connected layer

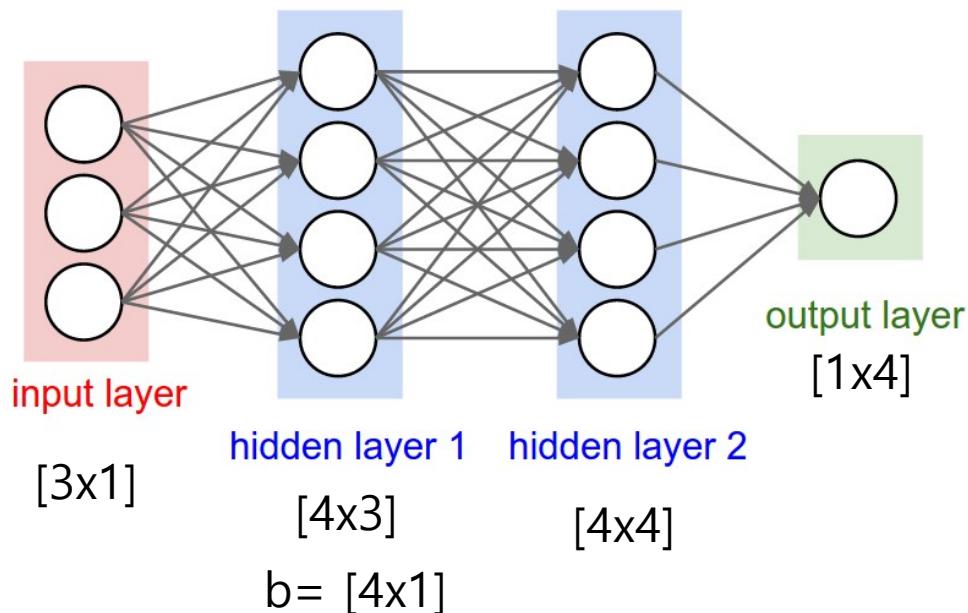


- **Naming conventions** : Artificial Neural Networks(ANN), Multi-Layer Perceptrons(MLP), neurons can be referred as *unit*.
- **Output layer** : commonly no activation function. Class scores, real-valued numbers, real-valued target
- **Sizing neural networks** : 4+2 neurons, $3 \times 4 + 4 \times 2$ weights, 4+ 2 biases for total 26 learnable parameters (left : A 2-layer Neural network)

Neural Network Architecture

- Example feed-forward computation

- *Repeated matrix multiplications interwoven with activation function*



- *The forward pass of a fully connected layer corresponds to one matrix multiplication followed by a bias offset and an activation function*

Neural Network Architecture

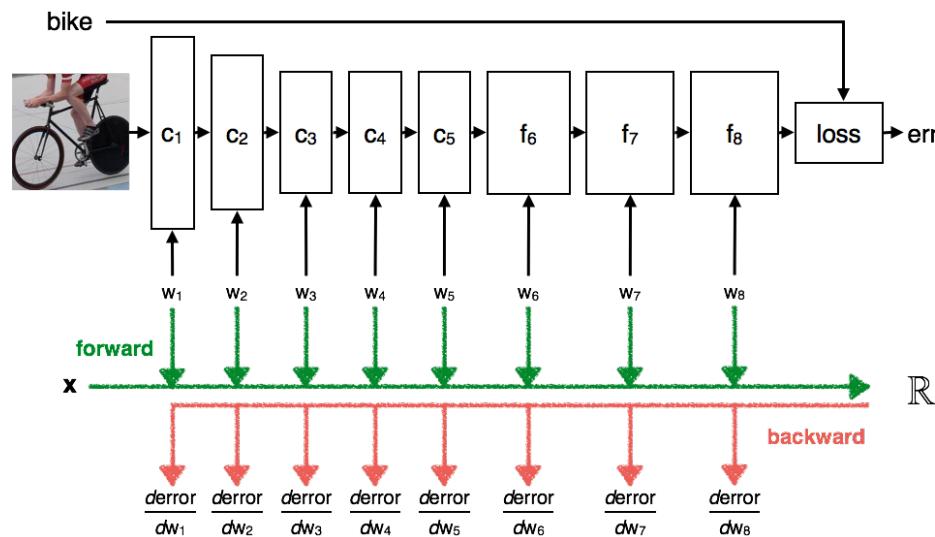
- **Representational Power**

- At least one hidden layer are *universal approximators* = the neural network can approximate any continuous function

(G. Cybenko, *Approximation by Superpositions of a Sigmoidal Function*, 1989)

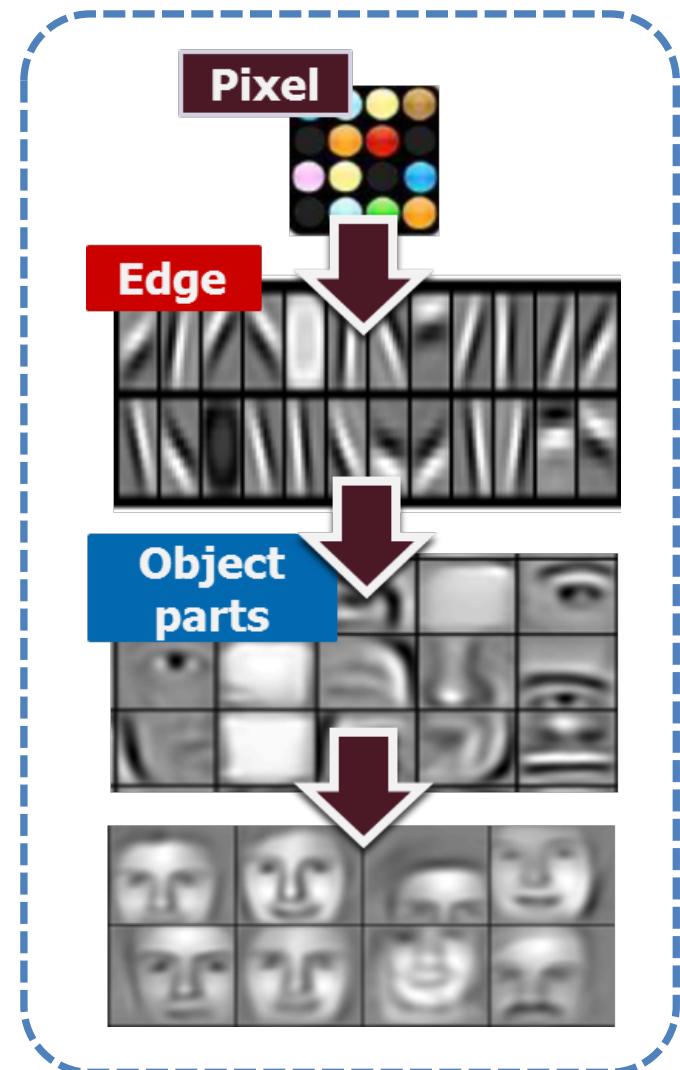
- Why use more layers and go deeper?

- Deep neural network works better in an empirical observation, despite the fact that their representation power is equal.
- For example, CNN depth is most important component for a good recognition system



Introduction

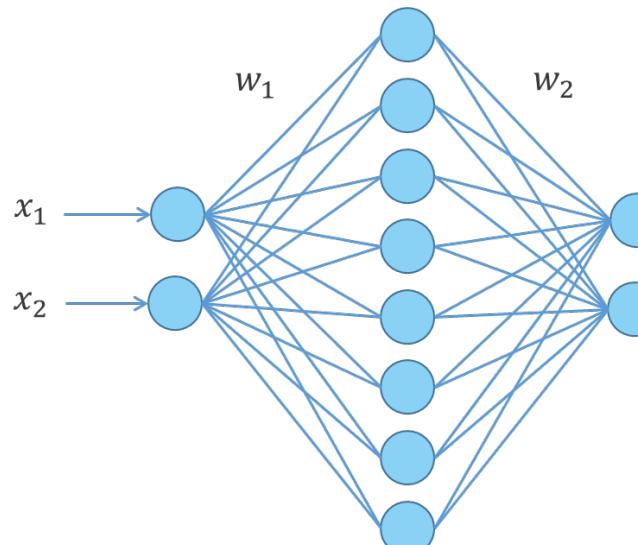
- **A deep network**
 - a neural network with multiple hidden layers ($hidden\ layer \geq 2$)
- **A deep learning**
 - a set of algorithms that model high-level abstractions in data by using deep networks.



[High-level abstraction]

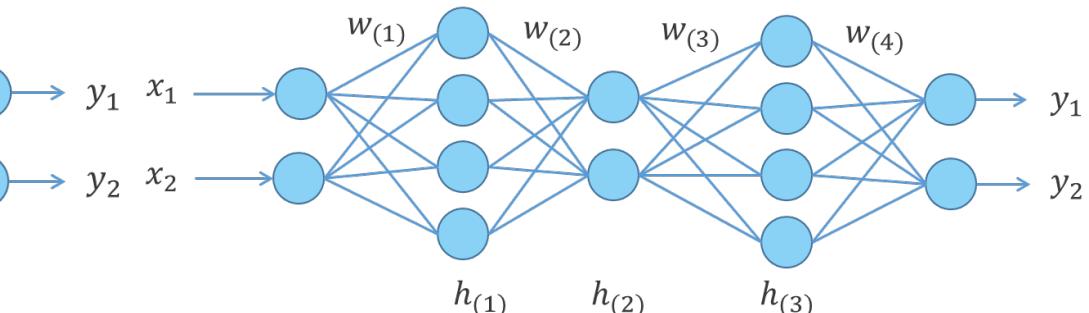
Characteristic of deep

- Deep means more hidden layers
 - The representation gets more hierarchical and abstract
 - It increases the model complexity, which leads to higher accuracy



[Shallow networks]

of paths from x_1 to y_1 = 8

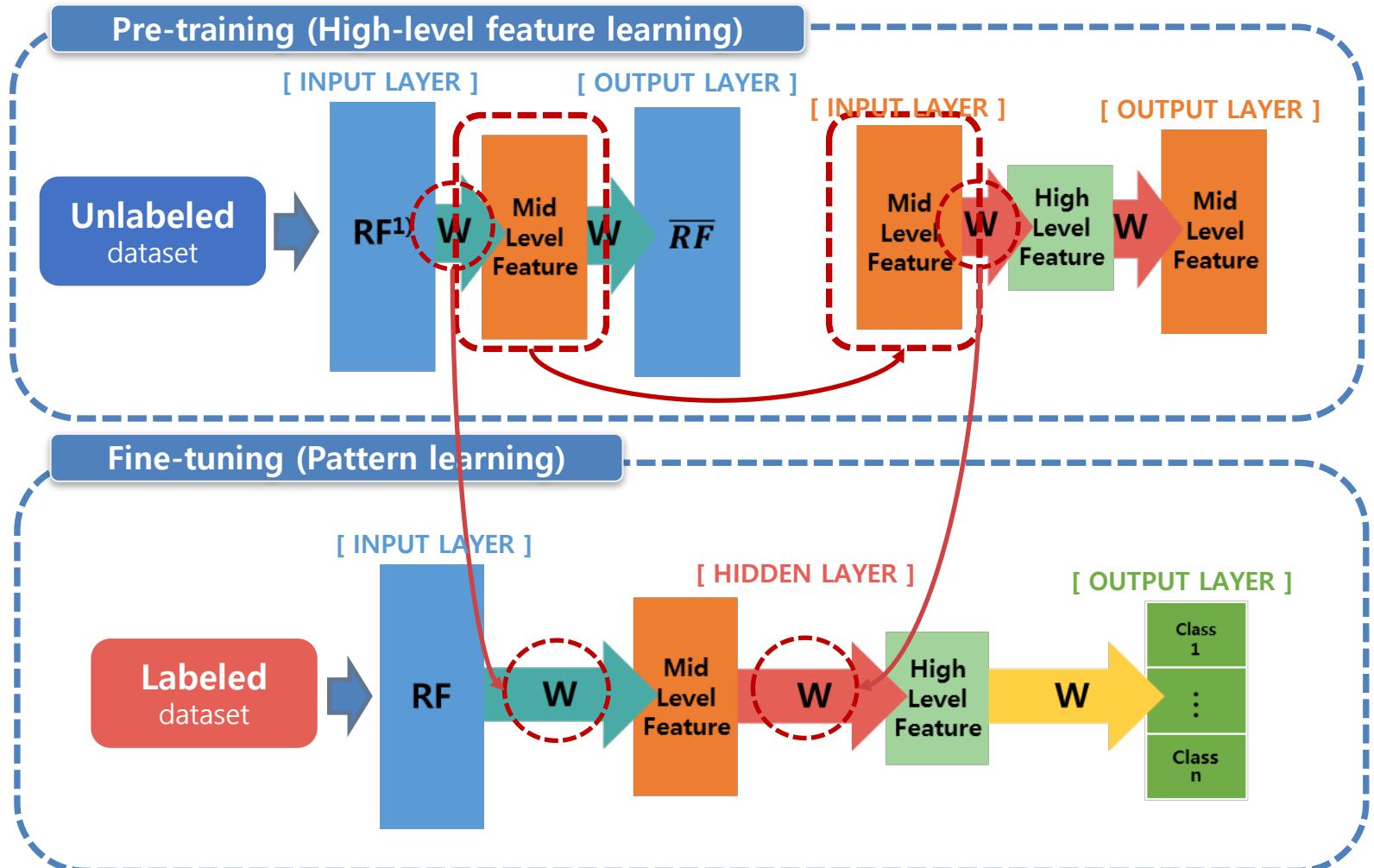


[Deep networks]

DL can learn discriminative features using hierarchical property

Deep Learning Process

- Deep architecture-based modeling



1) RF: Raw Features

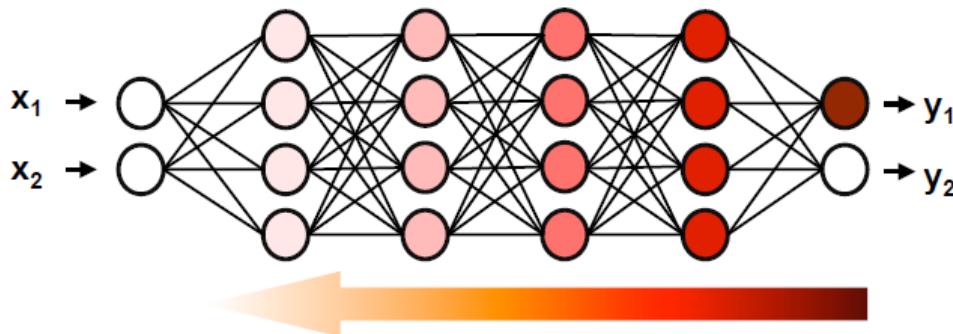
Good performance of Deep learning

- **Solve vanishing gradient problem**
 - Pre-training based on unsupervised learning
 - Rectified Linear Unit (ReLU)
- **Solve overfitting problem**
 - Dropout

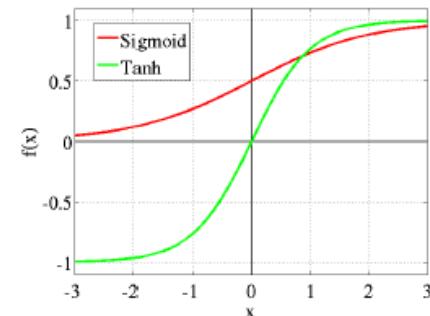
Vanishing gradient problem

- Problems with non-linear activation

- Gradient is progressively getting more dilute
- Below top few layers, correction signal is minimal



Backward error information vanishing

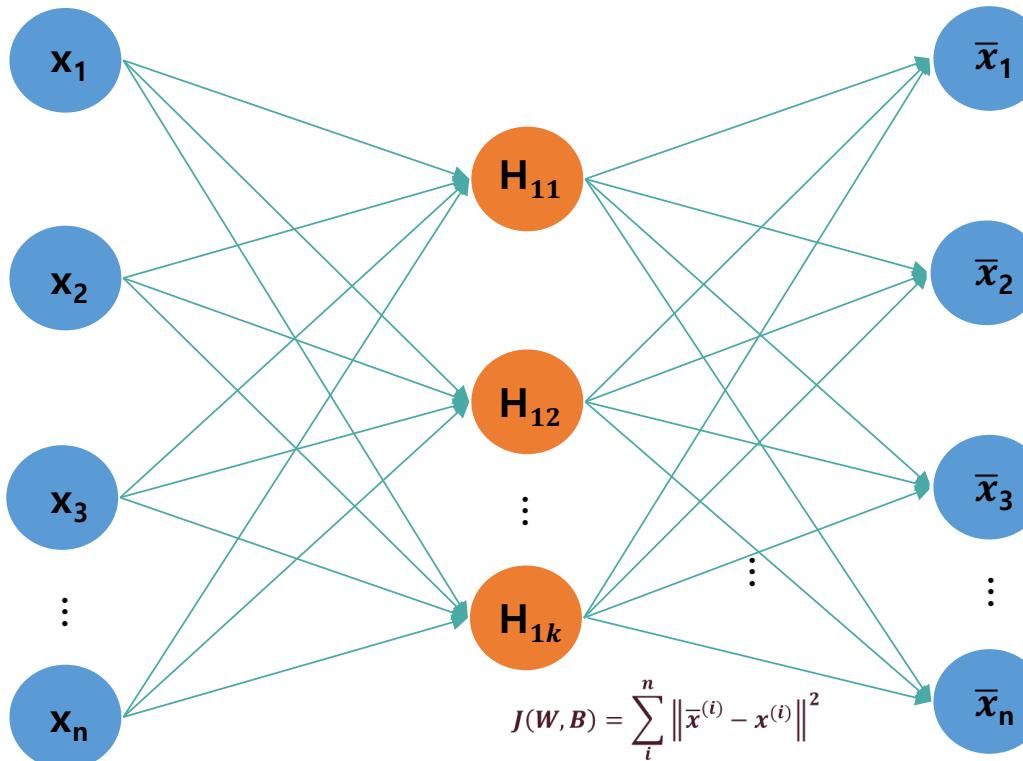


Activation function

$$\frac{1}{1+e^{-x}} \rightarrow \text{sigmoid}$$

Pre-training

- Bottom-up layerwise unsupervised learning
 - Auto Encoder (AE)
 - update weights to minimize values between real and model

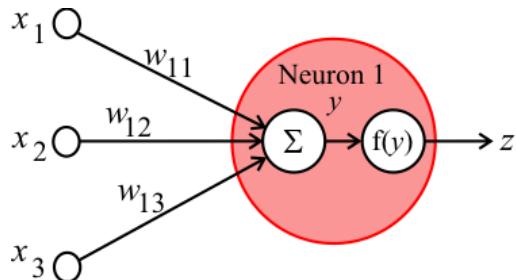


→ Finding weight and bias vectors that minimize reconstruction errors with back-propagation via gradient decent.

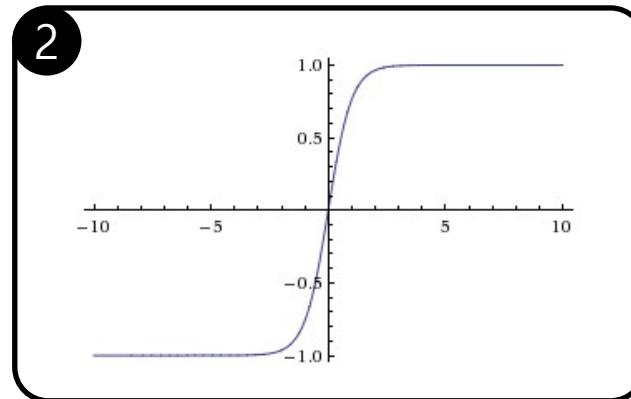
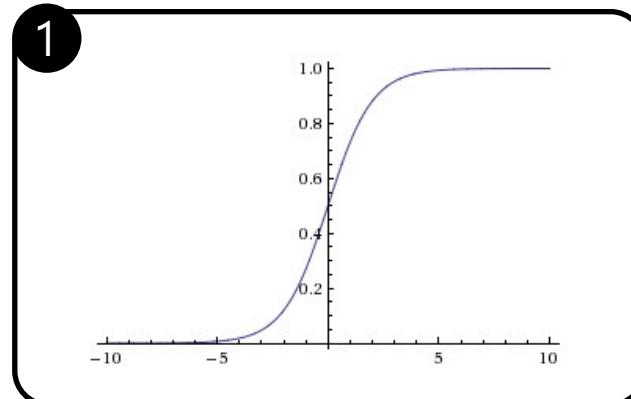
Activation function

- Activation functions

- Takes a single number and performs a certain fixed mathematical operation on it

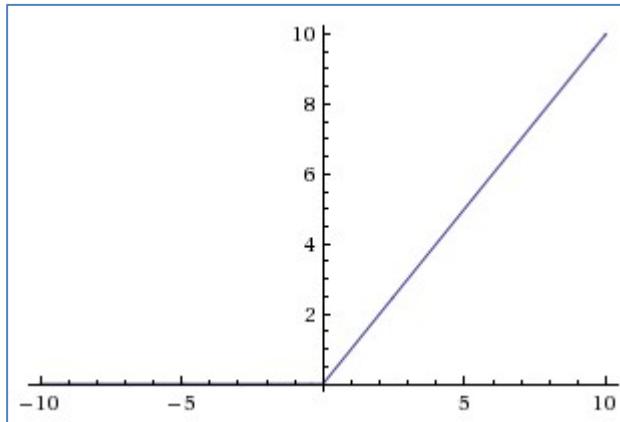


$$1 \quad z = f(y) = \text{logsig}(y) = \frac{1}{1 + e^{-y}}$$
$$2 \quad z = f(y) = \tanh(ay)$$



Rectified Linear Unit (ReLU)

- Defined as



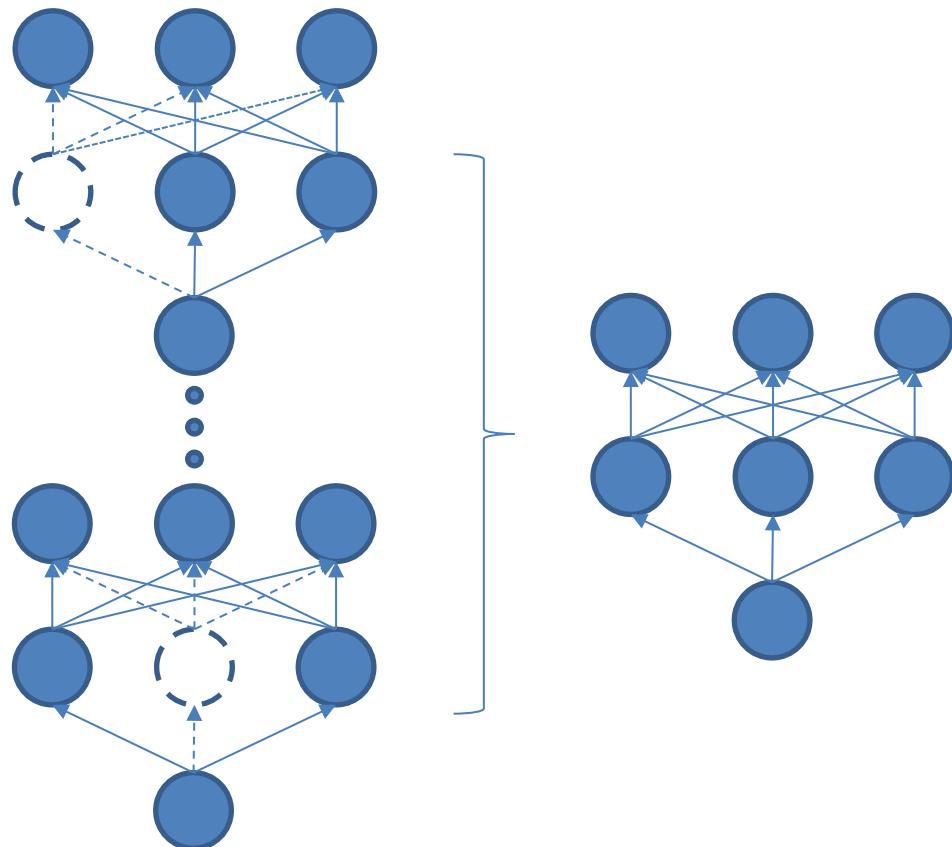
$$f(x) = \max(0, x)$$

- Advantage

- ReLU has a linear slope at positive value
 - ReLU solves gradient vanishing problem

Dropout

- Dropout

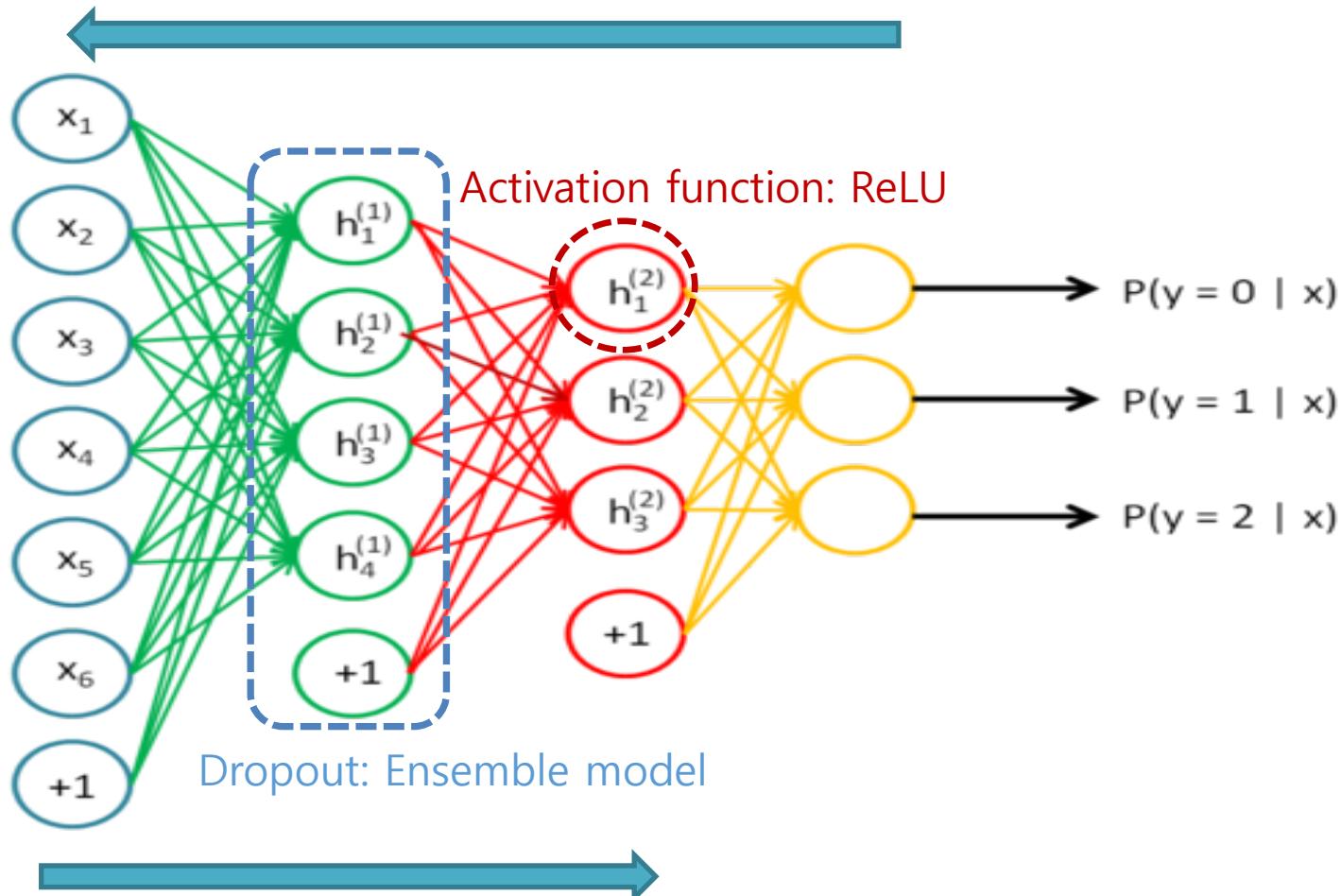


Multi-neural networks

- ✓ Randomly and temporarily omitting hidden units from the network with a probability along with all their incoming and outgoing connections
- ✓ Effecting to average the predictions produced by a very large number of different networks

Summary

Fine-tuning with supervised learning (back-propagation)



Pre-training with unsupervised learning (AE, RBM)

Learning NN

- **Forward propagation**

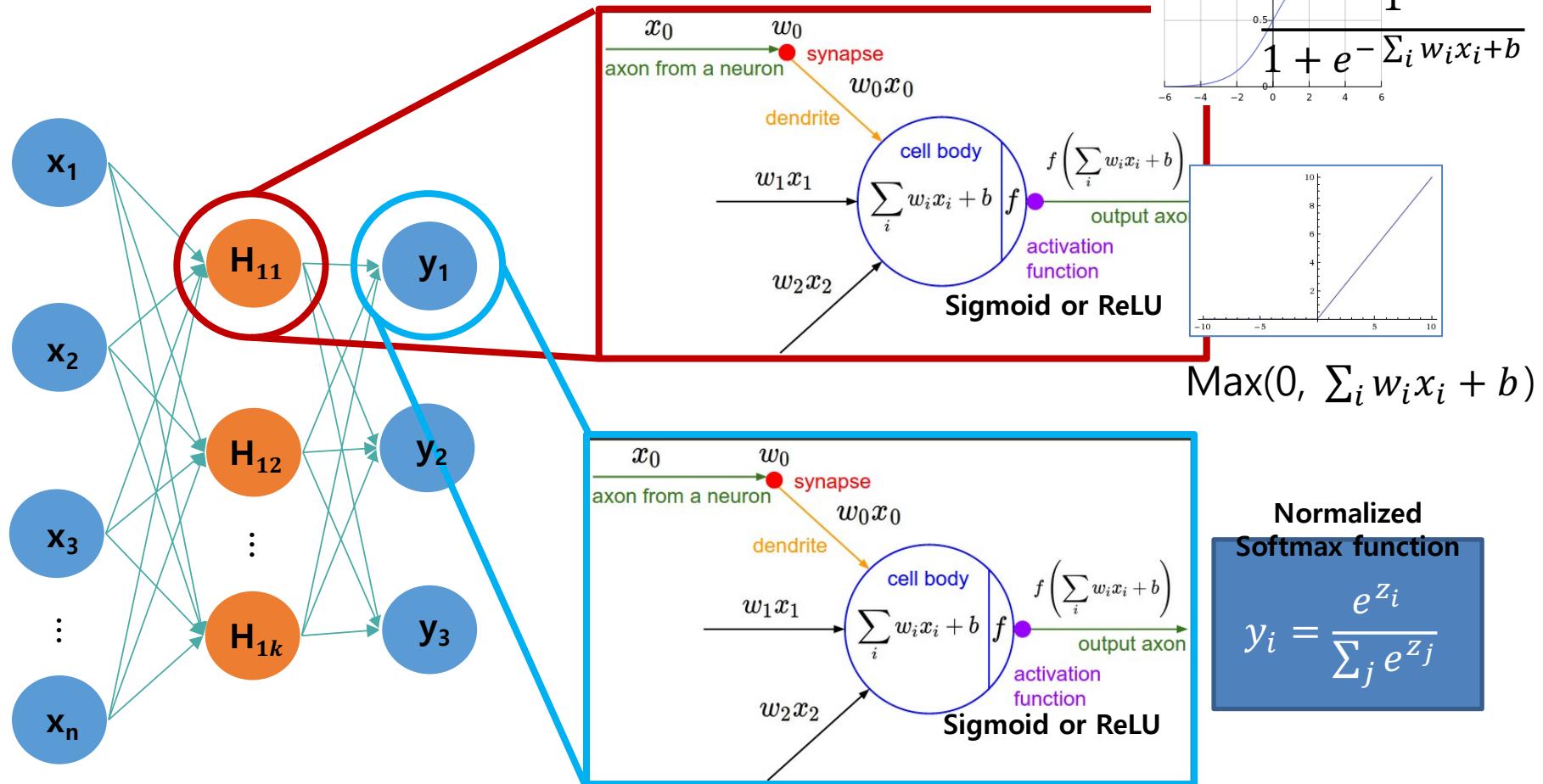
- Calculation of output values (prob. or scalar)
- Classification

- **Backward propagation**

- Weight update

Learning NN-Forward propagation

- Structure of NN



Learning NN-loss (cost) function

- linear regression

$$L = \frac{1}{2}(t - y)^2$$

MSE

- Nominal class

$$L = -(t_i * \ln y_i + (1 - t_i) \ln(1 - y_i))$$

Cross entropy (=negative log likelihood)

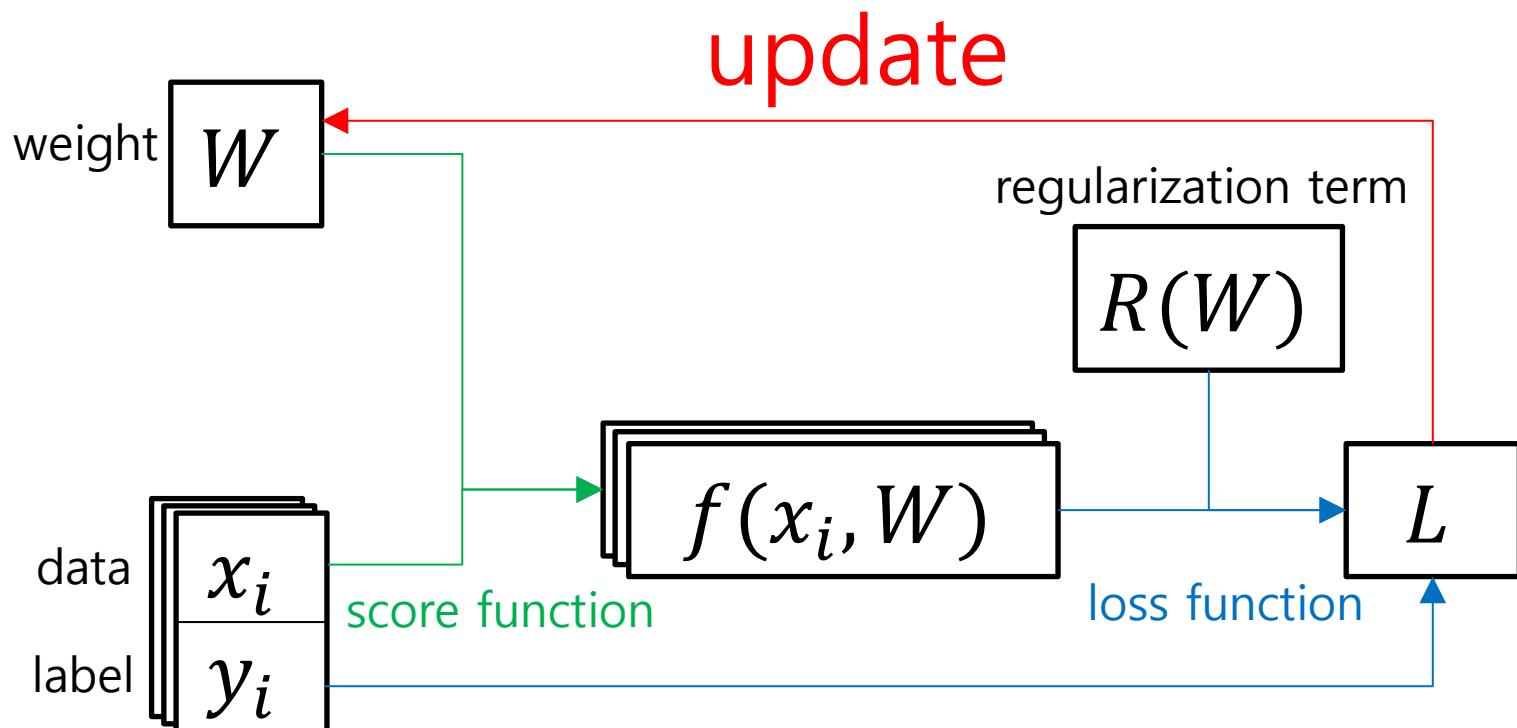
Learning NN-backward propagation

- Weight update

$$\checkmark W_{new} = W_{pre} + \rho \frac{\partial L}{\partial w}$$

Learning NN Summary

- Update W to Normalize L



Dataset

- iris data

The screenshot shows a software interface for managing datasets. At the top, there is a toolbar with a 'File' button, a dropdown menu set to 'File: iris.tab', a 'Reload' button, and a '...' button. Below the toolbar is an 'Info' section containing the following text:

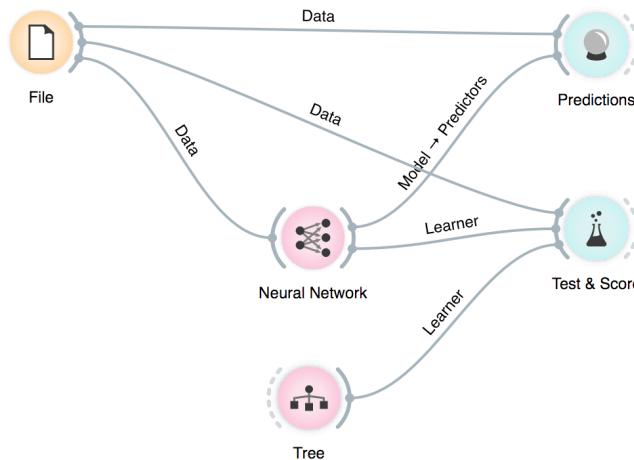
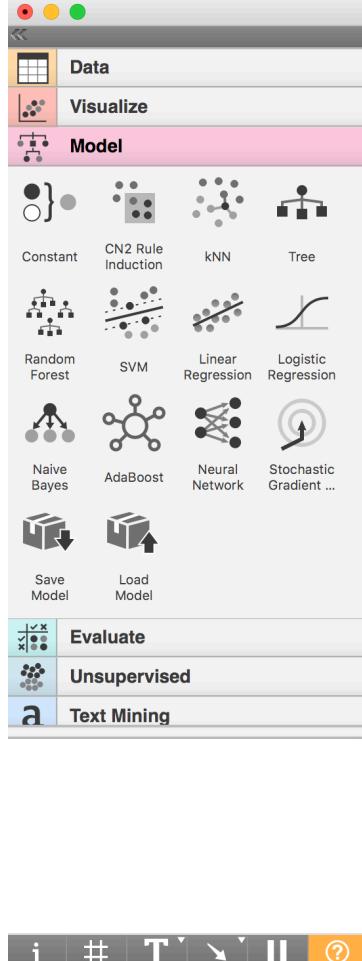
Iris flower dataset
Classical dataset with 150 instances of Iris setosa, Iris virginica and Iris versicolor.
150 instance(s), 4 feature(s), 0 meta attribute(s)
Classification: categorical class with 3 values.

Below the info section is a table titled 'Columns (Double click to edit)'. The table has four columns: Name, Type, Role, and Values. The rows are numbered 1 to 5:

	Name	Type	Role	Values
1	sepal length	N	numeric	feature
2	sepal width	N	numeric	feature
3	petal length	N	numeric	feature
4	petal width	N	numeric	feature
5	Iris	C	categorical	target Iris-setosa, Iris-versicolor, Iris-virginica

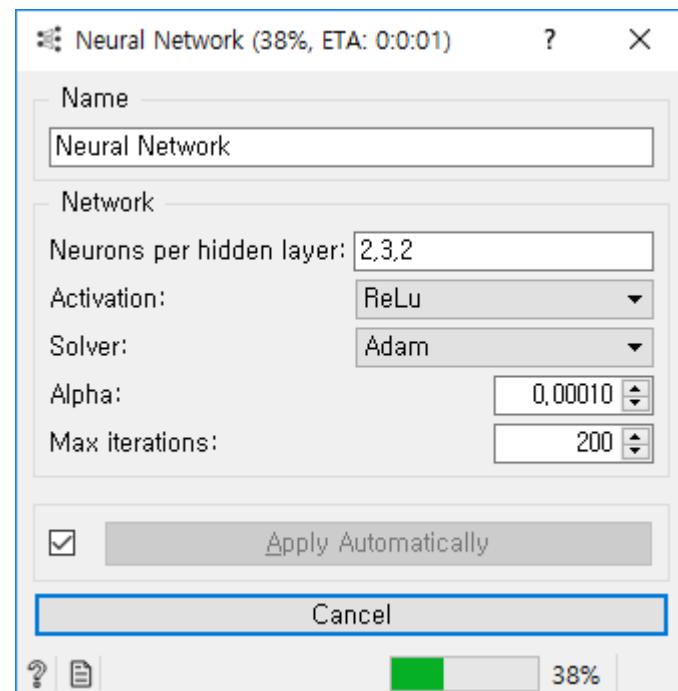
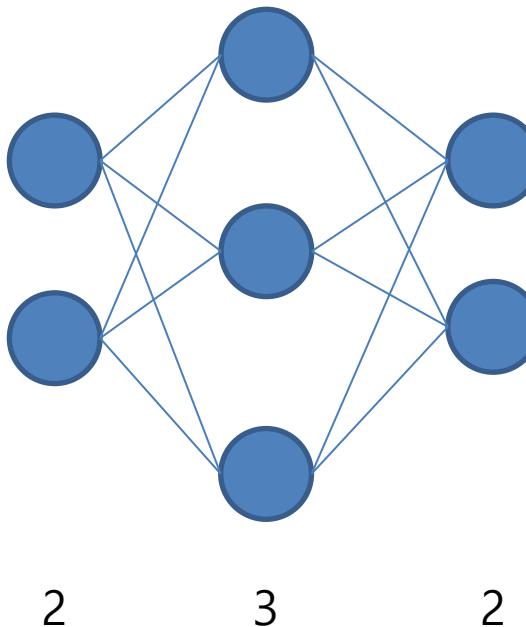
At the bottom of the window are two buttons: 'Browse documentation datasets' and 'Apply'. There are also '?' and '...' buttons at the very bottom left.

Build Schema



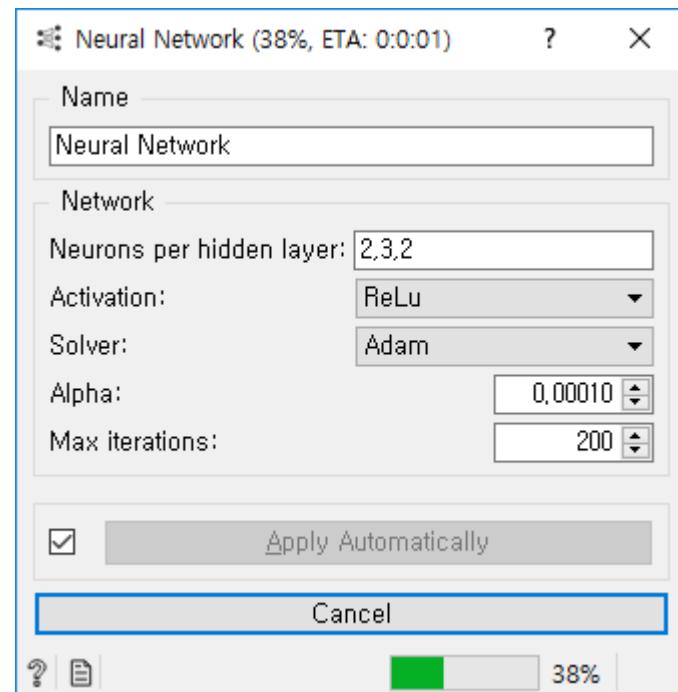
Build NN architecture

- Build NN architecture on Orange3

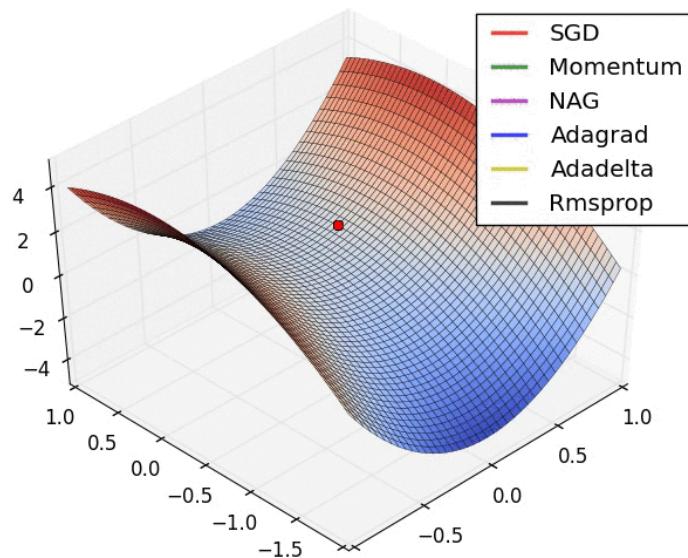


Hyperparameters and configuration

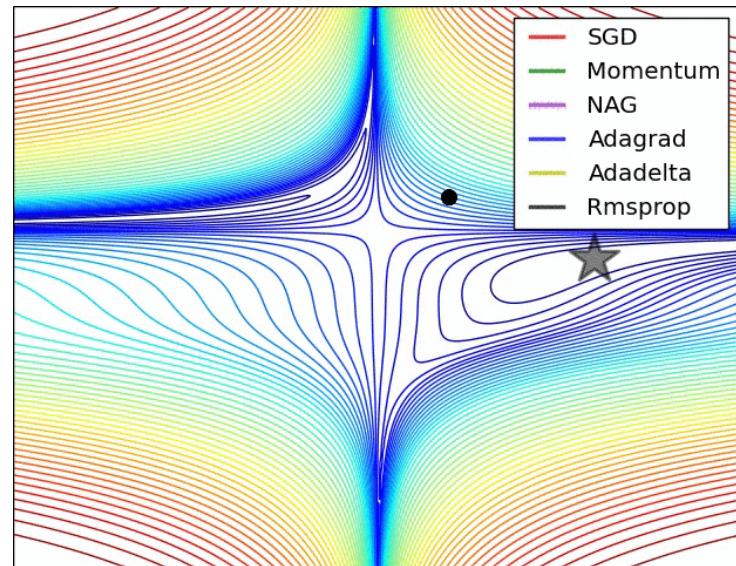
- **Neurons per hidden layer:** defined as the i th element represents the number of neurons in the i th hidden layer. E.g. a neural network with 3 layers can be defined as 2, 3, 2.
- **Activation function for the hidden layer:**
 - Identity: no-op activation, useful to implement linear bottleneck
 - Logistic: the logistic sigmoid function
 - tanh: the hyperbolic tan function
 - ReLu: the rectified linear unit function
- **Solver for weight optimization:**
 - L-BFGS-B: an optimizer in the family of quasi-Newton methods
 - SGD: stochastic gradient descent
 - Adam: stochastic gradient-based optimizer
- **Alpha: L2 penalty (regularization term) parameter**
- **Max iterations: maximum number of iterations**



Weight Optimization



Gradient decent optimization algorithms at log valley



Gradient Descent Optimization Algorithms at Beale's Function

Ref : https://www.reddit.com/r/MachineLearning/comments/2gopfa/visualizing_gradient_optimization_techniques/cklhott/

Observation of NN

The screenshot shows a neural network configuration window and a predictions table.

Neural Network Configuration:

- Name: Neural Network
- Network:
 - Neurons per hidden layer: 100,
 - Activation: ReLu
 - Solver: Adam
 - Alpha: 0.00010
 - Max iterations: 200
- Buttons:
 - Apply Automatically (checked)
 - Cancel

Info Panel:

- 150 instances.
- 1 feature: 1
- Classification
- store Original Order

Predicted class:

- Iris-setosa
- Iris-versicolor
- Iris-virginica

Predicted probabilities for:

- View distribution bars
- View full dataset

predictions

Probabilities

Predictions Table:

	Neural Network	iris	sepal length	sepal width
1	0.99 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	5.1	3.5
2	0.97 : 0.02 : 0.00 → Iris-setosa	Iris-setosa	4.9	3.0
3	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	4.7	3.2
4	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	4.6	3.1
5	1.00 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	5.0	3.6
6	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	5.4	3.9
7	1.00 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	4.6	3.4
8	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	5.0	3.4
9	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	4.4	2.9
10	0.98 : 0.02 : 0.00 → Iris-setosa	Iris-setosa	4.9	3.1
11	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	5.4	3.7
12	0.99 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	4.8	3.4
13	0.98 : 0.02 : 0.00 → Iris-setosa	Iris-setosa	4.8	3.0
14	1.00 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	4.3	3.0
15	0.99 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	5.8	4.0
16	1.00 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	5.7	4.4
17	1.00 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	5.4	3.9
18	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	5.1	3.5
19	0.99 : 0.01 : 0.00 → Iris-setosa	Iris-setosa	5.7	3.8
20	1.00 : 0.00 : 0.00 → Iris-setosa	Iris-setosa	5.1	3.8
21	0.98 : 0.02 : 0.00 → Iris-setosa	Iris-setosa	5.4	3.4

Performance comparison

Test & Score

Sampling

Cross validation
Number of folds: 10

Stratified

Cross validation by feature

Random sampling

Repeat train/test: 10

Training set size: 66 %

Stratified

Leave one out

Test on train data

Test on test data

Target Class

(Average over classes)

Evaluation Results

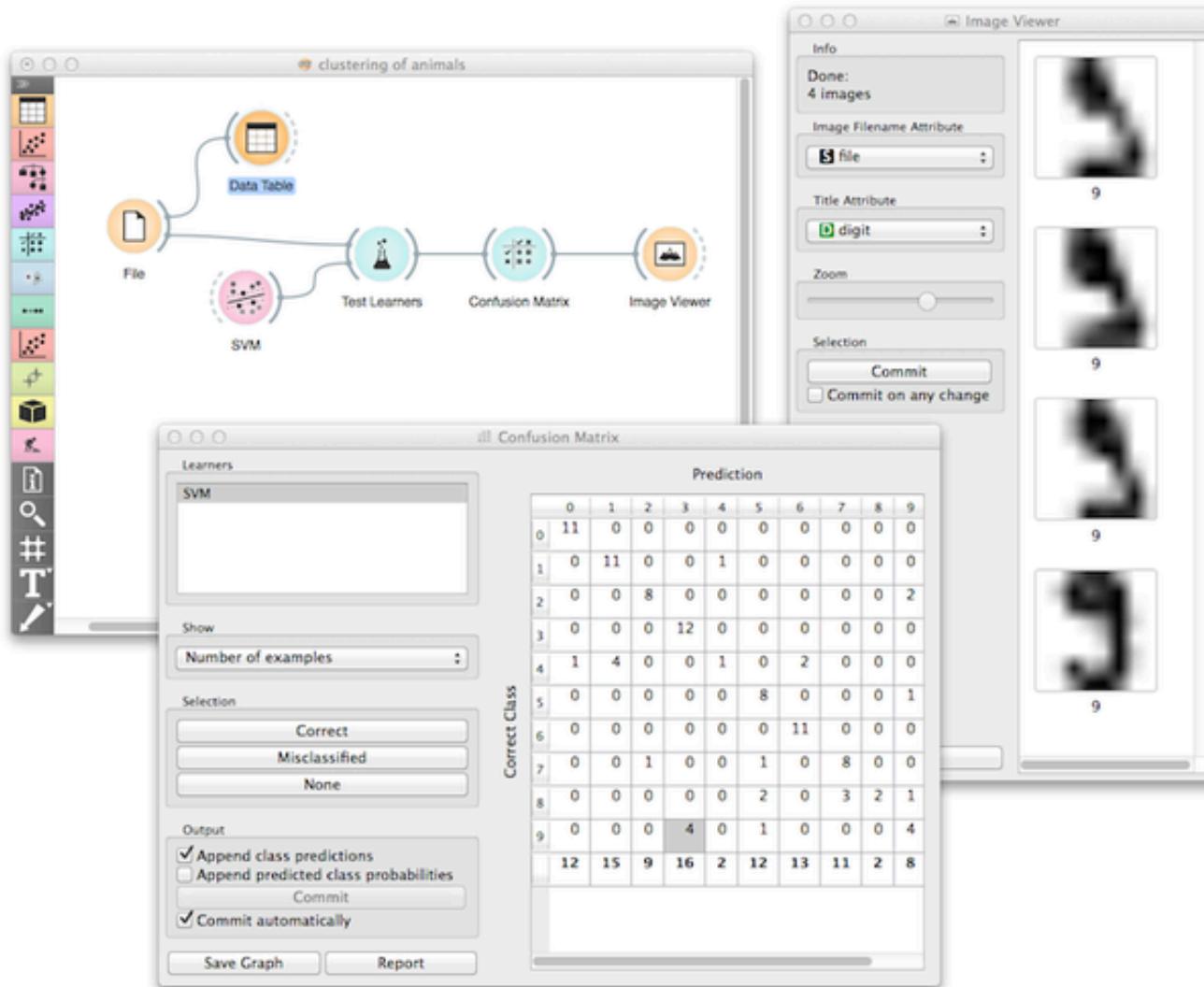
Method	AUC	CA	F1	Precision	Recall
Tree	0.960	0.935	0.936	0.936	0.935
Neural Network	0.993	0.941	0.941	0.942	0.941

Real Dataset

- **Digit data**
 - a famous digit data set
 - <https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits>
 - a data set in the Orange format with the image files (<https://blog.biolab.si/download/868/>)
 - **Self task**
 - to classify handwritten digits based on their bitmap representation



Build Schema

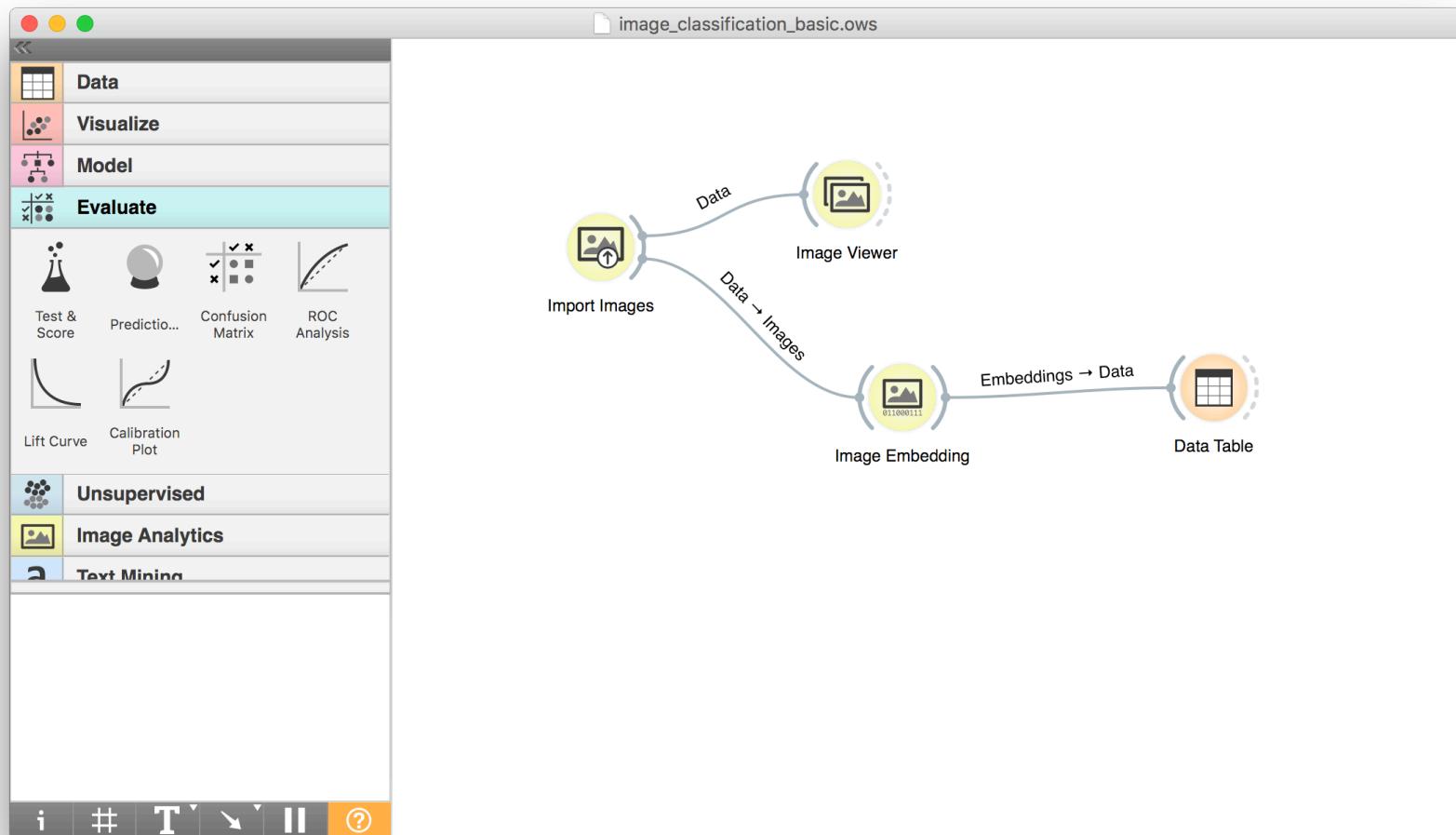


Self Study

- **classify handwritten digits based on their bitmap representation**
 - how do the images of the misclassified digits look like?
 - Improve performance by neural network

Build Schema

- Image classification example



MNIST web demo

- <http://myselph.de/neuralNet.html>

The screenshot shows a web browser window for "myselph.de" displaying a neural network for handwritten digit recognition. The main title is "Neural Net for Handwritten Digit Recognition in JavaScript". Below it, a sub-header reads "Handwritten Digit Recognition by Convolutional Neural Network". A message encourages users to "Draw a digit in the box below and click the 'recognize' button." A drawing area contains a handwritten digit '2'. To its right is the correctly recognized digit '6'. Below the drawing area are two checkboxes: "Display Preprocessing" (unchecked) and "Scale Stroke Width" (checked). There are also "clear" and "recognize" buttons. A detailed description of the neural network architecture follows, mentioning 784 input units, 200 hidden units, and 10 output units. At the bottom right is a grid of handwritten digits from 0 to 9.

myselph.de Rigid Body Game Physics ▾ Hodgkin-Huxley Neural Net for MNIST About me

Neural Net for Handwritten Digit Recognition in JavaScript

Handwritten Digit Recognition by Convolutional Neural Network

Neural Net for Handwritten Digit Recognition in JavaScript

Draw a digit in the box below and click the "recognize" button.

Display Preprocessing

Scale Stroke Width

clear recognize

A Javascript implementation of a neural net for handwritten digit recognition. The network has 784 input units (28 x 28 grayscale image, normalized to values ranging from [-1; 1]). These are fully connected to 200 hidden units, each having a bias parameter, giving $(784 + 1) * 200 = 157.000$ weights; the activations are fed through a logistic non-linearity. The hidden layer is fully connected to the output layer with 10 units, giving $(200 + 1) * 10 = 2010$ weights. The final output is computed with a 10-way softmax non-linearity, assigning class (0 - 9) probabilities to the input image.

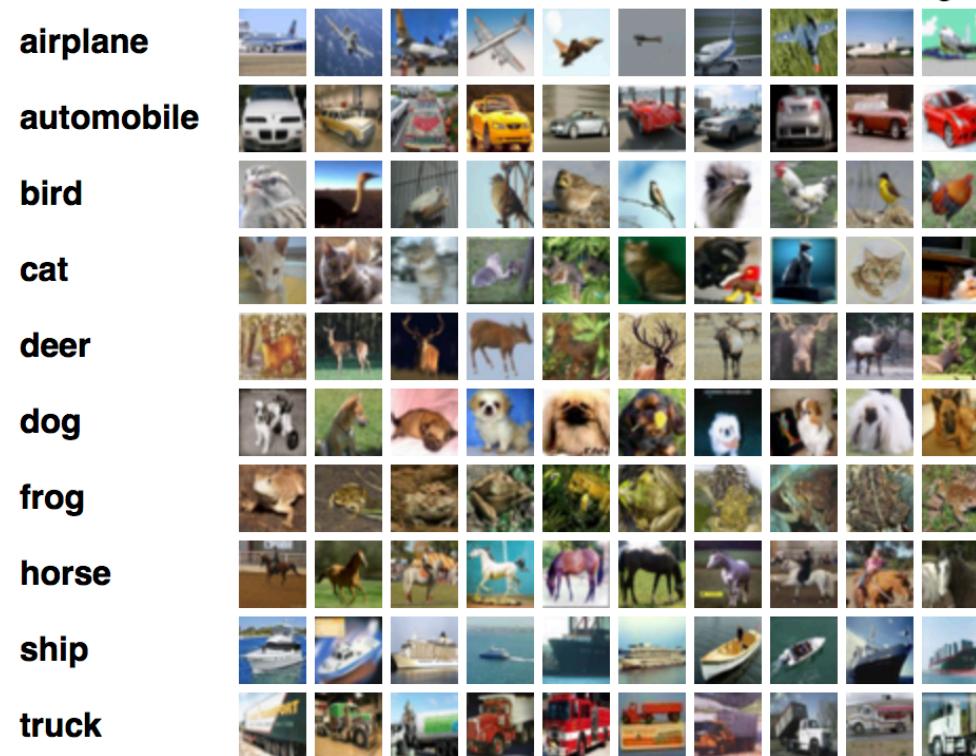
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

CIFAR-10

- **The CIFAR-10 dataset**

- consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Here are the classes in the dataset, as well as 10 random images from each:



Convolution Neural Network Example

- <https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

The screenshot shows a web browser window with the URL <https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html> in the address bar. The page title is "ConvNetJS CIFAR-10 demo".

Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).

Training Stats

pause
Forward time per example: 6ms
Backprop time per example: 10ms
Classification loss: 2.11001
L2 Weight decay loss: 0.00093
Training accuracy: 0.21
Validation accuracy: 0.08
Examples seen: 499
Learning rate: 0.01
Momentum: 0.9
Batch size: 4
Weight decay: 0.0001

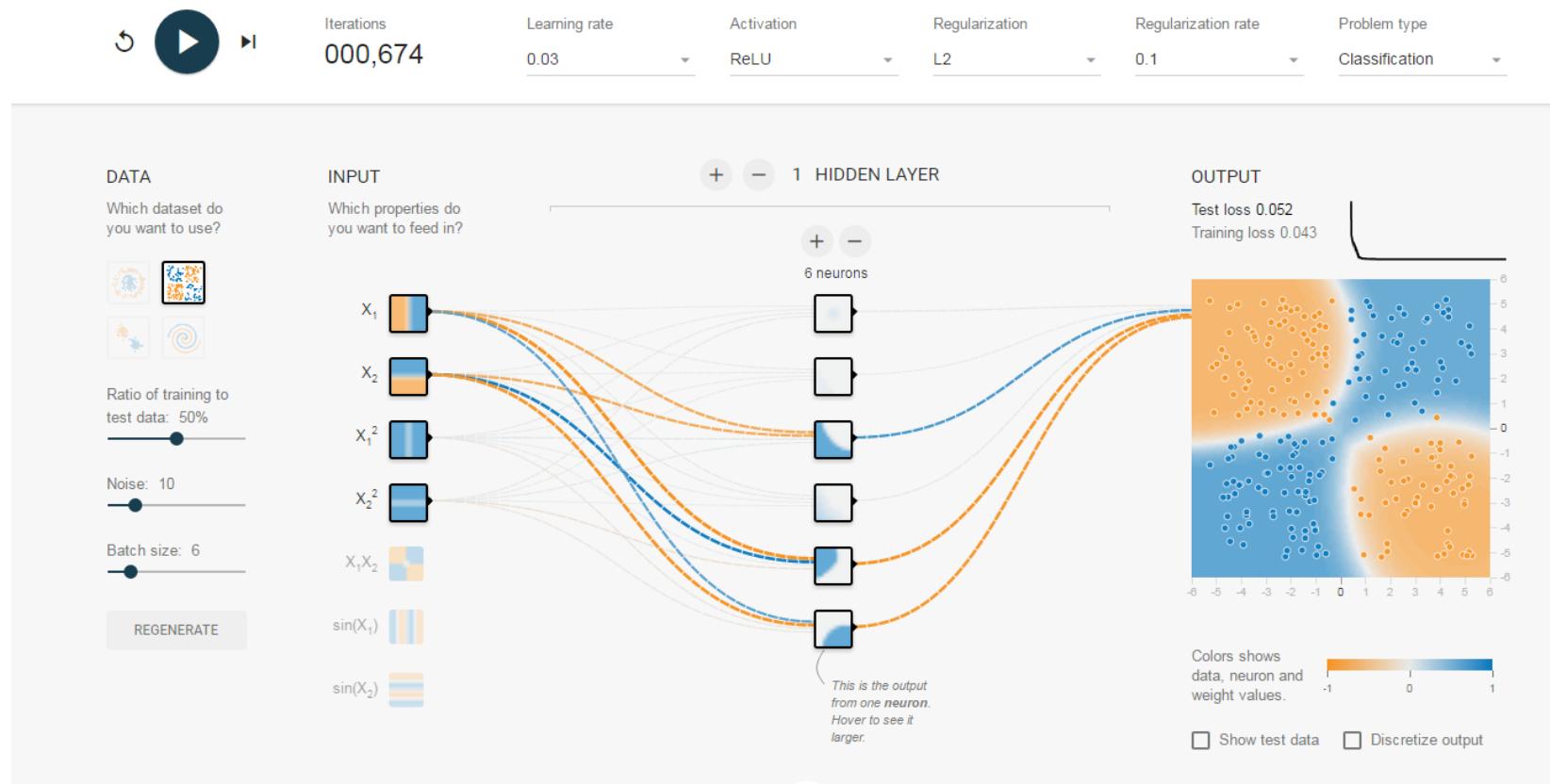
Loss:

2.38
2.34
2.31
2.27
2.23
2.19
2.15
2.11
2.08
2.04
2

0k 0.1k 0.2k 0.3k 0.4k 0.5k 0.6k 0.7k 0.8k 0.9k 1k

NN Playground

- <http://playground.tensorflow.org>



References

- <https://medium.com/machine-learning-101/chapter-3-decision-trees-theory-e7398adac567>
- <https://blog.biolab.si/tag/classification-tree/>
- <https://docs.orange.biolab.si/3/visual-programming/widgets/model/neuralnetwork.html>
- <https://blog.biolab.si/tag/images/>
- <https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regulation-techniques/>
- Stanford CS224d-Lecture6, 2017
- <https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>
- <https://www.cs.toronto.edu/~kriz/cifar.html>

Thank you

