Machine Learning

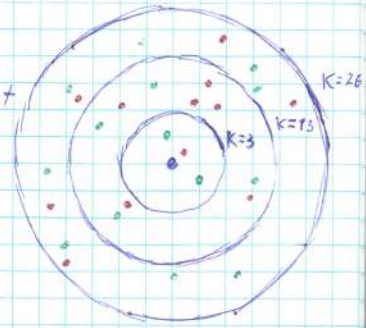
Types of ML Problems:

- **Unsupervised Learning** - the machine did not use marked down data. Could find Hidden patterns.

- **Supervised Learning** - the machine use marked down data as training one. Could predict output due to previous training data expirience.

- **Reinforcement Learning** - the machine use real-time simulation (system) where it have „rewards" and „punishments". Could simulate complicated behaviour and learn By own mistakes

K.N.N. ("K" - Nearest Neighbors)



Idea: Consider input data as point <vector> in n-dimentional space. Then use "K" as Hyperparamiter to find K-Nearest Neighbours. Simply count amount of points of each class among neighbours, so the answer is the greater amount class (for classification). Use mean ($\bar{X}$) of all neighbours (for Regression)

Problems:

- **Curse of Dimensionality** - The more dimentionality is, the greater point Dispercy is → less points would be in "K" area.

- **Feature Scale** - Features have different metrics, so one feature might be much more important then the other: [0...2000](m), [0...2](Km) → 2m ~ 2Km **Wrong!**

Distance Metrics:

Euclidean: $D(\bar{a}, \bar{b}) = \sqrt{\sum_{i=1}^{n} (\bar{b}_i - \bar{a}_i)^2}$

Manhattan: $D(\bar{a}, \bar{b}) = \sum_{i=1}^{n} |\bar{a}_i - \bar{b}_i|$

Minkowski: $D(\bar{a}, \bar{b}) = \left(\sum_{i=1}^{n} |\bar{a}_i - \bar{b}_i|\right)^{\frac{1}{p}}$

Learning

Accuracy Metrics:

It is a performance indecators (functions) that helps to evaluate the correctness of the model:

Classificator:

$$Precision = \frac{\text{correct Pred. (class „n")}}{\text{total (class „n")}}$$

$$Recall = \frac{\text{correct Pred. (class „n")}}{\text{total Pred. (class „n")}}$$

$$Accuracy = \frac{\text{Pred. Correct}}{\text{total pred.}}$$

$$F1 = \frac{2 Precision \cdot Recall}{Precision + Recall}$$

Regressor:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - mean(y))}$$

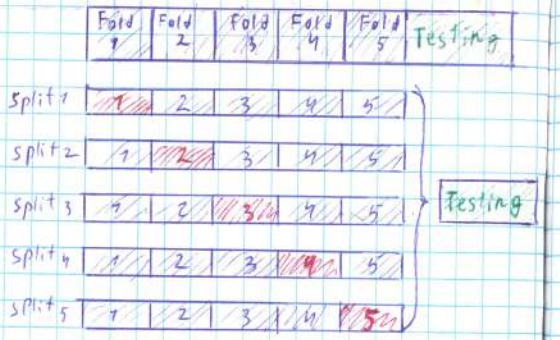
Loss $L(y, \hat{y})$:

It is a function that shows how did prediction „y" differs from correct answer „$\hat{y}$". The lessier it is, the better predicted result you have.

Overfitting:

Situation when model loss on training data $\longrightarrow 0$ and loss on real data $\longrightarrow 100$.

Split Data:



Data Normalization:

ALWAYS Normalize The Features!

$$x_{new} = \frac{x - \mu}{\sigma}$$

μ - mean value of feature,

σ - Standart deviation of feature.

Regularization:

A method of adding special **constraints** to the loss function to **decrease model complexity** and **Prevent overfitting**.

$$L_1 \text{ (Lasso)} = \ldots + \beta \sum_{i=1}^{m} |w_i|$$

$$L_2 \text{ (Ridge)} = \ldots + \beta \sum_{i=1}^{m} w_i^2$$

$$\text{Elastic Net} = \ldots + \beta_1 \sum_{i=1}^{m} |w_i| + \beta_2 \sum_{i}^{m} w_i^2$$

Linear Regression

INPUT:

$$X_{(m \times n)} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1n} \\ 1 & x_{21} & \cdots & x_{2n} \\ 1 & x_{31} & \cdots & x_{3n} \\ 1 & x_{41} & \cdots & x_{4n} \\ \vdots & \vdots & & \vdots \\ 1 & x_{m1} & \cdots & x_{mn} \end{bmatrix} \quad Y_{(m \times 1)} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix}$$

features — correct answers

Model:

$$W_{(K \times 1)} = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \\ \vdots \\ \omega_K \end{bmatrix}$$

weights

Output:

$$\hat{Y}_{(m \times 1)} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_m \end{bmatrix}$$

Prediction result

Train:

$$X \cdot W = \hat{Y}$$

$$L(w) = \frac{1}{m}(XW - y)^T(XW - y) = \frac{1}{m} \cdot \sum_{i=1}^{m} (\hat{y}_i - y)^2$$ # Mean Square Error

ANalitical Method (not used):

The main idea is to minimize loss function „L" by finding such weights „W" to $\frac{\partial L}{\partial W} = 0$

$$\frac{\partial L}{\partial W} = \frac{2}{m} X^T(XW - y) = 0$$

$$W = (X^T X)^{-1} X y \sim X^{+} y$$

Gradient descent:

The same thing but consider derivative Vector as gradient of the Loss function. (Repeat it natively till the acceptable result)

$$\frac{\partial L}{\partial W} \sim grad(L)$$

$$W_{new} = W - \alpha \cdot \frac{\partial L}{\partial W}$$

α - Learning Rate

If dependencies are Non-Linear:

Simply add more Polynomial features and normalize them.

$$X = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad X' = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Logistic Regression

IDEA:

Quiet the same as in Linear Regression, But this one is used in Classification and have belonging class probability as an output. Also, for each class there is own Weights Vector.

$$X_{(m \times n)} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1n} \\ 1 & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ 1 & x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

individuals — features

$$W_{(K \times L)} = \begin{bmatrix} \omega_{00} & \omega_{01} & \cdots & \omega_{0L} \\ \omega_{10} & \omega_{11} & \cdots & \omega_{1L} \\ \vdots & \vdots & & \vdots \\ \omega_{K0} & \omega_{K1} & \cdots & \omega_{KL} \end{bmatrix}$$

for each feature — for each Class

$$\hat{Y}_{(L \times 1)} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_L \end{bmatrix}$$

Probab. of Belo... to each clo...

Train:

$$logits(X_{individ}) = X_{individ} \cdot W \sim [1\, x_1\, x_2 \cdots x_n] \cdot \begin{bmatrix} \omega_{00} & \cdots & \omega_{0L} \\ \vdots & & \vdots \\ \omega_{K0} & \cdots & \omega_{KL} \end{bmatrix}$$

$$Softmax(logits(X_{individ})) = \hat{y}$$

The most possible class type is one with the highest probability

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \rightarrow \text{SoftMax} \left(\frac{e^{z_i}}{\sum_{i=1}^{K} e^{z_i}} \right) \rightarrow \begin{bmatrix} 0.02 \\ 0.9 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Loss:

$$L(w) = -\frac{1}{m} \sum_{i=1}^{m} \begin{cases} \log(1 - \hat{y}_i) & \text{if } y_i = 0 \\ \log(\hat{y}_i) & \text{if } y_i = 1 \end{cases}$$

$$L(W) = -\frac{1}{m} \sum_{i=1}^{m} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right)$$

Log Loss - return huge Value if Prediction $\hat{y}_i$ compleatl... dismatch the correct class answer „y_i"

Gradient descent:

$$\frac{\partial L}{\partial W} = \frac{1}{n} \cdot X_{individ}^T (\hat{y} - y)$$

Decision Trees.

Idea:

it is a logical algorithm that based on a binary tree. Each **node** contains a **question** (condition)

connections - all possible **results** of a question,

leafs - terminal node **ready** to predict.

After training (tree is complete) we could predict the result by simple tree traverse.

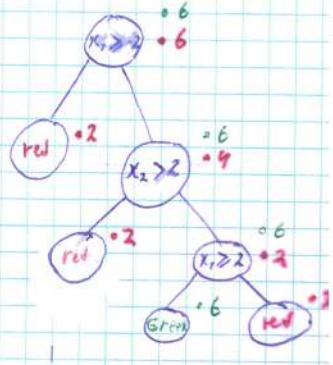
Training:

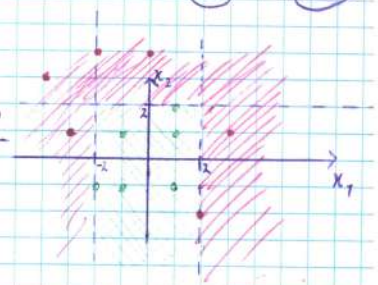
- Check all the leafs and calculate the **Entropy** or **Gini Index** for every possible combination of every feature value split ($x_1 \leqslant 1, x_2 \geqslant 3 \ldots$)
- Calculate **Information Gain** for each possible split By:

$$IG = entropy(parent) - \sum_{i=0} \omega_i \cdot entropy(child_i)$$

where "ω_i" - $\frac{size(child)}{size(parent)}$

- Choose split with the highest IG value and repeat until "max_tree_depth" or every final node become **leaf**.





$$entropy(X) = -\sum_{i=1} P(x_i) \cdot \log_2 P(x_i)$$

where "P()" - probability of

$P_{(class)} = \frac{\text{Total objects}}{\text{class amount}}$ a class.

$$gini\_index(X) = 1 - \sum_{i=1} P(x_i)^2$$

Ensembles.

Voting:

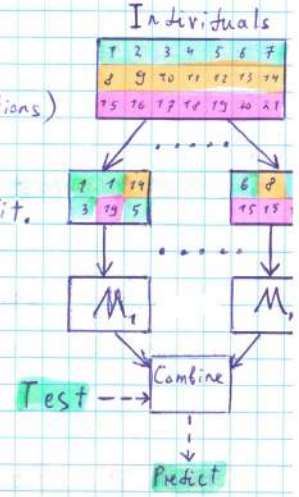
Use different algorithms with Prediction probabili
- greater than $\frac{1}{2}$ (50%) and simply get the mean value of all the results for **Regression.**

For Classification problem the idea is the same, bu we choose the most frequent class.

Bagging (parallel):

Firstly, get some random samples (with repetitions) and train some models to get "Weak learners" so they would not overfit.

Then, pass the test data through these models and mean the result.



Random Forest:

This is a variation of **Bagging** technique where we randomly sample not only **individuals** but their **features** too.

Also it is used **Decision Tree** as a model.

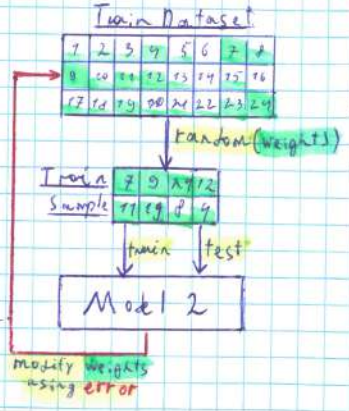
Could learn very deep and hard dependencies.

Boosting (sequential):

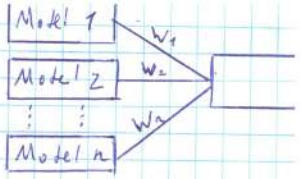
It is a process that uses a set of models (**Weak Learners**) combined into single **Strong Learner** model in order to increase the accuracy.

ADaptive Boosting (AdaBoost):

- Firstly add „$\frac{1}{n}$" weight to each individ (or sample) in Training Dataset.
- Randomly (depenting on weights) choose data to create training sample and train clear model (weak Lerner).
- Test this model on the same sample and mark up individuals were predicted with error.



- Increase weight at those individuals in Training dataset.
- Repeat this steps and combine all weak lerners into one strong learner using weights, which are depends on weak learner accuracy.

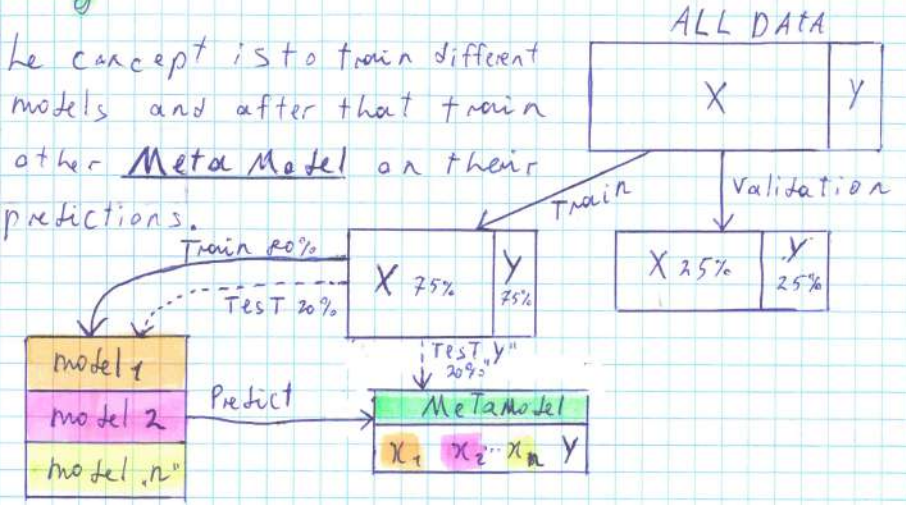


$$w_i = \frac{1}{2} \ln\left(\frac{1-error_i}{error_i}\right)$$

Gradient Boosting:

Blending:

The concept is to train different models and after that train other **Meta Model** on their predictions.

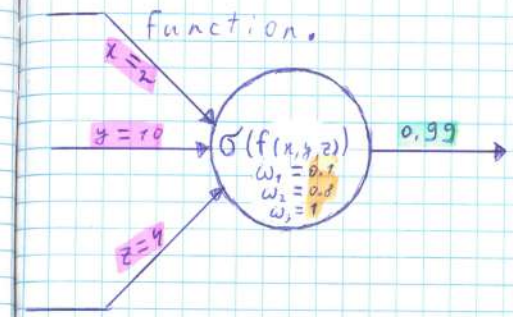


Stacking:

DEEP LEARNING

Neuron:

The main structure element of ANN's. The main idea behind it is to **linearly transform** some input factor (constant, vector, matrix, tensor...) using neuron function and then apply **non-linear activation** function.



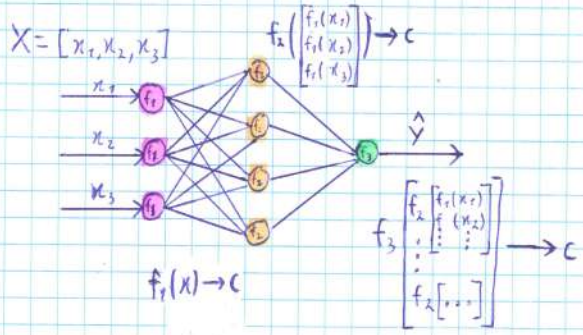
$$f(x,y,z) = \omega_1 \cdot x + \omega_2 \cdot \frac{y}{2} + \omega_3 \cdot z$$

$$\sigma(x) = Sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$\sigma(f(2,10,4)) = Sigmoid(0.1 \cdot 2 + 0.8 \cdot \frac{10}{2} + 1 \cdot 4) =$$
$$= Sigmoid(8.2) = 0.99$$

Layer:

The way to group neurons of same type into one object.



- Input layer
- Output layer
- other

Idea:

The goal of any activation function is to prevent collapse of all linear function into one. Mostly, it also normalize data and should be differentiable.

Consider sequential linear functions without activation

$$\underbrace{\omega_1(K_1x+b_1)}_{f_1(x)} \rightarrow \underbrace{\omega_2(K_2x+b_2)}_{f_2(x)} \rightarrow \underbrace{\omega_3(K_3x+b_3)}_{f_3(x)}$$

$$f_3(f_2(f_1(x))) = \omega_3(K_3(\omega_2(K_2(\omega_1(K_1x+b_1))+b_2))+b_3) \sim \boxed{\omega(Kx+b)}$$

could be simplified in to this expression (linear)

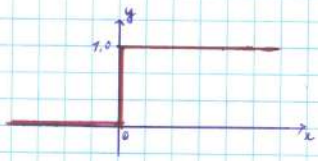
Same but using activation:

$\sigma f_3(\sigma f_2(\dots)) = \sigma(\omega_3(K_3(\sigma\omega_2\dots.+b_3)) \rightarrow$ could not be simplified because „σ"-function!

Binary Step:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

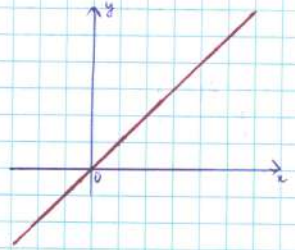
$$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{none} & \text{if } x = 0 \end{cases}$$



Linear:

$$f(x) = x$$

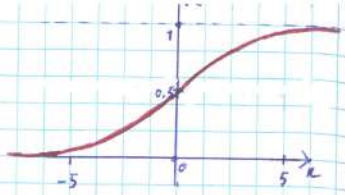
$$f'(x) = 1$$



Sigmoid (Logistic):

$$f(x) = \frac{1}{1+e^{-x}}$$

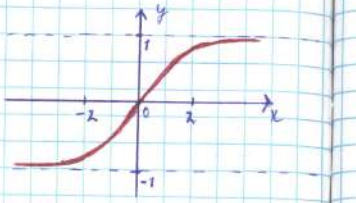
$$f'(x) = f(x) \cdot (1 - f(x))$$



Tanh:

$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

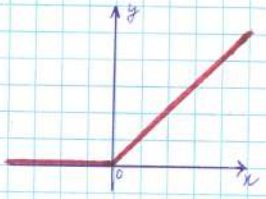
$$f'(x) = 1 - f(x)^2$$



ReLU (Reflected Linear Unit):

$$f(x) = \max(0; x) \sim \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

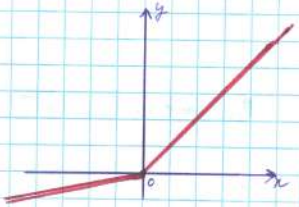
$$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



Leaky ReLU:

$$f(x) = \max(\alpha x; x) \sim \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

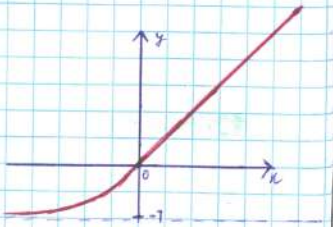
$$f'(x) = \begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



ELU (Exponential Linear Unit):

$$f(x) = \begin{cases} \alpha\, e^x - 1 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

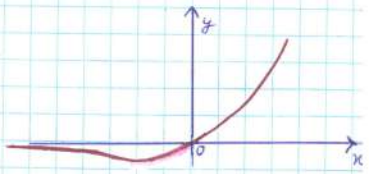
$$f'(x) = \begin{cases} f(x) + \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



Swish:

$$f(x) = x \cdot \text{sigmoid}(x) \sim \frac{x}{1+e^{-x}}$$

$$f'(x) = f(x) + \text{sigmoid}(x) \cdot (1 - f(x))$$

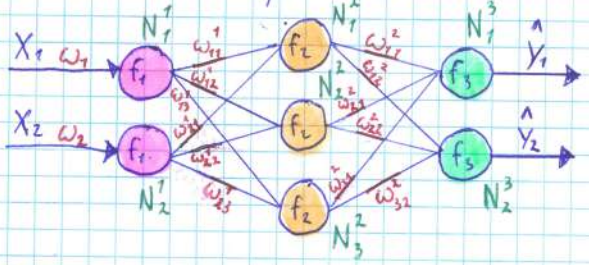


Idea:

The main idea is to update all the parameters (weights) using gradient descient to minimize Loss function.

Example:

Consider Simple Neural Network:



$f_1(x) = \tanh(\omega x)$

$f_2(x_1, x_2) = \sigma(\omega_1 x_1 + \omega_2 x_2)$

$f_3(x_1, x_2, x_3) = 1 \cdot (\omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3)$

Forward Pass:

$\hat{y} = f_3(f_2[f_1[x], \ldots], \ldots, \ldots)$

$\hat{y} = 1 \cdot (\omega_1^2(\sigma[\omega_1^1[\tanh(\omega_1 x)] + \omega_2^1 \ldots]) + \omega_2^2 \ldots + \omega_3^2 \ldots)$

Backward Pass:

Consider Loss function as MSE

$$L = \frac{1}{m}\sum_{i=1}^{m}(y - \hat{y})^2$$

Use LocalGradients to find Weight correction factors and further LocalGradients for each Neuron and it's connection.

To simplify, our goal is to find every partial derivative By each model parameter (weight).

for f_3:

$$\frac{\partial L}{\partial \omega_{11}^2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial f_3} \cdot \frac{\partial f_3}{\partial \omega_{11}^2} =$$

$$= \underbrace{-\frac{2}{m}(y - \hat{y})}_{L'} \cdot \underbrace{1}_{f_3'} \cdot \underbrace{f_2(f_1[x], \ldots)}_{\text{unweight previous result from } \omega_{11}^2}$$

(Loss) (Activation)

For output Layer:

$\text{LocalGrad}(N_m^{n_m}) = L' \cdot f_n'$

$\omega_{new} = \omega_{old} - \alpha \cdot \text{LocalGrad}(N_m^{n_m}) \cdot res(\omega$

for f_2:

$$\frac{\partial L}{\partial \omega_{11}^1} = \sum_{i=1}\left(\omega_i \cdot \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial f_{2i}}\right) \cdot f_2' \cdot \frac{\partial f_2}{\partial \omega_{11}^1} =$$

$$= \underbrace{\sum_{i=1}(\omega_i \cdot \text{LocalGrad}(N_i^2))}_{\text{sum of weighted previous Local gradients,}} \cdot \underbrace{\sigma(\ldots)(1 - \sigma(\ldots))}_{f_2' \text{ (Activation)}} \cdot \underbrace{f_1(x)}_{\text{unweight prev. result from } \omega_{11}^1}$$

For Hidden Layers:

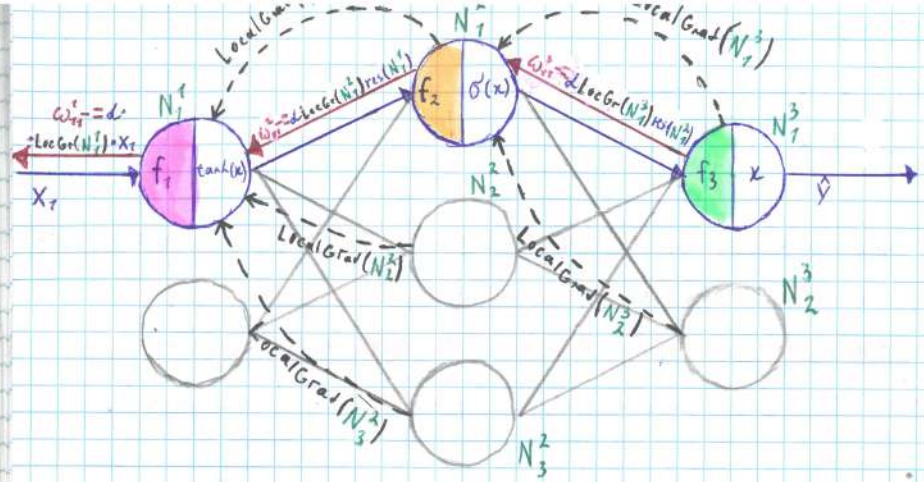
$\text{LocalGrad}(N_m^{n-1}) = \sum_{i=1}(\omega_i \cdot \text{PrevLocalGrad}(N_i^n)) \cdot f_{n-1}'$

for f_1:

$$\frac{\partial L}{\partial x_1} = \sum_{i=1}\left(\omega_i \cdot \sum_{k=1}\left(\omega_k \cdot \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial f_{2_k}}\right) \cdot f_{2_i}'\right) \cdot f_1' \cdot \frac{\partial f}{\partial \omega_1} =$$

$$= \underbrace{\sum_{i=1}(\omega_i \cdot \text{localGrad}(N_i^2))}_{\text{sum of weighted previous Local Gradients.}} \cdot \underbrace{1 - \tanh^2(\ldots)}_{f_1' \text{ (activation)}} \cdot \underbrace{x_1}_{\text{unweight prev. res from } \omega_1}$$

– Global Gradient


N_1^1
f_1
tanh(x)
X_1
N_1^2
f_2
$\sigma(x)$
N_2^2
N_3^2
N_1^3
f_3
x
N_2^3
$\hat{y}$
Local Grad (N_1^2)
Local Grad (N_2^2)
Local Grad (N_3^2)
Local Grad (N_1^3)
Local Grad (N_2^3)

CNN

Convolutional Neural Networks

Idea:

It is a NN that contains special **convolution Layers**.
Mostly it is used for processing images.

Convolution2D:

An operation that used to extract some locational information into **Feature maps** using **Kernels (filters)**

Kernel: square matrix nxn that represent a **filter** to „Detect" some patterns in image. Mostly, it consists of **parameters (weights)** which automatically change in train loop.

Convolution:

Stride = 2 ; Kernel = $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ 3x3

Image 7x7

$\begin{bmatrix} 1 & 9 & 8 & 4 & 4 & 6 & 7 \\ 4 & 8 & 6 & 7 & 5 & 1 & 7 \\ 4 & 0 & 5 & 9 & 3 & 8 & 4 \\ 7 & 3 & 6 & 9 & 0 & 5 & 4 \\ 7 & 4 & 1 & 1 & 8 & 1 & 2 \\ 7 & 6 & 6 & 9 & 8 & 7 & 6 \\ 3 & 6 & 3 & 5 & 4 & 2 & 7 \end{bmatrix}$

$Out_{11} = Sum\left(\begin{bmatrix} 1 & 9 & 8 \\ 4 & 8 & 6 \\ 4 & 0 & 5 \end{bmatrix} \odot \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}\right)$

$Out_{12} = Sum\left(\begin{bmatrix} 8 & 4 & 4 \\ 6 & 7 & 5 \\ 5 & 9 & 3 \end{bmatrix} \odot \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}\right)$

Output

$\begin{bmatrix} 21 & 7 & -24 \\ -7 & 30 & 12 \\ 7 & 25 & 18 \end{bmatrix}$

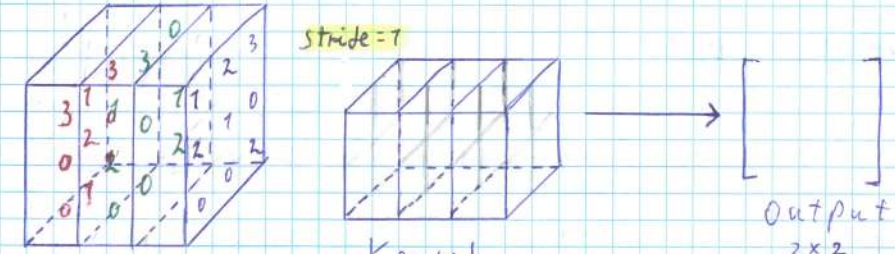
Stride: amount of pixels on which kernel would be shifted during Convolution.

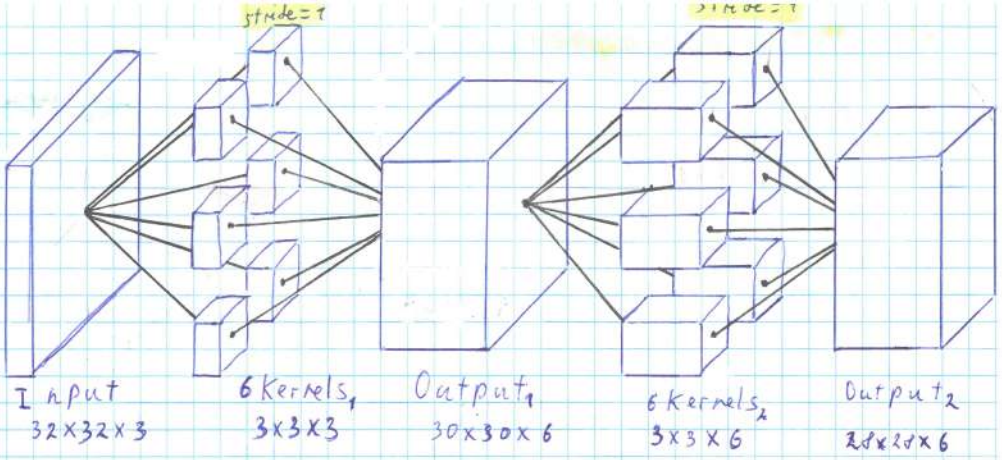
Padding: add frame of mean value of pixels to the image. Increases output image size after convolution.

Feature Map: Simply shows if Kernel was activated (when Feature Map numbers are high with Abs()) or not (when F.M. Abs(nums) are low). The one represents if there is a Kernel Pattern on the image.

Convolution 3D:

This one is the same as 2D but the **Kernel and input** are now **Tensors** and **feature map** is still **matrix**.





Pooling:

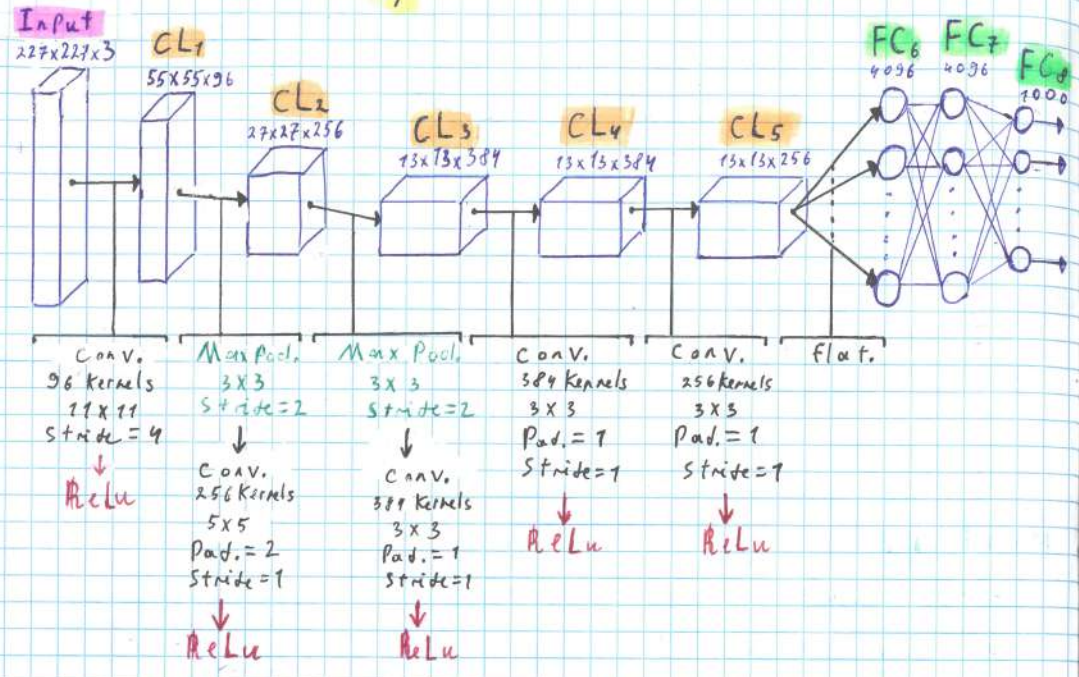
A technique of **size reduction** of feature map By splitting matrix into pieces and getting „Max" or „Average" or „Sum" value. Used to **decrease sensitivity** of next layers. Helps to recognize objects in different positions and sizes.

$$\begin{bmatrix} 1 & 7 & 2 & 6 \\ 2 & 4 & 0 & 2 \\ 1 & 4 & 0 & 8 \\ 0 & 0 & 4 & 9 \end{bmatrix} \xrightarrow{\text{Max Pooling}} \begin{bmatrix} 7 & 6 \\ 4 & 9 \end{bmatrix}$$

Multiple convolutions:

Consider convalutional layer as multiple kernels applied to single input and then stacking the results into one tensor. So this tensor could be the input to other convolutional layer with kernels of depth = stacked results.

Alex Net as example :



Vanishing Gradient Problem:

Consider NN have huge amount of layers („Deep") and it uses Back propagation to train. So when we would calculate weights:

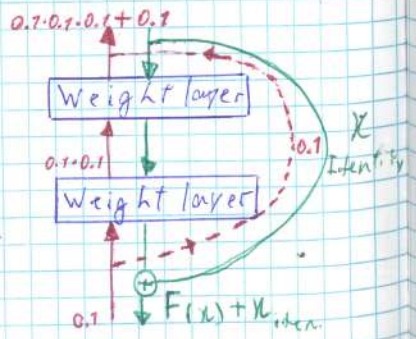
$$\omega_0 = \omega_0 - \alpha \frac{\partial L}{\partial \omega_0} \quad \dots \quad \omega_n = \omega_n - \alpha \frac{\partial L}{\partial \omega_n}$$

$$\frac{\partial L}{\partial \omega_0} = \underbrace{\frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial} \cdot \dots \cdot \frac{\partial}{\partial \omega_0}}_{n \text{ multipliers less then } 1.0} \qquad \frac{\partial L}{\partial \omega_n} = \underbrace{\frac{\partial L}{\partial f_n} \cdot \frac{\partial f_n}{\partial \omega_n}}_{2 \text{ multipliers less then } 1.0}$$

So $\boxed{\alpha \frac{\partial L}{\partial \omega_0} \to 0}$, That means that weight „ω_0" will not change so much. So the first layers would not learn.

Skip Connection

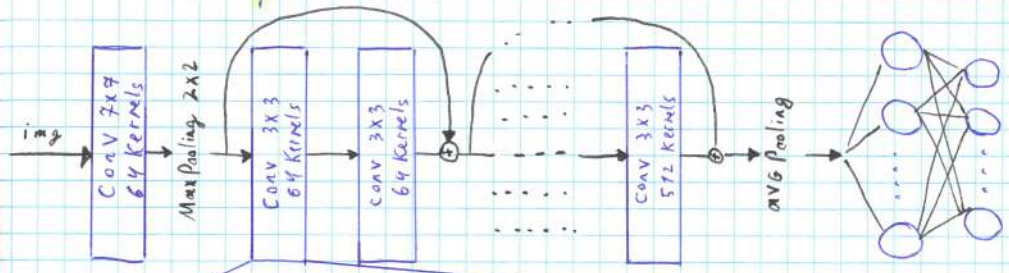
In this method we simply connect and push signal from upper layers to lower ones.
Also It pushes high level information to Deeper layers.

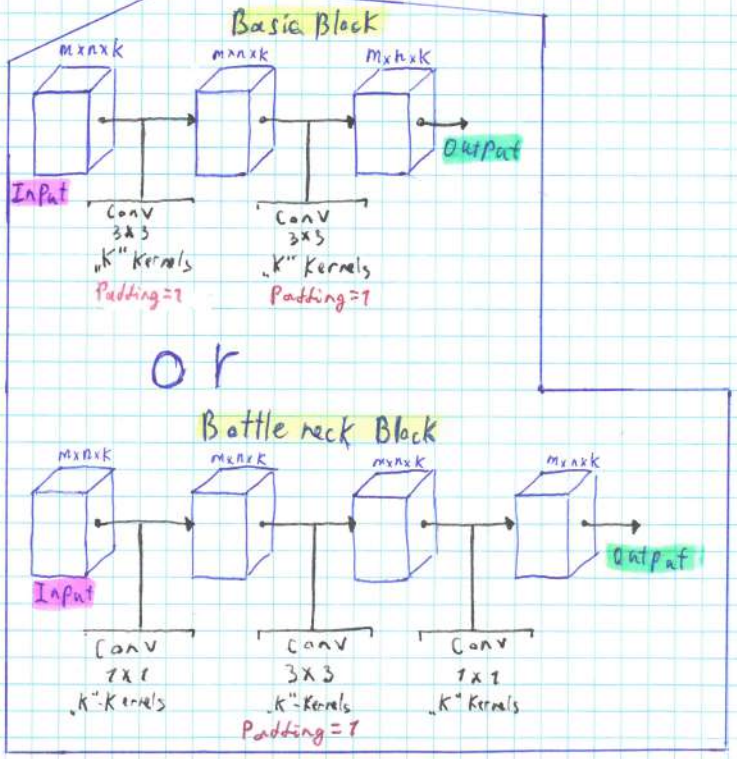


• In CNN: To perform such operation $X_{identity}$ should have the same Dimentionality as an output tensor. To do it, output of a „Block" should have Width and hight same as „X".

use Conv(Kernel=[1x1]; amount=n) to increase or decrease Depth of a tensor „X identity"

Res Net Example:





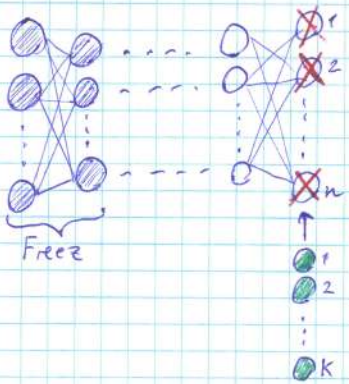
Transfer Learning

The Idea is to somehow transfer „Knowlege" from one network to another.

Fine tuning:

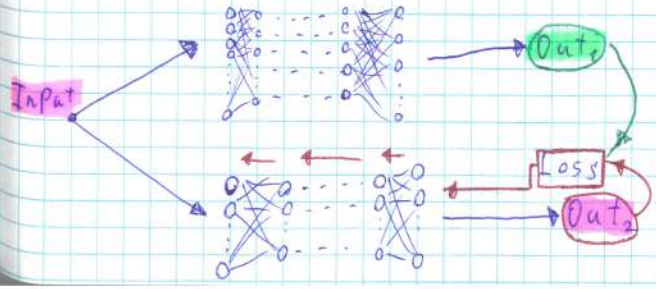
This method is used when you have an already trained model which solves fundamental problem. But you want to concretize or narrow down the problem.

- Copy trained model
- Drop down Output layer and replace with correct one.
- Freez some (or all) first layers.
- Re-train it on new Dataset.



Black Box Learning:

Uses if there exist huge already trained Neural Net and you want to get similar result or approximate it By smaller Network.

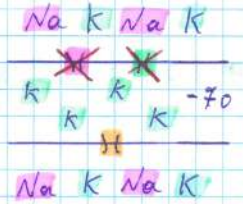


SPIKING Neural Networks (SNN)

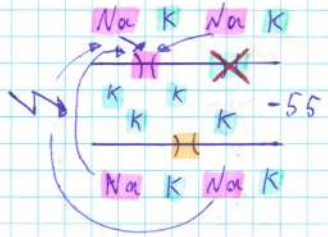
It is special type of ANN which principle of work is Based on neuromorphic concept.

Biological Concept:

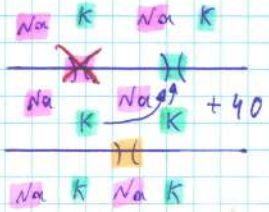
- **Firstly** consider that every neuron cell is charged negatively due to small amount of „K" ions and absence of „Na" ions (-70mV).



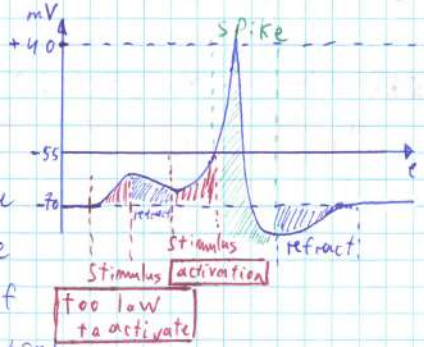
- **Stimulus:** when some comes, it increases the potential of cell. So when the potential reaches the threshold of (-55mV) it activates „Na" ions channels (Voltage Gates)



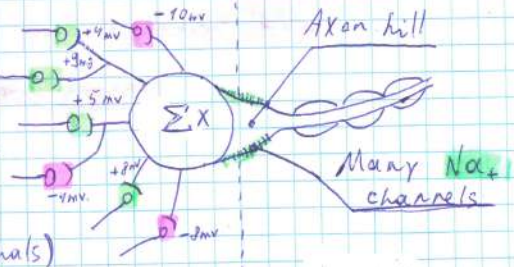
- **Spike:** when potential reaches (+40mV) „Na" ions channels closes and „K" ions channels opens so the potential rapidly decreases.



- **„Na" „K" pump:** it is special mechanism that works all the time. It takes 3 „Na" from the cell and inputs 2 „K" inside the cell. (Thats why it takes a lot of time to decrease or increase potential to the stable (-70mV).



Signal Processing:



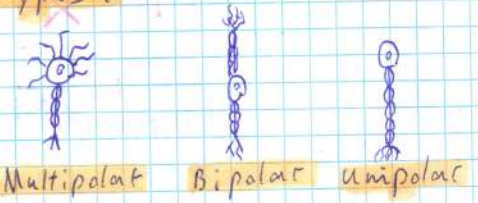
When signals (current from synapses) comes to the Soma, it changes the membrane potential (Actually soma simply sums all the signals) and when it reach the threshold ANalog Spike comes from Axon hill.

Pseudo Digital

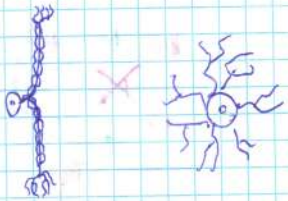
Neuron Cells:

Dendrites - Inputs
Axons - Output

Types:

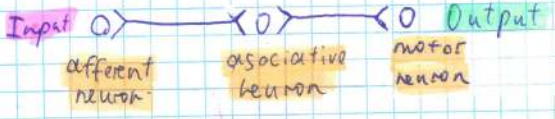


Multipolar Bipolar unipolar



Pseudounipolar

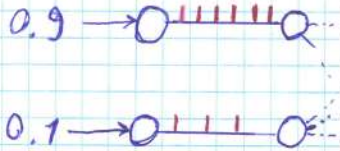
No axon.



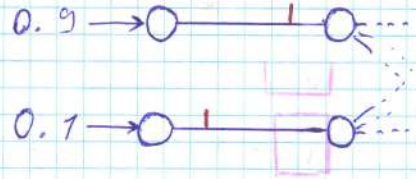
SNN Data Encoding:

The input and output data should be encoded as a sequence of spikes.

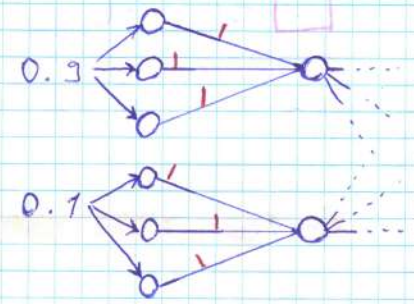
- **Rate coding:**
  Corresponding input Neuron return spikes more frequently if the input is larger.



- **Temporal coding:**
  Neuron with the largest input value fires first.



- **Population Coding:**
  Several input neurons (on one feature) to create spikes with Different delay. So the second layer get the sum of signals.



Models of Neurons:

$V(t)$ - Voltage of neuron (potential)

$\frac{dV}{dt}$ - How does the voltage changes.

C - Neuron Capacity

$\tau = R_{membrane} \cdot C_{membrane}$

I - Sum of all income signals (while other neurons spikes)

• Integrate and Fire

• Leaky Integrate and Fire (LIF).

The idea is the same as integrate and fire model But now the potential ("V"- voltage) always decreases exponentially.

$$C \cdot \frac{dV}{dt} = -leak + I \qquad \text{where: } leak = \frac{C}{\tau}(V_{(t)} - V_0)$$

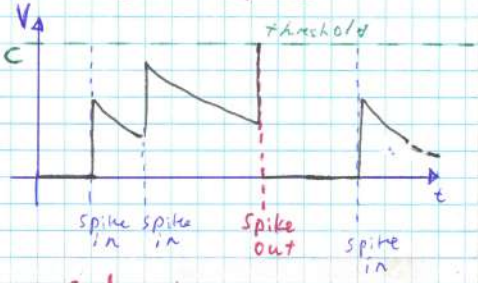
exponential decrease of voltage (always).

If spike comes from other neurons increase voltage on some „spike" constant.

Increase or slow down leakage. Uses to synchronize system frequency.

By substituting:

$$\frac{dV}{dt} = -\frac{1}{\tau}(V - V_0) + I$$



In other Interpretation:

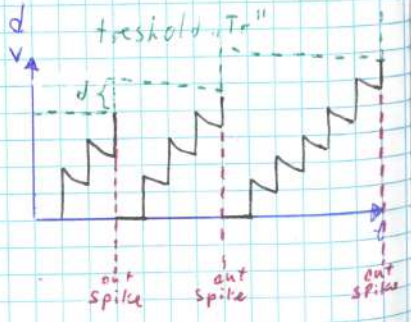
$V_{(t)}$ {

- Always: $\frac{dV}{dt} = -\frac{1}{\tau}(V - V_0)$ ← Leak
- If received I : $V = V + I$ ← Integrate
- If $V >$ Threshold : $V = 0$ ← Fire (non linearity)

• Adaptive (2D) LIF:

The same as Basic LIF But the threshold is now Dynamical „T_r" (Homeostasis analog)

$$\begin{cases} \frac{dV}{dt} = -\frac{1}{\tau}(V - V_0) + I \\ \frac{dT_r}{t} = \frac{1}{a}(1 - T_r) \\ \text{If } V \geq T_r \text{ Then: } V = 0,\ T_r += d \end{cases}$$



• Izhikeivich Model:

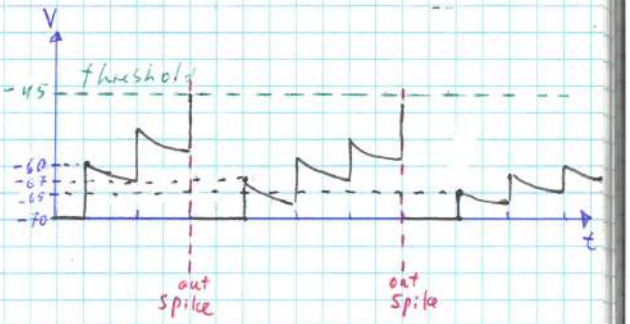
The concept is to „$u_{(t)}$" function, that adaptivly Decrease the Voltage per spike if there where multiple spikes inline Before.

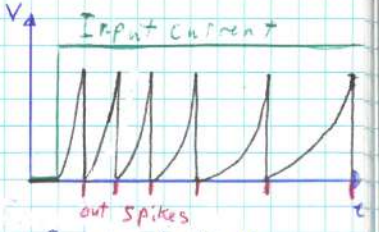
$$\begin{cases} C\cdot\frac{dV}{dt} = \underline{0.04}V^2 + \underline{5}V + \underline{140} + \boxed{I} - \boxed{u} \\ \frac{du}{dt} = a(bV - u) \\ \text{If } V \geq \text{Threshold}: V = \quad,\ u += d \end{cases}$$

If input spike comes, increase Voltage.

Decrease the Current per input spike. (Recovery Variable)

- a - timescale of „u". Smaller values leads to slower recovery (default $a = 0.02$)
- b - Sensitivity of „u"
- d - After Spike reset of „u". (By Default $d = 2$)





General View.

Hodgkin - Huxley Model:

This is one of the most precise and heavy computable model of neuron. It simulates the Ion channel process.

$$C\frac{dV}{dt} = \underbrace{I_K}_{\text{"K" Ion channel current}} + \underbrace{I_{Na}}_{\text{"Na" Ion channel current}} + \underbrace{I_L}_{\text{"Na" "K" pump}}$$

$$C\frac{dV}{dt} = \underbrace{g_K}_{\text{"K" Gating Variable}} \cdot (V_K - V) + \underbrace{g_{Na}}_{\text{"Na" Gating Variable}} \cdot (V_{Na} - V) + \underbrace{g_L}_{\text{leak Gating Variable}} \cdot (V_L - V)$$

Gating variables „$g(V,t)$" are functions that regulates when to open or close corresponding Ion channel.

4 Gates in 1 channel — 3 Gates in 1 channel

$$C\frac{dV}{dt} = \bar{g}_K \cdot f_K^{(4)} \cdot (V_K - V) + \bar{g}_{Na} \cdot f_{Na}^{(3)} \cdot h_{Na} (V_{Na} - V) + \bar{g}_L (V_L - V)$$

- $\bar{g}_K$: Maximum conductivity for „K"
- f_K: open/close function
- $\bar{g}_{Na}$: Maximum conductivity for „Na"
- f_{Na}: Activation (open) function
- h_{Na}: Deactivation (close) function
- $\bar{g}_L$: Constant Leak

$$f_K = \frac{df_K}{dt} = \alpha_K(1 - f_K) - \beta_K f_K \quad \text{where:} \quad \alpha_K = \frac{0.01(10-V)}{\exp\left(\frac{10-V}{10}\right) - 1}$$

$$\beta_K = 0.125 \cdot \exp\left(-\frac{V}{80}\right)$$

$$f_{Na} = \frac{df_{Na}}{dt} = \alpha_{Na}(1 - f_{Na}) - \beta_{Na} f_{Na} \quad \text{where:} \quad \alpha_{Na} = \frac{0.1(25-V)}{\exp\left(\frac{25-V}{10}\right) - 1}$$

$$\beta_{Na} = 4 \cdot \exp\left(-\frac{V}{18}\right)$$

$$h_{Na} = \frac{dh_{Na}}{dt} = \alpha_h(1 - h_{Na}) - \beta_h h_{Na} \quad \text{where:} \quad \alpha_h = 0.075 \cdot \exp\left(-\frac{V}{20}\right)$$

$$\beta_h = \frac{1}{\exp\left(\frac{30-V}{10}\right) + 1}$$

Where „α" - open coefficient and „β" - close coefficient.

Training:

SNN's could Not be trained By using basic methods such as's backpropagation because they operate with spikes

• Spike Timing dependent Plasticity (STDP):

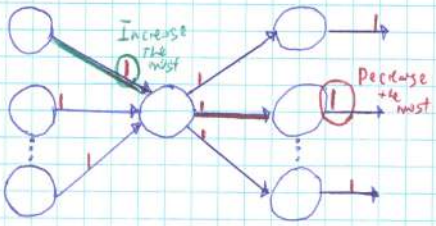
It is an Unsupervised learning algorithm that operates with only two basic rules:

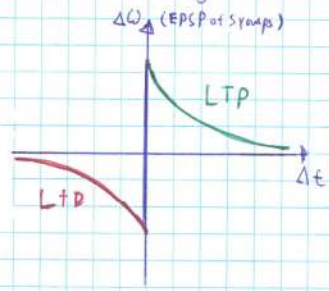
- If PreSynaptic Neuron Spikes and after that post synaptic; then increase this weight.
- If PostSynaptic Neuron Spikes and then Presynaptic; then Decrease this weight.

#The one is probably increasing synchronous activity.

Detailed Realization:

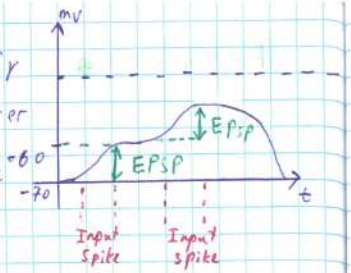
When the neuron Spikes, we check all it's input and out put connections, get „Δt" time interval in which other neurons Spikes. Now using our function Decrease all out put weights and increase input



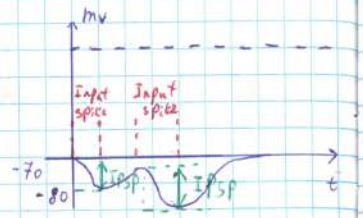


$$W_{new} \mathrel{+}= \Delta W \begin{cases} \text{if presynaptic: } \Delta W = A_{pre}\, e^{-\frac{\Delta t}{\tau}} \\ \text{if postsynaptic: } \Delta W = A_{post}\, e^{\frac{\Delta t}{\tau}} \\ \text{else: } \Delta W = 0 \end{cases}$$

EPSP: How much synapse would amplify the input signal. The more it is, the faster neuron would fire. (Excitatory Postsynaptic Potential.)



IPSP: How much synapse would decrease (Inhibit) signal. The higher it is, The more more unreachible is threshold. (Inhibitory postsynaptic Potential)



- **Homeostatic STDP:**

The idea is to make STDP more stable in recurent cases (all loops would infinitly increase their weights - „Hebbian rule").

Synaptic Normalization:

Each neuron have Synaptical resource:

$$\omega_A = \omega_{min} + \frac{(\omega_{max} - \omega_{min})}{}$$

Biological Concepts:

Types of synapses:

- Electrical
- Chemical

→

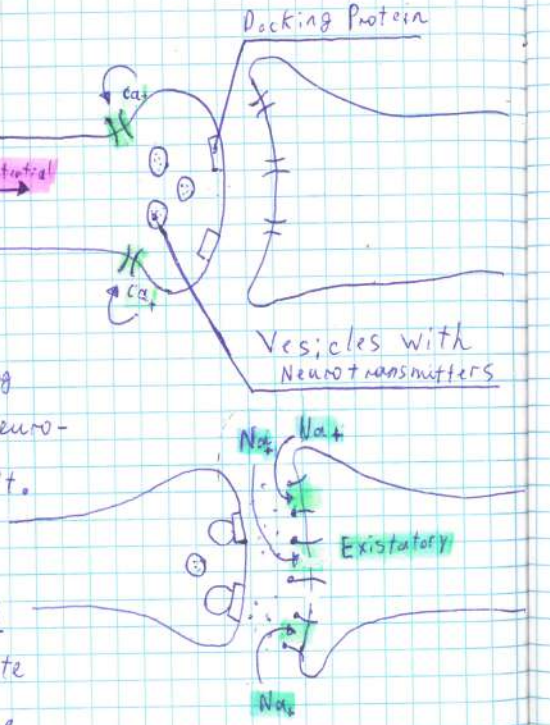
- Axo Dendritic (Excitatory)
- Axo Somatic (Inhibitory)
- Axo Axonic (Modulatory)

→

- Existatory
- Inhibitory
- Modulatory

How Does This Work:

- When the Action Potential comes to synapse it opens „Ca_{2+}" Ion channels.
- „Ca" connects to Vesicles and make them connect to the docking protein, which results the neurotransmitters are in Synaptic Cleft.
- Depending on Neurotransmitters corresponding receptors (Ligand Gated Channels) would activate and increase (Existatory) or decrease (Inhibitory) the membrane Potential.



Mostly, Neurotransmitters are synthesized in the Soma and transmitted By Motor Proteins to the synapses.