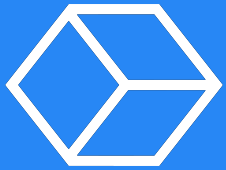


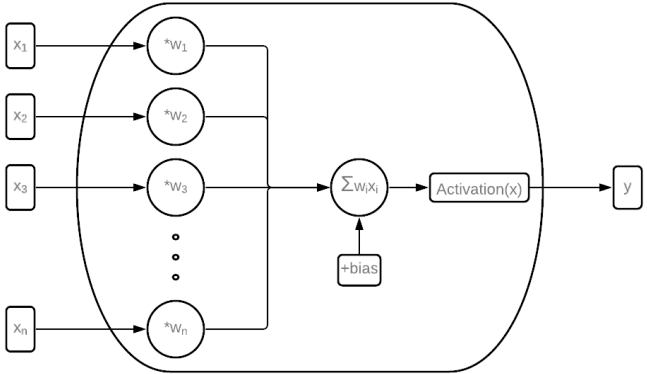
Introduction to neural networks

Laboratory of robotics and control systems
ITT PROJECT MANAGEMENT SERVICES L.L.C

Plan

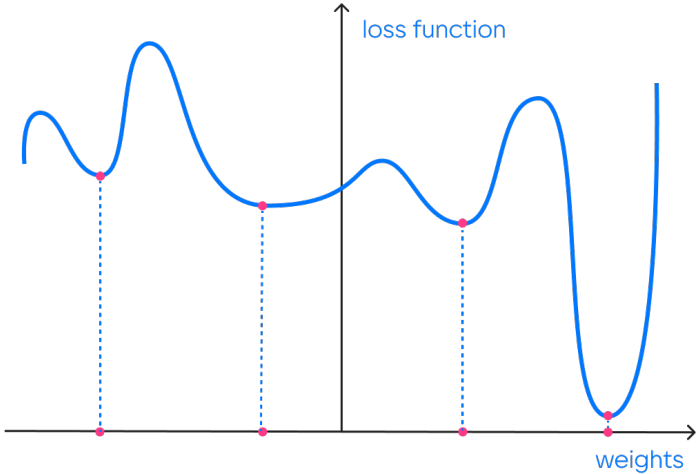
1. Neuronal connection
2. Activation functions
3. Fix overfitting
4. Fix underfitting
5. Classification Metrics

Neuron

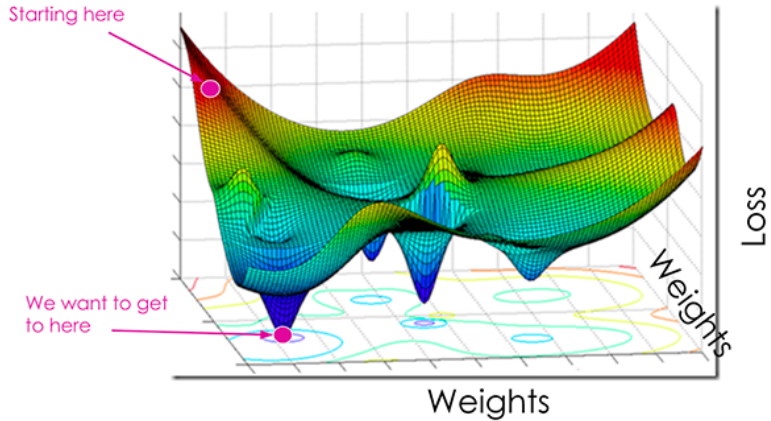


$$y = Activation(\sum_{j=1}^{n} w_j x_j + bias)$$

Numerous minimums



Numerous weights

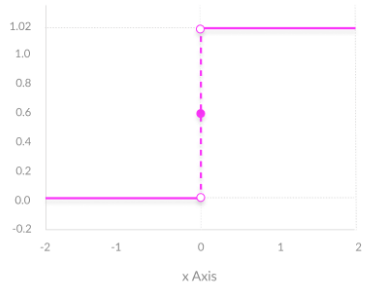


https://pyimagesearch.com/wp-content/uploads/2019/10/trainvallosslandscape.png

Table of Contents

Threshold function

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

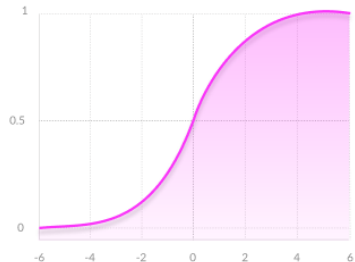


- **Advantages** Easily calculated. Provides with opportunity to divide into two classes. May be applied to binary classification.
- **Disadvantages** Not everywhere differentiable. Does not depend on the signal.

Sigmoid function

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

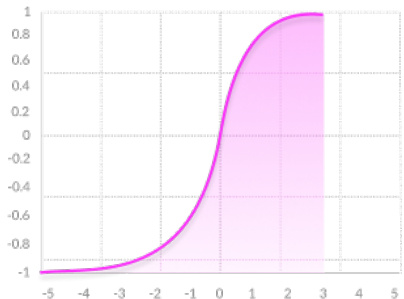
- **Advantages** The derivative depends on the signal. Nonlinear.
- **Disadvantages** Time-consuming calculations. Saturated neurons ”kill” the gradients.



Tanh function

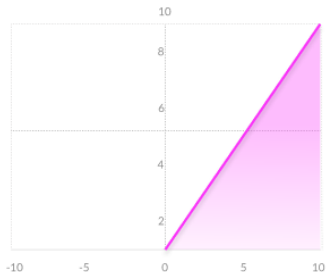
$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

- **Advantages** The derivative depends on the signal. Nonlinear. Zero-centered.
- **Disadvantages** Time-consuming calculations. Saturated neurons "kill" the gradients.



ReLu

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

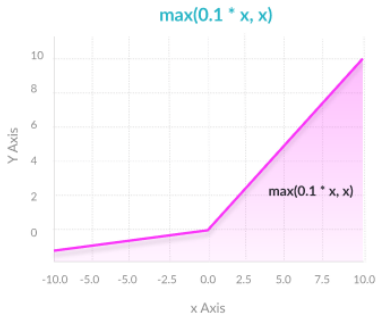


- **Advantages** Easily calculated. The derivative depends on the signal. Nonlinear.
- **Disadvantages** Problem of dying neurons.

Leaky ReLu

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.1x & \text{if } x < 0 \end{cases}$$

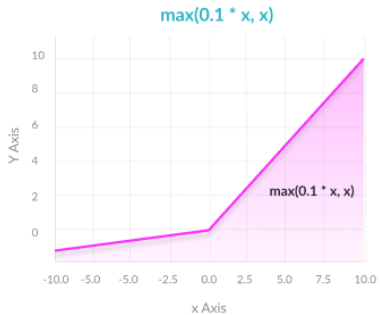
- **Advantages** Easily calculated. The derivative depends on the signal. Nonlinear. No problem of dying neurons.
- **Disadvantages** ?



Parametric ReLu

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

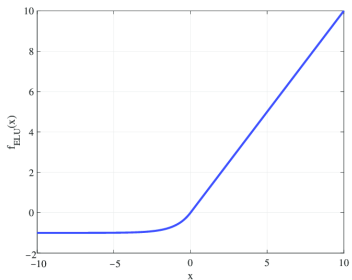
- **Advantages** Easily calculated. The derivative depends on the signal. Nonlinear. No problem of dying neurons. α is learnt during training.
- **Disadvantages** ?



ELu

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

- **Advantages** Easily calculated. The derivative depends on the signal. Nonlinear. No problem of dying neurons. α is learnt during training.
- **Disadvantages** Harder than PReLu, Leaky ReLu in calculations.



Maxout "Neuron"

$$f(x) \quad = \quad max(w_1^T x + b_1, w_2^T x + b_2)$$

- **Advantages** Easily calculated. The derivative depends on the signal. Nonlinear. w_i, b_i are learnt during training. No problem of dying neurons.
- **Disadvantages** Increase number of learning params.

Initializing weights

- calibrated random numbers initialization

  $\forall i,\, w_i \sim \mathcal{N}(0, \sigma^2), \sigma^2 = \frac{1}{n_{input}}$

  The main goal to make σ^2_{input} similar to σ^2_{output} at forward pass.

- Xavier Uniform

  $\forall i,\, w_i \sim\ U(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_i n+n_o ut}})$ $\qquad Var[U[a,b]] = \frac{(b-a)^2}{12}$

  The main goal to make σ^2_{input} similar to σ^2_{output} at forward pass.

  The main goal to make σ^2_{input} similar to σ^2_{output} at backward pass.

  For tanh activation function.

- Kaiming Normal

  $\forall i,\, w_i \sim \mathcal{N}(0, \sigma^2), \sigma^2 = \frac{2}{n_{in}}$

  Similar to Xavier-initialization. For ReLu activation functions.

Initializing weights

Default PyTorch initialization

- **nn.Linear, nn.Conv2d** - Kaiming Uniform Initialization

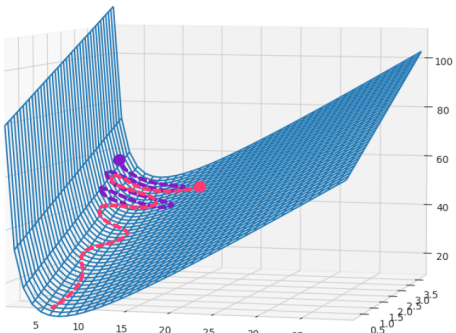
Customizing PyTorch initialization

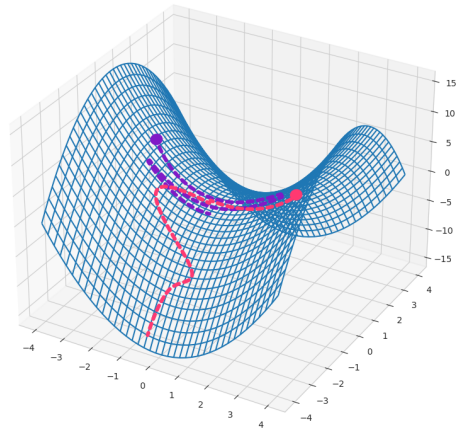
- Using torch.nn.init:
  - `xavier_uniform_` suitable for symmetric activations
  - `kaiming_normal_` suitable for ReLU-based networks
  - `constant_` suitable for initialization with fixed values
- Manual Initialization:
  `linear.weight.data.fill_(0.1)`

Tradeoff

- **Default Initialization**
- **Initialization by weights of pretrained model**
- **Custom Initialization**

Optimizer


100
80
60
40
20
5
10
15
20
25
30
35
0.5
1.0
1.5
2.0
2.5
3.0
3.5


15
10
5
0
−5
−10
−15
−4
−3
−2
−1
0
1
2
3
4

Optimizer examples

- GD
- SGD
- Batch SGD
- Momentum SGD
- Nesterov Momentum
- AdaGrad
- AdaDelta
- RmsProp
- Adam = Momentum + RmsProp
- Nadam = Nesterov Momentum + RmsProp

GD vs SGD

Gradient Descent

$$E(\omega) = \sum_{i}^{N} \frac{L(x_i, y_i, \omega)}{N} + \lambda R(\omega)$$

$$\frac{dE(\omega)}{d\omega} = \frac{1}{N} \sum_{i}^{N} \frac{dL(x_i, y_i, \omega)}{d\omega} + \lambda \frac{dR(\omega)}{d\omega}$$

- Slow Convergence

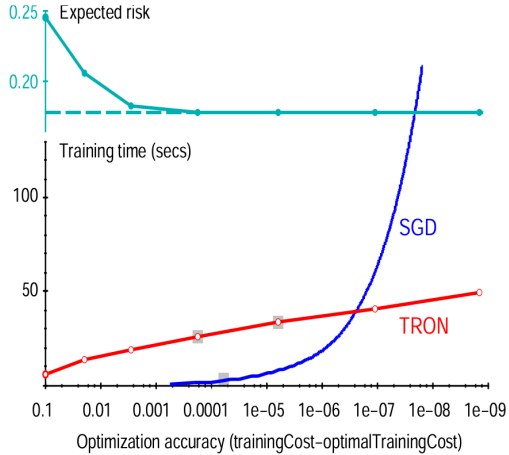
Stochastic Gradient Descent

$$E(\omega) = L(x_i, y_i, \omega) + \lambda R(\omega)$$

$$\frac{dE(\omega)}{d\omega} = \frac{dL(x_i, y_i, \omega)}{d\omega} + \lambda \frac{dR(\omega)}{d\omega}$$

- **Advantages:** Fast convergence.
- **Disadvantages:** High differences in eigenvector curvatures affect convergence. Noise negatively impacts convergence.

GD vs SGD



TRON (Trust Region Newton Method) RCV1 - dataset with newspapers and their categories

Paper

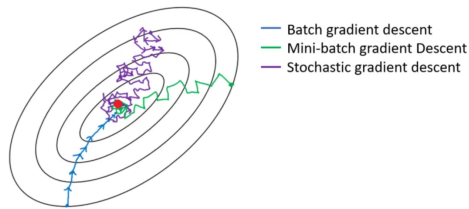
Batch SGD

$$E(\omega) = \sum_{i}^{N_b} \frac{L(x_i, y_i, \omega)}{N} + \lambda R(\omega)$$

$$\frac{dE(\omega)}{d\omega} = \frac{1}{N_b} \sum_{i}^{N_b} \frac{dL(x_i, y_i, \omega)}{dw} + \lambda \frac{dR(\omega)}{d\omega}$$

- **Advantages**
  Fast convergence. High differences in eigenvector curvatures values affect less on convergence. Decreasing of noise allow to increase learning rate and solve the optimization task with less number of steps. Get more stable convergence.

- **Disadvantages**
  Requires additional GPU memory. (not a problem actually)

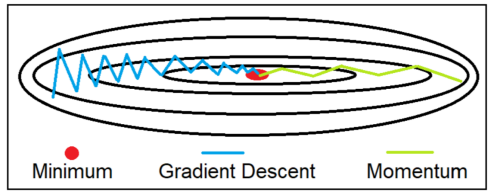


comparison

Momentum SGD

$$V_{t+1} = \beta V_t + \nabla L(\omega_t)$$
$$\omega_{t+1} = \omega_t - \alpha V_{t+1}$$

- Use information from previous steps
- β - hyperparameter
- Solve problem of local minimums
- http://distil.pub/2017/momentum



comparison

Nesterov Momentum SGD

$$V_{t+1} = \beta V_t + \nabla L(\omega_t + \beta V_t)$$
$$\omega_{t+1} = \omega_t - \alpha V_{t+1}$$

- Move before E calculations
- In some classes of functions converges faster than Momentum SGD

Figure 7. (a) Momentum vs. (b) Nesterov momentum.

comparison

Adagrad

$$cache_{t+1} = cache_t + \nabla L(\omega_t)^2$$

$$\omega_{t+1} = \omega_t - \alpha \frac{\nabla L(\omega_t)}{\sqrt{cache_{t+1}} + \epsilon}$$

- Deal with weights values oscillations.
- Try to specify α near local minimums and plateaus.
- $cache$ value always increases during training process.
- No need to change α with hands at every step.

RMSProp

$$cache_{t+1} = \beta cache_t + (1-\beta)\nabla L(\omega_t)^2$$

$$\omega_{t+1} = \omega_t - \alpha \frac{\nabla L(\omega_t)}{\sqrt{cache_{t+1}} + \epsilon}$$

- Deal with weights values oscillations.
- $cache$ value depends on current situation.

Adam

$$V_{t+1} = \gamma V_t + (1-\gamma)\nabla L(\omega_t)$$
$$cache_{t+1} = \beta cache_t + (1-\beta)\nabla L(\omega_t)^2$$
$$\omega_{t+1} = \omega_t - \alpha \frac{V_{t+1}}{\sqrt{cache_{t+1}} + \epsilon}$$

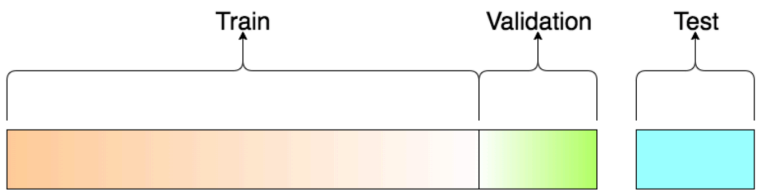
- Combine previous ideas.
- Why is SGD usually used in Research? (memory)

Comparing optimizers

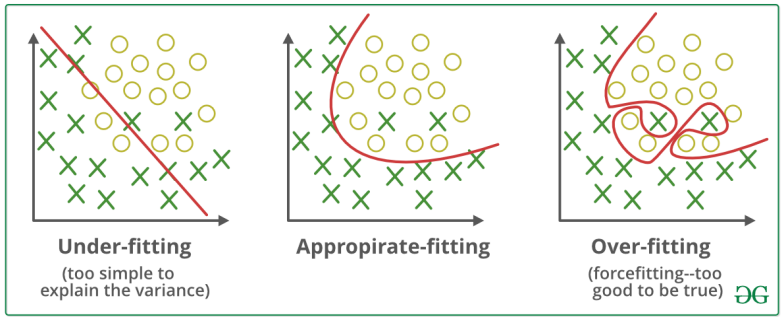
- birdview
- sedlo
- ravine
- medium
- tool

Train/val/test split

- How to identify that training is finished? (Look at Loss or Metrics)
- How to identify is model's generalization valid? (Look at perfomance on another dataset)
- How to get another dataset? (Split current)



Training Problems



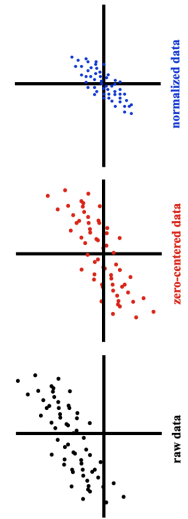
- Model does not learn features to identify a valid generalization.

- Results of training that are obtained by memorizing the dataset, rather than identifying a generalization.

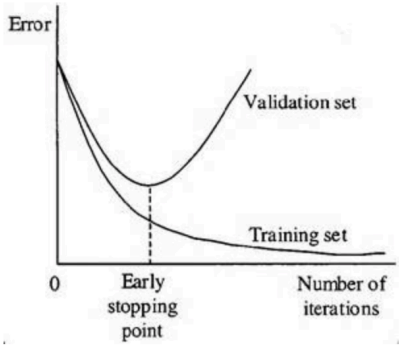
Table of Contents

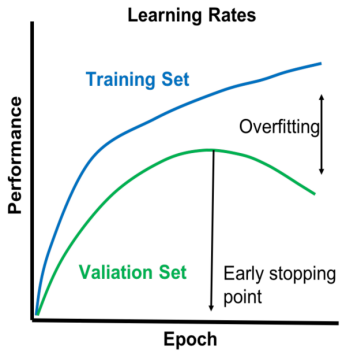
Data and Model

- Increase volume of dataset.
- Normalize data.
- Epoch - passing through the whole dataset.
- Learning with multiple epochs. Important: Data can be shuffled. Different problems require different number of epochs.
- Choose smaller model's architecture.



Early Stopping





- Early stopping point describes the most valid model state during training process.

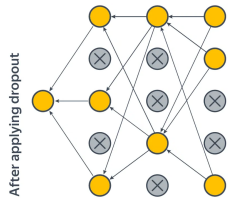
Dropout

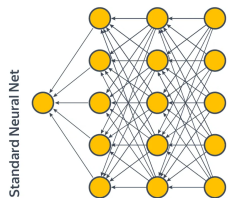
No Dropout

- $x^k = f_k(x^{k-1})$

Dropout

- $x^k = \frac{1}{1-p} * f_k(x^{k-1}) \odot mask$
- $mask = Bernoulli(1-p)$
  Last hidden layer: $mask = [1, 0, 1, 0, 1]$
- p - probability of zeroing a neuron of a layer
- Available only in train. Should be turned off in eval.





Dropout

- **Advantages:** probability of overfitting decreases, model becomes more stable to noise. (Averaging weights values through training on different submodels)
- **Disdvantages:** Effective generalization of model decreases. (Training on submodels)

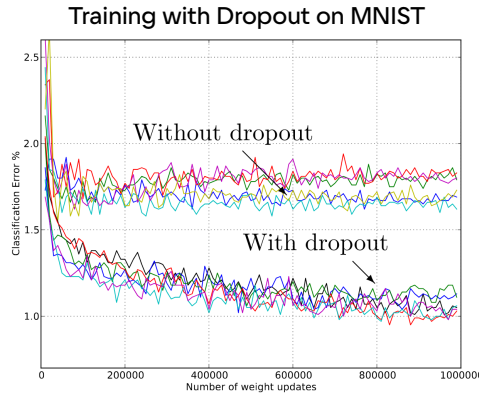
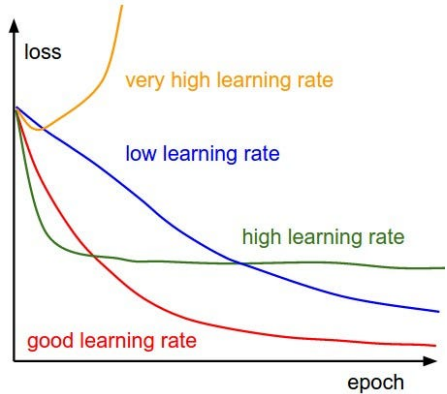


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Table of Contents

Learning rate scheduler

- Learning Rate Decay
- Learning Rate by Adam



Batch Norm

The same as the normalizing data but throught the all layers of model

- μ_β, σ_β values depend on current batch
- γ, β values are learning during training

Adding BN to Inception

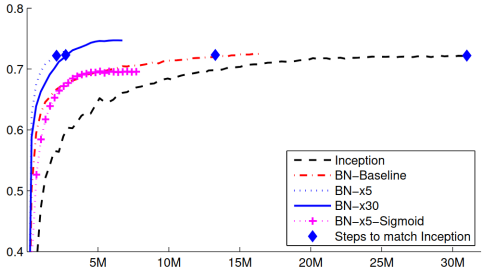


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma^2_\mathcal{B} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Norm

From train to eval

- Merge this layer with neighbouring one.
- Continue using batches for calculating μ_β, σ_β
- Use one input. In this case μ_β, σ_β are equal to running average values or exponential moving average of values during traininig proccess.
- Use one input. Fine-tune μ_β, σ_β after traininig proccess.

Others ways

- Layer Norm
- Batch Renorm

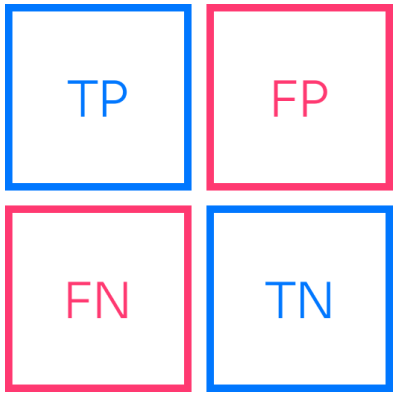
Data and Model

- Increase model's size. Example: approximate y(x) = 1 / (x2 + 1) with polynomial function.
- Use pretrained models.
- Less Dropouts
- Less Weight decays. $E(\omega) = \sum_i^N \frac{L(x_i, y_i, \omega)}{N} + \lambda R(\omega)$, where $R(\omega) = \|\omega\|^2, \|\omega\|$

Table of Contents

Types of predictions

- TP - true positive
- FP - false positive
- TN - true negative
- FN - false negative



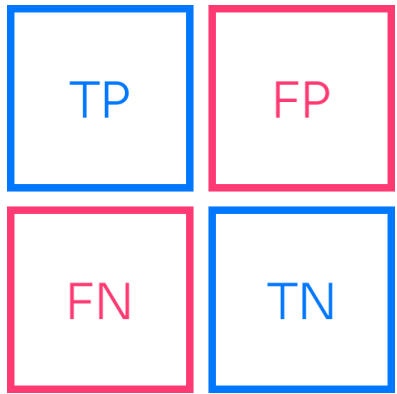
Classification metric choice

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$



Loss vs metrics

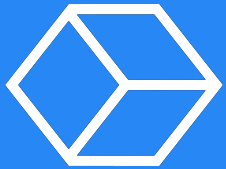
Loss

- The loss function is used to optimize your model.
- differential
- uninterpretable

Metrics

- A metric is used to judge the performance of your model.
- non-differentiable
- interpretable

Feedback



Thank you for your time

Laboratory of robotics and control systems
ITT PROJECT MANAGEMENT SERVICES L.L.C