NEURAL NETWORKS AND BACKPROPAGATION

Machine Learning for Autonomous Robots

Dr. Alexander Fabisch
DFKI, Robotics Innovation Center

December 20, 2022 – Bremen, Deutschland

University of Bremen

German Research Center for Artificial Intelligence

1. Recap: Linear Regression and Logistic Regression
2. Neural Networks
3. Learning Neural Networks with Backpropagation
4. Automatic Differentiation
5. Summary and applications

Recap

Linear Model

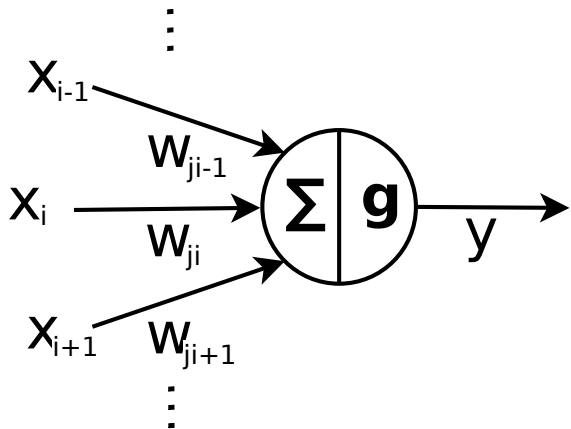


Figure: A linear model computes the **weighted** sum of inputs and optionally applies a non-linear scaling function g.

Multi-dimensional output:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}, \quad \boldsymbol{y} \in \mathbb{R}^F, \quad \boldsymbol{x} \in \mathbb{R}^D, \quad \boldsymbol{W} \in \mathbb{R}^{F \times D}$$

Error function (sum of squared errors):

$$E(\boldsymbol{W}) = \frac{1}{2} \sum_n ||\boldsymbol{W}\boldsymbol{x}_n - \boldsymbol{y}_n||^2$$

Gradient:

$$\nabla_{\boldsymbol{W}} E_n(\boldsymbol{W}) = (\boldsymbol{W}\boldsymbol{x}_n - \boldsymbol{y}_n)\boldsymbol{x}_n^T$$

Multi-Class Logistic Regression

Multi-dimensional output (one-hot encoding, softmax):

$$\boldsymbol{y} = g(\boldsymbol{W}\boldsymbol{x}),\ \boldsymbol{y} \in [0,1]^F,\ \boldsymbol{x} \in \mathbb{R}^D,\ \boldsymbol{W} \in \mathbb{R}^{F \times D},\ g_i(t) = \frac{e^{t_i}}{\sum e^{t_j}}$$

Error function (cross-entropy):

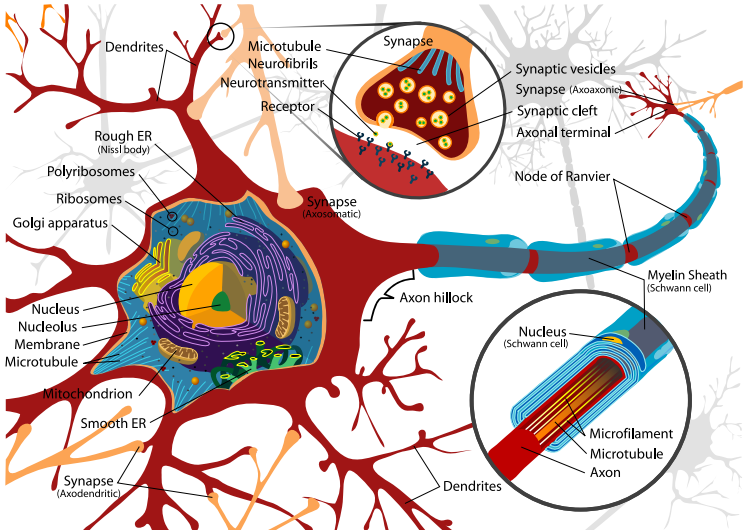
$$E(\boldsymbol{W}) = -\sum_n \sum_f \boldsymbol{y}_{nf} \ln \left(g(\boldsymbol{W}\boldsymbol{x}_n)\right)_f$$

Gradient:

$$\nabla_{\boldsymbol{W}} E_n(\boldsymbol{W}) = \left(g(\boldsymbol{W}\boldsymbol{x}_n) - \boldsymbol{y}_n\right) \boldsymbol{x}_n^T$$

Neural Networks

Natural Neuron



Artificial Neuron

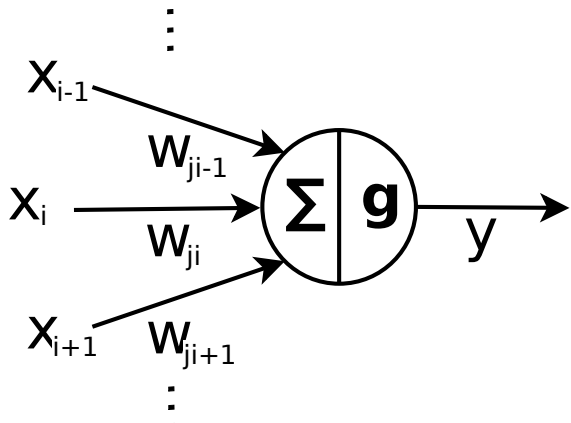


Figure: An artificial neuron computes the **weighted** sum of inputs and applies a non-linear activation function g.

Artificial Neuron

The **output** y_j of a neuron node is calculated by taking the sum of all signals x_i weighted by a **connection strength** w_{ji} and transforming the sum by the **activation function** g.

$$y_j = g\left(\sum_i^n w_{ji} x_i\right)$$

We can compute the outputs of multiple neurons that have the same inputs with

$$\boldsymbol{y} = g\left(\boldsymbol{W}\boldsymbol{x}\right) .$$

Does that remind you of something?

Artificial Neural Network (ANN)

- ANNs are **interconnected assemblies** of computational **nodes** which are loosely based on the animal **neuron**.
- The computational ability of the neural network is encoded in the inter-unit connection strengths (**weights**).
- A neural network has **input** and **output nodes** which receive and give out signals to the environment, respectively.
- All other nodes are called **hidden nodes**.

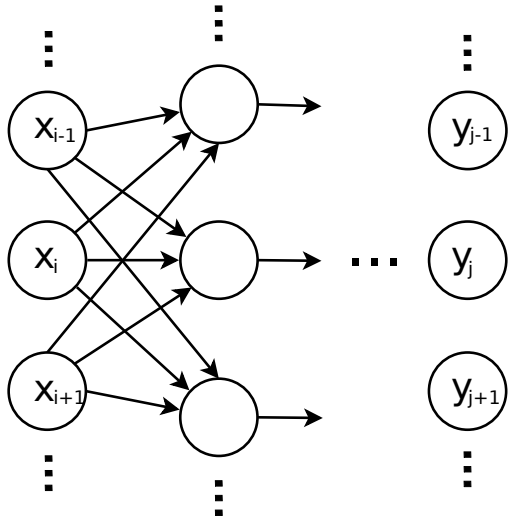


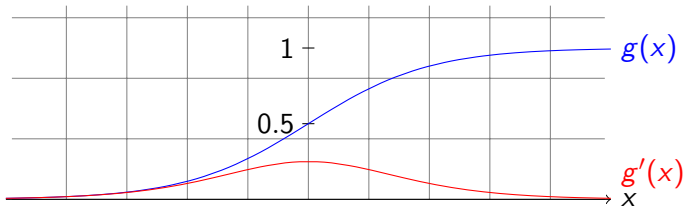
Figure: Where are the input nodes, output nodes and hidden nodes?

Activation Functions

Most commonly used activation functions:

- identity function $g(a) = a$
- rectified linear unit: $g(a) = a$ if $a > 0$, else 0
- leaky rectified linear unit: $g(a) = a$ if $a > 0$, else ϵa
- sigmoid functions:
  - logistic sigmoid $g(a) = \frac{1}{1+\exp(-a)} \in [0, 1]$
  - hyperbolic tangent: $g(a) = \tanh(a) \in [-1, 1]$

Logistic Sigmoid



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

Architecture

The architecture of a neural network plays an important role for its behavior.

- **number** of neurons
- **pattern of connectivity** (most neural nets are organized in layers: input layer, hidden layer(s) and output layer)
- **feedforward**: directed acyclic graphs
- **recurrent**: recurrent connection from upper layer to a lower layer, or recurrent connection from the same neuron, results in *complex time-dependent dynamics* (deutsch: rückläufig)

Multilayer Neural Network

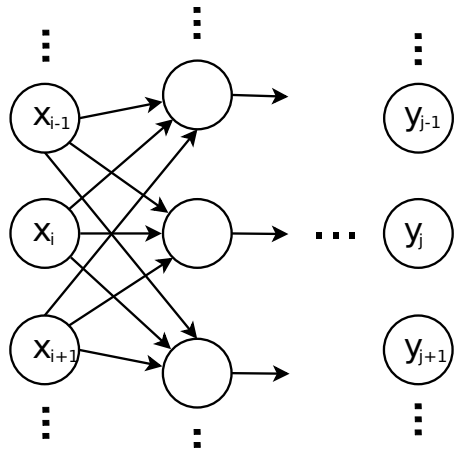
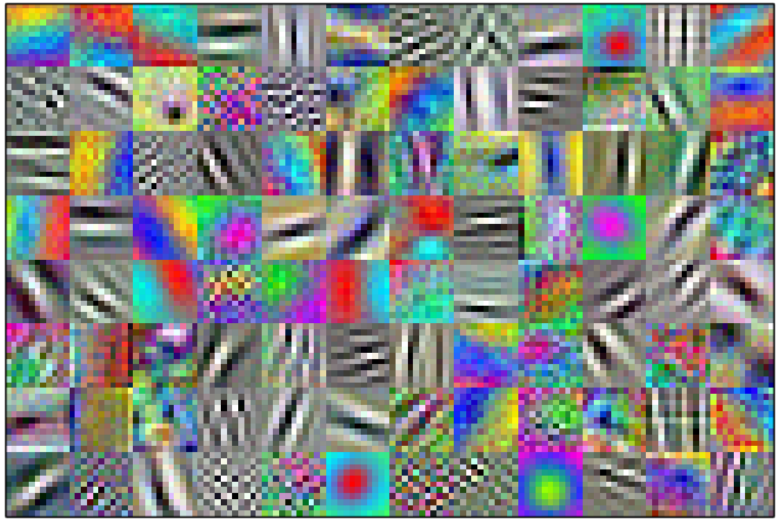


Figure: Neurons of successive layers are fully connected.

Weight Visualizations

Weight Visualizations



http://sklearn-theano.github.io/images/plot_overfeat_layer1_filters_002.png

Backpropagation

Supervised Learning

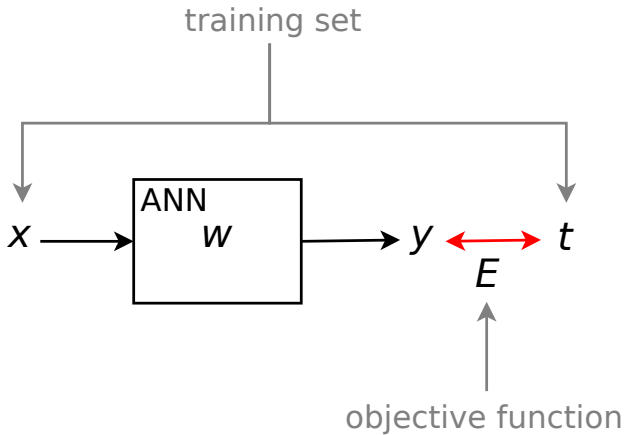


Figure: Weights are updated **iteratively** using the discrepancy between the **desired output** $\boldsymbol{t}$ and the **output given by the network for a given input pattern** $\boldsymbol{x}$.

Weight Update

The weights are modified, e.g. using **gradient descent**

$$w_{ji}(t+1) = w_{ji}(t) - \alpha \Delta w_{ji}(t) \qquad 0 < \alpha \leq 1\,,$$

where

- $w_{ji}(t)$ and $w_{ji}(t+1)$ is the weight of input i for neuron j before and after modification, respectively,
- and α is the learning rate

How can we compute $\Delta w_{ji}(t)$?

- **Backpropagation** is a method to compute the gradient of the error function E with respect to the weights $\boldsymbol{w}$ of a neural net

$$\nabla_{\boldsymbol{w}} E = \begin{pmatrix} \vdots \\ \frac{\partial E}{\partial w_{ji}} \\ \vdots \end{pmatrix}.$$

- The gradient can be used in the update equation (here: written with vectors)

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \alpha \Delta \boldsymbol{w}(t) \qquad 0 < \alpha \leq 1, \quad \Delta \boldsymbol{w}(t) = \nabla_{\boldsymbol{w}(t)} E$$

Objective Function

- We seek to minimize the total error E over all patterns in the training set given by (the index n labels the patterns)

$$E = \sum_n E^n.$$

- Let us assume that the error function E^n is a function of network outputs $y_1, \ldots, y_F$,

$$E^n = E^n(y_1, \ldots, y_F)\,.$$

- Our goal is to find a procedure (error back-propagation) for evaluating the derivative of the error function with respect to the weights of the network.

Forward Propagation

- Each neuron j in a neural net computes a weighted sum

$$a_j = \sum_i w_{ji} x_i,$$

  where x_i is the i-th input, which is connected to neuron j and w_{ji} is the weight associated with that connection.
- The output of neuron j denoted by y_j is obtained by

$$y_j = g(a_j).$$

- In the next layer, the output y_j of the previous layer will be written as x_i and the output of the last layer will be the output of the neural net.

Artificial Neuron

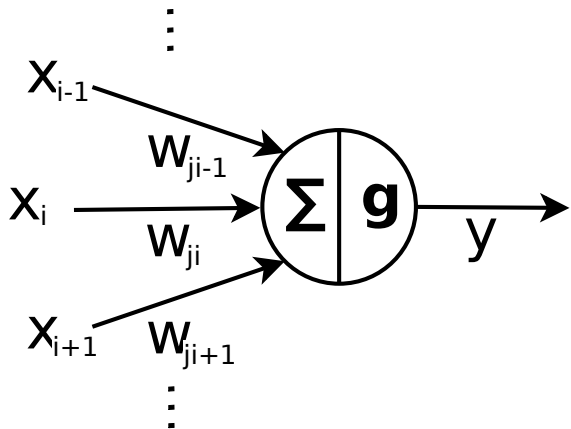


Figure: An artificial neuron computes the **weighted** sum of inputs and applies a non-linear scaling function g.

Tools (Differentiation Rules)

- Sum rule:

$$(f + g)' = f' + g'$$

- Chain rule:

$$(f \circ g)' = (f' \circ g) \cdot g'$$

$$\frac{\partial f}{\partial x} = \sum_y \frac{\partial f}{\partial y} \frac{\partial y}{\partial x}$$

- Product rule:

$$(f \cdot g)' = f' \cdot g + f \cdot g'$$

Notation

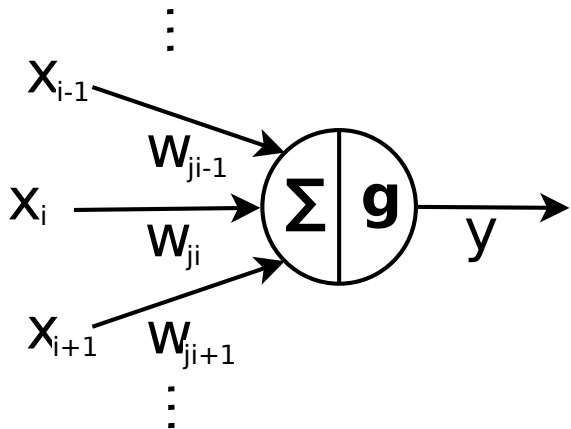


Figure: i refers to an input and j refers to the corresponding neuron. Note that in the next layer the output of neuron j might be called input with index i.

Gradient

- The partial derivative of E^n with respect to w_{ji} is

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} .$$

- It is obvious that

$$\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial \sum_{i'} w_{ji'} x_{i'}}{\partial w_{ji}} = x_i .$$

- By *forward propagation*, we can calculate x_i.
- The quantity $\frac{\partial E^n}{\partial a_j}$ is replaced by the symbol δ_j so that

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j x_i .$$

Deltas (Hidden Neurons)

- For a hidden neuron i, we derive an equation that represents δ_i in terms of δ_j's of neurons j **to which it sends its output**.
- We **back-propagate** the δ_j's for the network.

$$\delta_i = \frac{\partial E^n}{\partial a_i} = \sum_j \frac{\partial E^n}{\partial a_j} \frac{\partial a_j}{\partial a_i} = \sum_j \delta_j \frac{\partial a_j}{\partial a_i} .$$

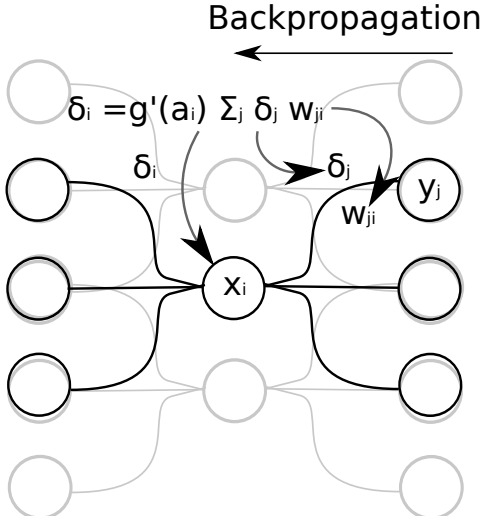
- We know that $a_j = \sum_{i'} w_{ji'} \underline{x_{i'}} = \sum_{i'} w_{ji'} \underline{g(a_{i'})}$. It follows that

$$\frac{\partial a_j}{\partial a_i} = g'(a_i) w_{ji} .$$

- The equation for the back-propagation is thus given by

$$\delta_i = \sum_j \delta_j g'(a_i) w_{ji} = g'(a_i) \sum_j \delta_j w_{ji} .$$

Deltas (Hidden Neurons)



Deltas (Output Neurons)

- For an output neuron k,

$$\delta_k = \frac{\partial E^n}{\partial a_k} = \frac{\partial E^n}{\partial y_k}\frac{\partial y_k}{\partial a_k} = \frac{\partial E^n}{\partial y_k} g'(a_k)$$

- Let as assume that the activation function of the output neurons is linear and is given by $g(a) = a$. **Since $g'(a_k) = 1$, we get**

$$\delta_k = \frac{\partial E^n}{\partial y_k}\,.$$

Deltas (Output Neurons)

- Assuming a sum of squared error function

$$E^n = \frac{1}{2}\sum_{k=1}^{c}(y_k - t_k^n)^2\,,$$

  we get

$$\delta_k = \frac{\partial E^n}{\partial y_k} = y_k - t_k^n\,.$$

- We see that δ_k represents in this case an error for the output units.

Summary (Backpropagation)

Error back-propagation for evaluating the derivative of E^n:

1. For a given input, do forward propagation: $a_j = \sum_i w_{ji} x_i, \quad y_j = g(a_j)$.
2. Compute the $\delta_k = y_k - t_k^n$ for each output neuron.
3. Back propagate the δ's using $\delta_i = g'(a_i) \sum_j \delta_j w_{ji}$ to obtain δ_i for each hidden neuron in the network.
4. Apply $\frac{\partial E^n}{\partial w_{ji}} = \delta_j x_i$ to calculate the required derivatives.

The derivative for the total error can be obtained using

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E^n}{\partial w_{ji}} .$$

Batch vs. Stochastic

For training the network the weights of the network are modified using

$$w_{ji} = w_{ji} - \alpha \Delta w_{ji} \qquad 0 < \alpha \leq 1\,,$$

For **online learning** (stochastic gradient descent) we choose

$$\Delta w_{ji} = \frac{\partial E^n}{\partial w_{ji}}$$

For **batch learning** (gradient descent) we choose

$$\Delta w_{ji} = \frac{\partial E}{\partial w_{ji}}\,.$$

Automatic Differentiation

Types of Differentiation

We can distinguish

- Symbolic differentiation
  - symbolic expression of gradient for all possible inputs
  - cannot compute derivative of complex control flows (branching, loops, recursion)
- Numeric differentiation
  - approximation of gradient for specific input
  - computationally expensive for large number of variables
- Automatic differentiation (AD)
  - evaluation of gradient for specific input

Automatic Differentiation

We can distinguish

- Forward mode / forward accumulation mode / tangent linear mode
  - implementation: dual numbers, single pass
  - better for multivariate functions with many outputs
  - worse for functions with large number of arguments
- Reverse mode / reverse accumulation mode
  - implementation: generalized backpropagation, two phases
  - better for functions with large number of arguments
  - worse for multivariate functions with many outputs

More details: `http://jmlr.org/papers/v18/17-468.html`

Dual Numbers

A dual number has the form

$$v + \dot{v}\epsilon,$$

where

- $v, \dot{v} \in \mathbb{R}$
- $\epsilon^2 = 0$ and $\epsilon \neq 0$

Basic operations:

- Addition: $(1 + 2\epsilon) + (3 + 4\epsilon) = 4 + 6\epsilon$
- Subtraction: $(1 + 2\epsilon) - (3 + 4\epsilon) = -2 - 2\epsilon$
- Multiplication: $(1 + 2\epsilon) \cdot (3 + 4\epsilon) = 3 + 6\epsilon + 4\epsilon + \underbrace{8\epsilon^2}_{0} = 3 + 10\epsilon$

Differentiation with Dual numbers

$$f(x) = x^2 + 2x + 3$$

What is $f'(x)$?

$$f'(x) = 2x + 2$$

For example, $f'(3) = 8$, $f'(-2) = -2$

$$\begin{aligned} f(3+\epsilon) &= (3+\epsilon)^2 + 2(3+\epsilon) + 3 \\ &= 9 + 6\epsilon + 6 + 2\epsilon + 3 \\ &= 18 + \underline{8\epsilon} \end{aligned}$$

$$\begin{aligned} f(-2+\epsilon) &= (-2+\epsilon)^2 + 2(-2+\epsilon) + 3 \\ &= 4 - 4\epsilon - 4 + 2\epsilon + 3 \\ &= 3\underline{-2\epsilon} \end{aligned}$$

Summary and applications

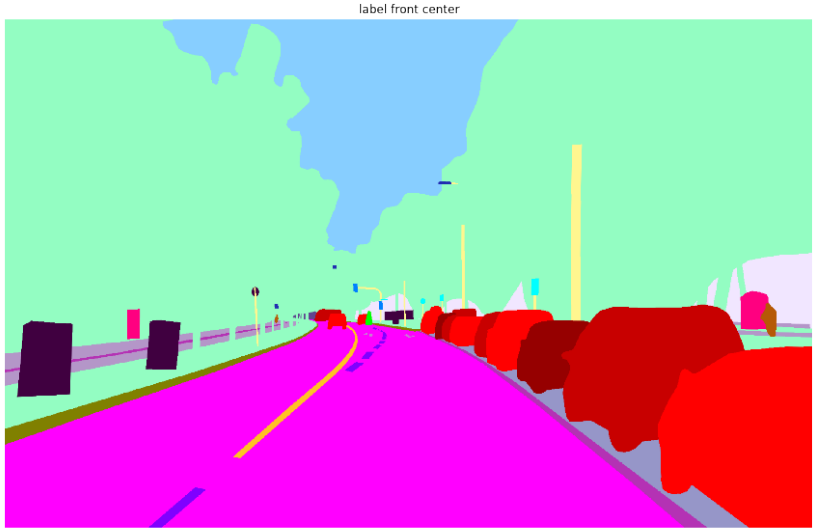
- Recap: linear regression and logistic regression
- Very loose motivation by natural neurons
- Different activation functions
- We often use a layer structure
- Derivation of backpropagation
- Backpropagation is automatic differentiation

Object Detection

For example:

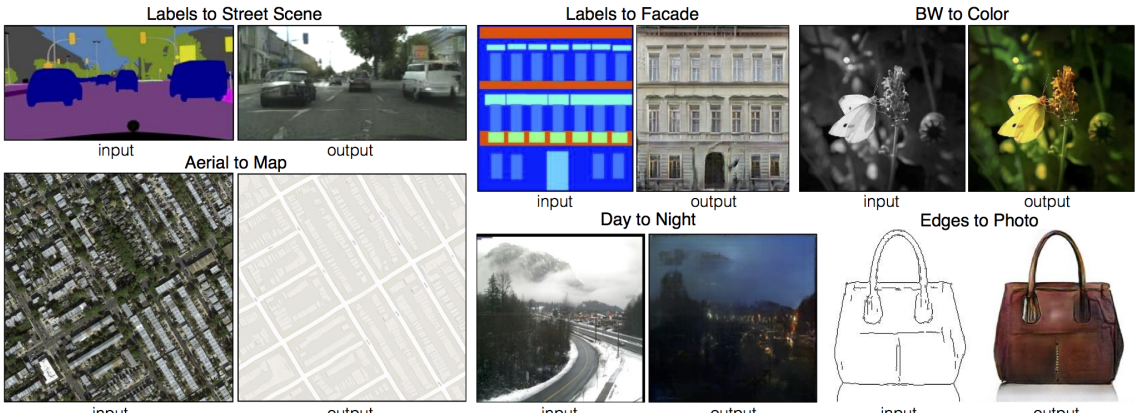
https://github.com/facebookresearch/detectron2

Semantic Segmentation



Dataset: `http://www.a2d2.audi`

Image Enhancement



Translation

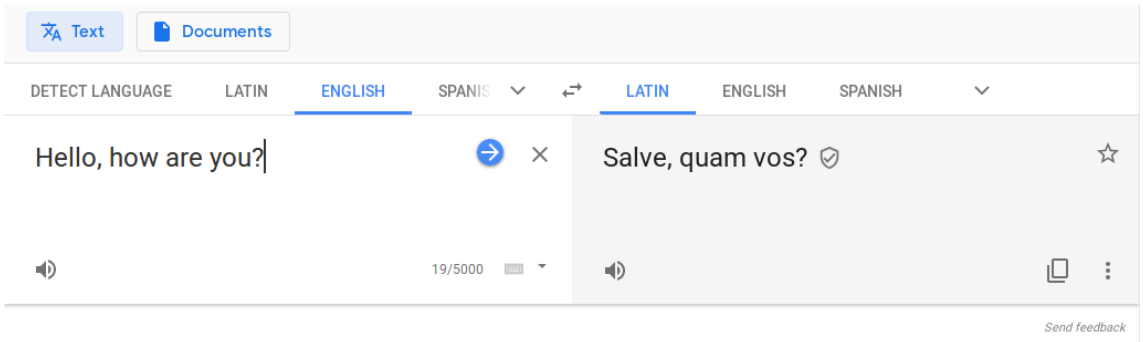


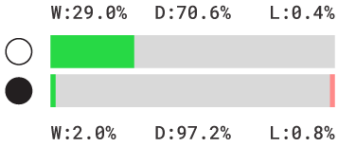
Image Captioning

a group of giraffes standing in a grassy area

Playing Board Games

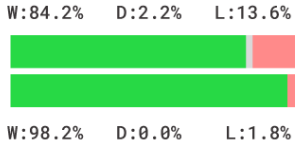
Chess

AlphaZero vs. Stockfish

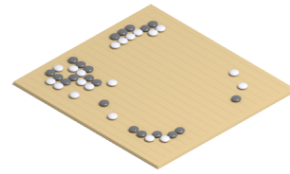


Shogi

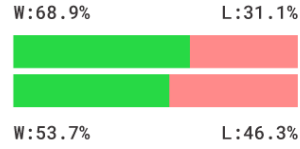
AlphaZero vs. Elmo



Go



AlphaZero vs. AG0



Playing Computer Games

Art

Thank You!
Please feel free to ask questions in the forums.