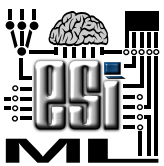
Machine Learning

Advanced Neural Networks

Abdelkrime Aries

Laboratoire de la Communication dans les Systèmes Informatiques (LCSI)
École nationale Supérieure d'Informatique (ESI, ex. INI), Algiers, Algeria

Academic year: 2024-2025



Attribution 4.0 International (CC BY 4.0)

https://creativecommons.org/licenses/by/4.0/deed.en

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Machine Learning

Advanced Neural Networks: Motivation

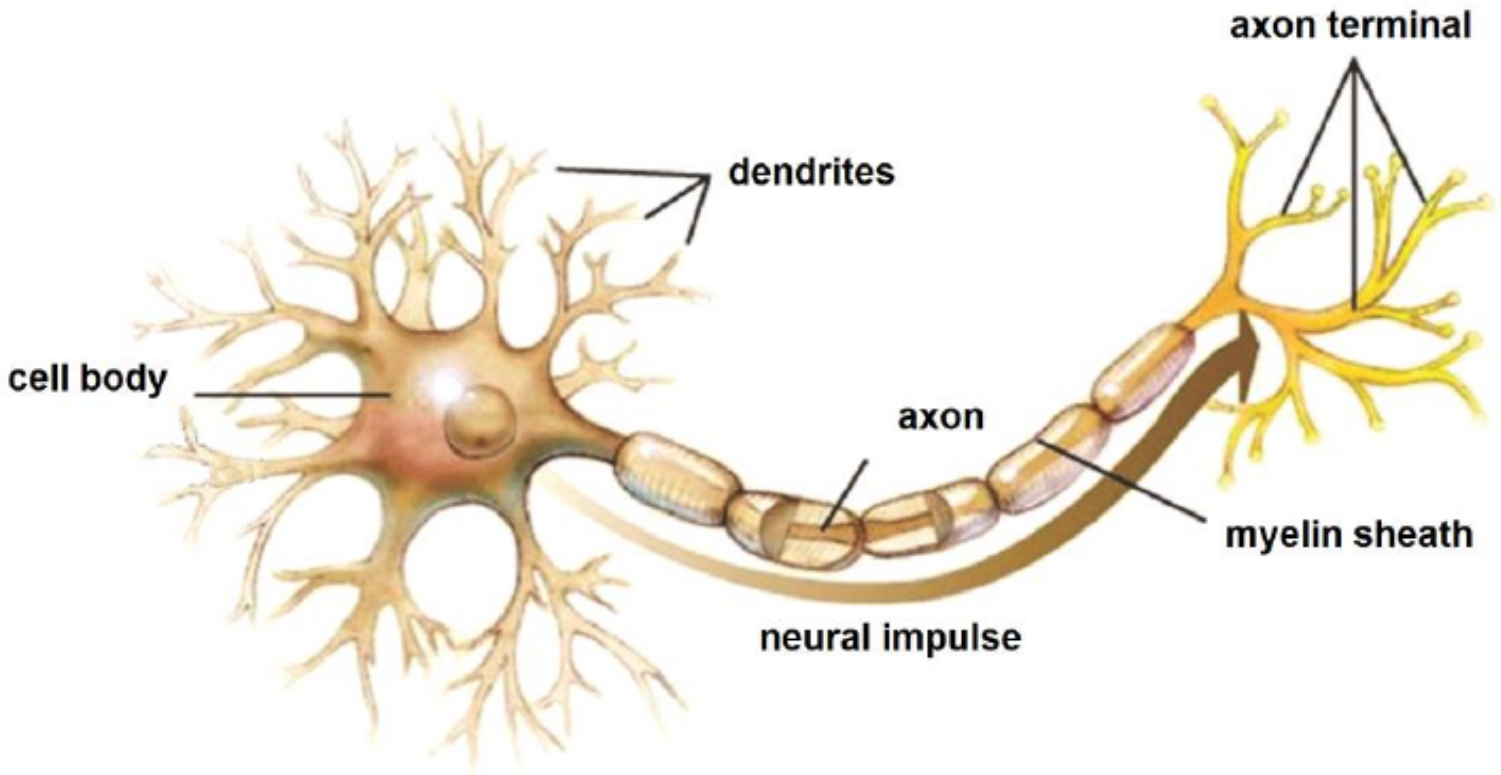


Figure: Biological neuron **[Cain, 2017]**

Machine Learning

Advanced Neural Networks: Network types

Neural Networks

Feed Forward (FFNN)

MultiLayer Perceptron (MLP)

- Dense (Fully connected)
- Embedding
- Droupout
- Layer normalization
- Batch normalization

Convolutional (CNN)

- Conv
- Pooling (max, avg)
- Flatten
- Up-sampling

Attention

- Dot-product (Luong)
- Additive (Bahdanau)
- Transformer

Recurrent (RNN)

Vanilla RNN

- Elman network
- Jordan network

Advanced RNN

- Long short-term memory (LSTM)
- Gated recurrent unit (GRU)

Energy based model (EBM)

- Hopfield network
- Boltzmann machine

Differentiable neural computers

- Neural Turing Machine (NTM)

Arch

- Encoder-Decoder
- Auto-encodeur
- Generative adversarial network (GAN)
- Deep Belief Network
- Graph neural network

Machine Learning

Advanced Neural Networks: Plan

1. **Neuron**
   - Network
   - Activation functions
   - Cost functions
   - Optimization function
2. **Feedforward Neural Networks (FFNN)**
   - Multi-layers architecture
   - Auto-encoders
   - Convolutional Neural Network (CNN)
   - Regularization
3. **Recurrent Neural Networks (RNN)**
   - Architecture (RNN)
   - Long Short-Term Memory (LSTM)
   - Gated Recurrent Unit (GRU)
4. **Attention**
   - Attention mechanism
   - Multi-Head Attention
   - Self-attention
   - Transformer

Section 1

Neuron

Advanced Neural Networks

Neuron

- Let's remember **Logistic regression**
- We make a linear combination of features' values: this can be seen as the accumulation of signals

$$z(x) = \theta_0 + \sum_{j=1}^{N} \theta_j x_j$$

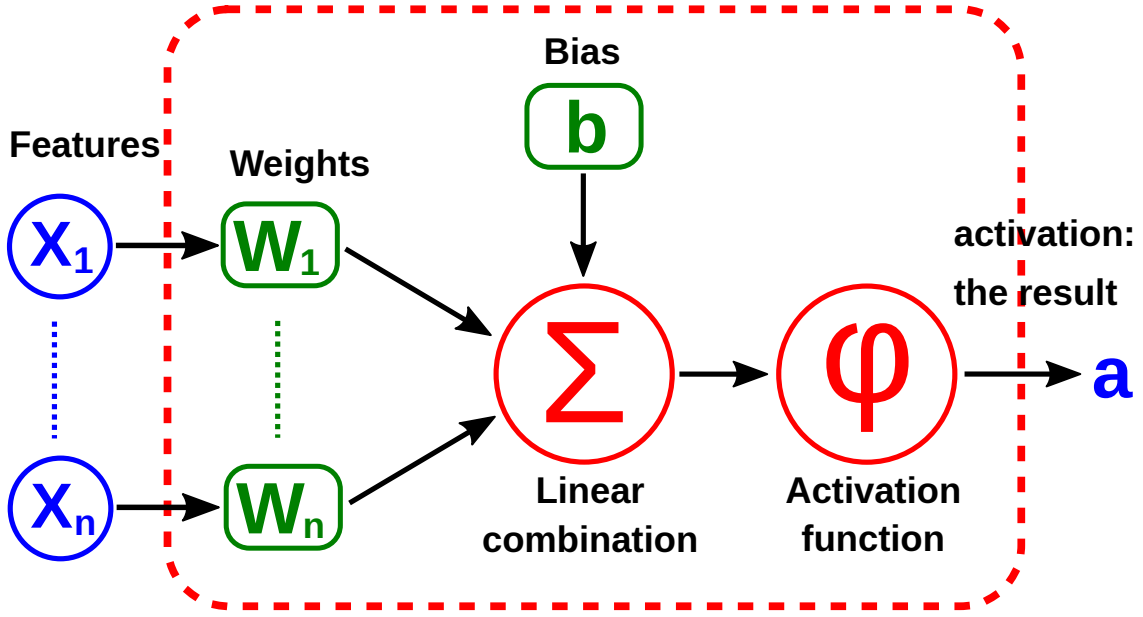
- Then, we apply the logistic function to this sum: we call it an ***activation function*** (send the signal or not according to a threshold)

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

- To calculate the classification error, we use a ***cost function***
- To train the parameters, we use an ***optimization function***

Advanced Neural Networks

Neuron

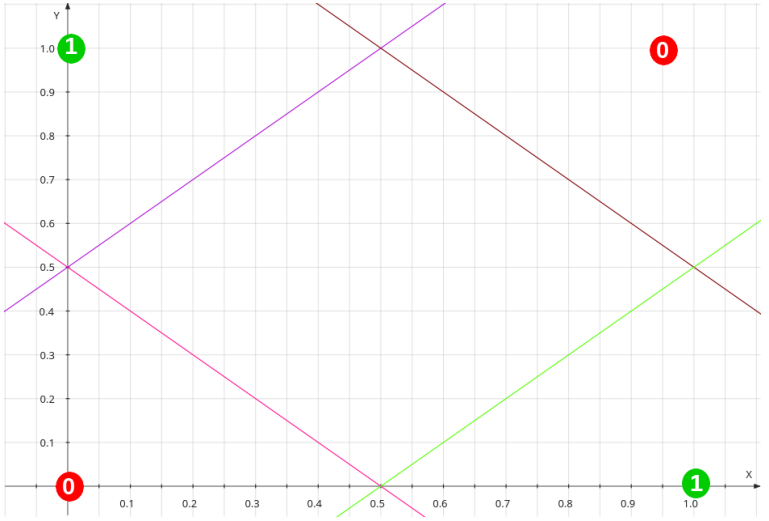


$$\hat{y} = \varphi(b + \sum_{j=1}^{n} X_j W_j)$$

Advanced Neural Networks

Neuron: Motivation

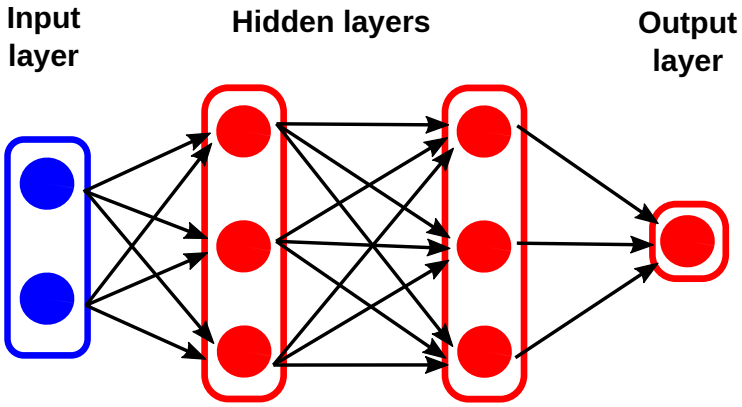
- To train complex functions, a single neuron is not enough
- For example, Two variables' XOR function
- If we train a logistic regression model (a neuron with the logistic function as activation function), the algorithm tries to separate the samples with a decision line
- Whatever the decision line, there is at least one misclassified sample



Advanced Neural Networks: Neuron

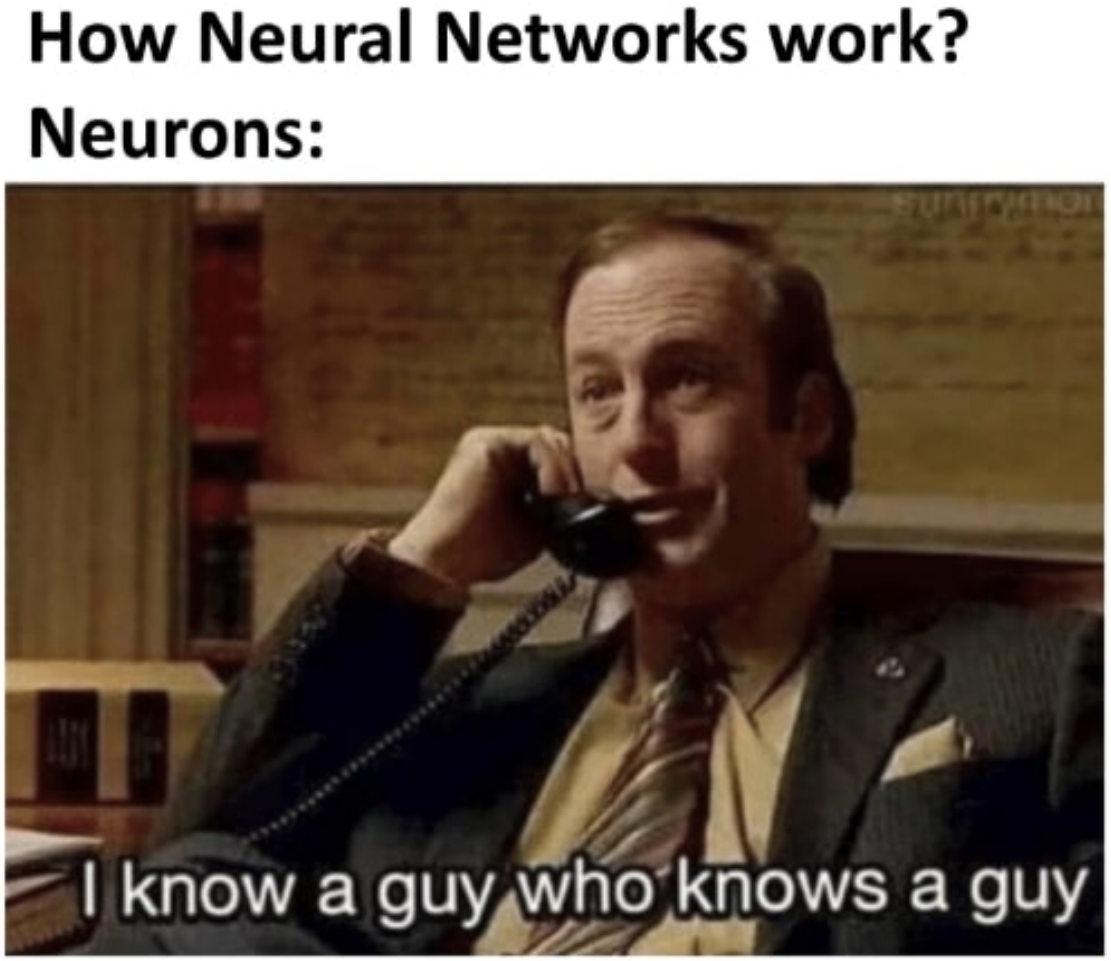
Network: Architecture

- **Input layer**: contains input values and no parameters
- **Hidden layers**: each layer receives the outputs of the previous layer. It contains a number of neurons which calculate the weighted sum with an activation function.
- **Output layer**: contains the neurons that calculate the output.



Advanced Neural Networks: Neuron

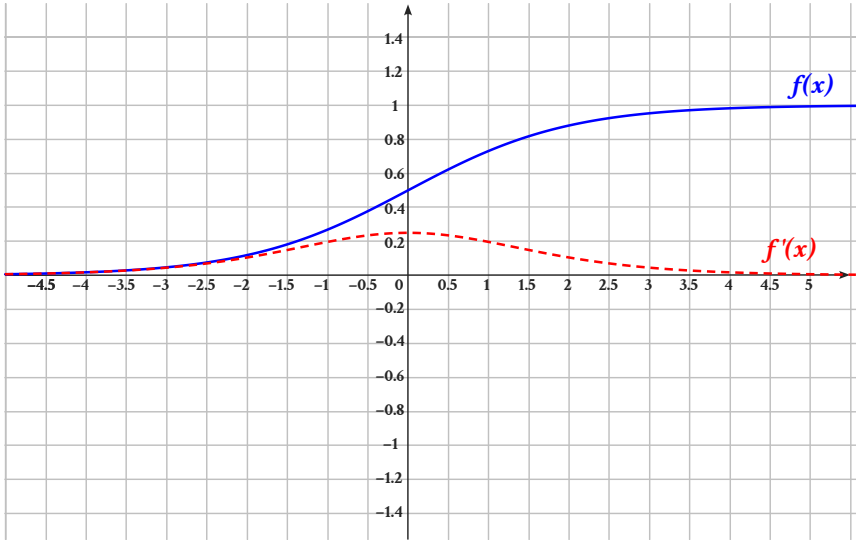
Network: Some humor



Advanced Neural Networks: Neuron

Activation functions: Sigmoid functions

Logistic function

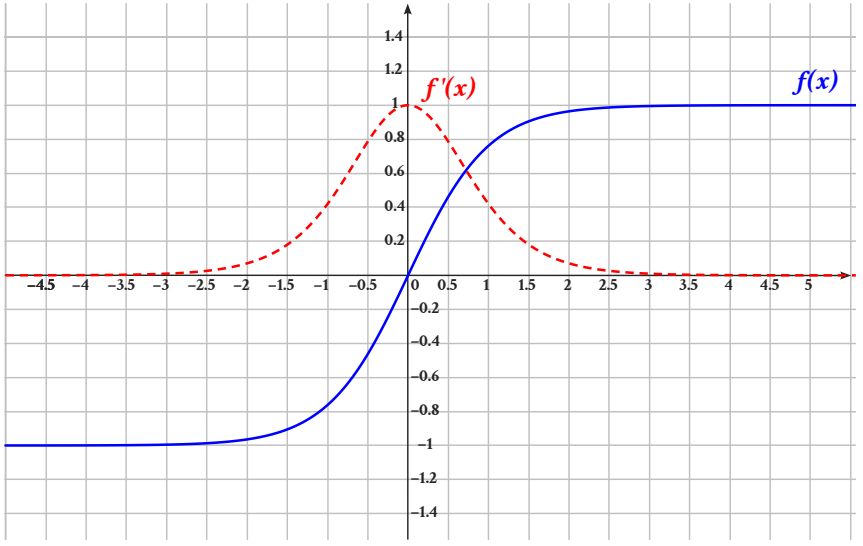


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma(x) \in]0, 1[$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Hyperbolic tangent



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

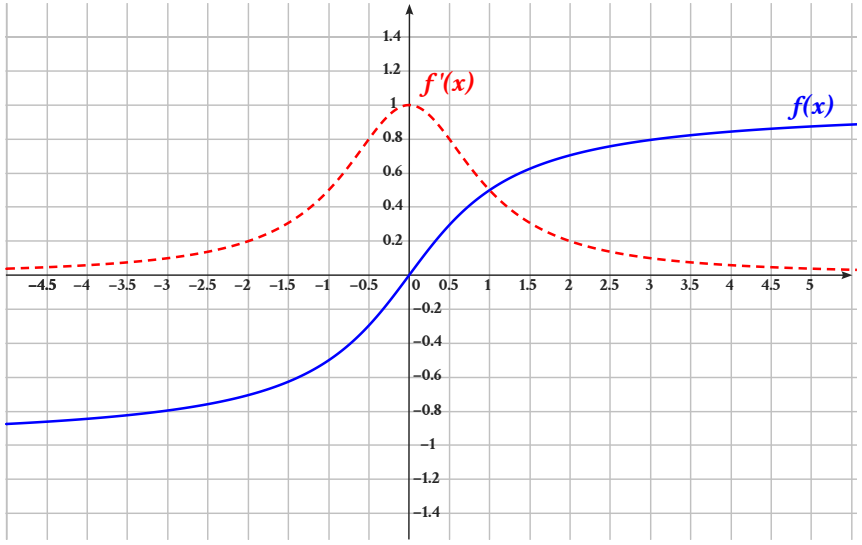
$$tanh(x) \in]-1, 1[$$

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

Advanced Neural Networks: Neuron

Activation functions: Sigmoid functions (2)

Arc tangent

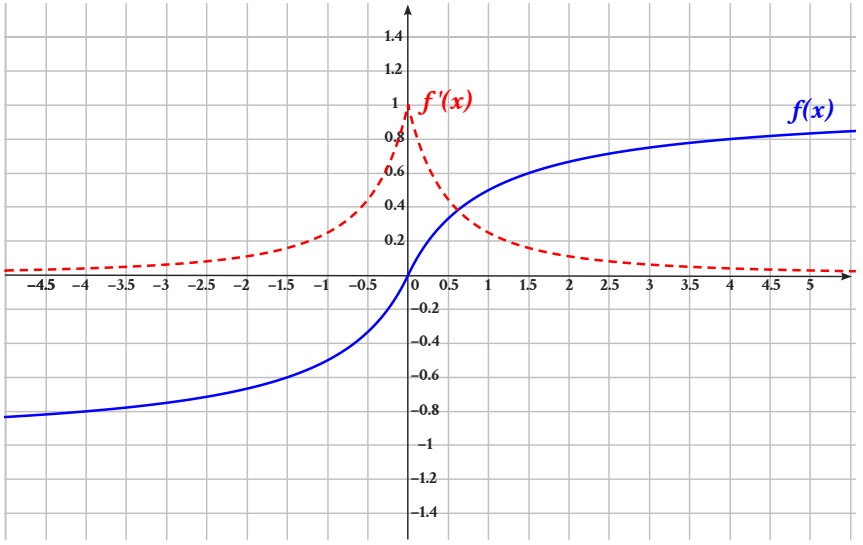


$$f(x) = \frac{2}{\pi} tan^{-1}(x)$$

$$f(x) \in]-1, 1[$$

$$f'(x) = \frac{2}{\pi} \frac{1}{x^2+1}$$

Softsign



$$f(x) = \frac{x}{1+|x|}$$

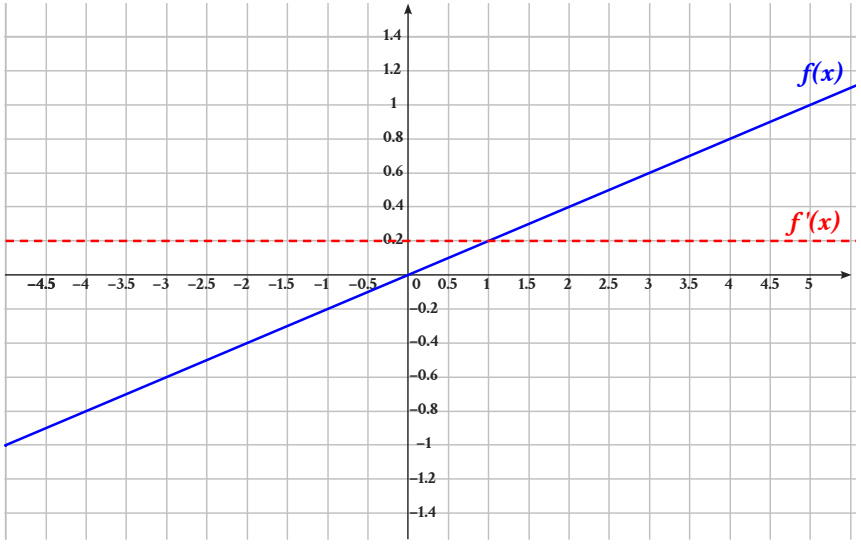
$$f(x) \in]-1, 1[$$

$$f'(x) = \frac{1}{(1+|x|)^2}$$

Advanced Neural Networks: Neuron

Activation functions: Linear functions

Linear function

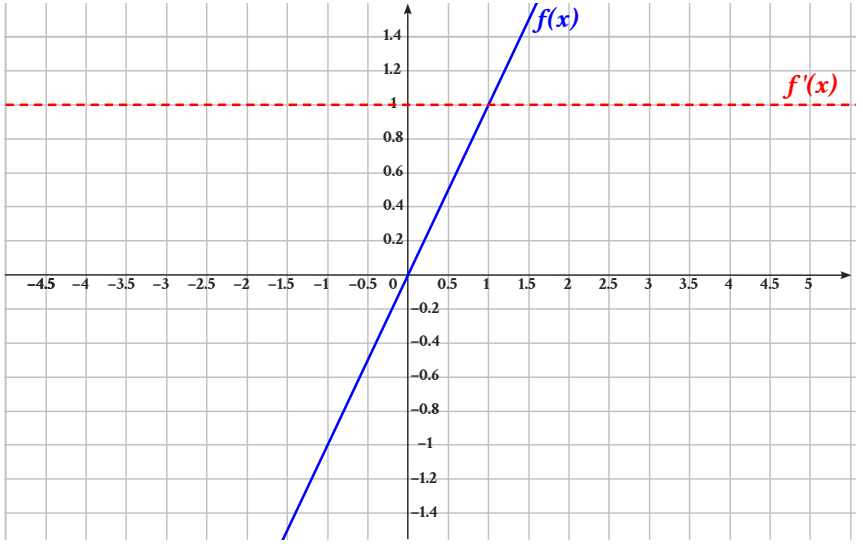


$$f(x) = kx$$

$$f(x) \in]-\infty, +\infty[$$

$$f'(x) = k$$

Identity function



$$f(x) = x$$

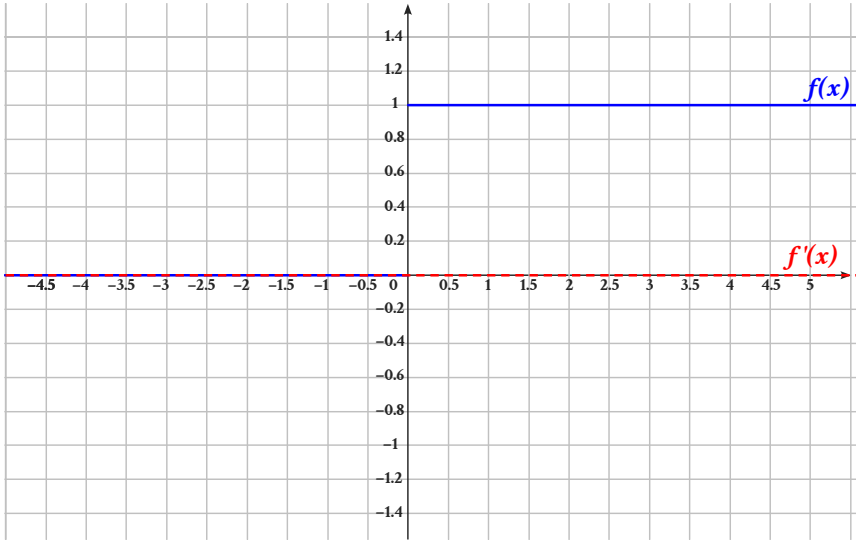
$$f(x) \in]-\infty, +\infty[$$

$$f'(x) = 1$$

Advanced Neural Networks: Neuron

Activation functions: Step functions

Heaviside function

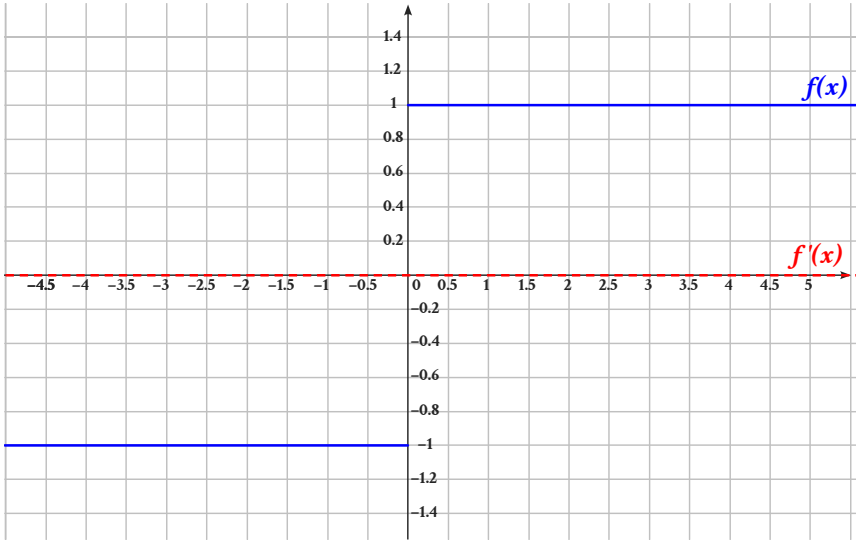


$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$H(x) \in \{0, 1\}$$

$$H'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ +\infty & \text{otherwise} \end{cases}$$

Sign function



$$S(x) = \begin{cases} -1 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases} = 2H(x) - 1$$

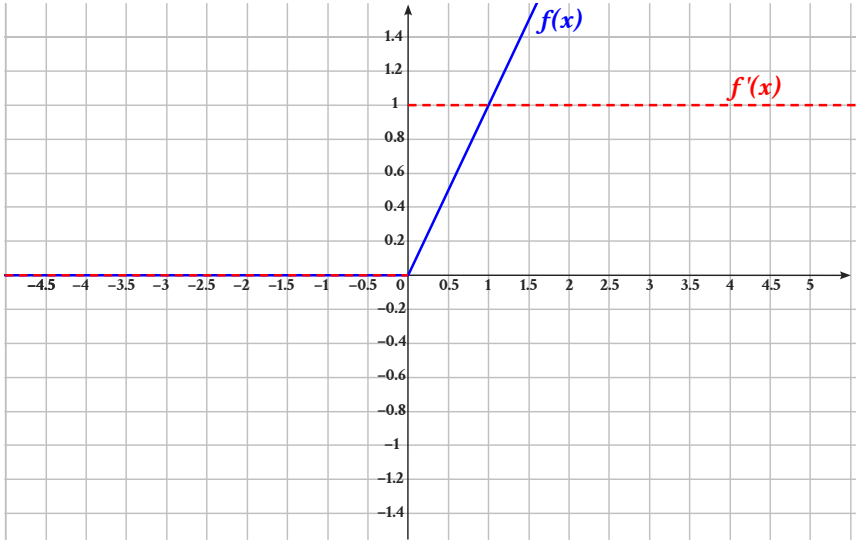
$$S(x) \in \{-1, 1\}$$

$$S'(x) = 2H'(x)$$

Advanced Neural Networks: Neuron

Activation functions: Hockey stick functions

Rectified Linear Unit

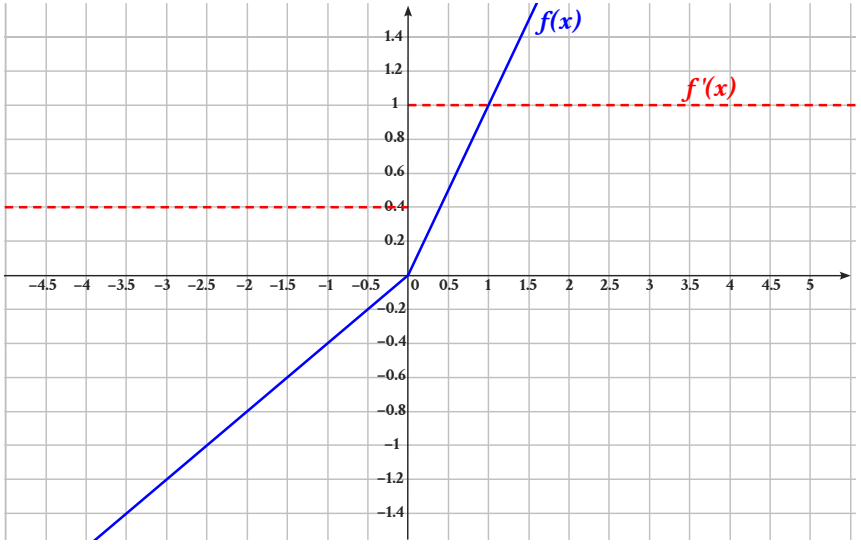


$$ReLU(x) = xH(x) = \max(0, x)$$

$$ReLU(x) \in [0, +\infty[$$

$$ReLU'(x) = H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

Parametric ReLU



$$PReLU(\alpha, x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

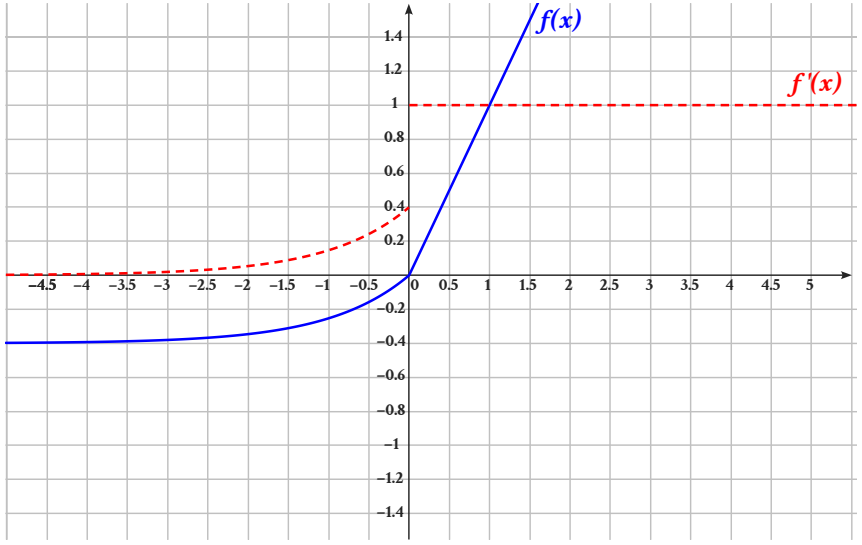
$$PReLU(x) \in]-\infty, +\infty[$$

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

Advanced Neural Networks: Neuron

Activation functions: Hockey stick functions (2)

Exponential Linear Unit

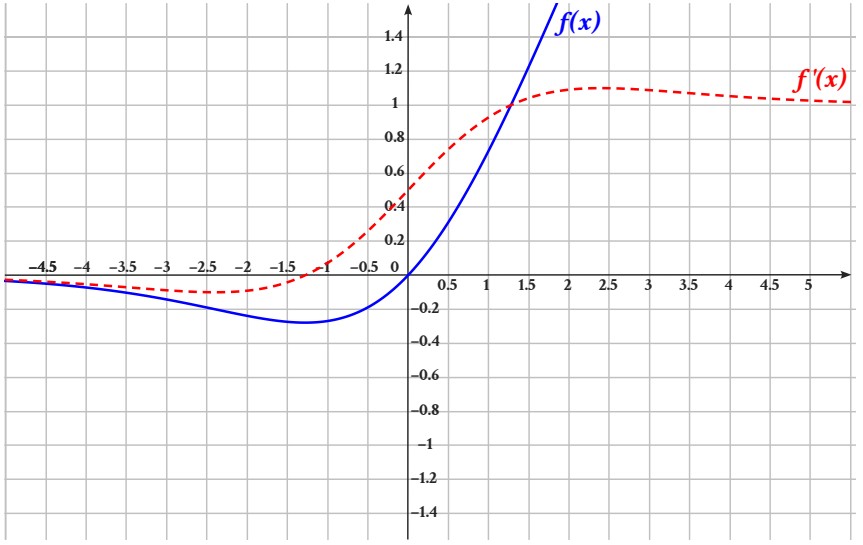


$$ELU(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

$$ELU(\alpha, x) \in]-\infty, +\infty[$$

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

Sigmoid Linear Unit



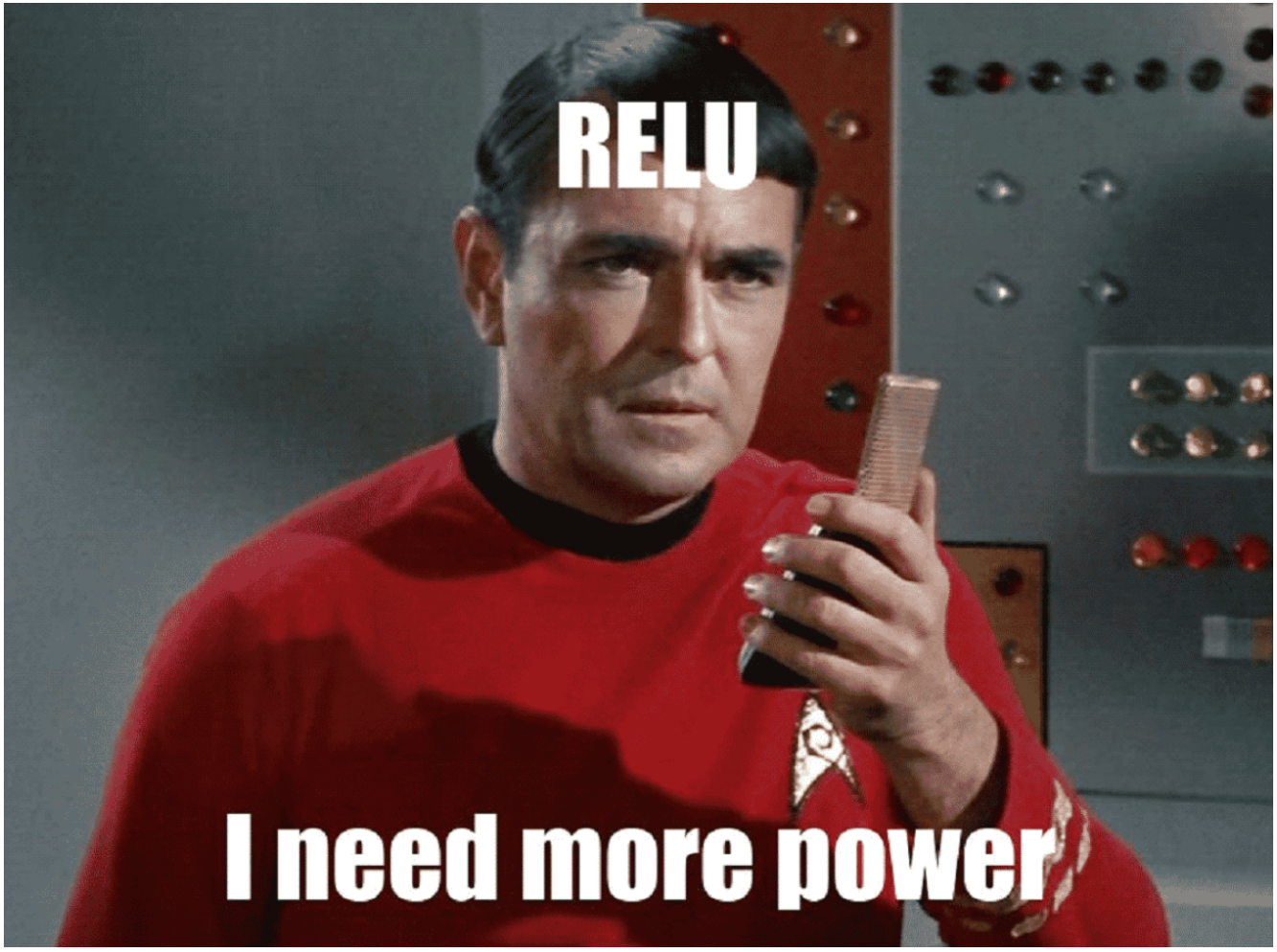
$$SLU(x) = x\sigma(x) = \frac{x}{1+e^{-x}}$$

$$SLU(x) \in]-\infty, +\infty[$$

$$SLU'(x) = \sigma(x)(1 + x\sigma(-x))$$

Advanced Neural Networks: Neuron

Activation functions: Some humor



Advanced Neural Networks: Neuron

Cost functions: Regression (MSE)

Mean Squared Error, L2 loss

$$MSE = \frac{1}{2}(y - \hat{y})^2$$

$$\frac{\partial MSE}{\partial \hat{y}} = \hat{y} - y$$

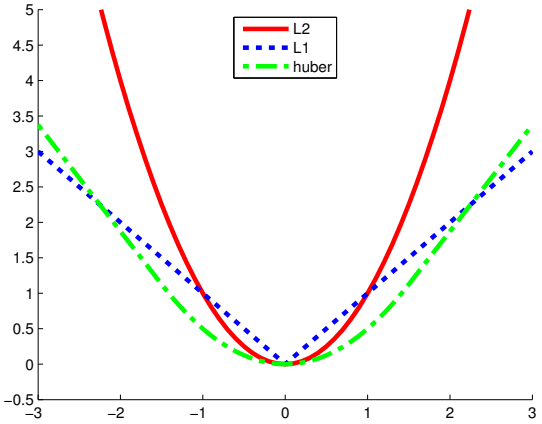


Figure: Regression cost functions **[Rosenberg, 2017]**

Advanced Neural Networks: Neuron

Cost functions: Regression (MSE)

- **Advantages**
  - A quadratic function has only one global minimum.
- **Disadvantages**
  - MSE is less robust to outliers.

Advanced Neural Networks: Neuron

Cost functions: Regression (MAE)

Mean Absolute Error, L1 loss

$$MAE = |y - \hat{y}|$$

$$\frac{\partial MAE}{\partial \hat{y}} = \begin{cases} +1 & \text{if } \hat{y} > y \\ -1 & \text{if } \hat{y} < y \\ [-1, +1] & \text{if } \hat{y} = y \end{cases}$$

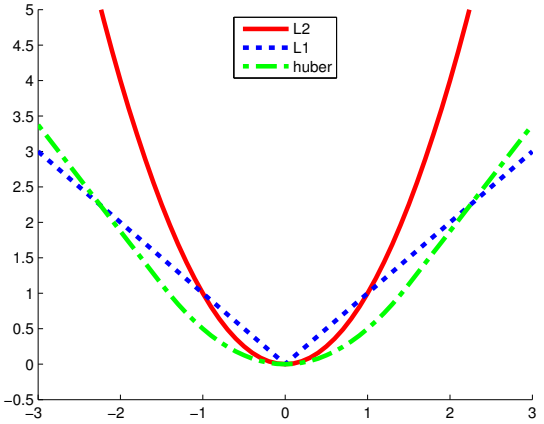


Figure: Regression cost functions **[Rosenberg, 2017]**

Advanced Neural Networks: Neuron

Cost functions: Regression (MAE)

- **Advantages**
  - Less sensitive to outliers
- **Disadvantages**
  - The amplitude of the gradient does not depend on the size of the error, only on the sign of y - ŷ. This leads to the gradient magnitude being large even when the error is small

Advanced Neural Networks: Neuron

Cost functions: Regression (Huber)

Huber Loss

$$J = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

$$\frac{\partial J}{\partial \hat{y}} = \begin{cases} \hat{y} - y & \text{if } |\hat{y} - y| \leq \delta \\ -\delta & \text{if } \hat{y} - y < -\delta \\ +\delta & \text{if } \hat{y} - y > \delta \end{cases}$$

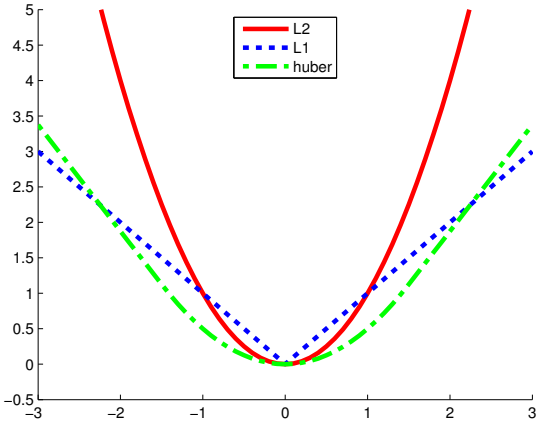


Figure: Regression cost functions **[Rosenberg, 2017]**

Advanced Neural Networks: Neuron

Cost functions: Regression (Huber)

- **Advantages**
  - Same advantages as MSE and MAE
- **Disadvantages**
  - Hyper-parameter δ must be tuned

Advanced Neural Networks: Neuron

Cost functions: Binary classification (BCE)

Binary Cross Entropy Loss

$$J = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y} - \hat{y}^2}$$

- **Advantages**
  - works well with activation functions that model probabilities (sigmoid)
  - Therefore, best estimate of the probabilities
- **Disadvantages**
  - linear separation margin is not guaranteed to be optimal

Advanced Neural Networks: Neuron

Cost functions: Binary classification (Hinge)

Hinge Loss (SVM Loss)

$$J = \max(0, 1 - y\hat{y}), y \in \{-1, 1\}$$

$$\frac{\partial J}{\partial \hat{y}} = \begin{cases} 0 & \text{if } y\hat{y} \geq 1 \\ -y & \text{otherwise} \end{cases}$$

- **Advantages**
  - results in better precision
- **Disadvantages**
  - non-smooth and non-differentiable, therefore fewer optimization algorithms
  - does not model the probability $p(y|x)$ directly

Advanced Neural Networks: Neuron

Cost functions: Multi-class classification (Cross entropy)

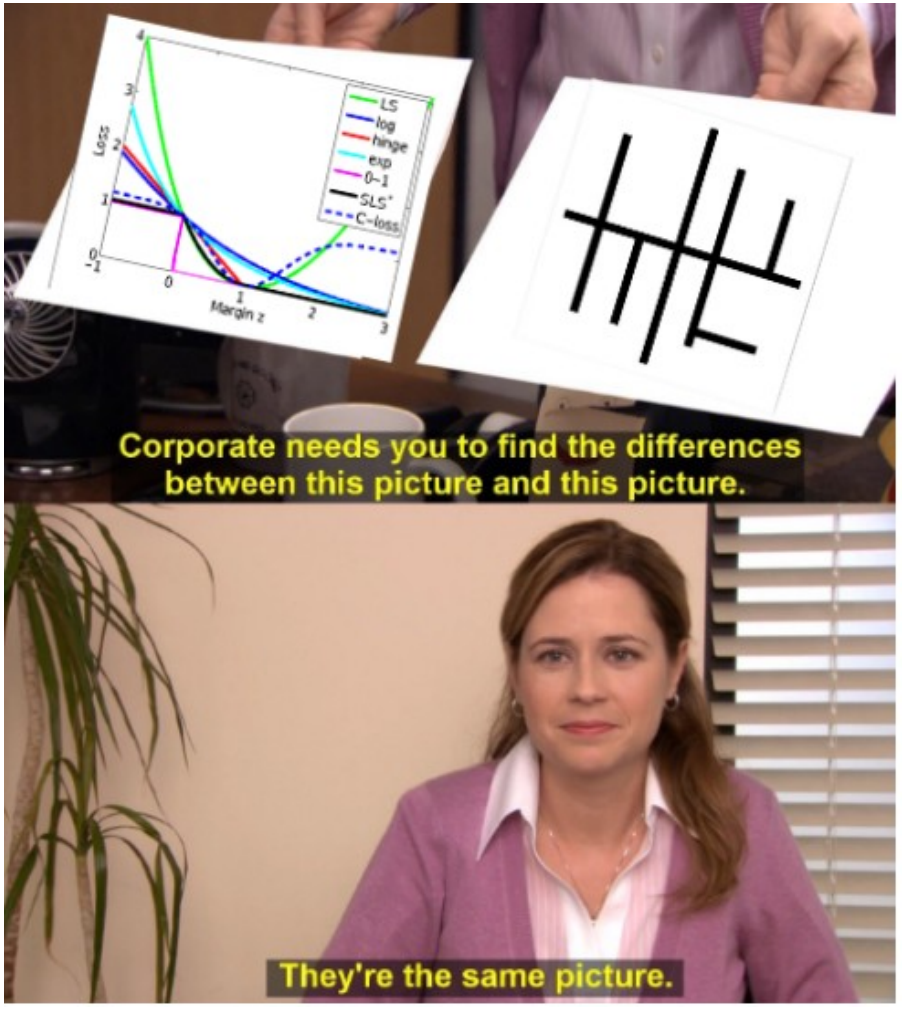
Cross Entropy Loss

$$J = -\sum_{k=1}^{K} y_k \log(\hat{y}_k), \text{ où } K \text{ est le nnombre des classes}$$

$$\frac{\partial J}{\partial \hat{y}_k} = -\frac{y_k}{\hat{y}_k}$$

Advanced Neural Networks: Neuron

Cost functions: Some humor



Advanced Neural Networks: Neuron

Optimization function: Gradient Descent

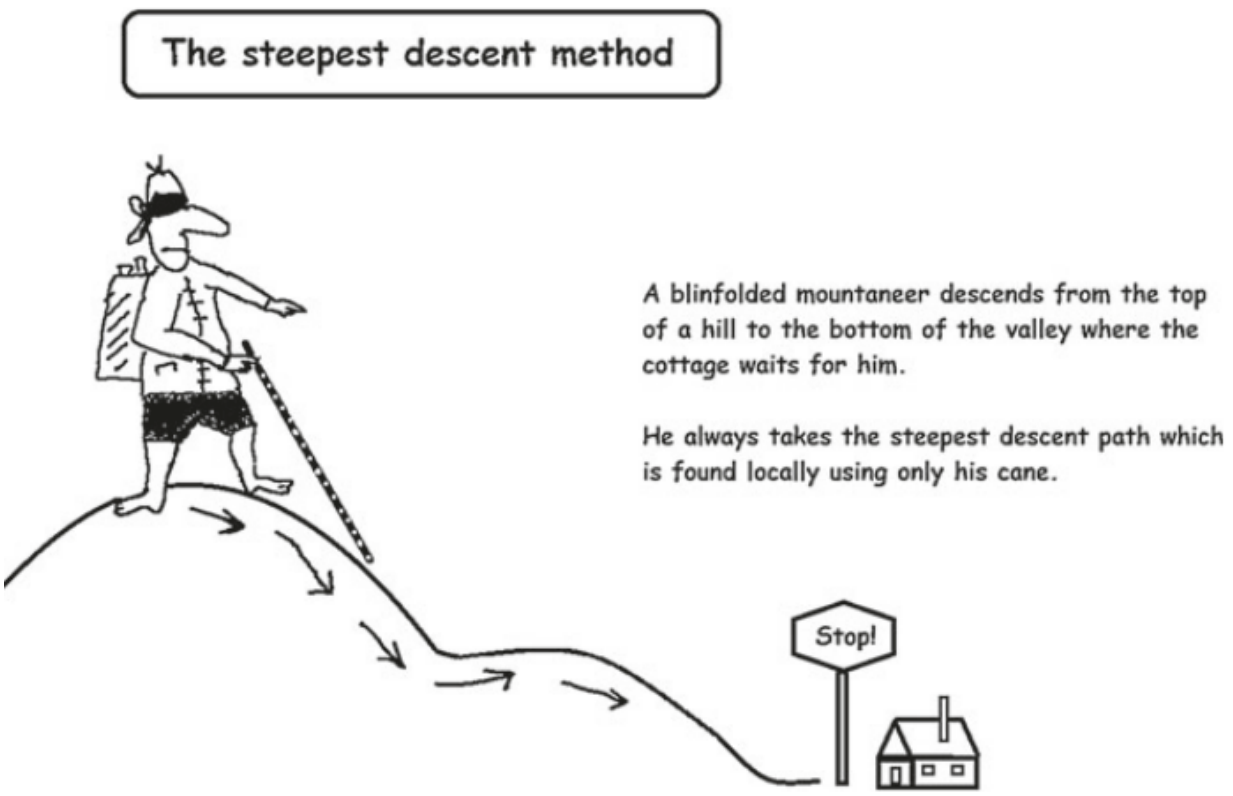


Figure: Illustration of gradient descent **[Calin, 2020]**

Advanced Neural Networks: Neuron

Optimization function: Gradient Descent

Data: X, Y, α, T
Result: θ
initialize θ ; $t = 0$;
while $t < T$ *and no convergence* **do**
    $\theta = \theta - \alpha \Delta_\theta J(X, Y; \theta)$;
    t = t + 1 ;
end

Algorithm 1: Gradient Descent

Advanced Neural Networks: Neuron

Optimization function: Gradient Descent

- **Properties**:
  - Solve the optimal value in the direction of gradient descent.
  - The method converges with a linear manner.
- **Advantages**:
  - The solution is a global optimum when the objective function is convex.
- **Disadvantages**:
  - For each parameter update, the gradients of all samples must be calculated. So the computational cost is very high.
  - The entire workout may be too large to process in memory
  - can converge to a local minimum when the objective function is not convex

Advanced Neural Networks: Neuron

Optimization function: Stochastic Gradient Descent

Data: X, Y, α, T
Result: θ
initialize θ ; $t = 0$;
while $t < T$ *and no convergence* **do**
    Randomly choose $(X^{(i)}, Y^{(i)}) \in (X, Y)$;
    $\theta = \theta - \alpha \Delta_\theta J(X^{(i)}, Y^{(i)}; \theta)$;
    t = t + 1 ;
end

Algorithm 2: Stochastic Gradient Descent

Advanced Neural Networks: Neuron

Optimization function: Stochastic Gradient Descent

- **Properties**:
  - Parameters are updated based on each sample
- **Advantages**:
  - Can avoid local minima
- **Disadvantages**:
  - It is difficult for the model to converge to a minimum

Advanced Neural Networks: Neuron

Optimization function: Mini-Batch Gradient Descent

Data: X, Y, α, T, b
Result: θ
initialize θ ; $t = 0$;
while $t < T$ *et pas de convergence* **do**
    Randomly shuffle the data;
    Split the data into b batches;
    foreach $(X^b, Y^b) \in batches$ **do** $\theta = \theta - \alpha \Delta_\theta J(X^b, Y^b; \theta)$;
    // We can update the parameters with the average of the steps
    t = t + 1 ;
end

Algorithm 3: Mini-Batch Gradient Descent

Advanced Neural Networks: Neuron

Optimization function: Mini-Batch Gradient Descent

- **Properties**:
  - Training is done on subsets of the dataset
  - A hyper-parameter b for the size of a batch
- **Advantages**:
  - Can avoid local minima due to frequent updates
  - Effective in case of large training data
- **Disadvantages**:
  - An additional hyper-parameter must be configured

Advanced Neural Networks: Neuron

Optimization function: Adaptive Gradient Descent (AdaGrad)

Data: X, Y, α, T
Result: θ
initialize θ ; $t = 0$;
initialize v ($|v| = |\theta|$) to zero;
while $t < T$ *and no convergence* **do**
    $v = v + (\Delta_\theta J(X, Y; \theta))^2$;
    $\theta = \theta - \frac{\alpha}{\sqrt{v+\epsilon}} \Delta_\theta J(X, Y; \theta)$;// epsilon is used to avoid division by 0, typically 1e-8
    t = t + 1 ;
end

Algorithm 4: Adaptive Gradient Descent **[Duchi et al., 2011]**

Advanced Neural Networks: Neuron

Optimization function: Adaptive Gradient Descent (AdaGrad)

- **Properties**:
  - The learning rate is adaptively adjusted based on the sum of squares of all historical gradients.
- **Advantages**:
  - The learning rate of each parameter adjusts adaptively
- **Disadvantages**:
  - Over time, the learning rate falls toward zero; the algorithm is not desirable for non-convex functions.

Advanced Neural Networks: Neuron

Optimization function: Root Mean Square Propagation (RMSProp)

Data: X, Y, α, T, β
Result: θ
initialize θ ; $t = 0$;
initialize v ($|v| = |\theta|$) to zero;
while $t < T$ *and no convergence* **do**
  $v = \beta v + (1 - \beta)(\Delta_\theta J(X, Y; \theta))^2$;
  $\theta = \theta - \frac{\alpha}{\sqrt{v}+\epsilon} \Delta_\theta J(X, Y; \theta)$;
  t = t + 1 ;
end

Algorithm 5: RMSProp **[Hinton et al., 2014]**

Advanced Neural Networks: Neuron

Optimization function: Root Mean Square Propagation (RMSProp)

- **Properties**:
  - Change the total gradient accumulation mode to exponential moving average.
  - By default, $\beta = 0.9$
- **Advantages**:
  - Addresses the inefficient learning problem in late AdaGrad.
  - Suitable for the optimization of non-stationary and non-convex problems.
- **Disadvantages**:
  - At the end of training, the update process can be stuck around the local minimum.

Advanced Neural Networks: Neuron

Optimization function: Adaptive Moment Estimation (Adam)

Data: $X, Y, \alpha, T, \beta_1, \beta_2$
Result: θ
initialize θ ; $t = 0$;
initialize v et m ($|v| = |m| = |\theta|$) to zero;
while $t < T$ *and no convergence* **do**
  $g = \Delta_\theta J(X, Y; \theta)$;
  $m = \beta_1 m + (1 - \beta_1) g$; $\hat{m} = \frac{m}{1-\beta_1^t}$;
  $v = \beta_2 v + (1 - \beta_2) g^2$; $\hat{v} = \frac{v}{1-\beta_2^t}$;
  $\theta = \theta - \frac{\alpha \hat{m}}{\sqrt{\hat{v}} + \epsilon}$;
  t = t + 1 ;
end

Algorithm 6: Adam **[Kingma and Ba, 2015]**

Advanced Neural Networks: Neuron

Optimization function: Adaptive Moment Estimation (Adam)

- **Properties**:
  - Use of first-order and second-order moments to dynamically adjust the learning rate for each parameter.
  - Addition of bias correction.
  - Default values: $\beta_1 = 0.9, \beta_2 = 0.999$
- **Advantages**:
  - The gradient descent process is relatively stable.
  - It is suitable for most non-convex optimization problems with large datasets and high-dimensional spaces.
- **Disadvantages**:
  - The method may not converge in certain cases.

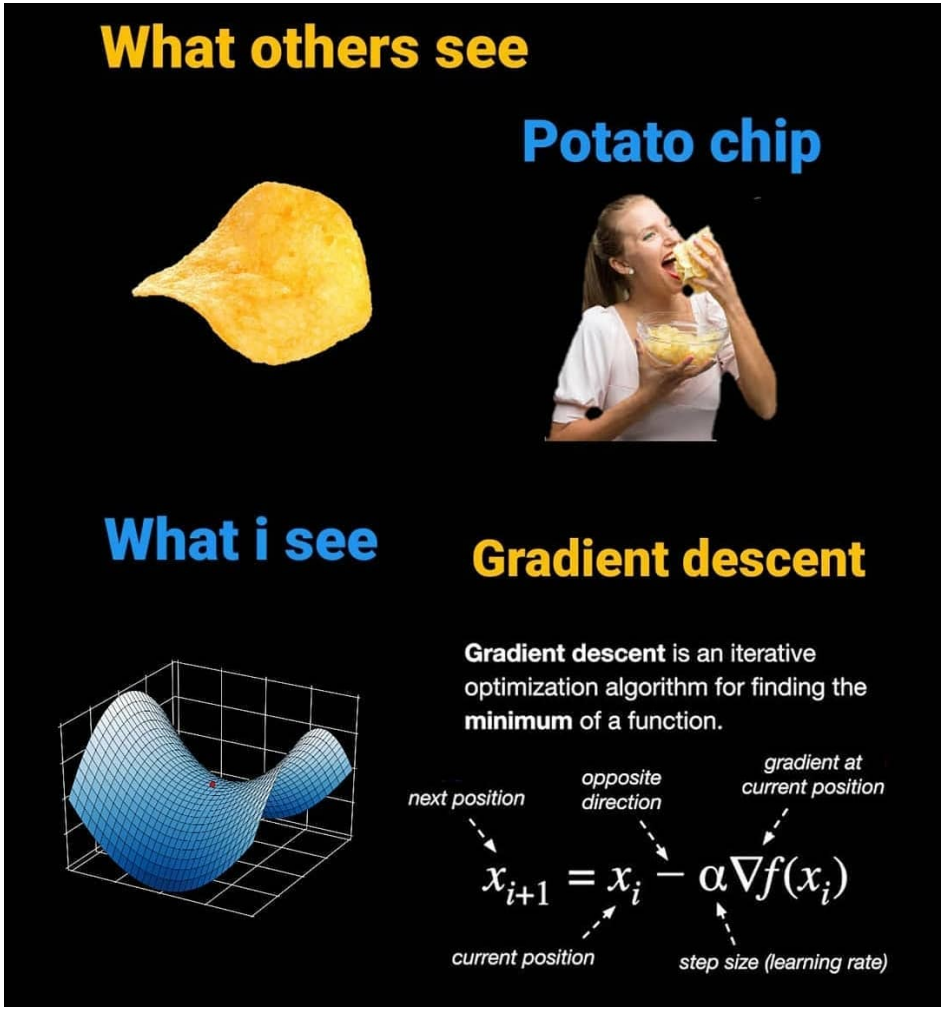
Advanced Neural Networks: Neuron

Optimization function: Others

- **COCOB [Orabona and Tommasi, 2017]**
  - https://arxiv.org/pdf/1705.07795.pdf
- **Yogi [Zaheer et al., 2018]**
  - https://proceedings.neurips.cc/paper/2018/file/90365351ccc7437a1309dc64e4db32a3-Paper.pdf
- **NovoGrad [Ginsburg et al., 2019]**
  - https://arxiv.org/pdf/1905.11286.pdf
- **Lookahead [Zhang et al., 2019]**
  - https://arxiv.org/pdf/1907.08610v1.pdf
- **LAMB [You et al., 2020]**
  - https://arxiv.org/abs/1904.00962
- **AdaBelief [Zhuang et al., 2020]**
  - https://arxiv.org/pdf/2010.07468.pdf

Advanced Neural Networks: Neuron

Optimization function: Some humor



Section 2

Feedforward Neural Networks (FFNN)

Advanced Neural Networks

Feedforward Neural Networks (FFNN)

Definition

A "Feedforward Neural Network" is an architecture of Neural Networks. In this one, there is no "feed" back from the outputs of the neurons to the inputs throughout the network. **[Sazlı, 2006]**

In other word, there is no temporal phenomena treated in the problem in question; current output does not depend on the past one.

Example

- Predicting the class of an image only based on its pixels

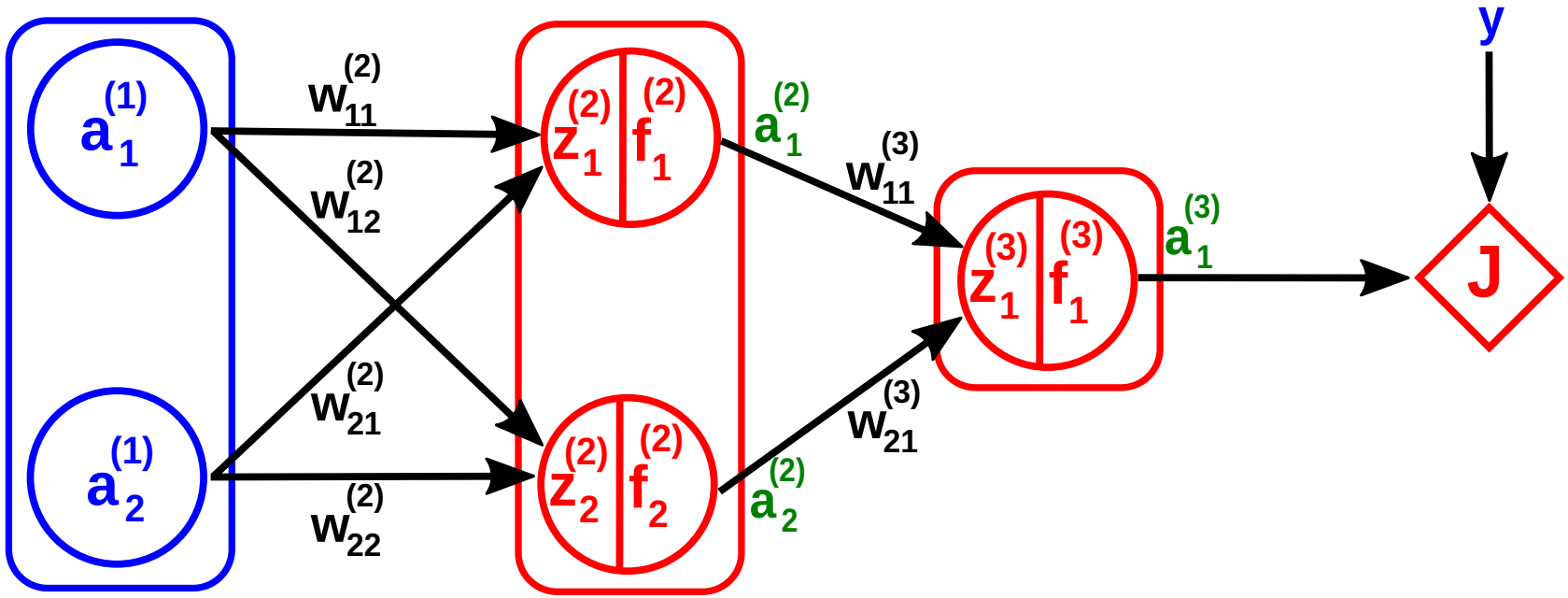
Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Notation

$$z_j^{(l)} = \sum_i w_{ij}^{(l)} a_i^{(l-1)} + b_i^{(l)}$$

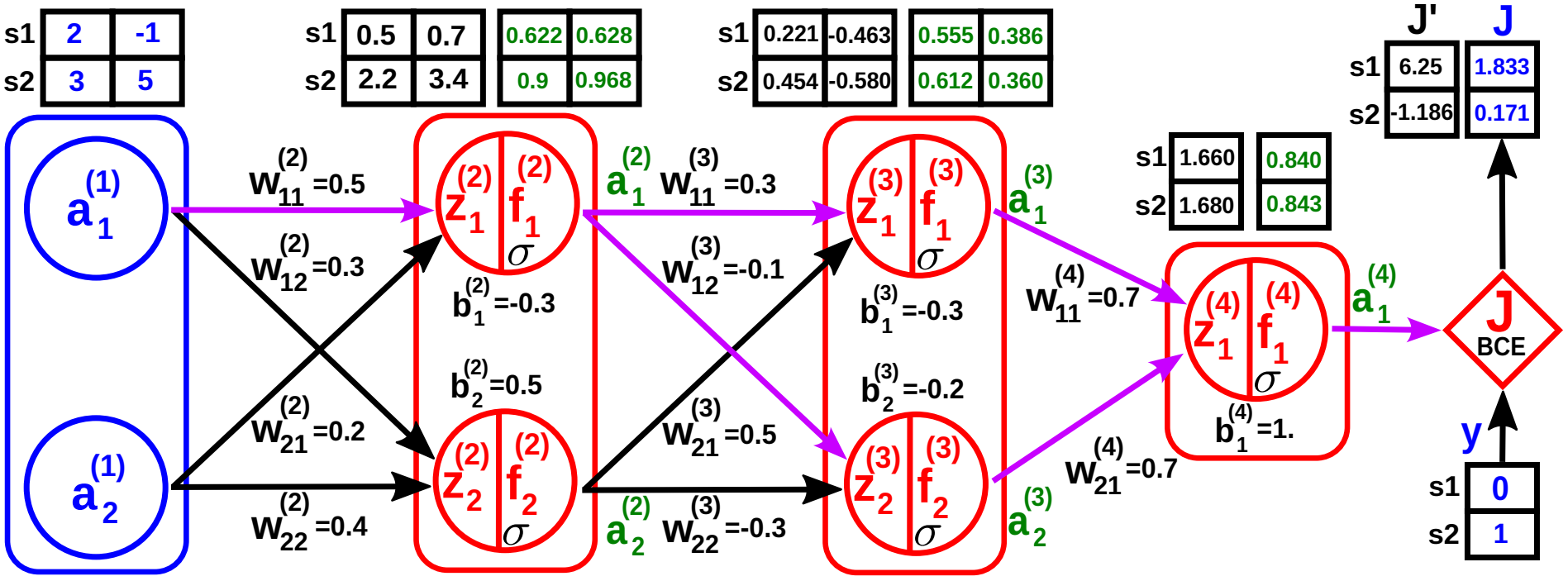
$$a_i^{(1)} = x_i$$

$$a_i^{(l)} = f_i^{(l)}(z_i^{(l)})$$



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Example



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Backpropagation [example] (1)

Update $w_{11}^{(4)}$

$$\frac{\partial J}{\partial w_{11}^{(4)}} = \overbrace{\frac{\partial J}{\partial f_1^{(4)}} \frac{\partial f_1^{(4)}}{\partial z_1^{(4)}}}^{\delta_1^{(4)}} \overbrace{\frac{\partial z_1^{(4)}}{\partial w_{11}^{(4)}}}^{a_1^{(3)}}$$

$$\frac{\partial J}{\partial f_1^{(4)}} = \frac{(0.840,0.843)-(0,1)}{(0.840,0.843)-(0.840,0.843)^2} = (6.25, -1.186)$$

$$\frac{\partial f_1^{(4)}}{\partial z_1^{(4)}} = (0.840, 0.843)(0.160, 0.157) = (0.134, 0.132)$$

$$\delta_1^{(4)} = (6.25, -1.186)(0.134, 0.132) \approx (0.838, -0.157)$$

$$\frac{\partial J}{\partial w_{11}^{(4)}} = avg((0.838, -0.157)(0.555, 0.612)) \approx avg(0.465, -0.096) = 0.184$$

$$w_{11}^{(4)} = 0.7 - 1 * (0.184) = 0.516 \text{ , supposing } \alpha = 1$$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Backpropagation [example] (2)

Update $w_{21}^{(4)}$

$$\frac{\partial J}{\partial w_{21}^{(4)}} = \overbrace{\frac{\partial J}{\partial f_1^{(4)}} \frac{\partial f_1^{(4)}}{\partial z_1^{(4)}}}^{\delta_1^{(4)}} \overbrace{\frac{\partial z_1^{(4)}}{\partial w_{21}^{(4)}}}^{a_2^{(3)}}$$

$\delta_1^{(4)} = (0.838, -0.157)$

$\frac{\partial J}{\partial w_{21}^{(4)}} = avg((0.838, -0.157)(0.386, 0.360)) \approx avg(0.323, -0.056) = 0.134$

$w_{21}^{(4)} = 0.7 - 1 * (0.134) = 0.566$, supposing $\alpha = 1$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Backpropagation [example] (3)

Update $w_{11}^{(3)}$

$$\frac{\partial J}{\partial w_{11}^{(3)}} = \overbrace{\underbrace{\frac{\partial J}{\partial f_1^{(4)}}}_{\delta_1^{(4)}} \underbrace{\frac{\partial f_1^{(4)}}{\partial z_1^{(4)}} \frac{\partial z_1^{(4)}}{\partial f_1^{(3)}}}_{w_{11}^{(4)}} \frac{\partial f_1^{(3)}}{\partial z_1^{(3)}}}^{\delta_1^{(3)}} \underbrace{\frac{\partial z_1^{(3)}}{\partial w_{11}^{(3)}}}_{a_1^{(2)}}$$

Here, we use the past $w_{11}^{(4)}$

$$\frac{\partial f_1^{(3)}}{\partial z_1^{(3)}} \approx (0.555, 0.612)(0.445, 0.388) = (0.247, 0.237)$$

$$\delta_1^{(3)} = (0.838, -0.157) * 0.7 * (0.247, 0.237) \approx (0.145, -0.026)$$

$$\frac{\partial J}{\partial w_{11}^{(2)}} = avg((0.145, -0.026)(0.622, 0.900)) = avg(0.090, -0.023) = 0.033$$

$$w_{11}^{(2)} = 0.3 - 1 * (0.033) = 0.267 \text{ , supposing } \alpha = 1$$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Backpropagation [example] (4)

Update $w_{12}^{(3)}$

$$\frac{\partial J}{\partial w_{12}^{(3)}} = \overbrace{\underbrace{\frac{\partial J}{\partial f_1^{(4)}}}_{\delta_1^{(4)}} \underbrace{\frac{\partial f_1^{(4)}}{\partial z_1^{(4)}} \frac{\partial z_1^{(4)}}{\partial f_2^{(3)}}}_{w_{11}^{(4)}} \frac{\partial f_2^{(3)}}{\partial z_2^{(3)}}}^{\delta_2^{(3)}} \underbrace{\frac{\partial z_2^{(3)}}{\partial w_{12}^{(3)}}}_{a_1^{(2)}}$$

Here, we use use the past $w_{12}^{(4)}$

$$\frac{\partial f_2^{(3)}}{\partial z_2^{(3)}} \approx (0.386, 0.360)(0.614, 0.64) = (0.237, 0.230)$$

$$\delta_2^{(3)} = (0.838, -0.157) * 0.7 * (0.237, 0.230) \approx (0.139, -0.025)$$

$$\frac{\partial J}{\partial w_{11}^{(2)}} = avg((0.139, -0.025)(0.622, 0.900)) = avg(0.086, -0.023) = 0.032$$

$$w_{11}^{(2)} = -0.1 - 1 * (0.032) = -0.132 \text{ , supposing } \alpha = 1$$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Backpropagation [example] (5)

Update $w_{11}^{(2)}$

$$\frac{\partial J}{\partial w_{11}^{(2)}} = \overbrace{\left(\delta_1^{(3)} \overbrace{\frac{\partial z_1^{(3)}}{\partial f_1^{(2)}}}^{w_{11}^{(3)}} + \delta_2^{(3)} \overbrace{\frac{\partial z_2^{(3)}}{\partial f_1^{(2)}}}^{w_{12}^{(3)}}\right)}^{\delta_1^{(2)}} \frac{\partial f_1^{(2)}}{\partial z_1^{(2)}} \overbrace{\frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}}}^{a_1^{(1)}}$$

$$\frac{\partial f_1^{(2)}}{\partial z_1^{(2)}} = (0.622, 0.900)(0.378, 0.100) = (0.235, 0.09)$$

$$\delta_1^{(2)} = ((0.145, -0.026) * (0.3) + (0.139, -0.025) * (-0.1)) * (0.235, 0.09) \approx (0.006956, -0.00047683)$$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Backpropagation [General case]

- $\delta^{(l)}$ are calculated, where l is the layer's order

$$\delta^{(out)} = \frac{\partial J}{\partial f^{(out)}} \frac{\partial f^{(out)}}{\partial z^{(out)}}, \ \delta^{(l)} = \frac{\partial f^{(l)}}{\partial z^{(l)}} w^{(l+1)} \delta^{(l+1)}$$

- Gradients are calculated

$$\frac{\partial J}{\partial w^{(l)}} = a^{(l-1)} \delta^{(l)}, \ \frac{\partial J}{\partial b^{(l)}} = \delta^{(l)}$$

- Parameters are updated

$$w = w - \alpha \frac{\partial J}{\partial w^{(l)}}, \ b = b - \alpha \frac{\partial J}{\partial b^{(l)}}$$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Multi-layers architecture: Backpropagation (Some humor)

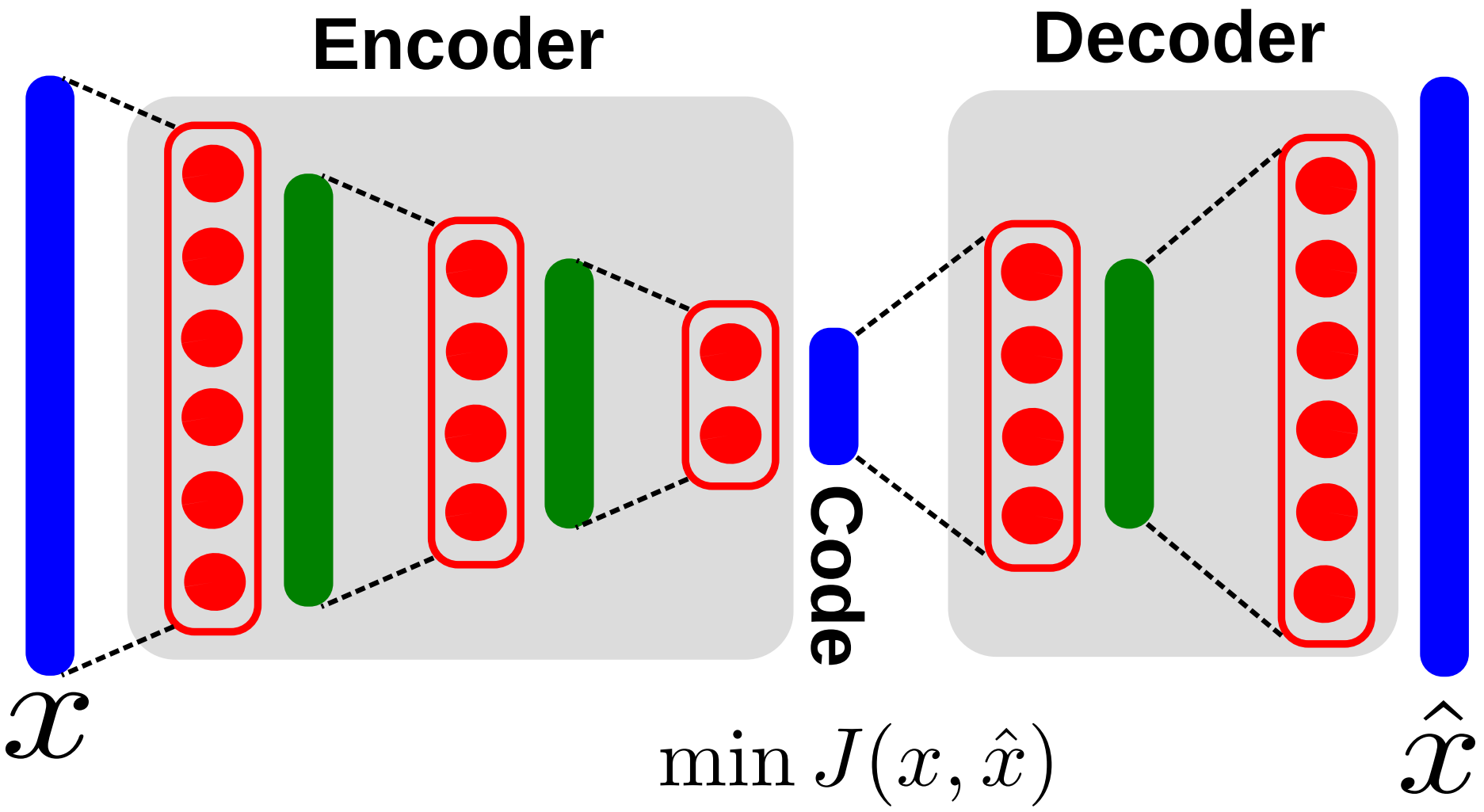
Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Auto-encoders

- It is a multi-layer neural network.
- It learns a compression algorithm.
- It is characterized by **[Chollet, 2016]**:
  - Data-driven: unlike compression algorithms like JPEG.
  - Lossy data: the constructed output is not entirely identical to the input.
  - Unsupervised learning.
- It is not particularly effective for the compression task.

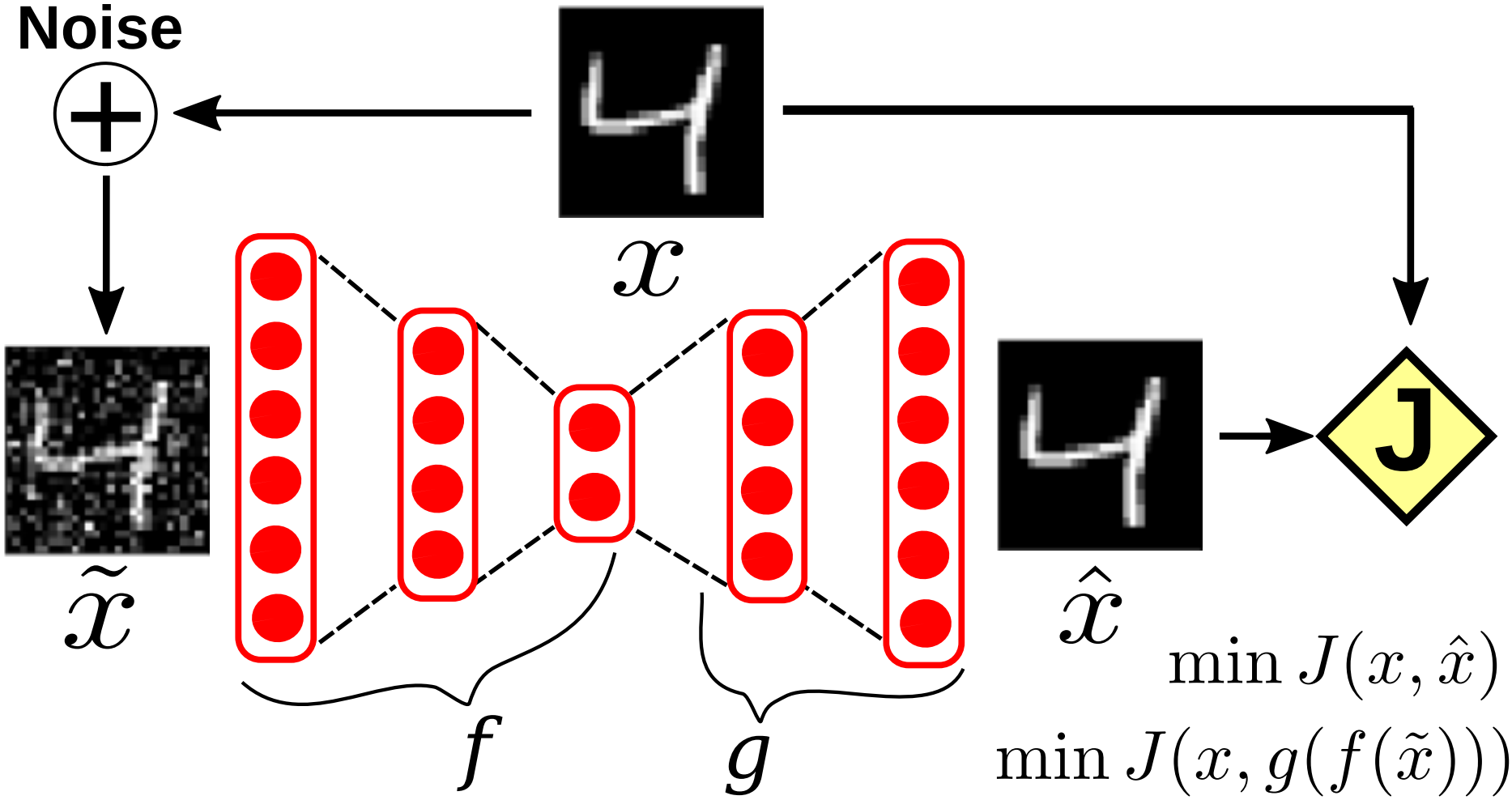
Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Auto-encoders: Architecture



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Auto-encoders: Denoising Auto-encoder



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Auto-encoders: Denoising Auto-encoder (Some applications)

- Improving the quality of speech (audio) **[Lu et al., 2013]**.
- Cleaning dirty documents (refer to this competition: https://www.kaggle.com/c/denoising-dirty-documents).
- Retrieving historical documents **[Neji et al., 2019]**.
- Completing hidden parts of the face **[Li et al., 2017]**.

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Auto-encoders: Sparse Auto-encoder

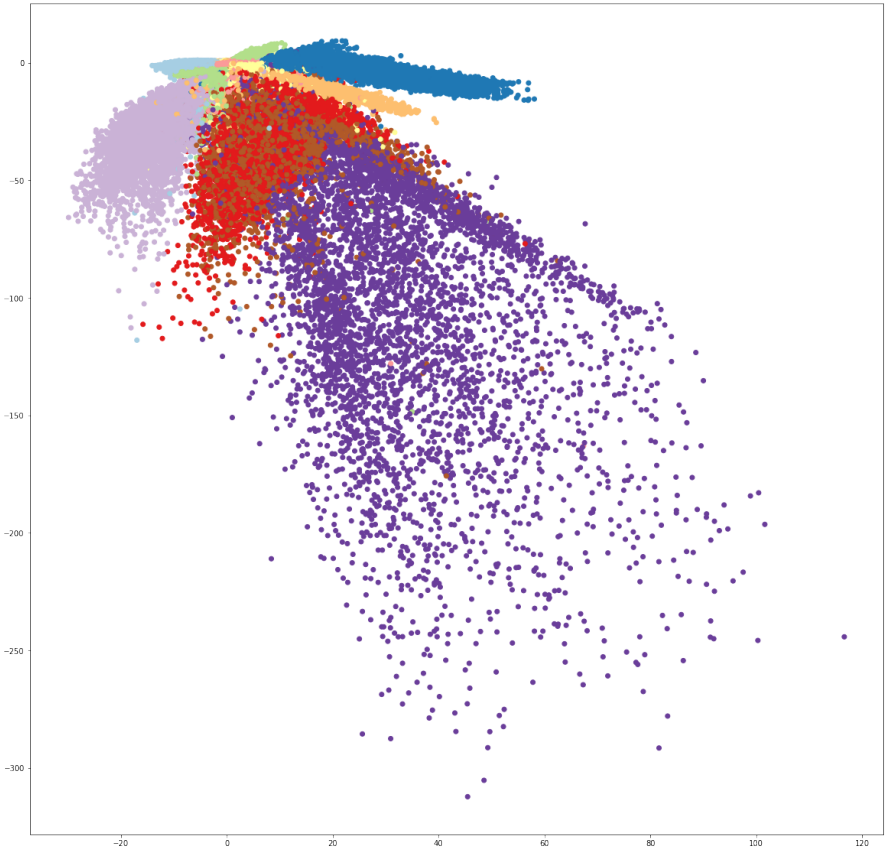
- Size of hidden layers must be equal to or larger than the input size.
- It is used to automatically learn features (feature engineering) to be used in another task, such as classification.
- Parameter (weight) regularization is applied.
- **L1** and **Kullback-Leibler divergence** are used.
- Some parameters converge to zero, allowing the learning of representations such as the contours of an object in an image.

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

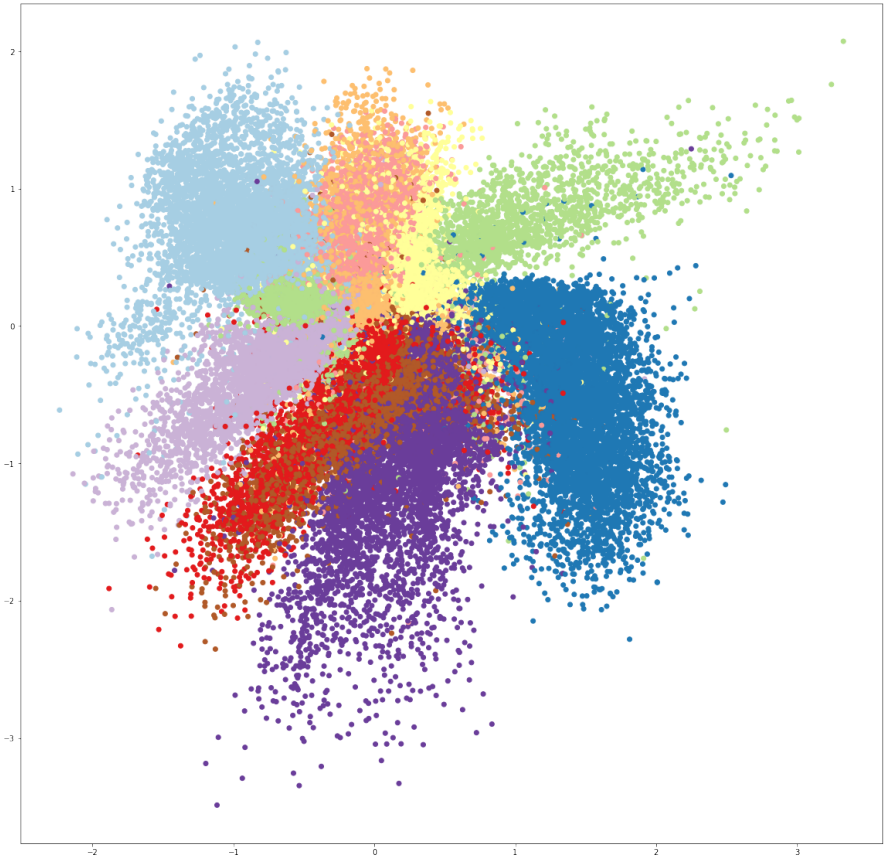
Auto-encoders: Variational Auto-encoder (Motivation)

- https://github.com/projeduc/ESI_ML/blob/main/demos/NN/TF_Autoencoder.ipynb

Clustering Auto-encoder



Variational Auto-encoder



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

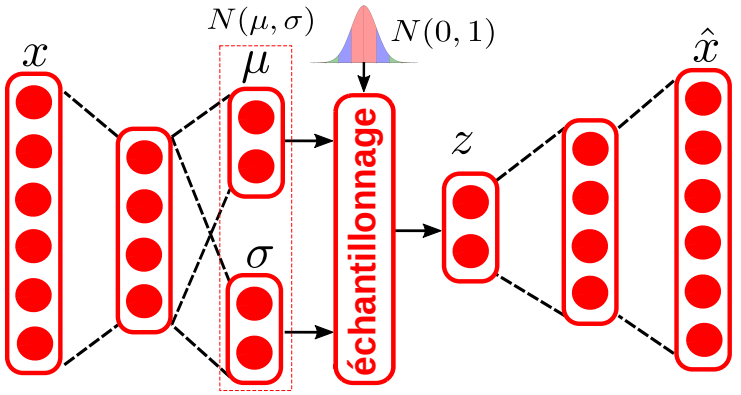
Auto-encoders: Variational Auto-encoder

- To generate new content, some random values can be input into the decoder.
- The problem is that the encoder learns to perfectly separate the different clusters.
- Solution: force the auto-encoder to learn a distribution.

$$z = \mu + \sigma * N(0,1)$$

$$J'(x,\hat{x}) = J(x,\hat{x}) + KL(N(\mu,\sigma), N(0,1))$$

$$KL(p\|q) = \sum_i p(x_i) log(\frac{p(x_i)}{q(x_i)})$$



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Auto-encoders: Variational Auto-encoder (Summary)

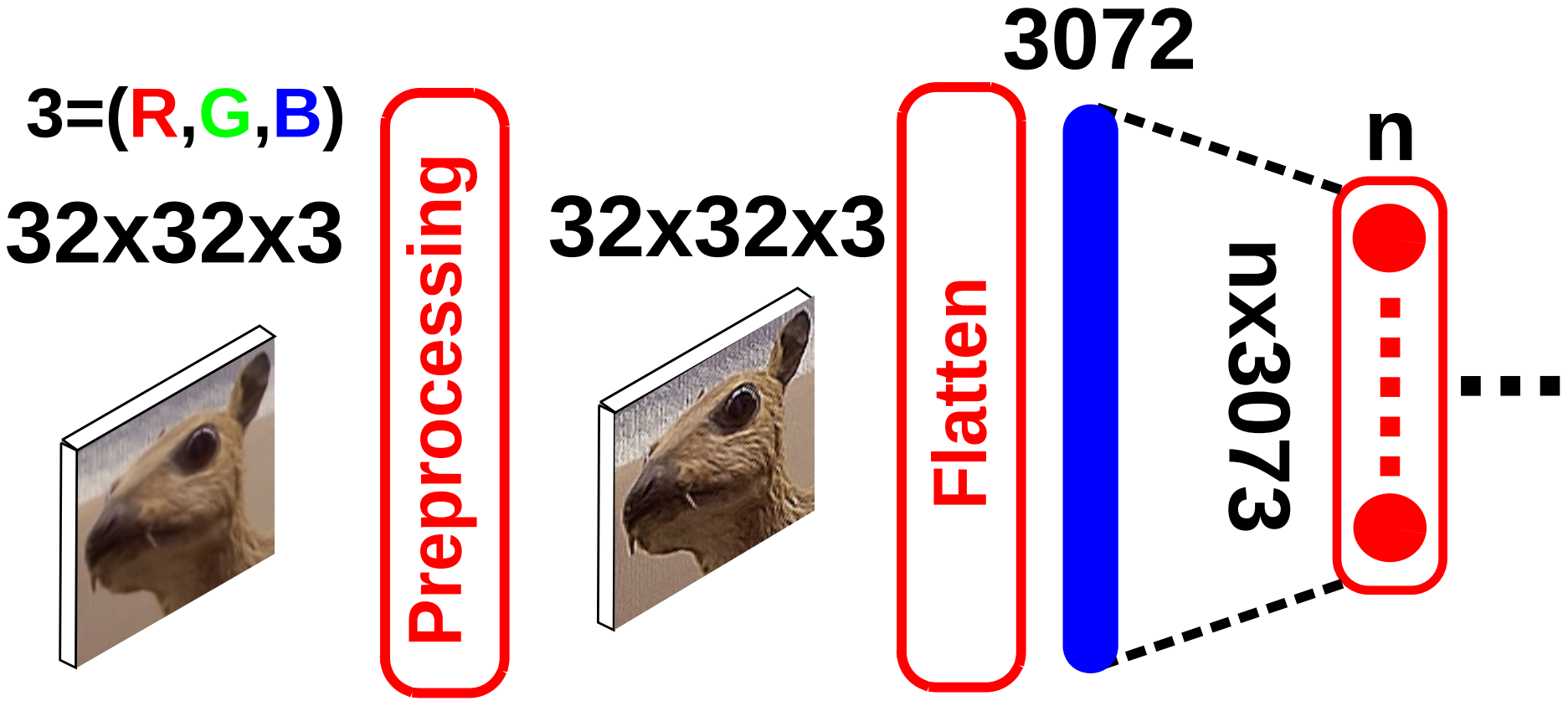
- Standard auto-encoder learns a perfect separation of clusters (non-smooth representation of latent states).
- **Solution**: force the model to learn a distribution (typically similar to the normal distribution).
- The model may learn broad distributions.
- **Solution**: use regularization (KL Divergence) to narrow them.
- The model may learn negative values for σ.
- **Solution**: learn $\log \sigma$.

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Auto-encoders: Some humor

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Convolutional Neural Network (CNN): Image processing (traditional architecture)



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Convolutional Neural Network (CNN): Image processing (preprocessing)

- There are several preprocessing techniques (see **[Alginahi, 2010]**).
- We focus on convolution-based techniques (modifying the value of a pixel relative to its neighbors).
- Two parameters: **padding** (surrounding the image with zeros to preserve the original size) and **stride** (the step size of the kernel/mask as it slides).

Original

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

Kernel

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

Padding: 1

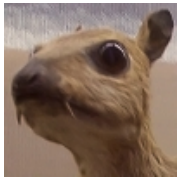
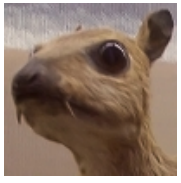
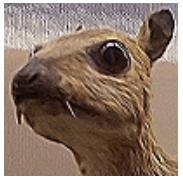
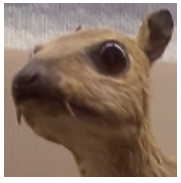
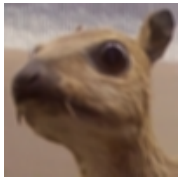
Processed

-3	-1	1	3	21
21	16	16	26	-23
-4	0	3	6	30
-3	3	7	11	25

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Convolutional Neural Network (CNN): Image processing (example of preprocessing)

An example from Wikipedia

Operation	Original	Kernel	Transformed
Edge detection		$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen		$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur		$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Convolutional Neural Network (CNN): Limitations of the previous solution

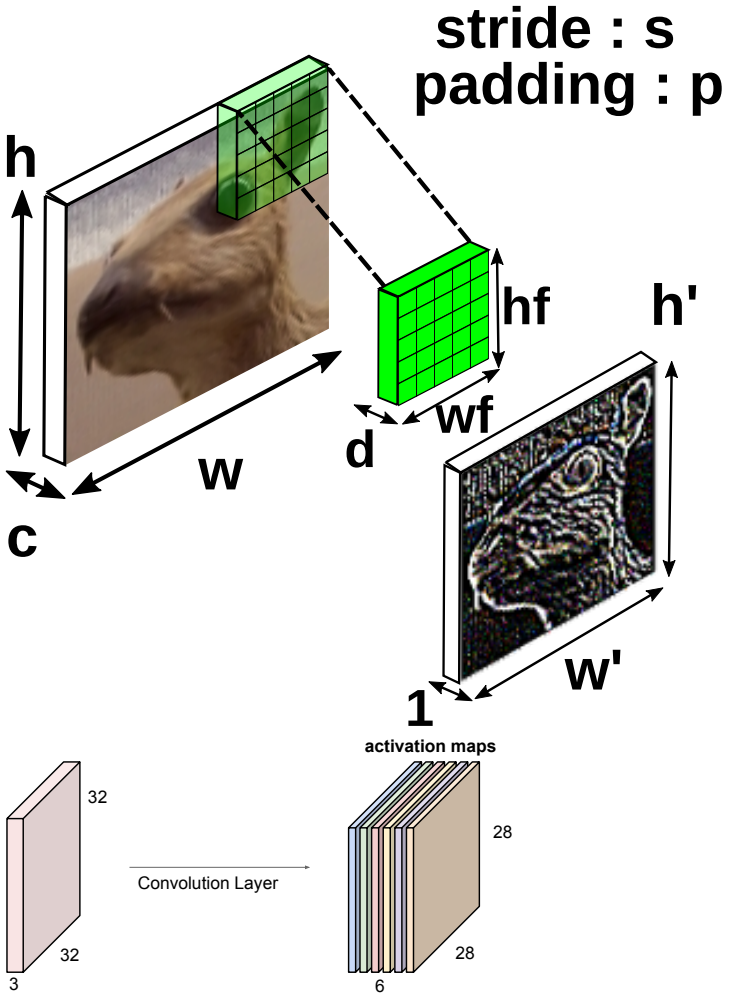
According to LeCun and his colleagues **[Lecun et al., 1998]**:

- When the size of the images is large, there will be a large number of parameters to train.
- To achieve this, a large dataset must be provided.
- The memory required to store the parameters will be very large.
- To train the model on images, preprocessing such as translations and distortions must be applied.
- Variations in images (such as the position of an object) can only be captured when multiple layers are used.

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

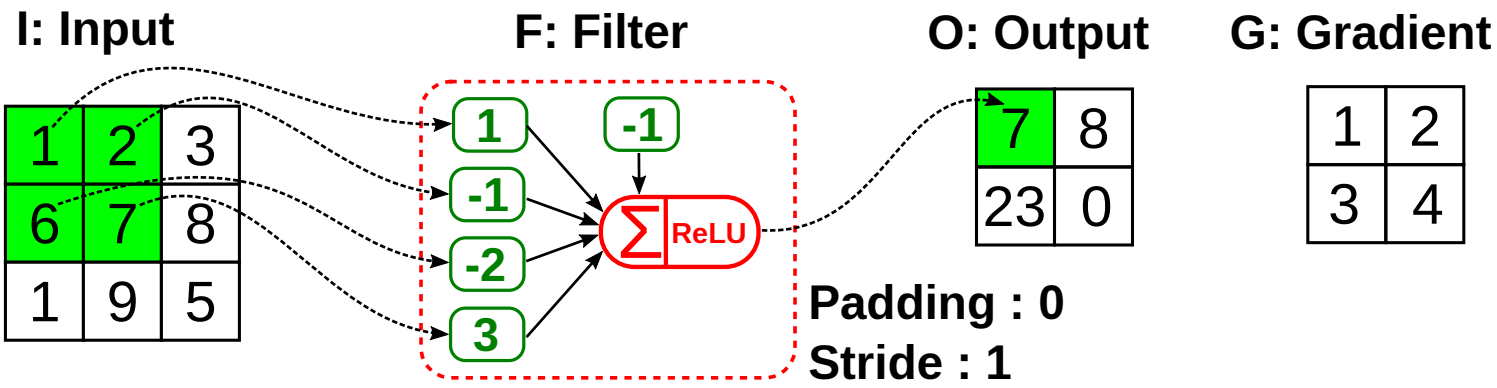
Convolutional Neural Network (CNN): Conv2D layer

- Preserve images' spatial structure.
- $w' = \frac{w - w_f + 2P}{S} + 1$, $h' = \frac{h - h_f + 2P}{S} + 1$
- A layer can have k filters/kernels.
- Number of parameters: $w_f * h_f * c * k$ plus k biases.
- Example, image: 32x32x3; kernel: 5x5; s: 1; p: 0; k: 6.
  Number of trainable parameters: 456



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Convolutional Neural Network (CNN): Conv2D layer (Example)



$$O_{11} = ReLU(\Sigma = b + F_{11}I_{11} + F_{12}I_{12} + F_{21}I_{21} + F_{22}I_{22})$$
$$O_{12} = ReLU(\Sigma = b + F_{11}I_{12} + F_{12}I_{13} + F_{21}I_{22} + F_{22}I_{23})$$
$$O_{21} = ReLU(\Sigma = b + F_{11}I_{21} + F_{12}I_{22} + F_{21}I_{31} + F_{22}I_{32})$$
$$O_{22} = ReLU(\Sigma = b + F_{11}I_{22} + F_{12}I_{23} + F_{21}I_{32} + F_{22}I_{33})$$

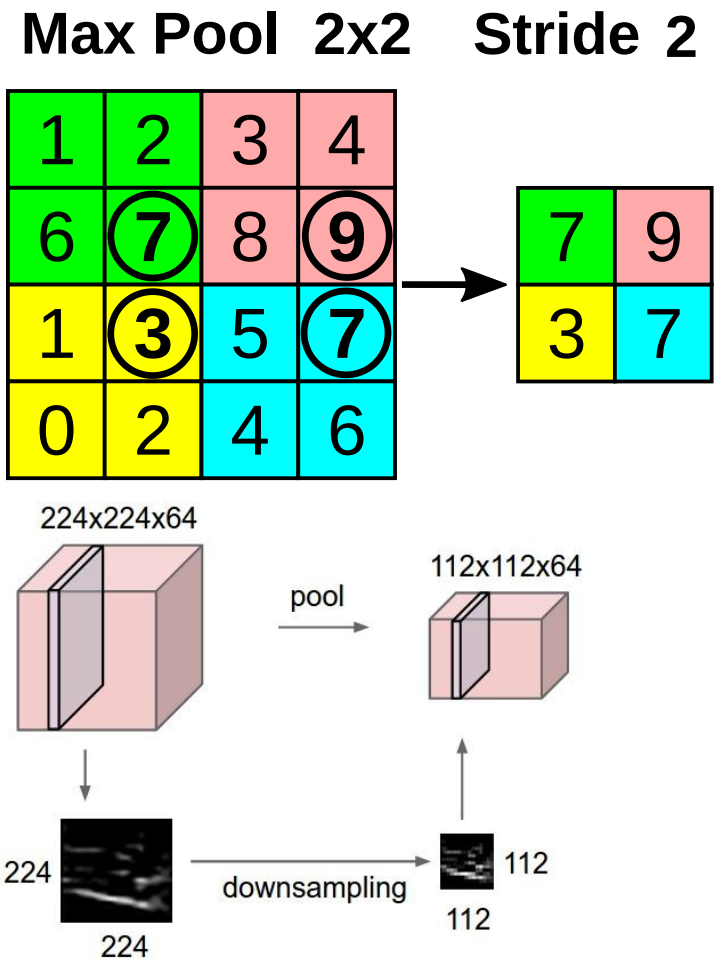
$$\frac{\partial J}{\partial I_{22}} = \underbrace{\frac{\partial J}{\partial O_{11}}}_{G_{11}=1} \underbrace{\frac{\partial ReLU}{\partial \sum}}_{1} \underbrace{\frac{\partial \sum}{\partial I_{22}}}_{F_{22}=3} + \underbrace{\frac{\partial J}{\partial O_{12}}}_{G_{12}=2} \underbrace{\frac{\partial ReLU}{\partial \sum}}_{1} \underbrace{\frac{\partial \sum}{\partial I_{22}}}_{F_{21}=-2} + \underbrace{\frac{\partial J}{\partial O_{21}}}_{G_{21}=3} \underbrace{\frac{\partial ReLU}{\partial \sum}}_{1} \underbrace{\frac{\partial \sum}{\partial I_{22}}}_{F_{12}=-1} + \underbrace{\frac{\partial J}{\partial O_{22}}}_{G_{22}=4} \underbrace{\frac{\partial ReLU}{\partial \sum}}_{0} \underbrace{\frac{\partial \sum}{\partial I_{22}}}_{F_{11}=1}$$

$$\frac{\partial J}{\partial I_{12}} = \underbrace{\frac{\partial J}{\partial O_{11}}}_{G_{11}=1} \underbrace{\frac{\partial ReLU}{\partial \sum}}_{1} \underbrace{\frac{\partial \sum}{\partial I_{12}}}_{F_{12}=-1} + \underbrace{\frac{\partial J}{\partial O_{12}}}_{G_{12}=2} \underbrace{\frac{\partial ReLU}{\partial \sum}}_{1} \underbrace{\frac{\partial \sum}{\partial I_{12}}}_{F_{11}=1}$$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

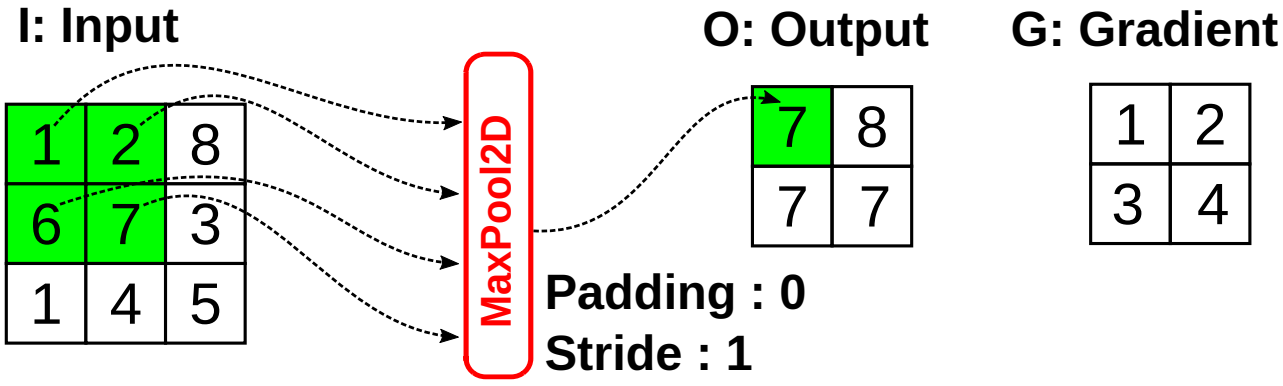
Convolutional Neural Network (CNN): Pooling

- Make the representation smaller and more manageable.
- $w' = \frac{w - w_f + 2P}{S} + 1, h' = \frac{h - h_f + 2P}{S} + 1$
- No parameters.
- **Max Pool**: The gradient is only passed to the winning cell. If there are multiple max values, choose one (usually the first).
- **Average Pool**: The average gradient is passed to participating cells. Each cell will have a gradient $\frac{\text{gradient}}{w_f * h_f}$.



Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Convolutional Neural Network (CNN): MaxPool2D layer (Example)



$$O_{11} = Max(I_{11} + I_{12} + I_{21} + I_{22}) = I_{22}$$
$$O_{12} = Max(I_{12} + I_{13} + I_{22} + I_{23}) = I_{23}$$
$$O_{21} = Max(I_{21} + I_{22} + I_{31} + I_{32}) = I_{22}$$
$$O_{22} = Max(I_{22} + I_{23} + I_{32} + I_{33}) = I_{22}$$

$$\frac{\partial J}{\partial I_{22}} = \underbrace{\frac{\partial J}{\partial O_{11}}}_{G_{11}=1} \underbrace{\frac{\partial Max}{\partial I_{22}}}_{1} + \underbrace{\frac{\partial J}{\partial O_{12}}}_{G_{12}=2} \underbrace{\frac{\partial Max}{\partial I_{22}}}_{0} + \underbrace{\frac{\partial J}{\partial O_{21}}}_{G_{21}=3} \underbrace{\frac{\partial Max}{\partial I_{22}}}_{1} + \underbrace{\frac{\partial J}{\partial O_{22}}}_{G_{22}=4} \underbrace{\frac{\partial Max}{\partial I_{22}}}_{1}$$

$$\frac{\partial J}{\partial I_{12}} = \underbrace{\frac{\partial J}{\partial O_{11}}}_{G_{11}=1} \underbrace{\frac{\partial Max}{\partial I_{12}}}_{0} + \underbrace{\frac{\partial J}{\partial O_{12}}}_{G_{12}=2} \underbrace{\frac{\partial Max}{\partial I_{12}}}_{0}$$

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Convolutional Neural Network (CNN): Some humor

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Regularization: Droupout

- When training large neural networks on a small dataset, overfitting can occur.
- One solution is to reduce the size of the network.
- Another is to temporarily deactivate connections in a random manner.
- This technique is called **dropout**.
- The dropout layer takes a dropout rate as a parameter.
- This layer deactivates some outputs (considers them as 0) during training (not during inference).

Advanced Neural Networks: Feedforward Neural Networks (FFNN)

Regularization: Some humor

If Thanos were a Data Scientist

Section 3

Recurrent Neural Networks (RNN)

Advanced Neural Networks

Recurrent Neural Networks (RNN)

Definition

A ”Recurrent Neural Network” is an architecture of Neural Networks. In this one, there is a ”feed” back from the outputs of the neurons to the inputs throughout the network. **[Sazlı, 2006]**

In other word, there is a temporal phenomena treated in the problem in question; current output depends on the past one.

Example

- Predicting future weather based on some features, plus the current one

Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN)

- **Elman network**

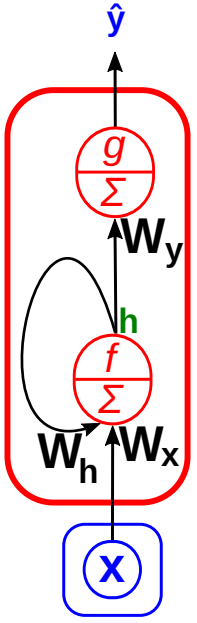
$$h_t = f(w_x x_t + w_h h_{t-1} + b_h)$$
$$\hat{y}_t = g(w_y h_t + b_y)$$

- **Jordan network**

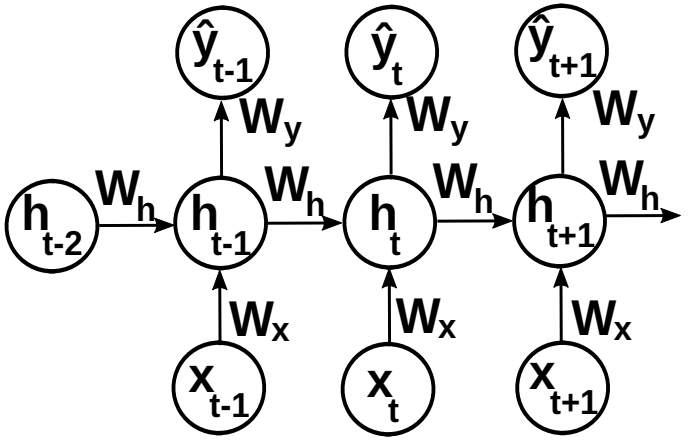
$$h_t = f(w_x x_t + w_h \hat{y}_{t-1} + b_h)$$
$$\hat{y}_t = g(w_y h_t + b_y)$$

RNN

$$h_t = f(W_x x_t + W_h h_{t-1} b_h)$$
$$y_t = g(W_y h_t + b_y)$$



Unfolded network



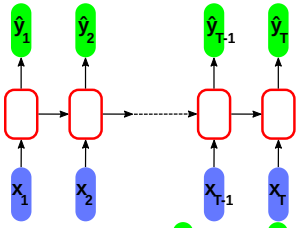
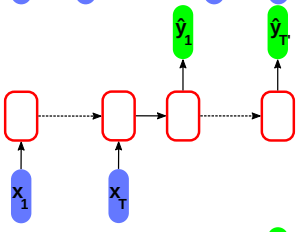
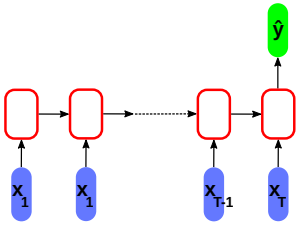
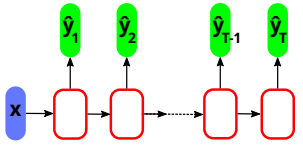
Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN): Backpropagation

- Define the cost function for each output.
- Use backpropagation through time **[Werbos, 1990]**
  - Unroll the recurrent network over time.
  - The network will be similar to a feed-forward one.
  - Accumulate gradients starting from the last state.
  - Update parameters when reaching the first state.

Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN): Applications

Type	Illustration	Example
Many to many		Named entity detection
Many to many (Seq2seq)		Machine translation
Many to one		Sentiment classification
One to many		Image caption generation

Advanced Neural Networks: Recurrent Neural Networks (RNN)

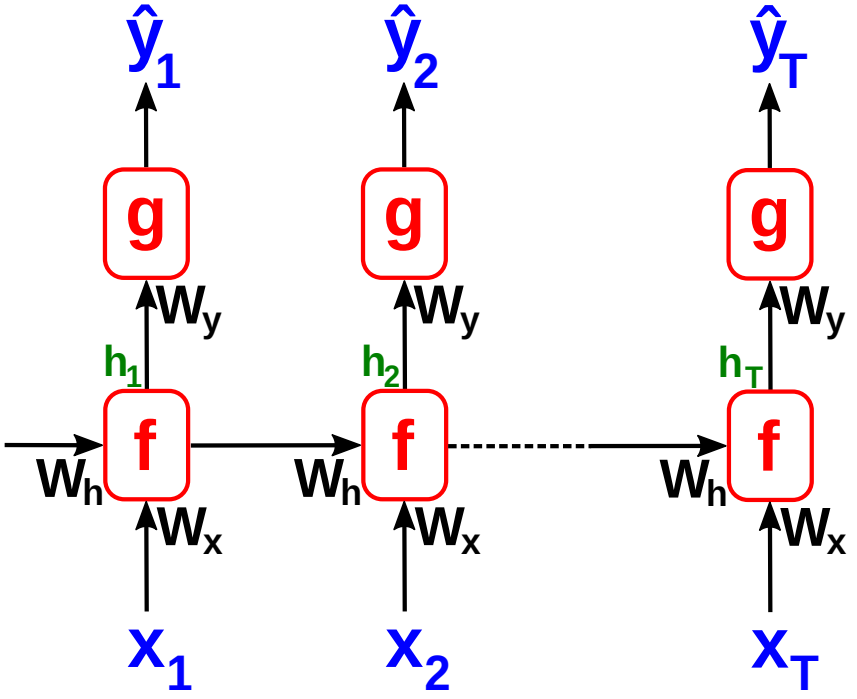
Architecture (RNN): Applications (direct many to many)

h_0 is a predefined victor

$$h_t = f(x_t, h_{t-1}, W_x, W_h)$$

$$\hat{y}_t = g(h_t, W_y)$$

$$J(\hat{y}, y) = \frac{1}{T} \sum_{t=1}^{T} j(\hat{y}_t, y_t)$$



Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN): Applications (direct many to many: Back-propagation)

$$\frac{\partial J}{\partial W_y} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial j(\hat{y}_t, y_t)}{\partial W_y} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial j(\hat{y}_t, y_t)}{\partial \hat{y}_t}\frac{\partial g(h_t, W_y)}{\partial W_y}$$

$$\frac{\partial J}{\partial W_h} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial j(\hat{y}_t, y_t)}{\partial W_h} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial j(\hat{y}_t, y_t)}{\partial \hat{y}_t}\frac{\partial g(h_t, W_y)}{\partial h_t}\frac{\partial h_t}{\partial W_h}$$

$$\frac{\partial h_t}{\partial W_h} = \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial W_h} + \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_h}$$

$$\frac{\partial J}{\partial W_x} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial j(\hat{y}_t, y_t)}{\partial W_x} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial j(\hat{y}_t, y_t)}{\partial \hat{y}_t}\frac{\partial g(h_t, W_y)}{\partial h_t}\frac{\partial h_t}{\partial W_x}$$

$$\frac{\partial h_t}{\partial W_x} = \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial W_x} + \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_x}$$

Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN): Applications (direct many to many: Problem)

$\frac{\partial h_t}{\partial W_h}$ can be simplified:

$$\frac{\partial h_t}{\partial W_h} = \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial W_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, W_x, W_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, W_x, W_h)}{\partial W_h}$$

- Can result in ***vanishing gradients*** (e.g. $f = \sigma(\sum)$)
- Also, ***exploding gradients*** (e.g. $f = ReLU(\sum)$)
- Possible solution: Truncating time steps **[Jaeger, 2002]**

Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN): Applications (Seq2seq)

Encoder

h_0 is a predefined victor

$$h_t^e = f^e(x_t, h_{t-1}^e, W_x^e, W_h^e)$$

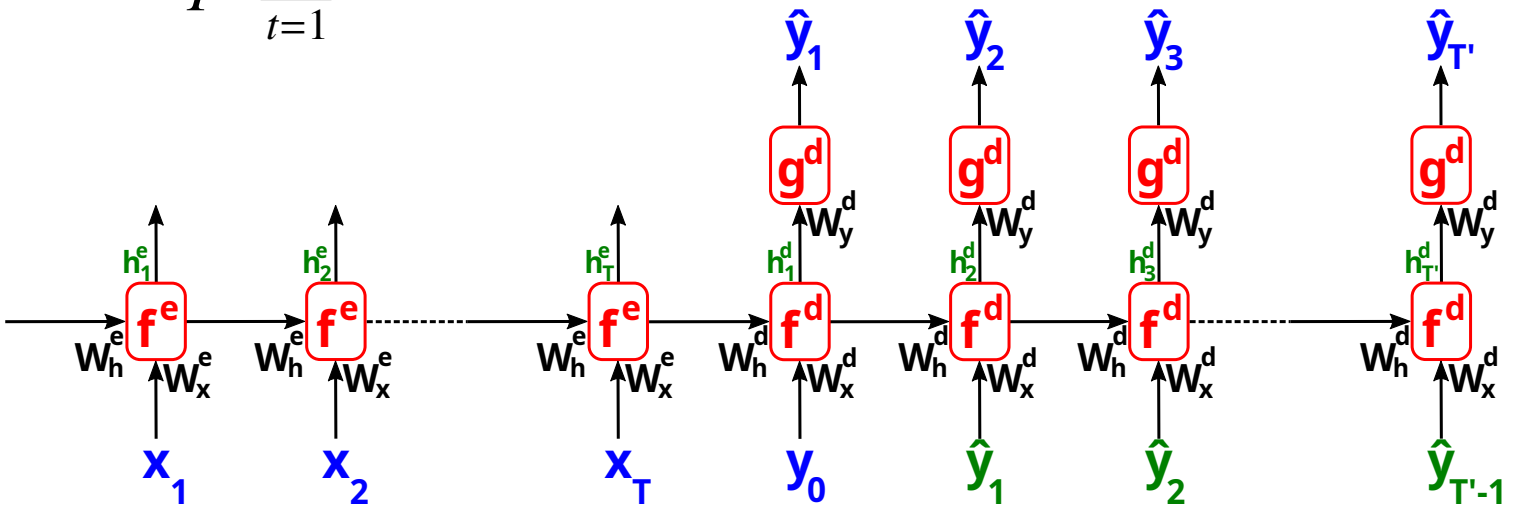
$$J(\hat{y}, y) = \frac{1}{T'} \sum_{t=1}^{T'} j(\hat{y}_t, y_t)$$

Decoder

$h_0^d = h_T^e$, y_0 is a predefined victor

$$h_t^d = f^d(\hat{y}_{t-1}, h_{t-1}^d, W_x^d, W_h^d)$$

$$\hat{y}_t = g^d(h_{t-1}^d, W_y^d)$$



Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN): Applications (Seq2seq: Back-propagation)

- Decoder cell is updated as direct Many2Many, replacing x_t by $\hat{y}_{t-1}$
- Encoder cell is updated as direct Many2Many, but without direct output

TODO: Revise (my brain is killing me)

$$\frac{\partial J}{\partial W_h^e} = \frac{\partial J}{\partial h_T^e}\frac{\partial h_T^e}{\partial W_h^e} \quad \frac{\partial J}{\partial W_x^e} = \frac{\partial J}{\partial h_T^e}\frac{\partial h_T^e}{\partial W_x^e}$$

$$\frac{\partial J}{\partial h_T^e} = \frac{\partial J}{\partial h_0^d} = \frac{1}{T'}\sum_{t=1}^{T'}\frac{\partial j(\hat{y}_t, y_t)}{\partial h_0^d} = \frac{1}{T'}\sum_{t=1}^{T'}\frac{\partial j(\hat{y}_t, y_t)}{\partial \hat{y}_t}\frac{\partial g(h_t^d, W_y^d)}{\partial h_t^d}\frac{\partial h_t^d}{\partial h_0^d}$$

$$\frac{\partial h_t^d}{\partial h_0^d} = \frac{\partial f^d(\hat{y}_{t-1}, h_{t-1}^d, W_x^d, W_h^d)}{\partial h_{t-1}^d}\frac{\partial h_{t-1}^d}{\partial h_0^d} = \prod_{i=1}^{t}\frac{\partial f^d(\hat{y}_{i-1}, h_{i-1}^d, W_x^d, W_h^d)}{\partial h_{i-1}^d}$$

Advanced Neural Networks: Recurrent Neural Networks (RNN)

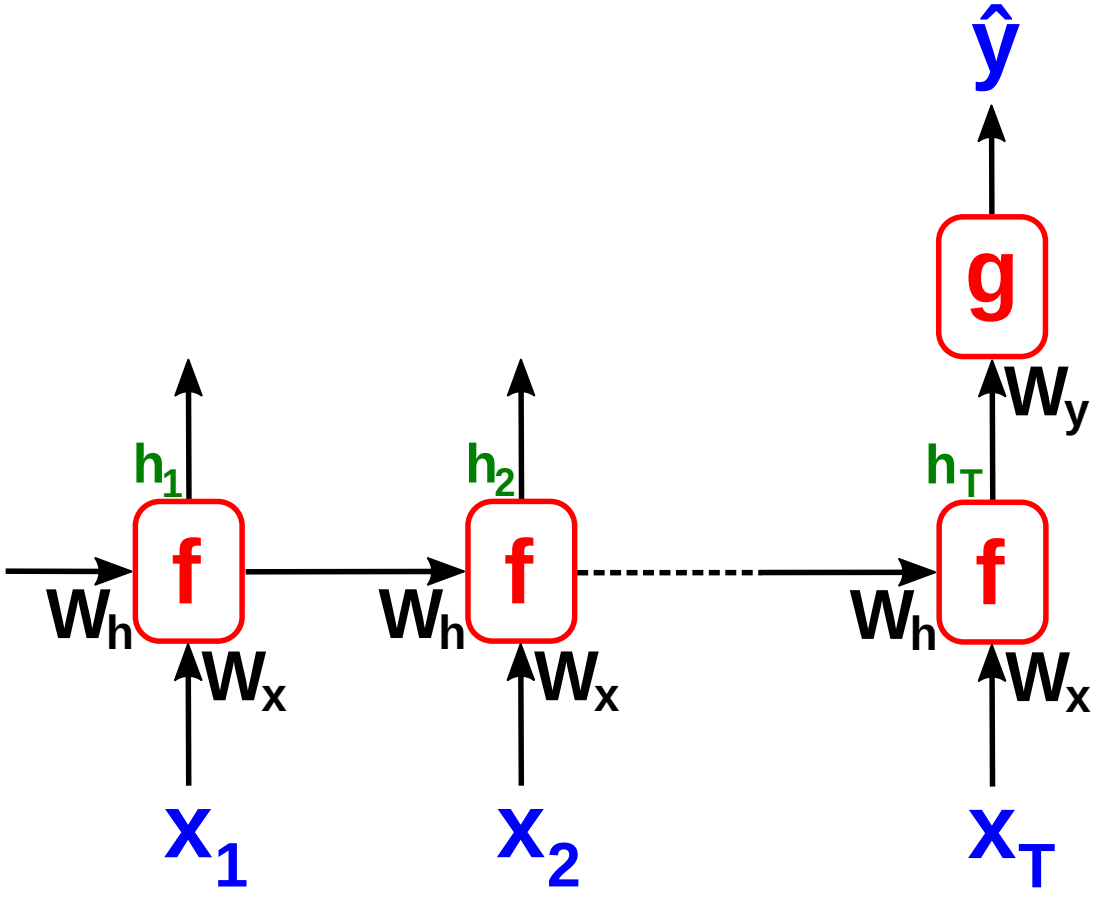
Architecture (RNN): Applications (many to one)

h_0 is a predefined victor

$$h_t = f(x_t, h_{t-1}, W_x, W_h)$$

$$\hat{y} = g(h_T, W_y)$$

$$J(\hat{y}, y) = j(\hat{y}, y)$$



Advanced Neural Networks: Recurrent Neural Networks (RNN)

Architecture (RNN): Applications (many to one: Back-propagation)

$$\frac{\partial J}{\partial W_y} = \frac{\partial j(\hat{y}, y)}{\partial W_y} = \frac{\partial j(\hat{y}, y)}{\partial \hat{y}} \frac{\partial g(h_T, W_y)}{\partial W_y}$$

$$\frac{\partial J}{\partial W_h} = \frac{\partial j(\hat{y}, y)}{\partial W_h} = \frac{\partial j(\hat{y}, y)}{\partial \hat{y}} \frac{\partial g(h_T, W_y)}{\partial h_T} \frac{\partial h_T}{\partial W_h}$$

$$\frac{\partial h_t}{\partial W_h} = \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial W_h} + \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_h}$$

$$\frac{\partial J}{\partial W_x} = \frac{\partial j(\hat{y}, y)}{\partial W_x} = \frac{\partial j(\hat{y}, y)}{\partial \hat{y}} \frac{\partial g(h_T, W_y)}{\partial h_T} \frac{\partial h_T}{\partial W_x}$$

$$\frac{\partial h_t}{\partial W_x} = \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial W_x} + \frac{\partial f(x_t, h_{t-1}, W_x, W_h)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_x}$$

Advanced Neural Networks: Recurrent Neural Networks (RNN)

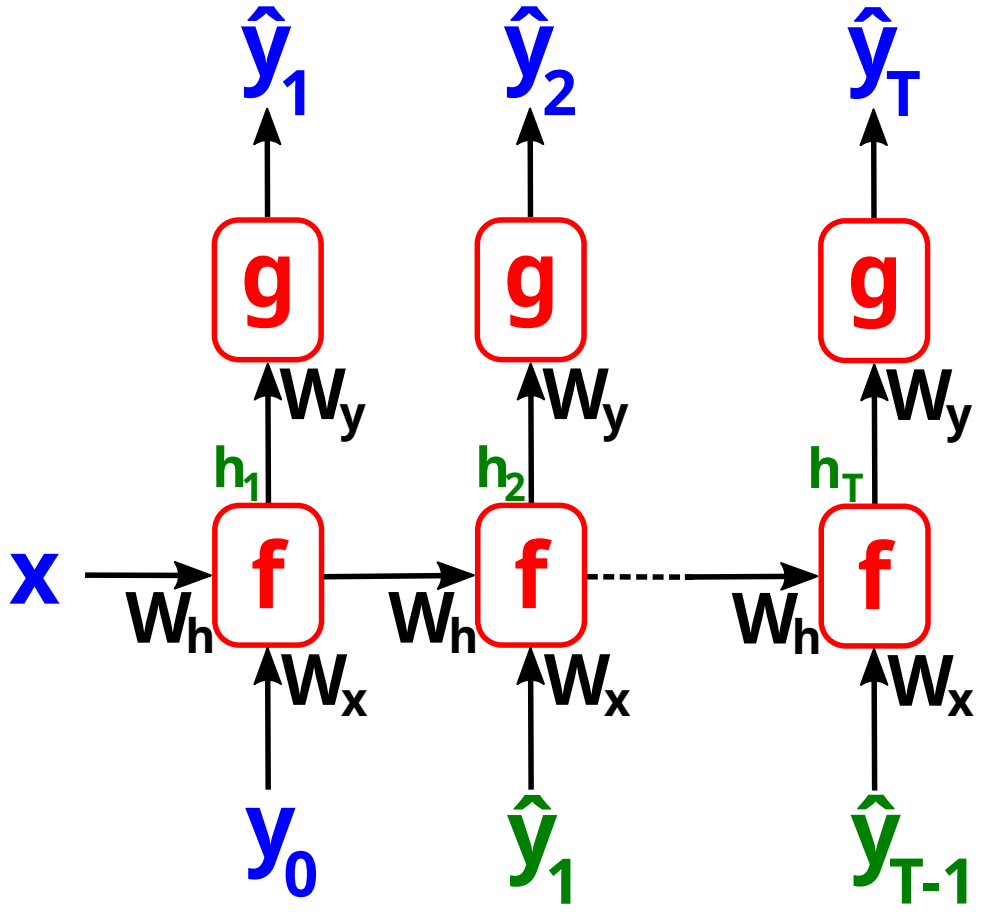
Architecture (RNN): Applications (one to many)

$$h_1 = f(y_0, x, W_x, W_h)$$

$$h_t = f(\hat{y}_{t-1}, h_{t-1}, W_y, W_h)$$

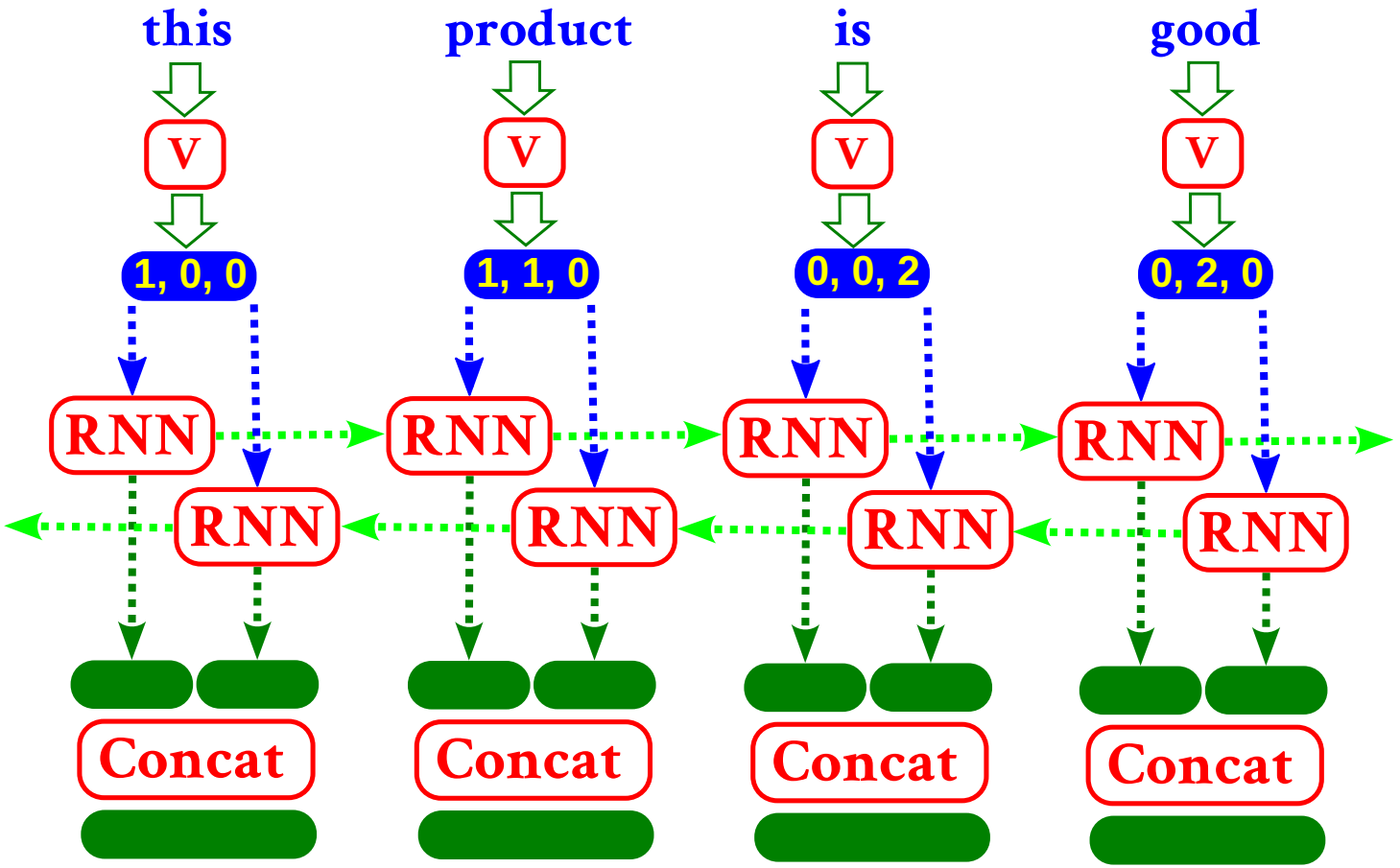
$$\hat{y}_t = g(h_t, W_y)$$

$$J(\hat{y}, y) = \frac{1}{T} \sum_{t=1}^{T} j(\hat{y}_t, y_t)$$



Advanced Neural Networks: Recurrent Neural Networks (RNN)

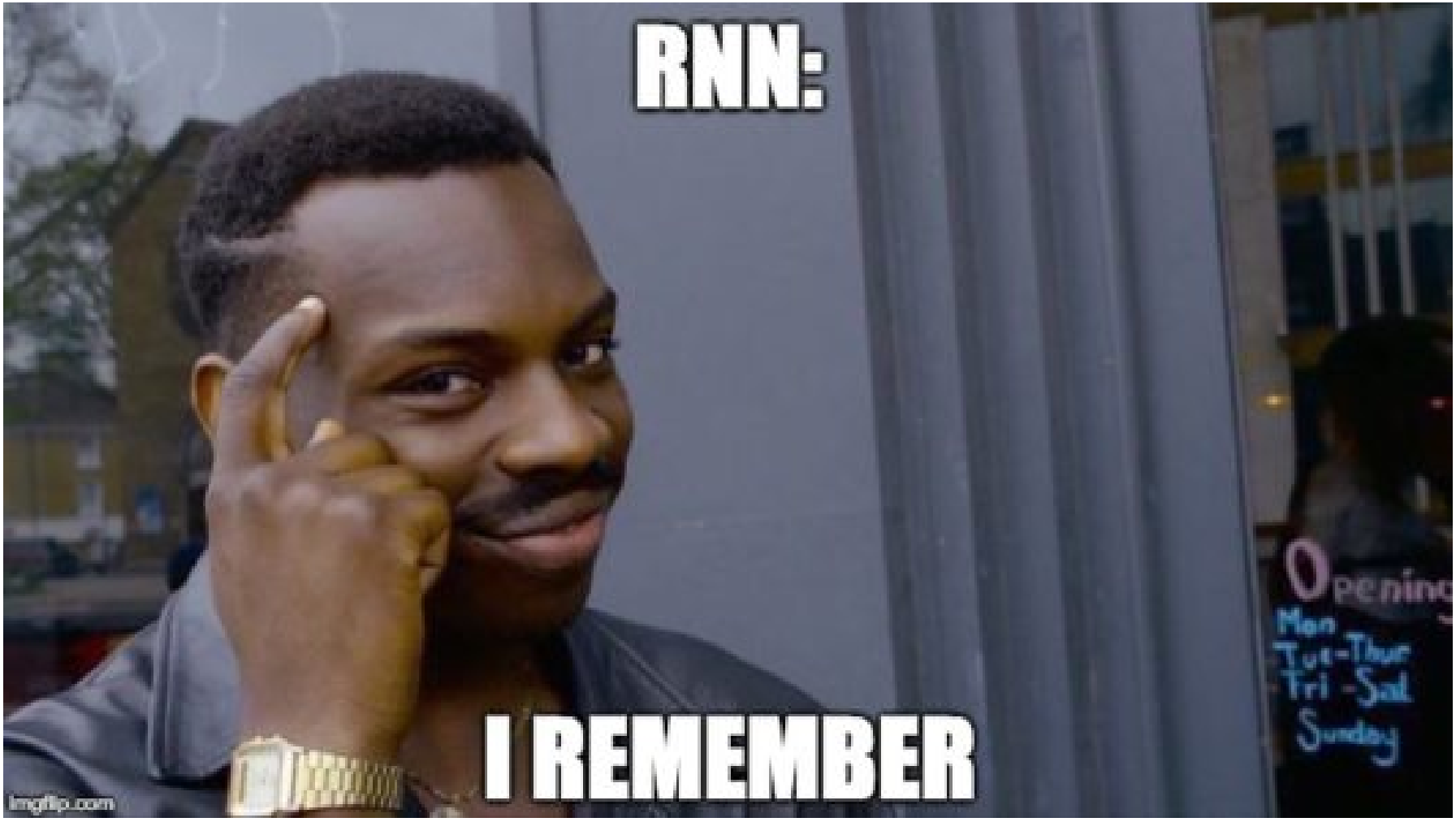
Architecture (RNN): Bidirectional RNN



V Words' vectorization function (embedding)

Advanced Neural Networks: Recurrent Neural Networks (RNN)

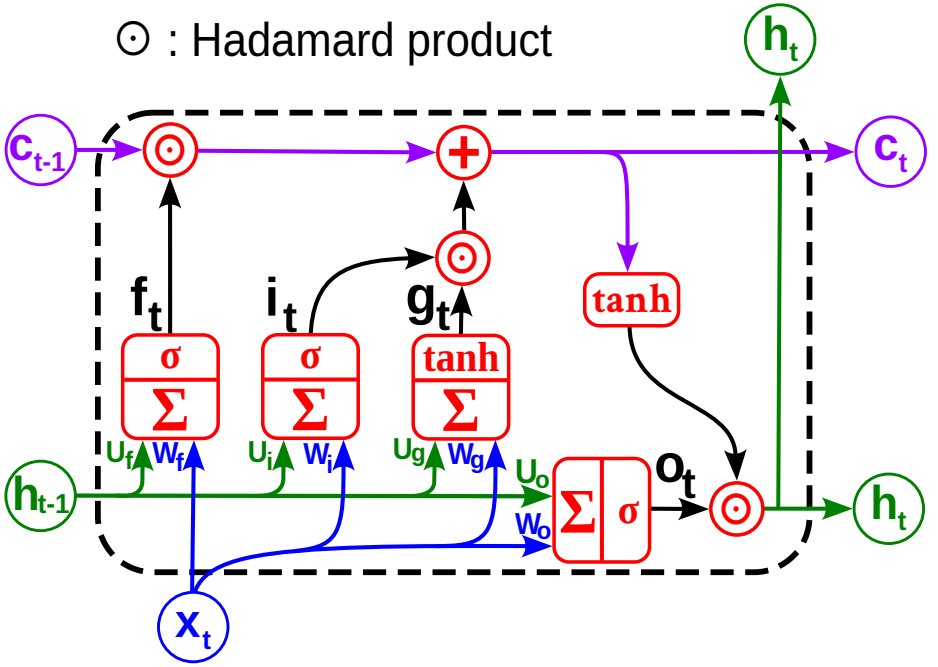
Architecture (RNN): Some humor



Advanced Neural Networks: Recurrent Neural Networks (RNN)

Long Short-Term Memory (LSTM): Architecture

- i: **input gate**; How much information will be added to the cell's intern state?
- f: **Forget gate**; Must the past state be forgotten?
- o: **Output gate** ; Combining the cell's intern and current states



- g: **Input node**; Will the state be increased or decreased? (mostly considered to be a part of input gate)

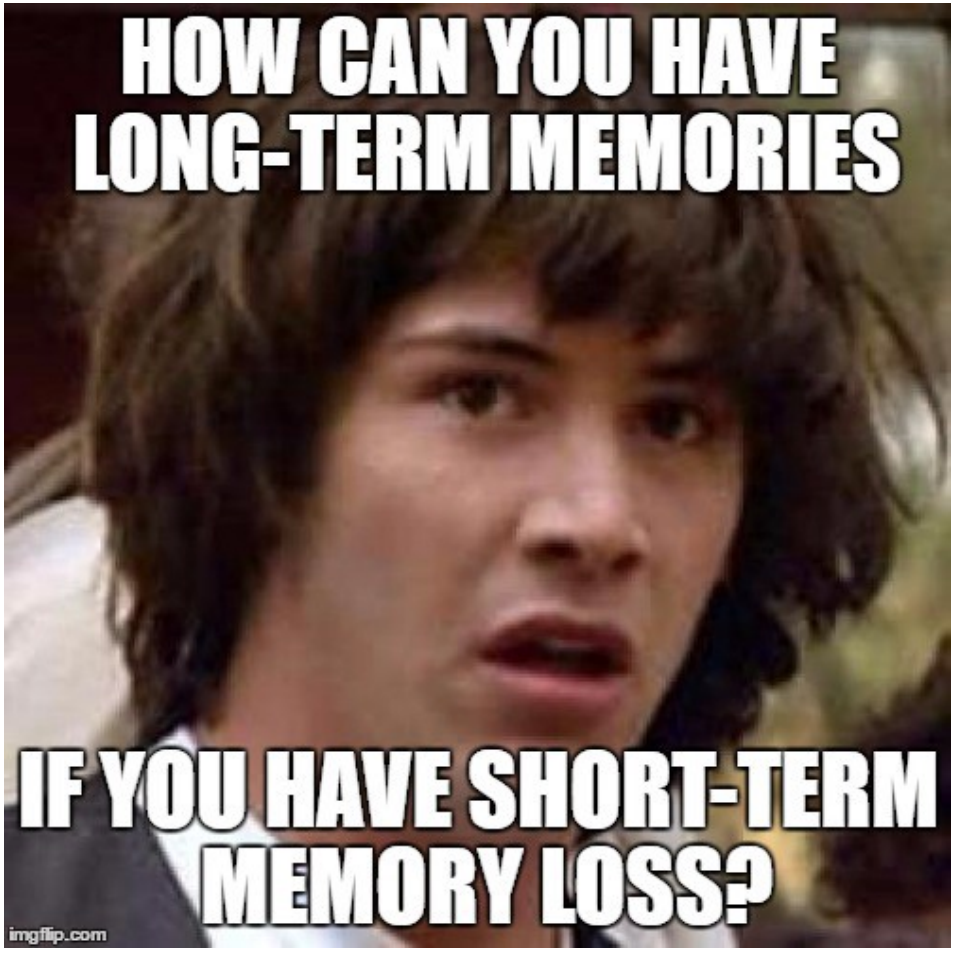
Advanced Neural Networks: Recurrent Neural Networks (RNN)

Long Short-Term Memory (LSTM): Equations

$$f_t = \sigma(W_f X_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma(W_i X_t + U_i h_{t-1} + b_i)$$
$$g_t = \tanh(W_g X_t + U_g h_{t-1} + b_g)$$
$$o_t = \tanh(W_o X_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} \oplus i_t \circ g_t$$
$$h_t = o_t + \tanh(c_t)$$

Advanced Neural Networks: Recurrent Neural Networks (RNN)

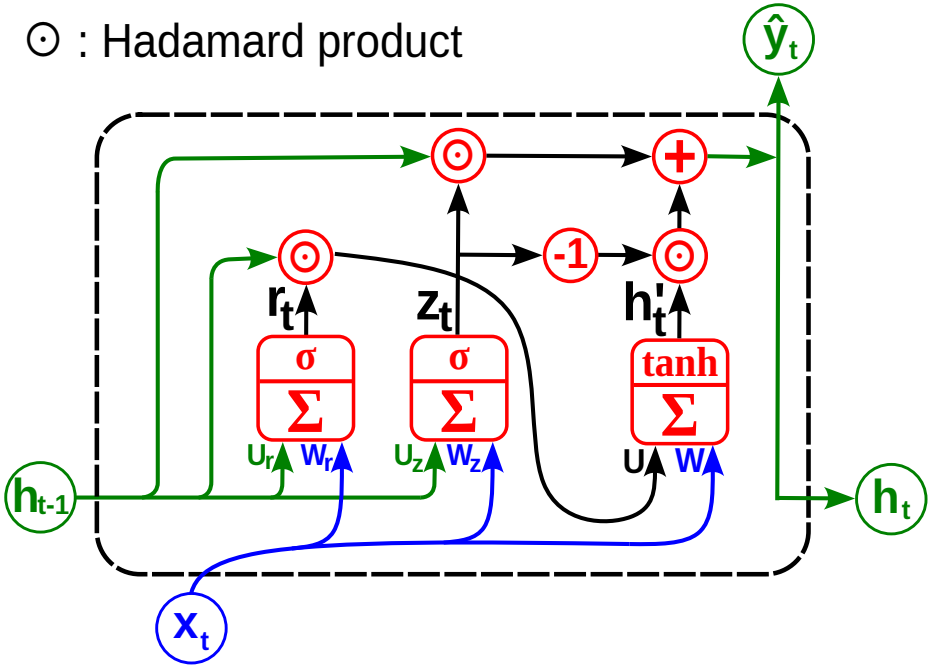
Long Short-Term Memory (LSTM): Some humor



Advanced Neural Networks: Recurrent Neural Networks (RNN)

Gated Recurrent Unit (GRU): Architecture

- z: **Update gate**; Merges LSTM's input and forget gates.
- r: **Reset gate**; How much update from the past state?
- h': **Candidate hidden state**; Current state.



Advanced Neural Networks: Recurrent Neural Networks (RNN)

Gated Recurrent Unit (GRU): Equations

$$z_t = \sigma(W_z X_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r X_t + U_r h_{t-1} + b_r)$$
$$h'_t = \tanh(W X_t + U(h_{t-1} \circ r_t) + b)$$
$$h_t = z_t \circ h_{t-1} \oplus (1 - z_t) \circ h'_{t-1}$$

Advanced Neural Networks: Recurrent Neural Networks (RNN)

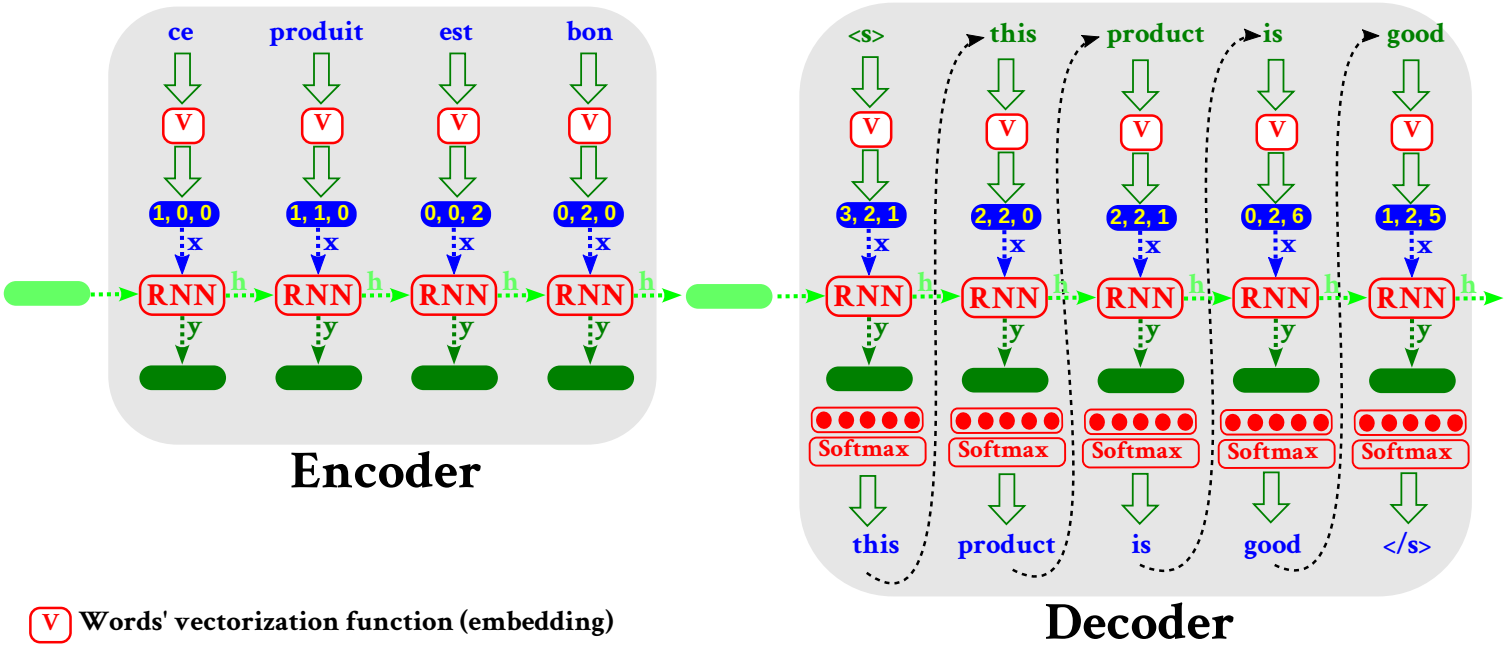
Gated Recurrent Unit (GRU): Some humor

Section 4

Attention

Advanced Neural Networks

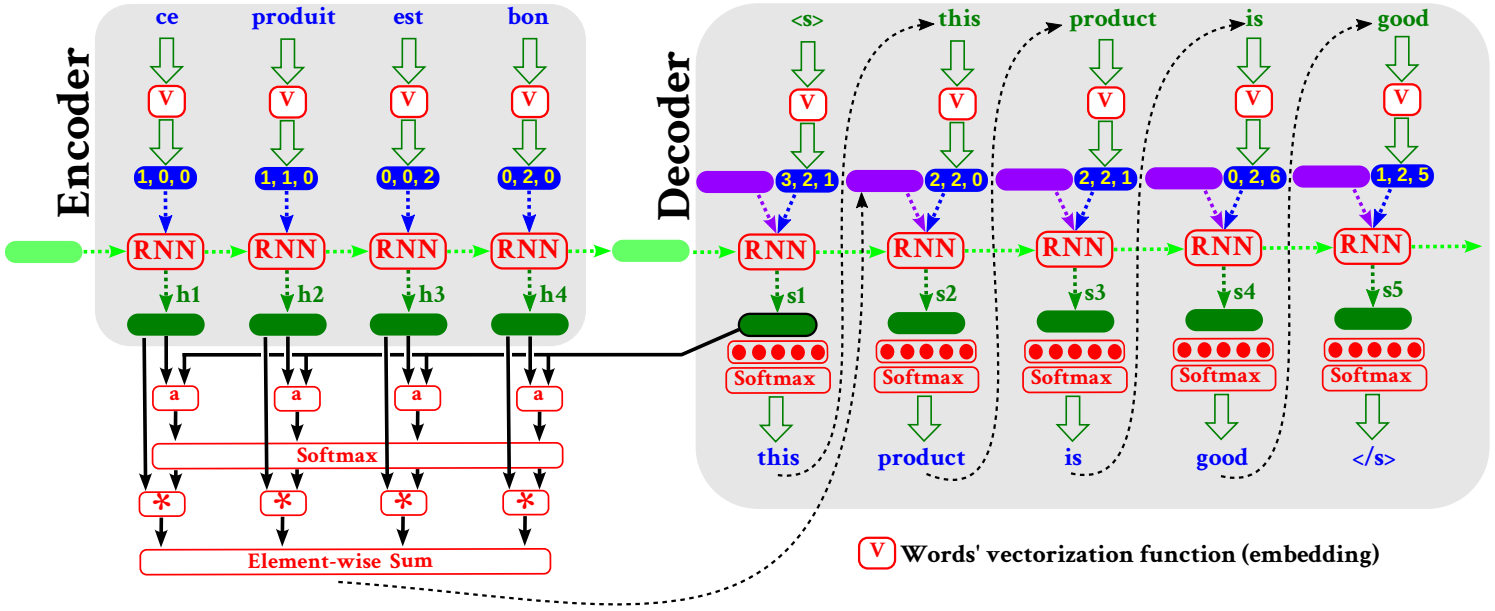
Attention: Motivation



- Model: Seq2Seq (Encoder-Decoder)
- Long-term dependencies are not considered.
- The generation of target words depends only on previously generated words and the last words of the original text.

Advanced Neural Networks: Attention

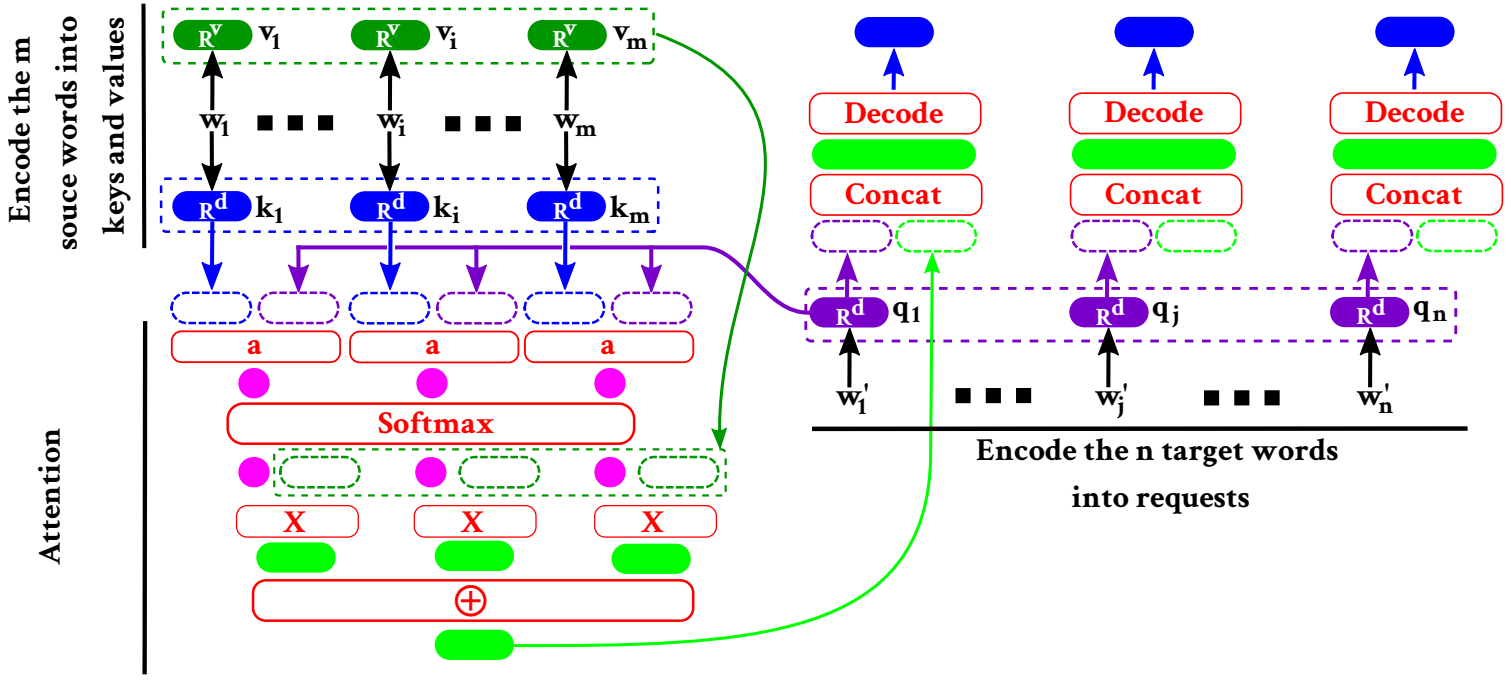
Attention mechanism



- Keep T hidden states from the encoder: h_i
- For each s_{t-1}, calculate its alignment with the hidden states.
  $e_{t,i} = a(s_{t-1}, h_i)$
- Calculate the weights of the hidden states: $\alpha_{t,i} = \text{softmax}(e_{t,i})$
- Calculate the state vector $c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$

Advanced Neural Networks: Attention

Attention mechanism: Formalism (1)



- Keys (k_i): m states which are used to estimate values' weights
- Values (v_i): m states used to construct the context
- Queries (q_j): current representation

Advanced Neural Networks: Attention

Attention mechanism: Formalism (2)

- Let $w_1 \cdots w_m$ a source sequence
  - $k_i = Embedding_k(w_i)$ a vector of d elements
  - $v_i = Embedding_v(w_i)$ a vector of v elements
- Let w'_j a current target word
  - $q_j = Embedding_q(w'_j)$ a vector of d elements
- Calculate a similarity between each key k_i and our current target word

$$e_i = a(q_j, k_i)$$

- Calculate the percentage similarity (softmax can be masked)

$$\alpha_i = softmax(e_i) = \frac{exp(e_i)}{\sum_k exp(e_k)}$$

- The attention is a vector formed by cumulating percentages of each value vector

$$attention(q, K, V) = \sum_{i=1}^{m} \alpha_i v_i$$

Advanced Neural Networks: Attention

Attention mechanism: Attention functions (non parametric)

$$Q \in \mathbb{R}^{n \times d},\ K \in \mathbb{R}^{m \times d},\ a(Q,K) \in \mathbb{R}^{n \times m},\ V \in \mathbb{R}^{m \times v}$$

Dot product (**tf.keras.layers.Attention**)

$$a(Q,K) = Q \cdot K^\top$$

Scaled dot product

$$a(Q,K) = \frac{Q \cdot K^\top}{\sqrt{d}}$$

Cosine similarity

$$a(Q,K) = \frac{Q \cdot K^\top}{||Q||\,||K||}$$

Advanced Neural Networks: Attention

Attention mechanism: Attention functions (parametric)

$$Q \in \mathbb{R}^{n \times d},\ K \in \mathbb{R}^{m \times d},\ a(Q,K) \in \mathbb{R}^{n \times m},\ V \in \mathbb{R}^{m \times v}$$

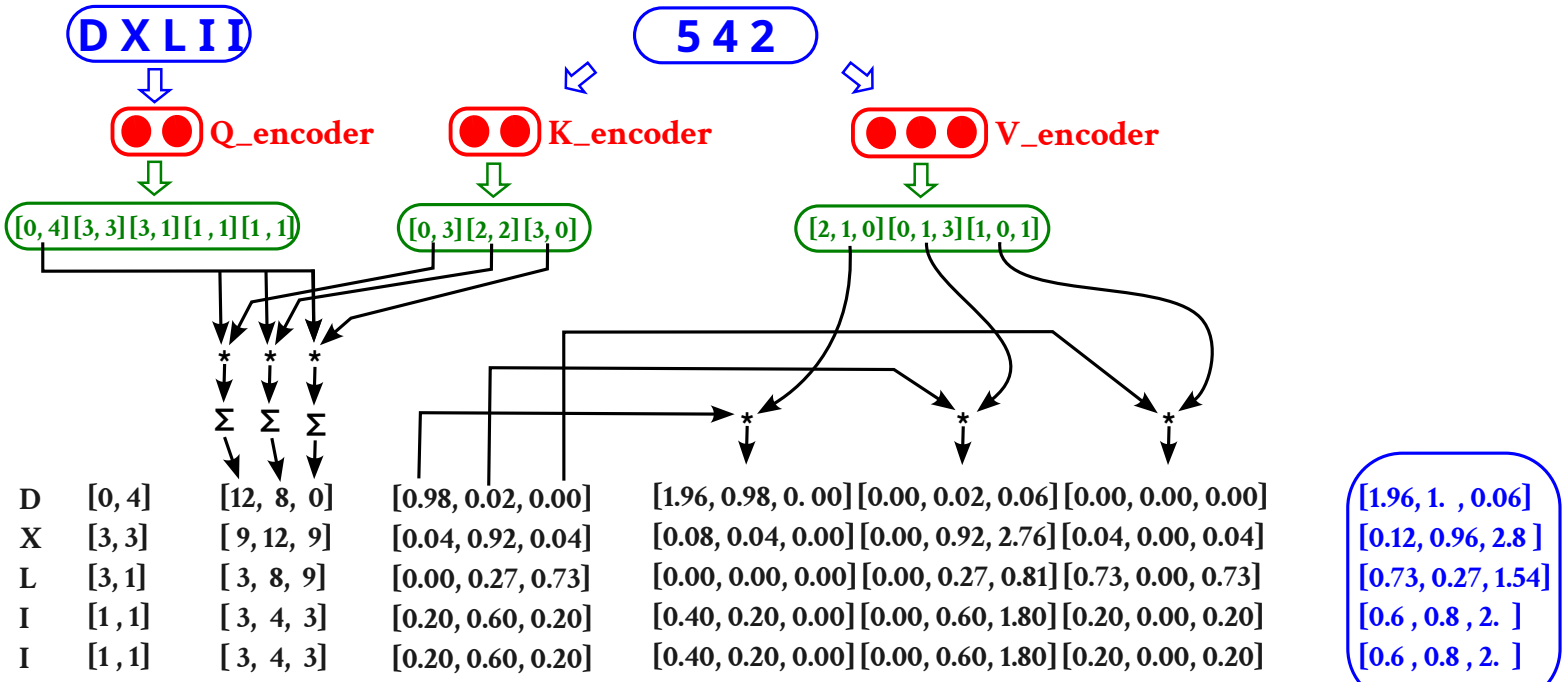
Additive attention (**tf.keras.layers.AdditiveAttention**)
Bahdanau Attention

$$a(Q,K) = W_v^\top \cdot \tanh(W_q Q + W_k K)$$

Advanced Neural Networks: Attention

Attention mechanism: Numerical example

Arabic numeral into Roman numeral transformation; 542 = DXLII.
We want to encode each Roman digit according to their attention with the Arabic digits



$e = Q \cdot K^T$
Attention function

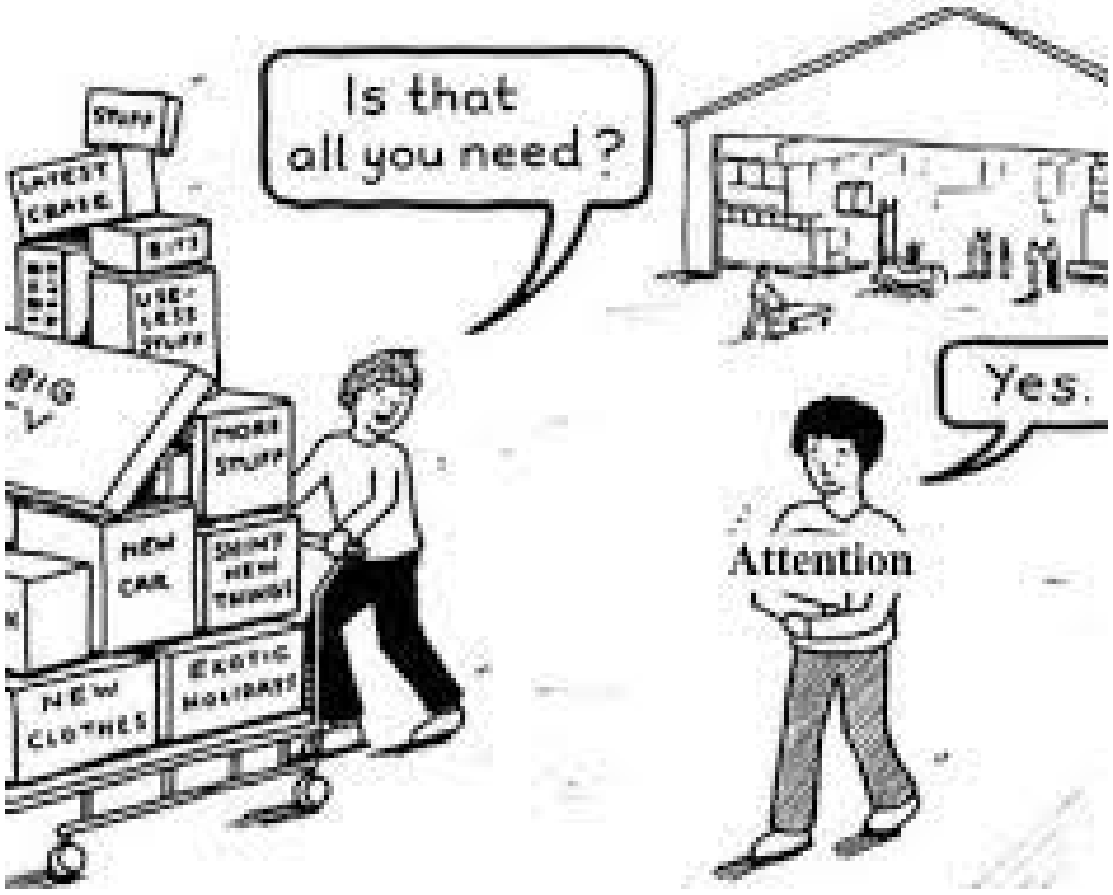
$\alpha = Softmax(e)$
Attention percentage

$V'_i = V_i * \alpha_i$
Multiply Each value vector by the equivalent percentage

$\sum V'_i$
Element-wise Sum over weighted value vectors

Advanced Neural Networks: Attention

Attention mechanism: Some humor



Advanced Neural Networks: Attention

Multi-Head Attention

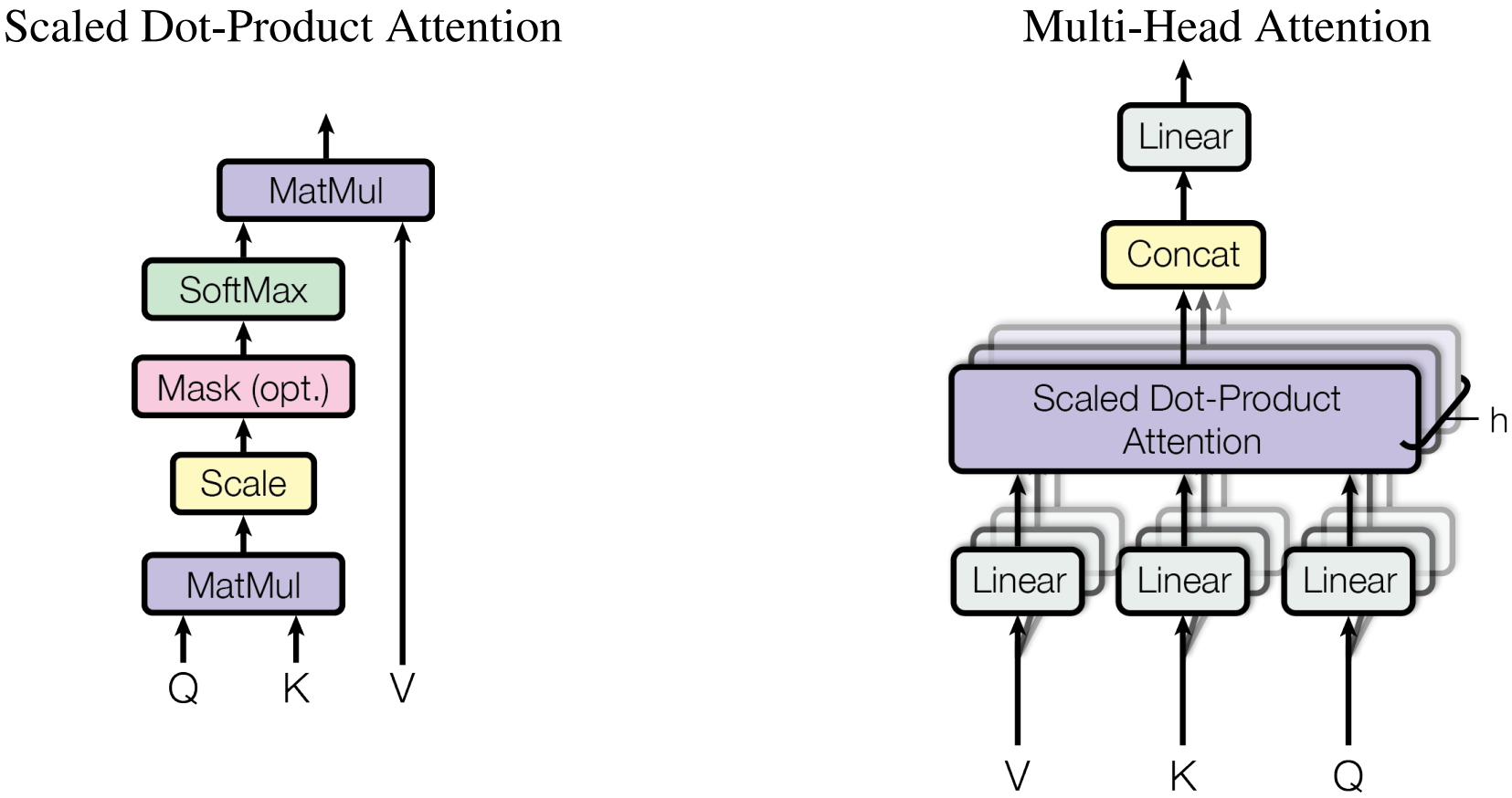


Figure: Multi-Head Attention **[Vaswani et al., 2017]**

Advanced Neural Networks: Attention

Multi-Head Attention: Formulas

$$Q \in \mathbb{R}^{n \times d},\ K \in \mathbb{R}^{m \times d},\ MultiHead \in \mathbb{R}^{n \times d_{model}},\ V \in \mathbb{R}^{m \times v}$$

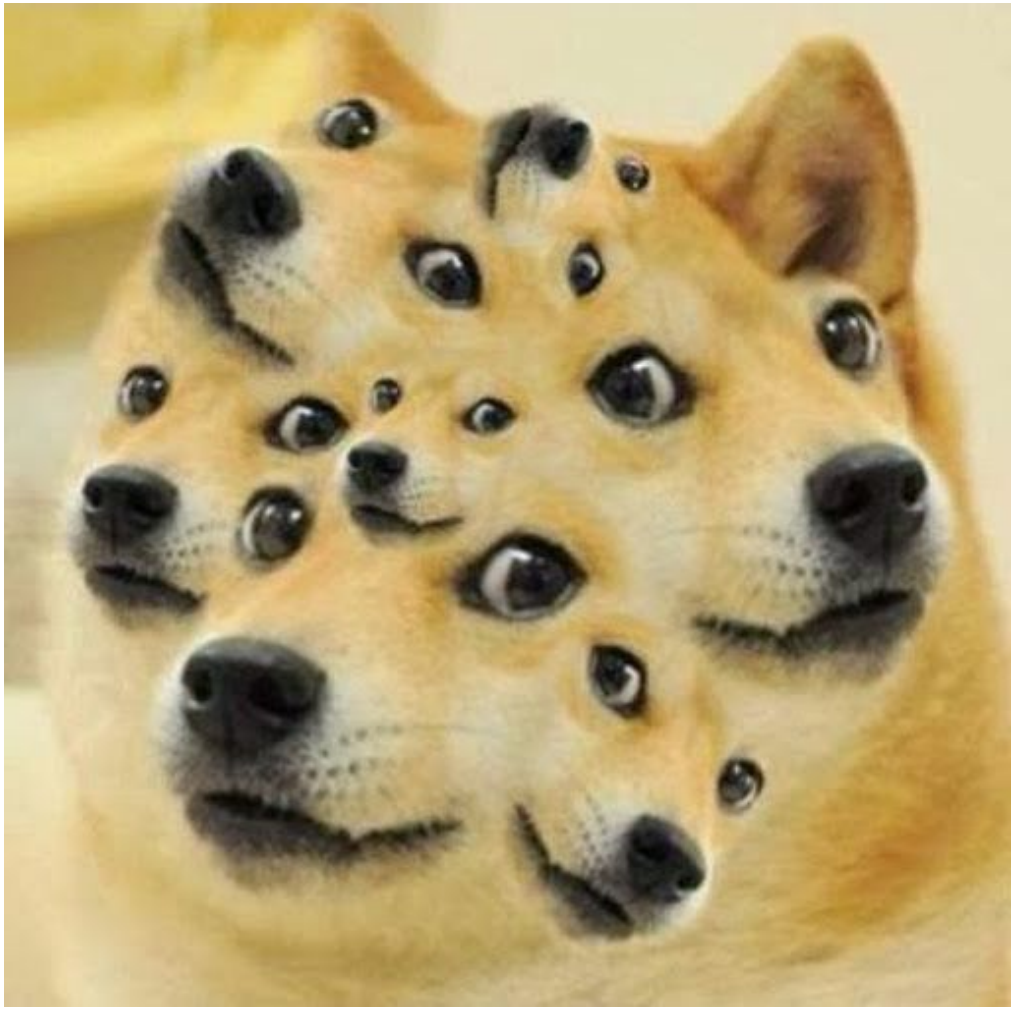
$$W_i^Q \in \mathbb{R}^{d_{model} \times d},\ W_i^K \in \mathbb{R}^{d_{model} \times d},\ W_i^V \in \mathbb{R}^{d_{model} \times v},\ W^O \in \mathbb{R}^{hv \times d_{model}}$$

$$head_i = Attention(QW_i^Q, KW_i^Q, VW_i^V)$$

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h) \cdot W^O$$

Advanced Neural Networks: Attention

Multi-Head Attention: Some humor



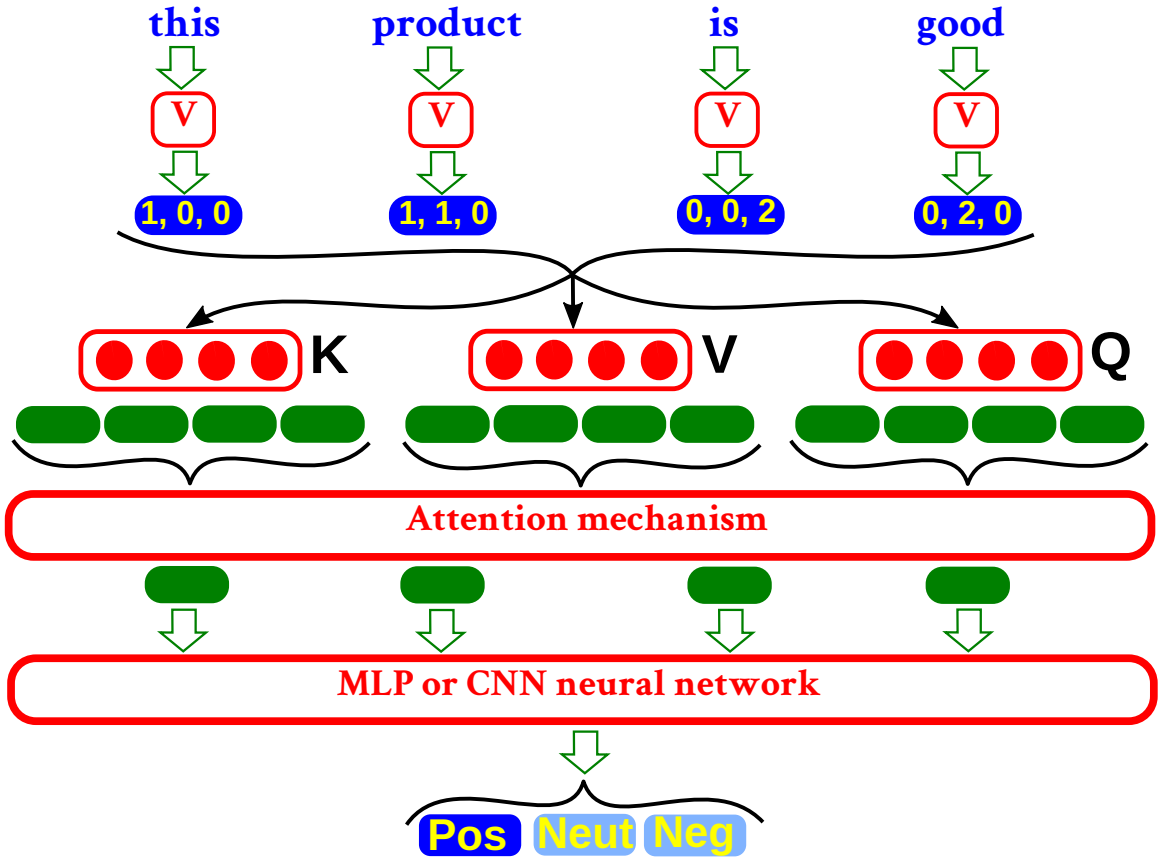
Advanced Neural Networks: Attention

Self-attention

- **GOAL**: Encode an element of a sequence with respect to the entire sequence.
- **Example**: Encode a word with respect to the sentence to capture its semantics.
- **Method**:
  - Use MLPs to learn the query, key, and value for each element.
  - Set the maximum number of elements in a sequence.
  - Truncate sequences that exceed this number.
  - Pad those that do not reach this number with a predefined code.
  - Optionally use multi-head attention to learn multiple representations.
  - Add positional embeddings to retain positional information in the sequence.

Advanced Neural Networks: Attention

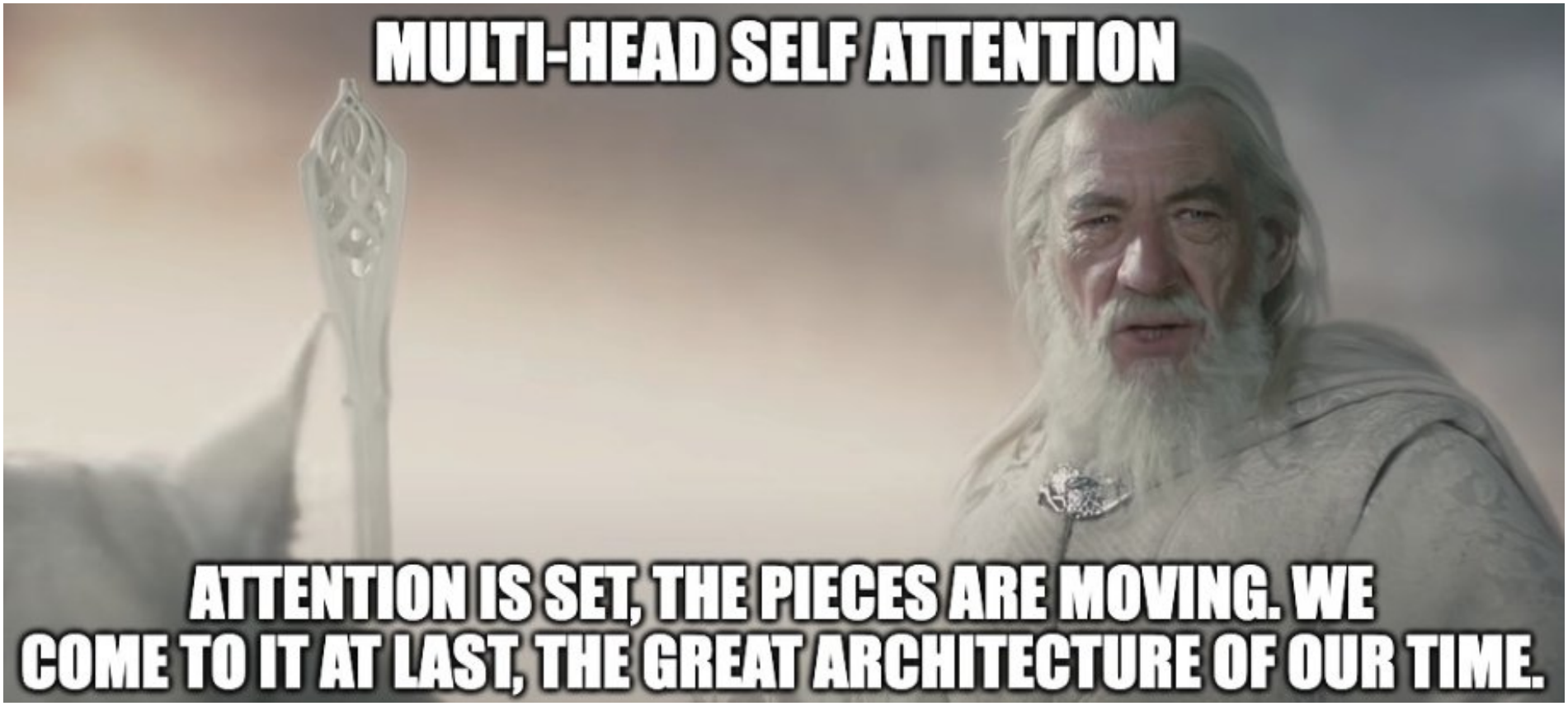
Self-attention: Example



V Words' vectorization function (embedding)

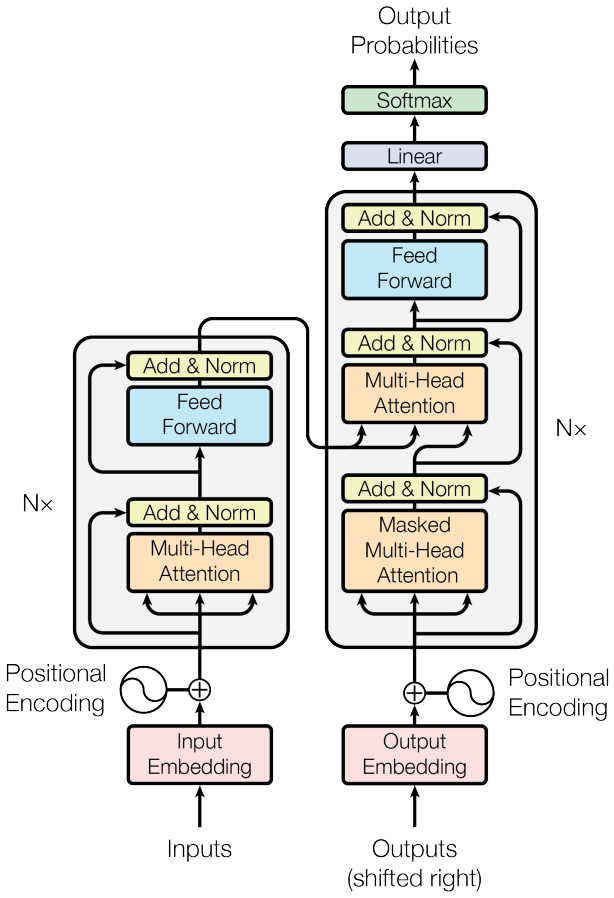
Advanced Neural Networks: Attention

Self-attention: Some humor



Advanced Neural Networks: Attention

Transformer



=

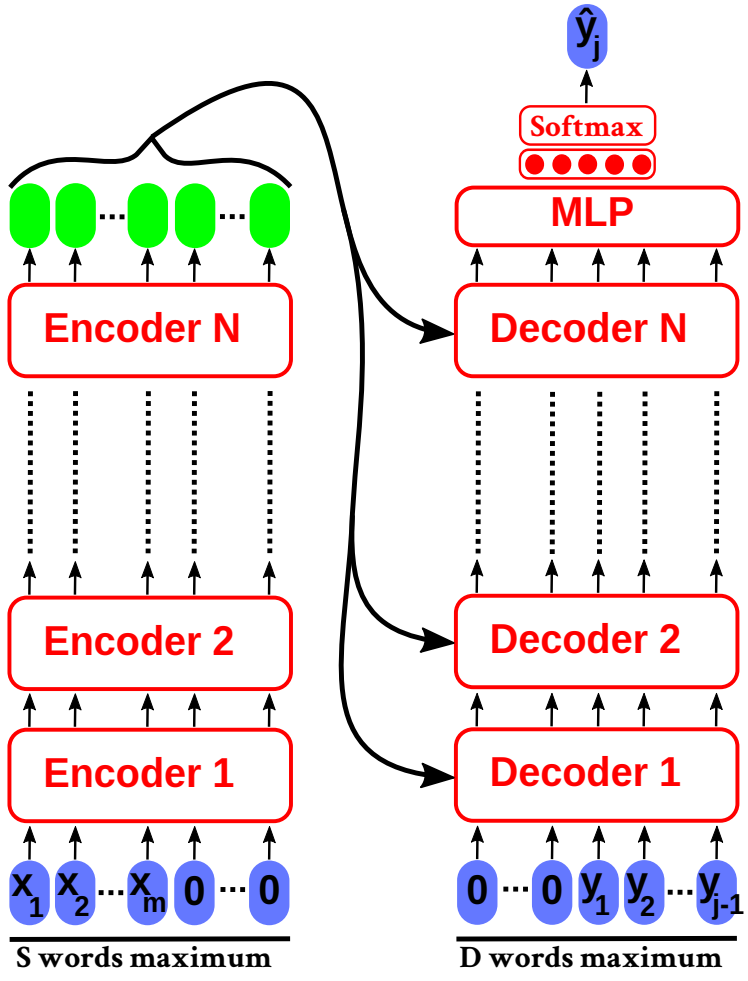


Figure: Transformers' architecture
[Vaswani et al., 2017]

Advanced Neural Networks: Attention

Transformer: Encoder

- Encodes each input element based on its context (the entire input).
- Uses multi-head self-attention
- Completely bi-directional
  - In BiRNN architecture, the encoder represents long-term relation indirectly (using a context).
  - In Encoders it is directly represented, but the notion of element's position is lost.
- We can introduce position's embedding into the word's initial representation (before being transformed into keys, values and queries).

Advanced Neural Networks: Attention

Transformer: Decoder

- ***Autoregressive***: The output is a **direct** function of the past elements

$$\hat{y}_t = \sum_{i=1}^{\rho} \varphi_i y_{t-i} + \epsilon_t$$

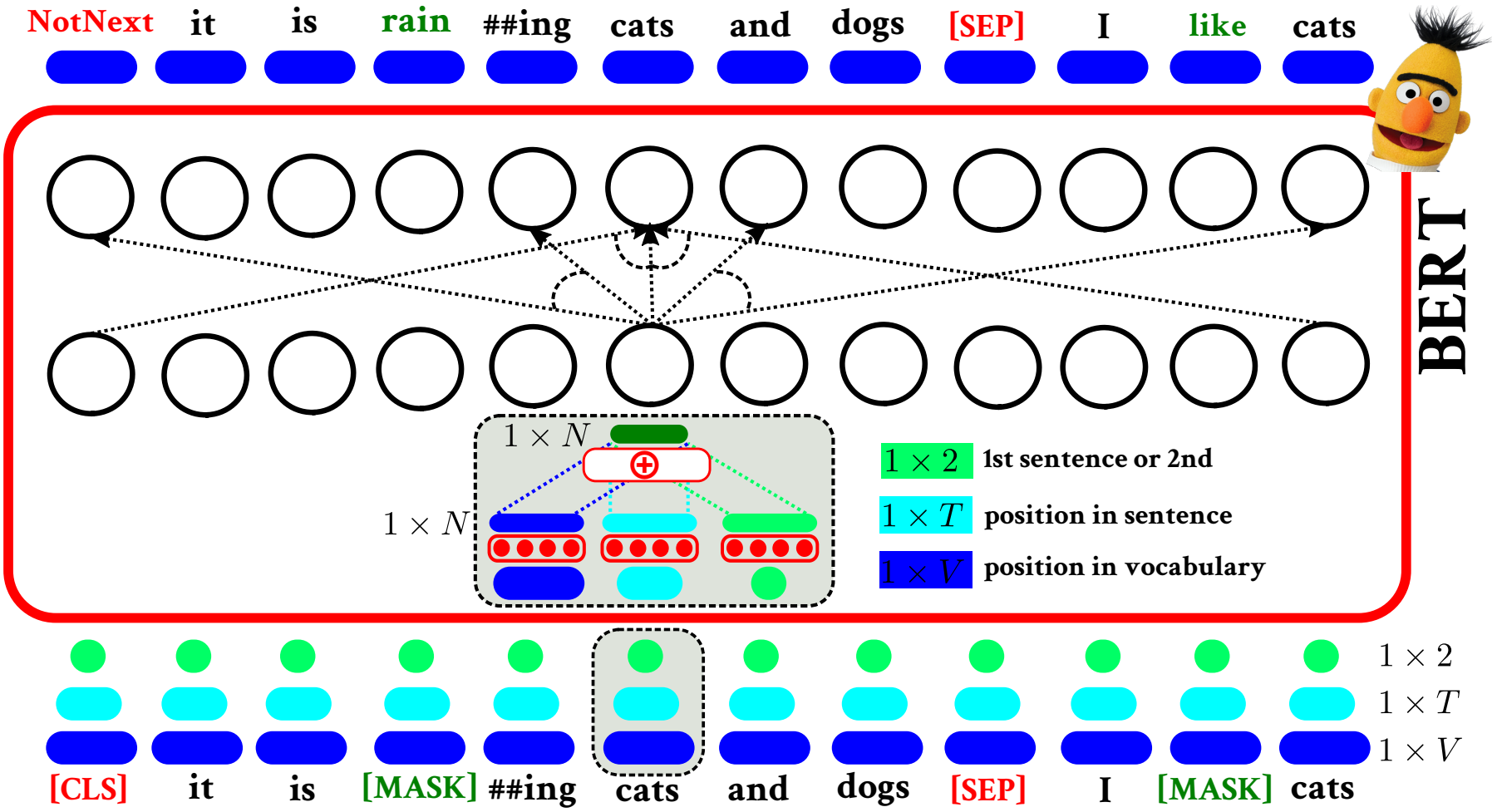
- Not as RNNs where the dependence is indirect

$$\hat{y}_t = \arg\max_y p(y|X;\ y_1, \cdots, y_{i-1};\ W;\ b)$$

- Each time an element is generated, it will be added into the input to generate the next element.
- To stop generating, a maximum size is used. Also, a special token is used to mark the end of generation.

Advanced Neural Networks: Attention

Transformer: Pretrained models (BERT: Enc) [Devlin et al., 2019]



Advanced Neural Networks: Attention

Transformer: Pretrained models (ViT: Enc) [Dosovitskiy et al., 2021]

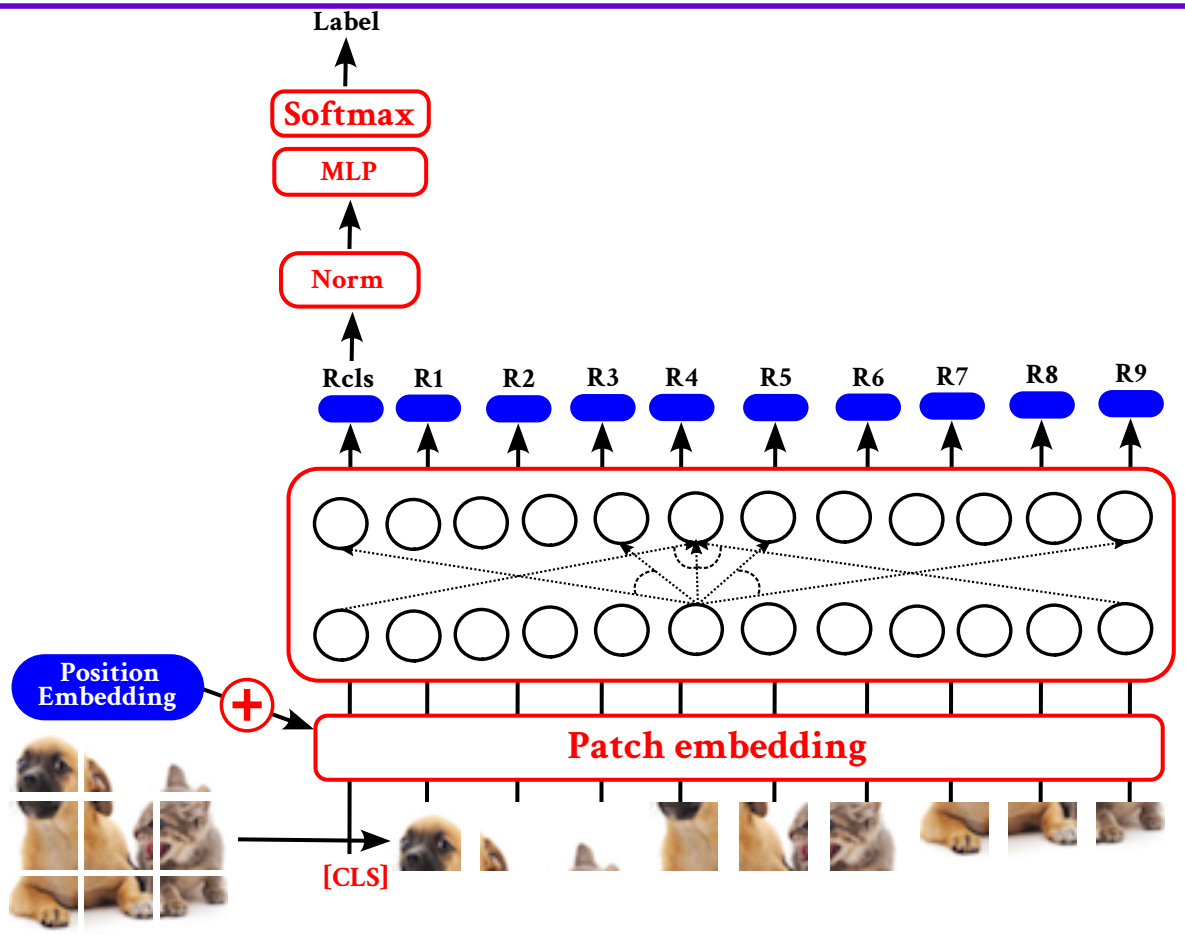


Figure: Example of images encoder [modified from [Zhang et al., 2021]]

Advanced Neural Networks: Attention

Transformer: Pretrained models (GPT: Dec) [Radford et al., 2018]

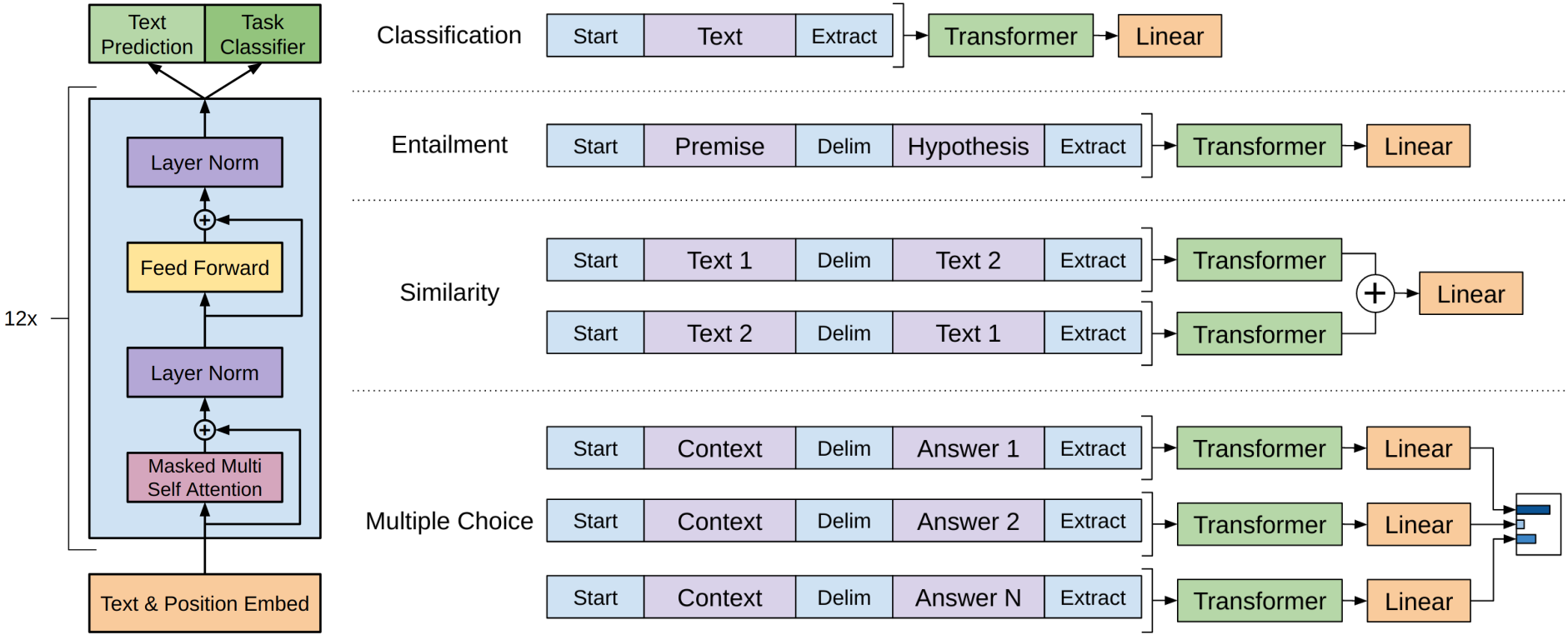


Figure: GPT's architecture and different tasks [Radford et al., 2018]

Advanced Neural Networks: Attention

Transformer: Pretrained models (T5: Enc-Dec) [Raffel et al., 2020]

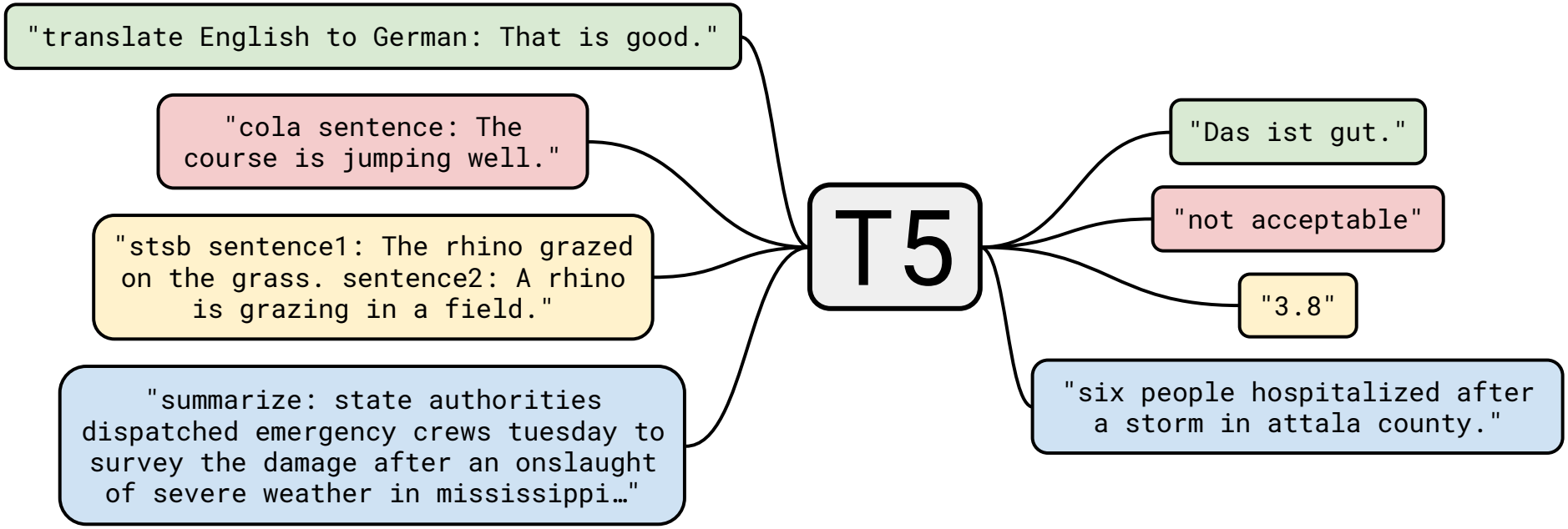


Figure: T5 (Text-to-Text Transfer Transformer) training [Raffel et al., 2020]

Advanced Neural Networks: Attention

Transformer: Pretrained models (BART: Enc-Dec) [Lewis et al., 2020]

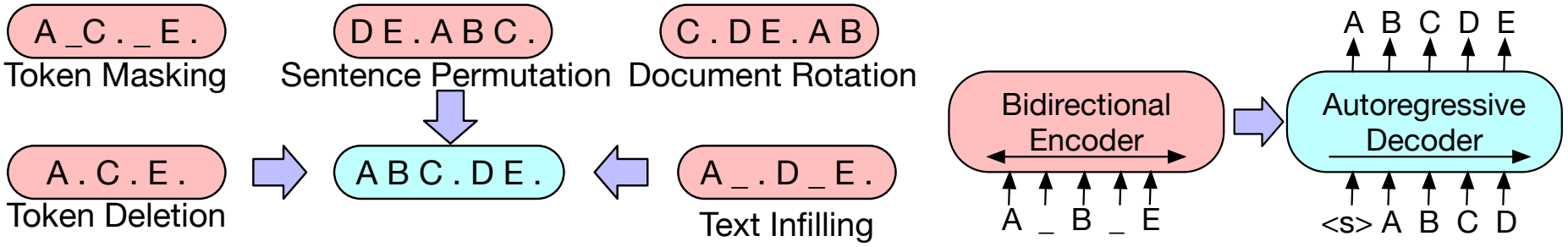


Figure: BART training [Lewis et al., 2020]

Advanced Neural Networks: Attention

Transformer: Some humor

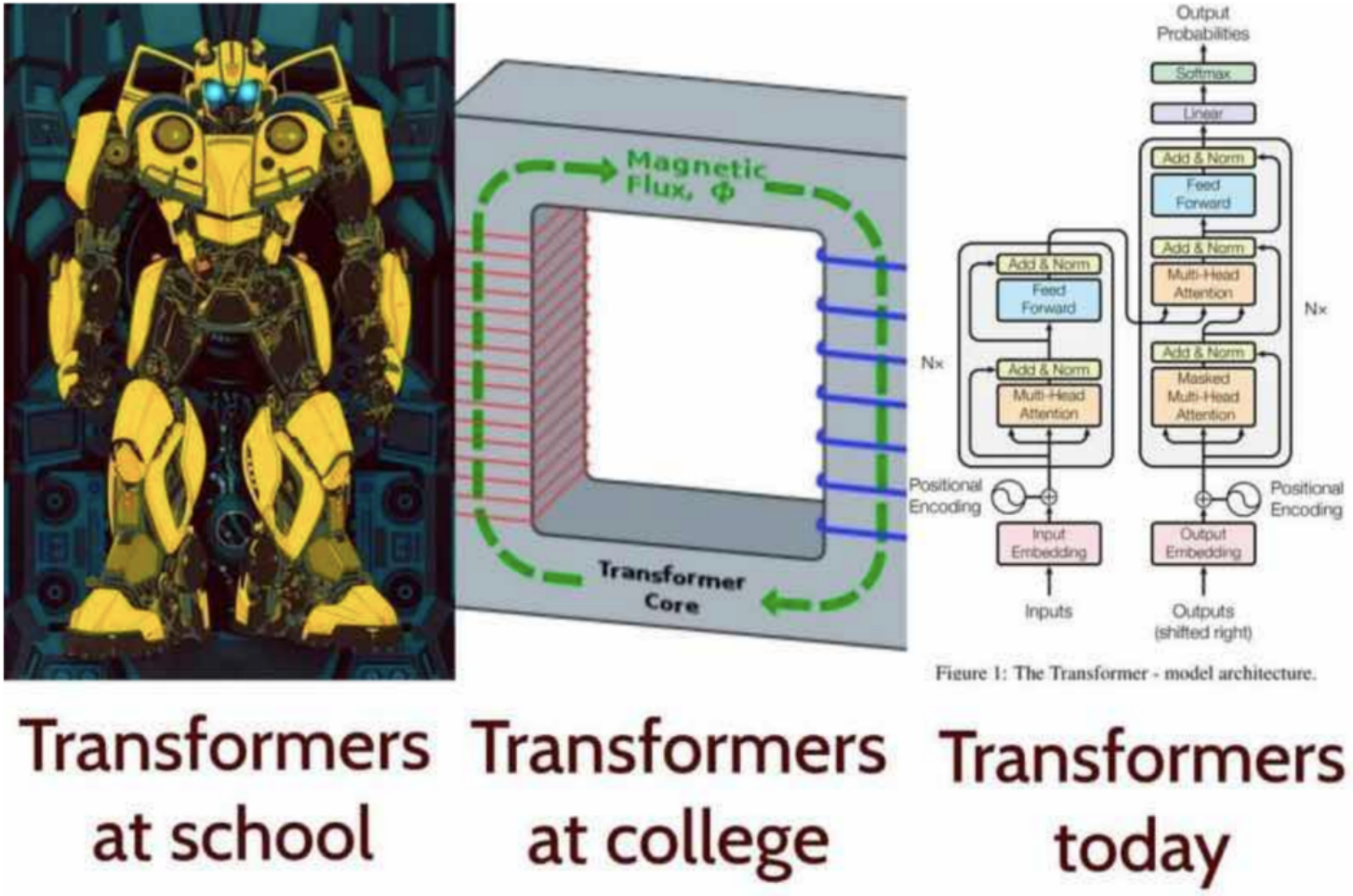


Figure 1: The Transformer - model architecture.

Section 5

Bibliography

Bibliography

Alginahi, Y. (2010).

Preprocessing techniques in character recognition.

Character recognition, 1:1–19.

Amidi, A. and Amidi, S. (2020).

Cs 230 — deep learning.

Online Course.

https://stanford.edu/~shervine/l/fr/teaching/cs-230/ [2020-06-15].

Cain, G., editor (2017).

Artificial Neural Networks: New Research.

Computer Science, Technology and Applications. Nova Science Publishers.

Calin, O. (2020).

Deep Learning Architectures: A Mathematical Approach.

Springer Series in the Data Sciences. Springer International Publishing.

Chollet, F. (2016).

Building autoencoders in keras.

Online Tutorial.

https://blog.keras.io/building-autoencoders-in-keras.html [2020-06-10].

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019).

BERT: Pre-training of deep bidirectional transformers for language understanding.

In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021).

An image is worth 16x16 words: Transformers for image recognition at scale.

In *International Conference on Learning Representations*.

https://arxiv.org/pdf/2010.11929.pdf.

Duchi, J., Hazan, E., and Singer, Y. (2011).

Adaptive subgradient methods for online learning and stochastic optimization.

J. Mach. Learn. Res., 12(null):2121–2159.

Ginsburg, B., Castonguay, P., Hrinchuk, O., Kuchaiev, O., Lavrukhin, V., Leary, R., Li, J., Nguyen, H., Zhang, Y., and Cohen, J. M. (2019).
Training deep networks with stochastic gradient normalized by layerwise adaptive second moments.

Hinton, G., Srivastava, N., and Swersky, K. (2014).
Lecture 6a: Overview of mini-batch gradient descent.
Course "Neural Networks for Machine Learning".
https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf [2024-01-01].

Jaeger, H. (2002).
Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach, volume 5.
Citeseer.

JORDAN, J. (2018).
Variational autoencoders.
Blog.
https://www.jeremyjordan.me/variational-autoencoders/ [2020-06-16].

Kingma, D. P. and Ba, J. (2015).
Adam: A method for stochastic optimization.
In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324.
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf [2020-06-14].

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020).
BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.
In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.
https://aclanthology.org/2020.acl-main.703.

Li, F.-F., Krishna, R., and Xu, D. (2019).
Lecture 5: Convolutional neural networks.
Course "Convolutional Neural Networks for Visual Recognition".
http://cs231n.stanford.edu/slides/2020/lecture_5.pdf [2020-06-15].

Li, Y., Liu, S., Yang, J., and Yang, M.-H. (2017).
Generative face completion.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3911–3919.

Lu, X., Tsao, Y., Matsuda, S., and Hori, C. (2013).
Speech enhancement based on deep denoising autoencoder.
In *Interspeech*, pages 436–440.

Neji, H., Nogueras-Iso, J., Lacasta, J., Ben Halima, M., and Alimi, A. M. (2019).
Adversarial autoencoders for denoising digitized historical documents: The use case of incunabula.
In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, volume 6, pages 31–34.

Orabona, F. and Tommasi, T. (2017).
Training deep networks without learning rates through coin betting.
Advances in Neural Information Processing Systems, 30.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018).
Improving language understanding by generative pre-training.
https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020).
Exploring the limits of transfer learning with a unified text-to-text transformer.
Journal of Machine Learning Research, 21(140):1–67.
http://jmlr.org/papers/v21/20-074.html.

Rosenberg, D. S. (2017).
Loss functions for regression and classification.
Presentation (Bloomberg ML EDU).

Sazlı, M. H. (2006).
A brief review of feed-forward neural networks.
Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering, 50(01).

Sun, S., Cao, Z., Zhu, H., and Zhao, J. (2019).
A survey of optimization methods from a machine learning perspective.
IEEE Transactions on Cybernetics, pages 1–14.

Vasilev, I. (2019).
Advanced Deep Learning with Python : Design and implement advanced next-generation AI solutions using TensorFlow and PyTorch.
Packt.
Codes : https://github.com/ivan-vasilev/advanced-deep-learning-with-python [2020-06-16].


Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017).
Attention is all you need.


Werbos, P. J. (1990).
Backpropagation through time: what it does and how to do it.
Proceedings of the IEEE, 78(10):1550–1560.
http://axon.cs.byu.edu/~martinez/classes/678/Papers/Werbos_BPTT.pdf [2020-06-16].


You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. (2020).
Large batch optimization for deep learning: Training bert in 76 minutes.
https://arxiv.org/abs/1904.00962.


Zaheer, M., Reddi, S., Sachan, D., Kale, S., and Kumar, S. (2018).
Adaptive methods for nonconvex optimization.
Advances in neural information processing systems, 31.


Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021).
Dive into deep learning.
arXiv preprint arXiv:2106.11342.
https://d2l.ai/index.html [2022-05-29].


Zhang, M. R., Lucas, J., Hinton, G., and Ba, J. (2019).
Lookahead Optimizer: K Steps Forward, 1 Step Back.
Curran Associates Inc., Red Hook, NY, USA.


Zhuang, J., Tang, T., Ding, Y., Tatikonda, S. C., Dvornek, N., Papademetris, X., and Duncan, J. (2020).
Adabelief optimizer: Adapting stepsizes by the belief in observed gradients.
Advances in neural information processing systems, 33:18795–18806.

THIS IS A NEURAL NETWORK.

IT MAKES MISTAKES.
IT LEARNS FROM THEM.

BE LIKE A NEURAL NETWORK.