Statistical Machine Learning (BE4M33SSU)
Lecture 6: Artificial Neural Networks

Jan Drchal

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science
Outline

Topics covered in the lecture:

- Neuron types
- Layers
- Loss functions
- Computing loss gradients via backpropagation
- Gradient Descent
- Parameter initialization
- Regularization
Neural Networks Overview

- **Training examples:** \( \mathcal{T}^m = \{(x_i, y_i) \in (\mathcal{X} \times \mathcal{Y}) \mid i = 1, \ldots, m\} \), where \( \mathcal{X} \subseteq \mathbb{R}^n \) and \( \mathcal{Y} \subseteq \mathbb{R}^K \)

- Neural network is a composition of simple linear or non-linear functions (neurons) parametrized by *weights* and *biases*

- Here we consider \( \mathcal{H} \) a hypothesis space of neural networks having a fixed architecture

- Learning methods are based on Empirical Risk Minimization:

\[
R_{\mathcal{T}^m}(h_\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(y_i, h_\theta(x_i)),
\]

where \( h_\theta \in \mathcal{H} \) denotes a neural network parametrized by \( \theta \)
McCulloch-Pitts Perceptron (1943)

\[ \hat{y} = f(s) = f \left( \sum_{i=1}^{n} w_i x_i + b \right) = f \left( \langle w, x \rangle + b \right) \]

- It is the linear classifier we have already seen.
McCulloch-Pitts Perceptron: Treating Bias

- Treat bias as an extra fixed input $x_0 = 1$ weighted $w_0 = b$:
  \[
  \hat{y} = f (\langle w, x \rangle + b) = f (\langle w, x \rangle + w_0 \cdot 1) = f (\langle w', x' \rangle)
  \]

- $x' = (1, x_1, \ldots, x_n)^T \in \mathbb{R}^{n+1}$
- $w' = (w_0, w_1, \ldots, w_n)^T \in \mathbb{R}^{n+1}$
- Unless otherwise noted we will use $x, w$ instead of $x', w'$
**Activation Functions**

- **Step Function**
  - Formula: $1$ if $s > 0$, $0$ otherwise
  - Graph: Constant at $1$ if $s > 0$, $0$ if $s < 0$

- **Bipolar Step Function**
  - Formula: $1$ if $s > 0$, $-1$ if $s < 0$
  - Graph: Constant at $1$ if $s > 0$, Constant at $-1$ if $s < 0$

- **Linear**
  - Formula: $f(s) = s$
  - Graph: Straight line passing through origin with slope $1$

- **Sigmoid**
  - Formula: $\sigma(s) \triangleq \frac{1}{1 + e^{-s}}$
  - Graph: S-shaped curve

- **Hyperbolic Tangent**
  - Formula: $\text{tanh}(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$
  - Graph: S-shaped curve

- **ReLU**
  - Formula: $f(s) = \max(0, s)$
  - Graph: Line at $0$ for $s < 0$, Line at $s$ for $s > 0$

**Logistic sigmoid:** $\sigma(s) \triangleq \frac{1}{1 + e^{-s}} = \frac{e^s}{e^s + 1}$

**Note:** $\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} = 2 \sigma(s) - 1$
Linear Neuron

- Training examples: $\mathcal{T}^m = \{(x_i, y_i) \in (\mathbb{R}^{n+1} \times \mathbb{R}) \mid i = 1, \ldots, m\}$
- Single neuron with linear activation function $\equiv$ linear regression:

$$ \hat{y} = s = \langle x, w \rangle, \quad \hat{y} \in \mathbb{R} $$

- Inputs: $X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1n} \\ 1 & : & \cdots & : \\ 1 & x_{m1} & \cdots & x_{mn} \end{pmatrix} = \begin{pmatrix} x_1^T \\ \vdots \\ x_m^T \end{pmatrix}$
- Targets: $y = (y_1, \ldots, y_m)^T, \quad y_i \in \mathbb{R}$
- Outputs: $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)^T, \quad \hat{y}_i \in \mathbb{R}$
- For the whole dataset we get:

$$ \hat{y} = Xw, \quad \hat{y} \in \mathbb{R}^m $$
Linear Neuron: Maximum Likelihood Estimation

Assumption: data are Gaussian distributed with mean $\langle x_i, w \rangle$ and variance $\sigma^2$:

$$y_i \sim \mathcal{N} (\langle x_i, w \rangle, \sigma^2) = \langle x_i, w \rangle + \mathcal{N} (0, \sigma^2)$$

Likelihood for i.i.d. data:

$$p (y \mid w, X, \sigma) = \prod_{i=1}^{m} p (y_i \mid w, x_i, \sigma) = \prod_{i=1}^{m} (2\pi\sigma^2)^{-\frac{1}{2}} e^{-\frac{1}{2\sigma^2}(y_i-\langle w, x_i \rangle)^2} =$$

$$= (2\pi\sigma^2)^{-\frac{m}{2}} e^{-\frac{1}{2\sigma^2} \sum_{i=1}^{m} (y_i-\langle w, x_i \rangle)^2} =$$

$$= (2\pi\sigma^2)^{-\frac{m}{2}} e^{-\frac{1}{2\sigma^2} (y-Xw)^T (y-Xw)}$$

Negative Log Likelihood (switching to minimization):

$$\mathcal{L} (w) = \frac{m}{2} \log (2\pi\sigma^2) + \frac{1}{2\sigma^2} (y-Xw)^T (y-Xw)$$
**Linear Neuron: Maximum Likelihood Estimation (contd.)**

- Note that

\[
\sum_{i=1}^{m} (y_i - \langle w, x_i \rangle)^2 = (y - Xw)^T (y - Xw)
\]

is the **sum-of-squares** or **squared error** (SE)

- Minimization of \( \mathcal{L}(w) \equiv \text{least squares estimation} \)

- Solving \( \frac{\partial \mathcal{L}}{\partial w} = 0 \) we get \( w^* = (X^T X)^{-1} X^T y \) (see seminar)
Logistic Sigmoid and Probability

- Denote: \( \hat{y} = \sigma(s) \), \( \hat{y} \in (0, 1) \)

- Sigmoid output can represent a parameter of the Bernoulli distribution:

  \[
  p(y|\hat{y}) = \text{Ber}(y|\hat{y}) = \hat{y}^y (1 - \hat{y})^{1-y} = \begin{cases} 
  \hat{y} & \text{for } y = 1 \\
  1 - \hat{y} & \text{for } y = 0
  \end{cases}
  \]

- Models confidence of the positive class \( y = 1 \)

- Binary classifier:

  \[
  h(\hat{y}) = \begin{cases} 
  1 & \text{if } \hat{y} > \frac{1}{2} \\
  0 & \text{else}
  \end{cases}
  \]

\[
\sigma(s) = \frac{1}{1 + e^{-s}}
\]
Logistic Regression

- MCP neuron using sigmoid activation function $\equiv$ logistic regression:

$$ \hat{y} = \sigma(\langle w, x \rangle), \hat{y} \in (0, 1) $$

- Inputs: $X = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1n} \\ \vdots & \ddots & \vdots & \vdots \\ 1 & x_{m1} & \ldots & x_{mn} \end{pmatrix} = \begin{pmatrix} x_{1}^{T} \\ \vdots \\ x_{m}^{T} \end{pmatrix}$

- Target class: $y = \begin{pmatrix} y_{1} \\ \vdots \\ y_{m} \end{pmatrix}^{T}, y_{i} \in \{0, 1\}$

- Output class: $\hat{y} = \begin{pmatrix} \hat{y}_{1} \\ \vdots \\ \hat{y}_{m} \end{pmatrix}^{T}, \hat{y}_{i} \in (0, 1)$

- Note that the logistic regression (including the decision rule) solves actually a classification task
Logistic Regression MLE Leads to the Cross-Entropy

- Likelihood, for the logistic regression:

  \[ p(y|w, X) = \prod_{i=1}^{m} \text{Ber}(y_i|\hat{y}_i) = \prod_{i=1}^{m} \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1-y_i} \]

- Negative Log Likelihood:

  \[ \mathcal{L}(w) = \sum_{i=1}^{m} - \left[ y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) \right] \]

- This loss function is called the **cross-entropy**

- The \( \ell(y_i, \hat{y}_i) \) is the negative log probability of the correct answer \( y_i \in \{0, 1\} \) given by the model output \( \hat{y}_i \in (0, 1) \)
Maximum Likelihood Estimation

- Maximum Likelihood Estimation: \( w^* = \arg\min_w L(w) \)

- Derivative of the loss w.r.t. to the sigmoid argument:
  \[
  \frac{\partial L}{\partial s_i} = \hat{y}_i - y_i \quad \text{(see seminar)}
  \]

- Gradient w.r.t. logistic regression parameters:
  \[
  \frac{\partial L}{\partial w} = \sum_{i=1}^{m} \frac{\partial L}{\partial s_i} \cdot \frac{\partial s_i}{\partial w} = \sum_{i=1}^{m} x_i(\hat{y}_i - y_i) = X^T(\hat{y} - y)
  \]

- \( \frac{\partial L}{\partial w} = 0 \) has no analytical solution \( \implies \) use numerical methods
Linear Layer

- Multiple linear neurons: linear (dense, fully-connected, affine) layer
- Output $k$: $\hat{y}_k = \langle x, w_k \rangle$, $k = 1, 2, \ldots, K$
- All outputs using weight matrix $W$: $\hat{y} = x^T W$
- Multiple samples: $\hat{Y} = XW$

$$W = (w_1 \ldots w_K) = \begin{pmatrix} w_{01} & \cdots & w_{0K} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nK} \end{pmatrix}$$

$$X = \begin{pmatrix} x_1^T \\ \vdots \\ x_m^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1n} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \cdots & x_{mn} \end{pmatrix}$$

$$\hat{Y} = \begin{pmatrix} \hat{y}_1^T \\ \vdots \\ \hat{y}_m^T \end{pmatrix} = \begin{pmatrix} \hat{y}_{11} & \cdots & \hat{y}_{1K} \\ \vdots & \ddots & \vdots \\ \hat{y}_{m1} & \cdots & \hat{y}_{mK} \end{pmatrix}$$
Multinominal classification, $K$ mutually exclusive classes

Definition: $\sigma_k(s) \triangleq \frac{e^{s_k}}{\sum_{c=1}^{K} e^{s_c}}$, where $K$ is the number of classes

Softmax represents a categorical probability distribution: $\sigma_k \in (0, 1)$ for $k \in \{1 \ldots K\}$ and $\sum_{k=1}^{K} \sigma_k = 1$

Describes class membership probabilities: $p(y = k|s) = \sigma_k(s)$

Softmax input (vector $s$) is often called called the *logits*
Softmax Layer MLE

- **Target:** \( \mathbf{y} = (y_1 \ldots y_m)^T, y_i \in \{1, 2, \ldots, K\} \)
- **One-hot encoding** for sample \( i \) and class \( k \): let \( y_{ik} \triangleq [y_i = k] \)
- **Likelihood:**

\[
p(\mathbf{y}|\mathbf{w}, \mathbf{X}) = \prod_{i=1}^m \prod_{c=1}^K \hat{y}_{ic}^{y_{ic}}
\]

- **Negative Log Likelihood:**

\[
\mathcal{L}(\mathbf{w}) = - \sum_{i=1}^m \sum_{c=1}^K y_{ic} \log(\hat{y}_{ic})
\]

Again the **cross-entropy**

- See seminar for the gradient
**Multinominal Logistic Regression**

- **linear layer + softmax layer = multinominal logistic regression:**

\[ \hat{y}_k = \sigma_k(x^T W) \]

- **Classifier:**

\[ h(x, W) = \underset{k}{\text{argmax}} \hat{y}_k \]
## Loss Functions: Summary

<table>
<thead>
<tr>
<th>problem</th>
<th>output</th>
<th>suggested loss function</th>
</tr>
</thead>
</table>
| binary classification| sigmoid neuron | cross-entropy  
$$- \frac{1}{m} \sum_{i=1}^{m} [y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)]$$ |
| multinominal classification| softmax | multinominal cross-entropy  
$$- \frac{1}{m} \sum_{i=1}^{m} \sum_{c=1}^{K} y_{ic} \log(\hat{y}_{ic})$$ |
| regression          | linear neuron | mean squared error  
$$\frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$ |
| multi-output regression | linear layer | mean squared error  
$$\frac{1}{m} \sum_{i=1}^{m} \sum_{c=1}^{K} (y_{ic} - \hat{y}_{ic})^2$$ |
Multilayer Perceptron (MLP)

- Feed-forward ANN
- Fully-connected layers
- MLP for regression would typically use linear output layer
Recurrent Neural Network (RNN)

- Fully-Connected Recurrent Neural Network (FRNN)
- Both inputs and outputs are sequences
- Feedback connections → memory (similarly to sequential circuitry)
Modular and Hierarchical Architectures

- Directed Acyclic Graphs (DAGs)
- Layers can be organized in *modules*
- Hierarchies of modules, module reuse
Backpropagation Overview

- A method to compute a gradient of the *loss function* with respect to its parameters: $\nabla \mathcal{L}(w)$
- $\nabla \mathcal{L}(w)$ is in turn used by optimization methods like gradient descent
- Here, we present the "modular" backpropagation (see Nando de Freitas’ Machine Learning course: https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/)
- Let us use multinominal logistic regression as an example
The loss function is the multinominal cross-entropy in this case:

\[
\mathcal{L}(\mathbf{w}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{c=1}^{K} [y_i = c] \log \left( \frac{\exp (\langle \mathbf{x}_i, \mathbf{w}_c \rangle)}{\sum_{k=1}^{K} \exp (\langle \mathbf{x}_i, \mathbf{w}_k \rangle)} \right)
\]
Backpropagation Based on Modules

- Computation of $\nabla L(w)$ involves repetitive use of the *chain rule*
- We can make things simpler by divide and conquer approach
- Divide to simplest possible modules (that can be later combined into complex networks)
- Represent even the loss function as a module
- Passing messages

![Diagram](image)
Let $\delta^l = \frac{\partial L}{\partial z^l}$ be the sensitivity of the loss to the module input for layer $l$, then:

$$\delta_i^l = \frac{\partial L}{\partial z_i^l} = \sum_j \frac{\partial L}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial z_i^l} = \sum_j \delta_j^{l+1} \frac{\partial z_j^{l+1}}{\partial z_i^l}$$

We need to know how to compute derivatives of outputs w.r.t. inputs only!
Backpropagation: Parameters

- Similarly if the module has parameters we want to know how the loss changes w.r.t. them:

\[
\frac{\partial \mathcal{L}}{\partial w^l_i} = \sum_j \frac{\partial \mathcal{L}}{\partial z^{l+1}_j} \cdot \frac{\partial z^{l+1}_j}{\partial w^l_i} = \sum_j \delta^{l+1}_j \frac{\partial z^{l+1}_j}{\partial w^l_i}
\]

- Derivatives of module outputs w.r.t. to the parameters are all we need
Backpropagation: Steps

- So for each module we need only to specify these three messages:

  forward: \( z^{l+1} = f(z^l) \)

  backward: \( \frac{\partial z^{l+1}}{\partial z^l} \)

  parameter (optional): \( \frac{\partial z^{l+1}}{\partial w^l} \)

1. **Forward Pass**

   \( z^1 = x_1 \)

   \( z^2 = f_1(z^1) \)

   \( z^3 = f_2(z^2) \)

   \( \mathcal{L} = z^4 = f_3(z^3) \)

2. **Backward Pass** (Backpropagation)

   \( \delta^1 = \frac{\partial \mathcal{L}}{\partial z^1} \)

   \( \delta^2 = \frac{\partial \mathcal{L}}{\partial z^2} \)

   \( \delta^3 = \frac{\partial \mathcal{L}}{\partial z^3} \)

   \( \delta^4 = \frac{\partial \mathcal{L}}{\partial z^4} \)

3. **Parameters**
Example: Linear Layer

- **forward**: $z_{j}^{l+1} = \sum_{i=0}^{n} w_{ij} z_{i}^{l}, \quad j = 1, \ldots, K$

- **backward**: $\frac{\partial z_{j}^{l+1}}{\partial z_{i}^{l}} = w_{ij}, \quad i = 0, \ldots, n, \quad j = 1, \ldots, K$

- **parameter**: $\frac{\partial z_{j}^{l+1}}{\partial w_{ik}} = [j = k] z_{i}^{l}$
Example: Squared Error

- **forward**: \( z^{l+1} = (y - z^l)^2 \)

- **backward**: \( \frac{\partial z^{l+1}}{\partial z^l} = 2(z^l - y), \quad i \in \{1, \ldots, n\} \)
Gradient Descent (GD)

- **Task**: find parameters which minimize loss over the training dataset:

\[
\theta^* = \arg\min_{\theta} L(\theta)
\]

where \( \theta \) is a set of all parameters defining the ANN

- Gradient descent: \( \theta_{k+1} = \theta_k - \alpha_k \nabla L(\theta_k) \)
  where \( \alpha_k > 0 \) is the learning rate or stepsize at iteration \( k \)

- More about (Stochastic) Gradient Descent the next time . . .
Parameter Initialization

- Is it a good idea to set initially all weights to a constant?
- **No.** All neurons would behave the same: the same $\delta$s are backpropagated. We need to *break the symmetry*

- Use small random numbers, e.g., sample from a Gaussian distribution with zero mean:
  - works well for shallow networks,
  - for deep networks we might get into trouble
Gaussian Initialization Example

- MLP, 10 \texttt{tanh} layers, 500 units each. Each input fed with \( \mathcal{N}(0, 1) \)
- Weights initialized to \( \mathcal{N}(0, \sigma^2) \); showing layer output distributions

\[
\sigma = 1
\]

\[
\sigma = 0.01
\]
Vanishing Gradient

- For large $\sigma$ (saturation) the $\tanh$ derivative is almost zero
- For small $\sigma$ (output close to zero):
  - the derivative is at most 1,
  - the weights are very small and $\frac{\partial z_{j}^{l+1}}{\partial z_{i}^{l}} = w_{ij}$ holds for the preceding linear layer
- In both cases: $\delta \to 0$ as the number of layers increases
Rectified Linear Unit (ReLU)

- \( f(s) = \max(0, s) \)
- Helps with the *vanishing gradients* problem: the gradient is constant for \( s > 0 \), while for sigmoid-like activations it becomes increasingly small
- Fast to compute
- Leads to sparse representations: \( s < 0 \) turns the neuron completely off which blocks the gradient propagation → dead units → Leaky ReLU
- Unbounded: use regularization to prevent numerical problems
Xavier Initialization

- Glorot and Bengio: *Understanding the difficulty of training deep feedforward neural networks*, 2010
- For the linear neuron $s = \sum_i w_i x_i$, let $w_i$ and $x_i$ be independent random variables, $w_i$ and $x_i$ are i.i.d., $\mathbb{E}(x_i) = \mathbb{E}(w_i) = 0$:

$$
\text{Var}(s) = \text{Var} \left( \sum_i w_i x_i \right) = \sum_i \text{Var}(w_i x_i) = \\
= \sum_i \mathbb{E} \left( [w_i x_i - \mathbb{E}(w_i x_i)]^2 \right) = \sum_i \mathbb{E} \left( [w_i x_i - \mathbb{E}(w_i)\mathbb{E}(x_i)]^2 \right) = \\
= \sum_i \mathbb{E}(w_i^2 x_i^2) = \sum_i \mathbb{E}(w_i^2)\mathbb{E}(x_i^2) = \\
= \sum_i \mathbb{E}([w_i - \mathbb{E}(w_i)]^2)\mathbb{E}([x_i - \mathbb{E}(x_i)]^2) = \\
= \sum_i \text{Var}(x_i)\text{Var}(w_i) = n_{in}\text{Var}(x)\text{Var}(w)
$$
Xavier Initialization (contd.)

- We have $\text{Var}(s) = n_{in}\text{Var}(x)\text{Var}(w)$
- We want $\text{Var}(s) = \text{Var}(x)$
- Choose $\text{Var}(w) = \frac{1}{n_{in}}$
- Works well for $\tanh$ as it is linear near zero
- Do not forget to standardize ANN input data

$$\tanh$$
Regularization

How to deal with overfitting?

- get more data
- find simpler model, search for optimal architecture, e.g., number, type and size of layers
- constrain model by *regularization*

Most types of regularization are based on constraining the parameter space

Bayesian point of view: introduce prior distribution on model parameters
L2 Regularization (Weight Decay): Motivation

- Limit hypothesis space by limiting the size of the weight vector
- For L2 regularization we will push down $w^Tw = \|w\|^2_2$
- You already know this from SVMs!
- **Intuition**: sigmoid-like neurons kept near zero potential (via small weights) behave similarly to linear neurons
- L2 regularization (weight decay): zero mean Gaussian prior

\[ \sigma(s) = \frac{1}{1 + e^{-s}} \]
Example: L2 Regularization for Linear Regression

- Recall the linear regression likelihood:

\[
p(y|w, X) = \left(2\pi\sigma^2\right)^{-\frac{m}{2}} e^{-\frac{1}{2\sigma^2}(y-Xw)^T(y-Xw)}
\]

- Define a Gaussian prior with zero mean and variance \(\sigma_0^2\) for the parameters:

\[
p(w) = \left(2\pi\sigma_0^2\right)^{-\frac{1}{2}} e^{-\frac{1}{2\sigma_0^2}w^Tw}
\]

- Then the posterior is:

\[
p(w|y, X) = \frac{p(y|w, X) \cdot p(w)}{p(y|X)}
\]

The denominator does not depend on the parameters \(w\):

\[
p(w|y, X) \propto p(y|w, X) \cdot p(w)
\]
MAP Estimate

- Maximizing $p(w|y, X)$ gives us the Maximum a posteriori (MAP) estimate:

$$w_{MAP} = \arg\max_w p(w|y, X) = \arg\min_w (-\log p(w|y, X))$$

where

$$-\log p(w|y, X) = \frac{1}{2\sigma^2} (y-Xw)^T(y-Xw) + \frac{1}{2\sigma_0^2}w^Tw + C$$

- We can omit $C$, define $\lambda = \frac{\sigma^2}{\sigma_0^2}$ and minimize the loss function:

$$\mathcal{L}(w) = (y-Xw)^T(y-Xw) + \lambda w^Tw$$

- The term $\lambda w^Tw = \lambda \|w\|^2_2$ minimizes the size of the weight vector

- Note that we omit bias in $\lambda w^Tw$
Recall the solution for the linear regression $w^* = (X^TX)^{-1}X^Ty$

What if $X^TX$ has no inverse?

We can modify the solution by adding a small element to the diagonal:

$$w^* = (X^TX + \lambda I)^{-1}X^Ty, \quad \lambda > 0$$

It turns out that the solution is the minimizer of our regularized loss function:

$$\mathcal{L}(w) = (y - Xw)^T(y - Xw) + \lambda w^Tw,$$

see seminar for the derivation
Other Regularization Related Approaches

- L1 regularization: sum absolute values, i.e., use $\lambda \| w \|_1$
- Randomize inputs: same as the weight decay for linear neurons
- Dataset augmentation
- Early stopping: start with small weights, stop when validation loss starts to grow, often used for limited time-budget
- Weight sharing and sparse connectivity: Convolutional Neural Networks (next lecture)
- Model averaging (see lectures on Ensembling)
- Dropout and DropConnect
Next Lecture

- Stochastic Gradient Descent
- Deep Neural Networks
- Convolutional Neural Networks
- Transfer learning
\[ y = \sum_{i=1}^{n} w_i x_i + b \]
$w_0 = b$

$x_1$

$w_1$

$x_2$

$w_2$

$\vdots$

$x_n$

$w_n$

$\sum$

$s$

$\hat{y}$
Step Function

Bipolar Step Function

Linear

\[ f(s) = s \]

Sigmoid

Hyperbolic Tangent

ReLU

\[ \sigma(s) = \frac{1}{1 + e^{-s}} \]

\[ f(s) = \max(0, s) \]
$0.8x + 2 + \mathcal{N}(0, 1)$
\[ \sigma(s) = \frac{1}{1 + e^{-s}} \]
\[
\sum \hat{y} = w_{01} + w_{n1} x_1 + w_{n2} x_2 + \cdots + w_{nK} x_n
\]
softmax

\[ s_1 \rightarrow 1 \rightarrow \hat{y}_1 \]

\[ s_2 \rightarrow 2 \rightarrow \hat{y}_2 \]

\[ s_3 \rightarrow 3 \rightarrow \hat{y}_3 \]

\[ \ldots \]

\[ s_K \rightarrow K \rightarrow \hat{y}_K \]
\[ z^1 = x_1 \]
\[ z^2 = f_1(z^1) \]
\[ z^3 = f_2(z^2) \]
\[ z^4 = f_3(z^3) \]
\[ L = z^4 \]
\[ \delta^1 = \frac{\partial L}{\partial z^1} \]
\[ \delta^2 = \frac{\partial L}{\partial z^2} \]
\[ \delta^3 = \frac{\partial L}{\partial z^3} \]
\[ \delta^4 = \frac{\partial L}{\partial z^4} = 1 \]
\[ z^1 = x_1 \]

\[ z^2 = f_1(z^1) \]

\[ z^3 = f_2(z^2) \]

\[ \mathcal{L} = z^4 = f_3(z^3) \]

\[ \delta^1 = \frac{\partial \mathcal{L}}{\partial z^1} \]

\[ \delta^2 = \frac{\partial \mathcal{L}}{\partial z^2} \]

\[ \delta^3 = \frac{\partial \mathcal{L}}{\partial z^3} \]

\[ \delta^4 = \frac{\partial \mathcal{L}}{\partial z^4} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1 \]
1. Forward Pass

\[ z^1 = x_1 \]
\[ z^2 = f_1(z^1) \]
\[ z^3 = f_2(z^2) \]
\[ L = z^4 = f_3(z^3) \]

2. Backward Pass

(Backpropagation)

\[ \delta^1 = \frac{\partial L}{\partial z^1} \]
\[ \delta^2 = \frac{\partial L}{\partial z^2} \]
\[ \delta^3 = \frac{\partial L}{\partial z^3} \]
\[ \delta^4 = \frac{\partial L}{\partial z^4} \]

3. Parameters
tanh(x)
$\tanh'(x)$
\[ f(s) = \max(0, s) \]
\[ \sigma(s) = \frac{1}{1 + e^{-s}} \]