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

- 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:

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

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

\[ x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n \]
\[ w = (w_1, w_2, \ldots, w_n)^T \in \mathbb{R}^n \]
\[ b \in \mathbb{R} \]
\[ s = \langle w, x \rangle + b \in \mathbb{R} \]
\[ f(s) = \begin{cases} -1 & \text{if } s < 0 \\ 1 & \text{else} \end{cases} \]
\[ \hat{y} = h_{(w,b)}(x) \in \{-1, 1\} \]
\[ \hat{y} = f(s) = f \left( \sum_{i=1}^{n} w_i x_i + b \right) = f \left( \langle w, x \rangle + b \right) \]

- It is the linear classifier we have already seen.
Treat bias as an extra fixed input $x_0 = 1$ weighted $w_0 = b$:

$$\hat{y} = f \left( \langle \mathbf{w}, \mathbf{x} \rangle + b \right) = f \left( \langle \mathbf{w}, \mathbf{x} \rangle + w_0 \cdot 1 \right) = f \left( \langle \mathbf{w}', \mathbf{x}' \rangle \right)$$

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

- **Step Function**
- **Bipolar Step Function**
- **Linear**
  
  \[ f(s) = s \]

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

- **Hyperbolic Tangent**

  \[ f(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} = 2 \sigma(2s) - 1 \]

- **ReLU**

- **Logistic sigmoid**: \[ \sigma(s) \triangleq \frac{1}{1 + e^{-s}} = \frac{e^s}{e^s + 1} \]
Linear Neuron

- Training examples: $\mathcal{T}^m = \{(x_i, y_i) \in (\mathbb{R}^{n+1} \times \mathbb{R}) | i = 1, \ldots, m\}$

- Single neuron with linear activation function $\equiv$ linear regression:

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

- Inputs: $X = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1n} \\ 1 & : & \ldots & : \\ 1 & x_{m1} & \ldots & x_{mn} \end{pmatrix} = \begin{pmatrix} x_1^T \\ \vdots \\ x_m^T \end{pmatrix}$

- 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:

\[
\hat{y} = Xw, \quad \hat{y} \in \mathbb{R}^m
\]
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:

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

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

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

Negative Log Likelihood (switching to minimization):

$$\mathcal{L}(w) = \frac{m}{2} \log (2\pi\sigma^2) + \frac{1}{2\sigma^2} (y-Xw)^T(y-Xw)$$
Note that

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

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

Minimization of \( \mathcal{L}(w) \equiv \) 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:

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

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

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

- Inputs: $\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1n} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \ldots & x_{mn} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_m^T \end{pmatrix}$

- Target class: $\mathbf{y} = (y_1, \ldots, y_m)^T, y_i \in \{0, 1\}$

- Output class: $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_m)^T, \hat{y}_i \in (0, 1)$

- Note that the logistic regression (including the decision rule) actually solves 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:

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

- 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: \( w^* = \arg \min_{w} \mathcal{L}(w) \)

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

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

- \( \frac{\partial \mathcal{L}}{\partial w} = 0 \) 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: $\hat{Y} = X W$

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

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

$$\hat{Y} = \begin{pmatrix} \hat{y}_1^T \\ \vdots \\ \hat{y}_m^T \end{pmatrix} = \begin{pmatrix} \hat{y}_{11} & \cdots & \hat{y}_{1K} \\ \vdots & \ddots & \vdots \\ \hat{y}_{m1} & \cdots & \hat{y}_{mK} \end{pmatrix}$$
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: \( \mathbf{y} = \left( y_1 \ldots y_m \right)^T, y_i \in \{1, 2, \ldots, K\} \)

- 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})
  
  \text{Again the} \text{cross-entropy}
  
- See seminar for the gradient
Multinominal Logistic Regression

- linear layer + softmax layer = multinominal logistic regression:

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

Classifier: \( h(\mathbf{x}, \mathbf{W}) = \arg\max_k \hat{y}_k \)
## 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 \[
- \frac{1}{m} \sum_{i=1}^{m} \left[ y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) \right] \] |
| 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/](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:

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

- Computation of $\nabla \mathcal{L}(w)$ 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 of a module with inputs $z^l_{\text{in}}$, output $z^{l+1}_{\text{out}}$, and gradients $\delta^l$, $\delta^{l+1}$]
**Backpropagation: Backward Pass Message**

- 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:

\[
\frac{\partial L}{\partial w^l_i} = \sum_j \frac{\partial L}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial w^l_i} = \sum_j \delta_j^{l+1} \frac{\partial z_j^{l+1}}{\partial w^l_i}
\]

- 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**

\[
\begin{align*}
z^1 &= x_1 \\
z^2 &= f_1(z^1) \\
z^3 &= f_2(z^2) \\
\mathcal{L} &= z^4 = f_3(z^3)
\end{align*}
\]

2. **Backward Pass** (Backpropagation)

\[
\begin{align*}
\delta^1 &= \frac{\partial \mathcal{L}}{\partial z^1} \\
\delta^2 &= \frac{\partial \mathcal{L}}{\partial z^2} \\
\delta^3 &= \frac{\partial \mathcal{L}}{\partial z^3} \\
\delta^4 &= \frac{\partial \mathcal{L}}{\partial z^4}
\end{align*}
\]

3. **Parameters**

\[
\begin{align*}
\frac{\partial \mathcal{L}}{\partial w^1}
\end{align*}
\]
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:

\[ \theta^* = \operatorname{argmin}_\theta \mathcal{L}(\theta) \]

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

- Gradient descent: \( \theta_{k+1} = \theta_k - \alpha_k \nabla \mathcal{L}(\theta_k) \)
  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 initially set 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 \text{tanh} layers, 500 units each. Each input fed with \( N(0, 1) \)
- Weights initialized to \( N(0, \sigma^2) \); 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: $\delta \rightarrow 0$ 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 $\rightarrow$ dead units $\rightarrow$ Leaky ReLU
- Unbounded: use regularization to prevent numerical problems
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 $\text{tanh}$ as it is linear near zero
- Do not forget to standardize ANN input data

\[ \text{tanh} \]
Regularization

How to deal with overfitting?

- get more data
- find a 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 $\mathbf{w}^T \mathbf{w} = \|\mathbf{w}\|_2^2$
- 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 \( \sigma^2_0 \) for the parameters:

\[ p(w) = \left(2\pi\sigma^2_0\right)^{-\frac{1}{2}} e^{-\frac{1}{2\sigma^2_0}w^Tw} \]

- 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:

$$w_{MAP} = \argmax_w p(w|y, X) = \argmin_w ( - \log p(w|y, X) )$$

where

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

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

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

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

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

What if $X^T X$ has no inverse?

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

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

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

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

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
$w_0 = b$

$x_1$

$w_1$

$x_2$

$w_2$

\[ \vdots \]

$x_n$

$w_n$

$\sum_s \rightarrow \ \text{sigmoid} \rightarrow \ \hat{y}$
Step Function

Bipolar Step Function

Linear

\[ f(s) = s \]

Sigmoid

Hyperbolic Tangent

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

ReLU

\[ f(s) = \max(0, s) \]
$0.8x + 2 + \mathcal{N}(0, 1)$
\[ \sigma(s) = \frac{1}{1 + e^{-s}} \]
\[ \sum \hat{y}_k \]
softmax

\[ s_1 \]
\[ s_2 \]
\[ s_3 \]
\[ \ldots \]
\[ s_K \]

\[ \hat{y}_1 \]
\[ \hat{y}_2 \]
\[ \hat{y}_3 \]
\[ \ldots \]
\[ \hat{y}_K \]
\( z^1 = x_1 \)

\[
\begin{align*}
\sum & \quad \text{layer 1} \\
\frac{\partial L}{\partial z^1} & \quad \delta^1 = \frac{\partial L}{\partial z^1} \\
\frac{\partial L}{\partial w^1} & \\
\end{align*}
\]

\[
\begin{align*}
z^2 & = f_1(z^1) \\
\frac{\partial L}{\partial z^2} & \quad \delta^2 = \frac{\partial L}{\partial z^2} \\
\end{align*}
\]

\[
\begin{align*}
z^3 & = f_2(z^2) \\
\frac{\partial L}{\partial z^3} & \quad \delta^3 = \frac{\partial L}{\partial z^3} \\
\end{align*}
\]

\[
\begin{align*}
\mathcal{L} & = z^4 = f_3(z^3) \\
\frac{\partial L}{\partial z^4} & \quad \delta^4 = \frac{\partial L}{\partial z^4} = \frac{\partial L}{\partial \mathcal{L}} = 1 \\
\end{align*}
\]
\[ L = z^4 = f_3(z^3) \]

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

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

\[ z^1 = x_1 \]

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

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

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

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

layer 1

layer 2

softmax

loss

layer 3
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

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