Neural network parsing

Daniël de Kok & Erhard Hinrichs

Recap

$c \leftarrow c_0(S)$
while $\neg isFinal(c)$ **do**
    $t \leftarrow g(c)$
    $c \leftarrow t(c)$
end while

- c: parser state
- c_0: initial parser state
- t: transition
- g: parsing guide

- Next question: what function do we use as g?
- Answer: a probabilistic model: $p(t|c)$
- Benefits:
  - Well-understood methods for optimization.
  - Provides a good basis for beam search.

Data-driven models

Introduction

- Goal of of a guide during parsing: predict for a given c the best transition t.
- Using properties (**features**) of the parser state.

Example (parsing features)

Consider following two parsing features:

- f_1: 1 if $\sigma = [\cdots|\text{ART}|\text{NN}]$, 0 otherwise
- f_2: 1 if $\sigma = [\cdots|\text{APPR}|\text{NE}]$, 0 otherwise

Example (parsing features)

Consider following two parsing features:

- f_1: 1 if $\sigma = [\cdots|\text{ART}|\text{NN}]$, 0 otherwise
- f_2: 1 if $\sigma = [\cdots|\text{APPR}|\text{NE}]$, 0 otherwise

For a parser state with $\sigma = [\ldots, \text{APPR}, \text{NE}]$:

- $f_1(c_i) = 0$
- $f_2(c_i) = 1$
- $\mathbf{x}_i = [0, 1]$

Example (parsing features)

Consider following two parsing features:

- f_1: 1 if $\sigma = [\cdots|\text{ART}|\text{NN}]$, 0 otherwise
- f_2: 1 if $\sigma = [\cdots|\text{APPR}|\text{NE}]$, 0 otherwise

For a parser state with $\sigma = [\ldots, \text{ART}, \text{NN}]$:

- $f_1(c_i) = 1$
- $f_2(c_i) = 0$
- $\mathbf{x}_i = [1, 0]$

Feature-based rules

- Given the features:
  - f_1: 1 if $\sigma = [\cdots|\text{ART}|\text{NN}]$, 0 otherwise
  - f_2: 1 if $\sigma = [\cdots|\text{APPR}|\text{NE}]$, 0 otherwise
- $\textsc{Left-Arc}_{\text{DET}}$ when $f_1 = 1$

Feature-based rules

- Given the features:
  - f_1: 1 if $\sigma = [\cdots|\text{ART}|\text{NN}]$, 0 otherwise
  - f_2: 1 if $\sigma = [\cdots|\text{APPR}|\text{NE}]$, 0 otherwise
- $\textsc{Left-Arc}_{\text{DET}}$ when $f_1 = 1$
- $\textsc{Right-Arc}_{\text{PN}}$ when $f_2 = 1$

Data-driven learning

For each sentence S_i and the corresponding gold standard dependency structure g_i:

1. Parse S_i using an oracle to reproduce G_i.
2. Extract the transition sequence $C_{0,m}$ and the corresponding transitions $T_{0,m}$.
3. For each pair $(c_i, t_i) \in (C_{0,m}, T_{0,m})$:
   1. Convert c_i to a feature vector $\mathbf{x}_i \in \mathbb{R}^d$.
   2. Convert t_i to a natural number $y_i \in \mathbb{N}$.
   3. $(\mathbf{x}_i, y_i)$ is a training instance.

Example

Transition	σ	β	A
	[ROOT]	[Staatsanwalt, . . .]	$\emptyset$
SH	[ROOT, Staatsanwalt]	[muß, $\cdots$]	$\emptyset$
SH	[ROOT, Staatsanwalt, muß]	[AWO-Konten, $\cdots$]	$\emptyset$
LA_{SUBJ}	[ROOT, muß]	[AWO-Konten, $\cdots$]	$A_1 = \{(\text{muß}, \text{SUBJ}, \text{Staatsanwalt})\}$
SH	[ROOT, AWO-Konten]	[prüfen]	A_1
SH	[ROOT, AWO-Konten, prüfen]	[]	A_1
LA_{OBJA}	[ROOT, muß, prüfen]	[]	$A_2 = A_1 \cup \{(\text{prüfen}, \textit{OBJA}, \text{AWO-Konten})\}$
RA_{AUX}	[ROOT, muß]	[]	$A_3 = A_2 \cup \{(\text{muß}, \textit{AUX}, \text{prüfen})\}$
RA_{ROOT}	[ROOT]	[]	$A_4 = A_3 \cup \{(\text{ROOT}, \textit{ROOT}, \text{muß})\}$

Example

Transition	σ	β	A
	[ROOT]	[Staatsanwalt, . . .]	$\emptyset$
SH	[ROOT, Staatsanwalt]	[muß, · · ·]	$\emptyset$
SH	[ROOT, Staatsanwalt, muß]	[AWO-Konten, · · ·]	$\emptyset$
LA_{SUBJ}	[ROOT, muß]	[AWO-Konten, · · ·]	$A_1 = \{(\text{muß}, \text{SUBJ}, \text{Staatsanwalt})\}$
SH	[ROOT, AWO-Konten]	[prüfen]	A_1
SH	[ROOT, AWO-Konten, prüfen]	[]	A_1
LA_{OBJA}	[ROOT, muß, prüfen]	[]	$A_2 = A_1 \cup \{(\text{prüfen}, \textit{OBJA}, \text{AWO-Konten})\}$
RA_{AUX}	[ROOT, muß]	[]	$A_3 = A_2 \cup \{(\text{muß}, \textit{AUX}, \text{prüfen})\}$
RA_{ROOT}	[ROOT]	[]	$A_4 = A_3 \cup \{(\text{ROOT}, \textit{ROOT}, \text{muß})\}$

- The tuple $< \sigma, \beta, A >$ on each line is a parser state.
- The transition on the next line is the corresponding class.

Features

- Features are extracted from a parser state c_i using feature templates.
- Features can use any portion of the parser state:
  - Any token on the stack/buffer.
  - Attributes of a token, such as its part-of-speech tag or lemma.

Example feature set (Kübler, McDonald, and Nivre 2009)

Address	Form	Lemma	POS-tag	Features	Deprel
σ_0	+	+	+	+	
σ_1			+		
LDEP[σ_0]					+
RDEP[σ_0]					+
β_0	+	+	+	+	
β_1	+		+		
β_2			+		
β_3			+		
LDEP[β_0]					+
RDEP[β_0]					+

Feature vectors

- To use techniques from linear algebra, we represent each feature as a fixed component in a vector.

Feature vectors

- To use techniques from linear algebra, we represent each feature as a fixed component in a vector.
- We can represent four features in a vector of four components:

a. $\sigma = [\cdots|\text{die}|\text{AWO}]$
b. $\sigma = [\cdots|\text{in}|\text{Japan}]$
c. $\sigma = [\cdots|\text{das}|\text{Auto}]$
d. $\sigma = [\cdots|\text{in}|\text{Sofia}]$
e. $\sigma = [\cdots|\text{Sofia}],\ \beta = [\text{gewesen}|\cdots]$

$$\begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}$$

Feature vectors

- The parser state
  - $\sigma = [\cdots, \text{in}, \text{Sofia}]$
  - $\beta = [\text{gewesen}, \cdots]$
- Would correspond to the following feature vector:

a. $\sigma = [\cdots|\text{die}|\text{AWO}]$
b. $\sigma = [\cdots|\text{in}|\text{Japan}]$
c. $\sigma = [\cdots|\text{das}|\text{Auto}]$
d. $\sigma = [\cdots|\text{in}|\text{Sofia}]$
e. $\sigma = [\cdots|\text{Sofia}],\ \beta = [\text{gewesen}|\cdots]$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Softmax regression

Basic linear model

- We introduce a matrix $\mathbf{W} \in \mathbb{R}^{|T| \times |F|}$, where:
  - $|T|$: the number of transitions.
  - $|F|$: the number of features.

Basic linear model

- We introduce a matrix $\mathbf{W} \in \mathbb{R}^{|T| \times |F|}$, where:
  - $|T|$: the number of transitions.
  - $|F|$: the number of features.
- Each row of $\mathbf{W}$ is a transition-specific weighting of feature vectors.

Basic linear model

- We introduce a matrix $\mathbf{W} \in \mathbb{R}^{|T| \times |F|}$, where:
  - $|T|$: the number of transitions.
  - $|F|$: the number of features.
- Each row of $\mathbf{W}$ is a transition-specific weighting of feature vectors.
- If $\mathbf{x}$ is the feature vector of a parser state, then a simple guide would be:

$$g(\mathbf{x}) = \arg\max_{t \in T} (\mathbf{W}\mathbf{x})_t$$

The softmax function

The **softmax** function squashes the components of a vector $\mathbf{u}$, such that the components sum up to 1:

$$\text{softmax}(\mathbf{u})_i = \frac{e^{\mathbf{u}_i}}{\sum_k e^{\mathbf{u}_k^u}}$$

The softmax function

The **softmax** function squashes the components of a vector $\mathbf{u}$, such that the components sum up to 1:

$$\text{softmax}(\mathbf{u})_i = \frac{e^{\mathbf{u}_i}}{\sum_k e^{\mathbf{u}_k^u}}$$

- Softmax can represent a categorical probability distribution $p(y|\mathbf{x})$:
  - All output values are in the range $(0, 1)$.
  - $\sum_k \text{softmax}(\mathbf{u})_k = 1$
  - $\text{softmax}(\mathbf{u})_y$ is the probability of class y.

Softmax for classification

To convert our earlier scoring function, $\mathbf{Wx}$ to a probabilistic model:

Softmax for classification

To convert our earlier scoring function, $\mathbf{Wx}$ to a probabilistic model: apply softmax.

Softmax for classification

To convert our earlier scoring function, $\mathbf{W}\mathbf{x}$ to a probabilistic model: apply softmax.

$$
\begin{aligned}
p(y|x) &= \text{softmax}(u)_y \\
&= \text{softmax}(\mathbf{W}\mathbf{x})_y \\
&= \frac{e^{\mathbf{W}_y \mathbf{x}}}{\sum_k e^{\mathbf{W}_k \mathbf{x}}}
\end{aligned}
$$

Softmax for classification

$$
\begin{aligned}
p(y|x) &= \text{softmax}(u)_y \\
&= \text{softmax}(\mathbf{W}\mathbf{x})_y \\
&= \frac{e^{\mathbf{W}_y \mathbf{x}}}{\sum_k e^{\mathbf{W}_k \mathbf{x}}}
\end{aligned}
$$

Known as:

- **logistic regression** in statistics;
- **softmax layer** in neural networks;
- **conditional maximum entropy model** in CL.

Computing the softmax regression

Given $\mathbf{x} \in \mathbb{R}^4$ and $y \in \mathbb{R}^3$:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} & W_{1,4} \\ W_{2,1} & W_{2,2} & W_{2,3} & W_{2,4} \\ W_{3,1} & W_{3,2} & W_{3,3} & W_{3,4} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \right)$$

$$= \text{softmax}\left(\begin{bmatrix} W_{1,1} \cdot x_1 + W_{1,2} \cdot x_2 + W_{1,3} \cdot x_3 + W_{1,4} \cdot x_4 \\ W_{2,1} \cdot x_1 + W_{2,2} \cdot x_2 + W_{2,3} \cdot x_3 + W_{2,4} \cdot x_4 \\ W_{3,1} \cdot x_1 + W_{3,2} \cdot x_2 + W_{3,3} \cdot x_4 + W_{3,4} \cdot x_4 \end{bmatrix} \right)$$

Example

$$\mathbf{W} = \begin{bmatrix} 0.3 & -0.1 & 0.4 & -0.2 \\ -0.2 & 0.6 & 0.5 & -0.9 \\ 0.3 & -0.1 & 0.1 & 0.4 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Example

$$\mathbf{W} = \begin{bmatrix} 0.3 & -0.1 & 0.4 & -0.2 \\ -0.2 & 0.6 & 0.5 & -0.9 \\ 0.3 & -0.1 & 0.1 & 0.4 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{W}\mathbf{x}^\intercal = \begin{bmatrix} 0.1 & -1.1 & 0.7 \end{bmatrix}$$

Example

$$\mathbf{W} = \begin{bmatrix} 0.3 & -0.1 & 0.4 & -0.2 \\ -0.2 & 0.6 & 0.5 & -0.9 \\ 0.3 & -0.1 & 0.1 & 0.4 \end{bmatrix}, \mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{W}\mathbf{x}^\intercal = \begin{bmatrix} 0.1 & -1.1 & 0.7 \end{bmatrix}$$

$$\text{softmax}(\mathbf{W}\mathbf{x})^\intercal = \begin{bmatrix} 0.32 & 0.10 & 0.58 \end{bmatrix}$$

Training objective

- In order to train the model, we minimize the negative log likelihood:

$$-\mathcal{L}(W) = -\sum_{i=1}^{n} \log P(y^i|\boldsymbol{x}^i)$$

Training objective

- In order to train the model, we minimize the negative log likelihood:

$$-\mathcal{L}(W) = -\sum_{i=1}^{n} \log P(y^i|\boldsymbol{x}^i)$$

$$= -\sum_{i=1}^{n} \log \frac{e^{\boldsymbol{W}_{y^i}\boldsymbol{x}^i}}{\sum_{k \in Y} e^{\boldsymbol{W}_k \boldsymbol{x}^i}}$$

Training objective

- In order to train the model, we minimize the negative log likelihood:

$$
\begin{aligned}
-\mathcal{L}(W) &= -\sum_{i=1}^{n} \log P(y^i|\boldsymbol{x}^i) \\
&= -\sum_{i=1}^{n} \log \frac{e^{\boldsymbol{W}_{y^i}\boldsymbol{x}^i}}{\sum_{k\in Y} e^{\boldsymbol{W}_k \boldsymbol{x}^i}} \\
&= -\sum_{i=1}^{n} \boldsymbol{W}_{y^i}\boldsymbol{x}^i - \log \sum_{k\in Y} e^{\boldsymbol{W}_k \boldsymbol{x}^i}
\end{aligned}
$$

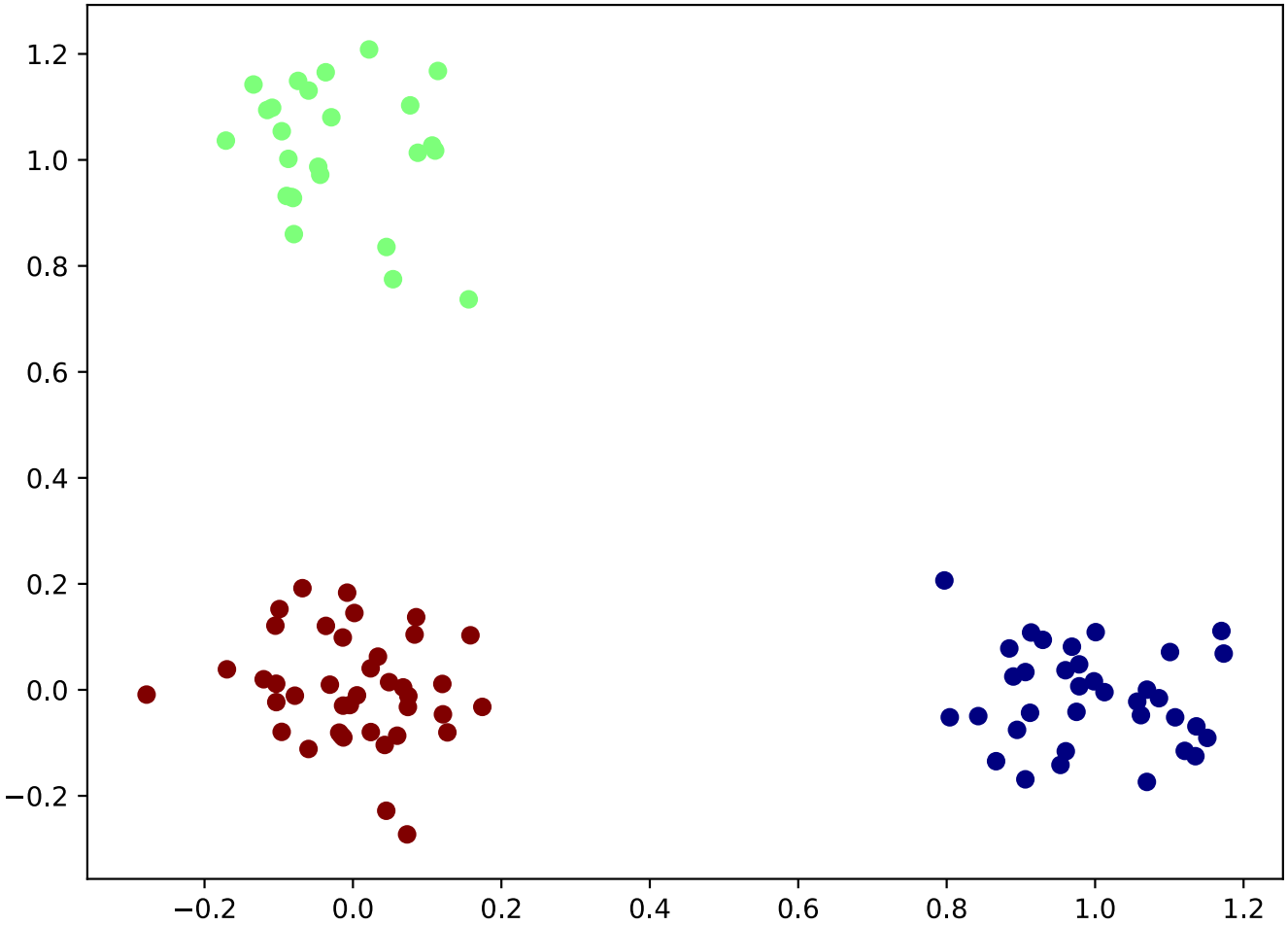
Training objective

- In order to train the model, we minimize the negative log likelihood:

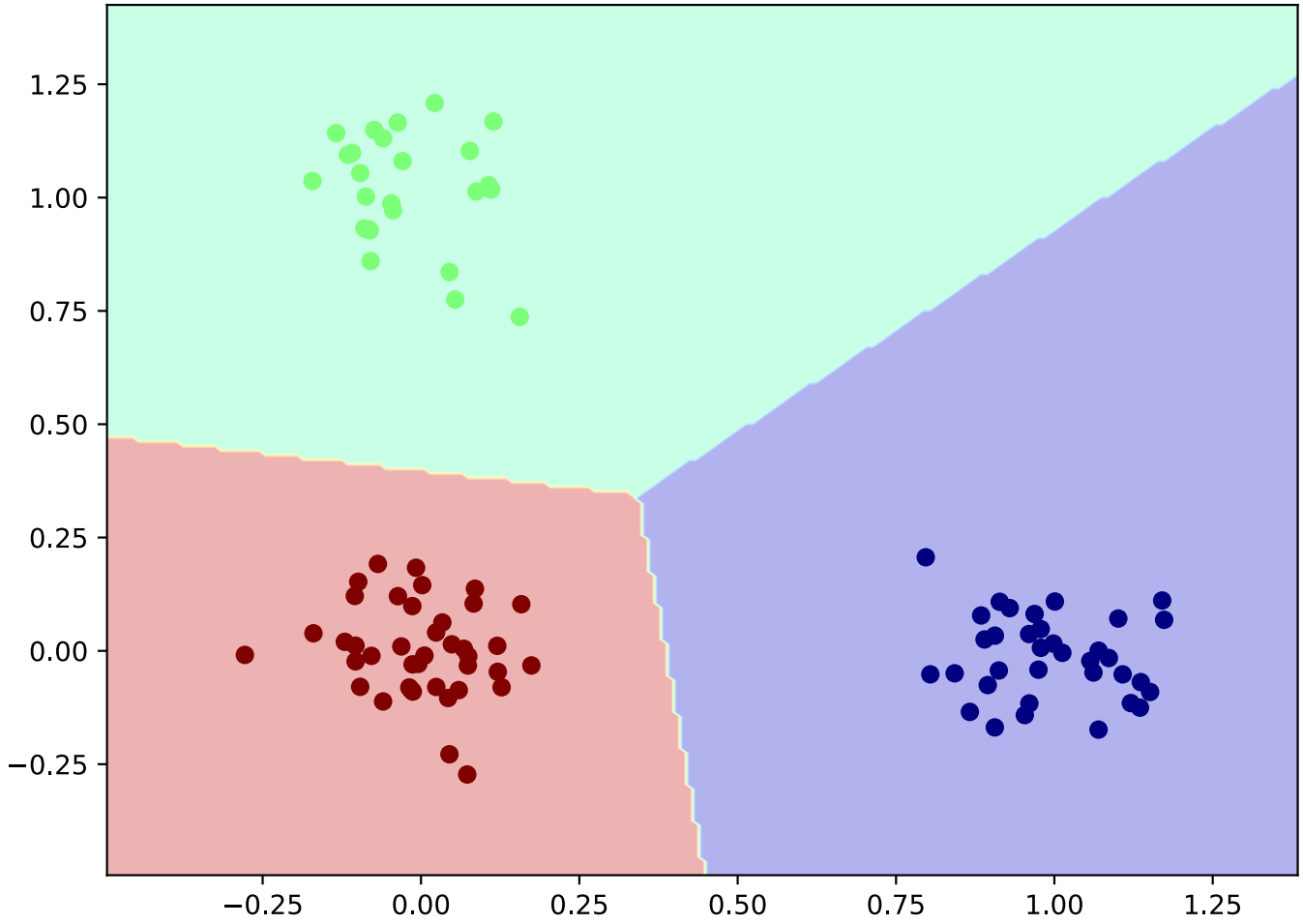
$$
\begin{aligned}
-\mathcal{L}(W) &= -\sum_{i=1}^{n} \log P(y^i|\boldsymbol{x}^i) \\
&= -\sum_{i=1}^{n} \log \frac{e^{\boldsymbol{W}_{y^i}\boldsymbol{x}^i}}{\sum_{k\in Y} e^{\boldsymbol{W}_k \boldsymbol{x}^i}} \\
&= -\sum_{i=1}^{n} \boldsymbol{W}_{y^i}\boldsymbol{x}^i - \log \sum_{k\in Y} e^{\boldsymbol{W}_k \boldsymbol{x}^i}
\end{aligned}
$$

- Such functions can be minimized using a library such as Tensorflow, PyTorch, or Dynet.

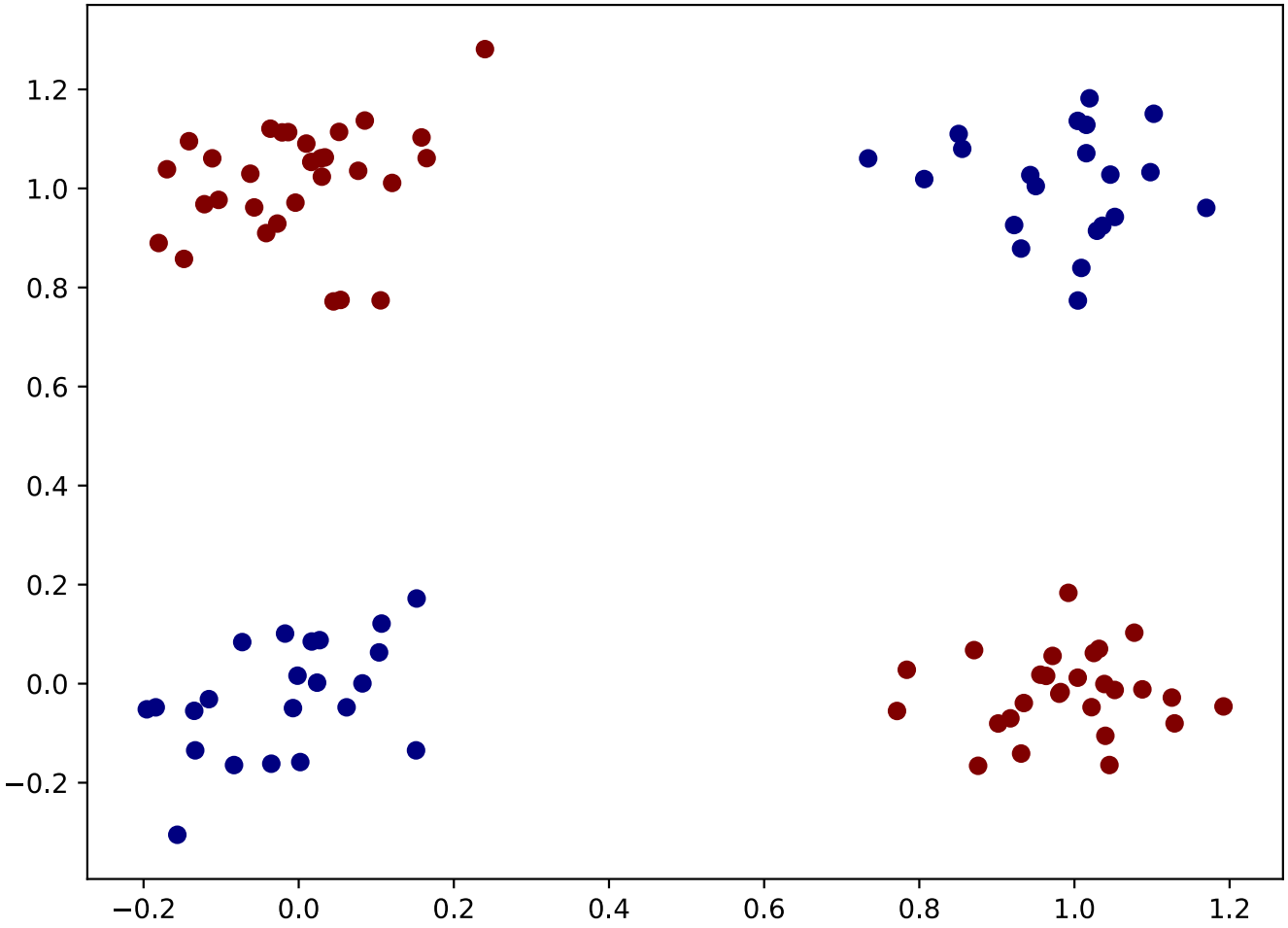
Sample classification task



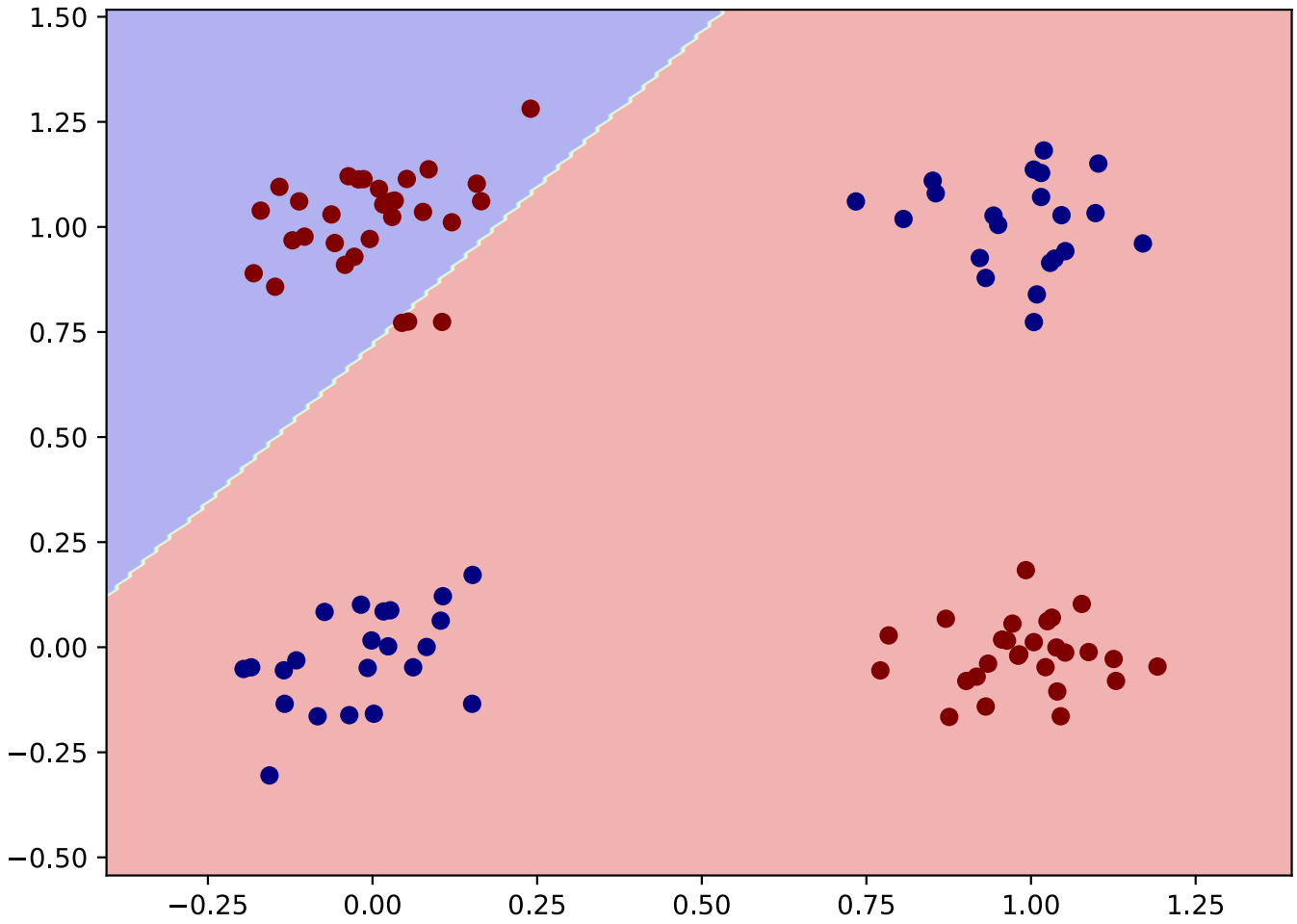
Softmax regression solution



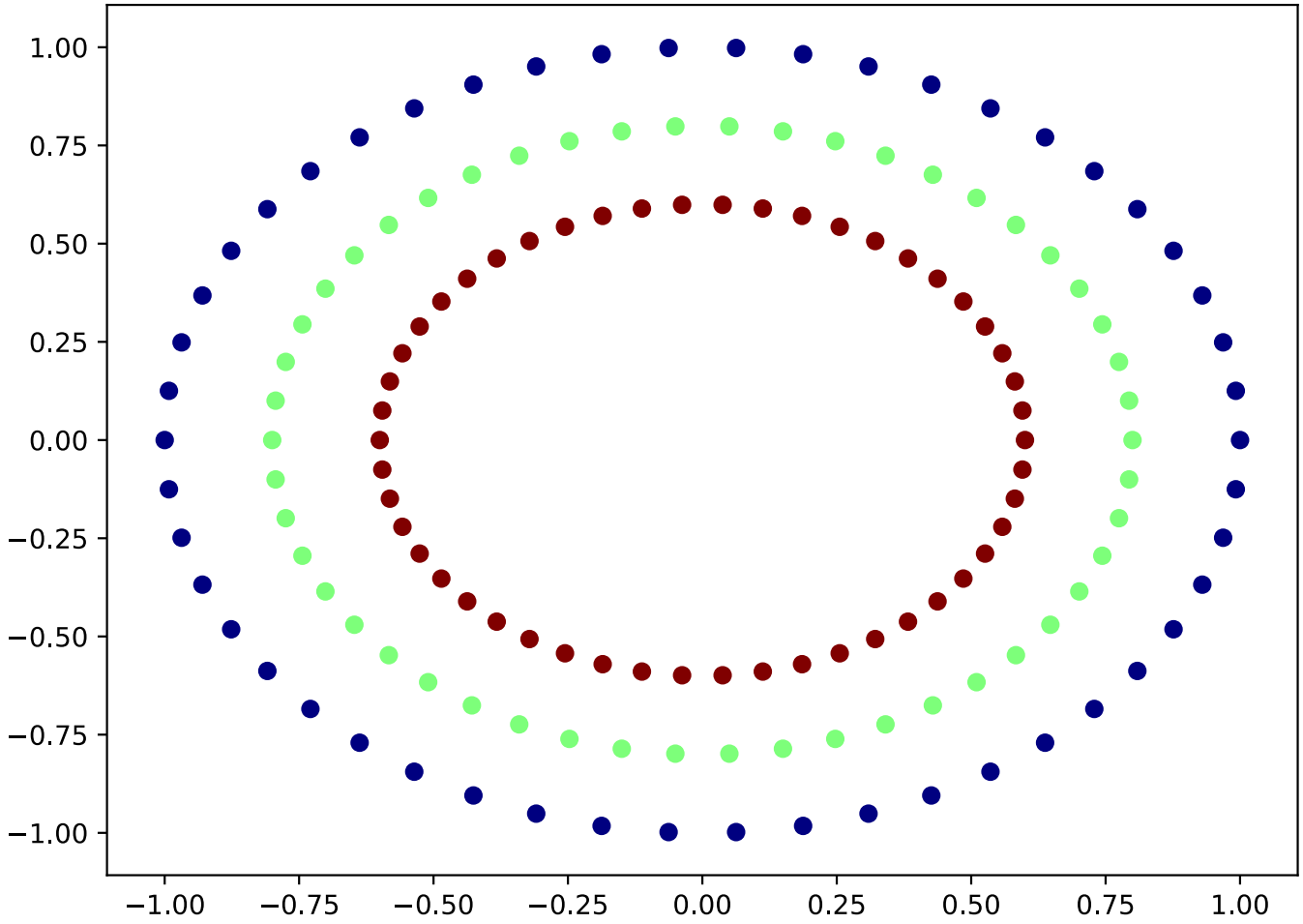
`xor` classification task (Minski and Papert 1969)



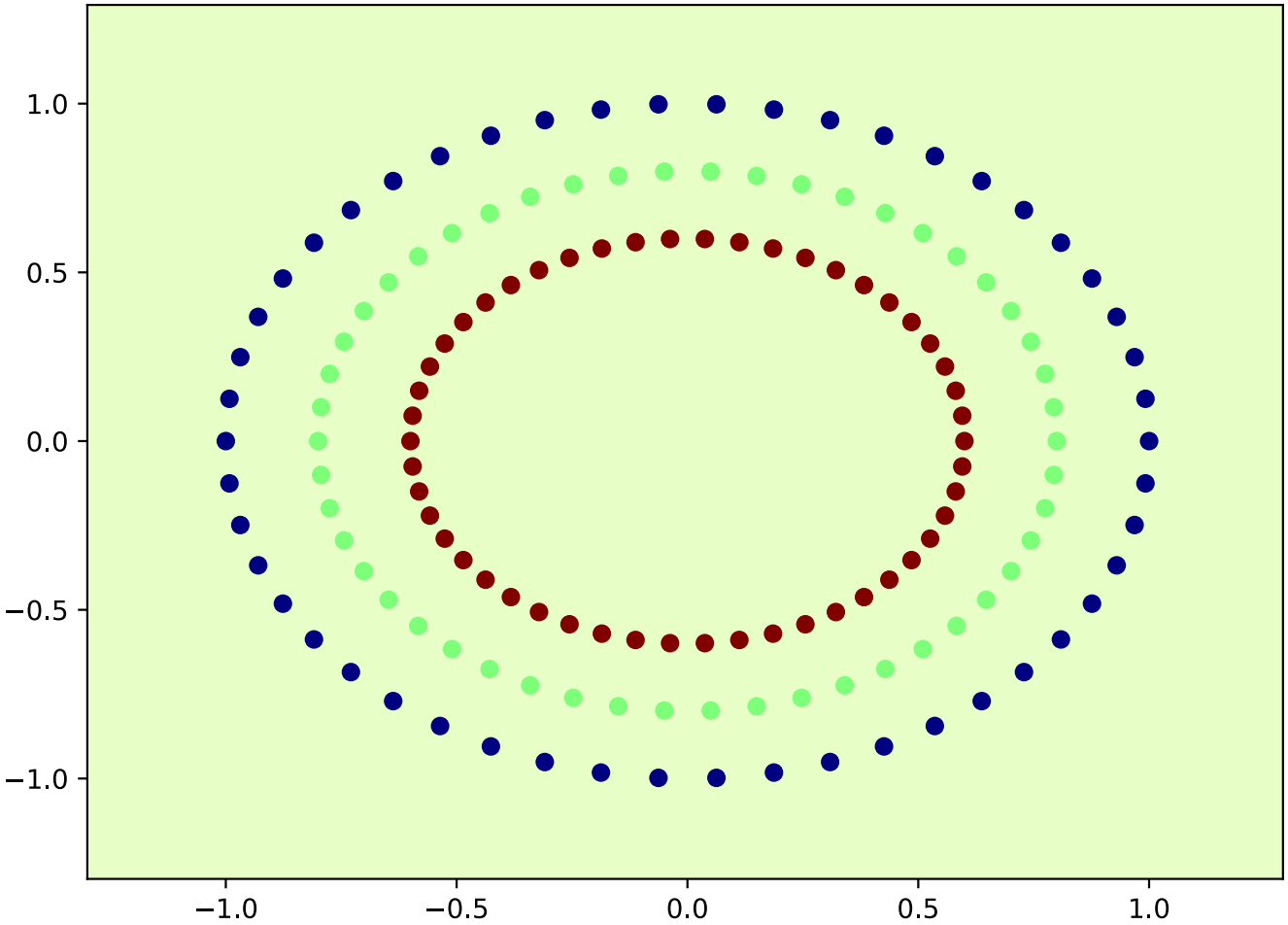
xor softmax regression solution



Three circles classification task



Three circles softmax regression solution



Linear classification and parsing

Linear models fail at modeling more complex interactions between primitive features.

Consider four features:

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

Linear classification and parsing (2)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

Typical transitions:

- $\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}^\intercal$: $\textsc{Left-Arc}_{\text{subj}}$
- $\begin{bmatrix}0 & 0 & 1 & 1\end{bmatrix}^\intercal$: $\textsc{Left-Arc}_{\text{subj}}$

Linear classification and parsing (2)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

Typical transitions:

- $\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}^\intercal$: $\textsc{Left-Arc}_{\text{subj}}$
- $\begin{bmatrix}0 & 0 & 1 & 1\end{bmatrix}^\intercal$: $\textsc{Left-Arc}_{\text{subj}}$

Example weight vector:
$\mathbf{W}_t = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}$

Linear classification and parsing (3)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\mathbf{W}_t = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Linear classification and parsing (3)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$\mathbf{W}_t = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$

What if we get the following input?

$\begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}^\intercal$

Linear classification and parsing (3)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$\mathbf{W}_t = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$

What if we get the following input?

$\begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}^\intercal$

- Results in a high probability of $\textsc{Left-Arc}_{\text{subj}}$.
- Clearly incorrect: cars don't buy things.
- This linear model fails at capturing the necessary dependencies between features.

Linear classification and parsing

To make this work in linear classification, we need **higher order features**:

1. $\sigma = [\cdots, \text{Autos}, \_] \wedge \sigma = [\cdots, \text{hupen}] \rightarrow \sigma = [\cdots, \text{Autos}, \text{hupen}]$
2. $\sigma = [\cdots, \text{wir}, \_] \wedge \sigma = [\cdots, \text{kaufen}] \rightarrow \sigma = [\ldots, \text{wir}, \text{kaufen}]$

Linear classification and parsing

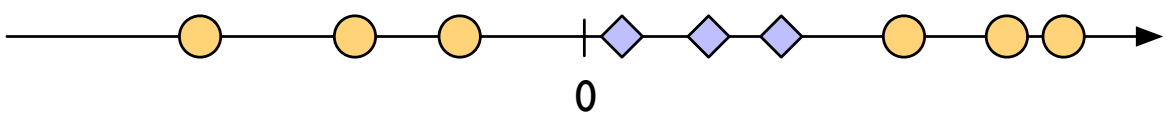
To make this work in linear classification, we need **higher order features**:

1. $\sigma = [\cdots, \text{Autos}, \_] \wedge \sigma = [\cdots, \text{hupen}] \rightarrow \sigma = [\cdots, \text{Autos}, \text{hupen}]$
2. $\sigma = [\cdots, \text{wir}, \_] \wedge \sigma = [\cdots, \text{kaufen}] \rightarrow \sigma = [\ldots, \text{wir}, \text{kaufen}]$

- Results in large, sparse feature vectors.
- Requires feature engineering: what addresses/layers to combine into a feature?
- A better model would capture such feature interactions.

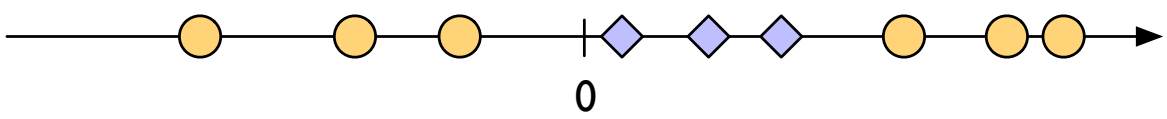
Feed-forward neural networks

Introduction



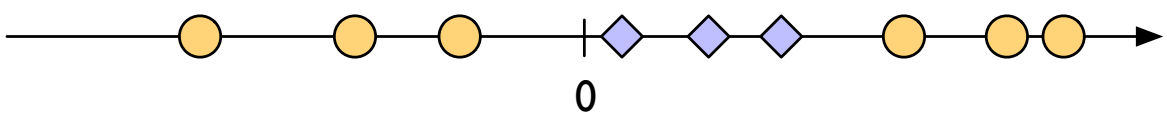
- One feature, data is not separable with a linear function.

Introduction



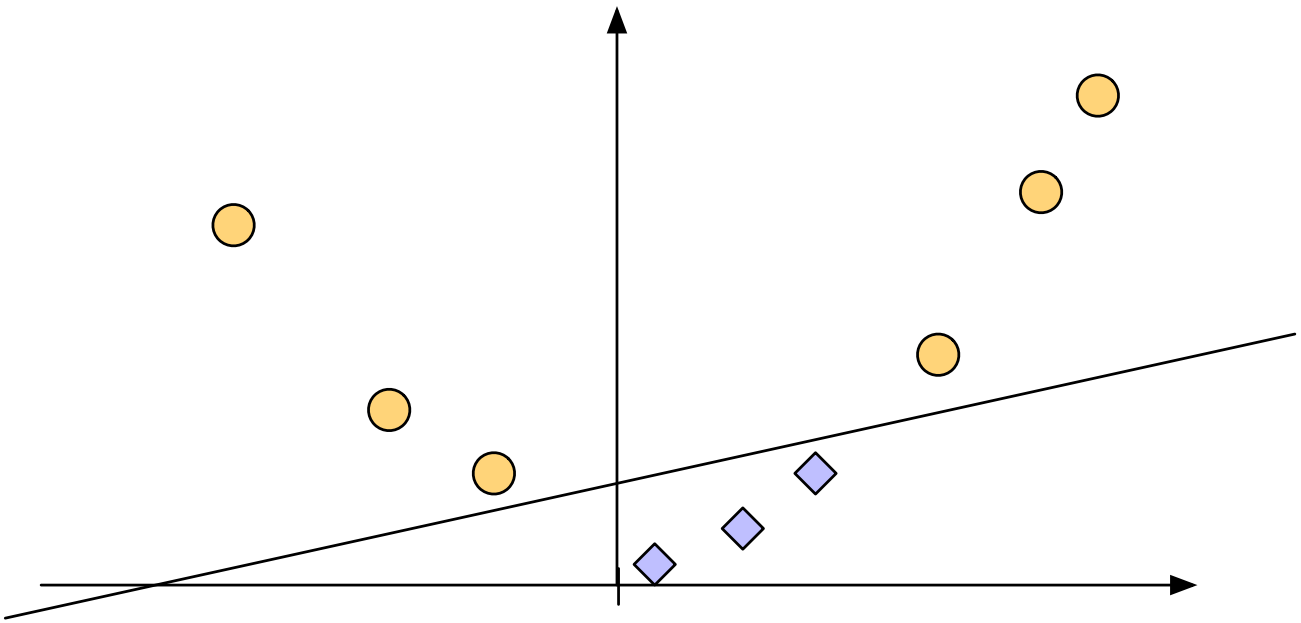
- One feature, data is not separable with a linear function.
- Can we make the data linearly separable using a transformation?

Introduction



- One feature, data is not separable with a linear function.
- Can we make the data linearly separable using a transformation?
- Use a quadratic function to map the data to a two-dimensional space.

Introduction



- After this transformation, we can separate the data using a linear function.

Non-linear transformations

$g(\mathbf{W}^h\mathbf{u})$ is a non-linear transformation of $\mathbf{u}$, where:

- $\mathbf{u} \in \mathbb{R}^i$
- $\mathbf{W^h} \in \mathbf{R}^{h \times i}$: a weight matrix, h is a hyperparameters that determines the number of transformations of the input.
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$: a non-linear element-wise function.

Non-linear transformations

$g(\mathbf{W}^h\mathbf{u})$ is a non-linear transformation of $\mathbf{u}$, where:

- $\mathbf{u} \in \mathbb{R}^i$
- $\mathbf{W^h} \in \mathbf{R}^{h \times i}$: a weight matrix, h is a hyperparameters that determines the number of transformations of the input.
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$: a non-linear element-wise function.

$g(\mathbf{W}^h\mathbf{u})$ is called a **hidden layer** in neural networks.

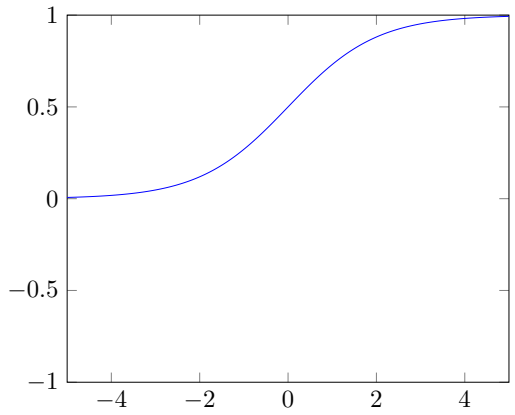
Feed forward neural network

A **feed-forward neural network** combines one or more such hidden layers and a final layer, such as softmax.

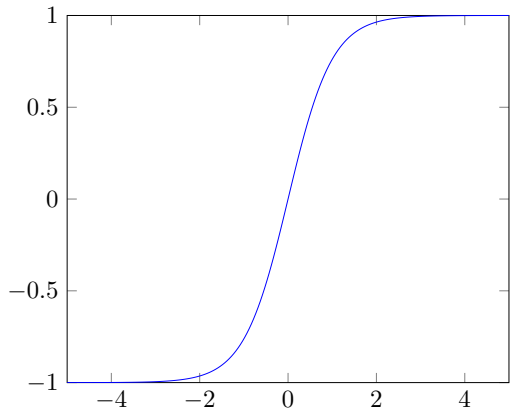
- Network with one hidden layer: $\text{softmax}(W g(W^h x))$
- Network with two hidden layers: $\text{softmax}(W g(W^{h2} g(W^{h1} x)))$
- Network with three hidden layers: $\text{softmax}(W g(W^{h3} g(W^{h2} g(W^{h1} x))))$

Non-linearities for g

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

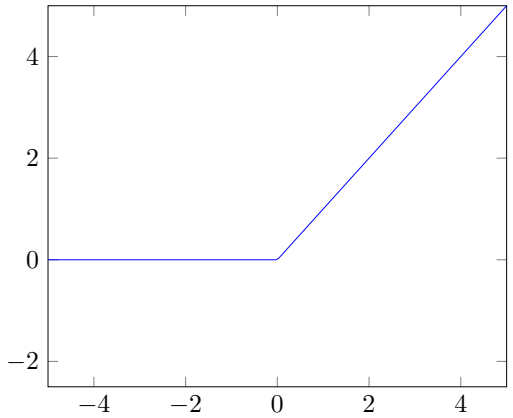


$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

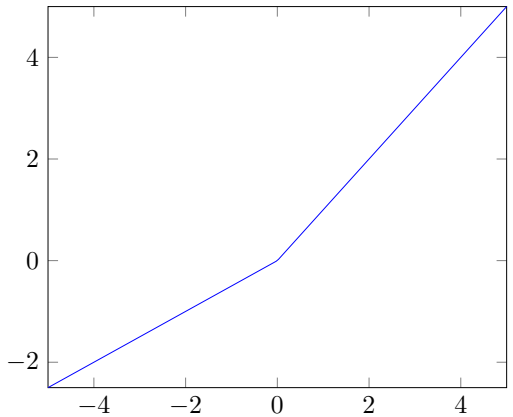


Non-linearities for g

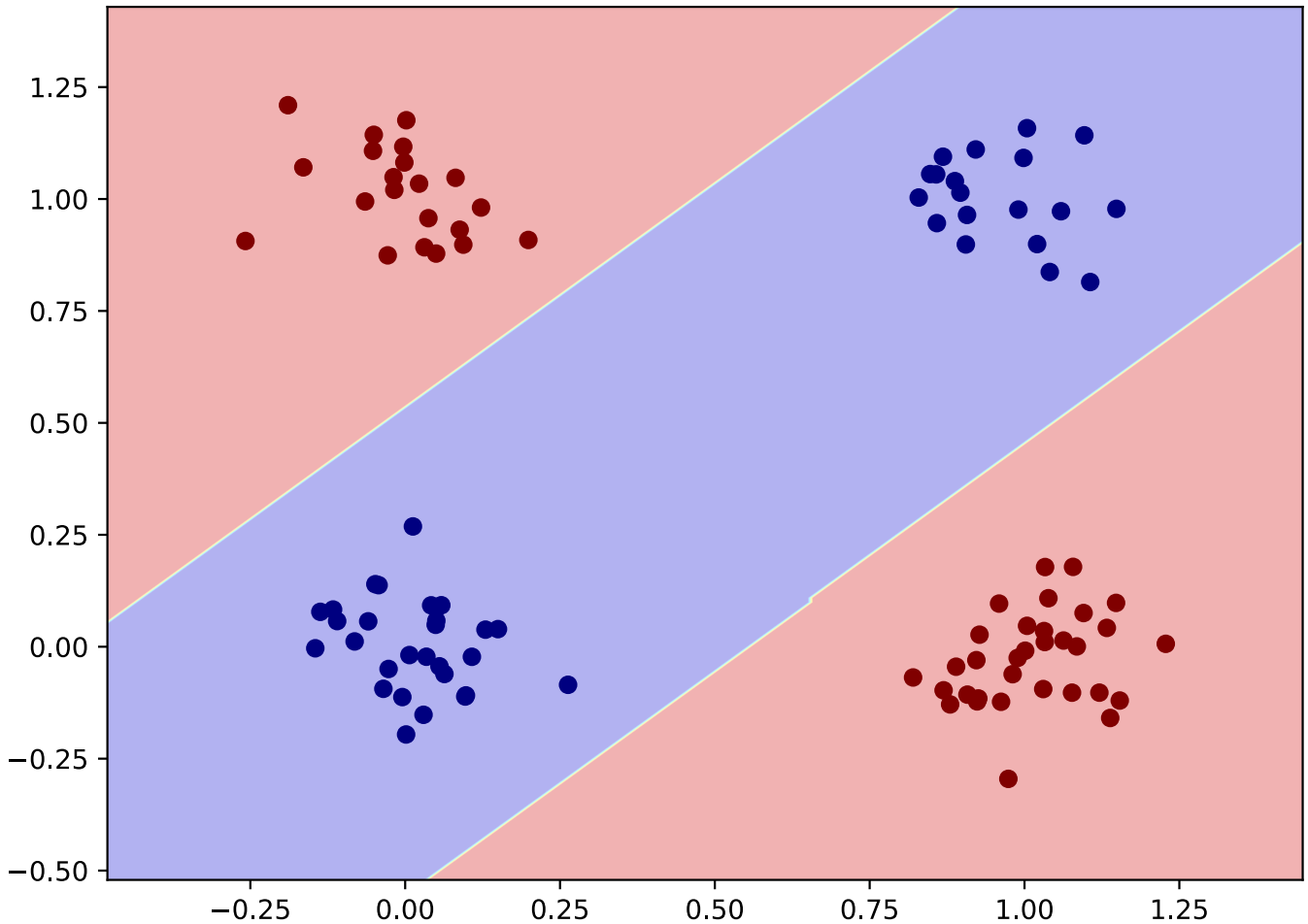
$$\mathrm{ReLU}(x) = \max(0, x)$$



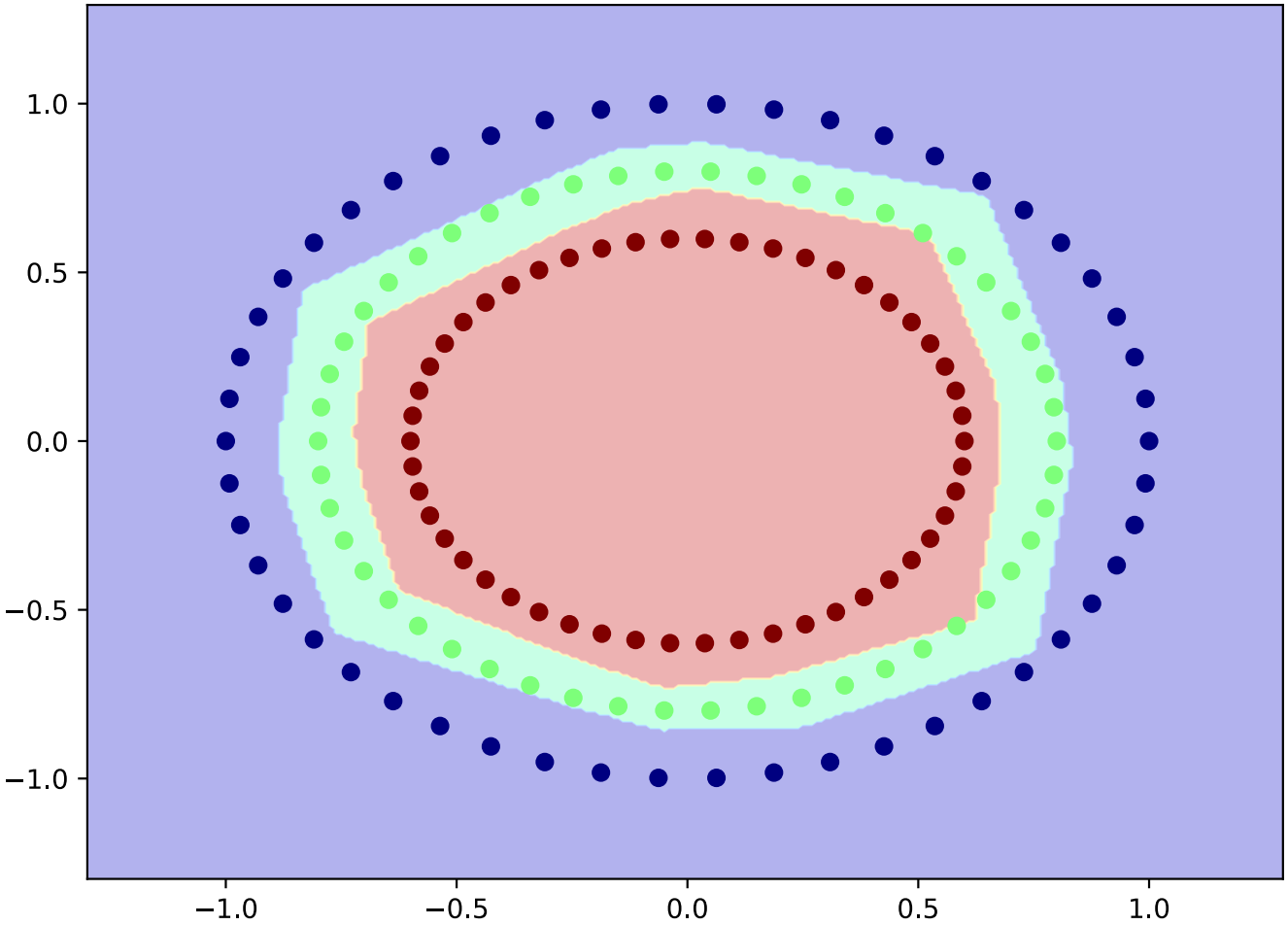
$$\text{Leaky}\,\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \end{cases}$$



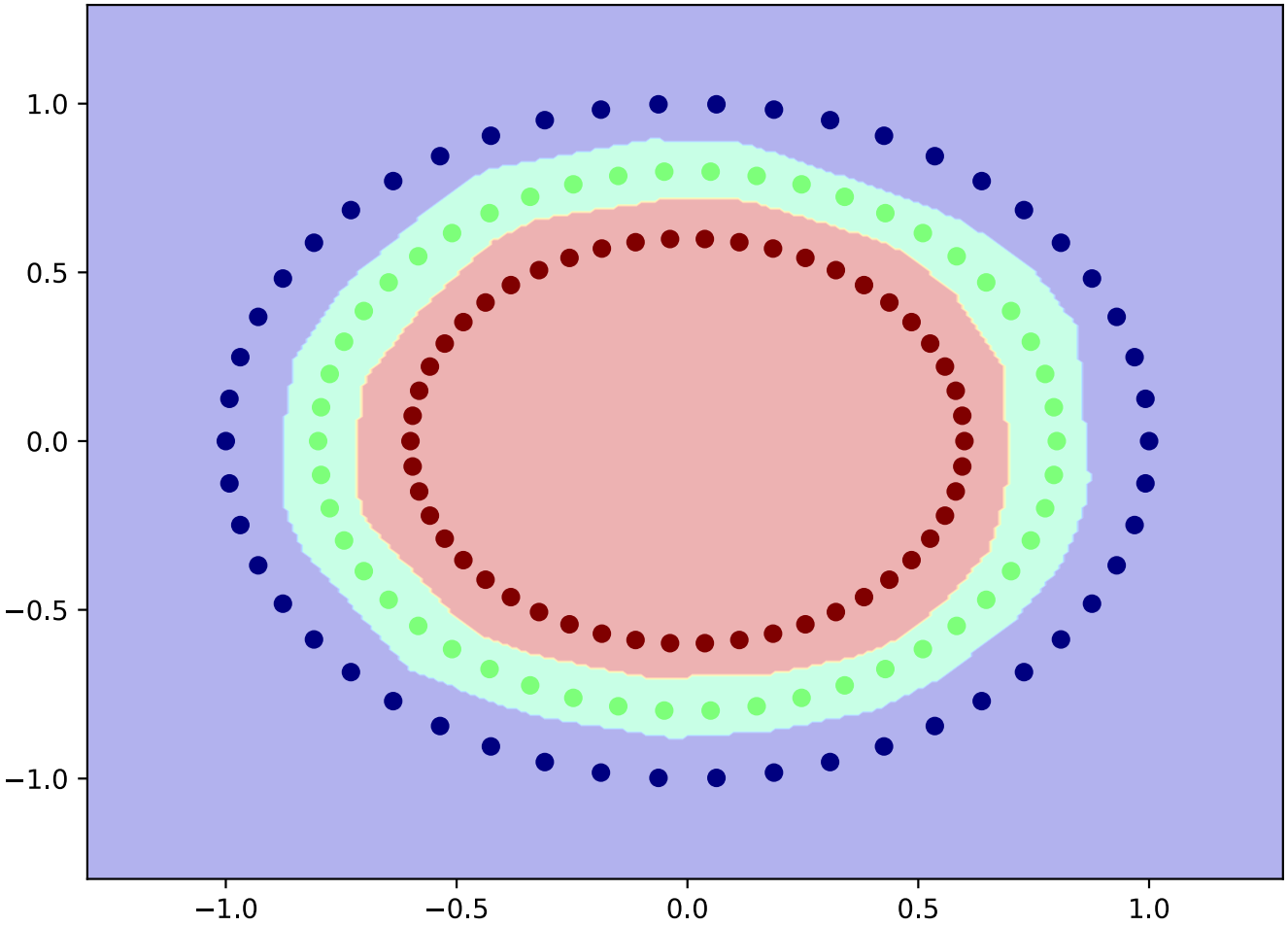
xor feed forward network solution



Three circles feed forward network solution



Three circles feed forward network solution



Non-linear classification and parsing

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

Can a feed forward network tackle the earlier problem?

Non-linear classification and parsing

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

Can a feed forward network tackle the earlier problem? Let's craft a weight matrix!

Non-linear classification and parsing

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

Can a feed forward network tackle the earlier problem? Let's craft a weight matrix!

$$\mathbf{W}_h = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

Non-linear classification and parsing (2)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}\right)$$

Non-linear classification and parsing (2)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}\right)$$

$$= \text{ReLU}\left(\begin{bmatrix} 2 \\ -2 \end{bmatrix}\right)$$

Non-linear classification and parsing (2)

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}\right)$$

$$= \text{ReLU}\left(\begin{bmatrix} 2 \\ -2 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

Non-linear classification and parsing: *wir kaufen*

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}\right)$$

Non-linear classification and parsing: *wir kaufen*

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}\right)$$

$$= \text{ReLU}\left(\begin{bmatrix} -2 \\ 2 \end{bmatrix}\right)$$

Non-linear classification and parsing: *wir kaufen*

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}\right)$$

$$= \text{ReLU}\left(\begin{bmatrix} -2 \\ 2 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

Non-linear classification and parsing: *Autos kaufen*

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}\right)$$

Non-linear classification and parsing: *Autos kaufen*

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}\right)$$

$$= \text{ReLU}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

Non-linear classification and parsing: *Autos kaufen*

1. $\sigma = [\cdots, \text{Autos}, \_]$ (*cars*)
2. $\sigma = [\cdots, \text{hupen}]$ (*honk*)
3. $\sigma = [\cdots, \text{wir}, \_]$ (*wir*)
4. $\sigma = [\cdots, \text{kaufen}]$ (*buy*)

$$\text{ReLU}\left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}\right)$$

$$= \text{ReLU}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Neural dependency parser

Chen and Manning (2014)

Chen and Manning (2014) proposes a parser that consisted of the following ingredients:

- Stack-projective transition system.
- Feed-forward neural network with one hidden layer.
  - Cube activation function ($g(x) = x^3$).
- Words, tag, and label features encoded as word embeddings.

Features

Address	Form	POS-tag	Deprel
σ_0	+	+	
σ_1	+	+	
σ_1	+	+	
β_0	+	+	
β_1	+	+	
β_2	+	+	
LDEP[σ_0, 1]	+	+	+
LDEP[σ_1, 1]	+	+	+
LDEP[σ_0, 2]	+	+	+
LDEP[σ_1, 2]	+	+	+
RDEP[σ_0, 1]	+	+	+
RDEP[σ_1, 1]	+	+	+
RDEP[σ_0, 2]	+	+	+
RDEP[σ_1, 2]	+	+	+
LDEP[LDEP[σ_0]]	+	+	+
LDEP[LDEP[σ_1]]	+	+	+
LDEP[RDEP[σ_0]]	+	+	+
LDEP[RDEP[σ_1]]	+	+	+

Improvement over non-neural parsing

Since we still miss one ingredient, **embeddings**, we will defer the discussion of the improvments over non-neural parsing.

References

Chen, Danqi, and Christopher Manning. 2014. "A Fast and Accurate Dependency Parser Using Neural Networks." In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (Emnlp)*, 740–50.

Kübler, Sandra, Ryan McDonald, and Joakim Nivre. 2009. "Dependency Parsing." *Synthesis Lectures on Human Language Technologies* 1 (1). Morgan & Claypool Publishers: 1–127.

Minski, Marvin L, and Seymour A Papert. 1969. "Perceptrons: An Introduction to Computational Geometry." *MA: MIT Press, Cambridge*.