Lecture 8: Structured Neural Networks

Machine Learning – Theory and Applications

TU Berlin – SoSe 2016

Learning with Sequential Data

Consider two gene sequences:

$$x = (A, T, C, A, G, \underbrace{A, C, A}_{s}, A, T, A)$$
$$x' = (C, \underbrace{G, T, A}_{s'}, C, A, A)$$

Example of a string kernel that compares two strings of variable length:

$$k_{\text{struct}}(x, x') = \sum_{(s,s') \in \mathcal{S}_d(x) \times \mathcal{S}_d(x')} k(s, s')$$

where $S_d(x)$ is the list of all substrings of x of length d.

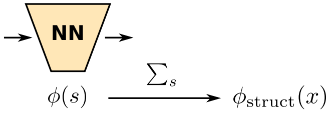
Learning with Sequential Data

- **Observation:** $k_{\mathrm{struct}}(x, x')$ has an associated feature map:

$$\phi_{\mathrm{struct}}(x) = \sum_{s \in \mathcal{S}_d(x)} \phi(s)$$

  where $\phi(s)$ is the feature map associated to $k(s, s')$.
- **Question:** Can a neural network represent this feature map?
- **Answer:** Apply a sliding window

$$x = (A, T, C, A, G, \underbrace{A, C, A}_{s}, A, T, A)$$



The sliding window principle is a basic idea of the convolutional neural network.

Implementing $\phi(s)$ with a neural network

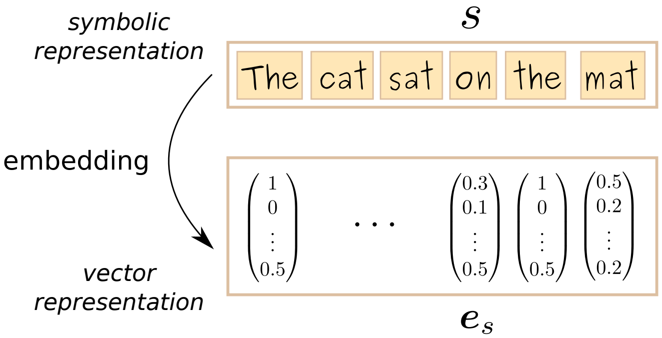
- **Observation:** In the previous genes example, $\phi : \{A, C, T, G\}^d \to \mathbb{R}^h$.
- **Problem:** A standard neural network can only implement functions of type $\phi : \mathbb{R}^d \to \mathbb{R}^h$.
- **Idea:** Learn for each substring s an embedding $\mathbf{e}_s \in \mathbb{R}^d$ and an feature map ϕ such that

  $$\forall_{s,s'} : \ \psi(\mathbf{e}_s)^\top \psi(\mathbf{e}_{s'}) \approx k(s, s').$$

  and use $\mathbf{e}_s$ as a replacement for s.

Word2vec and many other neural network techniques for text processing, are based on this embedding idea.

Data Representation vs. Symbolic vs. Vector

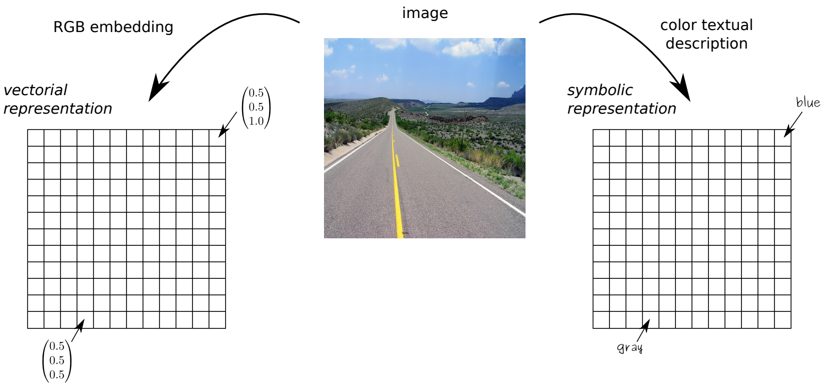


symbolic representation
s
The
cat
sat
on
the
mat
embedding
vector representation
$\begin{pmatrix}1\\0\\\vdots\\0.5\end{pmatrix} \dots \begin{pmatrix}0.3\\0.1\\\vdots\\0.5\end{pmatrix} \begin{pmatrix}1\\0\\\vdots\\0.5\end{pmatrix} \begin{pmatrix}0.5\\0.2\\\vdots\\0.2\end{pmatrix}$
$\boldsymbol{e}_s$

- Before processing by a neural network, a word must first be embedded into some vector space.

```
word2vec = dict(); ...; e = word2vec[s]
```

Data Representation vs. Symbolic vs. Vector



- RGB representation is already the most common one.
- No need to embed the color of pixels in some space.

Implementing $\phi_{\text{struct}}(\mathbf{x})$

- **Idea:** Describe an image $\mathbf{x}$ as a succession of local patches $\mathbf{s}$ and apply the same feature map to all patches.

- **Observation**: When the map is of type

$$\phi(\mathbf{s}) = \begin{pmatrix} \sigma(\mathbf{w}_1^\top \mathbf{s}) \\ \vdots \\ \sigma(\mathbf{w}_h^\top \mathbf{s}) \end{pmatrix}$$

then, $\phi_{\text{struct}}(\mathbf{x})$ can be implemented in terms of convolutions:

$$\phi_{\text{struct}}(\mathbf{x}) = \begin{pmatrix} \text{pool}(\sum_c \mathbf{w}_{1c} \star \mathbf{x}_c) \\ \vdots \\ \text{pool}(\sum_c \mathbf{w}_{hc} \star \mathbf{x}_c) \end{pmatrix}$$

where *pool* is a nonlinear pooling function and c is a color channel.

Implementing $\phi_{\text{struct}}(\mathbf{x})$

- The feature map can be implemented in terms of convolution

$$\phi_{\text{struct}}(\mathbf{x}) = \begin{pmatrix} \text{pool}(\sum_c \mathbf{w}_{1c} \star \mathbf{x}_c) \\ \vdots \\ \text{pool}(\sum_c \mathbf{w}_{hc} \star \mathbf{x}_c) \end{pmatrix}$$

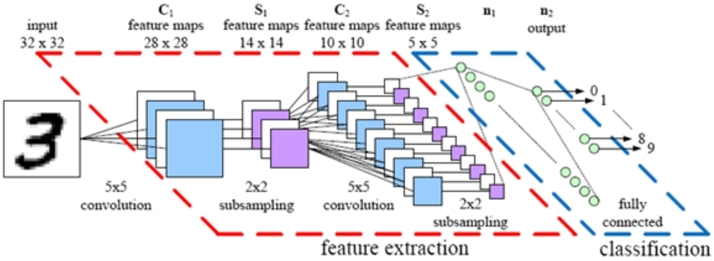
- ... but it can also be implemented can be implemented in terms of matrix products:

$$\phi_{\text{struct}}(\mathbf{x}) = \sum_{\mathbf{s} \in S(\mathbf{x})} \sigma(W\mathbf{s})$$

  where σ applies element-wise and $W = (\mathbf{w}_1, \dots, \mathbf{w}_h)$.

- The last one is better if S is relatively small, and when c, h are large. (But the first type of implementation still gives the name "convolution" to the neural network).

Convolutional Neural Network



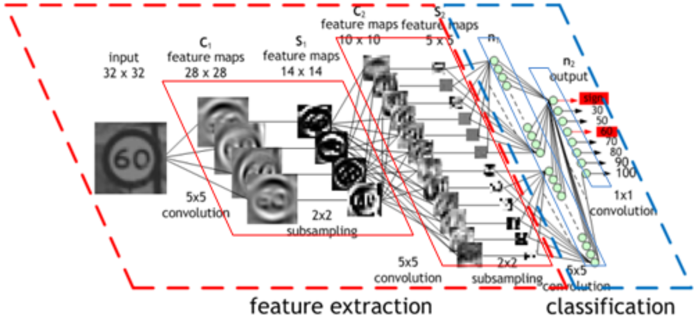
(Image taken from the website of *Parallel Architecture Research Eindhoven*)

A convolutional neural network alternates several stages of

- Convolutions (convolve the image with different filters)
- Pooling (pool features at neighboring locations)

Convolutional Neural Network

How is the data processed by a convolutional neural network?



(Image taken from the website of *Parallel Architecture Research Eindhoven*)

Error Backpropagation in a Convolution

Forward pass:

$$z_t = [w \star x]_t = \sum_{s=-\infty}^{\infty} w_s \cdot x_{t+s}$$

Backward pass:

$$\begin{aligned}
\frac{\partial E}{\partial x_u} &= \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot \frac{\partial z_t}{\partial x_u} \\
&= \sum_{t=-\infty}^{\infty} \sum_{s=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot w_s \cdot 1_{\{t+s=u\}} \\
&= \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot w_{u-t} \\
&= \left[\frac{\partial E}{\partial z} * w\right]_u
\end{aligned}$$

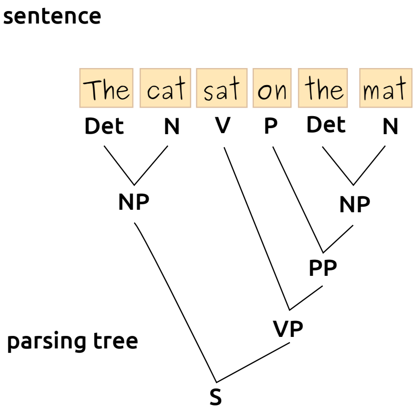
Parameter gradient:

$$\begin{aligned}
\frac{\partial E}{\partial w_u} &= \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot \frac{\partial z_t}{\partial w_u} \\
&= \sum_{t=-\infty}^{\infty} \sum_{s=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot x_{t+s} \cdot 1_{\{s=u\}} \\
&= \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot x_{u+t} \\
&= \left[\frac{\partial E}{\partial z} \star x\right]_u
\end{aligned}$$

Application of CNNs to Image Classification

- Convolutional neural network have outperformed preceding techniques on a large number of image recognition challenges (e.g. ILSVRC 2012) by a wide margin.
- Every year, convolutional neural networks are becoming bigger and deeper ($>$ 20 layers) but also better engineered and implemented.
- Convolutional neural networks have been successfully applied to other indirect tasks (e.g. extracting relevant part of an image, or adapting images to make them resemble to another object, or enhance certain high-level features of it).

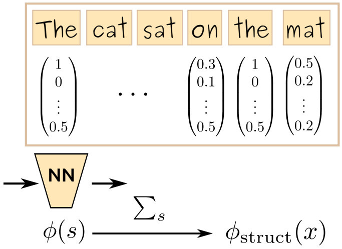
Neural Networks for Parsing Trees



- In some cases, input is better modeled as a tree than as a sequence (e.g. parsing tree for English text).
- **Idea:** Build feature map hierarchically by recursively applying a feature map from the leaves of the tree down to the root.

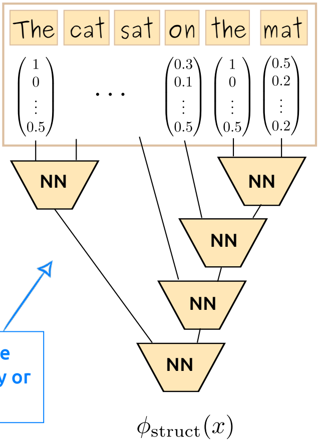
Neural Networks for Parsing Trees

1. Sliding window approach



The
cat
sat
on
the
mat
$\begin{pmatrix}1\\0\\\vdots\\0.5\end{pmatrix}$ … $\begin{pmatrix}0.3\\0.1\\\vdots\\0.5\end{pmatrix}$ $\begin{pmatrix}1\\0\\\vdots\\0.5\end{pmatrix}$ $\begin{pmatrix}0.5\\0.2\\\vdots\\0.2\end{pmatrix}$
NN
$\phi(s)$
$\sum_s$
$\phi_{\text{struct}}(x)$

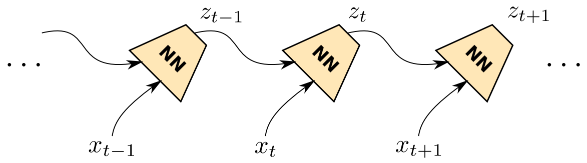
2. Tree-based recursive approach



The
cat
sat
on
the
mat
$\begin{pmatrix}1\\0\\\vdots\\0.5\end{pmatrix}$ … $\begin{pmatrix}0.3\\0.1\\\vdots\\0.5\end{pmatrix}$ $\begin{pmatrix}1\\0\\\vdots\\0.5\end{pmatrix}$ $\begin{pmatrix}0.5\\0.2\\\vdots\\0.2\end{pmatrix}$
NN
NN
NN
NN
NN
$\phi_{\text{struct}}(x)$

Tree structure is provided in advance by a parsing tree (obtained manually or using some software).

Neural Networks for Dynamical Systems (Recurrent Neural Networks)



- Class of neural networks specialized to model *dynamical systems*.
- Neural network implements a feature map of type:

$$\phi\colon \mathbb{R}^h \times \mathbb{R}^d \to \mathbb{R}^h$$

  that maps the input and the system's state at previous time to the state at current time.

Choosing a Transition Function

- Inspired by standard feed-forward networks:

$$z_t = \sigma(W \cdot [z_{t-1}, x_t])$$

  where σ is a nonlinearity applied element-wise.

- Or inspired by nonlinear dynamical system:

$$\dot{z} = -\alpha z + \beta\sigma(W^\top [z, x])$$

  Applying Euler discretization:

$$\frac{z_t - z_{t-1}}{(t) - (t-1)} = -\alpha z_t + \beta\sigma(W \cdot [z_{t-1}, x_t])$$

  Solving for z_t:

$$z_t = (1-\alpha) \cdot z_{t-1} + \beta\sigma(W \cdot [z_{t-1}, x_t])$$

RNNs vs. Markov Models

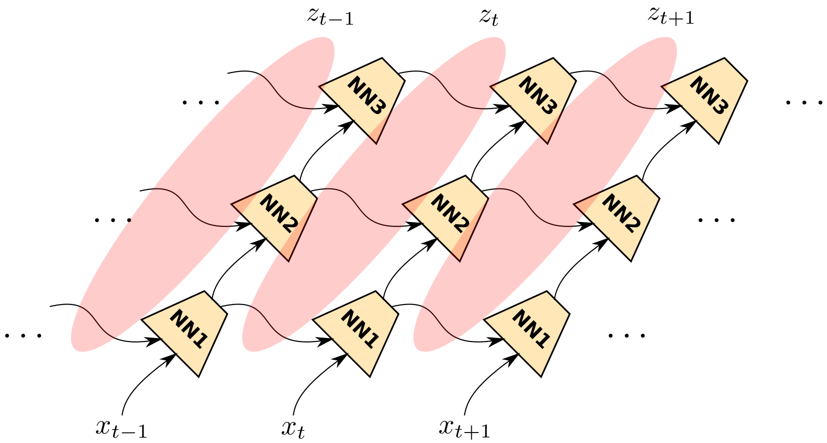
Markov Chains:

- Discrete set of states $z_t \in \{S_1, \dots, S_N\}$.
- Each state has a probability $P(z_t = S_i)$ reached with transition probability $P(z_t = S_j | z_{t-1} = S_i; \theta)$.
- Advantages:
  - Able to model uncertainty and to sample from the model.

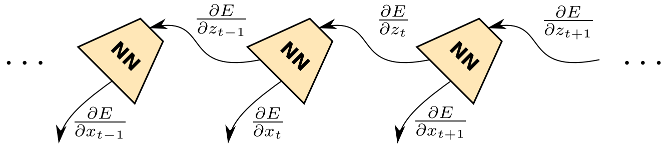
Recurrent Neural Networks:

- Continuous state space $z_t \in \mathbb{R}^h$
- Deterministic state z_t and transition function ϕ
- Advantages:
  - Can model more complex state spaces and choose any transition function ϕ.

Deeper Recurrent Neural Networks



Error backpropagation in an RNN



...
NN
NN
NN
...
$\frac{\partial E}{\partial z_{t-1}}$
$\frac{\partial E}{\partial z_t}$
$\frac{\partial E}{\partial z_{t+1}}$
$\frac{\partial E}{\partial x_{t-1}}$
$\frac{\partial E}{\partial x_t}$
$\frac{\partial E}{\partial x_{t+1}}$

Gradient of the state at time t can be backpropagated in time through the RNN:

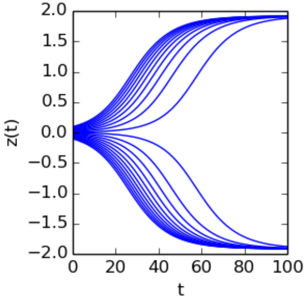
$$\frac{\partial E}{\partial z_1} = \frac{\partial z_2}{\partial z_1} \cdot \frac{\partial z_3}{\partial z_2} \cdot \dots \cdot \frac{\partial z_t}{\partial z_{t-1}} \cdot \frac{\partial E}{\partial z_t}$$

- **Observation:** Because of the multiplicative terms, gradients may have very large or very small magnitude. This is known as the exploding vanishing gradients problem. The problem has its roots in the chaotic nature of the modeled trajectory.

Chaos in RNNs

Study of the one-dimensional map:

$$z_t \mapsto 0.9 z_{t-1} + 0.2 \tanh(z_{t-1})$$



Observation: An infinitesimal variation of the state z_1 may cause a large variation of the error. (Also known as "butterfly effect".)

... that is ...

$$\left[\frac{\partial z_t}{\partial z_1}\right]_{z_1=0} = \lim \frac{\left[z_t\right]_{z_1=\varepsilon} - \left[z_t\right]_{z_1=0}}{\varepsilon}$$

$$= \frac{\text{something very big}}{\text{someting very small}}$$