# Melody Generation using Artificial Neural Networks

AICA Crashkurs

**Benedikt Zönnchen**

6. June 2024

## Outline

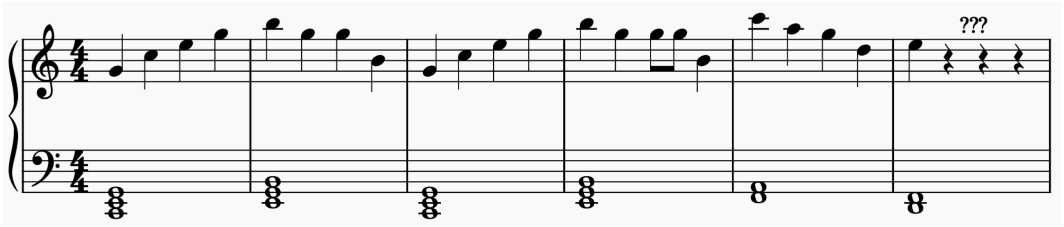
"[The mechanical engine] might act upon other things besides numbers, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine. [. . . ] Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent." – Ada Lovelace (1815 - 1852)

## Introduction

## Introduction: Problem Definition

**Problem**

Given a sequence of notes of length $m \geq 0$, we want to generate a “pleasing” or “fitting” continuation.



## Introduction

The generation of “high quality” melodies and harmonies is a difficult problem:

- The task is multidimensional (a vertical harmony and a horizontal melody).
- The meaning of a note emerges from its relation to other notes.
- These relations can be of high and low frequency.
- There is music that seems inimitable (e. g. Gustav Holst - The Planets - Jupter, the Bringer of Jollity).
- There seems to be a sweat spot between randomness and repetition or surprise and expectation.
- . . .

## Introduction: Representations

The relation between symbolic notation and sound, is similar to the relation between text and the spoken language:

- **Score**: Symbolic, abstract, discrete, carrier of musical ideas
- **Sound**: Continuous changes of pressure (waves), concrete, physical, carrier of all details

Between these two extremes, there are often additional steps from abstract to concrete.

## Introduction: Representations

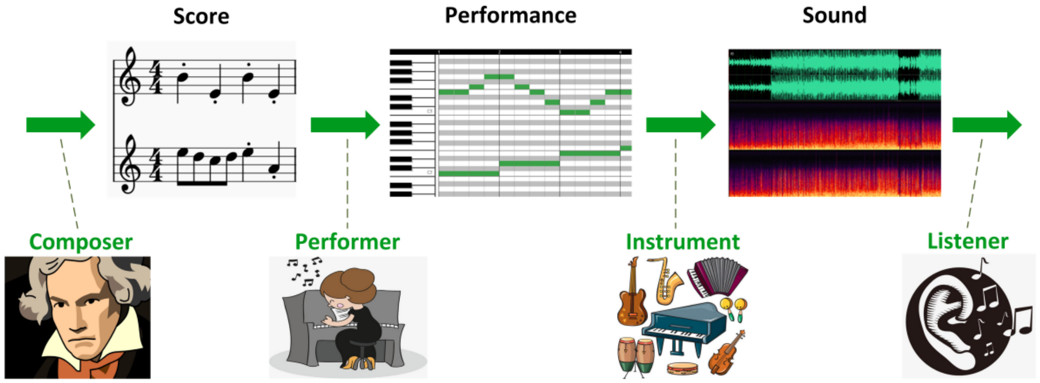


Figure: Source: [Ji et al., 2020]

## Introduction: Extremes of Algorithmic Compositions

According to [Nierhaus, 2009], we can distinguish between

- **genuine compositions** and
- **style imitations**,

but we are always somewhere in between these extremes.

## Introduction: Extremes of Algorithmic Compositions

In addition, we can distinguish between further extremes, namely

- **rule-based** and
- **data-driven**,

as well as

- **automatic** and
- **manual** composing.

## Introduction: Rule-based Techniques

### Rule-based Techniques

- **Markov-Chain (MC)**
- hidden Markov model (HMM)
- cellular automata
- generative grammar
- lindenmayer-systems
- fractals
- genetic algorithms
- ...

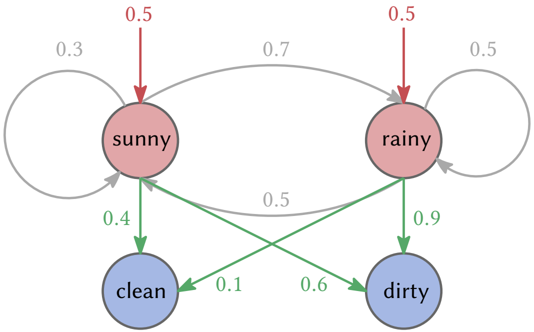
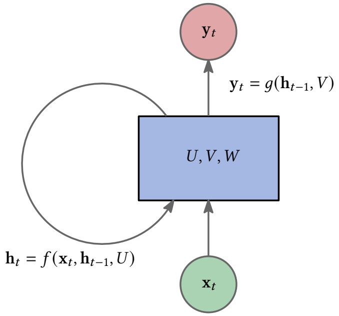


Figure: A simple hidden Markov model.

## Introduction: Data-based Techniques

### Data-based Techniques

- **Markov chains (MC)**
- hidden Markov models (HMM)
- artificial neural networks (ANN):
  - **feedforward neuronal netkworks (FNN)**
  - **recurrent neuronal networks (RNN)**
  - transformer
  - ...
- ...



$\mathbf{y}_t$
$\mathbf{y}_t = g(\mathbf{h}_{t-1}, V)$
$U, V, W$
$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}, U)$
$\mathbf{x}_t$

Figure: Sketch of an RNN.

## Introduction: History

**Rule-based composing**:

- Guido of Arezzo (1000): Generation of music from text.
- Atanasius Kirchner (1650): Automatic generation of contra point compositions via wooden sticks
- Würfelspiele by J. K. Kirnberger (1757 - 1812) and others such as Haydn and Mozart
- L. Hiller and L. Isaacson (1955): *The Iliac Suite*, a fully computer-generated composition based on Markov chains.
- Iannis Xenakis (1922 - 2001) e. g. *Metastasis*, John Cage (1912 - 1992) e. g. *Atlas Eclipticalis*: random-based compositions

## Introduction: History

**Data-driven composing**:

- vanilla recurrent neural network (monophonic), [Todd, 1989]
- LSTMs for melodies and harmonies, [Eck and Schmidhuber, 2002]
- FolkRNN, LSTM (monophonic), [Sturm et al., 2016]
- biaxial LSTM with an interesting architecture (polyphonic), [Johnson, 2017]
- MusicTransformer, composition of long sequences with increased dynamics (polyphonic), [Huang et al., 2018]
- AnticipateRNN, conditioned generation (polyphonic), [Hadjeres and Nielsen, 2018]
- Transformer, conditioned generation (polyphonic), [Hadjeres and Crestel, 2021]
- Anticipatory Music Transformer, event process conditioned by controls process (polyphonic, infilling) [Thickstun et al., 2023]

## Introduction: History

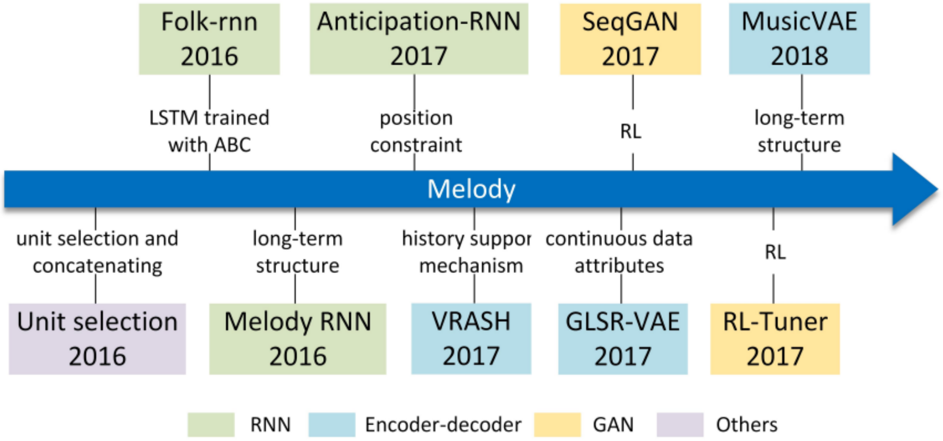


Figure: Source: [Ji et al., 2020]

## Introduction: History

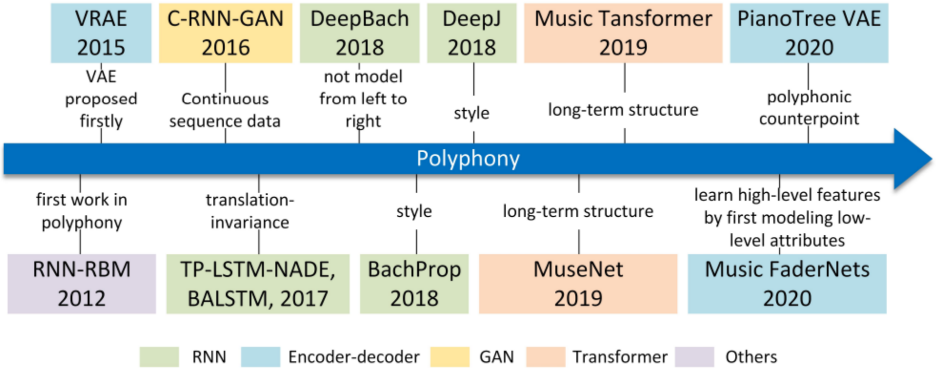


Figure: Source: [Ji et al., 2020]

## Introduction: Applications

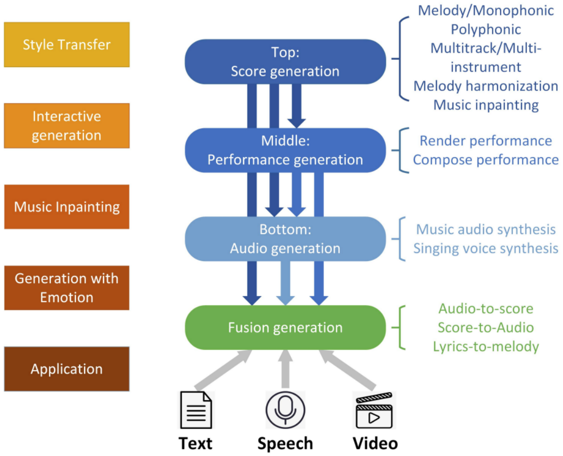


Figure: Source: [Ji et al., 2020]

## Delimitation

We are concerned with *abstract* melody generation (without harmony).

## Representations

# Representations

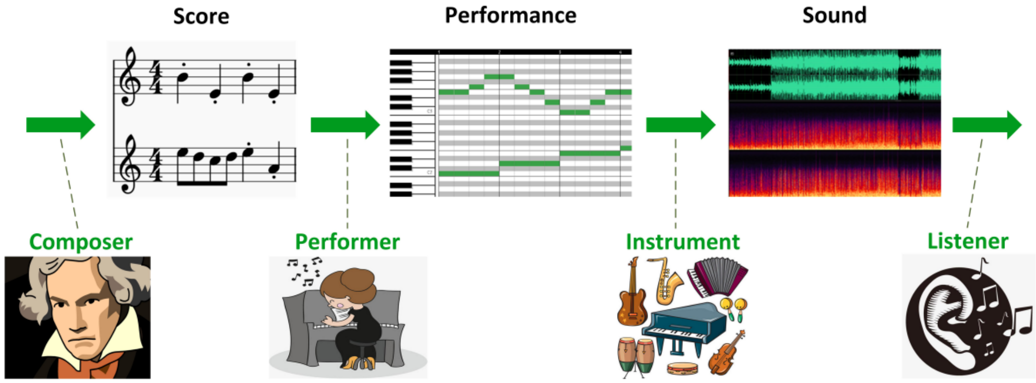


Figure: Source: [Ji et al., 2020]

## Representations

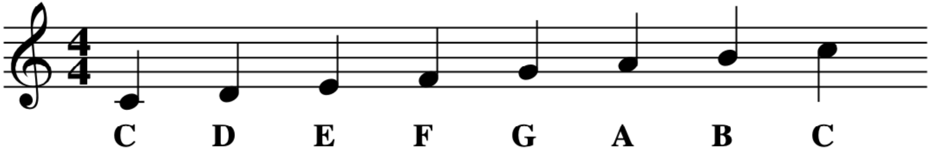
We work with two representations:

### Sequence of Notes/Rests

Each event is a note or rest, that is, a tuple (Pitch/Rest, Duration). Duration is measured in quarter notes. For example, (C3, 4) is a C3 with a duration of 4 quarter notes, that is, a dotted half note. We can also use MIDI values as pitch, that is, (60, 4) instead.

A melody would then look as follows:

$$(60, 3), (55, 1), (53, 2), (\text{Rest}, 2), \ldots$$



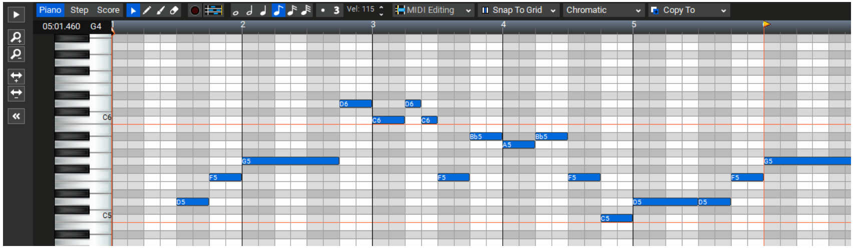
# Representations

## Piano Roll

Each event takes the same amount of time(let's say one quarter). There is a **NoteXOn** or for each note X or Rest. Additionally, there is an event **Hold** that holds the respective note or rest.

A melody would then look as follows:

$$60\ \_\ \_\ 55\ 53\ \_\ \_\ r\ \_\ldots$$



## Representations

The **sequence of notes** requires a larger alphabet. **Piano roll** uses more symbols for the same information.

In the case of polyphony, our alphabet explodes!

### Question: Polyphonic Representation

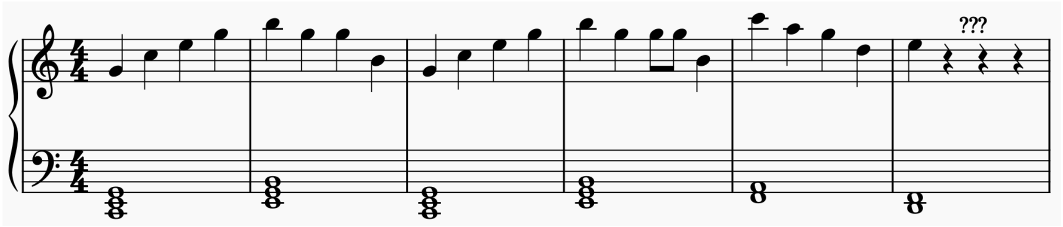
What could our representation look like in the case of polyphony?

## (Data-driven) Markov Chains

## Markov Chains

**Problem**

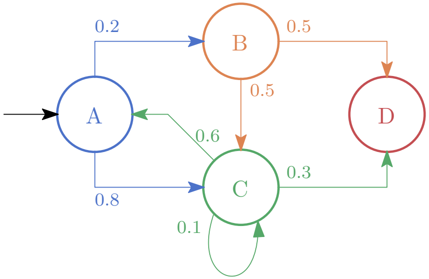
Given a sequence of notes of length $m \geq 0$, we want to generate a “pleasing” or “fitting” continuation.



## Markov Chains

**Idea**

Treat each note as a state of an automaton and calculate the **transition probabilities** based on the given data.



For such a first-order Markov chain, the next state depends solely on the previous one! The process is **memoryless**.

## Markov Chains

Suppose we have $n$ different notes (plus a symbol indicating the start and end of a piece), then there are $(n+1)^2$ transitions (some of which may have a probability of 0).

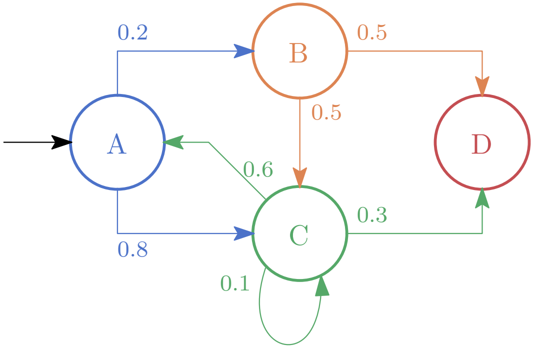
We iterate over all the data and count how often each transition occurs. Then, we divide the result by the total number of transitions.

Thus, we create a table or matrix (called Markov matrix) $\mathbf{P}$ where row $j$, indicates the probability distribution for note $j$.

$$\mathbf{p}_j = (0.0, 0.6, 0.0, 0.1, 0.3)$$

Entry $i$ in row $j$ thus indicates the probability of the transition from note $j$ to $i$.

# Markov Chains



|   | . | A | B | C | D |
|---|---|---|---|---|---|
| . | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| A | 0.0 | 0.0 | 0.2 | 0.8 | 0.0 |
| B | 0.0 | 0.0 | 0.0 | 0.5 | 0.5 |
| C | 0.0 | 0.6 | 0.0 | 0.1 | 0.3 |
| D | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

$$\mathbf{P} = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.2 & 0.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.5 \\ 0.0 & 0.6 & 0.0 & 0.1 & 0.3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

## Markov Chains: Learned MC

Suppose we have the following **sequence of notes**:

. (64, 4), (64, 2), (50, 3), (50, 3), (64, 4), (64, 2) .

therefore, we have three different notes and one special symbol

**(50, 3)**, **(64, 2)**, **(64, 4)**, **.**

We can count the appearance of each transition:

| | . | (50, 3) | (62, 2) | (64, 4) |
|---|---|---|---|---|
| . | 0 | 0 | 1 | 0 |
| (50, 3) | 0 | 1 | 0 | 1 |
| (62, 2) | 1 | 1 | 0 | 0 |
| (64, 4) | 0 | 0 | 2 | 0 |

## Markov Chains: Assignment

We can count the appearance of each transition:

| | . | (50, 3) | (62, 2) | (64, 4) |
|---|---|---|---|---|
| . | 0 | 0 | 1 | 0 |
| (50, 3) | 0 | 1 | 0 | 1 |
| (62, 2) | 1 | 1 | 0 | 0 |
| (64, 4) | 0 | 0 | 2 | 0 |

To estimate the probabilities we can normalize each row which gives us

$$\mathbf{P} = \begin{pmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.5 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

We can call this **machine learning**!

## Markov Chains: Assignment

**Question: Generating based on a Markov matrix**

Given $\mathbf{P}$, how can we generate a new melody?

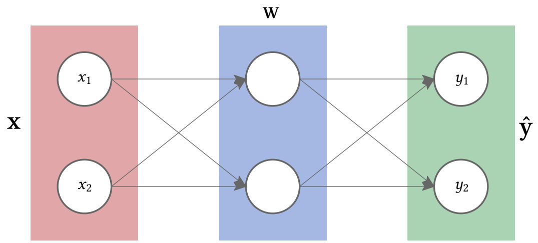
$$\mathbf{P} = \begin{pmatrix} 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.5 \\ 0.5 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{pmatrix}$$

## Feedforward Neural Networks

## Feedforward Neural Networks

**Idea**

Train a simple neural network that receives a sequence with *k* notes as input and predicts the next note.



In the example notebook, we use $k = 1$ and even omit activation functions. Thus, our network is a simple $n \times n$ matrix $\mathbf{W}$.

## Feedforward Neural Networks

Each of the $n$ notes is encoded with a so-called *one-hot vector* with $n$ components, where exactly one of them is equal to 1, for example:

$$\mathbf{x}^\top = (0, 0, 1, \ldots, 0, 0, 0)$$

and the next note $\hat{\mathbf{y}}$ is determined by sampling from the discrete probability distribution

$$\operatorname{softmax}(\mathbf{x}^\top \mathbf{W}) = \operatorname{softmax}(\mathbf{y}^\top) = \mathbf{p}^\top.$$

## Feedforward Neural Networks: Assignment

We train our feedforward neural network (FNN) with sequences of length 1. That is, from a musical piece in **sequence of notes** representation:

., (64, 4), (64, 2), (50, 3), (50, 3), (64, 4), (64, 2), .

we can generate a training data set:

| Input **x** | Output **y** |
|---|---|
| . | (64, 4) |
| (64, 4) | (64, 2) |
| (64, 2) | 50/3 |
| (50, 3) | (50, 3) |
| (50, 3) | (64, 4) |
| (64, 4) | (64, 2) |
| (64, 2) | . |

## Recurrent Neural Networks

## Recurrent Neural Networks

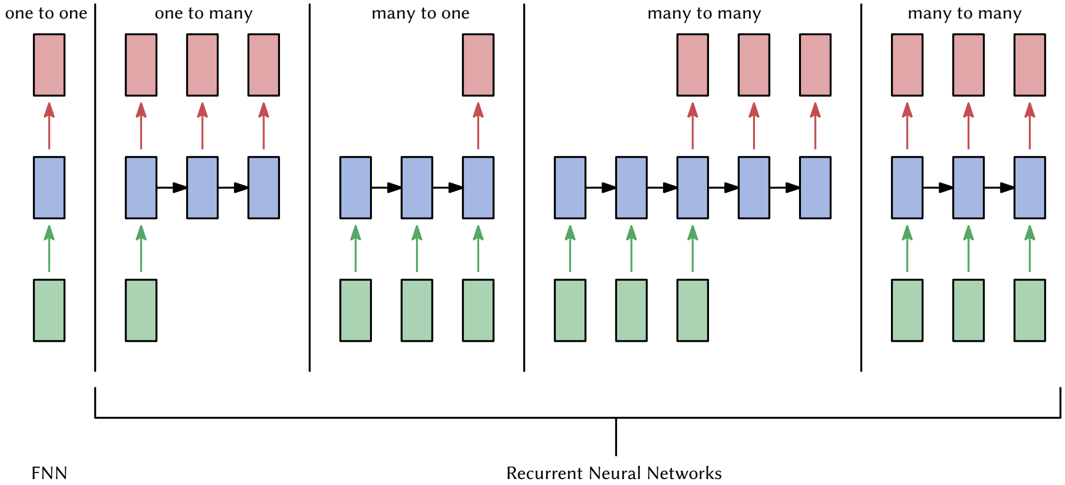
### Problem

In both approaches, the next note depends solely on the current note! These methods are **memoryless**.

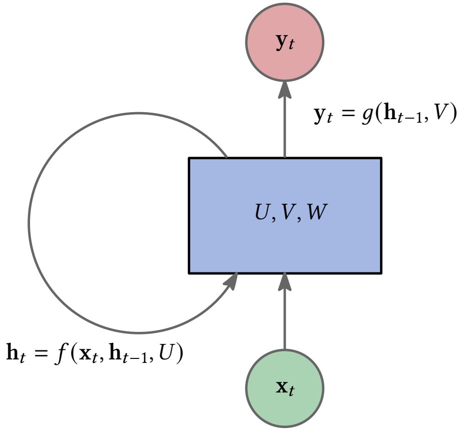
### Possible Improvements

- higher order Markov chains
- transformers
- **recurrent neural networks** (neural networks with **feedback loops**!)
- . . .

# Recurrent Neural Networks

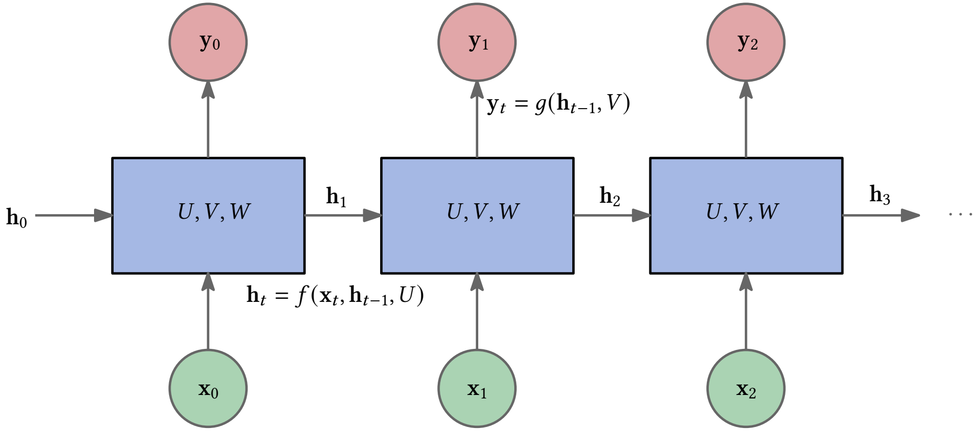


## Recurrent Neural Networks



$\mathbf{y}_t$
$\mathbf{y}_t = g(\mathbf{h}_{t-1}, V)$
$U, V, W$
$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}, U)$
$\mathbf{x}_t$

## Recurrent Neural Networks

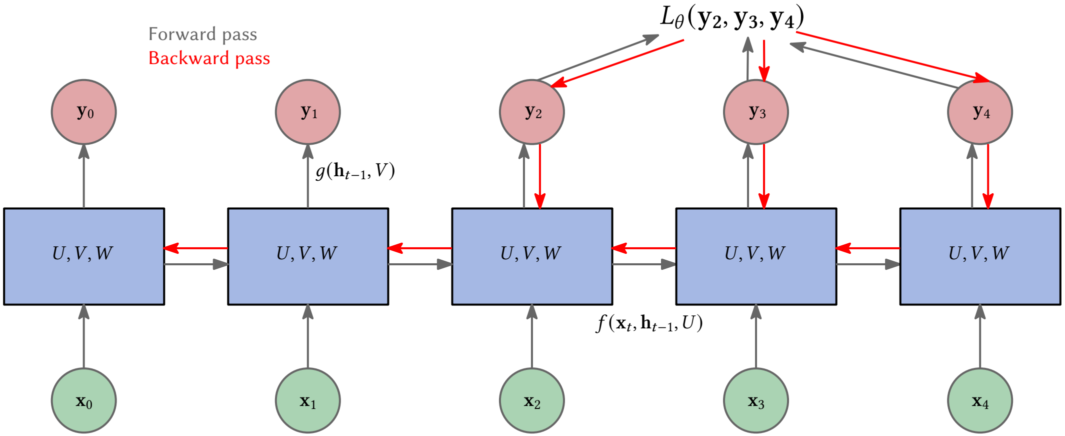


## Recurrent Neural Networks

**Theorem**

*For every* ***RNN****, there exists a* ***feedforward network*** *with the same behavior over a finite time [Rumelhart and McClelland, 1987].*

## Recurrent Neural Networks: Back-Propagation Through Time



## Recurrent Neural Networks: Back-Propagation Through Time

“If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs.” – Andrej Karpathy

## Recurrent Neural Networks: Back-Propagation Through Time

There is an exponential dependence between the error and the weights $W$

$$\frac{\partial \mathbf{h}_t}{\partial W} = \frac{\partial f(\mathbf{x}_t, \mathbf{h}_{t-1}, W)}{\partial W} + \frac{\partial f(\mathbf{x}_t, \mathbf{h}_{t-1}, W)}{\partial \mathbf{h}_{t-1}} \cdot \frac{\partial \mathbf{h}_{t-1}}{\partial W}$$

$\Rightarrow$ **Gradients tend to explode or vanish.**

$$\lim_{N \to \infty} (x \cdot w_{ij}^N) = \begin{cases} \infty & \text{für } w_{ij} > 1.0 \\ 0 & \text{für } w_{ij} < 1.0 \end{cases}$$

## Recurrent Neural Networks

Vanilla RNNs (in their original form) are rarely used anymore. Instead, **Long Short-term Memory RNNs**, or **LSTMs**, are utilized.

## Recurrent Neural Networks

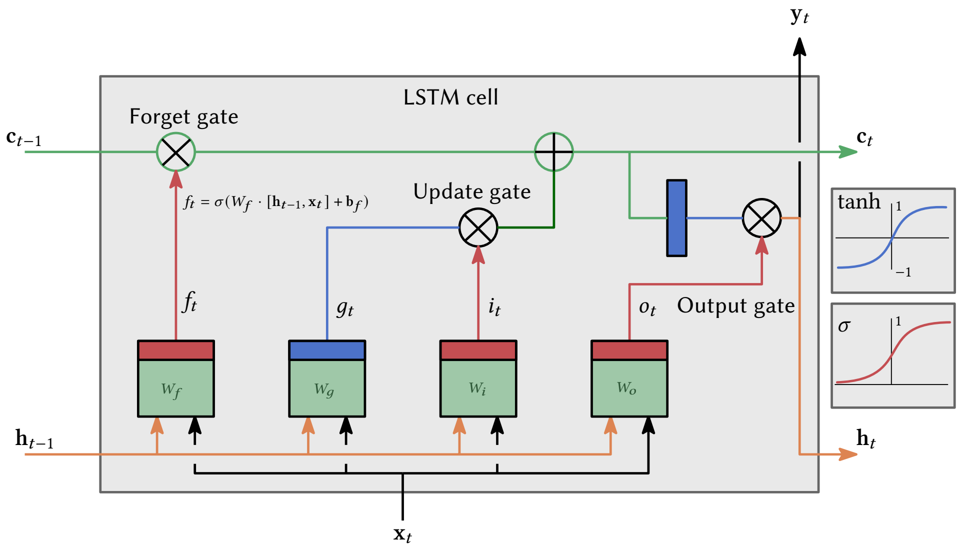
### Advantages

+ variable input length
+ computation time increases linear with output length
+ computation simulates a sort of memory
+ weights are shared over time

### Disadvantages

- sequential computation which can be slow
- information gets washed away (there is no learnable prioritizing)
- gradient tend to explode or vanish $\Rightarrow$ often impossible to train for long sequences

## Recurrent Neural Networks: Long Short-term Memory



## Recurrent Neural Networks: Long Short-term Memory

### Advantages (to vanilla RNNs)

+ trainable for longer sequences
+ a more stable memory
+ additive instead of multiplicative error during backpropagation

### Disadvantages (to transformers)

- sequential processing of sequences during training
- still unable to train for very long sequences
- transformer offer a more flexible way to learn relationships between notes / events

The sequential structure, i.e., the **information flow through time**, remains intact!

$\Rightarrow$ The information continues to be blurred over time.

# Recurrent Neural Networks: Assignment

Assume we train our RNN with sequences of length 4. That is, from a musical piece in **piano roll representation**

64 _ _ _ 55 53 _ 63 _ _ _ r _ _ _ 55 _ _ _ 56 _ _ _

we generate a training data set:

| **x** | | | | **y** |
|---|---|---|---|---|
| . | . | . | . | 64 |
| . | . | . | 65 | _ |
| . | . | 65 | _ | _ |
| . | 65 | _ | _ | _ |
| 65 | _ | _ | _ | 55 |
| _ | _ | _ | 55 | 53 |
| | | | . . . | |

| **x** | | | | **y** |
|---|---|---|---|---|
| | | | . . . | |
| _ | 56 | _ | _ | _ |
| 56 | _ | _ | _ | . |
| _ | _ | _ | . | . |
| _ | _ | . | . | . |
| _ | . | . | . | . |

## Recurrent Neural Networks: Assignment

### Question: Provided Information

In our representation, we capture only the information of pitch and duration with

64 _ _ _ 55 53 _ 63 _ _ _ r _ _ _ 55 _ _ _ 56 _ _ _

What information would probably also helpful for our training?

## Literature I

Eck, D. and Schmidhuber, J. (2002).
Finding temporal structure in music: blues improvisation with LSTM recurrent networks.
In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756.

Hadjeres, G. and Crestel, L. (2021).
The piano inpainting application.
*CoRR*, abs/2107.05944.

Hadjeres, G. and Nielsen, F. (2018).
Anticipation-RNN: enforcing unary constraints in sequence generation, with application to interactive music generation.
*Neural Computing and Applications.*

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Hawthorne, C., Dai, A. M., Hoffman, M. D., and Eck, D. (2018).
Music transformer: Generating music with long-term structure.
*arXiv preprint arXiv:1809.04281.*

Ji, S., Luo, J., and Yang, X. (2020).
A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions.
*CoRR*, abs/2011.06801.

# Literature II

Johnson, D. D. (2017).
Generating polyphonic music using tied parallel networks.
In *EvoMUSART*.

Nierhaus, G. (2009).
*Algorithmic Composition - Paradigms of Automated Music Generation*.
SpringerWienNewYork.

Rumelhart, D. E. and McClelland, J. L. (1987).
*Learning Internal Representations by Error Propagation*, pages 318–362.

Sturm, B. L., Santos, J. F., Ben-Tal, O., and Korshunova, I. (2016).
Music transcription modelling and composition using deep learning.
*CoRR*, abs/1604.08723.

Thickstun, J., Hall, D., Donahue, C., and Liang, P. (2023).
Anticipatory music transformer.

Todd, P. M. (1989).
A connectionist approach to algorithmic composition.
*Computer Music Journal*, 13:27–43.