Chapter 2

Background

fliang: Similarity to language modeling, char-rnn vs word-rnn similar so note-rnn vs chord-rnn should be as well. Benefit is that chord vocabulary MUCH larger than natural language

fliang: Try mimicking Franklin [9]

.

fliang: Motivation: Long-term dependencies are at the heart of what defines a style of music, with events spanning several notes or bars contributing to the formation of metrical and phrasal structure Cooper and Meyer [4].

fliang: Describe automatic composition vs harmonization vs analysis/scoring

Generative models are preferrable because they can be:

1. Conditioned for harmonization
2. Sampled for automatic composition
3. Queried for analysis/scoring

Automatic composition is described by Papadopoulos and Wiggins [19] as:

> a sequence (set) of rules (instructions, operations) for solving (accomplishing) a [particular] problem (task) [in a finite number of steps] of combining musical parts (things, elements) into a whole (composition)

Pearce et al. [21] emphasizes *algorithmic composition* should “be applied exclusively to activities in which the motivations have this artistic character.”

fliang: Describe piano roll vs sheet music

Piano roll music transcriptions (**??**) are quantized both in time ($t \in T$) and note frequencies ($n \in N$). frequencies quantized to a piano roll.

fliang: Motivate quantization with Western music

.

We can represent a piano roll transcription as a high-dimensional vecctor $X_{t,n} \in \mathbb{R}^{|T|\times|N|}$ where $X_{t,n}$ denotes the note velocities for note n at time t.

fliang: Describe MIDIs 127 iquantized pitches, isomorphic to musicXML

fliang: Describe Bach and chorales, 4 parts, soprano lead with ATB harmony, regular phrases, fermatas to denote phrase ends. Background section in chorale harmonization.

fliang: Define meter

Eck and Lapalme [7] first addressed. *Meter* is the sense of a periodic pattern of strong and weak beats which arise from periodic articulation of notes in common locations. It is implied in Western music, where bars establish perodic measures of equal length [13]. Meter provides us with key information about musical structure which can be used to predict chord changes and repepetition boundaries [4].

This chapter provides background information on two topics heavily utilized throughout this dissertation: Western music theory and recurrent neural networks. It introduce the definitions and notation used in later sections.

2.1 A primer on Western music theory

Music theory is a branch of musicology concerned with the study of the rules and practices of music. While the general field includes study of acoustic qualities such as dynamics and timbre, we restrict the scope of our research to modeling musical *scores* (e.g. fig. 2.1) and neglect issues related to articulation and performance (e.g. dynamics, accents, changes in tempo) as well as synthesis/generation of the physical acoustic waveforms.

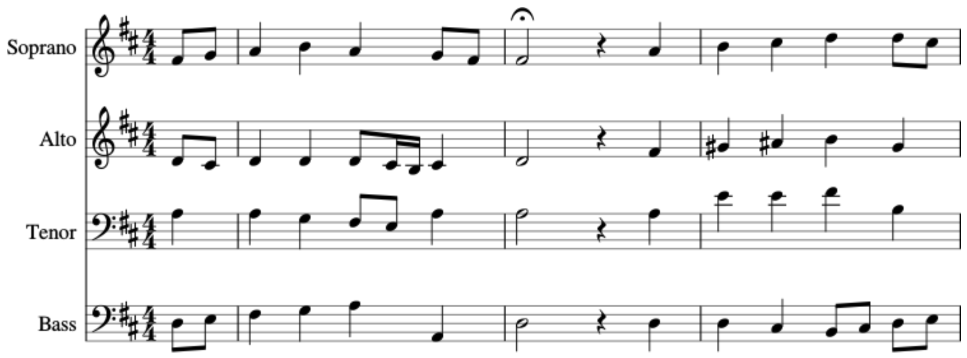

Fig. 2.1 Sheet music representation of some bars from a musical score (BWV133.6) with articulation instructions removed.

This is justified because the phsyical waveforms are more closely related to the skill of the performers and instruments used and are likely to vary significantly across different performances. Furthermore, articulations in the same musical piece may differ across transcriptions and performers. Despite these variations, a piece of music is still recognizable from just the notes, suggesting that notes are the defining element for a piece of music.

2.1.1 Notes: the basic building blocks

A *note* is the most fundamental element of music composition and represents a sound played at a certain *pitch* for a certain *duration*. In sheet music such as fig. 2.1, the are denoted by the filled/unfilled black heads which may also possess protruding stems. As a *score* can be viewed as a collection of notes over time, notes are the fundamental building blocks for musical compositions.

Pitch

Though pitch is closely related to physical frequency of vibration of a waveform (as measured in Hertz), pitch its a perceptual property whose semantic meaning is derived from a listener's perception. This distinction has been scrutinized by Terhardt [27], whose visual analogy in fig. 2.2 illustrates how a pitch can be heard even if its percieved frequency is absent just as one may see the word "PITCH" despite being presented with only a suggestive shadow.

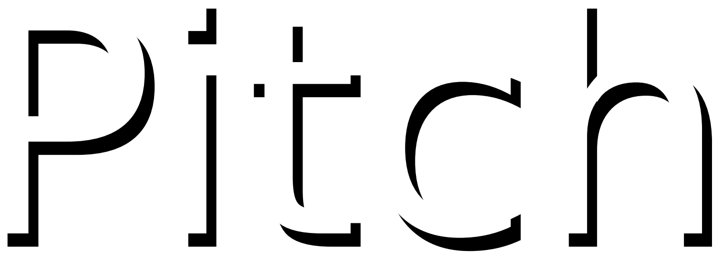

Fig. 2.2 Terhardt's visual analogy for pitch. Similar to how the viewer of this figure may percieve contours not present, pitch describes subjective information received by the listener even when physical frequencies are absent.

Despite its psychoacoustic nature, it is nevertheless useful to objectively quantify pitch as a frequency. To do so, we first need some definitions. The difference between two frequencies is called an *interval* and an *octave* is an interval corresponding to the distance between a frequency $f \in \mathbb{R}^+$ and its doubling $2f$ or halving $f/2$. Two frequencies spaced exactly an octave apart are perceived to be similar, suggesting that music is percieved on a logarithmic scale.

Most Western music is based on the *twelve-note chromatic scale*, which divides an *octave* into twelve distinct frequencies. The *tuning system* employed dictates the precise intervals between subdivisions, with *equal temperament tuning* (all subdivisions are equally spaced on a logarithmic scale) presently the most widely used method [6].

fliang: Talk about well-tempered tuning and bach

Under twelve-note equal temperament tuning, the distance between two consecutive subdivisions (1/12 of an octave) is called a *semitone* (British) or *half-step* (North American) and two semitones constitutes a *tone* or *whole-step*.

When discussing music, *note names* which enable succinct specification of a musical pitch are often employed. In *scientific pitch notation*, *pitch classes* which represent a pitch modulo the octave are specified by a letter ranging from *A* to *G* and optionally a single *accidental*. Pitch classes without accidentals are called *natural* and correspond to the white keys on a piano. Two accidentals are possible: sharps (#) raise the natural pitch class up one semitone and flats (♭) lower by one semitone. fig. 2.3 illustrates how these pitch classes map to keys on a piano.

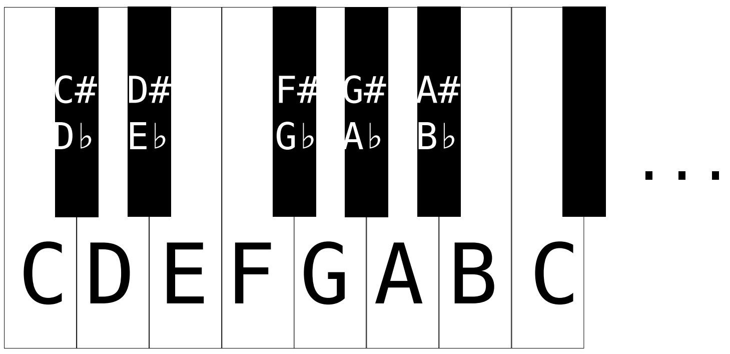

Fig. 2.3 Illustration of an octave in the 12-note chromatic scale on a piano keyboard.

Since pitch classes represent equivalence class of frequencies spaced an integral number of octaves apart, unambiguously specifying a pitch requires not only a pitch class but also an octave. In scientific pitch notation, this is accomplished by appending an octave number to a pitch class letter (see fig. 2.4). Together, a pitch class and octave number uniquely specify the notation for a pitch. On sheet music, the pitch of a note is indicated by its vertical position with respect to the *stave* (the five horizontal lines and four spaces).

fliang: Cite wiki scientific pitch notation

Duration

In addition to pitch, a note also possesses a *duration*. The duration of a note indicates how long it is to be played and is measured in fractions of a *whole note* (American) or *semibreve* (British).

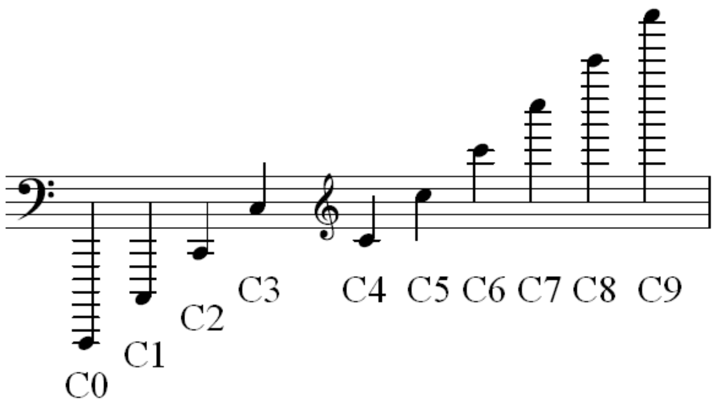

Fig. 2.4 Scientific pitch notation and sheet music notation of *C* notes at ten different octaves.

Perhaps the most common duration is a *quarter-note* (American) or *crotchet* (British). Other note durations are also possible and the most common along with their notation in sheet music are enumerated in fig. 2.5. The relationship between durations and physical time intervals is given by the *tempo*, which is usually denoted near the start of the piece in beats per minute.

fliang: Cite wiki Whole note

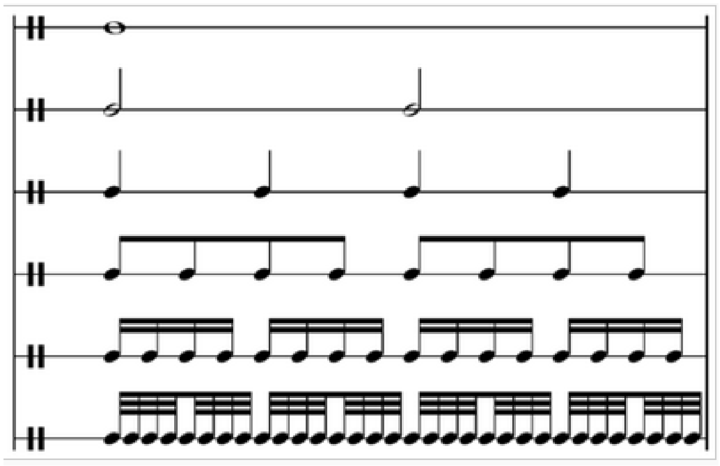

Fig. 2.5 Comparison of various note durations.

2.1.2 Key signature

Tonal music is a genre of music encompassing most of Western classical music. It is characterized by the prevalence of one pitch class (the *tonic*) around which the remainder of the piece is built.

A basic concept within tonal music is *mode*, which defines a subset of pitch classes that are "in key" with respect to the tonic. table 2.1 illustrates the pitch intervals between adjacent pitch classes within two important modes: the *major* (Ionian, I) and *minor* (Aeolian, VI) modes. The choice of tonic and mode is collectively referred to as the *key signature*.

Mode	Pitch Intervals (semitones)
Major (Ionian, I)	+2, +2, +1, +2, +2, +2
Minor (Aeolian, VI)	+2, +1, +2, +2, +1, +2

Table 2.1 Pitch intervals for the two most important modes[10]. The pitches in a scale can be found by starting at the tonic and successively offsetting by the given pitch intervals.

2.1.3 Polyphony, chords, and chord progressions

Whereas *monophonic* music is characterized by the presence of a single *part* sounding at most one note at any given time, *polyphonic* music contains multiple parts potentially sounding multiple pitches at the same time. Just as notes form the basis of monophonic music, chords are the fundamental building blocks for polyphonic music.

Chords: basic units for representing simultaneously sounding notes

A *chord* is a collection of three or more pitches all sounding simultaneously[23]. In Western classical music, they typically consist of a *root note* whose pitch class forms a base from which successive notes are built upon. The intervals between the pitch classes in a chord are commonly labeled using *qualities*, which are invariant across octaves. Different realizations of the same chord (e.g. octave choices for each pitch class) are called *voicings*.

table 2.2 lists some common chord qualities and their corresponding intervals from the root note. Chord names are given as a root pitch class followed by a quality, for example: C major, A minor, or G half-diminished 7th.

The lowest note in a chord is called the *bass* note and is oftentimes the root note. However, alternative voicings called *inversions* can place the root note on a different octave and cause the bass and root notes to differ.

Chord progressions and phrases

Sequences of chords are called *chord progressions*, which are oftentimes grouped with adjacent progressions within a score into coherent units called *phrases*. Many psychoacoustic phenomena such as stability, mood, and expectation can be attributed to phrase structure and

Chord quality	Intervals from root pitch class
Major	+4, +7
Major 6th	+4, +7, +8
Major 7th	+4, +7, +11
Minor	+3, +7
Minor 6th	+3, +7, +9
Minor 7th	+3, +7, +10
Dominant 7th	+4, +7, +10
Augmented	+4, +8
Diminished	+3, +6
Diminished 7th	+3, +6, +9
Half-diminished 7th	+3, +6, +10

Table 2.2 Common chord qualities and their corresponding intervals[10]

choice of chord progressions. For example, chord progressions called *harmonic cadences* are commonly used to conclude a phrase and create a sense of resolution[23]. Another important example are *modulations*, where a chord progression is used to transition the music into a different key signature.

As chords can be overlapping and contain notes straddling between two chords or involve uncommon chord qualities, identifying chord progresssions involves a degree of subjectivity. A common method for analyzing chord progressions is *roman numeral analysis*, where I denotes the tonic pitch class,successive roman numerals correspond to successive pitch classes in the key, and capitalization is used to denote major/minor. For example, the chord progression C major – A minor – D major 7th – G major in the C major key signature would be represented as $I - ii - IImaj7 - V$.

Transposition invariance

A common theme throughout our discussion has been ambiguity of the tonic. When discussing key signatures, the mode was defined using intervals relative to a choice of tonic. Similarly, roman numeral analysis of chord progressions is also conducted relative to a tonic. Even if a chord progression and tonic are both transposed by arbitrary pitch interval, the roman numeral analysis will remain unchanged.

The *transposition invariance* of chord progressions and modes is an important property of music. It enables us to offset an entire score by an arbitrary pitch interval without affecting the important psychoacoustic qualities captured by chord progressions and choice of mode.

2.2 Neural sequence probability modeling

Our work in later sections make heavy use of neural networks. In this section, we briefly review the relevant concepts and set up notation.

2.2.1 Neurons: the basic computation unit

Neurons are the basic abstraction which are combined together to form neural networks. A *neuron* is a parametric model of a function $f : \mathbb{R}^D \to \mathbb{R}$ from its D-dimensional input $\boldsymbol{x}$ to its output y. Our neurons will be defined as

$$f(\boldsymbol{x}) \coloneqq \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle) \tag{2.1}$$

which can be viewed as an inner product with *weights* $\boldsymbol{w}$ to produce an *activation* $z \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle \in \mathbb{R}$ which is then squashed to a bounded domain by a non-linear **activation function** $\sigma : \mathbb{R} \to [L, U]$. This is visually depicted in fig. 2.6, which also makes apparent the interpretation of weight w_i as the sensitivity of the output y to the input x_i.

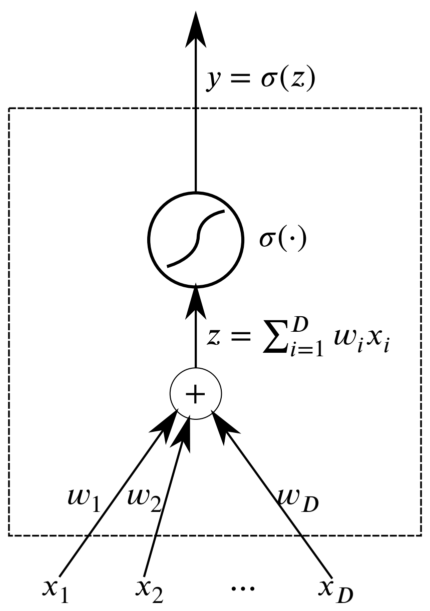

Fig. 2.6 A single neuron first computes an activation z and then passes it through an activation function $\sigma(\cdot)$

2.2.2 Feedforward neural networks

Multiple neurons may share inputs and have their outputs concatenated together to form a *layer* modelling a multivariate functions $f : \mathbb{R}^{D_{\text{in}}} \to \mathbb{R}^{D_{\text{out}}}$. Multiple layers can then be composed together to form a *feedforwd neural network*.

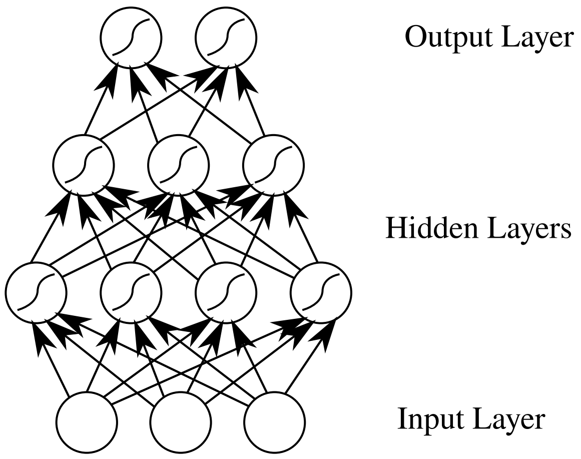

Fig. 2.7 Graph depiction of a feedforward neural network with 2 hidden layers

Although a single hidden layer is theoretically sufficient for a universal function approximator[5], the number of hidden units to guarantee reported theoretical bounds are usually unfeasibly large. Instead, recent work in *deep learning* has shown that deep models which contain many hidden layers can achieve strong performance across a variety of tasks[1].

The improved modeling capacity gained by composing multiple layers is due to the composition of multiple non-linear activation functions. In fact, it is easy to show that removing activation functions would make a deep network equivalent to a single matrix transform: let $\boldsymbol{W}_{l,l+1}$ denote the weights between layers l and $l+1$. The original neural network computes the function

$$\sigma\left(\boldsymbol{W}_{L,L-1}\sigma\left(\boldsymbol{W}_{L-1,L-2}\cdots\sigma\left(\boldsymbol{W}_{2,1}\boldsymbol{x}\right)\cdots\right)\right) \tag{2.2}$$

After removing the activation functions σ, we are left with

$$\boldsymbol{W}_{L,L-1}\boldsymbol{W}_{L-1,L-2}\cdots\boldsymbol{W}_{2,1}\boldsymbol{x} = \boldsymbol{x} = \tilde{\boldsymbol{W}}\boldsymbol{x} \tag{2.3}$$

where $\tilde{\boldsymbol{W}} = \left(\prod_{i=1}^{L-1}\boldsymbol{W}_{i,i+1}\right)$ is a matrix transform computing the same function as the neural network with activation functions removed.

2.2.3 Recurrent neural networks

While feedforward neural networks provide a flexible model for approximating arbitrary functions, they require a fixed-dimension input $\boldsymbol{x}$ and hence cannot be directly applied to sequential data $\boldsymbol{x} = (\boldsymbol{x}_t)_{t=1}^T$ where T may vary.

A naive method for extending feedforward networks would be to independently apply a feedforward network to compute $\boldsymbol{y}_t = f(\boldsymbol{x}_t \boldsymbol{\theta})$ at each timestep $1 \leq t \leq T$. However, this approach is only correct when each output $\boldsymbol{y}_t$ depends only on the input at the current time $\boldsymbol{x}_t$ and is independent of all prior inputs $\{\boldsymbol{x}_k\}_{k<t}$. This assumption is false in musical data: the current musical note usually is highly dependent on the sequence of notes leading up to it.

This shortcoming motivates *recurrent neural networks* (RNNs), which generalize feedforward networks by introducing time-delayed recurrent connections between hidden layers (Elman networks [8]) or from the output layers to the hidden layers (Jordan networks [16]). Mathematically, an (Elman-type) RNN is a discrete time dynamical system commonly parameterized as:

$$\boldsymbol{h}_t = \boldsymbol{W}_{xh}\sigma_{xh}\left(\boldsymbol{x}_t\right) + \boldsymbol{W}_{hh}\sigma_{hh}\left(\boldsymbol{h}_{t-1}\right) \tag{2.4}$$

$$\boldsymbol{y}_t = \boldsymbol{W}_{hy}\sigma_{hy}\left(\boldsymbol{h}_t\right) \tag{2.5}$$

where $\sigma_{..}(\cdot)$ are activation functions acting element-wise and $\theta = \{\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{W}_{hy}\}$ are the learned parameters. fig. 2.8 provides a graphical illustration of such a network. Notice that apart from the edges between hidden nodes, the network is identical to a regular feedforward network (fig. 2.7).

fliang: Why do we use Elman

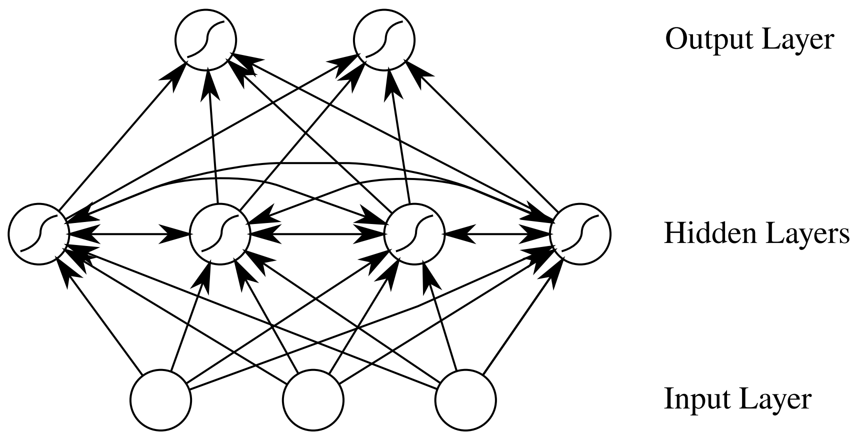

Fig. 2.8 Graph representation of an Elman-type RNN.

To apply the RNN over an input sequence $\boldsymbol{x}$, the activations of the hidden states are first initialized to an initial value $\boldsymbol{h} \in \mathbb{R}^{D_h}$. Next, for each timestep t the hidden layer activations are computed using the current input $\boldsymbol{x}_t$ and the previous hidden state activations $\boldsymbol{h}_{t-1}$. This motivates an alternative perspective on RNNs as a template consisting of a feedforward network with inputs $\{\boldsymbol{x}_t, \boldsymbol{h}_{t-1}\}$ (see fig. 2.9) replicated across time t.

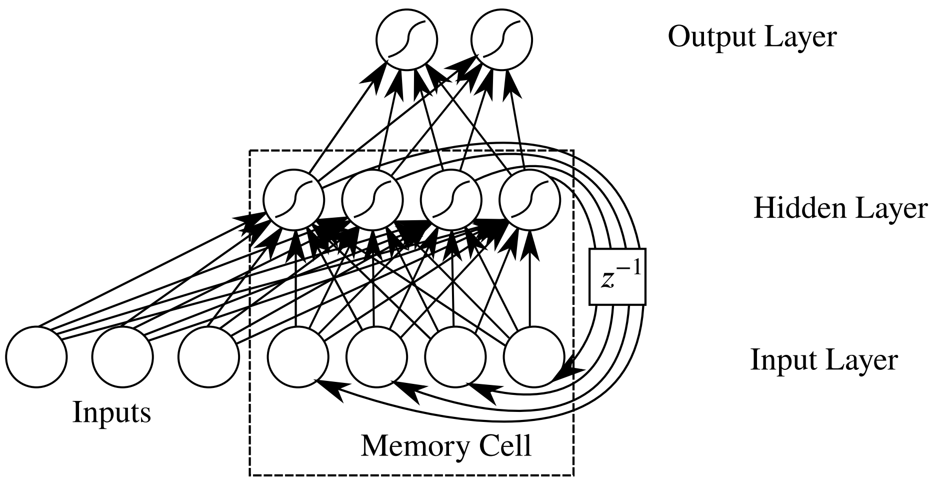

Fig. 2.9 Equivalent formulation of an Elman-type RNN treating the time-delayed hidden state $\boldsymbol{h}_{t-1}$ as additional inputs to a feedforward network

Memory cell abstraction

Notice that fig. 2.9 delineates the recurrent hidden state from the rest of the diagram, introducing an abstraction called a *memory cell*. This allows us to abstract away how $\boldsymbol{y}_t$ and $\boldsymbol{h}_t$ are computed from $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$, enabling discussion of RNNs applicable to many different implementations. Concretely, A memory cell is an interface which for each timestep t:

- Takes inputs
  - The current element in the input sequence $\boldsymbol{x}_t$
  - The previous hidden state $\boldsymbol{h}_{t-1}$
- Produces outputs
  - The next hidden state $\boldsymbol{h}_t = f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$
  - The outputs $\boldsymbol{y}_t = f_y(\boldsymbol{h}_t)$.

In future diagrams, we will abstractly represent the memory cell as a node labelled with $\boldsymbol{h}$.

Unrolling into a DAG

One important operation on RNNs is *unrolling* the template in fig. 2.9 into a chain of T replications with connected hidden states (fig. 2.10). This removes the time-delayed recurrent, converting the RNN into a finite directed acyclic graph where nodes represent pieces of data and edges $s \to t$ indicate that t is a function of s. This is identical to a feedforward network, justifying the view of RNNs as a dynamically-sized feedforward network with T layers and parameters tied across all layers.

fliang: Talk about how view of unrolled RNN as DNN inspires cross-polination e.g. gating to let information flow deeper: LSTM gates and highway networks, grid LSTMS

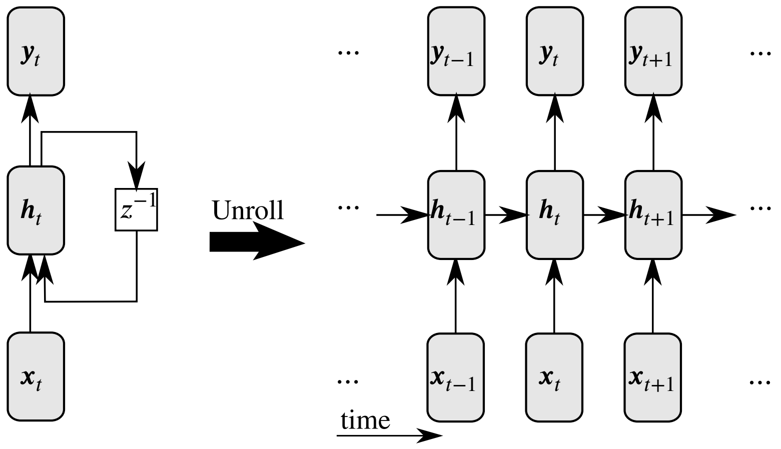
$\boldsymbol{y}_t$
$\boldsymbol{h}_t$
z^{-1}
$\boldsymbol{x}_t$
Unroll
$\boldsymbol{y}_{t-1}$ $\boldsymbol{y}_t$ $\boldsymbol{y}_{t+1}$
$\boldsymbol{h}_{t-1}$ $\boldsymbol{h}_t$ $\boldsymbol{h}_{t+1}$
$\boldsymbol{x}_{t-1}$ $\boldsymbol{x}_t$ $\boldsymbol{x}_{t+1}$
time

Fig. 2.10 Signal flow diagram representation of a single-layer RNN and its corresponding directed acyclic graph after unrolling

fig. 2.10 makes it obvious how the hidden state is carried along throughout the sequence of computations, giving rise to a useful alternative interpretation of the hidden state as a temporal memory mechanism. Under this interpretation, we can view the hidden state update $\boldsymbol{h}_t = f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ as *writing* information extracted from the current inputs $\boldsymbol{x}_t$ to the memory $\boldsymbol{h}_{t-1}$. Similarly, producing the outputs $\boldsymbol{y}_t = f_y(\boldsymbol{h}_t)$ can be seen as *reading* information from the hidden state.

fliang: Compare to N-grams; show how it's like an infinite context. One interpretation is to view the hidden state $\boldsymbol{h}_t$ as an infinite-length prior context window, summarizing all of the prior inputs into into a compact fixed-size vector.

Stacking memory cells to form multi-layer RNNs

Since the RNN outputs $\boldsymbol{y}$ also form a sequence with the same length as the inputs $\boldsymbol{x}$, they can be used as inputs into another RNN. This stacking of multiple memory cells is similar to the layering seen in deep neural networks, giving rise to the term *deep neural sequence models*

fliang: Cite?

. This is illustrated in fig. 2.11.

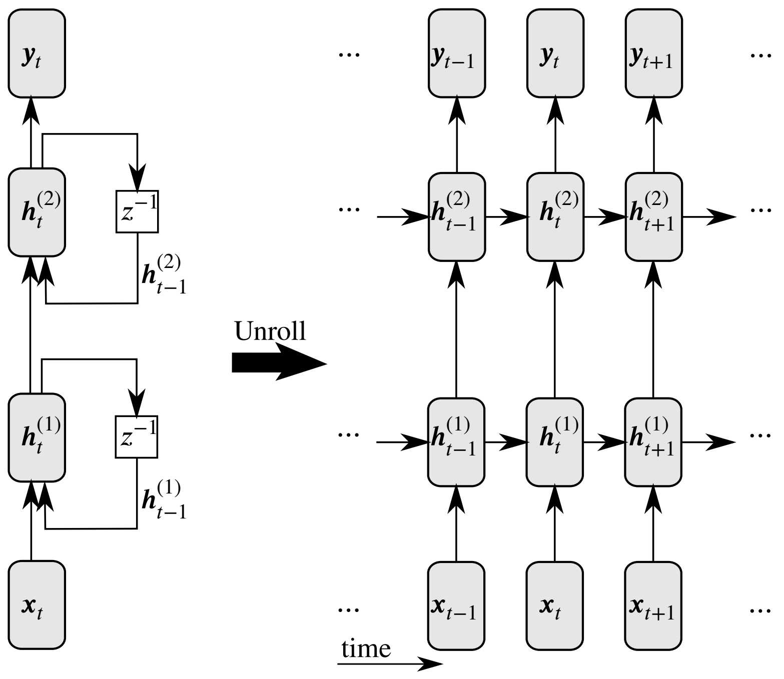

Fig. 2.11 Template and unrolling of a stacked 2-layer RNN

The greater modeling capabilities of multilayer RNNs can be attributed to two primary factors: composition of multiple non-linearities and an increase in the number of paths through which information can flow. The former is analogous to the feedforward case: stacking memory cells increases the number of non-linearities in the composite cell just like stacking multiple layers in feedforward networks. To understand the latter point, notice that in fig. 2.10 there is only a single path from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{y}_t$ hence the conditional independence $\boldsymbol{y}_t \perp\!\!\!\perp \boldsymbol{x}_{t-1} | \boldsymbol{h}_t^{(1)}$ is satisfied. However, in fig. 2.11 there are multiple paths from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{y}_t$ (e.g. passing through either $\boldsymbol{h}_{t-1}^{(2)} \to \boldsymbol{h}_t^{(2)}$ or $\boldsymbol{h}_{t-1}^{(1)} \to \boldsymbol{h}_t^{(1)}$) through which information may flow. Additionaly, the hidden state transitions occur on two seperate memory cells so parameters need not be tied and the stacked RNN can learn different time dynamics at each depth.

2.2.4 Modeling assumptions

fliang: Integrate better

We make the following assumptions about the sequence $\boldsymbol{x}_{1:T}$, $\boldsymbol{y}_{1:T}$, and $\boldsymbol{h}_{0:T}$:

1. Modified Markov assumption:

$$\forall t : P(\boldsymbol{h}_t|\boldsymbol{h}_{0:t-1}, \boldsymbol{x}_{1:t}) = P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) \tag{2.6}$$

2. Hidden State Stationarity:

$$\forall t_1, t_2 : P(\boldsymbol{h}_{t_1} = \boldsymbol{k}|\boldsymbol{h}_{t_1-1} = \boldsymbol{i}, \boldsymbol{x}_{t_1} = \boldsymbol{j}) = P(\boldsymbol{h}_{t_2} = \boldsymbol{k}|\boldsymbol{h}_{t_2-1} = \boldsymbol{i}, \boldsymbol{x}_{t_2} = \boldsymbol{j}) \tag{2.7}$$

3. Output Stationarity:

$$\forall t_1, t_2 : P(\boldsymbol{y}_{t_1} = \boldsymbol{j}|\boldsymbol{h}_{t_1} = \boldsymbol{i}) = P(\boldsymbol{y}_{t_2} = \boldsymbol{j}|\boldsymbol{h}_{t_2} = \boldsymbol{i}) \tag{2.8}$$

4. Output independence:

$$P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_{0:T}, \boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t, \boldsymbol{x}_t) \tag{2.9}$$

These assumptions imply the sequential factorization:

$$\begin{aligned}
&P(\boldsymbol{y}_{1:T}, \boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) && (2.10)\\
&= P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_{0:T}, \boldsymbol{x}_{1:T})P(\boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) && (2.11)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right) P(\boldsymbol{h}_{1:T}|\boldsymbol{h}_0, \boldsymbol{x}_{1:T}) \qquad \text{eq. (2.9)} && (2.12)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right)\left(\prod_{t=1}^{T} P(\boldsymbol{h}_t|\boldsymbol{h}_{0:t-1}, \boldsymbol{x}_{1:t})\right) && (2.13)\\
&= \left(\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t)\right)\left(\prod_{t=1}^{T} P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)\right) \qquad \text{eq. (2.6)} && (2.14)\\
&= \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t, \boldsymbol{x}_t)P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) && (2.15)\\
& && (2.16)
\end{aligned}$$

eq. (2.7) combined with eq. (2.8) imply that $P(\boldsymbol{y}_t|\boldsymbol{h}_t)$ and $P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1},\boldsymbol{x}_t)$ are time-invariant and can be modelled by the same recurrent function.

In RNNs, the hidden state dynamics $P(\boldsymbol{h}_t|\boldsymbol{h}_{t-1},\boldsymbol{x}_t)$ are deterministic:

$$\boldsymbol{h}_t = f_h(\boldsymbol{x}_t,\boldsymbol{h}_{t-1}) \tag{2.17}$$

Which means that $P(\boldsymbol{y}_{1:T},\boldsymbol{h}_{1:T}|\boldsymbol{h}_0,\boldsymbol{x}_{1:T}) = P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_0,\boldsymbol{x}_{1:T})$. This yields the factorization

$$P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_0,\boldsymbol{x}_{1:T}) = P(\boldsymbol{y}_{1:T},\boldsymbol{h}_{1:T}|\boldsymbol{h}_0,\boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t,\boldsymbol{x}_t) f_h(\boldsymbol{x}_t,\boldsymbol{h}_{t-1}) \tag{2.18}$$

fliang: Draw PGM

However, one minor problem remains. Let $\boldsymbol{z}_t = f_y(f_h(\boldsymbol{x}_t,\boldsymbol{h}_{t-1}))$ (with f_y and f_h as defined in

fliang: ref

) denote the outputs of the RNN model at time t. Note that $\boldsymbol{z}_t$ can be any real vector in $\mathbb{R}^{|V|}$

fliang: Define V to be the vocabulary

, but $P(\boldsymbol{x}_{t+1}|\boldsymbol{h}_{t-1},\boldsymbol{x}_t)$ is a probability vector constrained to sum to one.

Fortunately, we can treat $\boldsymbol{z}_t$ as the *scores* for a *Boltzmann distribution* (aka softmax):

$$P(\boldsymbol{y}_t = s|\boldsymbol{h}_{t-1},\boldsymbol{x}_t) = \frac{\exp\left(-\boldsymbol{z}_{t,s}/T\right)}{\sum_{k=1}^{K}\left(\exp -\boldsymbol{z}_{t,k}/T\right)} \tag{2.19}$$

where $T \in \mathbb{R}^+$ is a *temperature* parameter (set to $T = 1$ during training and varied during sampling). To keep notation compact, we omit writing this explicitly and understand $P(\boldsymbol{y}_t|\boldsymbol{h}_{t-1},\boldsymbol{x}_t)$ to mean the Boltzmann distribution parameterized by the scores $f_y(f_h(\boldsymbol{x}_t,\boldsymbol{h}_{t-1}))$.

Note the similarity between eq. (2.6)–eq. (2.9) and the assumptions for Hidden Markov models [22]. Discrepancies are due to the presence of an input sequence $\boldsymbol{x}_{1:T}$ in our sequence-to-sequence model.

fliang: Discuss validity of assumptions, namely output independence assuming hidden state and input summarize all prior context

2.2.5 Training RNNs: backpropogation through time

The parameters $\boldsymbol{\theta}$ of a RNN are typically learned from data to minimize a cost $\mathscr{E} = \sum_{1 \le t \le T} \mathscr{E}_t(\boldsymbol{x}_t)$ measuring the performance of the network on some task. This optimization is commonly carried out using iterative gradient descent methods, which require computation of the gradients $\frac{\partial \mathscr{E}}{\partial \boldsymbol{\theta}}$ at each iteration.

One approach for computing the necessary gradients is *backpropogation through time* (BPTT)[12], an adaptation of the backpropogation algorithm[17, 24] to the unrolled RNN computation graph. We can apply the chain rule to the unrolled RNN's computation graph in fig. 2.12 to obtain

$$\frac{\partial \mathscr{E}}{\partial \boldsymbol{\theta}} = \sum_{1 \le t \le T} \frac{\partial \mathscr{E}_t}{\partial \boldsymbol{\theta}} \tag{2.20}$$

$$\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{\theta}} = \sum_{1 \le k \le t} \left(\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{\theta}} \right) \tag{2.21}$$

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \prod_{t \ge i > k} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = \prod_{t \ge i > k} \boldsymbol{W}_{hh}^{\intercal} \operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h}_{i-1})\right) \tag{2.22}$$

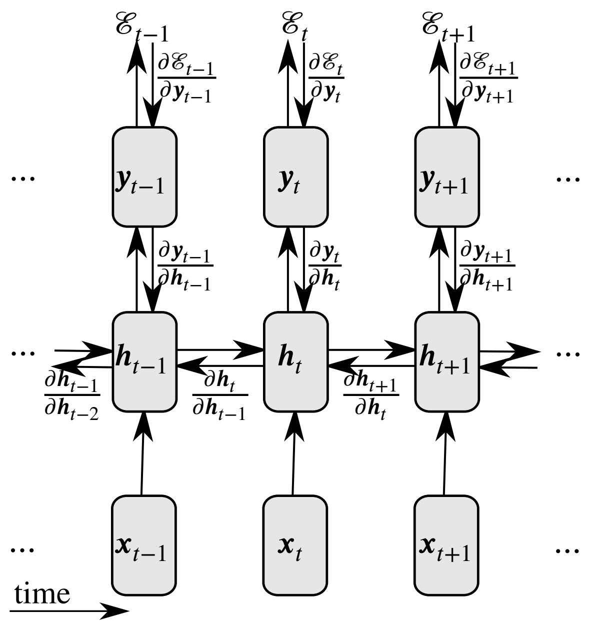
$\mathscr{E}_{t-1}$
$\mathscr{E}_t$
$\mathscr{E}_{t+1}$
$\frac{\partial \mathscr{E}_{t-1}}{\partial \boldsymbol{y}_{t-1}}$
$\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{y}_t}$
$\frac{\partial \mathscr{E}_{t+1}}{\partial \boldsymbol{y}_{t+1}}$
$\boldsymbol{y}_{t-1}$
$\boldsymbol{y}_t$
$\boldsymbol{y}_{t+1}$
$\frac{\partial \boldsymbol{y}_{t-1}}{\partial \boldsymbol{h}_{t-1}}$
$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t}$
$\frac{\partial \boldsymbol{y}_{t+1}}{\partial \boldsymbol{h}_{t+1}}$
$\boldsymbol{h}_{t-1}$
$\boldsymbol{h}_t$
$\boldsymbol{h}_{t+1}$
$\frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{h}_{t-2}}$
$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}$
$\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}$
$\boldsymbol{x}_{t-1}$
$\boldsymbol{x}_t$
$\boldsymbol{x}_{t+1}$
...
time

Fig. 2.12 The gradients passed along network edges during BPTT.

section 2.2.5 expresses how the error $\mathscr{E}_t$ at time t is a sum of *temporal contributions* $\frac{\partial \mathscr{E}_t}{\partial \boldsymbol{y}_t} \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial \boldsymbol{h}_k}{\partial \theta}$ measuring how θ's impact on $\boldsymbol{h}_k$ affects the cost at some future time $t > k$. The factors in section 2.2.5 measures the affect of the hidden state $\boldsymbol{h}_k$ on some future state $\boldsymbol{h}_t$

where $t > k$ and can be interpreted as transferring the error "in time" from step t back to step k [20].

Vanishing/exploding gradients

Unfortunately, naive implementations of section 2.2.3 and section 2.2.3 oftentimes suffers from two well known problems: the *vanishing gradient* and *exploding gradient*[2]. Broadly speaking, these problems are both related to the product in section 2.2.5 exponentially growing or shrinking for long timespans (i.e. $t \gg k$).

Following Pascanu et al. [20], let $\|\cdot\|$ be any submultiplicative matrix norm (e.g. Frobenius, spectral, nuclear, Shatten p-norms). Without loss of generality, we will use the *operator norm* defined as

$$\|A\| = \sup_{x \in \mathbb{R}^n; x \neq 0} \frac{|Ax|}{|x|} \tag{2.23}$$

where $|\cdot|$ is the standard Euclidian norm.

From submultiplicativity, we have that for any k

$$\left\| \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{h}_{k-1}} \right\| \leq \|\boldsymbol{W}_{hh}^{\mathsf{T}}\| \|\operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h}_{k-1})\right)\| \leq \gamma_{\boldsymbol{W}} \gamma_\sigma \tag{2.24}$$

where we have defined $\gamma_{\boldsymbol{W}} = \|\boldsymbol{W}_{hh}^{\mathsf{T}}\|$ and

$$\begin{aligned}
\gamma_\sigma &:= \sup_{h \in \mathbb{R}^n} \|\operatorname{diag}\left(\sigma'_{hh}(\boldsymbol{h})\right)\| && && (2.25)\\
&= \sup_{h \in \mathbb{R}^n} \max_i \sigma'_{hh}(\boldsymbol{h})_i && \text{Operator norm of diag} && (2.26)\\
&= \sup_{x \in \mathbb{R}} \sigma'_{hh}(x) && \sigma_{hh} \text{ acts elementwise} && (2.27)
\end{aligned}$$

Substituting back into section 2.2.5, we find that

$$\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \right\| = \left\| \prod_{t \geq i > k} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right\| \leq \prod_{t \geq i > k} \left\| \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right\| \leq (\gamma_{\boldsymbol{W}} \gamma_\sigma)^{t-k} \tag{2.28}$$

Hence, we see that a sufficient condition for vanishing gradients is for $\gamma_{\boldsymbol{W}} \gamma_\sigma < 1$, in which case $\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \right\| \to 0$ exponentially for long timespans $t \gg k$. $\square$

If γ_σ is bounded, sufficient conditions for vanishing gradients to occur may be written as

$$\gamma_{\boldsymbol{W}} < \frac{1}{\gamma_\sigma} \tag{2.29}$$

This is true for commonly used activation functions (e.g. $\gamma_\sigma = 1$ for $\sigma_{hh} = \tanh$, $\gamma_\sigma = 0.25$ for $\sigma_{hh} = \text{sigmoid}$).

The converse of the proof implies that $\|\boldsymbol{W}_{hh}^\intercal\| \geq \frac{1}{\gamma_\sigma}$ are necessary conditions for $\gamma_{\boldsymbol{W}}\gamma_\sigma > 1$ and exploding gradients to occur.

2.2.6 Long short term memory: solving the vanishing gradient

In order to build a model which learns long range dependencies, vanishing gradients must be avoided. This requires us to design our memory cells holding the hidden state $\boldsymbol{h}$ such that eq. (2.29) does not hold.

The *long short term memory* (LSTM) memory cell was proposed by Hochreiter and Schmid-huber [14] as a method for dealing with the vanishing gradient problem. It does so by enforcing *constant error flow* on section 2.2.5, that is

$$\boldsymbol{W}_{hh}^\intercal \sigma'_{hh}(\boldsymbol{h}_t) = \boldsymbol{I} \tag{2.30}$$

where $\boldsymbol{I}$ is the identity matrix.

Integrating the above differential equation yields $\boldsymbol{W}_{hh}\sigma_{hh}(\boldsymbol{h}_t) = \boldsymbol{h}_t$. Since this should hold for any hidden state $\boldsymbol{h}_t$, this means that:

1. $\boldsymbol{W}_{hh}$ must be full rank
2. σ_{hh} must be linear
3. $\boldsymbol{W}_{hh} \circ \sigma_{hh} = \boldsymbol{I}$

In the *constant error carousel* (CEC), this is ensured by setting $\sigma_{hh} = \boldsymbol{W}_{hh} = \boldsymbol{I}$. This may be interpreted as removing time dynamics on $\boldsymbol{h}$ in order to permit error signals to be transferred backwards in time (section 2.2.5) without modification (i.e. $\forall t \geq k : \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} = \boldsymbol{I}$).

In addition to using a CEC, a LSTM introduces three gates controlling access to the CEC:

- **Input gate**: scales input $\boldsymbol{x}_t$ elementwise by $\boldsymbol{i}_t \in [0, 1]$, *writes* to $\boldsymbol{h}_t$
- **Output gate**: scales output $\boldsymbol{y}_t$ elementwise by $\boldsymbol{o}_t \in [0, 1]$, *reads* from $\boldsymbol{h}_t$
- **Forget gate**: scales previous cell value $\boldsymbol{h}_{t-1}$ by $\boldsymbol{f}_t \in [0, 1]$, *resets* $\boldsymbol{h}_t$

Mathematically, the LSTM model is defined by the following set of equations:

$$\begin{aligned}
\boldsymbol{i}_t &= \text{sigmoid}(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{yi}\boldsymbol{y}_{t-1} + \boldsymbol{b}_i) && (2.31)\\
\boldsymbol{o}_t &= \text{sigmoid}(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{yo}\boldsymbol{y}_{t-1} + \boldsymbol{b}_o) && (2.32)\\
\boldsymbol{f}_t &= \text{sigmoid}(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{yf}\boldsymbol{y}_{t-1} + \boldsymbol{b}_f) && (2.33)\\
\boldsymbol{h}_t &= \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}_{xh}\boldsymbol{x}_t + y_{t-1}\boldsymbol{W}_{yh} + \boldsymbol{b}_h) && (2.34)\\
\boldsymbol{y}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{h}_t) && (2.35)
\end{aligned}$$

where $\odot$ denotes elementwise multiplication of vectors.

Notice that the gates ($\boldsymbol{i}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{f}_t$) controlling flow in and out of the CEC are all time varying. This can be interpreted as a mechanism enabling LSTM to explicitly learn which error signals to trap in the CEC and when to release them [14], enabling error signals to potentially be transported across long time lags.

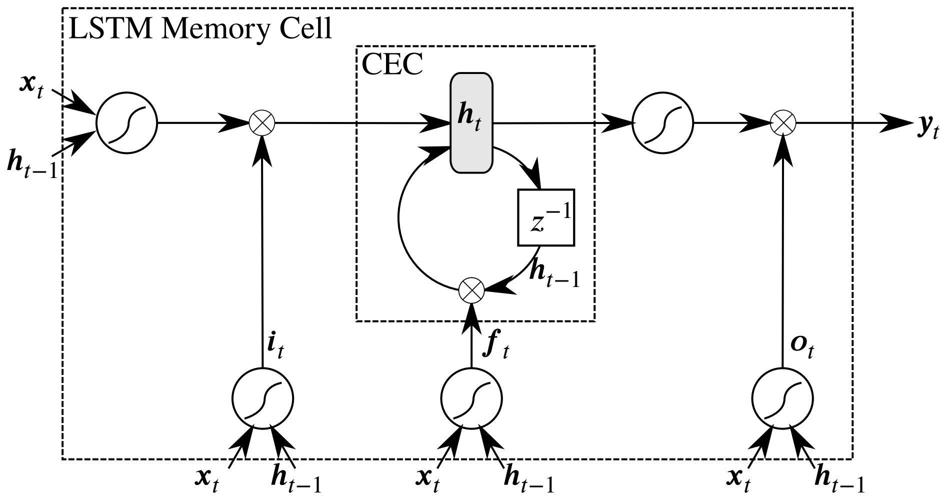

Fig. 2.13 Schematic for a single LSTM memory cell. Notice how the gates $\boldsymbol{i}_t$, $\boldsymbol{o}_t$, and $\boldsymbol{f}_t$ control access to the constant error carousel (CEC).

Some authors define LSTMs such that $\boldsymbol{h}_t$ is not used to compute gate activations, referring to the case where $\boldsymbol{h}_t$ is connected as "peephole connections"[11]. We will use LSTM to refer to the system of equations as written above.

Practicalities for successful applications of LSTMs

Many applications of LSTMs

fliang: cite examples

share some common practical techniques for ensuring successful training. Perhaps most important is *gradient norm clipping* [18, 20] where the gradient is rescaled whenever it exceeds a threshold. This is necessary because while vanishing gradients are mitigated by the use of CECs, LSTMs do not explicitly protect against exploding gradients.

Another common practice is the use of methods for reducing overfitting and improving generalization. In particular, *dropout* [25] can be applied to the connections between memory cells in a stacked RNN to regularize the learned features to be more robust to noise [29]. Additionally, *batch normalization*[15] can also be applied to to the memory cell hidden states to reduce co-variate shifts, accelerate training, and improve generalization.

Finally, applications of RNNs to long sequences can incur a prohibitively high cost for a single parameter update[Sutskever]. For instance, computing the gradient of an RNN on a sequence of length 1000 costs the equivalent of a forward and backward pass on a 1000 layer feedforward network. This issue is typically addressed by only back-propogating error signals a fixed number of timesteps back in the unrolled network, a technique known as *truncated BPTT*[28]. As the hidden states in the unrolled network have nevertheless been exposed to many timesteps, learning of long range structure is still possible.

References

[1] Bengio, Y. and Delalleau, O. (2011). On the expressive power of deep architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6925 LNAI:18–36.

[2] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

[3] Bharucha, J. J. and Todd, P. M. (1989). Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 13(4):44–53.

[4] Cooper, G. and Meyer, L. B. (1963). *The rhythmic structure of music*, volume 118. University of Chicago Press.

[5] Cybenko, G. (1993). Degree of approximation by superpositions of a sigmoidal function. *Approximation Theory and its Applications*, 9(3):17–28.

[6] Denton, C. and Fillion, M. (1997). The history of musical tuning and temperament during the classical and romantic periods.

[7] Eck, D. and Lapalme, J. (2008). Learning musical structure directly from sequences of music. *University of Montreal, Department of Computer Science, CP*, 6128.

[8] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[9] Franklin, J. A. (2006). Recurrent neural networks for music computation. *INFORMS Journal on Computing*, 18(3):321–338.

[10] Freedman, D. (2015). *Correlational Harmonic Metrics: Bridging Computational and Human Notions of Musical Harmony*. PhD thesis.

[11] Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE.

[12] Goller, C. and Kuchler, A. (1996). Learning task-dependent distributed representations by backpropagation through structure. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 347–352. IEEE.

[13] Handel, S. (1993). *Listening: An introduction to the perception of auditory events.* The MIT Press.

[14] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[15] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[16] Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495.

[17] Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7.

[18] Mikolov, T. (2012). *Statistical Language Models Based on Neural Networks*. PhD thesis.

[19] Papadopoulos, G. and Wiggins, G. (1999). Ai methods for algorithmic composition: A survey, a critical view and future prospects. In *AISB Symposium on Musical Creativity*, pages 110–117. Edinburgh, UK.

[20] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. *Proceedings of The 30th International Conference on Machine Learning*, (2):1310–1318.

[21] Pearce, M., Meredith, D., and Wiggins, G. (2002). Motivations and methodologies for automation of the compositional process. *Musicae Scientiae*, 6(2):119–147.

[22] Ramage, D. (2007). Hidden markov models fundamentals. *Lecture Notes. http://cs229. stanford. edu/section/cs229-hmm. pdf*.

[23] Randel, D. M. (1999). *The Harvard concise dictionary of music and musicians*. Harvard University Press.

[24] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[25] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[Sutskever] Sutskever, I. Training Recurrent Neural Networks - Ilia Sutskever - PhD thesis.

[27] Terhardt, E. (1974). Pitch, consonance, and harmony. *The Journal of the Acoustical Society of America*, 55(5):1061–1069.

[28] Williams, R. J. and Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501.

[29] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.