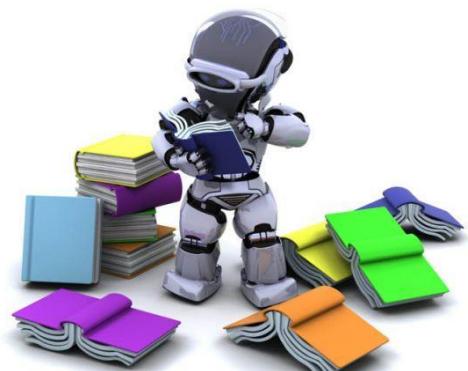


Introduction to Machine Learning

Foundations and Applications

Paul J. Atzberger
University of California Santa
Barbara



Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs)

Processing and generating variable-sized inputs/outputs.

Motivation: Neural networks with stateful processing of data in stages over time.

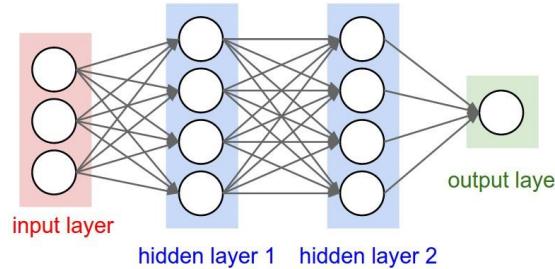
Mappings: sequences-to-sequences, sequence-to-vector, vector-to-sequence.

Applications: in Natural Language Processing (NLP), Audio Signals, Image Captioning, Language Translation, Handwriting Recognition.

Network architectures share weights but are dynamic depending on input/output length.

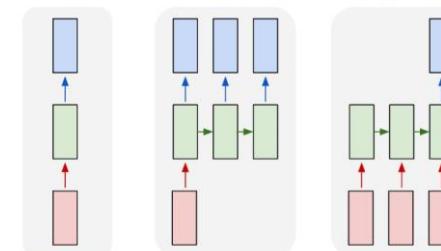
Recurrent connections pass state information from stage to stage.

Neural Network (NN): Fixed Input / Output Size

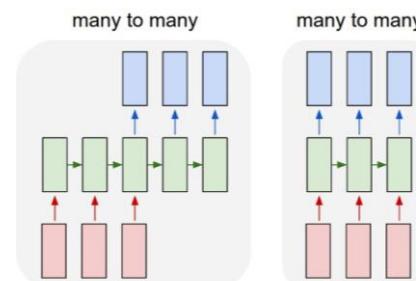


Variable Length Inputs/Outputs

one to one one to many many to one



many to many

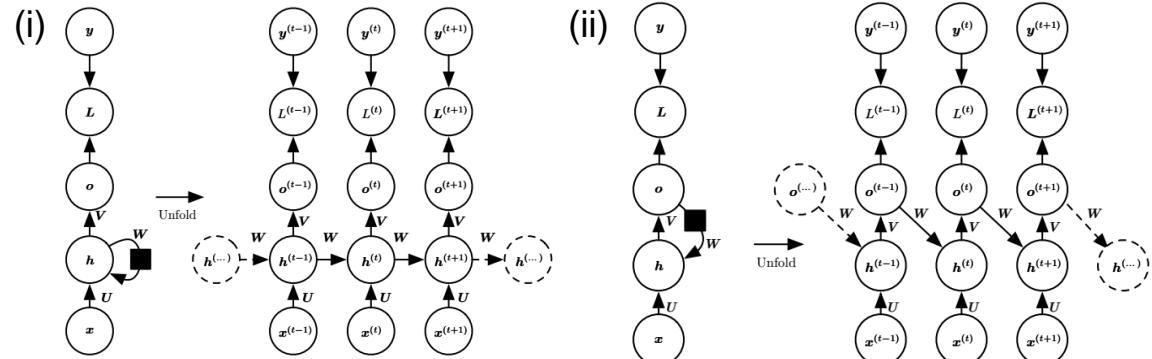


Karpathy 2016

RNNs: Common Architectures

RNNs can vary on how information transmitted to later times:

- (i) hidden-to-hidden coupling
- (ii) output-to-hidden coupling
- (iii) hidden-to-hidden single-output

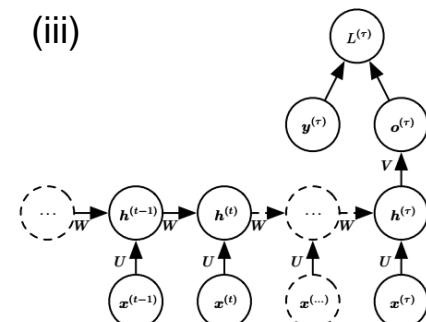


Architecture (i): Can realize universal Turing machine.

Architecture (ii): provides possible parallel training via *teacher-forcing* using target data output y .

Architecture (iii): similar to MLP, but with shared weights.

Training: Backpropagation Through Time (BPTT).



Goodfellow 2017

RNNs: Common Architectures

Architecture (i): hidden-to-hidden coupling.

The $g(z) = \tanh(z)$ typically used as activation in RNNs.

State updates:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = g(a^{(t)})$$

$$o^{(t)} = c + Vh^{(t)}$$

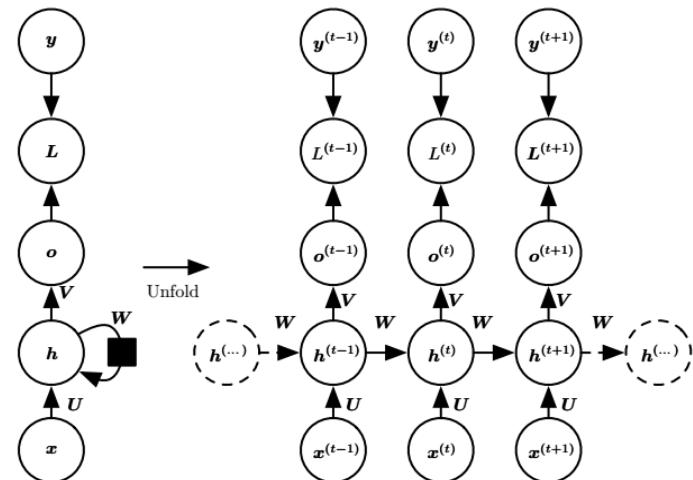
Proceeds from initial state $h^{(0)}$ over the steps $t = 1, 2, \dots, \tau$.

Output: $\tilde{y} = \text{softmax}(o^{(t)})$, \tilde{y}_j = probability of class j.

Training: Cross-Entropy loss $L(S) = \sum_t L^{(t)}(S)$

$$\begin{aligned} L^{(t)}(S) &= \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k -y_j^{[i]} \log(p(y_i = j | x^{(1),[i]}, x^{(2),[i]}, \dots, x^{(t),[i]})) \\ &= \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k -y_j^{[i]} \log(\tilde{y}_j^{[i]}) \end{aligned}$$

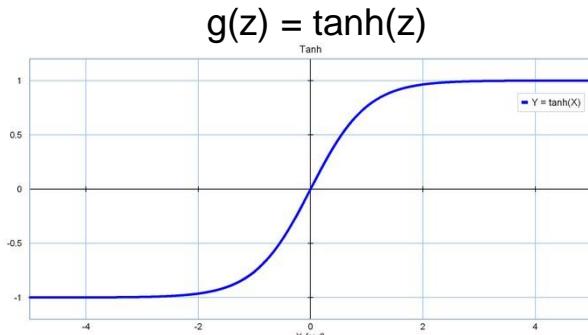
Expressive, but expensive to train, scales $\sim O(\tau)$,
BPTT, hard to parallelize.



Goodfellow 2017

1 – hot vector

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftarrow k^{\text{th}} \text{ class}$$



RNNs: Common Architectures

Architecture (ii): output-to-hidden coupling.

State updates:

$$a^{(t)} = b + W o^{(t-1)} + U x^{(t)}$$

$$h^{(t)} = g(a^{(t)})$$

$$o^{(t)} = c + V h^{(t)}$$

Proceeds from initial output $o^{(0)}$ over the steps $t = 1, 2, \dots, \tau$.

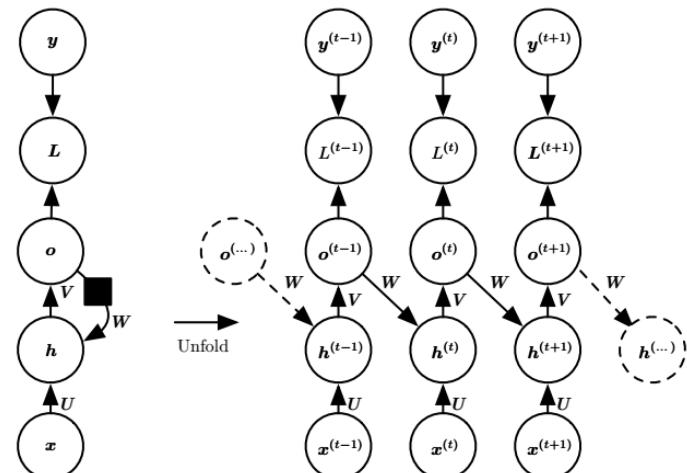
Output: $\tilde{y} = \text{softmax}(o^{(t)})$, \tilde{y}_j = probability of class j.

Training: Cross-Entropy loss $L(S) = \sum_t L^{(t)}(S)$

$$\begin{aligned} L^{(t)}(S) &= \frac{1}{m} \sum_{i=1}^m -y_j^{[i]} \log p(\tilde{y}_j^{[i]} | x^{(1),[i]}, x^{(2),[i]}, \dots, x^{(t),[i]}) \\ &= \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k -y_j^{[i]} \log (\tilde{y}_j^{[i]}) \end{aligned}$$

Trainable in parallel by replacing for step $o^{(t)} \sim y^{(t)}$,

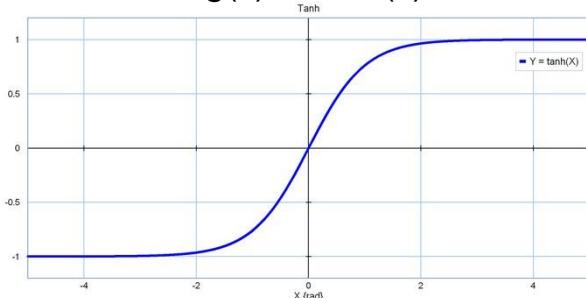
Teacher-Forcing Training.



Goodfellow 2017

1 – hot vector

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftarrow k^{\text{th}} \text{ class}$$



Teacher-Forcing Training

Architecture (ii): output-to-hidden coupling.

State updates:

$$a^{(t)} = b + W o^{(t-1)} + U x^{(t)}$$

$$h^{(t)} = g(a^{(t)})$$

$$o^{(t)} = c + V h^{(t)}$$

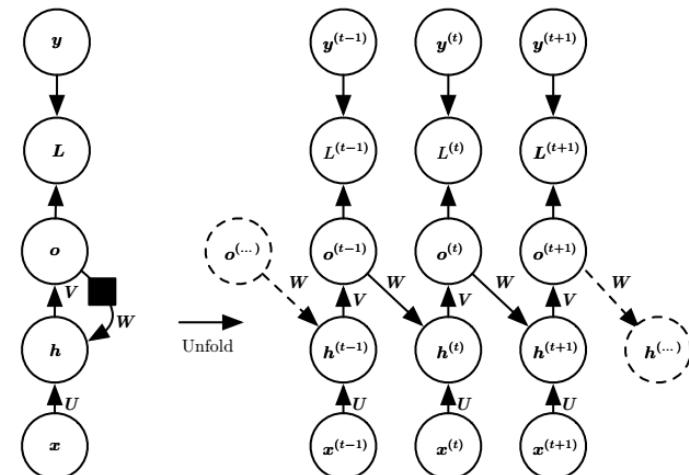
Trainable in parallel by replacing for step $o^{(t)} \sim y^{(t)}$,
Teacher-Forcing Training.

The network is trained in parallel by feeding into the next layer
 $o^{(t-1)} \sim y^{(t-1)}$ to the hidden unit $h^{(t)}$ (decouples times).

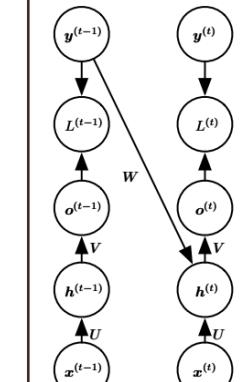
Testing is assessed by reintroducing recurrence feeding $o^{(t-1)}$ to $h^{(t)}$.

Works well in practice in many applications.

However, issues if training \mathbf{y} 's not representative of \mathbf{o} 's generated.

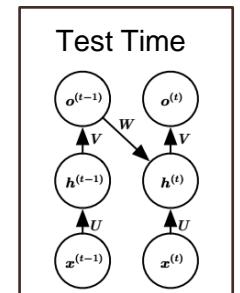


Training Time



Goodfellow 2017

Test Time



Sequence-to-Sequence RNNs

Sequence-to-Sequence Maps: Two RNNs combined to map sequences $(x^{(1)}, x^{(2)}, \dots, x^{(n_x)})$ to $(y^{(1)}, y^{(2)}, \dots, y^{(n_y)})$ with typically $n_x \neq n_y$.

First RNN (encoder) extracts a feature vector C from the sequence $\{x^{(i)}\}$ called the *context*.

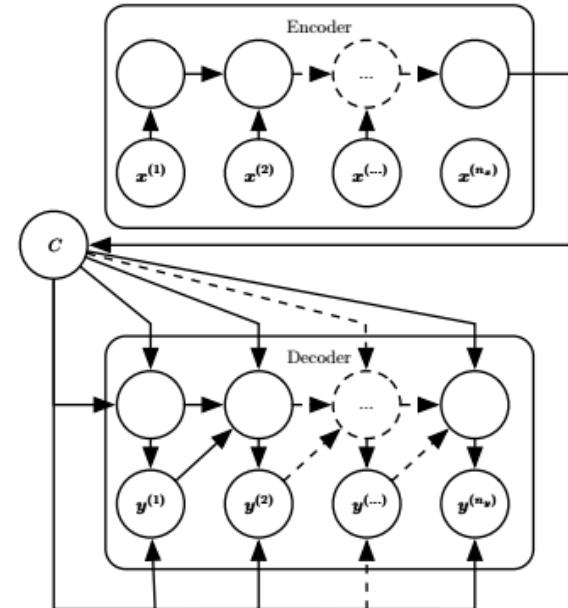
Second RNN (decoder) is driven by the *context* C to generate a new sequence $\{y^{(i)}\}$.

Context C is typically taken to be the last hidden state value $h^{(\tau)}$.

Limitations if fixed-sized C chosen too small. Variable-sized C is sometimes used, or an intermediate sequence.

Attention mechanisms also are used in practice.

Applications: Natural Language Processing (NLP), Language Translation, Image Captioning, and more.



Goodfellow 2017

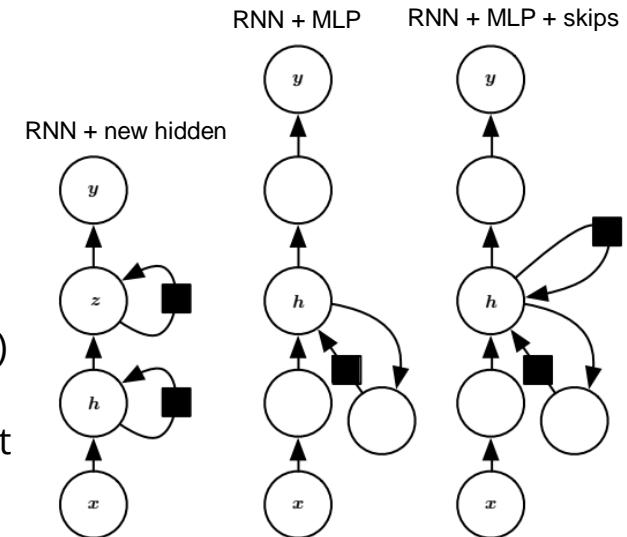
RNNs and Deep Neural Networks

Deep neural network ideas can be combined with RNNs.

Aim: achieve general transformations between states and steps.

A few ways this is done:

- (i) stack in layers new hidden states (MLP or other hypothesis spaces)
- (ii) process using MLP input-to-hidden, hidden-hidden, hidden-output
- (iii) introduce skip steps to help information flow



Goodfellow 2017

Trade-off: representation capacity vs optimization difficulty.

Optimization difficulty linked to depth increasing path-length
between the states at the different steps (exploding/vanishing gradients).

Skip steps can be used to reduce path-length.

Exploding/Vanishing Gradient Problem for RNNs

Exploding/vanishing gradient problem (illustration linear model):

$$h^{(t+1)} = Wh^{(t)} \rightarrow h^{(t+1)} = W^{t+1}h^{(0)}$$

For W symmetric, $W = Q\Lambda Q^T$ with Q orthogonal, $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$
 $W^{t+1} = Q\Lambda^{t+1}Q^T$.

Gradients are $\frac{\partial h^{(t+1)}}{\partial h^{(t)}} = W$ and $\frac{\partial h^{(t+1)}}{\partial h^{(0)}} = W^{t+1} = Q\Lambda^{t+1}Q^T$.

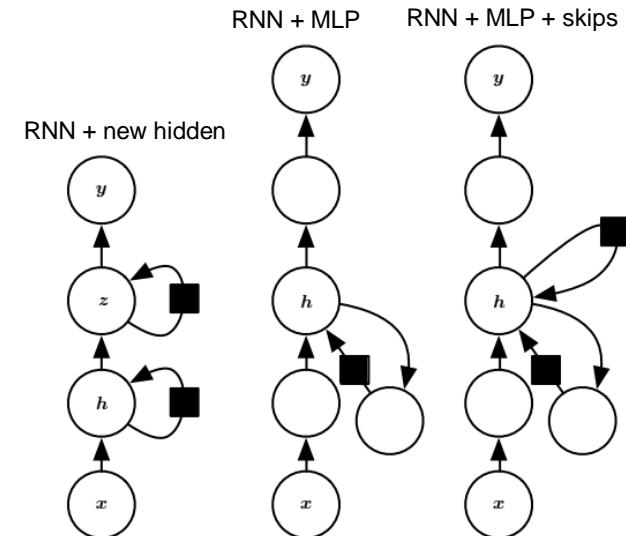
Eigenvalues $\lambda > 1$ the gradient explodes, and $\lambda < 1$ vanishes.

Non-linear setting: for NN we have similar behaviors.

Causes trouble for learning associations over long enough time-scales in RNNs.

Remedies: leaky units, removing/skip connections, gated units.

Gated units work well in practice for many tasks: LSTM, GRU Cells.



Goodfellow 2017

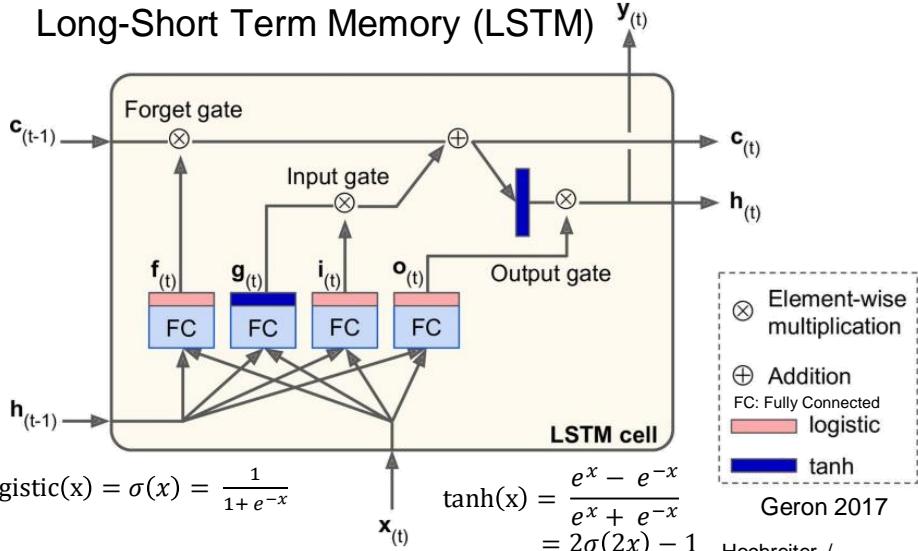
Gated Units: Long-Short Term Memory (LSTM)

Long-Short Term Memory (LSTM) introduces control units for

- (i) level $h_{(t)}$ state contributes to $c_{(t)}$, (input gate),
- (ii) level $c_{(t-1)}$ state contributes to $c_{(t)}$, (forget gate),
- (iii) level $o_{(t)}$ state contributes to $h_{(t)}, y_{(t)}$, (output gate).

LSTM Update:

$$\begin{aligned} i_{(t)} &= \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i) \\ f_{(t)} &= \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f) \\ o_{(t)} &= \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o) \\ g_{(t)} &= \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g) \\ c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \\ y_{(t)} &= h_{(t)} = o_{(t)} \otimes \tanh(c_{(t)}) \end{aligned}$$



Mitigates exploding/vanishing gradient problem

NN determines time-scales / key steps over which to retain information or to purge history.

Successful in practice on handwriting recognition, language processing, sentence generation, and other applications.

Hochreiter /
Schmidhuber (1995)
Geron 2017

Examples and Applications

Image Processing with RNNs

Even non-sequential data can be processed by RNNs.

Process sub-parts of the data as a sequence.

Attention mechanism: NN positions the location of sub-regions to read/write each stage.

RNNs can be used to generate data in parts.

Example: Reading address numbers from images of scenes of houses, buildings, streets.

RNN processes the sub-regions as a sequence of data to extract features and perform classification of address.

For large data, avoids processing all at once.

Recurrent Neural Network

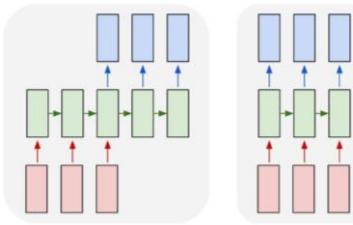
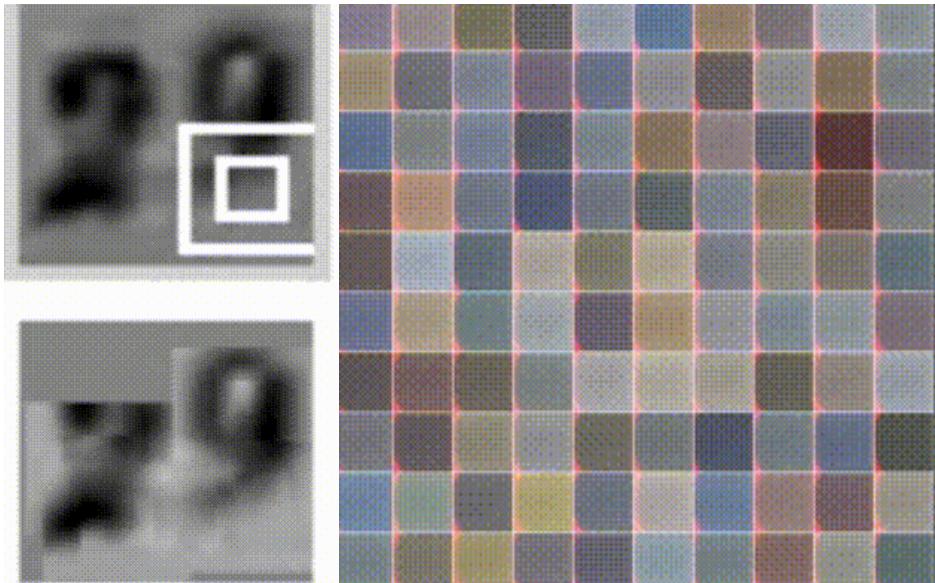


Image Processing / Generation (Street Addresses)
reading numbers
painting numbers



Karpathy 2015

RNNs and Text Datasets:

Text processing and generation with RNNs.

Predict next character in a sequence.

Alphabet with k symbols $\{\xi_1, \xi_2, \dots, \xi_k\}$.

RNN gives probability $y^{(t)}$ at stage t.

Probability for symbol j is $y_j^{(t)}$.

Example: Consider four letter symbolic set {h,e,l,o}.

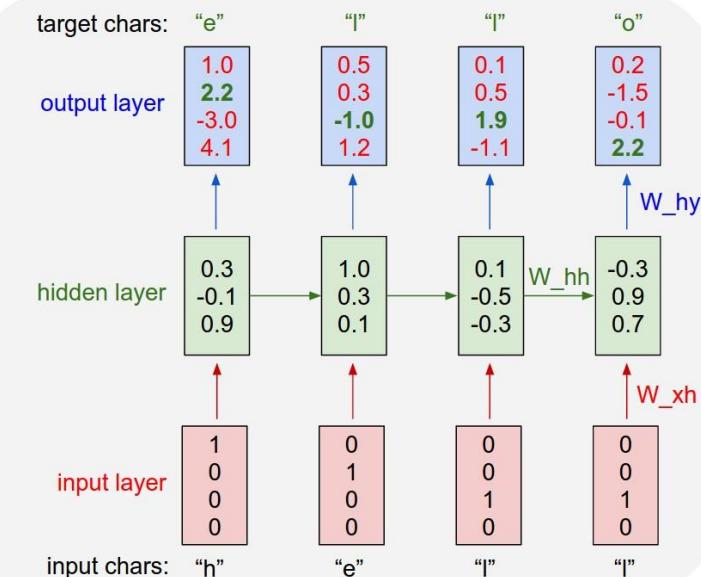
Train with sequence "hello."

Target sequence predictions h→e, e→l, l→l, l→o.

Train with cross-entropy loss.

Generate sequences: Feed an initial input and then feed the output back as the next input.

Text Generator: Predict Next Character



Karpathy 2015

RNNs and Text Datasets: Shakespeare's Works

Text processing and generation with RNNs.

RNN gives probability $y^{(t)}$ at stage t for next character symbol ξ_j .

Example: Train with cross-entropy loss on the works of Shakespeare (4.4MB of data).

Generate sequences: Feed an initial input and then feed the output back as the next input.

RNN only trained at the character level.

RNN samples the speakers names, spells words reasonably well, inserts commas, somewhat grammatical, etc...

RNN captures some of the inherent structure within the corpus of text.

RNN Generated Dialogue:

And open with their own conustersies; and thinking
your grave ship should ne'er break Humphrey's
eyes,

I am poor dear party to make his chamber
And hospish shameless frozen pride. Here name,
And light in plot legely in whom I said,
Glimmed by an argument of it sweet fears your
other mouth,
Such a great estimation would be run as this,
'Tis fit for them, 'tis talk before yourselves.

KING HENRY V:
And I must not see her, but go'st with unhappy
woe.

KING JOHN:
By Paris, it was fair.

DUCHESS:
Nay, I'll ever throw his honour.



RNNs and Music Synthesis

Music processing and generation with RNNs.

RNNs two sub-networks used to give probability $y^{(t)}$ at stage t for next chord and next duration, (chord expert / interval expert).

Example: Train with cross-entropy loss on jazz progressions and music sheets.

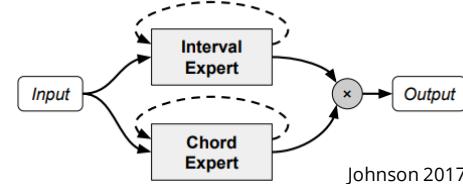
To generate music feed an initial input note/chord and then feed the output chord/duration back as the next input.

RNN only trained at the chord/duration level.

RNN samples melodies with variations, pauses, and other structures reminiscent of the musical style.

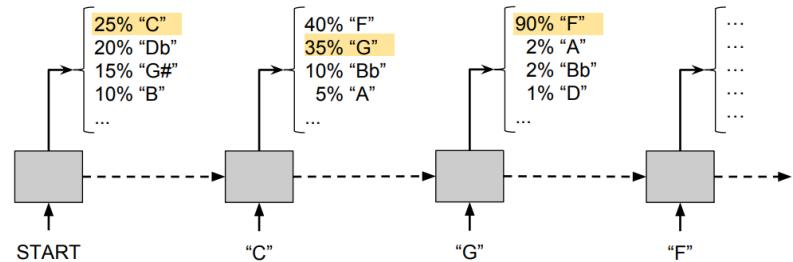
Shows promise of use of RNNs and D-NN in artistic/creative areas (soundtracks, entertainment, movies, etc...) [already starting to be used in industry for some tasks].

RNN Architecture



Johnson 2017

RNN Generator: Predicts Next Musical Note



Johnson 2017

Generated Music



piano
Johnson 2017

guitar chords



Summary

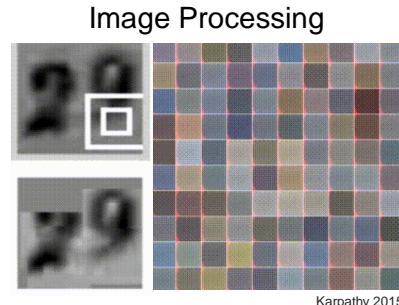
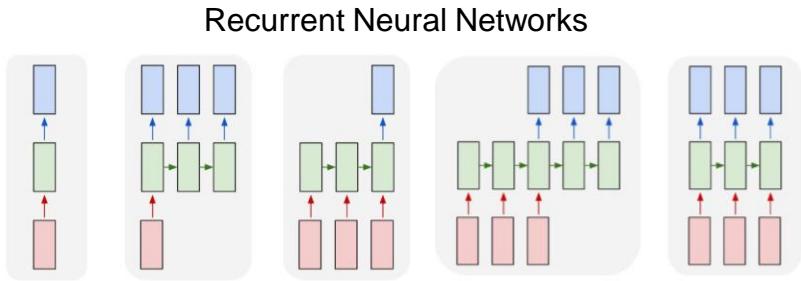
Summary: RNNs

Useful for processing and generating variable-sized inputs/outputs.

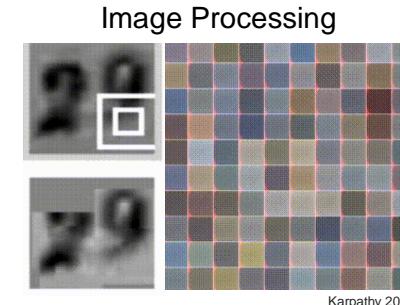
RNNs perform stateful processing of data in stages over time.

Mappings: sequences-to-sequences, sequence-to-vector, vector-to-sequence.

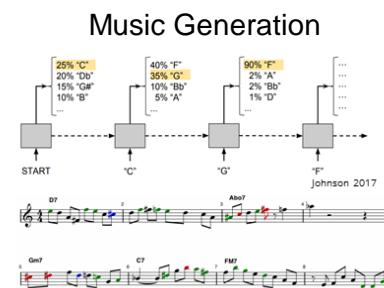
Many Applications: in Natural Language Processing (NLP), Audio Signals, Image Captioning, Language Translation, Handwriting Recognition, and other areas.



Recurrent Neural Networks



Gating Units



Music Generation

Text Generation

