Neural Networks for Machine Learning

Lecture 7a
Modeling sequences: A brief overview

Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

auto regressive model : predict the term from the sequence of previous terms

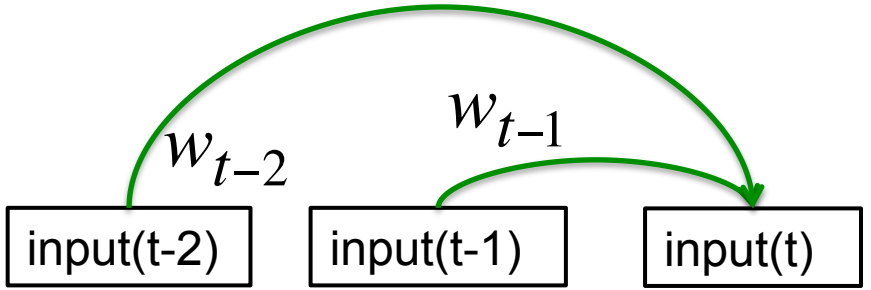
Getting targets when modeling sequences

- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain. english to french / audio to text.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.
- When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.
  - The target output sequence is the input sequence with an advance of 1 step.
  - This seems much more natural than trying to predict one pixel in an image from the other pixels, or one patch of an image from the rest of the image.
  - For temporal sequences there is a natural order for the predictions.
- Predicting the next term in a sequence blurs the distinction between supervised and unsupervised learning.
  - It uses methods designed for supervised learning, but it doesn't require a separate teaching signal.

Image does not deal with temporal sequences but similar approach work well for images.

Memoryless models for sequences

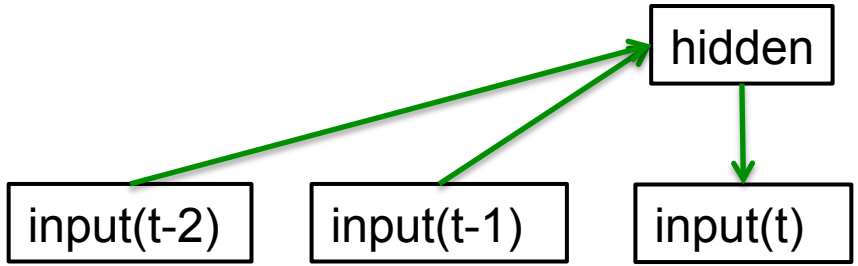
- Autoregressive models
  Predict the next term in a sequence from a fixed number of previous terms using “delay taps”.



weighted average of previous terms.

input can be a value or a vector of values

- Feed-forward neural nets
  These generalize autoregressive models by using one or more layers of non-linear hidden units.
  e.g. Bengio’s first language model.



Beyond memoryless models

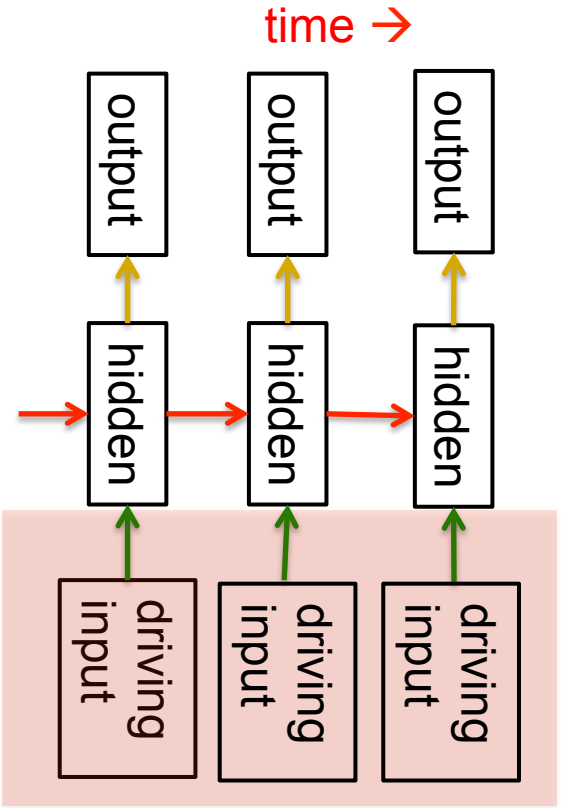
- If we give our generative model some hidden state, and if we give this hidden state its own internal dynamics, we get a much more interesting kind of model.
  - It can store information in its hidden state for a long time.
  - If the dynamics is noisy and the way it generates outputs from its hidden state is noisy, we can never know its exact hidden state.
  - The best we can do is to infer a probability distribution over the space of hidden state vectors.
- This inference is only tractable for two types of hidden state model.
  - The next three slides are mainly intended for people who already know about these two types of hidden state model. They show how RNNs differ.
  - Do not worry if you cannot follow the details.

so hidden state evolve according to its internal dynamics . produces some observations.
- IN gerneral inference from the observation is very hard .
-two model allows to infer probability distribution over the hidden state vector that might cause data.
- we assume data is being generated by model and try to infer the states model must be in to generate the data.

Linear Dynamical Systems (engineers love them!)

- These are generative models. They have a real-valued hidden state that cannot be observed directly.
  - The hidden state has linear dynamics with Gaussian noise and produces the observations using a linear model with Gaussian noise.
  - There may also be driving inputs.
- To predict the next output (so that we can shoot down the missile) we need to infer the hidden state.
  - A linearly transformed Gaussian is a Gaussian. So the distribution over the hidden

red arrows



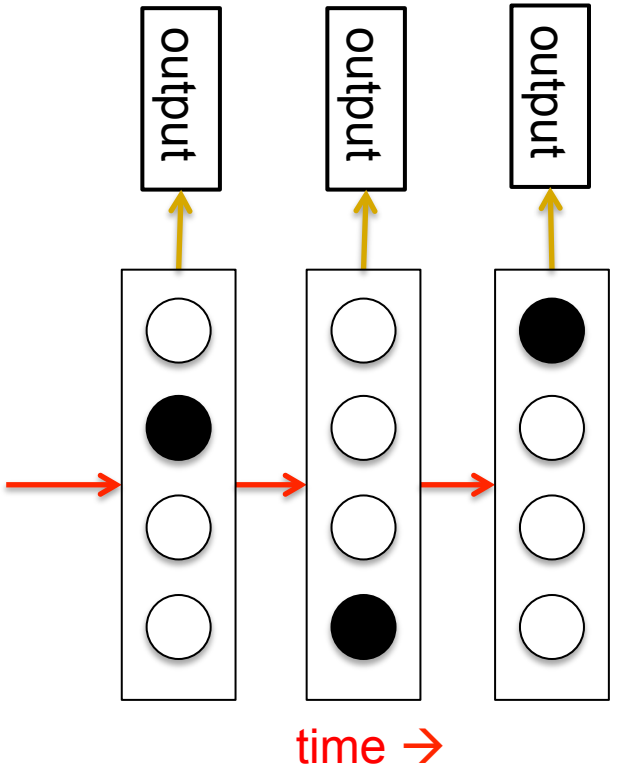
earlier usage was track planets from noisy observations.

- A linearly transformed Gaussian is a Gaussian . So the distritbution over the hidden state given the data so far is also Gaussian. (Full covariance Gaussian)
- quite complicated to compute
- Can be computed efficiently by Kalman Filtering.

- Given output we can't be sure which hidden state it is in but we can get a gaussian distribution over possible hidden state it might be in .

Hidden Markov Models (computer scientists love them!)

- Hidden Markov Models have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.
  - We cannot be sure which state produced a given output. So the state is "hidden". c
  - It is easy to represent a probability distribution across N states with N numbers.
- To predict the next output we need to infer the probability distribution over hidden states.
  - HMMs have efficient algorithms for inference and learning



- state which be are in does not completely determines what output it produces.

- appropriate for speech recognition
-DP algorithms exists.

A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data.
  - At each time step it must select one of its hidden states. So with N hidden states it can only remember log(N) bits about what it generated so far.

past states only remembered and to embed that log(n) bits required.

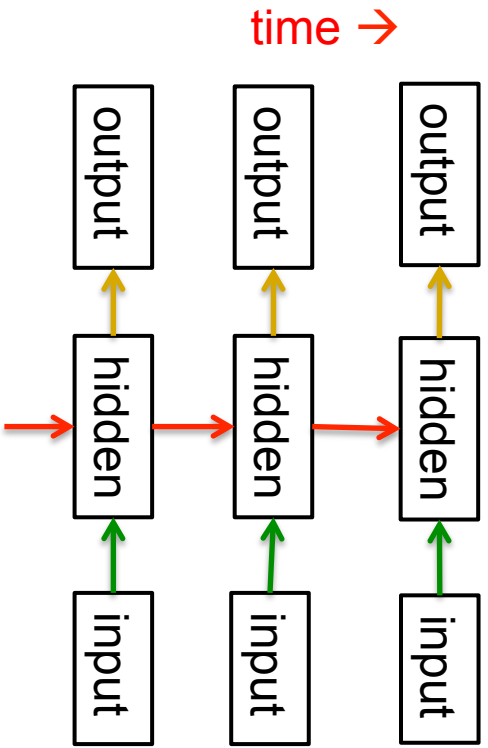
- Consider the information that the first half of an utterance contains about the second half:
  - The syntax needs to fit (e.g. number and tense agreement).
  - The semantics needs to fit. The intonation needs to fit.

with respect to first half semantics and intonation should be fit

  - The accent, rate, volume, and vocal tract characteristics must all fit.
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half. 2^100

Recurrent neural networks

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.



several diffrent units can be at once so the can remember several different things.
RNN allows non linear dynamics.

Do generative models need to be stochastic?

- Linear dynamical systems and hidden Markov models are stochastic models.
  - But the posterior probability distribution over their hidden states given the observed data so far is a deterministic function of the data.

- Recurrent neural networks are deterministic.
  - So think of the hidden state of an RNN as the equivalent of the deterministic probability distribution over hidden states in a linear dynamical system or hidden Markov model.

both LDS & HMM involves intrinsic noise
- inference algo ends up with prob. distribution.
- distribution are numbers deterministic function of data so far

Recurrent neural networks

- What kinds of behaviour can RNNs exhibit?
  - They can oscillate. Good for motor control?
  - They can settle to point attractors. Good for retrieving memories?
  - They can behave chaotically. Bad for information processing?
  - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects.
- But the computational power of RNNs makes them very hard to train.
  - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer).

2

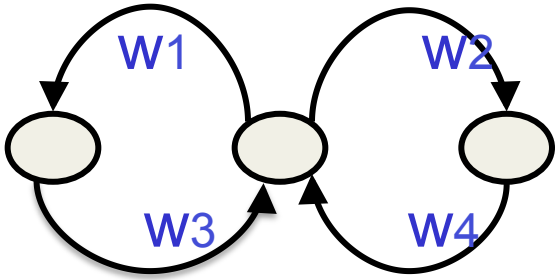
So the idea is you have a sort of rough idea of what you're trying to retrieve. You then let the system settle down to a stable point and those stable points correspond to the things you know about. And so by settling to that stable point you retrieve a memory.

Neural Networks for Machine Learning

Lecture 7b
Training RNNs with backpropagation

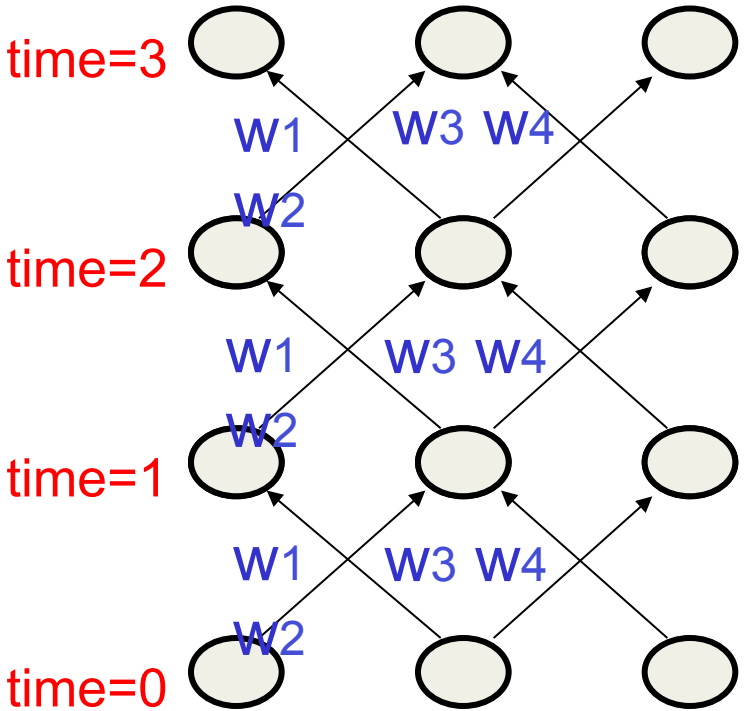
Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

The equivalence between feedforward nets and recurrent nets



Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.



assume like a FFNN
layered FFNNN weights to be constrained to be same at each layer

Reminder: Backpropagation with weight constraints

- It is easy to modify the backprop algorithm to incorporate linear constraints between the weights.
- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - So if the weights started off satisfying the constraints, they will continue to satisfy them.

$$To\ \ constrain: \quad w_1 = w_2$$

$$we\ \ need: \quad \Delta w_1 = \Delta w_2$$

$$compute: \quad \frac{\partial E}{\partial w_1} \quad and \quad \frac{\partial E}{\partial w_2}$$

$$use \quad \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \quad for\ w_1\ and\ w_2$$

Backpropagation through time

- We can think of the recurrent net as a layered, feed-forward net with shared weights and then train the feed-forward net with weight constraints.
- We can also think of this training algorithm in the time domain:
  - The forward pass builds up a stack of the activities of all the units at each time step.
  - The backward pass peels activities off the stack to compute the error derivatives at each time step.
  - After the backward pass we add together the derivatives at all the different times for each weight.

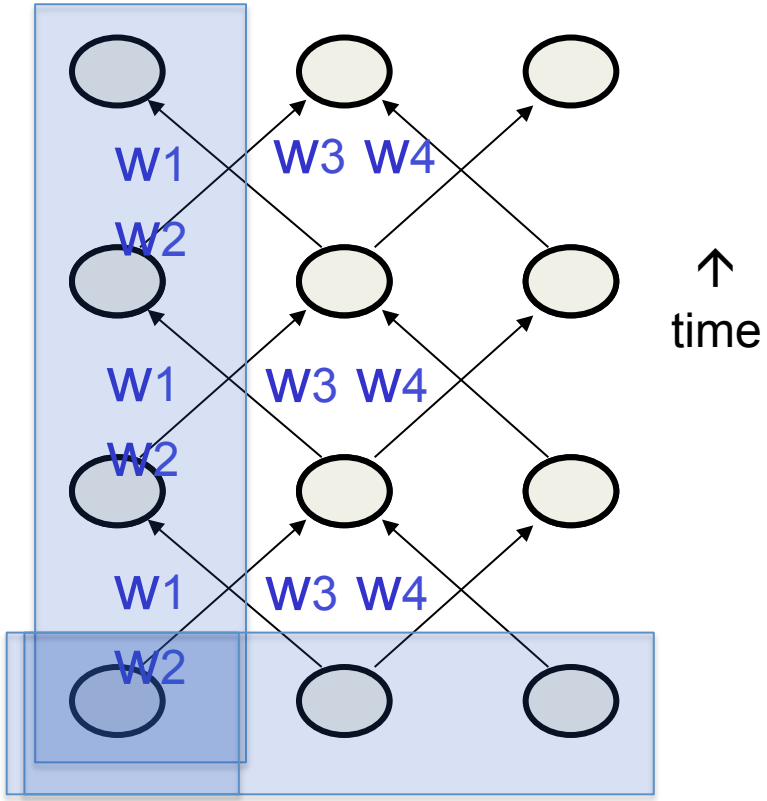
An irritating extra issue

- We need to specify the initial activity state of all the hidden and output units.
- We could just fix these initial states to have some default value like 0.5.
- But it is better to treat the initial states as learned parameters.
- We learn them in the same way as we learn the weights.
  - Start off with an initial random guess for the initial states.
  - At the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state.
  - Adjust the initial states by following the negative gradient.

learn initial state like parameter rather than activities

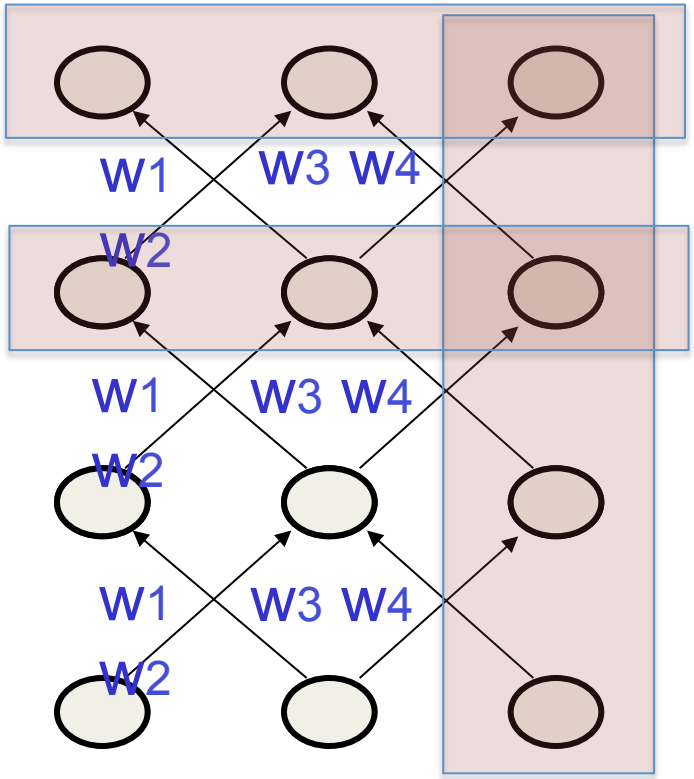
Providing input to recurrent networks

- We can specify inputs in several ways:
  - Specify the initial states of all the units.
  - Specify the initial states of a subset of the units.
  - Specify the states of the same subset of the units at every time step.
    - This is the natural way to model most sequential data.



Teaching signals for recurrent networks

- We can specify targets in several ways:
  - Specify desired final activities of all the units
  - Specify desired activities of all units for the last few steps
    - Good for learning attractors
    - It is easy to add in extra error derivatives as we backpropagate.
  - Specify the desired activity of a subset of the units.
    - The other units are input or hidden units.



specify several states to end we can force it to learn attractors.

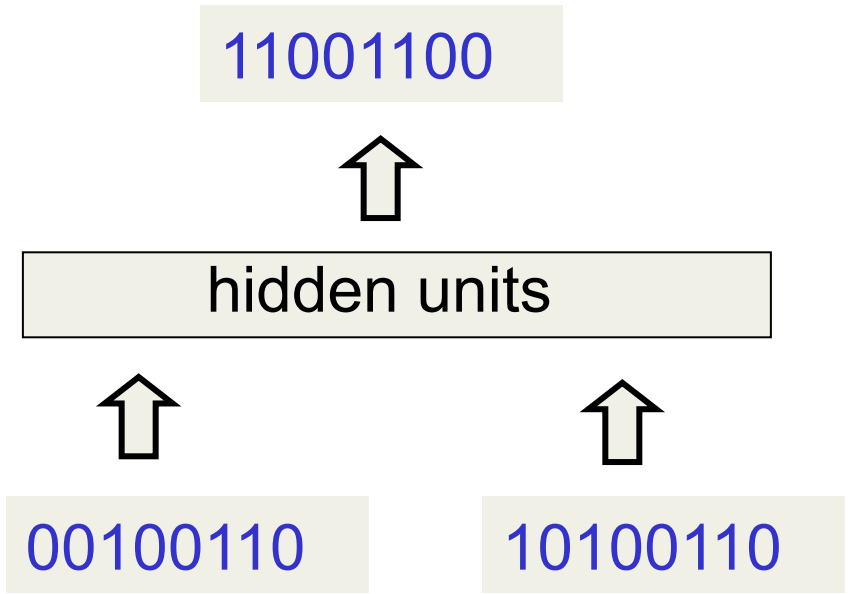
Neural Networks for Machine Learning

Lecture 7c
A toy example of training an RNN

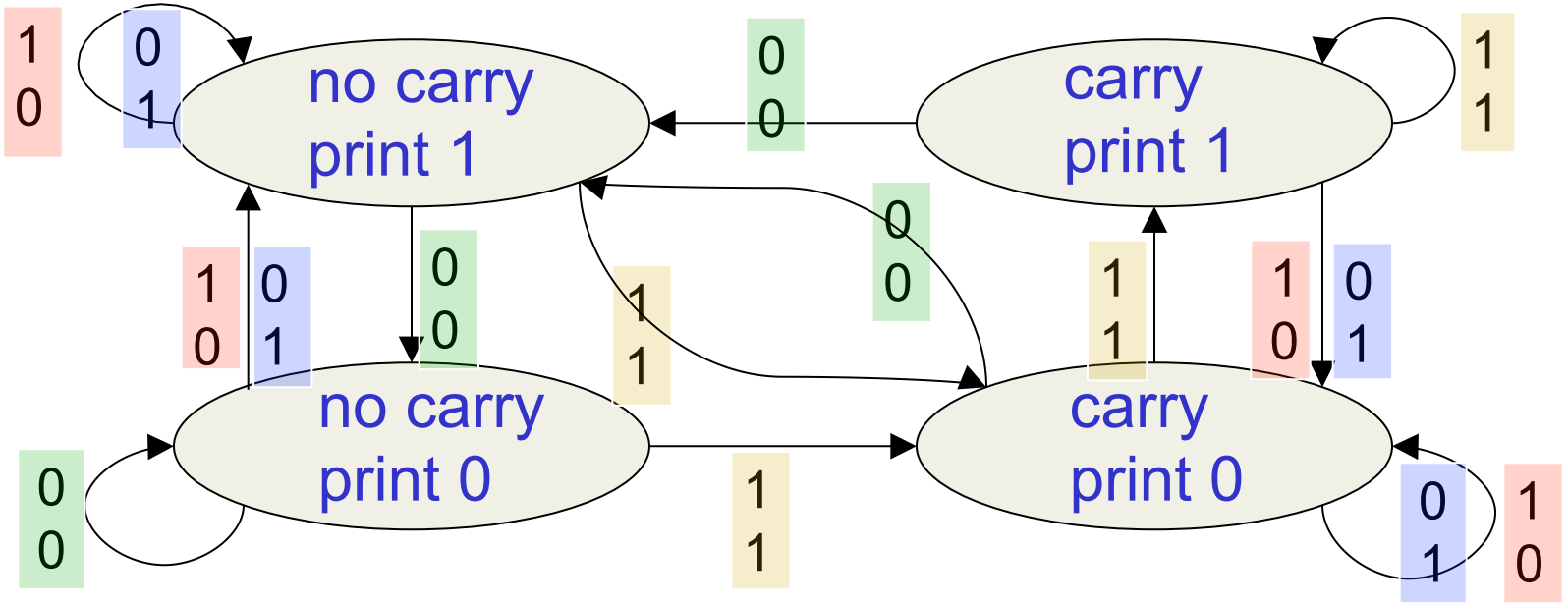
Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

A good toy problem for a recurrent network

- We can train a feedforward net to do binary addition, but there are obvious regularities that it cannot capture efficiently.
  - We must decide in advance the maximum number of digits in each number.
  - The processing applied to the beginning of a long number does not generalize to the end of the long number because it uses different weights.
- As a result, feedforward nets do not generalize well on the binary addition task.



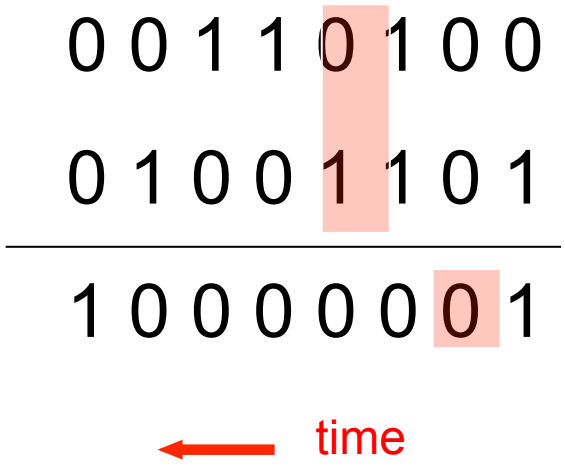
The algorithm for binary addition



This is a finite state automaton. It decides what transition to make by looking at the next column. It prints after making the transition. It moves from right to left over the two input numbers.

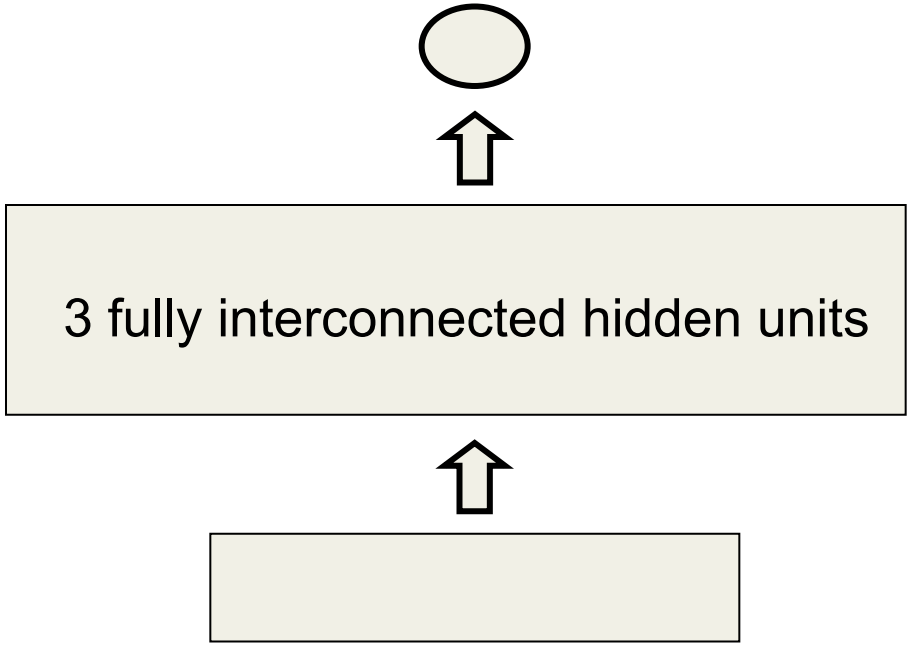
A recurrent net for binary addition

- The network has two input units and one output unit.
- It is given two input digits at each time step.
- The desired output at each time step is the output for the column that was provided as input two time steps ago. why 2 step time delay?
  - It takes one time step to update the hidden units based on the two input digits.
  - It takes another time step for the hidden units to cause the output. to produce output from the hidden state .



The connectivity of the network

- The 3 hidden units are fully interconnected in both directions.
  - This allows a hidden activity pattern at one time step to vote for the hidden activity pattern at the next time step.
- The input units have feedforward connections that allow then to vote for the next hidden activity pattern.



The connection b/w hidden units allow the pattern at one time step to influence the hidden activity pattern of next time steps.

What the network learns

- It learns four distinct patterns of activity for the 3 hidden units. These patterns correspond to the nodes in the finite state automaton.
  - Do not confuse units in a neural network with nodes in a finite state automaton. Nodes are like activity vectors.
  - The automaton is restricted to be in exactly one state at each time. The hidden units are restricted to have exactly one vector of activity at each time.
- A recurrent network can emulate a finite state automaton, but it is exponentially more powerful. With N hidden neurons it has 2^N possible binary activity vectors (but only N^2 weights)
  - This is important when the input stream has two separate things going on at once.
  - A finite state automaton needs to square its number of states.
  - An RNN needs to double its number of units.

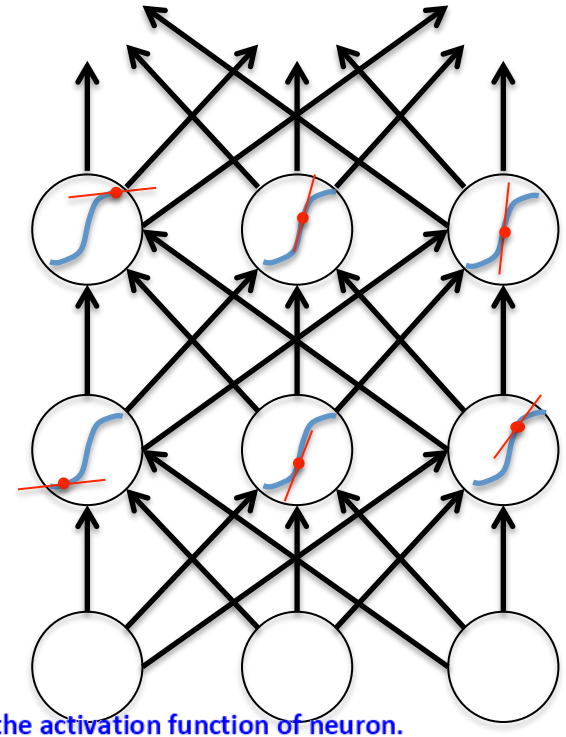
Neural Networks for Machine Learning

Lecture 7d
Why it is difficult to train an RNN

Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

The backward pass is linear

- There is a big difference between the forward and backward passes.
- In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding.
- The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double.
  - The forward pass determines the slope of the linear function used for backpropagating through each neuron.



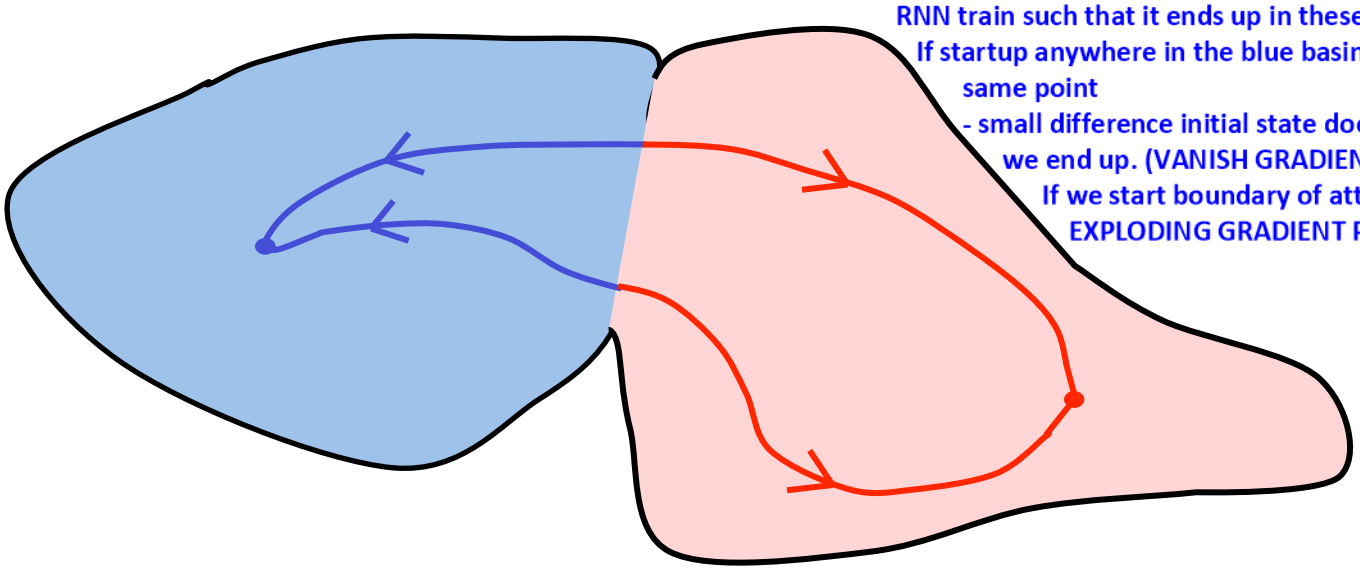
IT suffers from a problem of linear system if you iterate it will explode or die

Blue curve is the activation function of neuron.
red dot is the activation level of neurons on te forward pass.
red line are tangent of the blue at the point (red dot).

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  - If the weights are small, the gradients shrink exponentially.
  - If the weights are big the gradients grow exponentially.
- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers. for a shallow NN
- In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.
  - So RNNs have difficulty dealing with long-range dependencies.

Why the back-propagated gradient blows up



- If we start a trajectory within an attractor, small changes in where we start make no difference to where we end up.
- But if we start almost exactly on the boundary, tiny changes can make a huge difference.

Four effective ways to learn an RNN

- Long Short Term Memory
  Make the RNN out of little modules that are designed to remember values for a long time.
- Hessian Free Optimization: Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature.
  - The HF optimizer (Martens & Sutskever, 2011) is good at this.
- Echo State Networks: Initialize the input→hidden and hidden→hidden and output→hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input.
  - ESNs only need to learn the hidden→output connections.
- Good initialization with momentum
  Initialize like in Echo State Networks, but then learn all of the connections using momentum.

Neural Networks for Machine Learning

Lecture 7e
Long term short term memory

Geoffrey Hinton
Nitish Srivastava,
Kevin Swersky
Tijmen Tieleman
Abdel-rahman Mohamed

Dynamic state to be short term memory.
- allowed with the help of gates.
to decide whether the information should be gatted in and gatted out when needed
- succesful for task like recognizing handwritting

Long Short Term Memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- They designed a memory cell using logistic and linear units with multiplicative interactions.

the rest of RN determines the state of write gate. and whatever the input from the rest of net to the memory cell is get stored in the memory cell

- Information gets into the cell whenever its "write" gate is on. (logistic)
- The information stays in the cell so long as its "keep" gate is on.
- Information can be read from the cell by turning on its "read" gate.

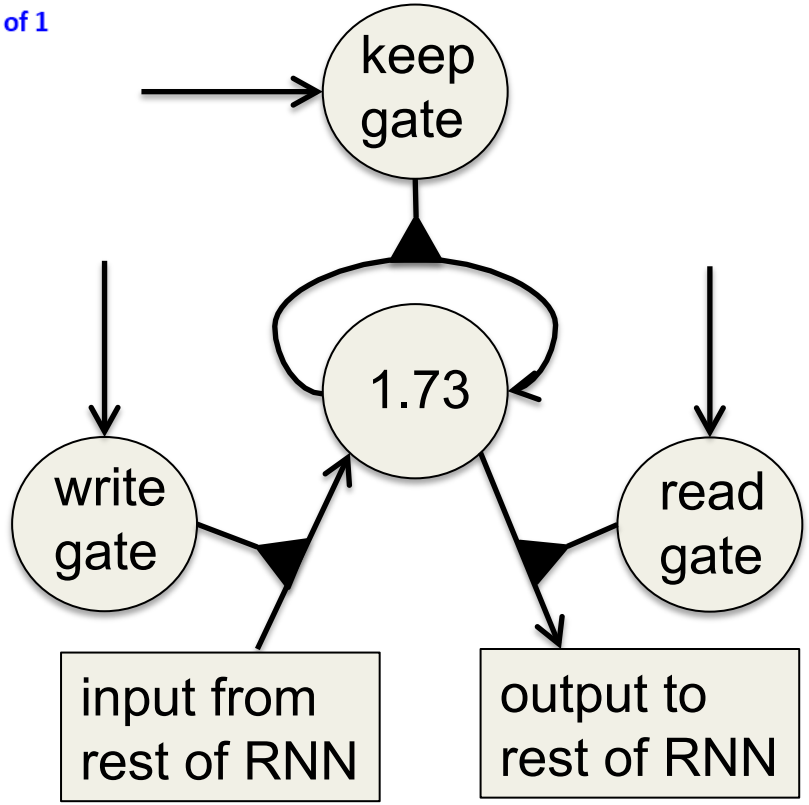
Implementing a memory cell in a neural network

memory cell has analog value that keeps writting to itself at each time step by weigh of 1

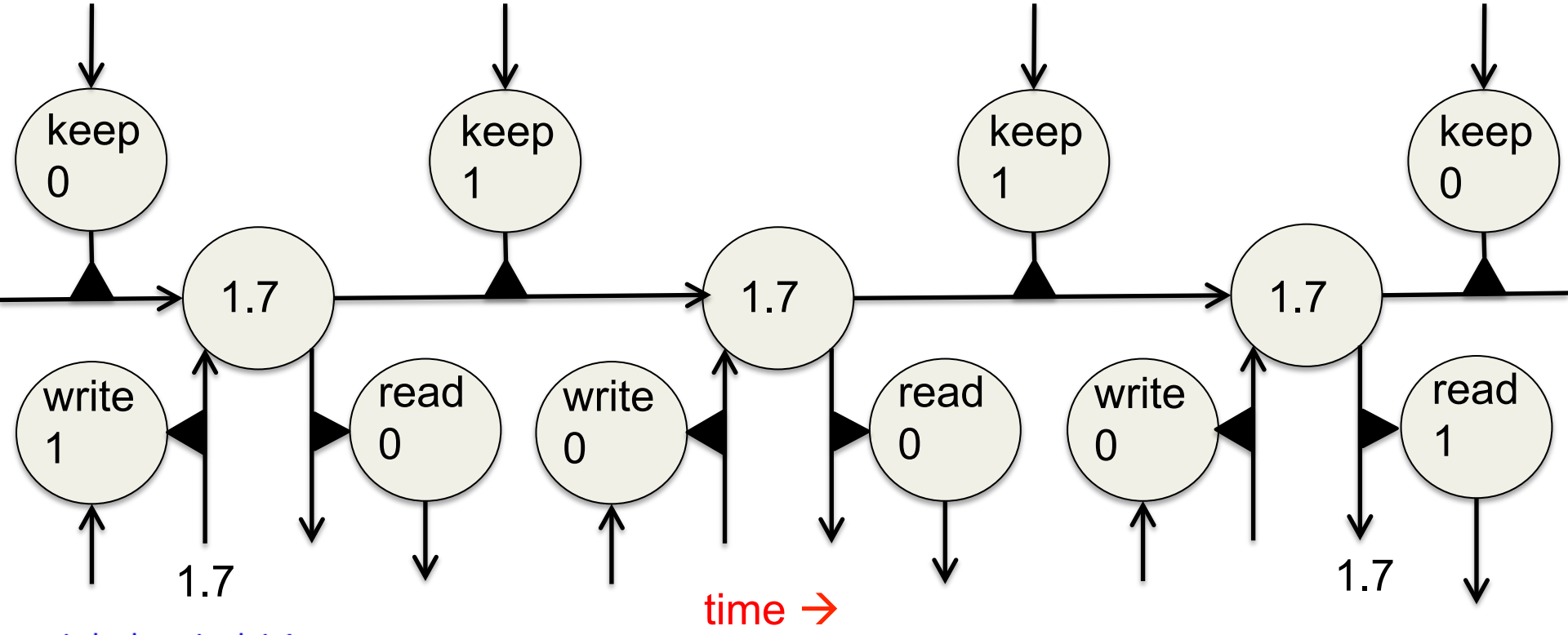
To preserve information for a long time in the activities of an RNN, we use a circuit that implements an analog memory cell.

- A linear unit that has a self-link with a weight of 1 will maintain its state.
- Information is stored in the cell by activating its write gate.
- Information is retrieved by activating the read gate.
- We can backpropagate through this circuit because logistics are have nice derivatives.

set the value of keep gate 0 to make information disaappear.



Backpropagation through a memory cell



wieght along triangle is 1 .
no. attenuation in back propogation

Reading cursive handwriting

- This is a natural task for an RNN.
- The input is a sequence of (x,y,p) coordinates of the tip of the pen, where p indicates whether the pen is up or down.
- The output is a sequence of characters.

- Graves & Schmidhuber (2009) showed that RNNs with LSTM are currently the best systems for reading cursive writing.
  - They used a sequence of small images as input rather than pen coordinates.

A demonstration of online handwriting recognition by an RNN with Long Short Term Memory (from Alex Graves)

- The movie that follows shows several different things:
- Row 1: This shows when the characters are recognized.
  - It never revises its output so difficult decisions are more delayed.
- Row 2: This shows the states of a subset of the memory cells.
  - Notice how they get reset when it recognizes a character.
- Row 3: This shows the writing. The net sees the x and y coordinates.
  - Optical input actually works a bit better than pen coordinates.
- Row 4: This shows the gradient backpropagated all the way to the x and y inputs from the currently most active character.
  - This lets you see which bits of the data are influencing the decision.