LING3804: Lecture Notes 2
Neural Models of Sequential Data

We've seen how neural units can be connected to each other in a static network.

But a set of neural units can also be connected to *itself* in cycles in a *recurrent* network.

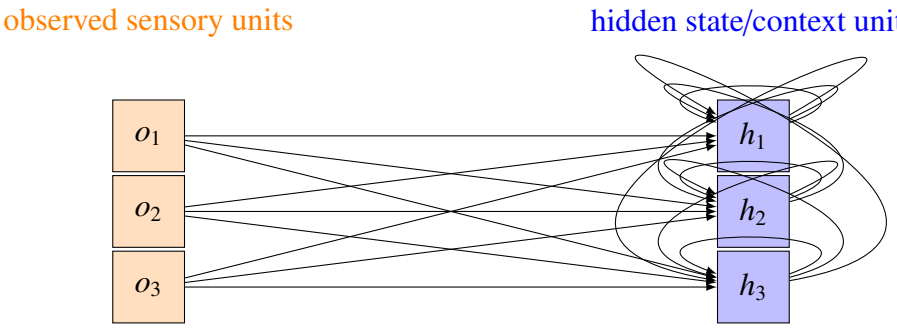
This is useful for modeling sequence data such as natural language sentences of arbitrary length!

Contents

2.1 Discriminative sequence models: finite state machines (Kleene, 1951) 1
2.2 Probability (Kolmogorov, 1933) 3
2.3 Random sampling from probability distributions 5
2.4 Generative sequence models: HMMs (Baum & Petrie, 1966) 5
2.4.1 Sampling from HMMs to generate words 7
2.4.2 Using HMMs to recognize context 8
2.5 Distributed representation of states 9

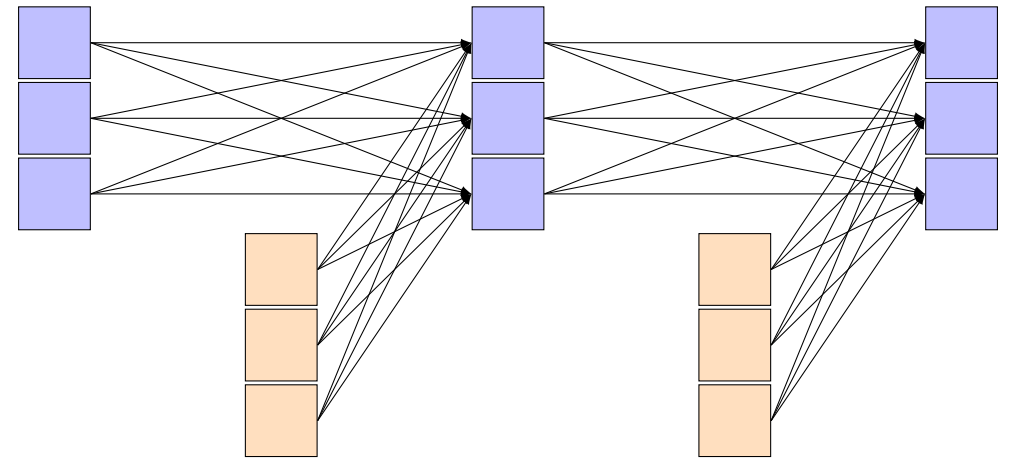
2.1 Discriminative sequence models: finite state machines (Kleene, 1951)

Here's a drawing of a set of neurons connected to themselves, and some units for observations:



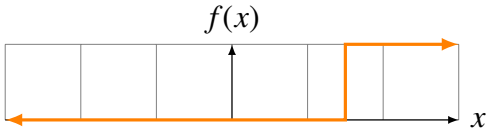
- the model is defined in terms of a 'context' vector of neural units;
- activation of the context vector defines a (mental) **state**;
- the context vector is connected to sensory units (observations);
- the context vector is also connected to *itself* at previous time step, forming a **circuit**;
- the model learns to **transition** between states by associating each previous and current state (these associations are determined by synaptic weights, as we'll see later);
- the learned transitions define a sequence of mental states for any sequence of observations.

Here's what it looks like unrolled through time:



For example, we can define a 'hotel safe' interface with a step threshold function:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 1.5 \\ 0 & \text{if } x < 1.5 \end{cases}$$



and we can give names to the stimulus and state neurons:

observed sensory units

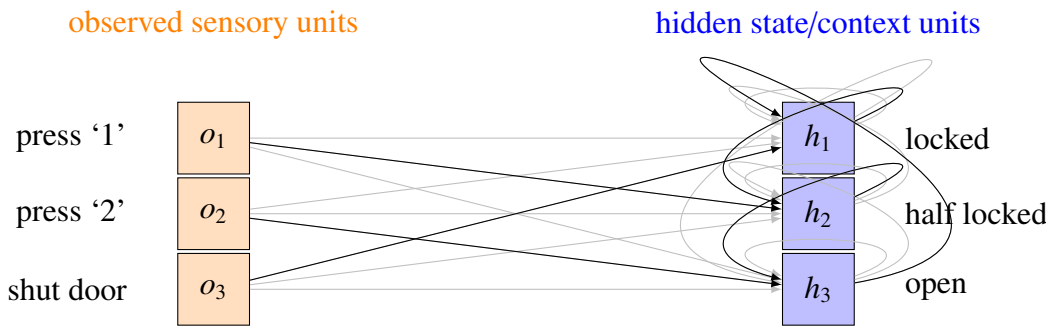
type '1'	1
type '2'	0
shut door	0

hidden state/context units

1	locked
0	half locked
0	open

so the above state is 'locked' (and nothing else) and the observation is type '1' (and nothing else).

Then we can assign weights 1 (black) or 0 (gray) to each connection:

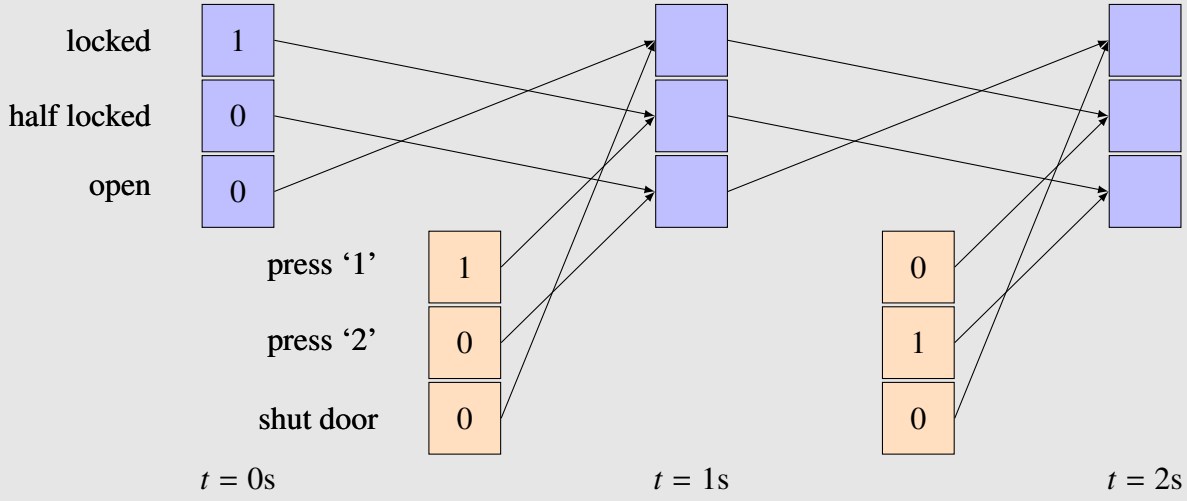


This will come open when we type '12' and then lock again when we shut it.

This is called a **finite-state automaton (FSA)**, or **finite-state machine**.

Practice 2.1:

Trace through the unrolled version of the above finite-state automaton:



Practice 2.2:

This automaton does something weird if we get the combination wrong: '2 1'. What will happen?

2.2 Probability (Kolmogorov, 1933)

Unfortunately FSAs are **discriminative**: they model hidden states given observations.

That means they can recognize or classify sequences but can't be used to generate responses.

We'll see later that will pose problems for training.

Generative models, on the other hand, model observations given hidden states.

But these models must make guesses and then reconcile them with observations.

We can use probabilities for that.

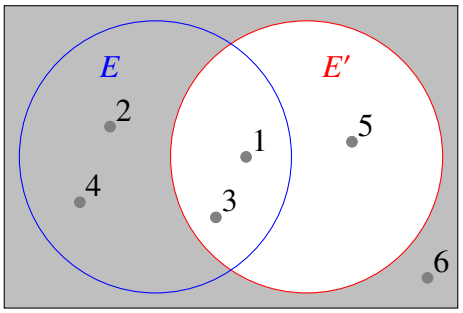
Probabilities are defined over a **sample space** of possible **outcomes** (e.g. die roll).

1. **outcomes** are the maximally specific things that happen at each trial (e.g. 1,2,3,4,5,6),
2. **events** are disjunctions of outcomes (e.g. roll odd: $\{1,3,5\}$, roll less than three: $\{1,2\}$, ...),
3. a **probability function** P maps events to real numbers between 0 and 1 (inclusive) such that:
   (a) the probability of the union of any set of disjoint events is the sum of their probabilities,
   (b) the probability of the set of all outcomes is 1.

These are the **Kolmogorov (1933) axioms of probability**.

We write **unconditional probabilities** like this: $\mathsf{P}(E) = n$, e.g. $\mathsf{P}(\{1,3,5\}) = \frac{1}{2}$, or $\mathsf{P}(e{\in}\{1,3,5\}) = \frac{1}{2}$.
Sometimes we leave out the braces for single-outcome events: $\mathsf{P}(3) = \frac{1}{6}$, or $\mathsf{P}(e{=}3) = \frac{1}{6}$.

We also define **conditional probabilities**: $\mathsf{P}(E \mid E') = \dfrac{\mathsf{P}(E \cap E')}{\mathsf{P}(E')}$, e.g. $\mathsf{P}(\{1,2,3,4\} \mid \{1,3,5\}) = \frac{2}{3}$.



We also define **joint probabilities** for combinations of events: $\mathsf{P}(\{\langle 1,2\rangle\})$, or $\mathsf{P}(e{=}1, f{=}2)$.

Practice 2.3:

Assuming two fair coins are tossed, each with a .5 probability of a heads outcome and a .5 probability of a tails outcome, what is the probability that at least one coin will come up heads?

Practice 2.4:

Assuming a fair die, solve the equation: $P(\{1,2\} \mid \{1,3,5\}) =$ ____

2.3 Random sampling from probability distributions

We can define a **random variable** over all the single-outcome events for a given sample space.

A **probability distribution** over this variable is then the set of probabilities of these events.

Once we have a probability distribution, we can **randomly sample** from it.

That means choosing a single outcome weighted by the distribution of single-outcome events.

We can view a set of neurons as a probability distribution if their activations obey the above axioms.

Sampling is then choosing a neuron and setting its activation to one and the rest to zero.

We can notate sampling using '$\sim$' or '$\rightsquigarrow$':

$$\text{From distribution: } \begin{bmatrix} .5 \\ .3 \\ .2 \end{bmatrix} \quad \text{we sample with: } P\begin{pmatrix} .5 \rightsquigarrow 1 \\ .3 \rightsquigarrow 0 \\ .2 \rightsquigarrow 0 \end{pmatrix} = .5, \quad P\begin{pmatrix} .5 \rightsquigarrow 0 \\ .3 \rightsquigarrow 1 \\ .2 \rightsquigarrow 0 \end{pmatrix} = .3, \quad P\begin{pmatrix} .5 \rightsquigarrow 0 \\ .3 \rightsquigarrow 0 \\ .2 \rightsquigarrow 1 \end{pmatrix} = .2\,.$$

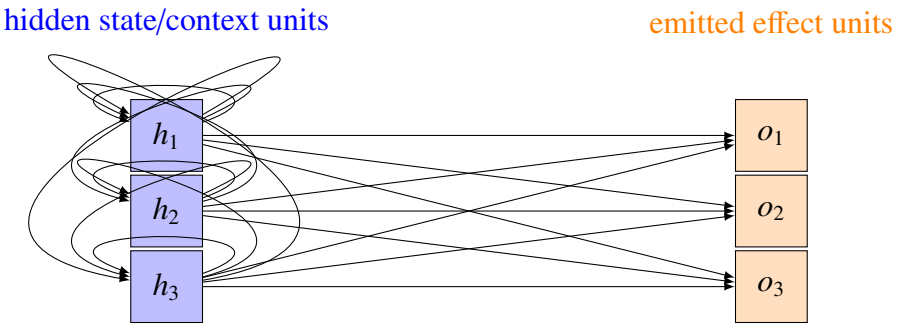
AI language models use this kind of sampling to generate words given context (prompts or queries).

2.4 Generative sequence models: HMMs (Baum & Petrie, 1966)

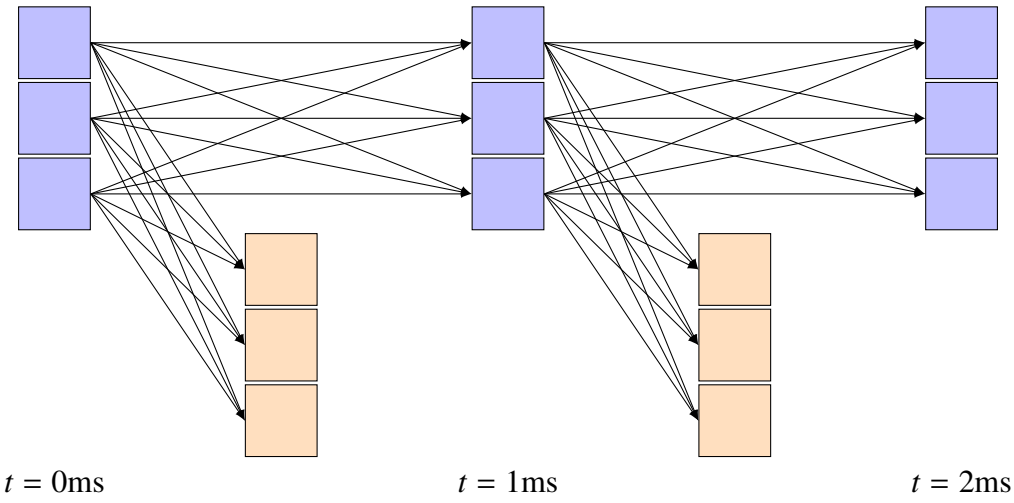
Unlike discriminative models, generative models condition observations on hidden states.

That means they can define probability distributions from which to sample output.

Here's a simple generative sequence model:

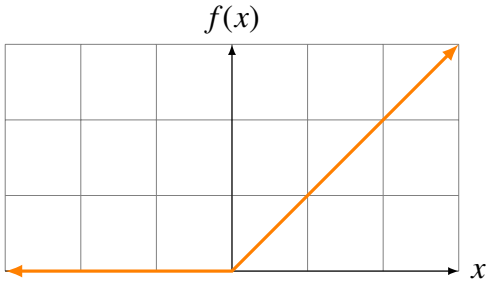


Here's what it looks like unrolled through time:

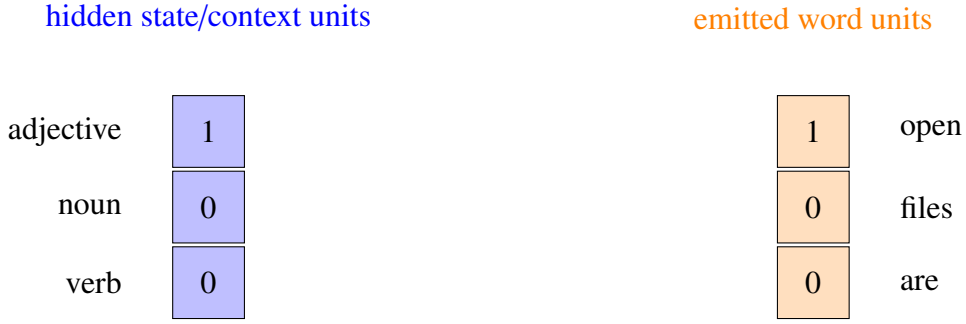


For example, we can define a language model with a 'rectified linear' threshold function:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

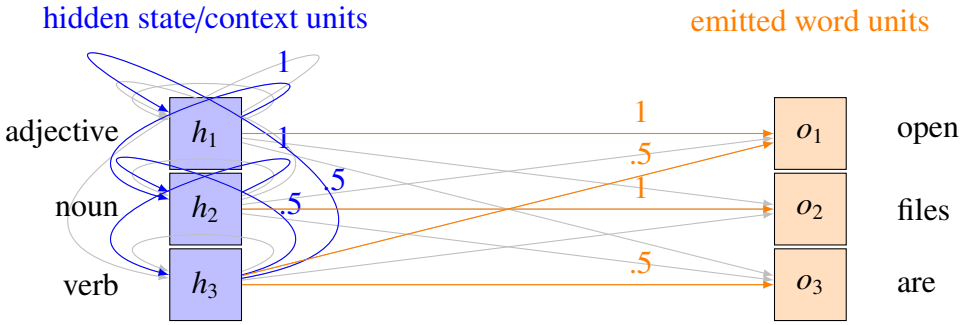


and we can give names to the state and emitted word neurons:



so the above state is 'adjective' and nothing else and the emitted word is 'open' and nothing else.

Then we can assign nonzero (colored) or zero (gray) weights to each connection:



These weights are probabilities:

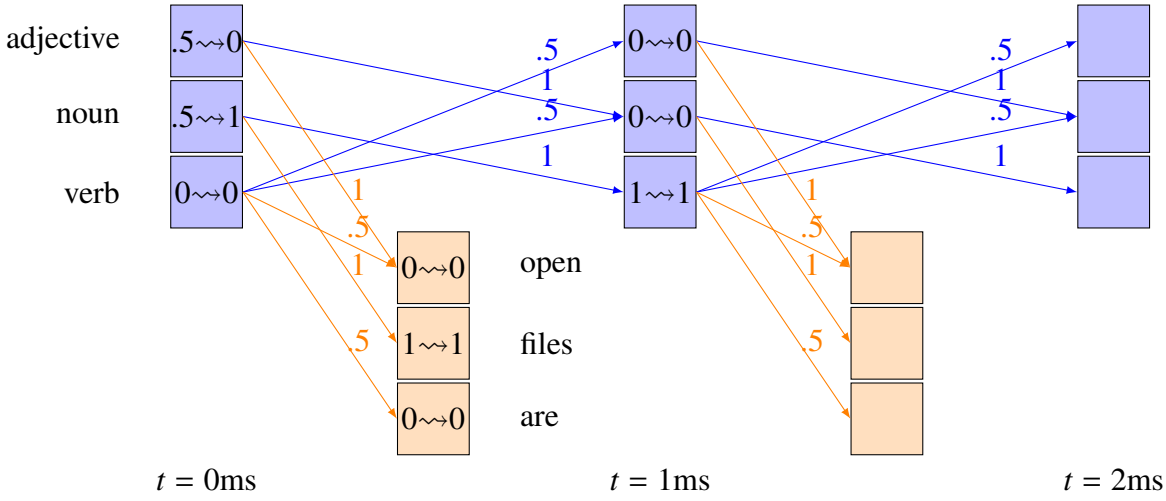
- the blue weights are probabilities of **transitioning** from one part of speech to the next,
- the orange weights are probabilities of **emitting** a word given a particular part of speech.

This cycles through parts of speech and emits word sequences: 'open files,' 'files are open,' ...

This is called a **hidden Markov model (HMM)**.

2.4.1 Sampling from HMMs to generate words

We can randomly sample from this model, as described in the previous section, to generate words:



This samples '*noun*' from an initial distribution, then generates the word '*files*', then...

Practice 2.5:

Finish sampling through the above unrolled hidden Markov model.

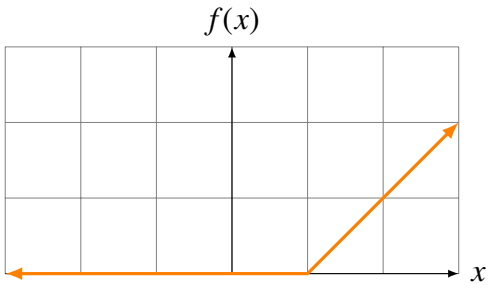
2.4.2 Using HMMs to recognize context

Now we can ***produce*** word sequences by randomly sampling hidden and observed variables.

But how can it ***recognize*** words? For that, we need to filter states via an explicit joint distribution.

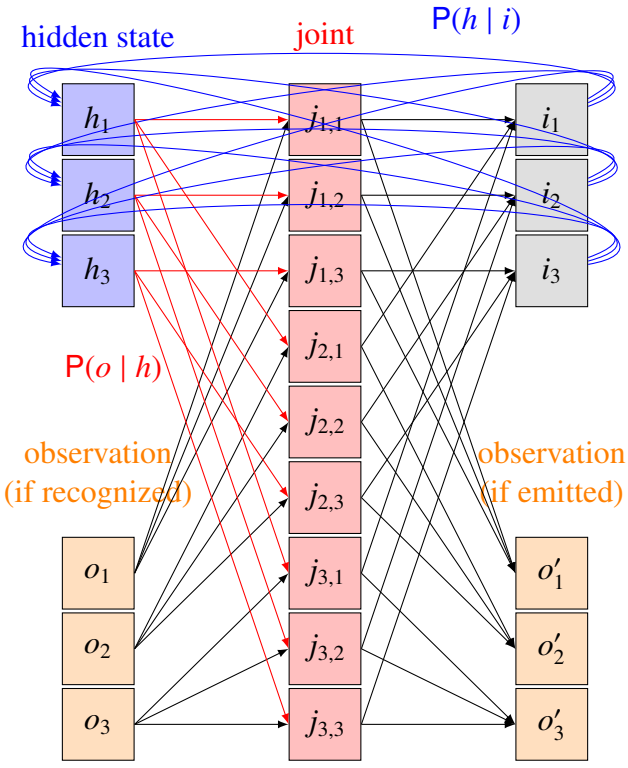
First we'll need a modified rectified linear threshold function:

$$f(x) = \begin{cases} x - 1 & \text{if } x \geq 1 \\ 0 & \text{if } x < 1 \end{cases}$$



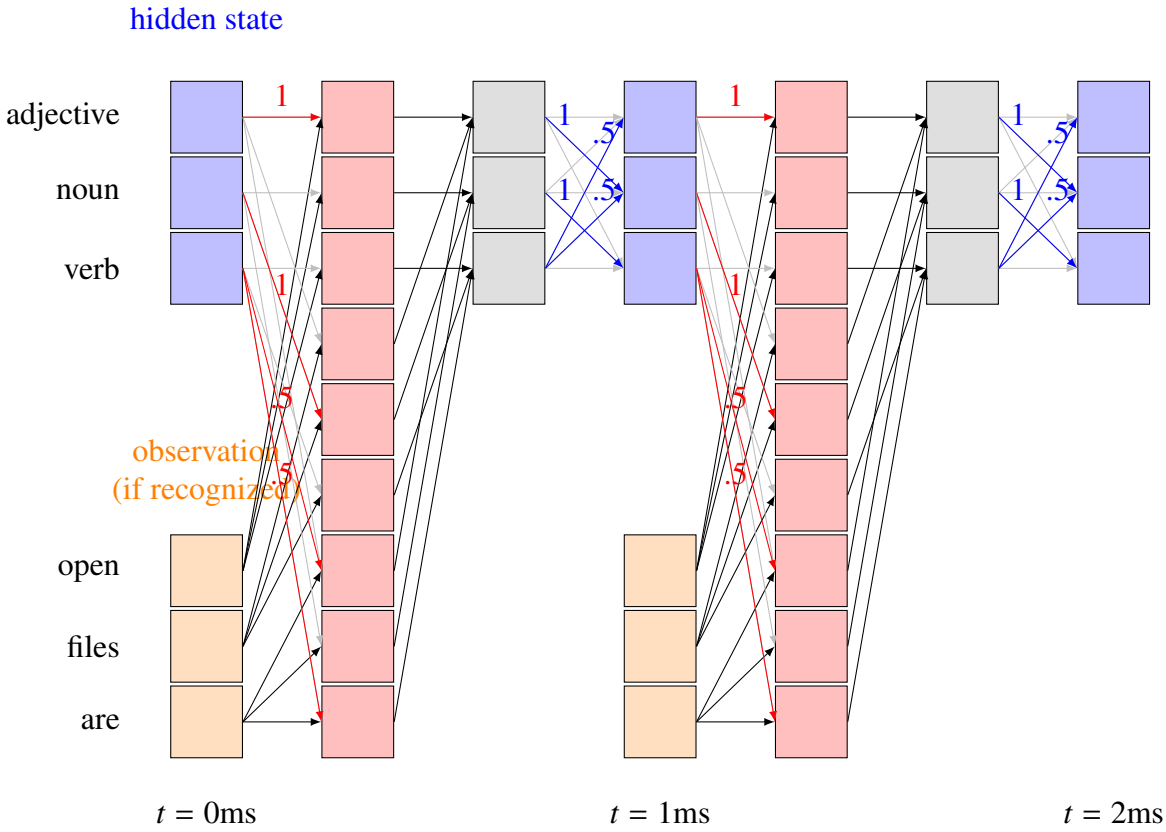
Here's a version that uses the same parameters for recognition and production

('joint' units use the modified rectified linear function, others use the standard one):



Here, the set of $j_{n,m}$ variables is a probability distribution over a joint sample space: $j_{n,m} = \langle o_n, h_m \rangle$.

Unrolled, as a recognizer, using the transition (blue) and emit (red) probabilities above:



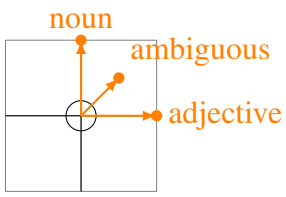
(Then to generate, we simply add each generated word to the recognizer input.)

2.5 Distributed representation of states

FSAs and HMMs are **localist** networks — they use each neural unit as a state indicator:

	adjective state	noun state	ambiguous state
adjective indicator	1	0	.5
noun indicator	0	1	.5

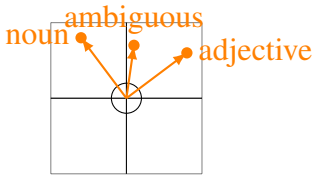
If we graph these coordinates as directed edges, we can see they are **orthogonal** (at right angles):



But learned networks are more likely to be **distributed** — having some value in each element:

	adjective state	noun state	ambiguous state
not an indicator	0.8	-0.6	0.1
not an indicator	0.6	0.8	0.7

Graphing these coordinates as directed edges (called **vectors**), we can see they are still orthogonal:



That means we can still represent probability distributions as weighted sums of these vectors.

These weighted sums are called **superpositions** and the weighted states are called **components**.

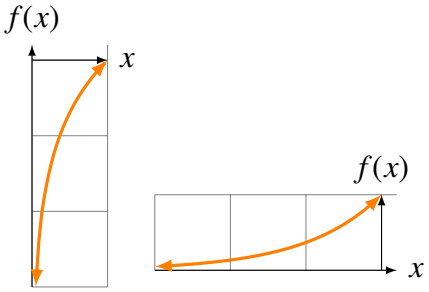
The magnitude (length) of each weighted component in the sum is proportional to its probability.

Distributed representations complicate our recognition network, but only a little bit.

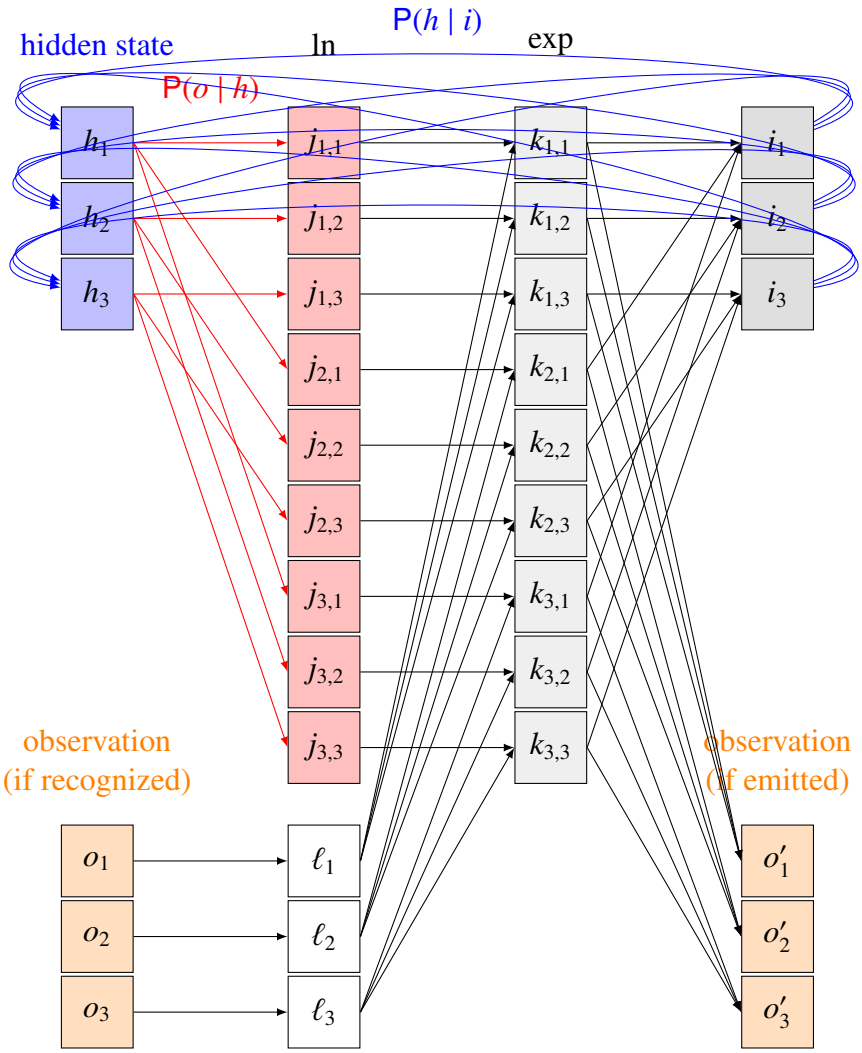
Below is a more generalized sequence model that works for distributed observations.

For this we'll need log and exponential threshold functions:

$$f(x) = \ln(x) \qquad f(x) = \exp(x)$$



Here's the network (all use linear activation, except 'ln' and 'exp' as noted):



The $k_{n,m}$ units multiply transition, emit and observation superpositions by adding their logarithms.

The resulting superpositions define correct distributions over parts of speech (i_n) or words (o'_n).

(We can get component probabilities by multiplying by state vectors to get similarities, seen later.)

References

Baum, L. E. & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37, 1554–1563.

Horton, J. C. & Adams, D. L. (2005). The cortical column: a structure without a function. *Philosophical Transactions of the Royal Society of London - Series B: Biological Sciences*, 360(1456), 837–862.

Kleene, S. C. (1951). *Representation of Events in Nerve Nets and Finite Automata*. Santa Monica, CA: RAND Corporation.

Kolmogorov, A. N. (1933). *Grundbegriffe der Wahrscheinlichkeitsrechnung*. Berlin: Springer. Second English Edition, *Foundations of Probability* 1950, published by Chelsea, New York.