HammingNN

Neural network based nearest neighbour pattern classifier

Outline

- Introduction
- Biological neuron basics
- Application to musical interval recognition
- Time-based (serial) pattern classification
- Weeding out noisy attributes
- Implications of converting continuous attributes into categorical attributes
- Next steps

Introduction

- About myself
- Why artificial neural networks
- Simple perceptrons are too limited
- ANN research: biologically implausible paradigms

I started my professional life as an electrical engineer; in 4th year at the University of Waterloo, we built logic circuits such as AND, OR, and XOR gates out of transistors and other components; we built a rudimentary computer using commercial TTL chips.
After working as an engineer for 7 years, mostly in computer systems, I went to medical school, and eventually specialized in psychiatry. While studying neurophysiology as a medical student, however, it dawned on me that it should be possible to design and fabricate functional neuron systems and networks as silicon chips (ie, integrated circuits).
I started reading about artificial neural networks, and was initially heartened by the earliest attempts, such as the McCulloch-Pitts neuron, in 1943, and the Perceptron of Rosenblatt, in 1962. But in 1969 Minsky and Papert published a book with the title “Perceptrons” in which they demonstrated mathematically that the simple perceptron was unable to deal with a number of important cases in pattern recognition. This book had a major influence on research in artificial neural networks: funding for work on perceptrons dried up, and a large number of complicated paradigms were introduced, such as backpropagation, Hopfield networks, adaptive resonance theory, and so on. To my way of thinking, these paradigms were far removed from biological plausibility. For example, backpropagation networks often required thousands of passes through the training data to learn with acceptable error rates.

New Insights

- Recognition of temporal patterns
- Forcing inputs for classical conditioning

While searching through the stacks in the medical library at McGill, I came across a book by a researcher in the department of anatomy at University College in London. I learned that by feeding the outputs of a group of neurons back to the inputs, one could process temporal patterns with simple perceptron-like topologies. I also learned that, if you provided a separate set of inputs which could be used to force the output neurons to fire, then you could build a simple network which could learn with very little training using classical conditioning paradigms; perhaps even one-trial learning would be possible! And this type of learning could be accomplished using the Hebbian rule for adjusting synaptic weights.

Temporal patterns

- For music or speech recognition
- A way of storing properties about objects
- Makes the perceptron into a finite state machine
- Simplifies machine vision

Classification of time-based patterns such as speech or music recognition would be the obvious candidates for neural networks with their outputs fed back to their inputs. However, the anatomy researcher's book also made a good case that we store information about the properties of things as temporal patterns. For example, a baby learns about the hardness of a wood block or the softness of a blanket by chomping down on it with its gums and storing the temporal pattern of a certain amount of force exerted by its jaws and the resulting pressure sensations on its gums.

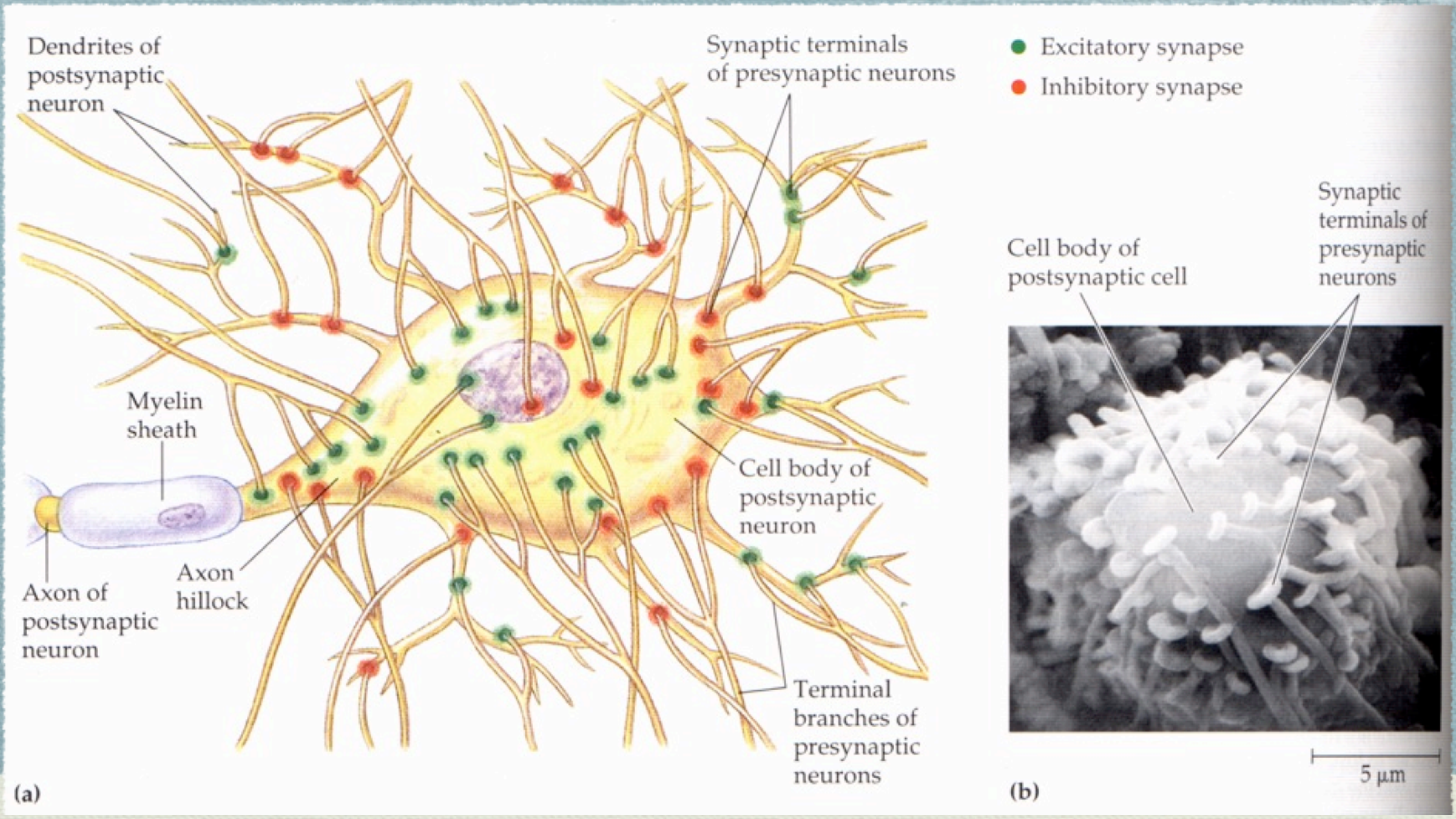
What I began to realize, is that feeding back outputs to inputs means in essence that the neural network can do pattern classification taking into account not only the current state of the inputs, but also previous states. This makes it a finite state machine, and overcomes the objections that had been raised by Minsky and Papert to the generalisability of the perceptron.

Finally, by thinking of vision as temporal pattern classification, we can simplify the problem of machine vision enormously.

Early development

- Programming in Hypercard and StarLogo
- MOPS
- MacForth
- Python

I made a number of attempts to program these ideas, starting with programming languages which allowed for individual entities to operate independently. I tried Hypercard, then Starlogo, then an object-oriented Forth called MOPS. But I made little real progress until I switched to MacForth. I was able to successfully demonstrate temporal pattern classification with this programming environment.
Eventually, though, computer and operating system upgrades led to problems with the MacForth implementation. When I was unable to contact the MacForth developer for support, I made the decision to reprogram in python. This has been extremely productive for static, non-temporal pattern classification, and I am fairly close to having something that could be used for general pattern classification tasks.
OK, that takes care of the introduction.

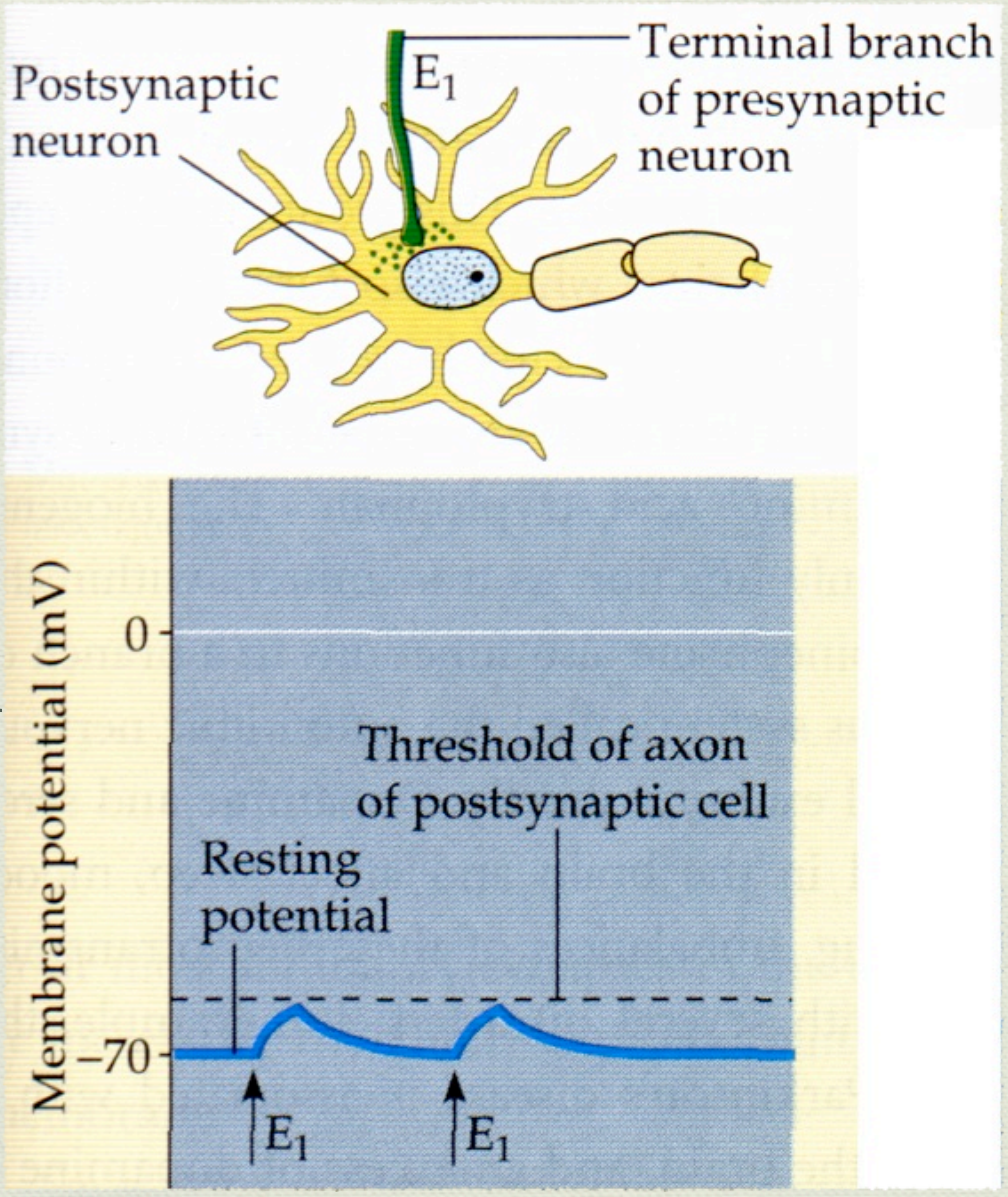


Dendrites of postsynaptic neuron
Synaptic terminals of presynaptic neurons
● Excitatory synapse
● Inhibitory synapse
Myelin sheath
Axon of postsynaptic neuron
Axon hillock
Cell body of postsynaptic neuron
Terminal branches of presynaptic neurons
(a)
Cell body of postsynaptic cell
Synaptic terminals of presynaptic neurons
5 μm
(b)

Biological neuron basics

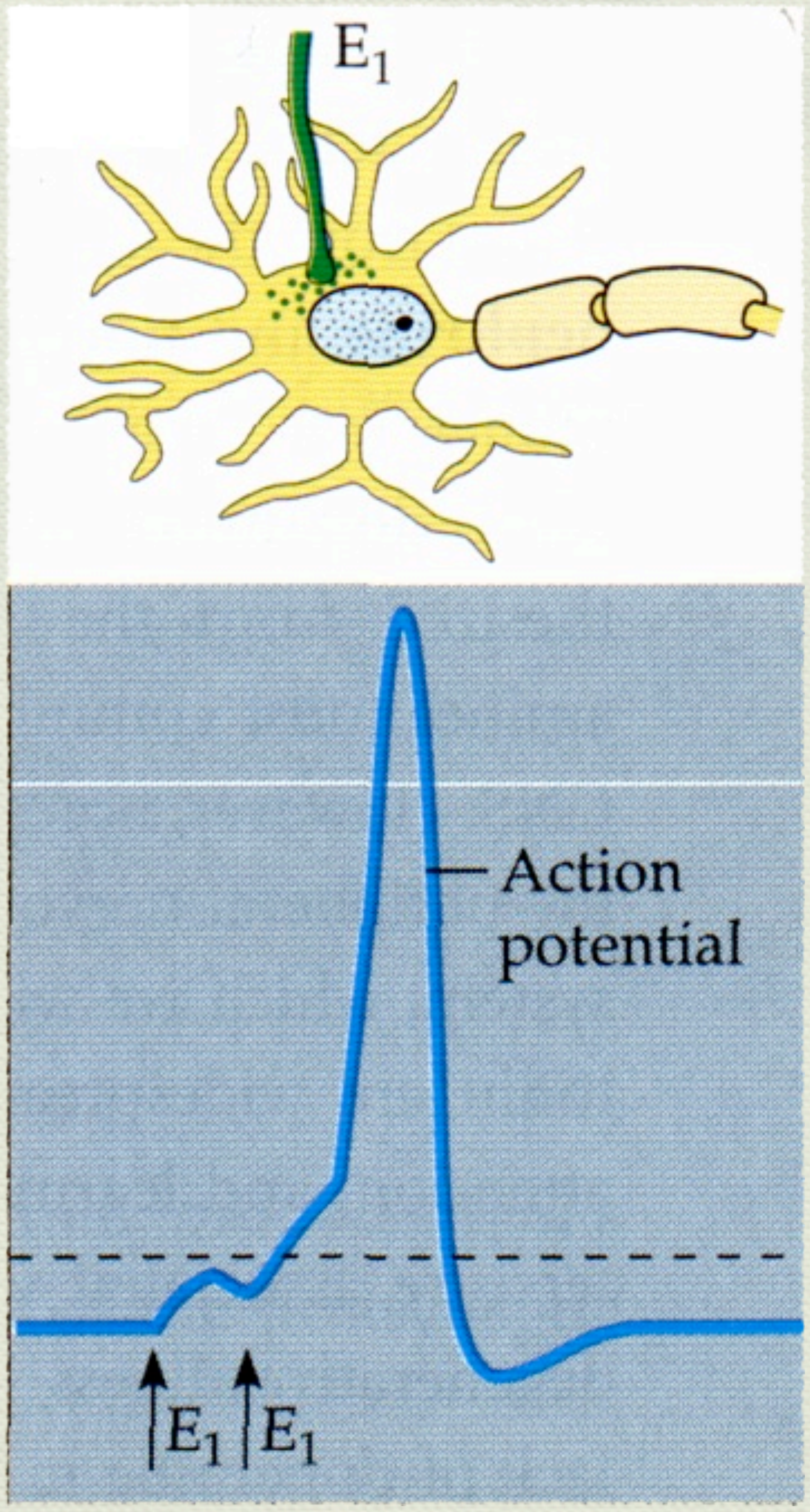
Now let’s get down to some details.
I realize that I am talking to experts in neurobiology, so please forgive me for going over stuff you are familiar with.
Here is an illustration from a textbook showing a postsynaptic neuron with terminal branches from presynaptic neurons terminating in synapses on its body and dendrites.

Summation of post-synaptic potentials



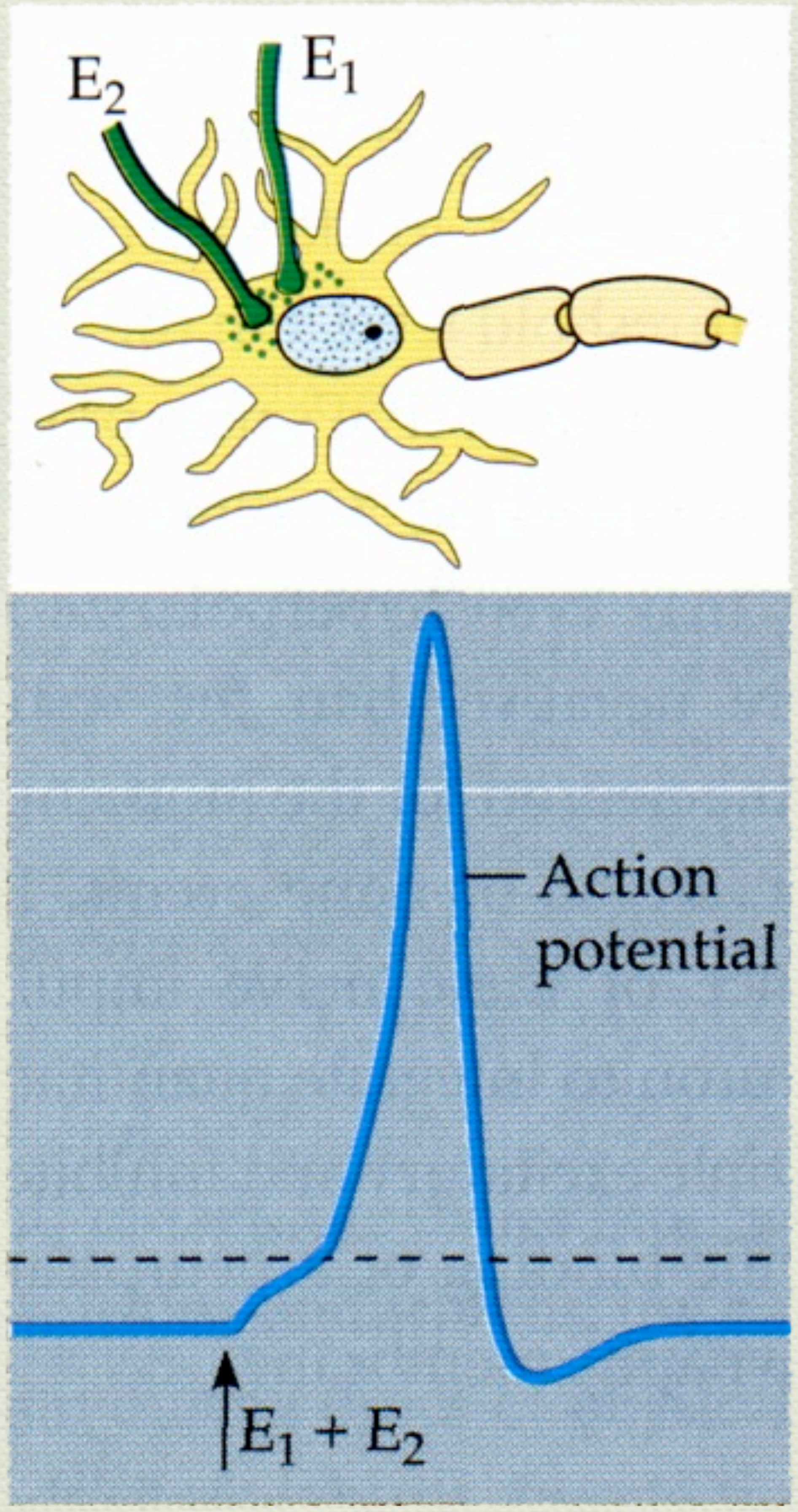
This illustration shows what happens to the membrane potential of the postsynaptic neuron near a synapse when that synapse is repeatedly fired by action potentials from the presynaptic neuron, but at a slow rate. Typically, the synaptic weight will not be sufficient for the depolarization to trigger an action potential.

Temporal summation



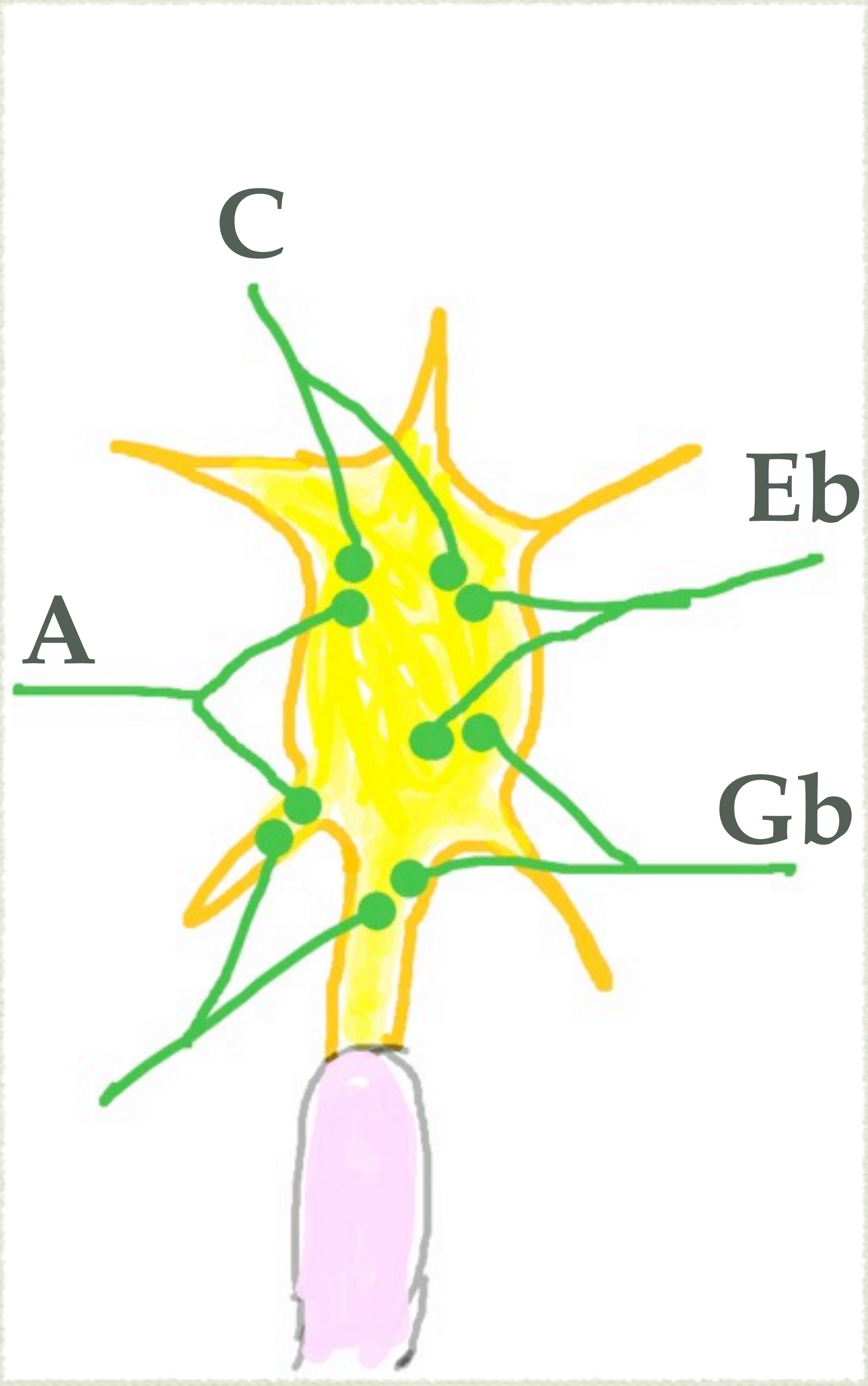
This is what happens when the successive pulses are closer together in time. Now the depolarizations when added together, exceed the threshold of the postsynaptic neuron and an action potential results.

Spatial summation



If the synapses from two different presynaptic neurons are located close together physically on the cell body or on a dendrite of a postsynaptic neuron, then an action potential can be generated if both neurons fire at about the same time. This represents a logical AND function.
Because the two presynaptic neurons need to fire at about the same time, this circuit is also a coincidence detector.
If the weights for each synapse were high enough, then either presynaptic neuron by itself could trigger an action potential. This would be a logical OR gate.
Finally, just to recall that inhibitory post-synaptic potentials will act so as to reduce the likelihood of the threshold being exceeded and an action potential occurring.

Interval recognition in music

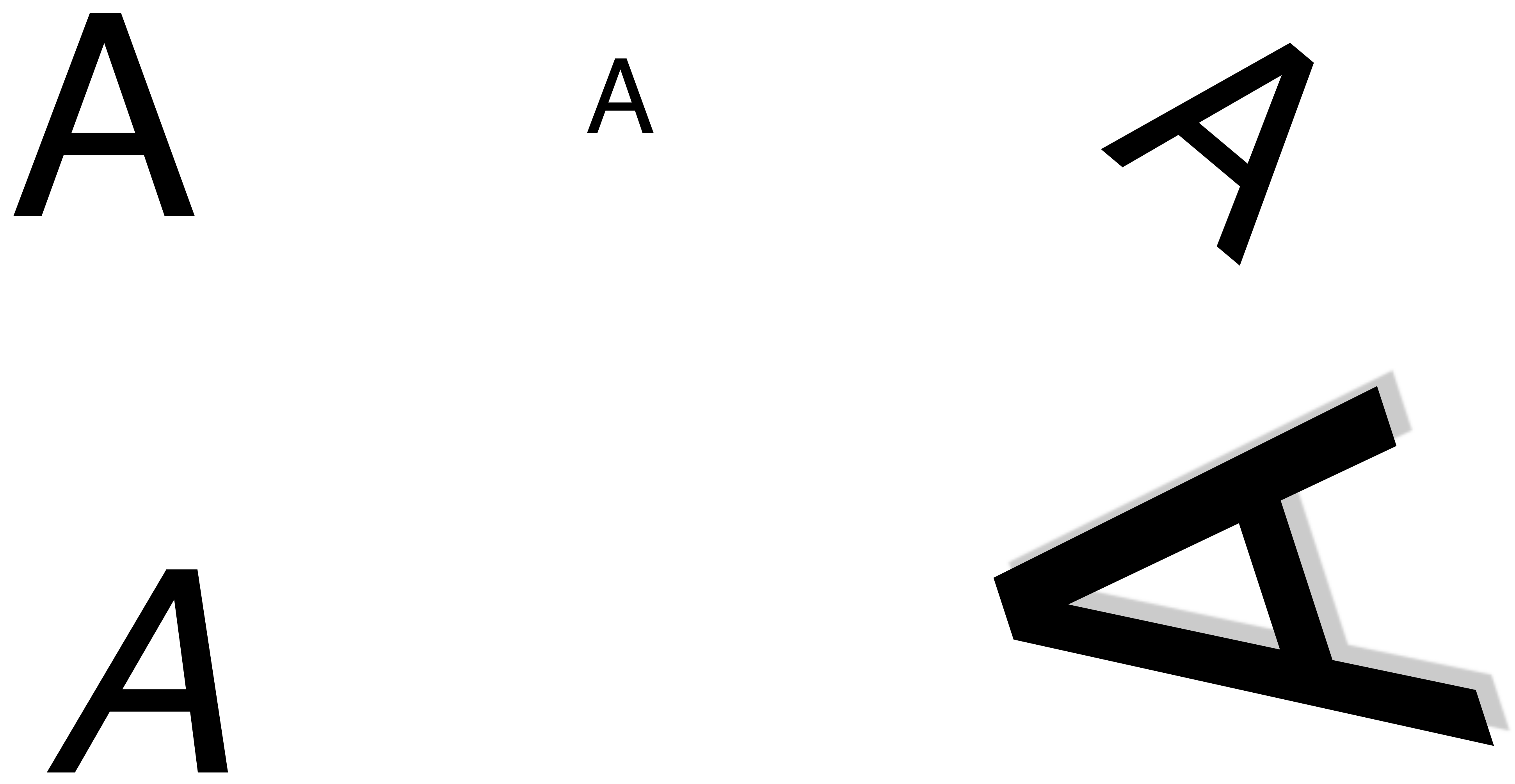


Let’s apply this to a basic problem in music recognition, for example, detecting a particular interval. Suppose that we have fibers impinging on a neuron which is supposed to detect minor third intervals. These fibers, from the auditory nerve and possibly from other auditory processing areas of the brain, carry information about the intensity and frequency of sound signals transduced by the cochlea. I’m making the assumption here that all these fibers carry spatially encoded information, and that the arrival of spikes at almost the same time on different fibers is important, not the frequency of the spikes or the interspike interval. For all the different minor third intervals (ie starting on different pitches) the set of synapses relevant to the interval for a given pitch will be located close to each other and far enough away from the sets of synapses for other pitches to avoid interference. Also for a given set of synapses, the weights of each synapse will need to be such that the intensity of each of the partials characteristic of a minor third interval are in the correct proportions to summate both temporally and spatially to produce an action potential.
Of course, this is an enormous oversimplification. But I hope that it demonstrates the principle.
When the notes of an interval are separated in time, as in a melody, we need a paradigm which takes into account both place and time information. The simplest would be to take our coincidence detector and add a time delay to one of the synaptic inputs. This could be done biologically with an additional neuron or string of neurons.

Time-based pattern classification

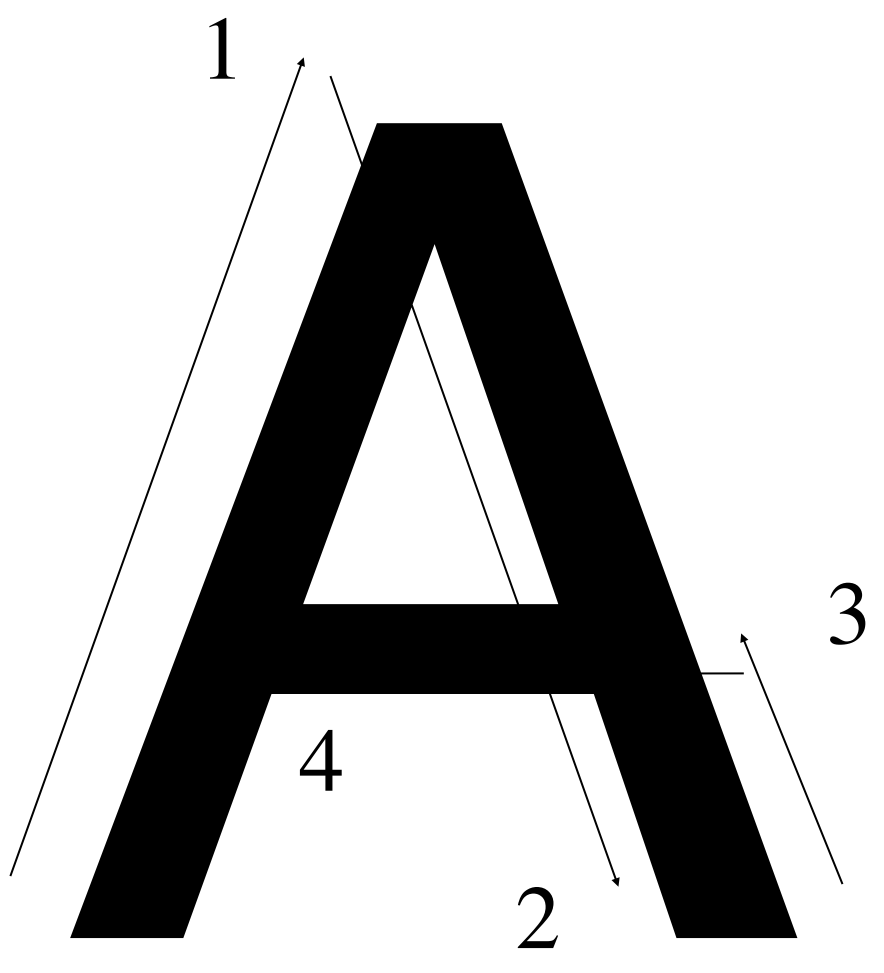
I mentioned earlier that feeding the outputs of a group of neurons back as inputs turns the group of neurons into a state machine, and allows for the classification of time-based, or serial patterns. Music and speech are, of course, time-based patterns. But it may be that our brains store much of what we know about our environment as serial patterns. Think of our sense of touch: when we apply a certain contact force to something (quantified by spindle receptors in our muscles and Golgi organ receptors in tendons), there is a subsequent signal from pressure receptors in our skin.
I think that there is good evidence that a lot of visual pattern recognition may also use serial patterns.

Problems with Spatial Pattern Recognition



Let's look at a typical machine vision problem, recognition of printed text. If we use a neural network to recognize individual letters coded as a pixel patterns, it works very well when all the letters are the same style, boldness, size, and orientation. But if there are large differences in size or rotation of the image, pixel-based recognition systems typically have to do a lot of image pre-processing to normalize these parameters.

Problems with Spatial Pattern Recognition - 2



- 1. Follow edge for distance *x*
- 2. CCW angle 30°; follow edge for distance *x*
- 3. Angle 180°; follow edge for distance *x/3*
- 4. CW angle 120°; follow edge for distance *x/2*

But there are other ways of describing the text characters. Suppose our wetware can detect edges. This letter A can then be described as a set of vectors, each edge having a length and a rotation.
Now if we describe the length of the vectors not in absolute units, but in terms of the ratio of a given vector's length with the length of the first vector, and the same for orientation, that is, as an angular displacement from the first vector.
The letter is now described as a sequence of relative vectors. There is no necessity to make expensive transformations to accommodate different sizes or orientations of letters prior to recognition.

Why serial patterns?

- Adaptive filters
- Control of robotic movement sequences
- Collision avoidance

A neural network that works well with serial patterns has all sorts of potential uses. Think of adaptive filters, for example in telecommunications. A sequence of outputs from such a network can be used to control robots.
Here's an interesting possibility: collision avoidance.
How can you tell, when you're on a boat, if you are on a collision course with another boat? If both you and the other boat are holding a steady course, then you know you are going to collide if the relative bearing from you to the other boat remains the same, and it's getting bigger.
It's highly likely that a housefly is so good at avoiding a fly swatter by using the same principle. Its compound eye is very good at detecting the direction from which light is coming. If it picks up something with one of the cells of its compound eye, and concentric cells subsequently become activated, that means the relative bearing remains the same while the object is getting closer. A collision course! For aircraft, phased array radar can provide similar directional information.
I'm sure you can come up with lots of other interesting ways to use neural networks with serial patterns.

Improving performance

- The dimensionality curse

My neural network paradigm is good at pattern classification of serial patterns; I tried this out several years ago with a macForth version.
More recently, my development efforts have gone into tweaking the performance of my paradigm with static patterns. I've had some discussions with Yoshua Bengio, a researcher at the University of Montreal that you probably know. He pointed out that a big problem in pattern classification has to do with what is known as the "dimensionality curse". Now I don't pretend to understand what this means mathematically, but basically when you have a lot of attributes or dimensions, the accuracy and/or the performance of nearest neighbour classifiers can take a big hit.
So I was interested in finding ways to reduce the number of attributes. By experimenting, I came up with a simple way to rank-order the attributes in a dataset according to how much they contributed to differentiating between classes. With this, I could simply ignore the attributes which didn't contribute a lot, which meant that the whole classification process could be speeded up and use a lot less resources.

The MNIST dataset

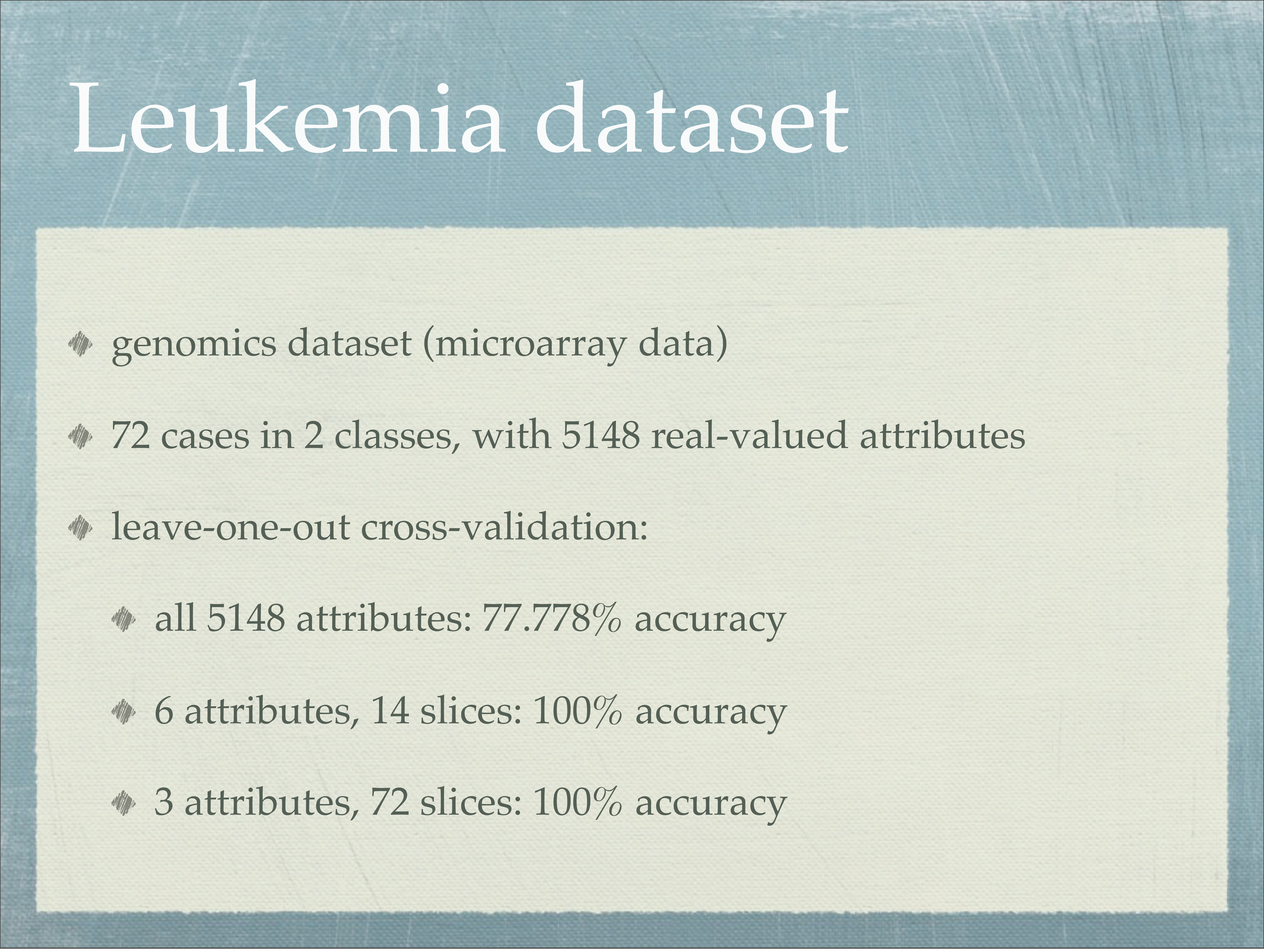
- Images of handwritten numerals, 0 thru 9
- Each image is 28x28 pixels, 255 grayscale levels
- Training set: 60,000 images
- Test set: 10,000 images
- With contrast enhancement to use black-white only:
  - for all 28x28 = 784 pixels: 96.60% accuracy
  - when using only 313 pixels: 96.75% accuracy

When I applied this to the MNIST dataset of handwritten numerals, for example, I found that by reducing the gray-scale from 255 levels to just two (black or white), and by using the 313 out of 784 pixels which differentiated the best between the 10 numerals, I was able to achieve higher levels of accuracy (with less computational resources) than when using the original dataset.

Continuous vs categorical attributes

Did I mention that this neural network classifier uses Hamming distance to calculate nearest neighbours? Hamming distance is the separation between two binary numbers, ie numbers expressed only in 0s and 1s.
To convert categorical attributes to binary numbers is easy; assign one bit to each category. But there is an even better way: just use one bit for each class. Here, we use a pre-processing stage in which we set up lists of values for each categorical attribute, one list per class. For a given case, if the value for the attribute is found in one of the lists, set the bit corresponding to that class. This approach functions like contrast enhancement to improve accuracy.
Continuous attributes are a bit trickier; my initial approach had been to calculate the span (ie subtract the minimum value from the maximum) and divide the span into a number of equally sized bins. The number of bins is an adjustable parameter, but for most of the datasets I used, accuracy was optimum when the number of bins was maybe 8 or 16.

But I then had the insight, what if we treat continuous attributes like categorical attributes, ie instead of one bit per bin, we would use one bit per class, and create cut-off points to maximize the separation between classes?



Leukemia dataset

- genomics dataset (microarray data)
- 72 cases in 2 classes, with 5148 real-valued attributes
- leave-one-out cross-validation:
  - all 5148 attributes: 77.778% accuracy
  - 6 attributes, 14 slices: 100% accuracy
  - 3 attributes, 72 slices: 100% accuracy

But I then had the insight, what if we treat continuous attributes like categorical attributes, ie instead of one bit per bin, we would use one bit per class, and create cut-off points to maximize the separation between classes? This slide shows results with one dataset.

But there is a problem. When I make the number of slices high enough, the preprocessing starts to become part of the training, and since I have been doing the preprocessing on the whole dataset, this amounts to partially training the neural network on the cases which I later test on. This contributes to the high accuracy. The answer is to do the partitioning for the cross-validation prior to the pre-processing step. This would reduce accuracy, I don't know by how much.
It would require a major rewrite of the software, though. And that's what I'm looking at doing right now.

Next steps

- Roads to follow:
  - Commercial product: classifier for static problems
  - Cognitive neuroscience research: explore insights into biological neural processes
  - Development of firmware approaches to time-based pattern recognition