18 Neurobiology; Variations on Neural Networks

NEUROBIOLOGY

[The field of artificial intelligence started with some wrong premises. The early AI researchers attacked problems like chess and theorem proving, because they thought those exemplified the essence of intelligence. They didn't pay much attention at first to problems like vision and speech understanding. Any four-year-old can do those things, and so researchers underestimated their difficulty.]
[Today, we know better. Computers can effortlessly beat four-year-olds at chess, but they still can't play with toys well. We've come to realize that rule-based symbol manipulation is not the primary defining mark of intelligence. Even rats do computations that we're hard pressed to match with our computers. We've also come to realize that these are different classes of problems that require very different styles of computation. Brains and computers have very different strengths and weaknesses, which reflect their different computing styles.]

[Neural networks are partly inspired by the workings of actual brains. Let's take a look at a few things we know about biological neurons, and contrast them with both neural nets and traditional computation.]

- CPUs: largely sequential, nanosecond gates, fragile if gate fails
  superior for arithmetic, logical rules, perfect key-based memory
- Brains: very parallel, millisecond neurons, fault-tolerant

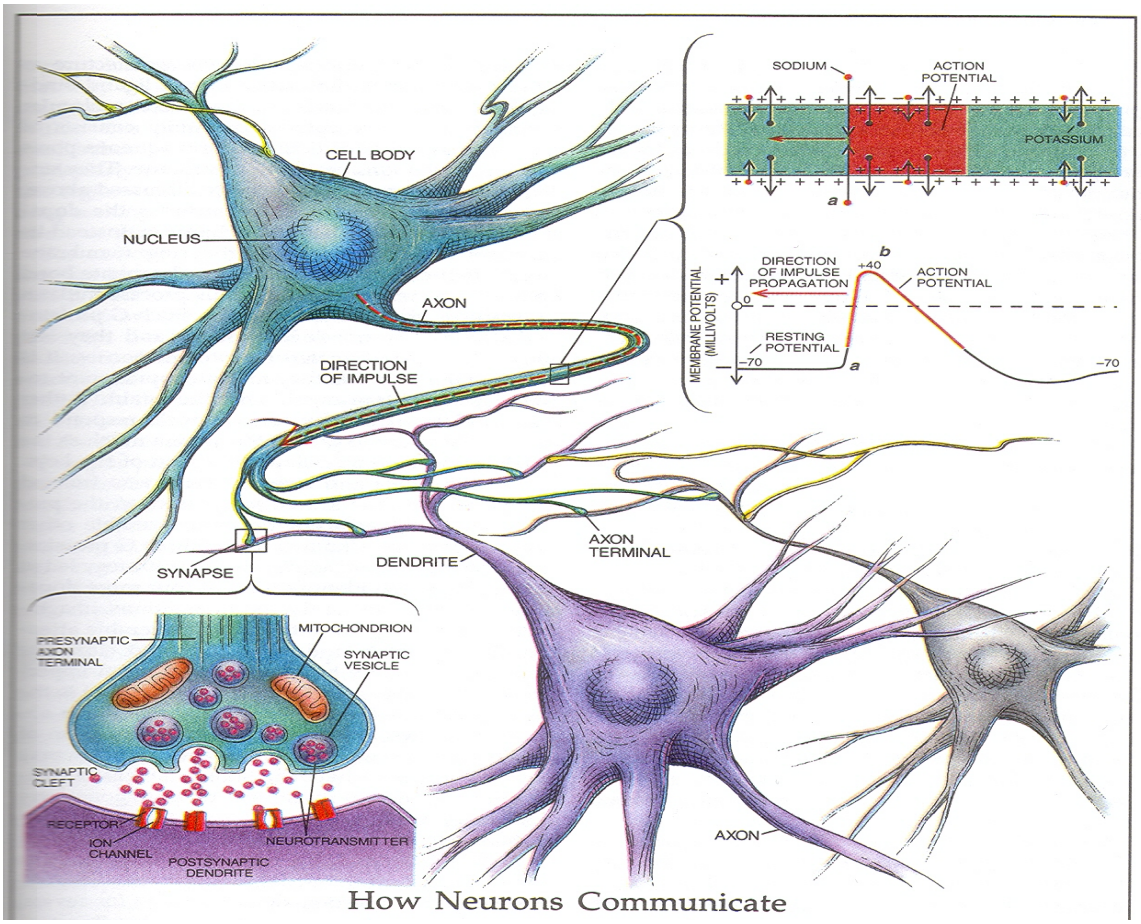
  [Neurons are continually dying. You've probably lost a few since this lecture started. But you probably didn't notice. And that's interesting, because it points out that our memories are stored in our brains in a diffuse representation. There is no one neuron whose death will make you forget that $2 + 2 = 4$. Artificial neural nets often share that resilience. Brains and neural nets seem to superpose memories on top of each other, all stored together in the same weights, sort of like a hologram.]
  [In the 1920's, the psychologist Karl Lashley conducted experiments to identify where in the brain memories are stored. He trained rats to run a maze, and then made lesions in different parts of the cerebral cortex, trying to erase the memory trace. Lashley failed; his rats could still find their way through the maze, no matter where he put lesions. He concluded that memories are not stored in any one area of the brain, but are distributed throughout it. Neural networks, properly trained, can duplicate this property.]

  superior for vision, speech, associative memory

  [By "associative memory," I mean noticing connections between things. One thing our brains are very good at is retrieving a pattern if we specify only a portion of the pattern.]

[It's impressive that even though a neuron needs a few milliseconds to transmit information to the next neurons downstream, we can perform very complex tasks like interpreting a visual scene in a tenth of a second. This is possible because neurons run in parallel, but also because of their computation style.]

[Neural nets try to emulate the parallel, associative thinking style of brains, and they are among the best techniques we have for many fuzzy problems, including some problems in vision and speech. Not coincidentally, neural nets are also inferior at many traditional computer tasks such as multiplying 10-digit numbers or compiling source code.]



neurons.pdf

- Neuron: A cell in brain/nervous system for thinking/communication
- Action potential or spike: An electrochemical impulse fired by a neuron to communicate w/other neurons
- Axon: The limb(s) along which the action potential propagates; "output"
  [Most axons branch out eventually, sometimes profusely near their ends.]
  [It turns out that giant squids have a very large axon they use for fast water jet propulsion. The mathematics of action potentials was first characterized in these giant squid axons, and that work won a Nobel Prize in Physiology in 1963.]
- Dendrite: Smaller limbs by which neuron receives info; "input"
- Synapse: Connection from one neuron's axon to another's dendrite
  [Some synapses connect axons to muscles or glands.]
- Neurotransmitter: Chemical released by axon terminal to stimulate dendrite

[When an action potential reaches an axon terminal, it causes tiny containers of neurotransmitter, called vesicles, to empty their contents into the space where the axon terminal meets another neuron's dendrite. That space is called the synaptic cleft. The neurotransmitters bind to receptors on the dendrite and influence the next neuron's body voltage. This sounds incredibly slow, but it all happens in 1 to 5 milliseconds.]

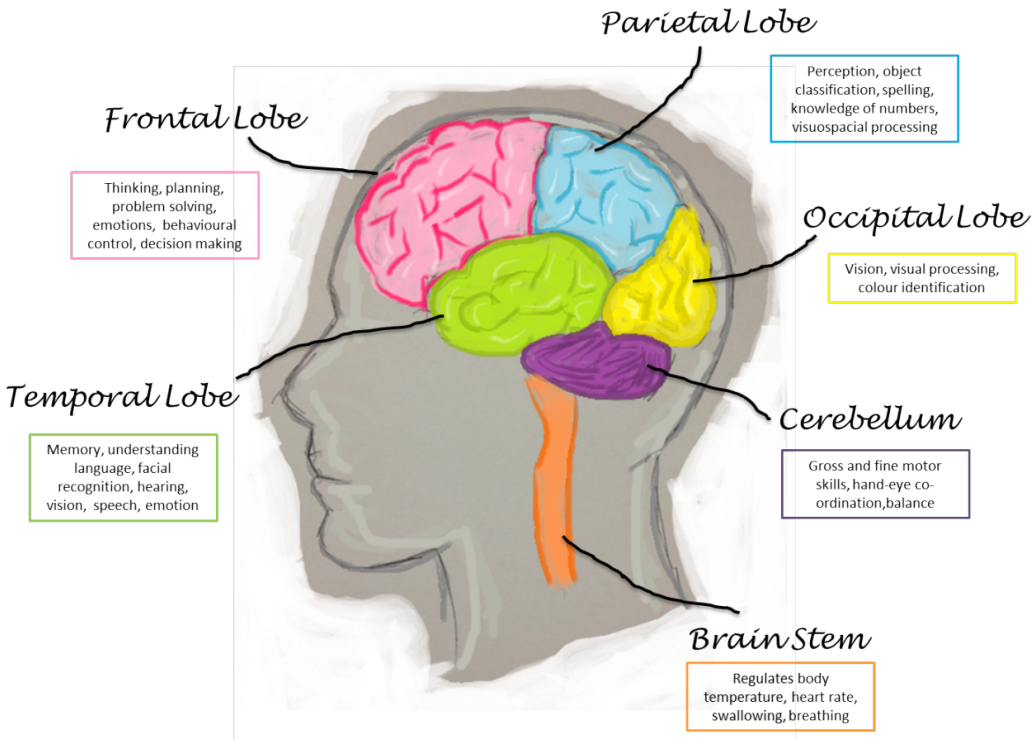
You have about 10^{11} neurons, each with about 10^4 synapses.
[ ]

Analogies: [between artificial neural networks and brains]

- Output of unit ↔ firing rate of neuron
  [An action potential is "all or nothing"—all action potentials have the same shape and size. The output of a neuron is not signified by voltage like the output of a transistor. The output of a neuron is the frequency at which it fires. Some neurons can fire at nearly 1,000 times a second, which you might think of as a strong "1" output. Conversely, some types of neurons can go for minutes without firing. But some types of neurons never stop firing, and for those you might interpret a firing rate of 10 times per second as a "0".]
- Weight of connection ↔ synapse strength
- Positive weight ↔ excitatory neurotransmitter (e.g., glutamine)
- Negative weight ↔ inhibitory neurotransmitter (e.g., GABA, glycine) [Gamma aminobutyric acid.]
  [A typical neuron is either excitatory at all its axon terminals, or inhibitory at all its terminals. It can't switch from one to the other. Artificial neural nets have an advantage here.]
- Linear combo of inputs ↔ summation
  [A neuron fires when the sum of its inputs, integrated over time, reaches a high enough voltage. However, the neuron body voltage also decays slowly with time, so if the action potentials are coming in slowly enough, the neuron might not fire at all.]
- Logistic/sigmoid fn ↔ firing rate saturation
  [A neuron can't fire more than 1,000 times a second, nor less than zero times a second. This limits its ability to overpower downstream neurons. We accomplish the same thing with the sigmoid function.]
- Weight change/learning ↔ synaptic plasticity
  [Donald] Hebb's rule (1949): "Cells that fire together, wire together."
  [This doesn't mean that the cells have to fire at exactly the same time. But if one cell's firing tends to make another cell fire more often, their excitatory synaptic connection tends to grow stronger. There's a reverse rule for inhibitory connections. And there are ways for neurons that aren't even connected to grow connections.]
  [There are simple computer learning algorithms based on Hebb's rule. They can work, but they're generally not nearly as fast or effective as backpropagation.]

[Backpropagation is one part of artificial neural networks for which the analogy is doubtful. There have been some proposals that the brain might do something vaguely like backpropagation, but it seems tenuous.]

[The brain is very modular.]



brain.png

[(The following items are all spoken, not written . . .)

- The part of our brain we think of as most characteristically human is the cerebral cortex, the seat of self-awareness, language, and abstract thinking.

But the brain has a lot of other parts that take the load off the cortex.

- Our brain stem regulates functions like heartbeat, breathing, and sleep.
- Our cerebellum governs fine coordination of motor skills. When we talk about "muscle memory," much of that is in the cerebellum, and it saves us from having to consciously think about how to walk or talk or brush our teeth, so the cortex can focus on where to walk and what to say.
- Our limbic system is the center of emotion and motivation, and as such, it makes a lot of the big decisions. I sometimes think that 90% of the job of our cerebral cortex is to rationalize decisions that have already been made by the limbic system. [ ]
- Our visual cortex (in the occipital lobe) performs a lot of processing on the input from your eyes to change it into a more useful form. Neuroscientists and computer scientists are particularly interested in the visual cortex for several reasons. Vision is an important problem for computers. The visual cortex is one of the easier parts of the brain to study in lab animals. The visual cortex is largely a feedforward network with few neurons going backward, so it's easier for us to train computers to behave like the visual cortex.]

[Although the brain has lots of specialized modules, one thing that's interesting about the frontal lobe is that it seems to be made of general-purpose neural tissue that looks more or less the same everywhere, at least before it's trained. If you experience damage to part of the frontal lobe early enough in life, while your brain is still growing, the functions will just relocate to a different part of the frontal lobe, and you'll probably never notice the difference.]

[As computer scientists, our primary motivation for studying neurology is to try to get clues about how we can get computers to do tasks that humans are good at. But neurologists and psychologists have also been part of the study of neural nets from the very beginning. Their motivations are scientific: they're curious how humans think, and how we can do the things we do.]

NEURAL NET VARIATIONS

[I want to show you a few basic variations on the standard neural network I showed you last class, and how some of these variations change backpropagation.]

Regression: usually linear output unit(s)—omit sigmoid fn.
[If you make that change, the gradient changes too, and you have to change the derivation of backprop. The derivation gets simpler, so I'll leave it as an exercise.]

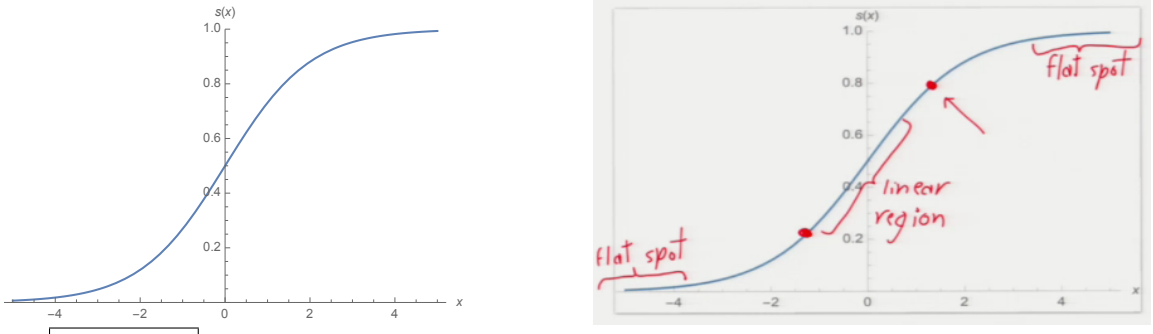
Classification: to choose from $k \geq 3$ classes, use softmax fn. [We deploy k separate output units. E.g., in the MNIST digit recognition problem, we would have $k = 10$ softmax output units, one for each digit class.]
Let $t = Wh$ be k-vector of linear combos in final layer.

$$\text{Softmax output is } z_j(t) = \frac{e^{t_j}}{\sum_{i=1}^{k} e^{t_i}}. \qquad \frac{\partial z_j}{\partial t_j} = z_j \times (1 - z_j), \quad \frac{\partial z_j}{\partial t_i} = -z_j z_i,\ j \neq i.$$

Each $z_j \in (0, 1)$; their sum is 1.
[Interpret z_j as the probability of the input belonging to class j.]
[If you have only 2 classes, just use one sigmoid output; it's equivalent to 2-way softmax.]

Sigmoid Unit Saturation

Problem: When unit output s is close to 0 or 1 for most training points, $s' = s(1-s) \approx 0$, so gradient descent changes s very slowly. Unit is "stuck." Slow training & bad local minima.



logistic.pdf [Draw flat spots, "linear" region, & maximum curvature points (at $s(\lambda) \doteq 0.21$ and $s(\lambda) \doteq 0.79$) of the sigmoid function. Ideally, we would stay away from the flat spots.]

[Wikipedia calls this the "vanishing gradient problem."]
[The more layers your network has, the more problematic this problem becomes. Most of the early attempts to train deep, many-layered neural nets failed.]

Mitigation: [None of these are complete cures.]

(1) For unit with fan-in η, initialize each incoming edge to
random weight with mean zero, std. dev. $1/\sqrt{\eta}$.
[The bigger the fan-in of a unit, the easier it is to saturate it. So we choose smaller random initial weights for units with bigger fan-in.]

(2) Set target values to 0.85 & 0.15 instead of 1 & 0.
[Recall that the sigmoid function can never be 0 or 1; it can only come close. If your target values are 1 & 0, you are pushing the output units into the flat spots! The numbers 0.15 and 0.85 are reasonable because the sigmoid function achieves its greatest curvature when its output is near 0.21 or 0.79. But experiment to find the best values.]
[Option (2) helps to avoid stuck output units, but not stuck hidden units. So . . .]

(3) Modify backprop to add small constant (typically ~ 0.1) to s'.
[This hacks the gradient so a unit can't get stuck. We're not doing *steepest* descent any more, because we're not using the real gradient. But often we're finding a better descent direction that will get us to a minimum faster. This hack originates with Scott Fahlman's *Quickprop* algorithm.]

(4) Cross-entropy loss fn instead of squared error.

For k-class softmax output, cross-entropy is $L(z, y) = -\sum_{i=1}^{k} y_i \ln z_i$.
$\uparrow$ true labels
$\uparrow$ prediction
$\Big\}$ k-vectors

Strongly recommended: choose labels so $\sum_{i=1}^{k} y_i = 1$.
[Typically, people choose one label to be 1 and the others to be 0. But by idea (2) above, it might be wiser to use less extreme values. From here on, we will assume that the target labels sum to 1.]

For [scalar] sigmoid output, $\quad L(z, y) = -y \ln z - (1 - y) \ln(1 - z)$ (aka logistic loss)

Derivatives for k-class softmax backprop:

$$\begin{aligned}
\frac{\partial L}{\partial z_j} &= -\frac{y_j}{z_j} \\
\nabla_{W_i} L &= \left(\sum_{j=1}^{k} \frac{\partial L}{\partial z_j} \frac{\partial z_j}{\partial t_i} \right) \nabla_{W_i} t_i = \left(-\frac{y_i}{z_i} z_i + \sum_j \frac{y_j}{z_j} z_j z_i \right) h = (z_i - y_i)\, h \\
\nabla_W L &= (z - y)\, h^\top \\
\nabla_h L &= \sum_{j=1}^{k} \frac{\partial L}{\partial z_j} \sum_{i=1}^{k} \frac{\partial z_j}{\partial t_i} \nabla_h t_i = \sum_{j=1}^{k} -\frac{y_j}{z_j} \left(z_j W_j - \sum_{i=1}^{k} z_j z_i W_i \right) = W^\top z - \sum_j y_j W_j = W^\top (z - y)
\end{aligned}$$

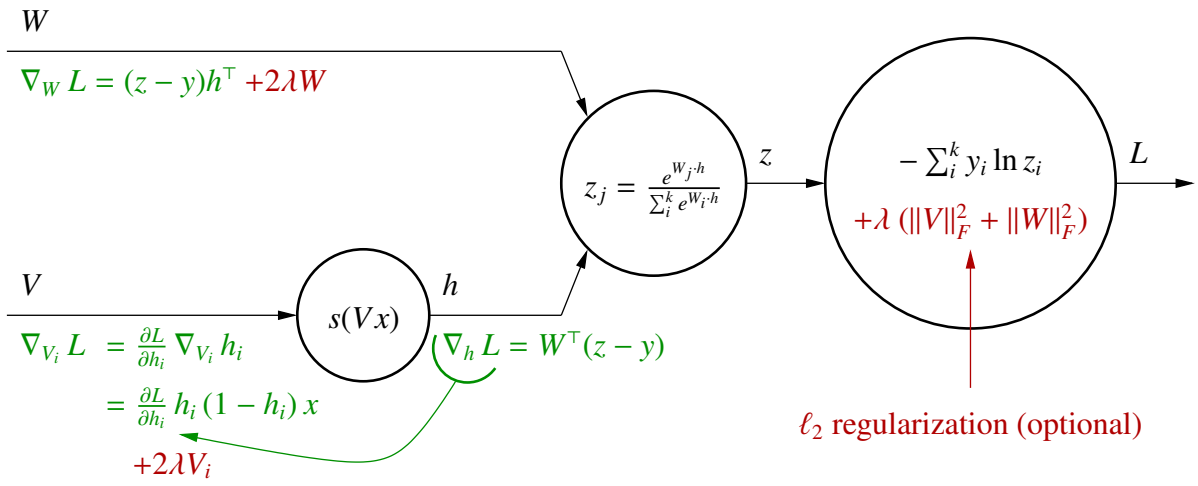
[Notice that the denominator of $\partial L / \partial z_j$ cancels out the numerator z_j in the softmax derivatives. This saves the unit from getting stuck when the softmax derivatives are small. It is related to the fact that the logistic loss goes to infinity as the predicted value z_j approaches zero or one. The vanishing gradient of the sigmoid is compensated for by the huge gradient of the logistic loss.]

For the sigmoid output, we also have $\nabla_W L = (z - y)\, h^\top$ and $\nabla_h L = W^\top (z - y)$.

[Like option (2), cross-entropy loss helps to avoid stuck output units, but not stuck hidden units.]

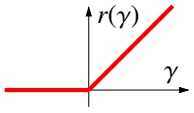
[Cross-entropy losses are only for sigmoid and softmax outputs. By contrast, for regression we typically use linear outputs, which don't saturate, so the squared error loss is better for them.]

[Now I will show you how to derive the backprop equations for a softmax output, the cross-entropy loss function, and ℓ_2 regularization—which helps to reduce overfitting, just like in ridge regression. Observe that because of the simplifications we made by combining derivatives, we don't compute $\nabla_z L$ explicitly, but we still have to backpropagate the value of z itself.]



[Draw this by hand. backpropsoft.pdf]

(5) Replace sigmoids with ReLUs: rectified linear units.
ramp fn aka hinge fn: $r(\gamma) = \max\{0, \gamma\}$



$$r'(\gamma) = \begin{cases} 1 & \gamma \geq 0 \\ 0 & \gamma < 0 \end{cases}$$

[The derivative is not defined at zero, but we just pretend it is.]

Popular for many-layered networks with large training sets.
[One nice thing about ramp functions is that they and their gradients are very fast to compute. Computers compute exponentials slowly. Even though ReLUs are linear in each half of their range, they're still nonlinear enough to easily compute functions like XOR.]
[Obviously, the gradient is sometimes zero, so you might wonder if ReLUs can get stuck too. Fortunately, it's rare for a ReLU's gradient to be zero for *all* the training data; it's usually zero for just some sample points. But yes, ReLUs sometimes get stuck too; just not as often as sigmoids.]
[The output of a ReLU can be arbitrarily large, creating the danger that it might overwhelm units downstream. This is called the "exploding gradient problem," and it is not a big problem in shallow networks, but it becomes a big problem in deep or recurrent networks.]

[Note that option (5) makes options (2)–(4) irrelevant.]