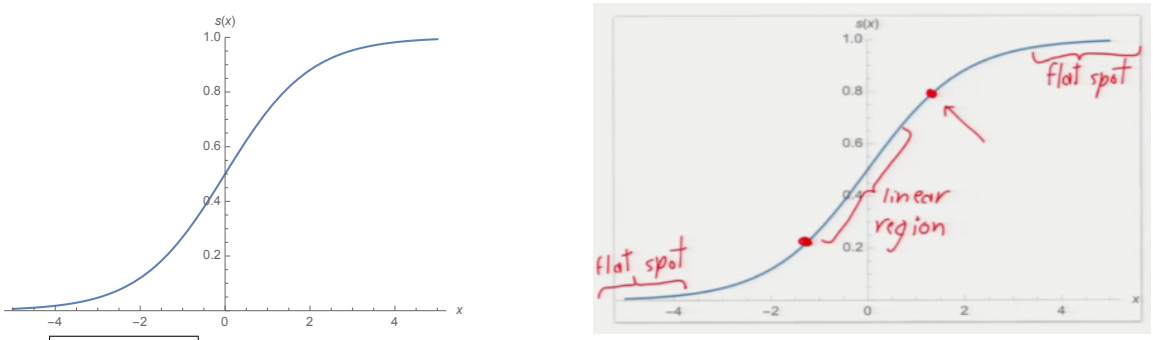
18 Vanishing Gradients; ReLUs; Output Units and Losses; Neurobiology

THE VANISHING GRADIENT PROBLEM; ReLUs

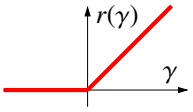
Problem: When unit output s is close to 0 or 1 for most training points, $s' = s(1-s) \approx 0$, so gradient descent changes s very slowly. Unit is "stuck." Slow training.



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.]

[This is called 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.]

Solution: Replace sigmoids with ReLUs: rectified linear units.
ramp 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.]

Most neural networks today use ReLUs for the hidden units.
[However, it is still common to use sigmoids for the output units in classification problems.]

[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.]

[Of course, if you replace sigmoids with ReLUs, you have to change the derivation of backprop to reflect the changes in the gradients.]

[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 training 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.]

Initializing Weights

[Recall that we initialize a neural network with random weight values to break the symmetry between hidden units. If we make those random values too small, breaking symmetry takes a long time. But if we make them too large, we may cause the vanishing gradient problem in sigmoid units, or the exploding gradient problem in ReLUs. Here's a rule of thumb for initializing weights into both sigmoid and ReLU units.]

For unit with fan-in η, initialize each incoming edge to random weight with mean zero, std. dev. $\frac{1}{\sqrt{\eta}}$.

[The idea is that the sum of all the incoming weights has mean zero and standard deviation of 1. The bigger the fan-in of a sigmoid unit, the easier it is to saturate it. So we choose smaller random initial weights for units with a bigger fan-in. We do the same with ReLUs to prevent their outputs from getting larger with every successive layer.]

OUTPUT UNITS

(1) Regression: usually linear output unit(s)—omit activation fn. [Unless you're regressing probabilities.] Usually trained with squred-error loss.

[If you want your regression function to sometimes be negative, clearly the output unit cannot be a ReLU or a sigmoid. For that purpose, a simple linear output unit is popular. You could say there is no activation function, or you could say that the activation function is the identity function. A linear output unit with a squared-error loss is doing least-squares linear regression on learned features (hidden units).]

(2) Classification: to choose from $k \geq 2$ classes, use softmax units.

[A set of k separate output units can do softmax regression on learned features (hidden units). E.g., in the MNIST digit recognition problem, we would have $k = 10$ softmax output units, one for each digit class.]

Let $y \in \mathbb{R}^k$ be vector of labels [for one training point, indicating its membership in the k classes].
In the final layer, compute $t = Wh \in \mathbb{R}^k$ and apply softmax activation to obtain prediction $z \in \mathbb{R}^k$.

Softmax output is $z_i(t) = \dfrac{e^{t_i}}{\sum_{j=1}^{k} e^{t_j}}$. Each $z_i \in (0, 1)$; $\displaystyle\sum_{i=1}^{k} z_i = 1$.

[Interpret z_i as an estimate of the posterior probability that the input belongs to class i.]

Strongly recommended: choose labels so $\displaystyle\sum_{i=1}^{k} y_i = 1$.

[Usually, for any single training point, people choose one label to be 1 and the others to be 0, which is called one-hot encoding. But one-hot encoding has a small disadvantage: each prediction z_i can never be exactly 1 or 0. If you choose target labels such as 0.1 or 0.9, a neural network with lots of layers, all sufficiently wide, can "interpolate" the labels, meaning that $z = y$ for every training point and the cost function achieves its global minimum. (Unless there are co-located training points with different labels.) From here on, we will only assume that the target labels sum to 1.]

To fix vanishing gradient problem, use cross-entropy loss fn instead of squared error.
[This method prevents the vanishing gradient problem in output units, but it can't be applied to hidden units.]

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

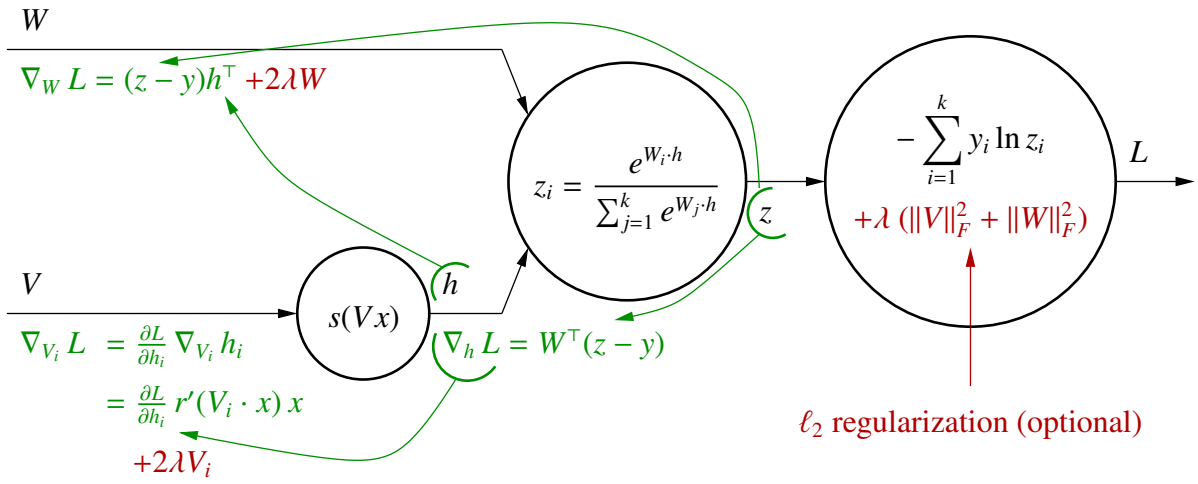
$\uparrow$ true labels
$\uparrow$ prediction
$\Big\}$ k-vectors

Derivatives for k-class softmax backprop: [correct only if $\sum_{i=1}^k y_i = 1$ for each training point]

$$
\begin{aligned}
L(z,y) &= -\sum_{i=1}^k y_i \ln z_i = -\sum_{i=1}^k y_i \left(t_i - \ln \sum_{j=1}^k e^{t_j} \right) = -\sum_{i=1}^k y_i t_i + \ln \sum_{j=1}^k e^{t_j}, \\
t_i &= W_i \cdot h, \qquad \nabla_{W_i} t_i = h, \qquad \nabla_h t_i = W_i, \\
\nabla_{W_i} L &= -y_i h + \left(e^{t_i} \Big/ \sum_{j=1}^k e^{t_j} \right) h = (z_i - y_i)\, h, \\
\nabla_W L &= (z-y)\, h^\top, \\
\nabla_h L &= -\sum_{i=1}^k y_i W_i + \left(\sum_{i=1}^k e^{t_i} W_i \right) \Big/ \left(\sum_{j=1}^k e^{t_j} \right) = -\sum_{i=1}^k (y_i - z_i) W_i = W^\top (z-y).
\end{aligned}
$$

[Notice that both $\nabla_W L$ and $\nabla_h L$ are linear in the error $z - y$. So the softmax unit with cross-entropy loss does not get stuck when the softmax derivatives are small. This is related to the fact that the cross-entropy loss goes to infinity as the predicted value z_i approaches zero or one. The vanishing gradient of the softmax unit is compensated for by the exploding gradient of the cross-entropy loss.]

[Now I will show you how to perform backpropagation for ReLU hidden units, a k-class 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 substituting softmax into cross-entropy, 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]

(3) Two-class classification: softmax outputs with $k = 2$ are sigmoid outputs.
$z_1 = s(t_1 - t_2)$; $\quad z_2 = 1 - z_1 = s(t_2 - t_1)$.
Cross-entropy loss is $L(z,y) = -y_1 \ln z_1 - y_2 \ln z_2 = -y_1 \ln z_1 - (1 - y_1) \ln(1 - z_1)$.

[You'll recognize this as the logistic loss from logistic regression. Our derivation of the gradients for softmax units applies to sigmoid units too. You don't necessarily have to provide the second, redundant sigmoid output z_2 to applications, but it's useful internally as part of computing $\nabla_W L$ and $\nabla_h L$.]

[Cross-entropy losses are only for sigmoid and softmax outputs. For linear or ReLU outputs, cross-entropy makes no sense, but squared error loss makes sense.]

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.]

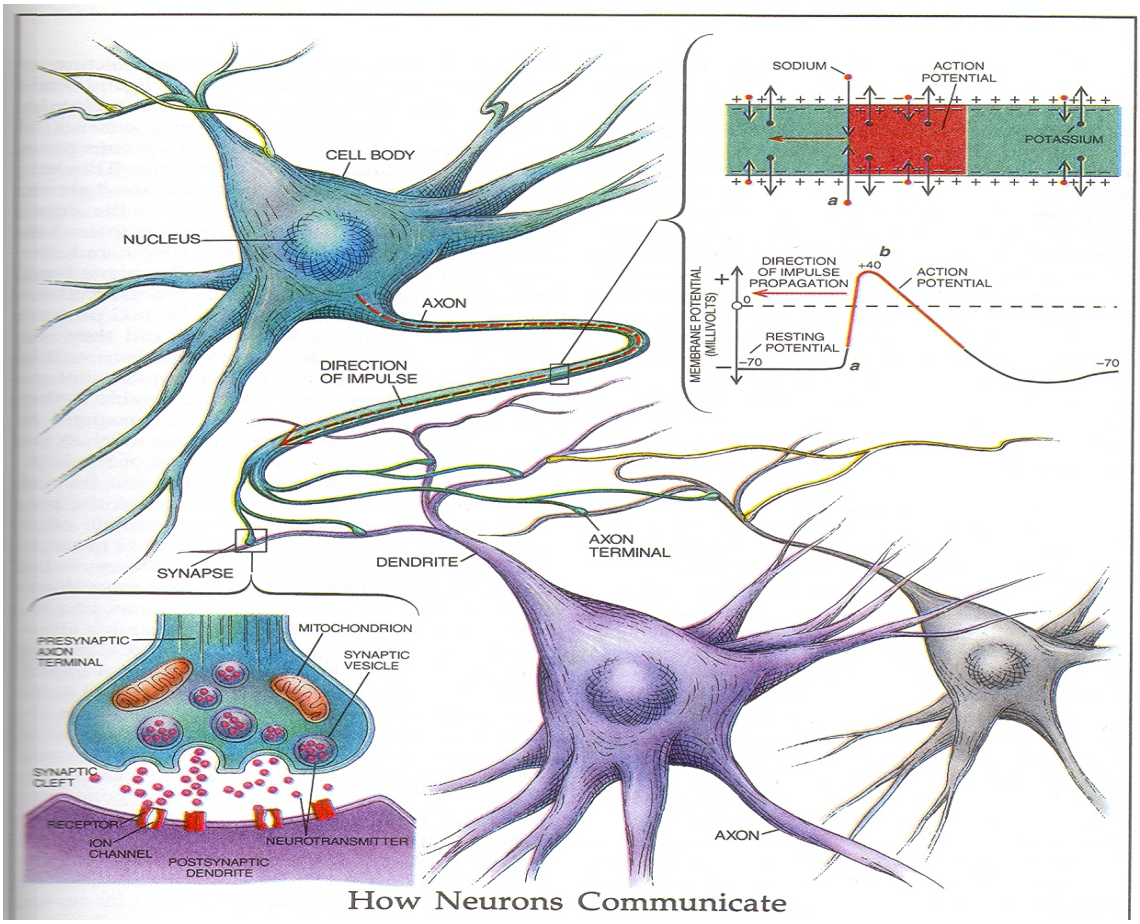
  [In the 1920's, the psychologist Karl Lashley conducted experiments to identify where in the brain memories are stored. He trained rats to run a maze, and then made lesions in different parts of the cerebral cortex, trying to erase the memory trace. Lashley failed; his rats could still find their way 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 the best techniques we have for many fuzzy problems, including most 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 squids have a very large axon they use for fast propulsion by expelling jets of water. The mathematics of action potentials was first characterized in these squid axons, and that work won a Nobel Prize in Physiology in 1963.]
- Dendrite: Smaller limb 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.]

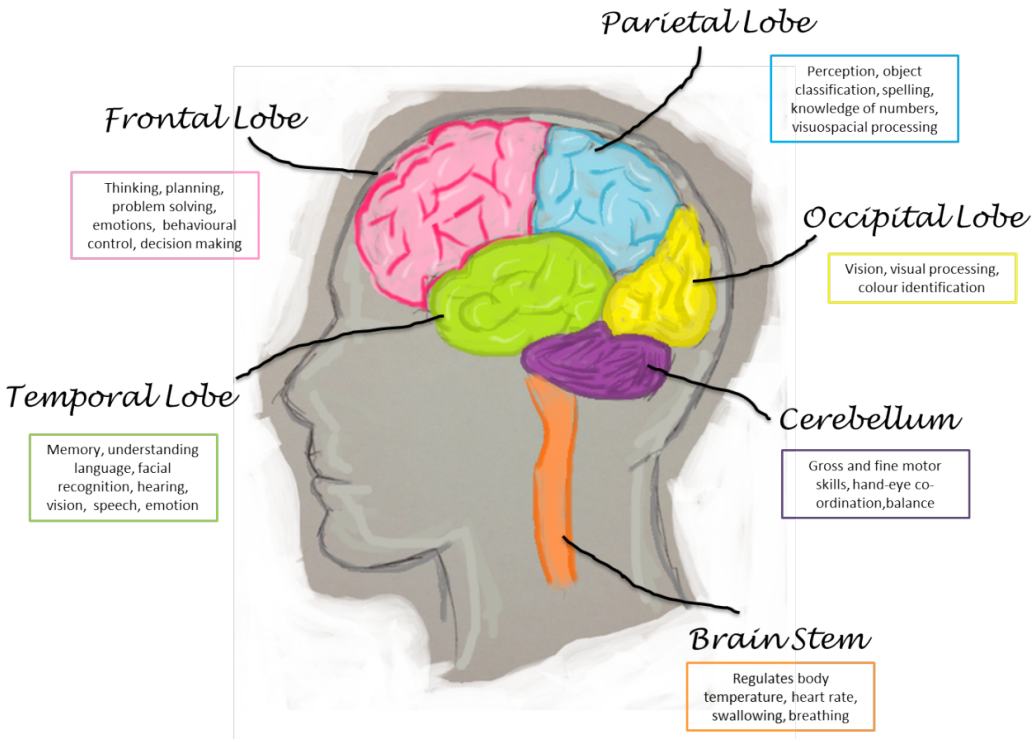
[(The following items are all spoken, not written. See the figure on the next page.)

The brain is very modular.

- 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.



brain.png

- 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 healthy parts 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.]