- Neurobiology
- Variations on NNs

Neurobiology

- NNs were inspired by how the brain works

CPU vs. Brain:

CPUs: largely sequential have, nanosecond gates, fragile if gate fails
- CPU >> for arithmetic, logical rules, perfect key-based memory

Brains: Very parallel, millisecond neurons, fault-tolerant

Neurons are always dying
↳ memories stored in brain in diffuse representations: no single neuron's death will make you forget something
- artificial NNs have the same resilience
- "memories" stored w/ same set of weights

Brains superior for:
- Vision
- Speech
- Associative Memory
  - good at noticing connections between things

NNs try to emulate parallelism & associativity of brains
- inferior for typical comp. tasks like multiplying 10-digit numbers or compiling source code

neuron: cell in brain
action potential/spike: impulse fired by neuron to communicate w/ other neurons
axon: limb along which action potential travels output
dendrite: neuron's receptors input
Synapse: Connection from one neuron's axon to another's dendrites
Neurotransmitter: chemical released by axon terminal to stimulate the dendrite

Brain Vs. NN:

NN	Brain
Output of a Unit	firing rate of neuron
Strong "1" output	"all or nothing" action rate, high firing rate
Weight of Connection	Synapse Strength
pos. weight	excitatory neurotransmitter
neg. weight	inhibitory neurotransmitter

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. On the other hand, artificial neural networks have an advantage here in that weights can change freely.

- Linear combination of inputs $\iff$ 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 with time, so if the action potentials are coming in slowly enough, the neuron might not fire at all.

- Logistic/sigmoid function $\iff$ firing rate saturation

  A neuron can't fire more than 1000 times a second, nor less than zero times a second. This limits its ability to overpower downstream neurons, and we simulate this behavior with the sigmoid function.

- Weight change/learning $\iff$ synaptic plasticity

  Hebb's rule (1949): "Cells that fire together, wire together."

  Typically, one cell's firing tends to make another cell fire more often, and their excitatory synaptic connection tends to grow stronger. Similarly, there's a reverse rule for inhibitory connections, and there are ways for neurons that aren't even connected to grow connections.

Backpropagation is one part of artificial neural networks for which an analogy is doubtful. There have been some proposals that the brain might do something similar to backpropagation, but it's tenuous.

Neural Network Variations

W/ regression, linear output unit → no sigmoid

W/ classification, use Softmax function:

Sum of outputs = 1

$$z_j(t) = \frac{e^{t_j}}{\sum e^{t_i}}, \quad \frac{\partial z_j}{\partial t_j} = \begin{cases} z_j(1-z_j) & i=j \\ -z_j z_i & i \neq j \end{cases}$$

Sigmoid Unit Saturation

output close to 0 or 1 for most training pts → $S' = S(1-S) \approx 0$

→ slow gradient descent, unit is "stuck"

→ converge to bad local minima

Vanishing Gradient Problem

- more layers the network has → bigger issue

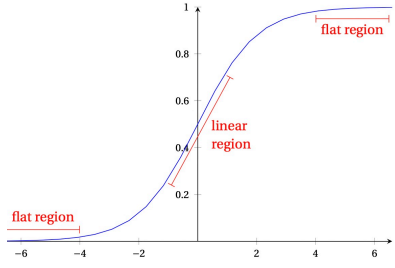


Figure 18.1: The sigmoid function and its flat regions.

Possible Solutions:

① if a unit has n incoming edges, initialize each incoming edge w/ $\mu = 0$ and $\sigma = \frac{1}{\sqrt{n}}$

- the bigger the fan-in, the easier it is to saturate
  - ↳ choose smaller random initial weights for large n

② Set target values to 0.85 & 0.15 instead of 0 & 1

- Sigmoid has greatest curviture at 0.21 & 0.79

③ add a small constant ≈ 0.1 to S' in backpropagation

- hacks gradient, unit can't get stuck
- not quite "steepest descent" anymore — quickprop
- instead, we find direction w/ quicker convergence

④ Cross-entropy loss instead of squared error

for k-class softmax output: $L(\vec{z}, \vec{y}) = -\sum_{i=1}^{k} y_i \ln z_i$

Strongly recommended: choose labels such that $\sum_{i=1}^{k} y_i = 1$

↳ typically one-hot encoding, but ② suggests otherwise

Scalar sigmoid output: $L(z, y) = -y \ln z - (1-y)\ln(1-z)$ logistic loss

Derivatives for k-class Softmax backprop:

$$\frac{\partial L}{\partial z_j} = \frac{-y_i}{z_j}$$

$$\nabla_{w_i} L = \left(\sum_{j=1}^{k} \frac{\partial L}{\partial z_j} \frac{\partial z_j}{\partial t_i} \right) \cdot \nabla_{w_i} t_i = \left(-\frac{y_i}{z_i} z_i + \sum_{j=1}^{k} \frac{y_j}{z_j} z_j z_i \right) \vec{h} = (z_i - y_i)\vec{h}$$

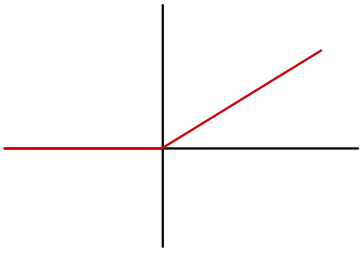
$$\rightarrow \nabla_W L = (\vec{z} - \vec{y})\vec{h}^T$$

$$\nabla_{\vec{h}} L = \sum_{j=1}^{k} \frac{\partial L}{\partial z_j} \sum_{i=1}^{k} \frac{\partial z_j}{\partial t_i} \cdot \nabla_{\vec{h}} t_i = \sum_{j=1}^{k} -\frac{y_j}{z_j} \left(z_j w_j - \sum_{i=1}^{k} z_j z_i w_i \right) = W^T(\vec{z} - \vec{y})$$

Cross-entropy loss helps avoid Stuck output units, can't help Stuck hidden units
- only for Sigmoid & softmax outputs
- w/ regression, we use linear outputs that don't saturate
  - ↳ squared error loss better

⑤ Replace Sigmoids w/ ReLUs: $r(v) = \max(0, v)$,

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



- ReLU popular for deep networks & large training sets
- gradient & $r(v)$ fast to compute
- Still nonlinear → can compute XOR
- gradient sometimes 0, but rare for ReLU's gradient to be 0 for all pts.
- arbitrarily large output
  - ↳ might overwhelm units downstream exploding grad. problem

Option 5 makes ② ③ ④ useless