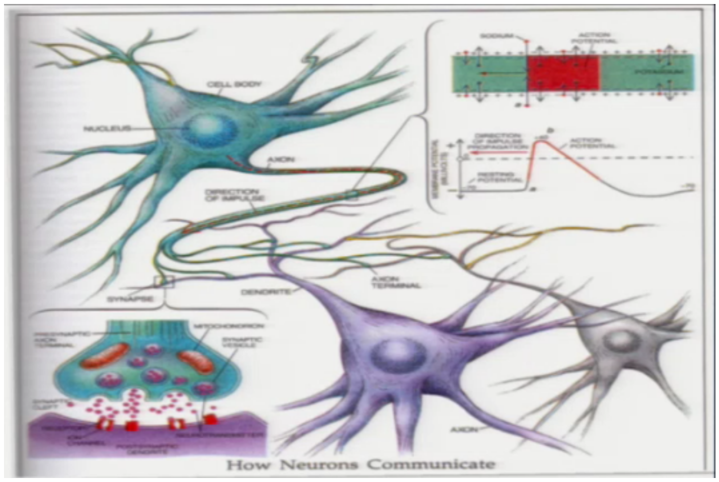
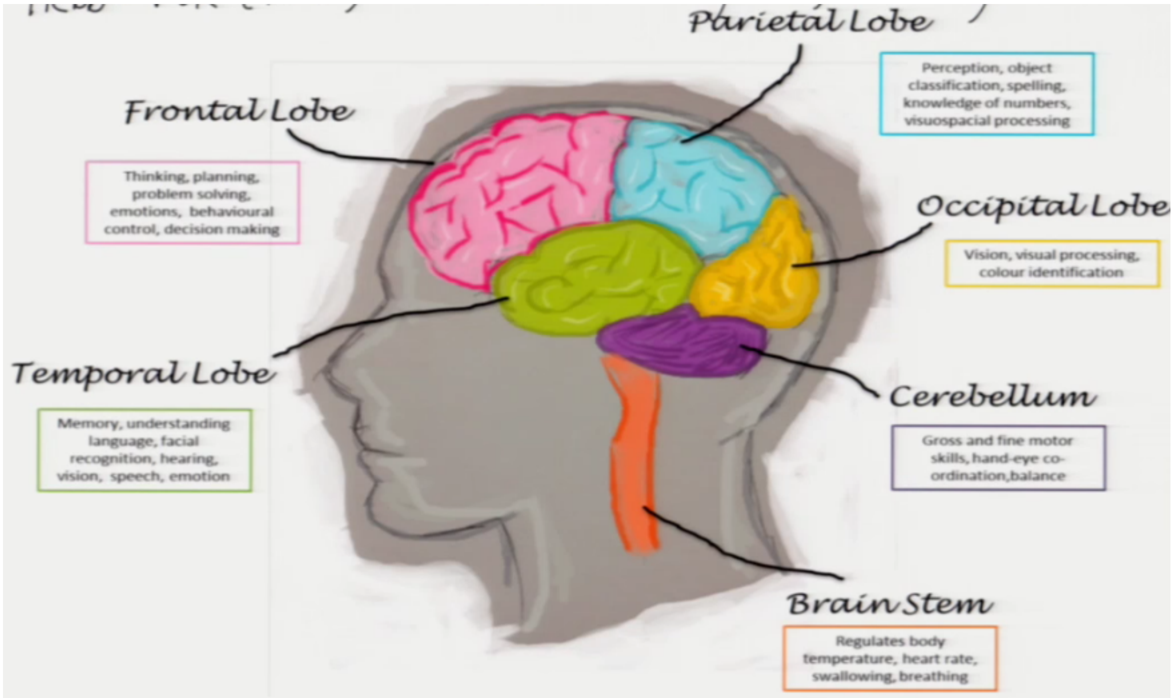
04/04/2016

Neurons

- CPUs: largely sequential, nanosecond gates, fragile if gate fails, superior for 234x718, local rules, perfect key-based memories.
- Brains: very parallel, millisecond neurons, fault-tolerant, superior for vision, speech, associative memory.



- Neuron: a cell in brain/nervous system for thinking/communicating.
- Action potential or spike: An electrochemical impulse fired by a neuro nto communicate with other neurons.
- Axon: the limb(s) along which action potentials propagate; "output."
- Dendrite: Smaller limbs by which neuron receives info; "input."
- Synapse: Connection from one neuron's axon to another's dendrite.
- Neurotransmitter: Chemicals released by axon terminal to stimulate dendrite.
- You have about 10^{11} neurons, each with about 10^4 synapses.
- Analogies:
  - Output of unit $\leftrightarrow$ firing rate of neuron.
  - Weights of connection $\leftrightarrow$ synapse strength.
  - Positive weight $\leftrightarrow$ excitatory neurotransmitters (e.g. glutamine).
  - Negative weight $\leftrightarrow$ inhibitory neurotransmitters (e.g. GABA, glycibe)
  - Linear combination of inputs $\leftrightarrow$ summation.
  - Logistic/sigmoid function $\leftrightarrow$ firing rate saturation.
  - Weight change/learning $\leftrightarrow$ synaptic plasticity. Hebb's rule (1949): "cells that fire together, wire together.""



Neural Net Variations

- Regression: Usually omit sigmoid function from output unit(s).
- Classification:
  - Logistic loss function (aka cross-entropy) often preferred to squared error:

$$L(z, y) = -\sum_i (y_i \ln z_i + (1 - y_i) \ln(1 - z_i))$$

  - For 2 classes, use one sigmoid output; for $k \geq 3$ classes, use softmax function.
    - Let $t = Wh$ be $k-$vector of linear combination in final layer.

Backpropagation with softmax output + logistic loss fn

Softmax output is $\underbrace{z_j(t) = \frac{e^{t_j}}{\sum_{i=1}^{k} e^{t_i}}}_{\in (0,1) \quad \sum_j z_j = 1}$ $\qquad \frac{dz_j}{dt_j} = z_j(1 - z_j) \qquad \underbrace{\frac{dz_j}{dt_i} = -z_i z_j}_{i \neq j}$

$t_i = W_i \cdot h$

W

$$\nabla_{W_i} L = \sum_{j=1}^{k} \frac{\partial L}{\partial z_j} \nabla_{W_i} z_j$$

$$= \left(\frac{\partial L}{\partial z_i} - \sum_{j=1}^{k} \frac{\partial L}{\partial z_j} z_j\right) z_i h$$

$$+ 2\lambda W_i$$

$$z_j = \frac{e^{W_j \cdot h}}{\sum_{i=1}^{k} e^{W_i \cdot h}}$$

z

$$\frac{\partial L}{\partial z_j} = \frac{1 - y_j}{1 - z_j} - \frac{y_j}{z_j}$$

$$-\sum_{i=1}^{k} \begin{matrix} y_i \ln z_i + \\ (1 - y_i)\ln(1 - z_i) \end{matrix}$$

$$+ \lambda |w|^2$$

L

optional L_2-regularization

[w is vector containing all the weights in matrices V & W.]

h

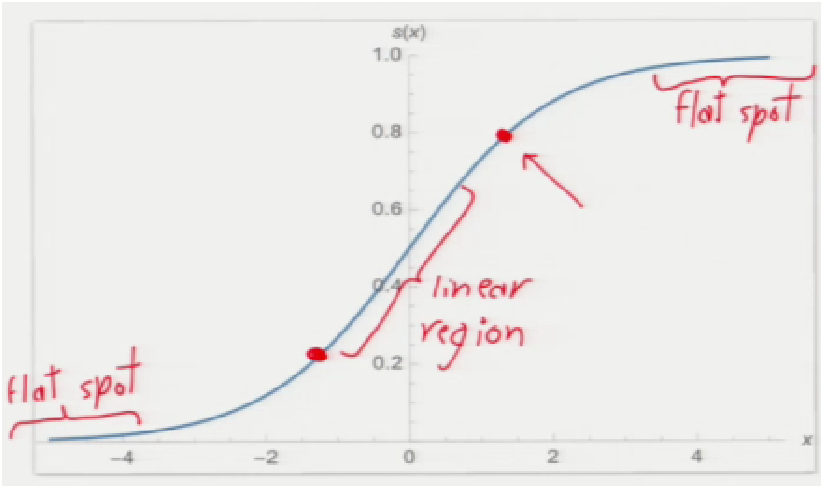
$$\frac{\partial L}{\partial h_i} = \sum_{j=1}^{k} \frac{\partial L}{\partial z_j} \frac{\partial z_j}{\partial h_i}$$

$$= \sum_{j=1}^{k} \frac{\partial L}{\partial z_j} \left(W_{ji} - \sum_{\ell=1}^{k} W_{\ell i} z_\ell\right) z_j$$

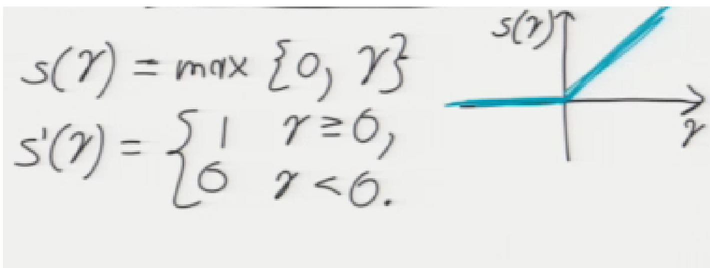
Derivatives of inputs to hidden units h are computed same way as previously.

Unit Saturation

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



- Mitigation:
  1. Set target value (y) to 0.15 and 0.85 instead of 0 and 1.
  2. Modify back-propagation to add small constant (typically ~ 0.1) to s'.
  3. Initial weight of edge into unit with fan-in η: random with mean zero, standard deviation $\sqrt{\eta}$.
  4. Replace sigmoid with ReLUs: rectified linear units. ramp function aka hinge function:



Heuristics for Avoiding Bad Local Minima

- 1 or 4 above.
- Stochastic gradient descent. A local minimum for batch descent is not a minimum for one typical training point.
- Momentum. Gradient descent changed "velocity" slowly. Carries us right through shallow local minima to deeper ones.

```
Δw ← −ϵ∇w
repeat:
    w ← w + Δw  (Δw is speed)
    Δw ← −ϵ∇w + βΔw  (β how strongly momentum persists. 0 ≤ β < 1)
```