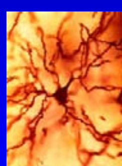
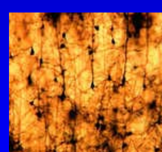
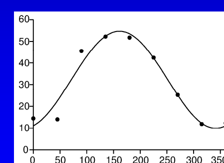
# Lecture 2

# Basic Neurobiology & Machine Learning for Brain-Computer Interfacing

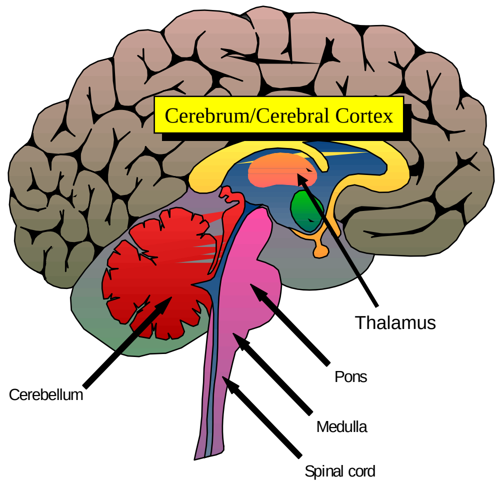





## Today’s Roadmap

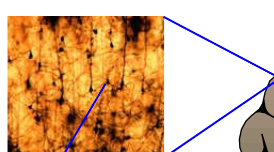
- PART I: Basic Neuroscience for BCI
  - The neuron doctrine (or dogma)
  - Neuronal signaling
    - Action Potentials (= spikes)
    - Synapses
  - Brain organization and function
- PART II: Basic Machine Learning for BCI
  - Supervised Learning
    - Regression: Linear, polynomial
    - Radial Basis Functions
    - Artificial Neural Networks

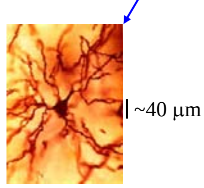
# Our 3-pound Universe



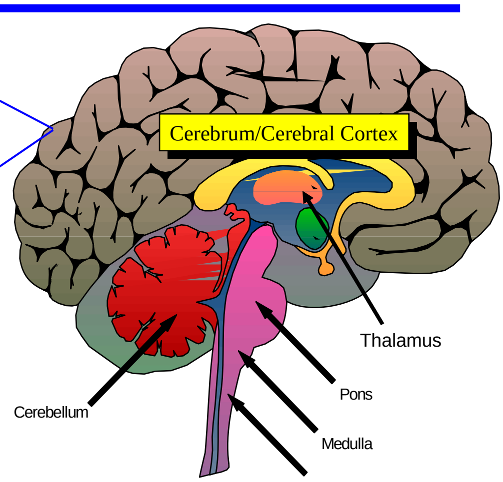



# Enter…the neuron (“brain cell”)

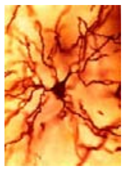




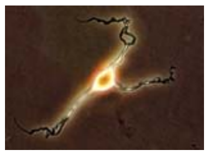

A Pyramidal Neuron



# The Neuron Doctrine/Dogma

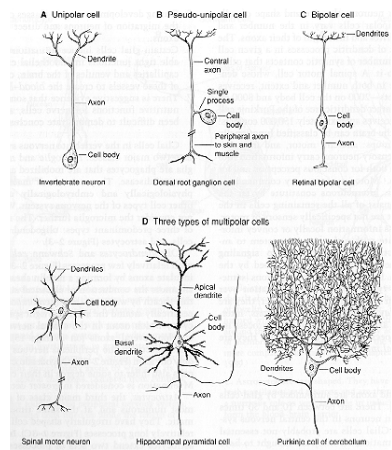


Cerebral Cortex Neuron



Neuron from the Thalamus

Neuron from the Cerebellum




From Kandel, Schwartz, Jessel, Principles of Neural Science, 3rd edn., 1991, pg. 21

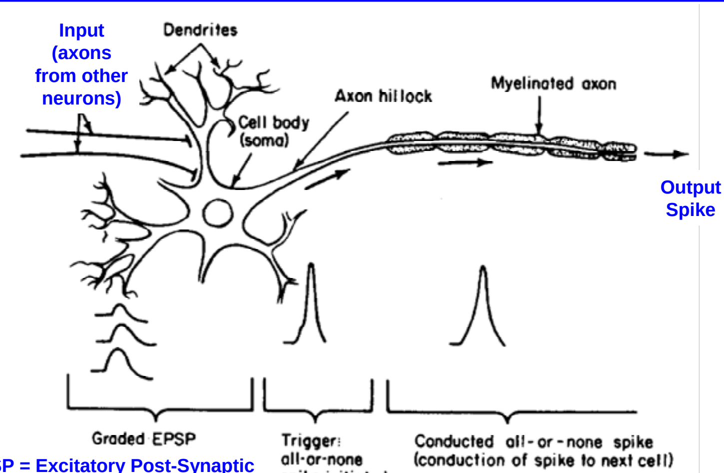
Neuron Doctrine:

"The neuron is the appropriate basis for understanding the computational and functional properties of the brain"

First suggested in 1891 by Waldeyer



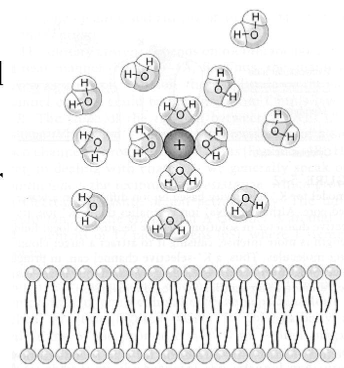
# The Idealized Neuron




(EPSP = Excitatory Post-Synaptic Potential)

## What is a Neuron?

- A "leaky bag of charged liquid"
- Contents of the neuron enclosed within a *cell membrane*
- Cell membrane is a *lipid* bilayer
  - Bilayer is impermeable to charged ion species such as $Na^+$, $Cl^-$, $K^+$, and $Ca^{2+}$
  - Embedded ionic channels or "gates" allow ions in or out



From Kandel, Schwartz, Jessel, Principles of Neural Science, 3rd edn., 1991, pg. 67



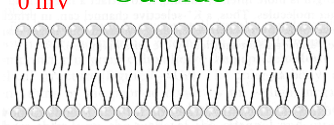
## The Electrical Personality of a Neuron

- Each neuron maintains a *potential difference* across its membrane
  - Inside is –70 to –80 mV relative to outside
  - *Ionic pump* maintains -70 mV difference by expelling $Na^+$ out and allowing $K^+$ ions in

$[Na^+]$, $[Cl^-]$, $[Ca^{2+}]$

$[K^+]$, $[A^-]$

0 mV Outside



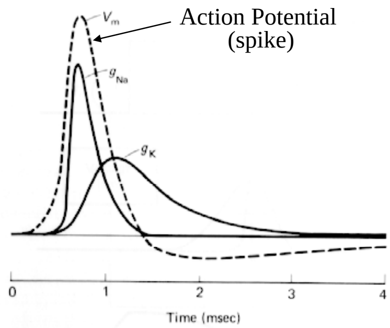
-70 mV Inside

$[K^+]$, $[A^-]$

$[Na^+]$, $[Cl^-]$, $[Ca^{2+}]$

# The Output of a Neuron: Action Potentials

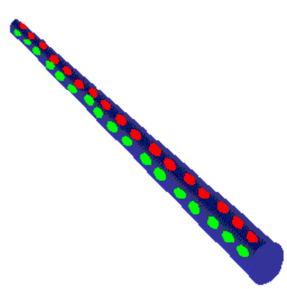
- Voltage-gated channels cause action potentials (spikes)
  1. Rapid $Na^+$ influx causes rising edge
  2. $Na^+$ channels deactivate
  3. $K^+$ outflux restores membrane potential




From Kandel, Schwartz, Jessel, Principles of Neural Science, 3rd edn., 1991, pg. 110



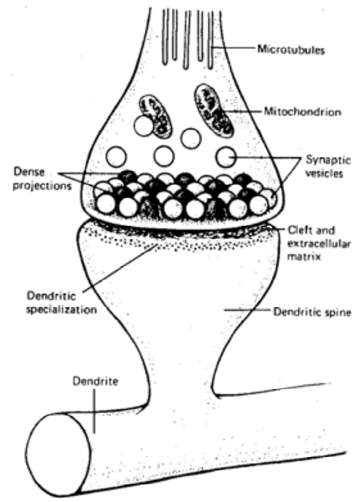
# Propagation of a Spike along an Axon



 From: http://psych.hanover.edu/Krantz/neural/actpotanim.html 

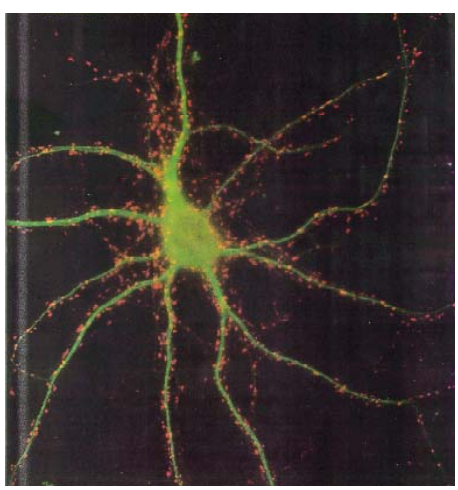
## Communication between Neurons: Synapses

- Synapses are the "connections" between neurons
  - Electrical synapses (gap junctions)
  - Chemical synapses (use neurotransmitters)
- Synapses can be excitatory or inhibitory
- Synapse Doctrine: Synapses are the basis for memory and learning






## Distribution of synapses on a real neuron…



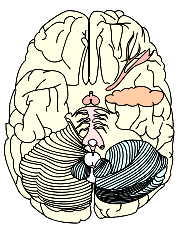
## Organization of the Nervous System

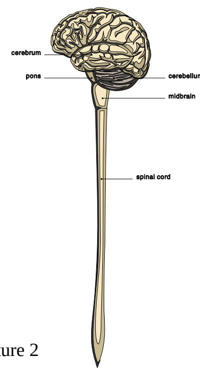
Central Nervous System

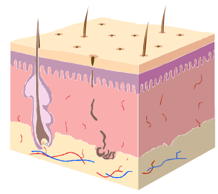
Brain | Spinal Cord

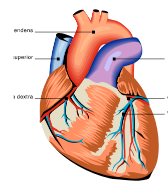
Peripheral Nervous System

Somatic | Autonomic









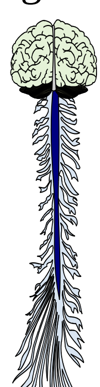
## Autonomic and Central Nervous System

Autonomic: Nerves that connect to the heart, blood vessels, smooth muscles, and glands

CNS = Brain + Spinal Cord

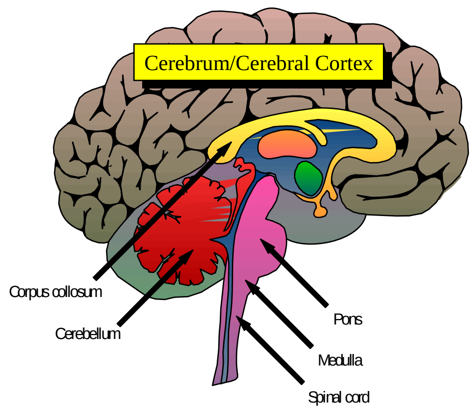
Spinal Cord:

- Local feedback loops control reflexes
- Descending motor control signals from the brain activate spinal motor neurons
- Ascending sensory axons transmit sensory feedback information from muscles and skin back to brain



## Major Brain Regions: Cerebral Hemispheres

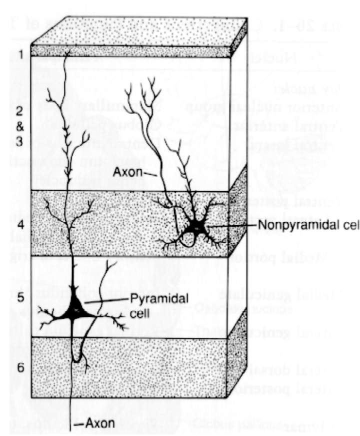
- Consists of: Cerebral cortex, basal ganglia, hippocampus, and amygdala
- Involved in perception and motor control, cognitive functions, emotion, memory, and learning





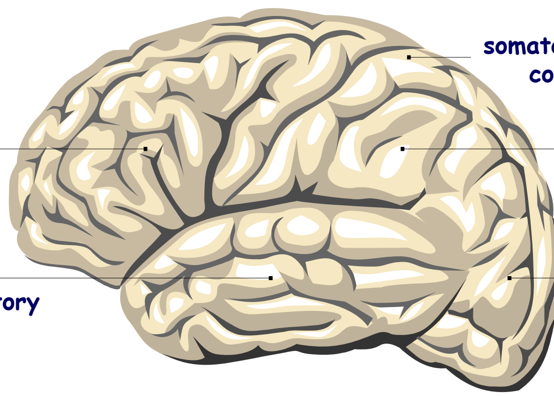

## Cerebral Cortex: A Layered Sheet of Neurons

- Cerebral Cortex: Convoluted surface of cerebrum about 1/8th of an inch thick
- Six layers of neurons
- Approximately 30 billion neurons
- Each neuron makes about 10,000 synapses: approximately 300 trillion connections in total



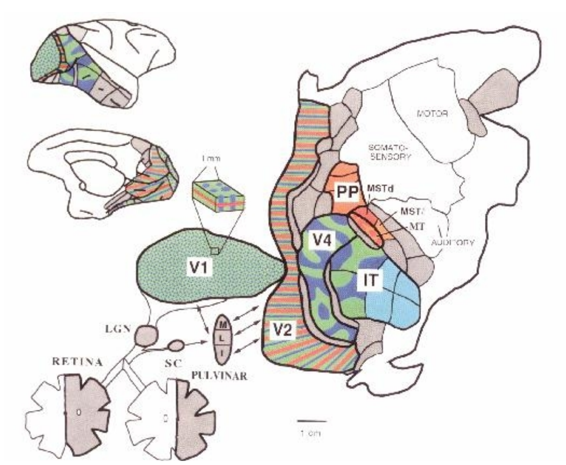

From Kandel, Schwartz, Jessel, Principles of Neural Science, 3rd edn., 1991, pgs.

# Specialization of Function in Cerebral Cortex

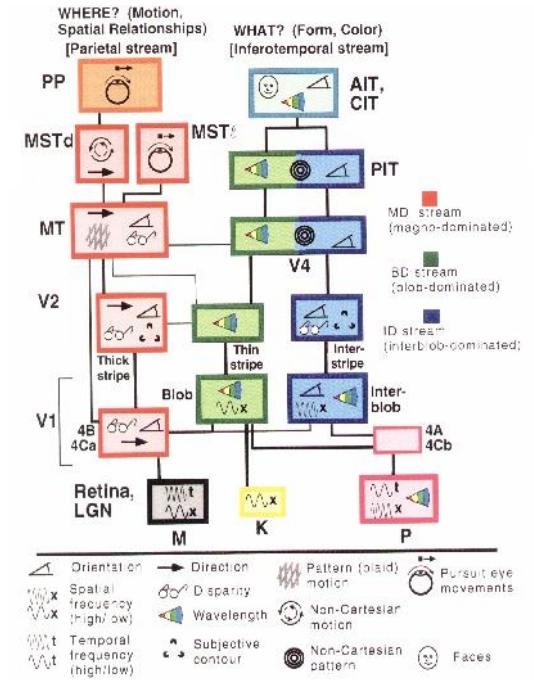





# Hierarchical Organization of Visual Cortex

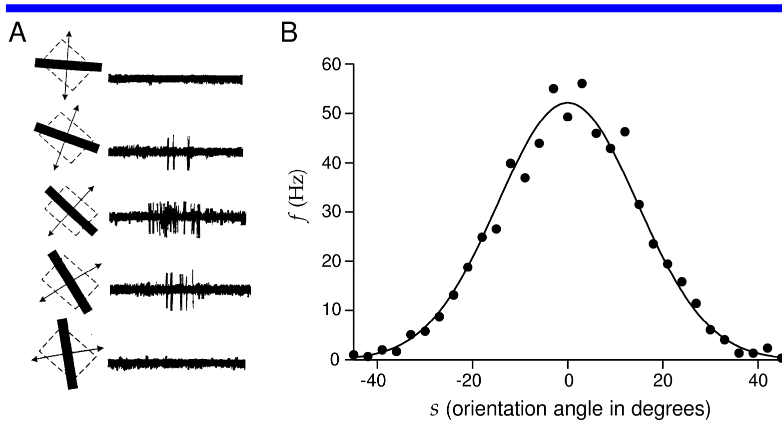


# The Visual Processing Hierarchy





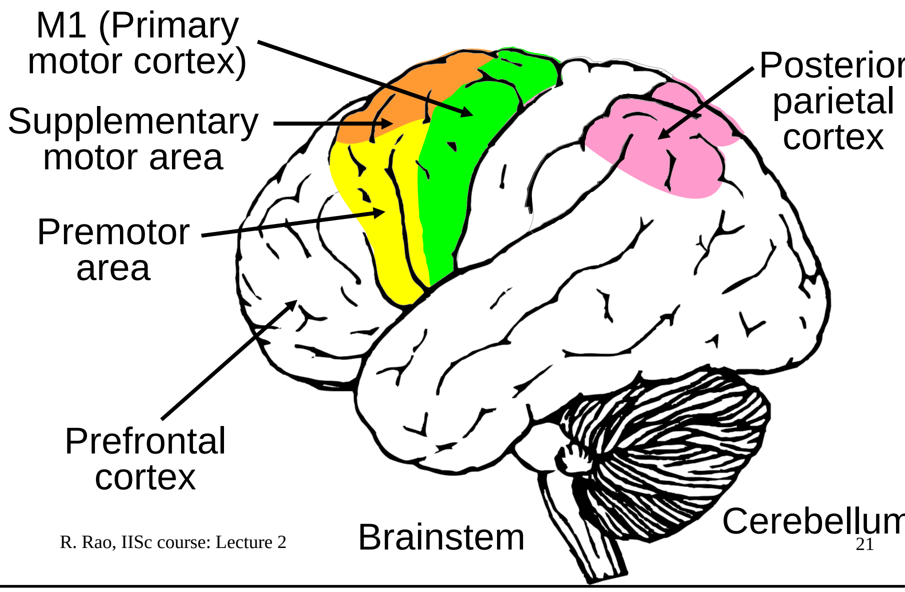

# Tuning Curve of a Visual Cortical Neuron




Spike trains as a function of bar orientation

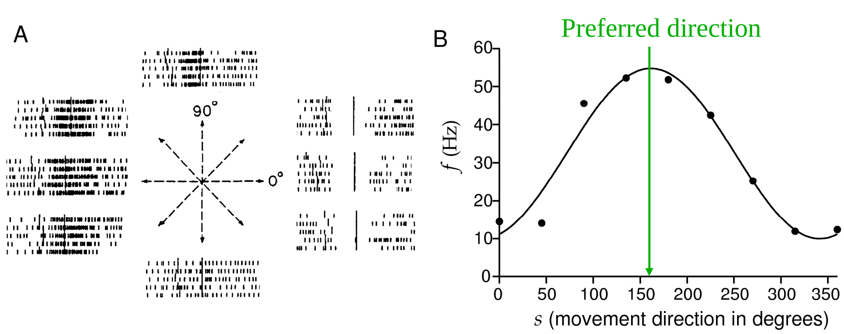
Gaussian Tuning Curve

# The Motor Hierarchy





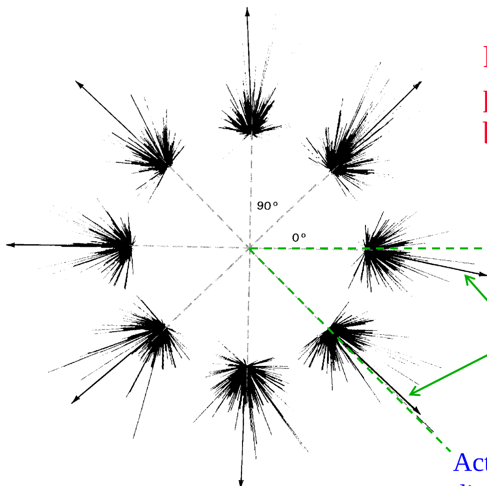

# Tuning Curve of a Neuron in M1




Spike trains as a function of hand reaching direction

Cosine Tuning Curve

## Movement Direction can be Predicted from a Population of M1 Neurons' Firing Rates




Population vector = sum of preferred directions weighted by their firing rates

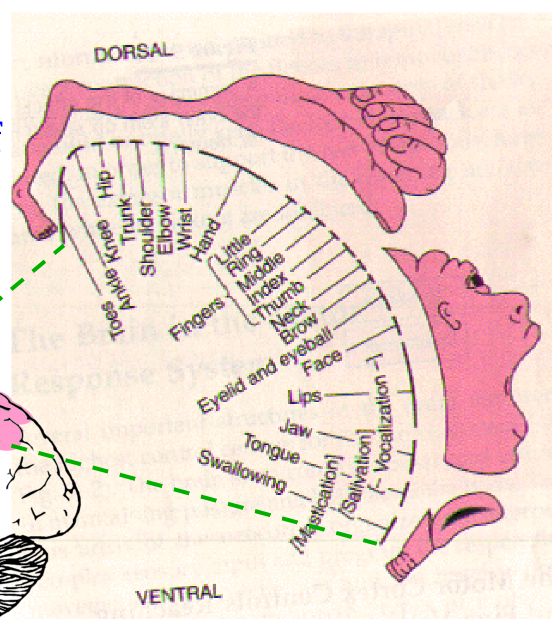
Actual arm movement direction

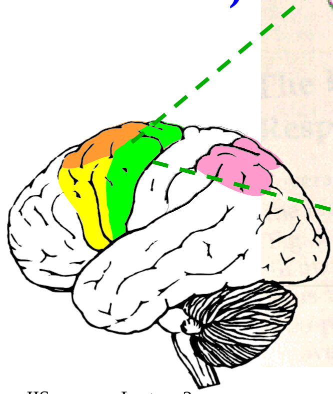
**Population vectors (decoded movement direction)**

Actual arm movement direction



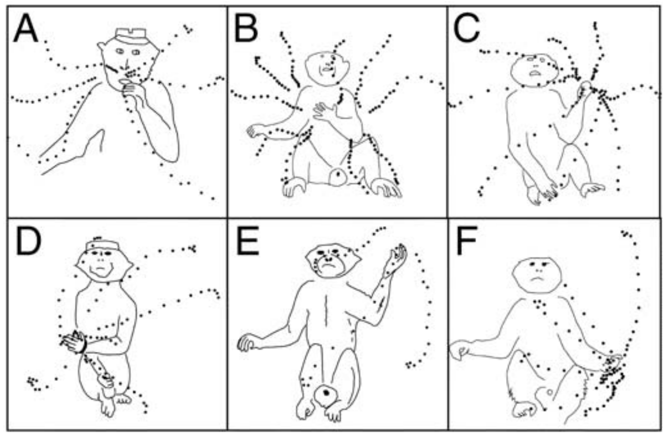
## Somatotopic Organization of M1 (a.k.a. the "homunculus")



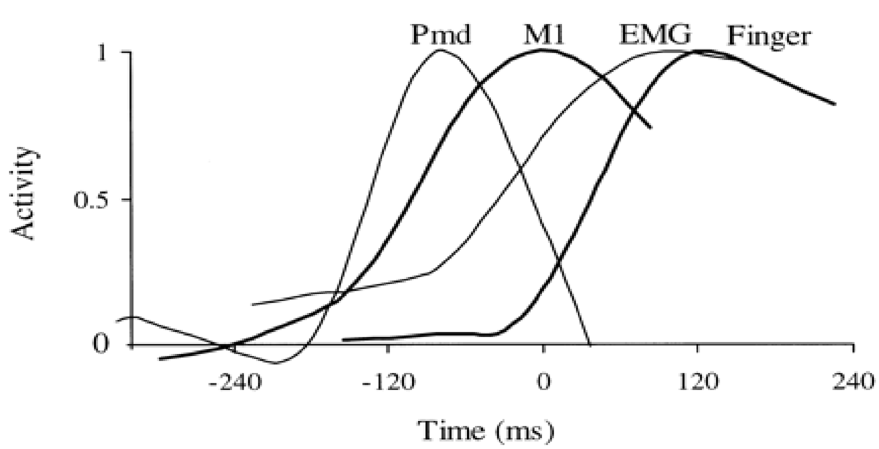
## Electrically stimulating M1 elicits primitive movements

Electrically stimulating *Premotor Area* elicits more complex movements






## Activity in Motor Hierarchy during Reaching



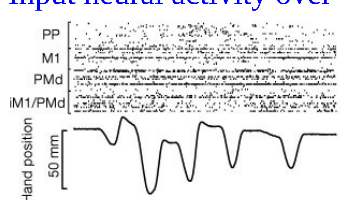
# Summary: Brain versus Digital Computing

- **Device count**:
  - Human Brain: $10^{11}$ neurons (each neuron ~ $10^4$ connections)
  - Silicon Chip: $10^{10}$ transistors with sparse connectivity
- **Device speed**:
  - Biology has 100μs temporal resolution
  - Digital circuits approaching 100ps clock (10 GHz)
- **Computing paradigm**:
  - Brain: Massively parallel computation & adaptive connectivity
  - Digital Computers: sequential information processing via CPU with fixed connectivity
- **Capabilities**:
  - Digital computers excel in math & symbol processing…
  - Brains: Better at solving ill-posed problems (speech, vision)?



# Part II: Basic Machine Learning for BCI

Input neural activity over time




Learn mapping

Output over time (hand position)

## Why machine learning for BCIs?

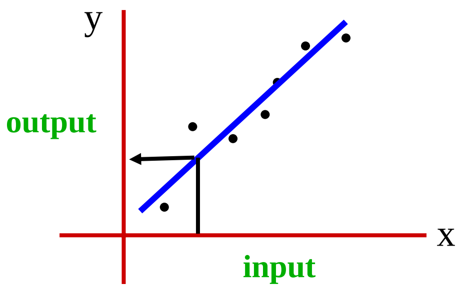
- In most BCI applications, we have ***example inputs and outputs***
  - Inputs = Neural data; Outputs = Position of hand or robot, class of imagined movement etc.
- We wish to ***learn a function*** mapping arbitrary inputs to outputs
  - **Supervised learning**
- Dominant paradigms in BCI literature
  - Map neural activity to continuous outputs (e.g., hand position) ⇒ ***regression*** (Invasive BCIs).
  - Classify brain patterns into one of several classes, and use this to select action ⇒ ***classification*** (EEG BCIs)



## Outline

- *Regression*
  - Linear, polynomial
  - RBFs, perceptrons, multilayer neural networks
- *Classification*
  - Linear classifiers, support vector machines
  - Multi-class classifiers
- *Cross-validation*
  - Model selection, preventing overfitting

# Linear Regression




| x | y |
|---|---|
| 1 | 3.1 |
| 2 | 6.4 |
| 3.1 | 8.9 |
| 0.9 | 2 |

Assumption: Output is a linear function of input, i.e.,

$$\mathbf{y_i = wx_i + noise}$$

where noise is independent, gaussian, unknown fixed variance



# Linear Regression

**Given**: Data $(\mathbf{y_i, x_i})$ where $\mathbf{y_i}$ are drawn from $\mathbf{N(wx_i, \sigma^2)}$

Likelihood of data $(\mathbf{y_i, x_i})$ for a given **w** is:

$\Pi_i\, p(y_i \mid w, x_i)$ *which is equal to*

$\Pi_i \exp( -0.5\, (y_i - wx_i))^2/\sigma^2$ *(ignoring constants)*

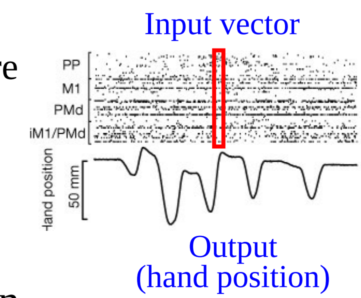
**Goal**: Maximize the likelihood of data given w

i.e., maximize: $\Sigma_i\ -0.5(y_i - wx_i)^2/\sigma^2$

i.e., minimize: $\Sigma_i\ (y_i - wx_i)^2$

**Easy to show that** $\mathbf{w = \Sigma\, x_i\, y_i \,/\, \Sigma\, (x_i)^2}$

But…typically, inputs in BCIs are ***vectors*** of multiple neurons' activities, multiple EEG measurements, etc.




Need Multivariate Regression



# Multivariate regression

Suppose inputs $\mathbf{x}_i$ are n-element vectors: $y_i = \mathbf{w}^T\mathbf{x}_i$ + noise

Write the *m* data points as:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Then, $\mathbf{Y} = \mathbf{Xw} + \mathbf{noise}$

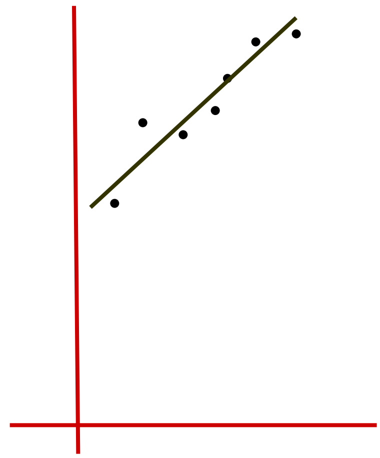
Maximum likelihood $\mathbf{w}$ is

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{Y})$$

# Linear regression: constants

What if data does not go through origin?

| x | y |
|---|---|
| 1 | 8.1 |
| 2 | 11.4 |
| 3.1 | 13.7 |
| 0.7 | 7 |





# Linear Regression: constants

Solution: Add a dummy input fixed at 1 and learn its coefficient (constant offset)

| x | y |
|---|---|
| 1 | 8.1 |
| 2 | 11.4 |
| 3.1 | 13.7 |
| 0.7 | 7 |

→

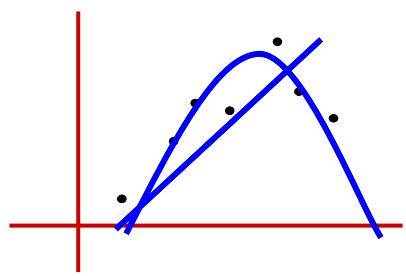
**z** (spanning $z_0$ and $z_1$)

| $z_0$ | $z_1$ (= x) | y |
|---|---|---|
| 1 | 1 | 8.1 |
| 1 | 2 | 11.4 |
| 1 | 3.1 | 13.7 |
| 1 | 0.7 | 7 |

Learn **w** for the new function $y = \mathbf{w}^T\mathbf{z}$ + noise

$= w_1 x + w_0$ + noise

## What if the data looks like this?



Need to generalize to non-linear regression…any ideas?

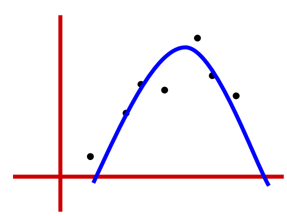


## Non-Linear Regression: Polynomials

- **Use same trick as for constants:**
  - Replace input x by modified input vector **z**

Example: Quadratic Regression with original input $\mathbf{x} = [x_1\ x_2]$

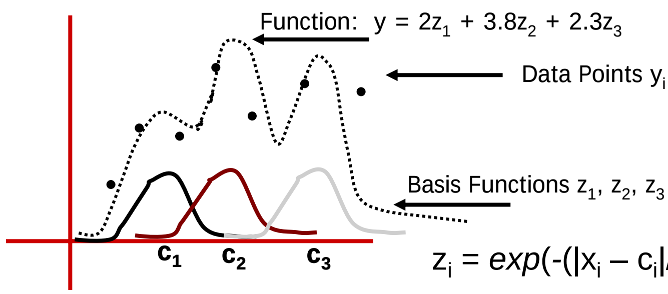
| $z_0$ | $z_1$ | $z_2$ | $z_3$ | $z_4$ | $z_5$ | y |
|---|---|---|---|---|---|---|
| … | … | … | … | … | … | … |
| **1** | $\mathbf{x_{i1}}$ | $\mathbf{x_{i2}}$ | $\mathbf{(x_{i1})^2}$ | $\mathbf{(x_{i2})^2}$ | $\mathbf{x_{i1}x_{i2}}$ | $\mathbf{y_i}$ |
| … | … | … | … | … | … | … |



  - Learn the coefficients **w** from the model $y = \mathbf{w}^T\mathbf{z}$ + noise which is equivalent to: $y = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2\ldots$

## More Non-Linear Regression: Radial Basis Functions (RBFs)

- Create features that are arbitrary "basis" functions (or kernel functions) of the input vector
  - e.g., $z_i = \textit{KernelFunction}(|x_i - c_i|/\gamma_i)$ where $c_i$ s and $\gamma_i$ s are constants to be learned
  - Learn the coefficients **w** from $y = \mathbf{w}^T\mathbf{z} + \text{noise}$

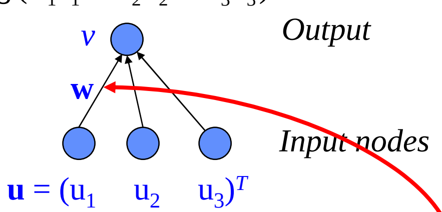



$z_i = exp(-(|x_i - c_i|/\gamma_i)^2)$



## Artificial Neural Networks: Perceptrons

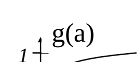
$$v = g(\mathbf{w}^T\mathbf{u})$$
$$= g(w_1u_1 + w_2u_2 + w_3u_3)$$




The most common activation function:

Sigmoid function:

$$g(a) = \frac{1}{1 + e^{-\beta a}}$$




Want to learn a mapping from inputs to outputs, given training data $(\mathbf{u}^m, d^m)$. How is **w** learned?

## Learning the Weights: Gradient Descent

- Given training examples $(\mathbf{u}^m, d^m)$ (m = 1, …, N), define an error function (cost function or "energy" function)

$$E(\mathbf{w}) = \frac{1}{2}\sum_m (d^m - v^m)^2$$

$$\text{where} \quad v^m = g(\mathbf{w}^T \mathbf{u}^m)$$



## Learning the Weights: Gradient Descent

- Would like to estimate **w** so that error $E(\mathbf{w})$ is minimized
  - Gradient Descent: Change **w** in proportion to $-dE/d\mathbf{w}$ (why?)

$$\mathbf{w} \to \mathbf{w} - \varepsilon \frac{dE}{d\mathbf{w}}$$

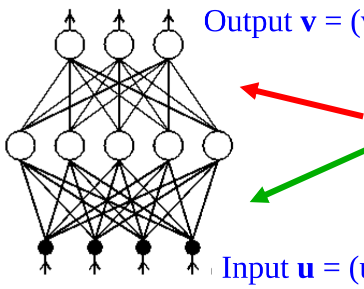
$$\frac{dE}{d\mathbf{w}} = -\sum_m (d^m - v^m)\frac{dv^m}{d\mathbf{w}} = -\sum_m (d^m - v^m) g'(\mathbf{w}^T \mathbf{u}^m)\mathbf{u}^m$$

Derivative of sigmoid

# Multilayer Networks

- One layer networks can only learn a limited class of functions. E.g., cannot learn XOR function
- To learn arbitrary functions, need multiple layers

Output $\mathbf{v} = (v_1\ v_2 \ldots v_J)^T$; Desired = $\mathbf{d}$



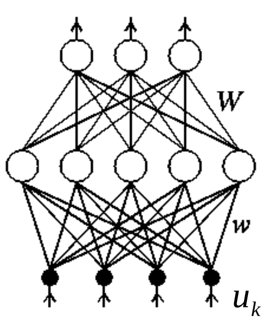
How do we learn these weights?

Input $\mathbf{u} = (u_1\ u_2 \ldots u_K)^T$



# Idea: "Backpropagation" Learning Rule

$$v_i = g(\sum_j W_{ji} g(\sum_k w_{kj} u_k))$$



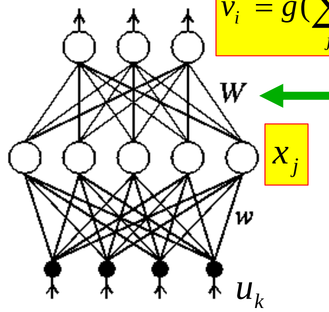

Start with random weights $\{\mathbf{W}, \mathbf{w}\}$

Given input $\mathbf{u}$, network produces output $\mathbf{v}$

Find $\mathbf{W}$ and $\mathbf{w}$ that minimize total squared output error over all output units (labeled $i$):

$$E(\mathbf{W}, \mathbf{w}) = \frac{1}{2} \sum_i (d_i - v_i)^2$$

## Backpropagation: Output Weights




$$v_i = g(\sum_j W_{ji} x_j)$$

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

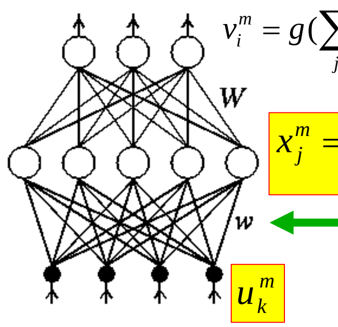
Learning rule for hidden-output weights **W**:

$$W_{ji} \to W_{ji} - \varepsilon \frac{dE}{dW_{ji}} \qquad \text{\{gradient descent\}}$$

$$\frac{dE}{dW_{ji}} = -(d_i - v_i) g'(\sum_j W_{ji} x_j) x_j$$



## Backpropagation: Hidden Weights




$$v_i^m = g(\sum_j W_{ji} x_j)$$

$$x_j^m = g(\sum_k w_{kj} u_k^m)$$

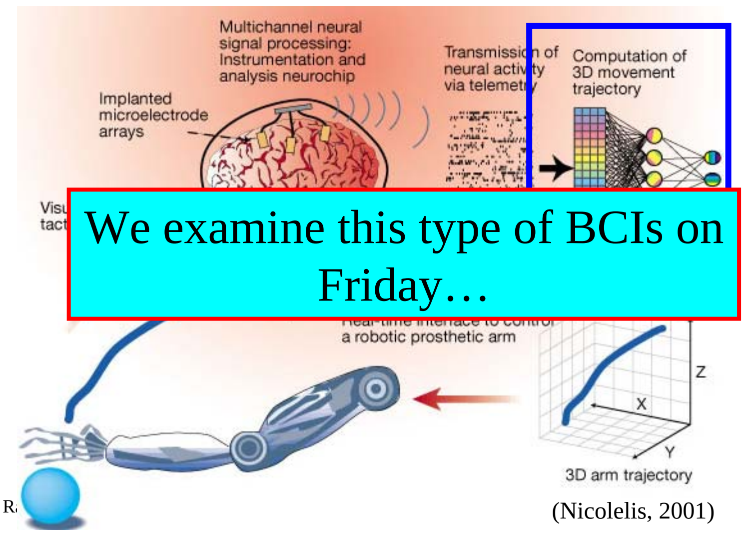
$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

Learning rule for input-hidden weights **w**:

$$w_{kj} \to w_{kj} - \varepsilon \frac{dE}{dw_{kj}} \quad \text{But}: \frac{dE}{dw_{kj}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{kj}} \qquad \text{\{chain rule\}}$$

$$\frac{dE}{dw_{kj}} = \left[ -\sum_{m,i} (d_i^m - v_i^m) g'(\sum_j W_{ji} x_j^m) W_{ji} \right] \cdot \left[ g'(\sum_k w_{kj} u_k^m) u_k^m \right]$$

# Example Application in BCI




We examine this type of BCIs on Friday…

(Nicolelis, 2001)



# Outline

- Supervised Learning: Regression
  - Linear, polynomial.
  - RBFs, perceptrons, multilayer networks.
- Supervised Learning: Classification
  - Linear classifiers, support vector machines
  - Multi-class classification
- Cross-validation
  - Model selection, preventing overfitting

Next Lecture will cover this plus Non-Invasive BCIs

See you tomorrow!