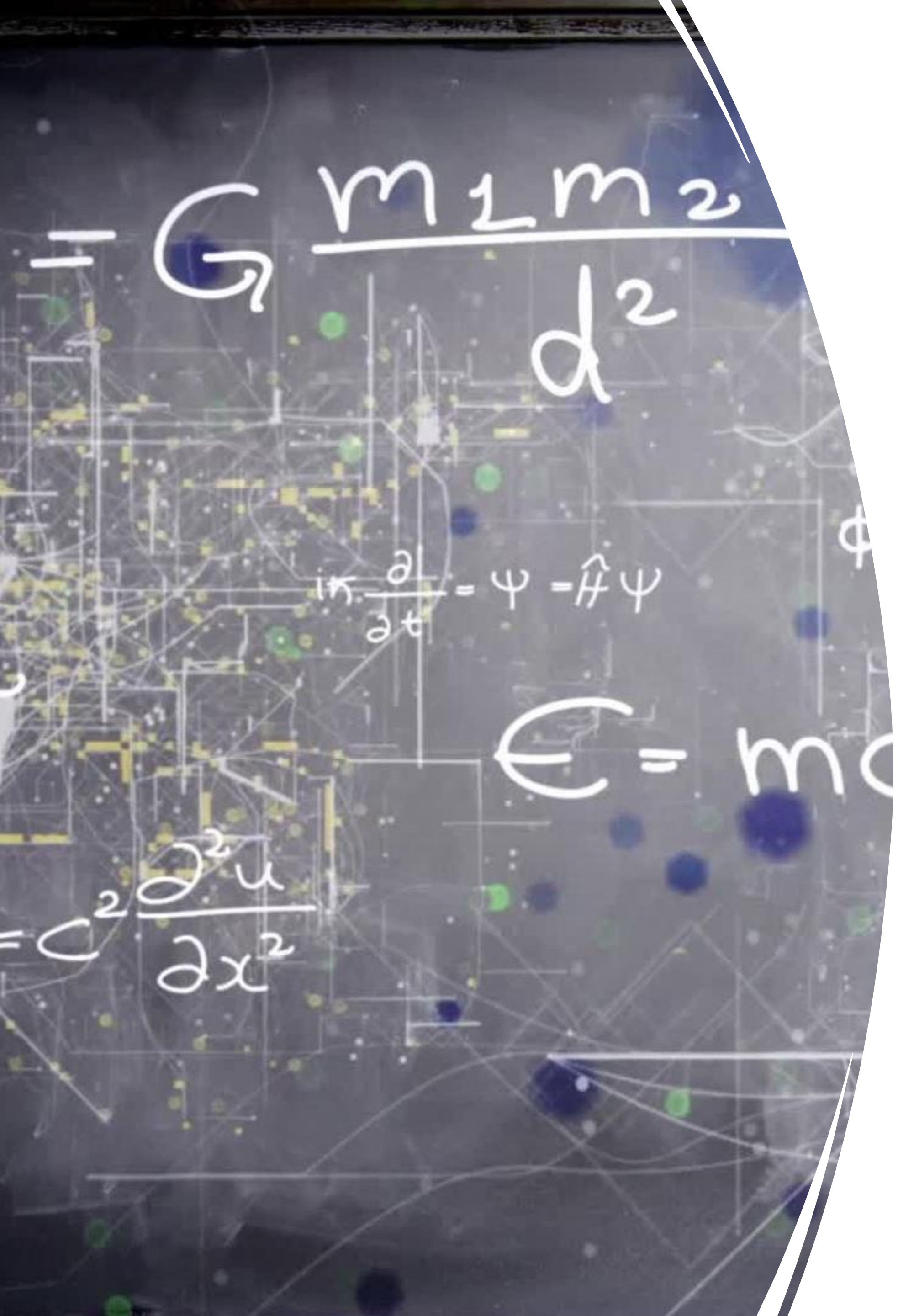


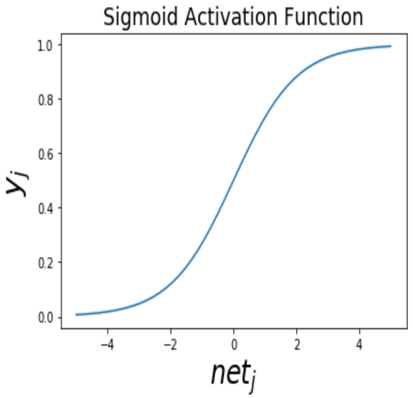
Neural Networks

Theoretical and Mathematical Foundations

CSCI 4850/5850

Rate Coding Activation Functions

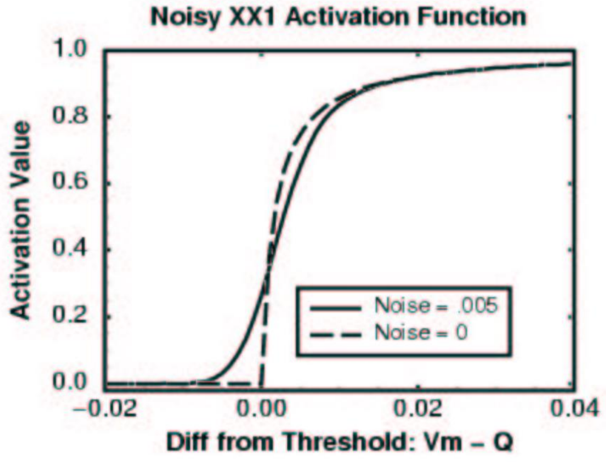
- Even though action potentials are discrete events, they are often summarized together as a firing rate
- Zero means no APs and one means firing at the maximum possible rate
- A little noisy as well…



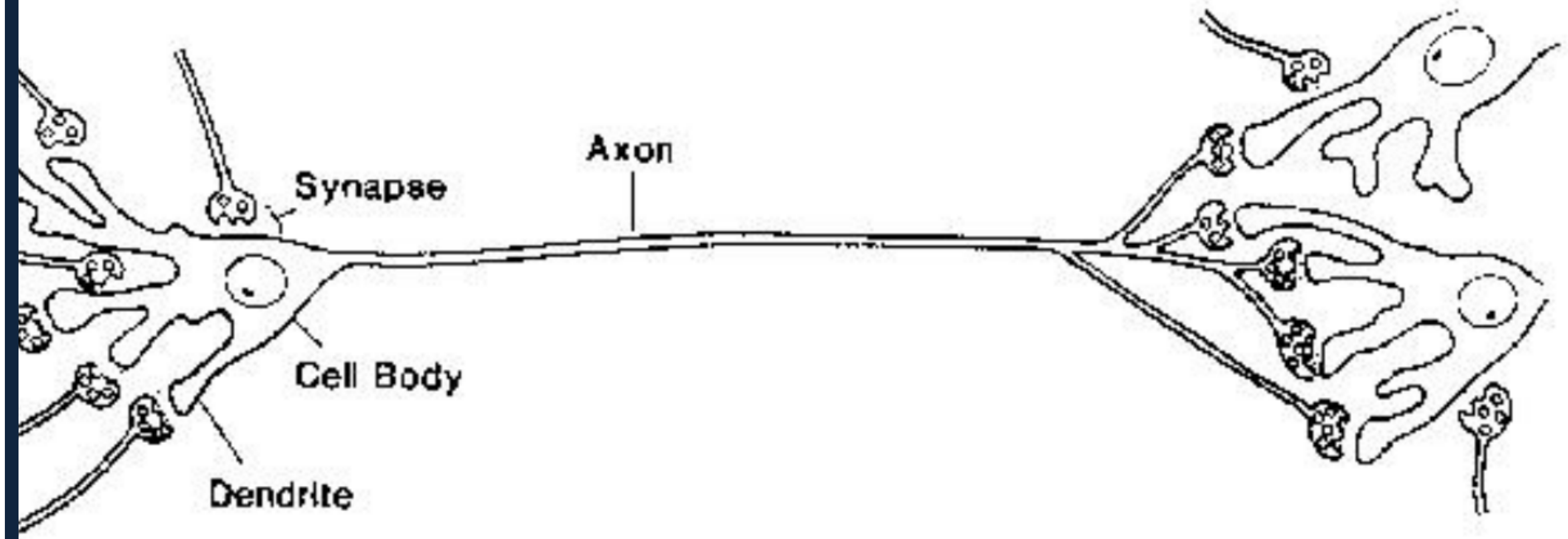
$$y = \frac{1}{1 + x^{-1}}$$

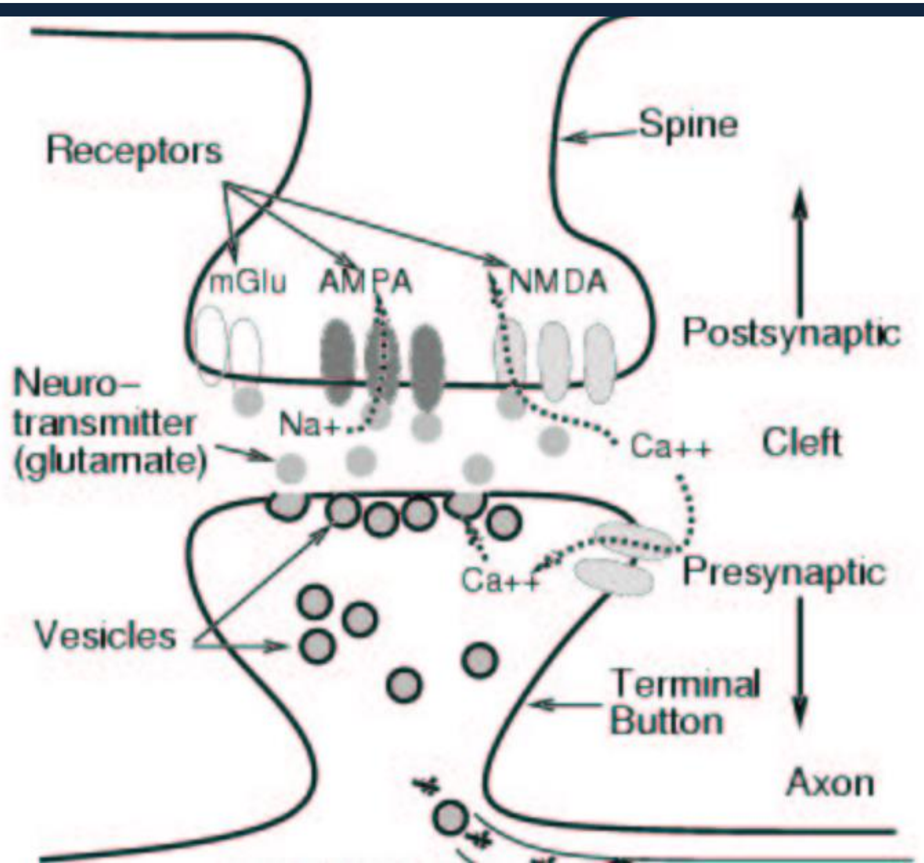
Output rate

Total input



Memory



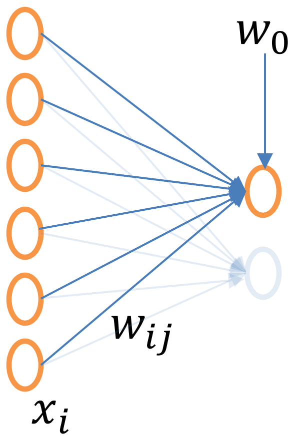


- Information or memory is stored in the connections between neurons (synapses)
- Summarized as the “weight” of the connection (large/small connections)

Abstracting Details: Net Input

- The *net input* for unit i can be “folded” into a single value, net_j, which weights the activation of the sending neuron x_i by the strength of the synaptic connection between i and j, w_{ij}.
- The sum of all weighted activations into a unit plus a *bias weight*, w_0, is the total net input.

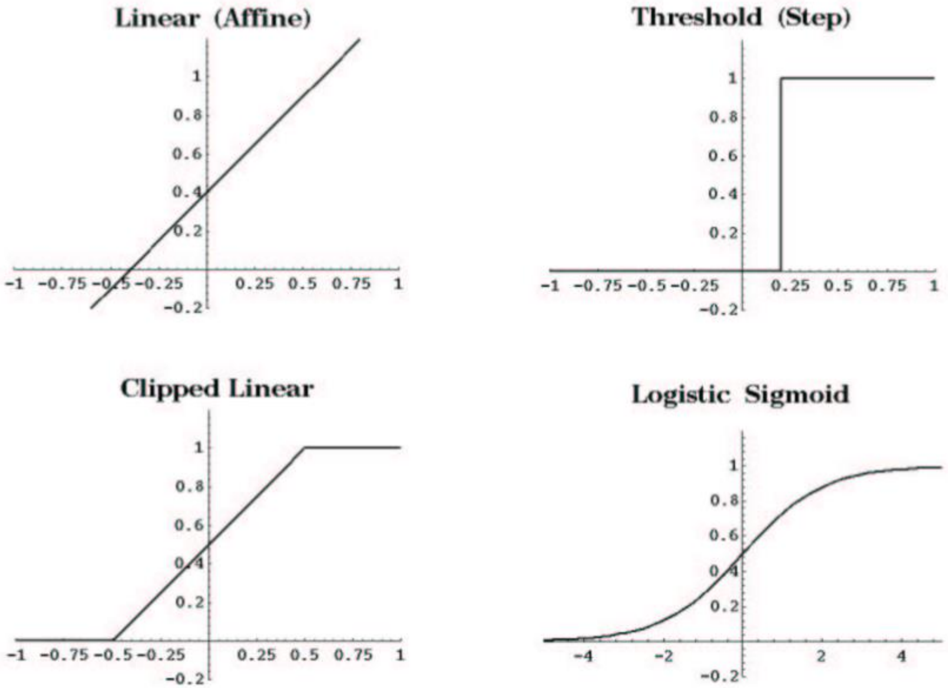
$$net_j = w_0 + \sum_{i=1}^{n} x_i w_{ij}$$

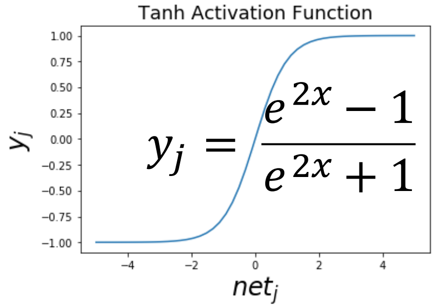


Abstracting Details: Activation Functions

- The *net input* can then be transformed into a *rate code* using an *activation function*
- A commonly used function is the logistic sigmoid:

$$f(net_j) = \frac{1}{1 + e^{-net_j}}$$





Simple Hardwired ANN

Vectors

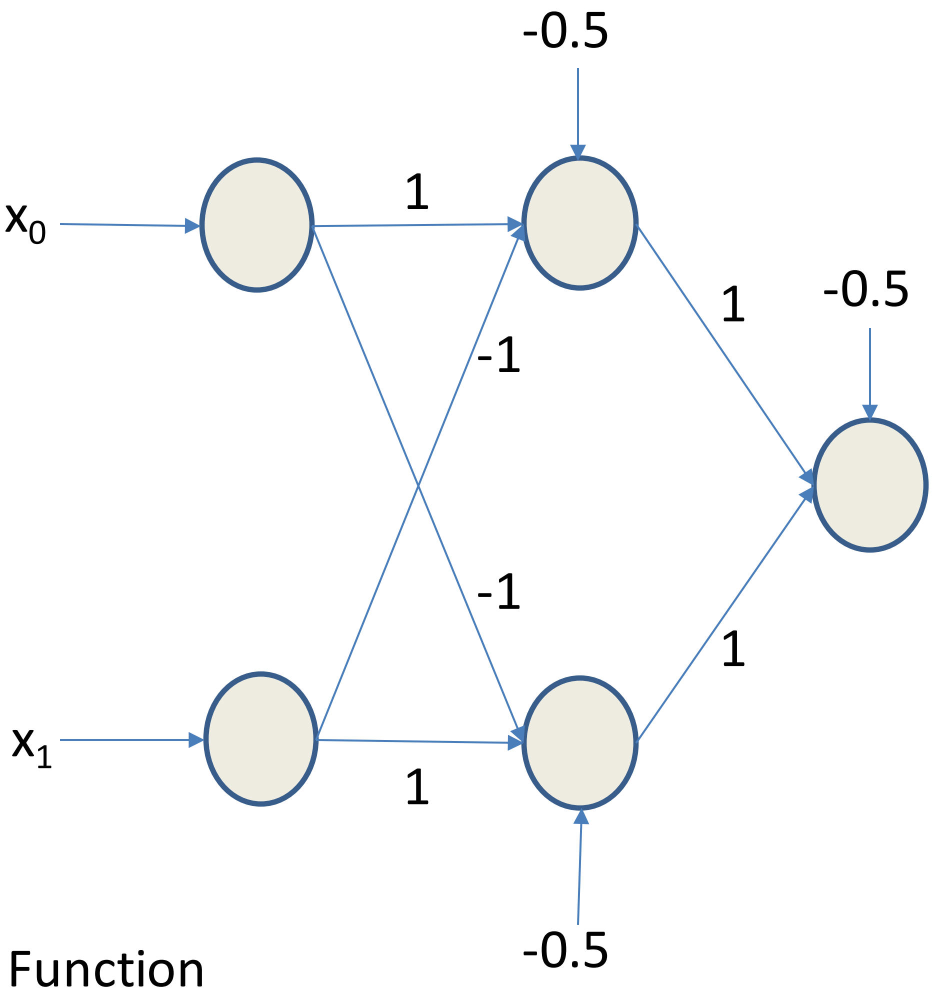
X -> F(X)

0 0 -> ?

0 1 -> ?

1 0 -> ?

1 1 -> ?



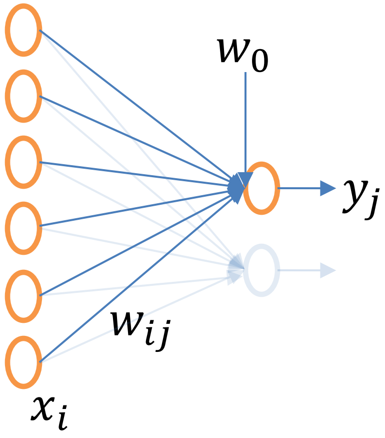
Threshold Activation Function

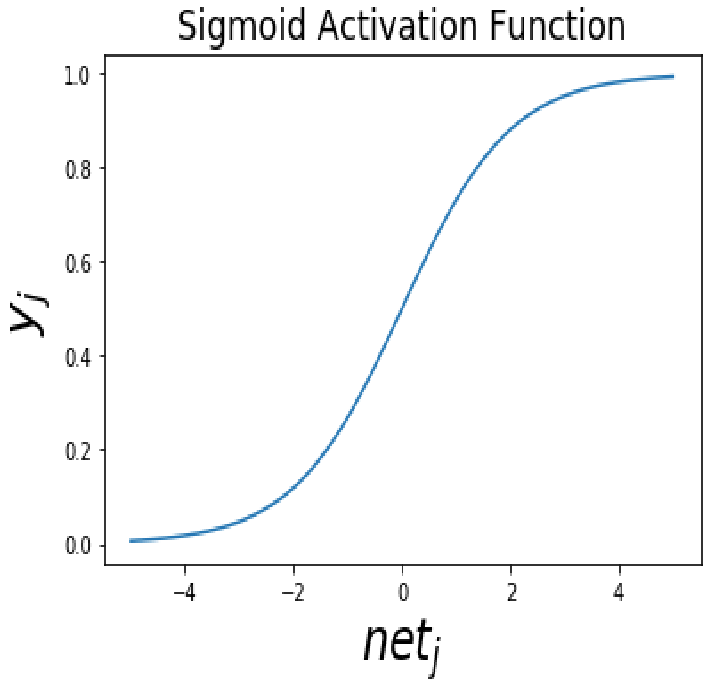
f(net)=1 if net > 0

f(net)=0 if net <= 0

Together: Simple Neural Units

$$net_j = w_0 + \sum_{i=1}^{n} x_i \, w_{ij}$$



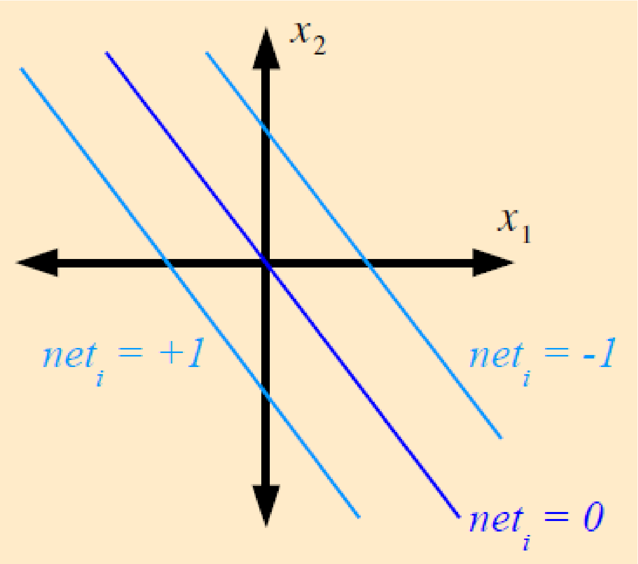


$$y_j = \frac{1}{1 + e^{-net_j}}$$

What does Net Input mean?

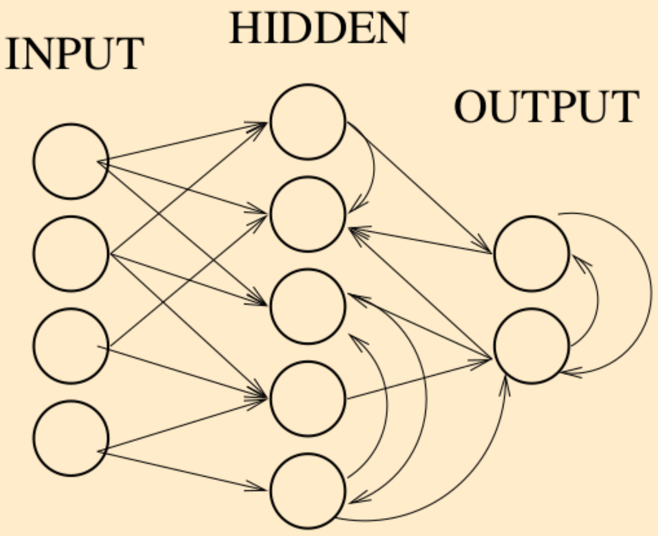
- The change in postsynaptic membrane potential is reflected in the *net input*
- Net input is an **affine function** of the presynaptic cell activity (input)
    - In the field of machine learning, we often refer this as a *linear* relationship even though it's technically affine (due to the bias weight, w_0)
- Geometrically, there is a **region** where the net input is *positive* and a **region** where the net input is *negative* which is separated by the n-dimensional **hyperplane** where the net input is zero…

$$net_j = w_0 + \sum_{i=1}^{n} x_i w_{ij}$$

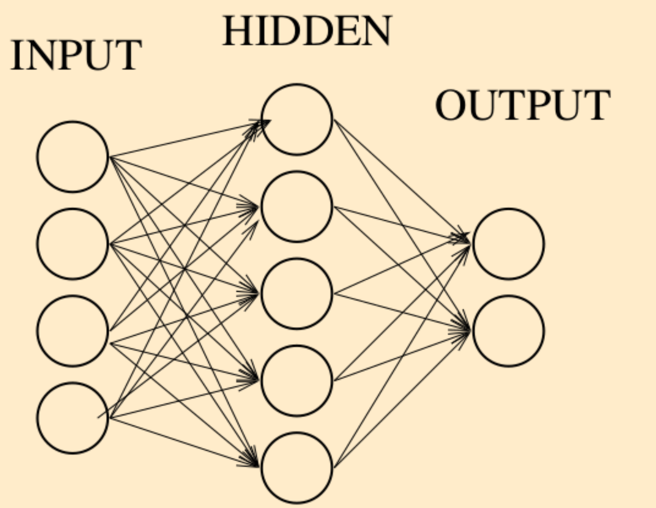


Abstracting Details: Architecture

- ANNs typically consist of input, output, and hidden layers of artificial neural units
- May be “single layer” or “multi-layer”, but the pattern of connections is called the ANNs *architecture*

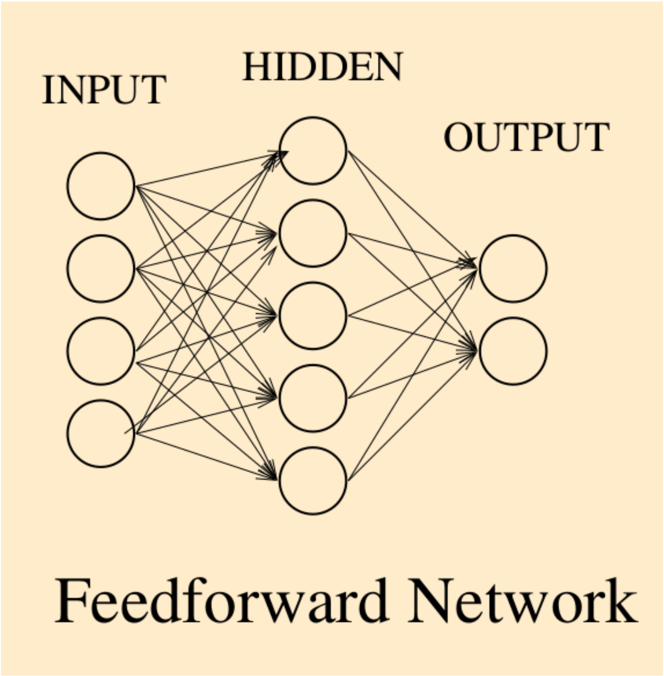


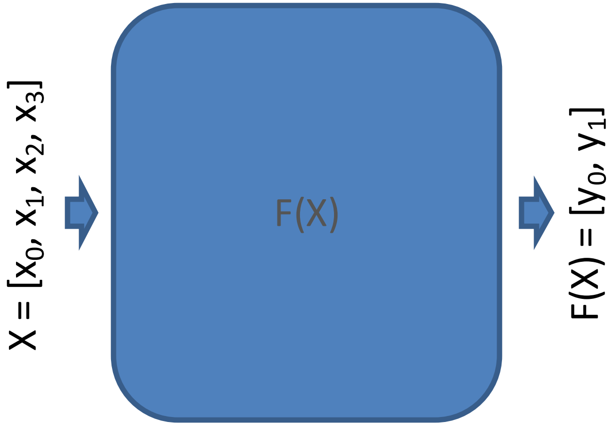
Recurrent Network



Feedforward Network

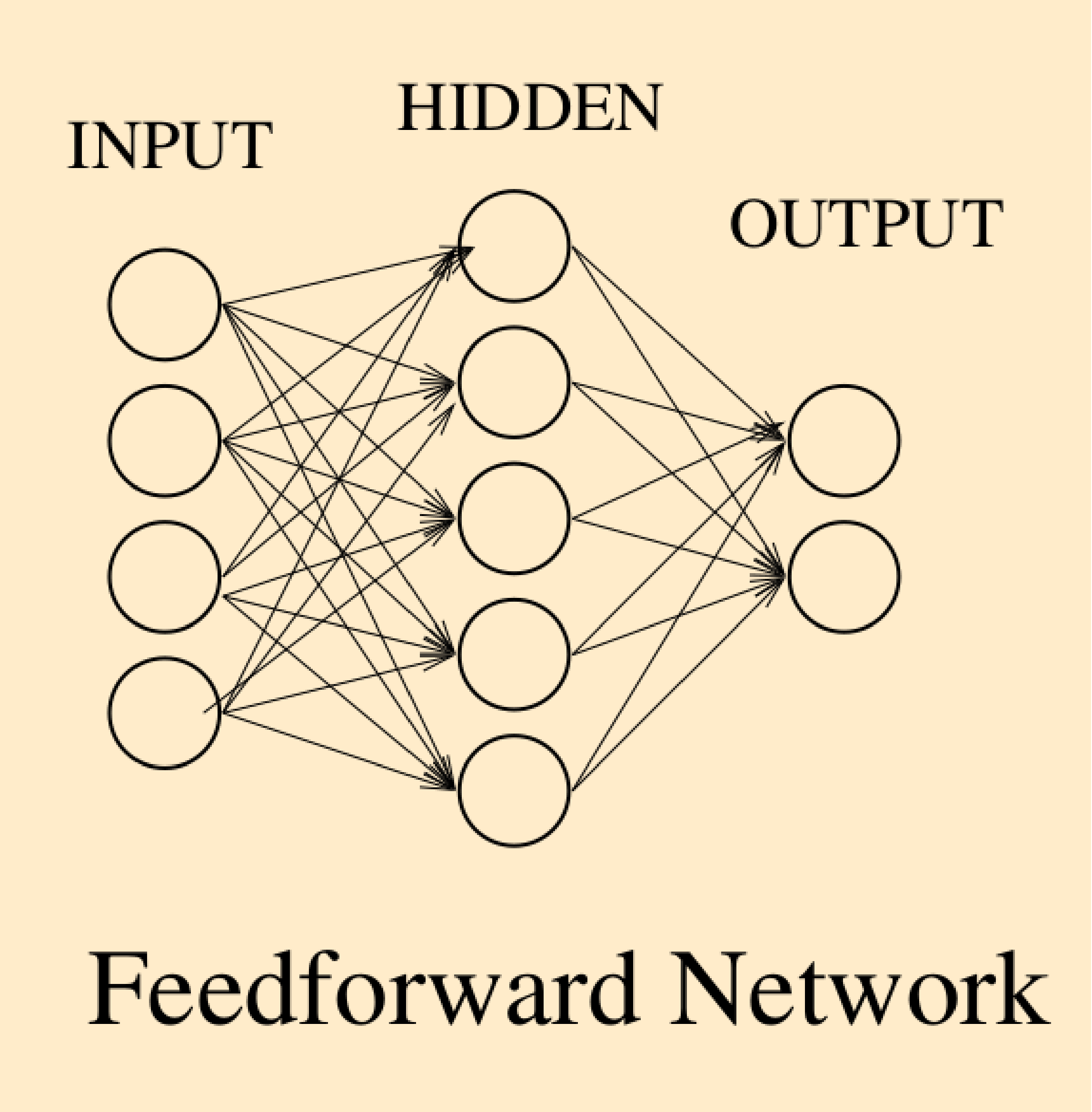
Standard Artificial Neural Network: Feed-forward Networks





Universal Approximation

- It has been proven that given *enough* units and using only a single layer of hidden units, essentially *any* function can be approximated to an arbitrary degree of precision
- Hornik, Stinchcombe, and White, 1989



Knowledge Retention/Representation

- Neural networks typically utilize 2 kinds of *memory* for storing or processing information:
  - **Weight-based**: long-term, non-volatile stored in the synaptic connections between units
  - **Activation-based**: short-term, volatile stored in the activation values of the neural units

Learning

Notice that most of the information encoded by a feed-forward network is in the connection (and bias) weights

So, the question becomes, how does one calculate the weights?

Hand-wired?

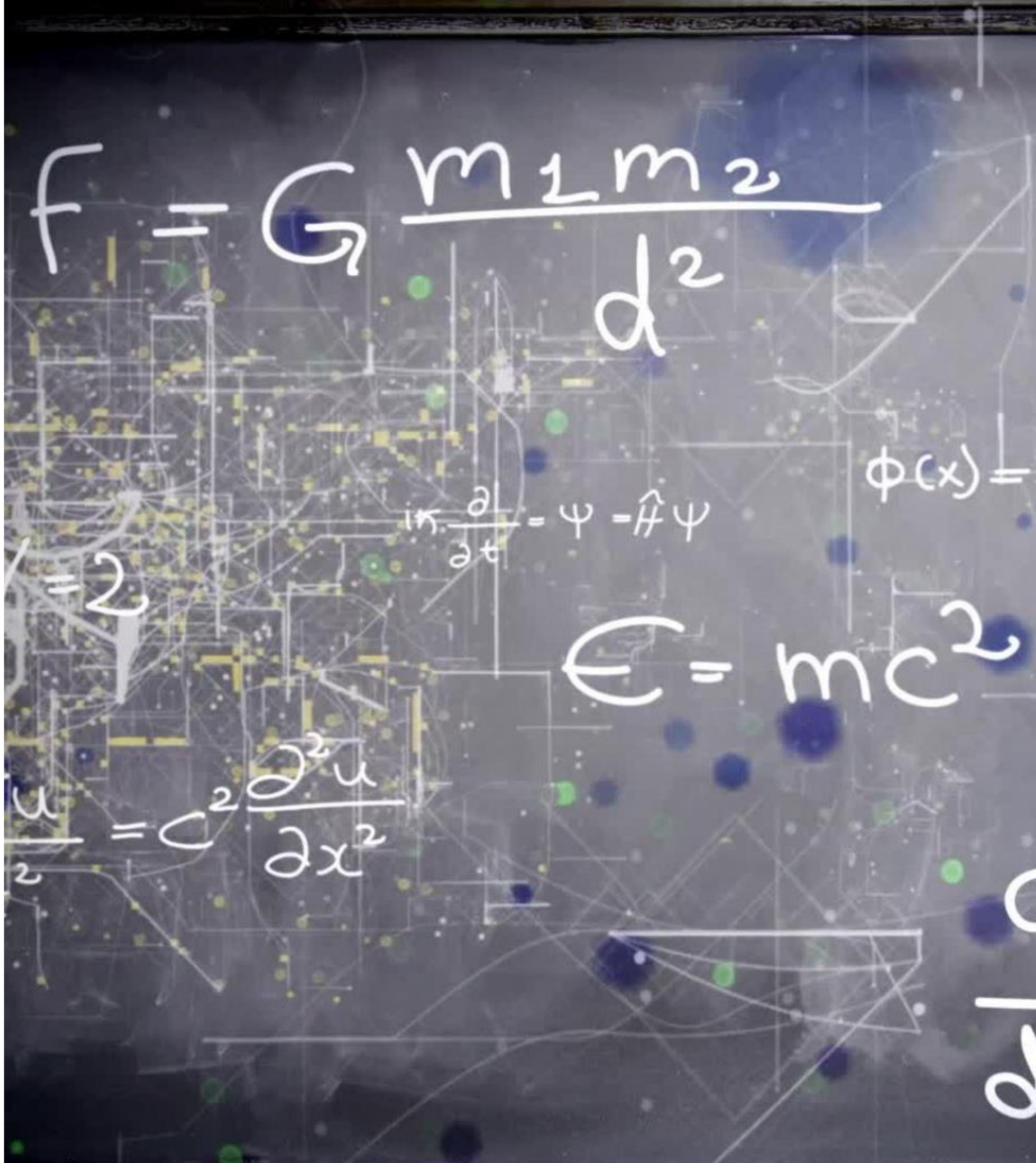
Trial-and-error?

Random generation?

Hmmm…

Foundation: Calculus

- Learning often involves making **incremental changes** to a system, and **calculus is the mathematics of change**.
- Activation levels in **recurrent** neural architectures are naturally **dynamic**, and calculus is the foundational mathematics of **dynamical systems**.



The Derivative

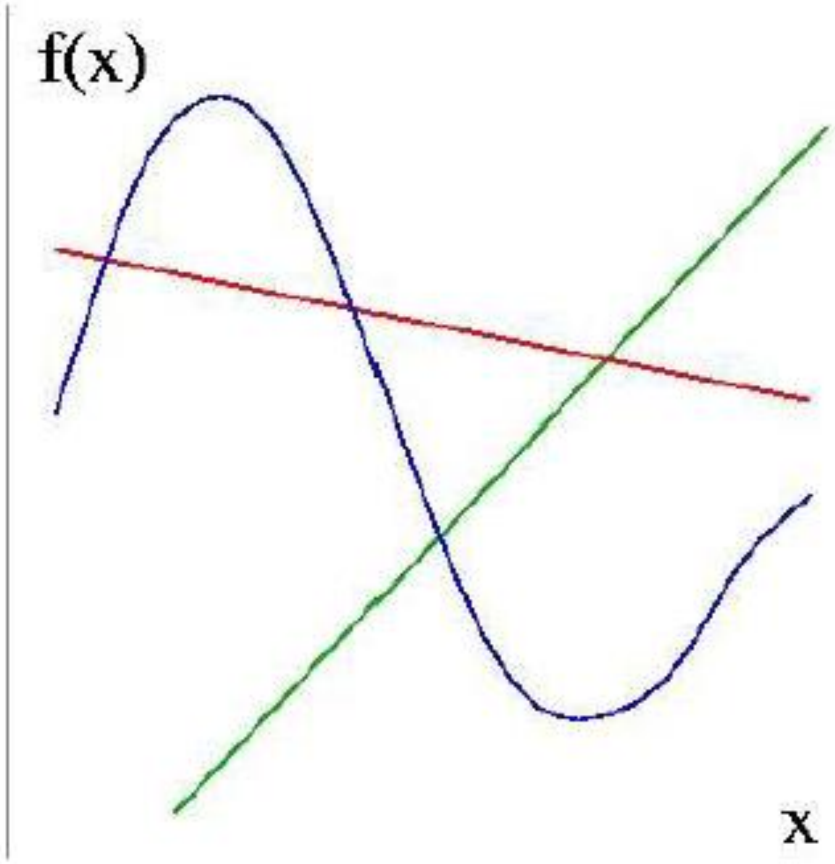
- A function is ***differentiable*** if its first partial derivative exists everywhere in its domain for each of its arguments. A function is ***smooth*** if all partial derivatives exist.

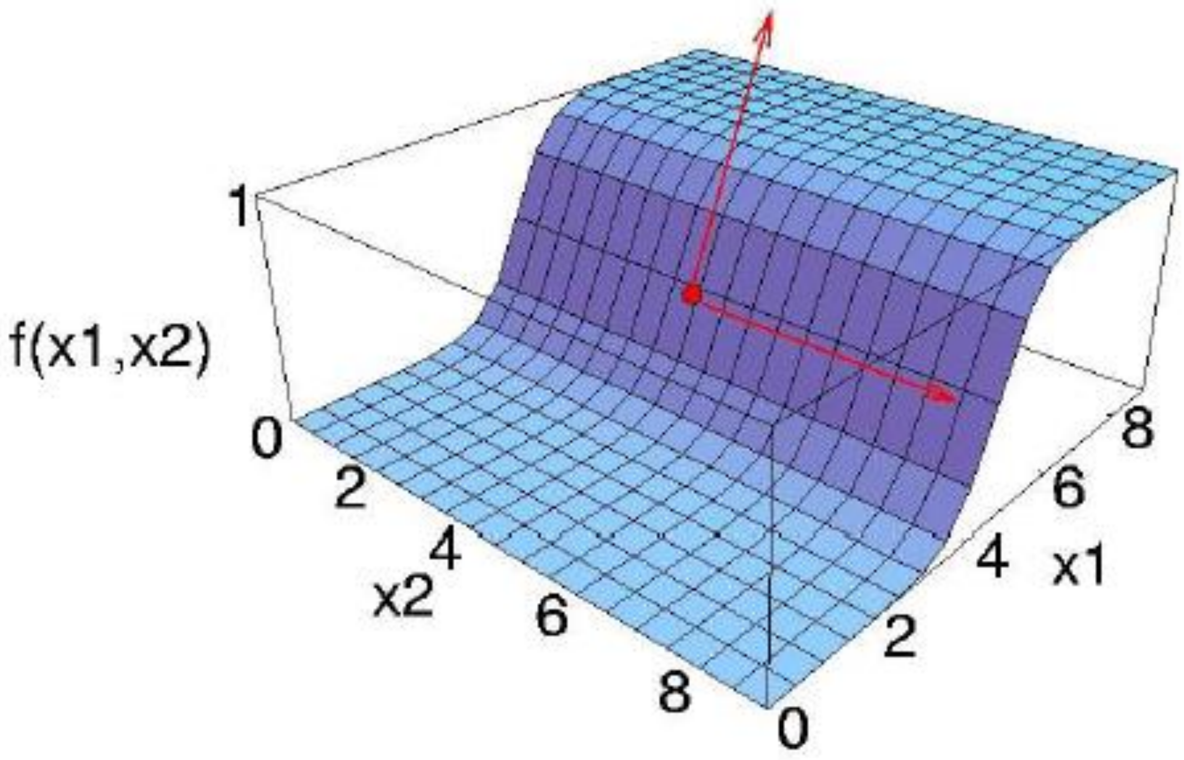
$$f'(x) = \frac{d f(x)}{d x} = \lim_{\epsilon \to 0} \frac{\left(f(x+\epsilon) - f(x)\right)}{\epsilon}$$

$$f''(x) = \frac{d}{d x}\left(\frac{d f(x)}{d x}\right)$$

$$\frac{\partial f(x_1, \ldots, x_i, \ldots, x_n)}{\partial x_i} = \lim_{\epsilon \to 0} \frac{\left(f(x_1, \ldots, x_i + \epsilon, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)\right)}{\epsilon}$$

Geometric Interpretation of Derivatives





The Chain Rule

$$\frac{d\ f(g(x))}{d\ x} = \frac{d\ f(u)}{d\ u}\frac{d\ g(x)}{d\ x}$$

Also works for partial derivatives…

$$f(x) = f(s(x), t(x))$$

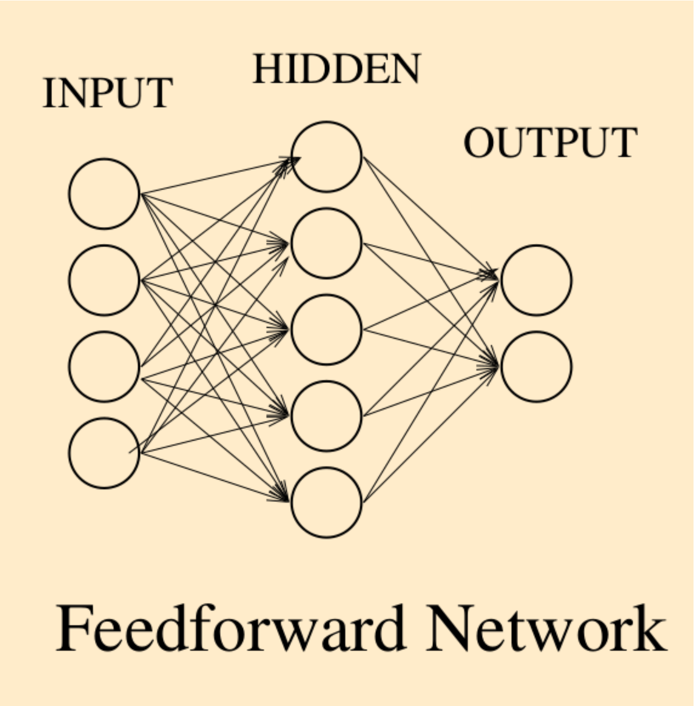
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial s}\frac{\partial s}{\partial x} + \frac{\partial f}{\partial t}\frac{\partial t}{\partial x}$$

$$f(x) = f(g_1(x), g_2(x), \ldots, g_n(x))$$

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{n}\frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial x}$$

Linear Algebra

- Neural units are usually grouped into layers, so it's more natural to group variable values for all units in a layer into **vectors**.
- Connection weights to a single unit can be grouped into a **weight vector**, but when layers are completely interconnected, it makes more sense to group them into a **weight matrix**.

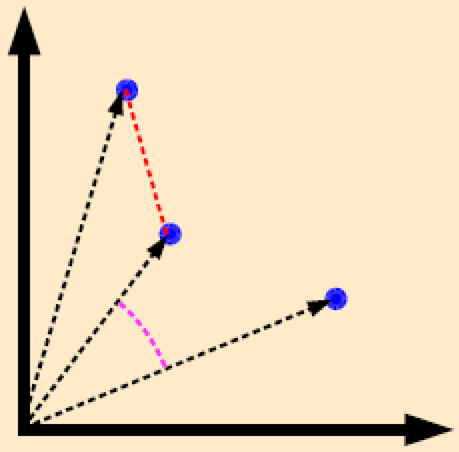


$$\boldsymbol{net} = [net_1, net_2, net_3, net_4, net_5]$$
$$\boldsymbol{act} = [f(net_1), f(net_2), f(net_3), f(net_4), f(net_5)]$$

Linear Review

- A **vector** is similar to a 1-D array, but geometrically we think of the vector as a **point** in an ***n*-dimensional** vector **space**, where *n* is the length of the vector.
- Concepts:
  - Euclidean Distance
  - Cosine Similarity
  - Othogonality
  - Orthonormality
  - Linear Independence

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = (x_1, x_2, x_3, x_4)$$



Vector Arithmetic

- Multiply a vector by a scalar

$$a\boldsymbol{x} = (ax_1, ax_2, \ldots, ax_n)^{\mathrm{T}}$$

- Adding two vectors

$$\boldsymbol{x} + \boldsymbol{y} = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n)^{\mathrm{T}}$$

- Length of a vector (magnitude, norm)

$$|\boldsymbol{x}| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Vector Distances

- Euclidean distance between two vectors

$$|\boldsymbol{x} - \boldsymbol{y}| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

- Dot product (or inner product) of two vectors

$$\boldsymbol{x} \cdot \boldsymbol{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

- Angular distance between two vectors

$$\boldsymbol{x} \cdot \boldsymbol{y} = |\boldsymbol{x}||\boldsymbol{y}| \cos\theta$$

- Cosine distance between two vectors

$$1 - \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{|\boldsymbol{x}||\boldsymbol{y}|}$$

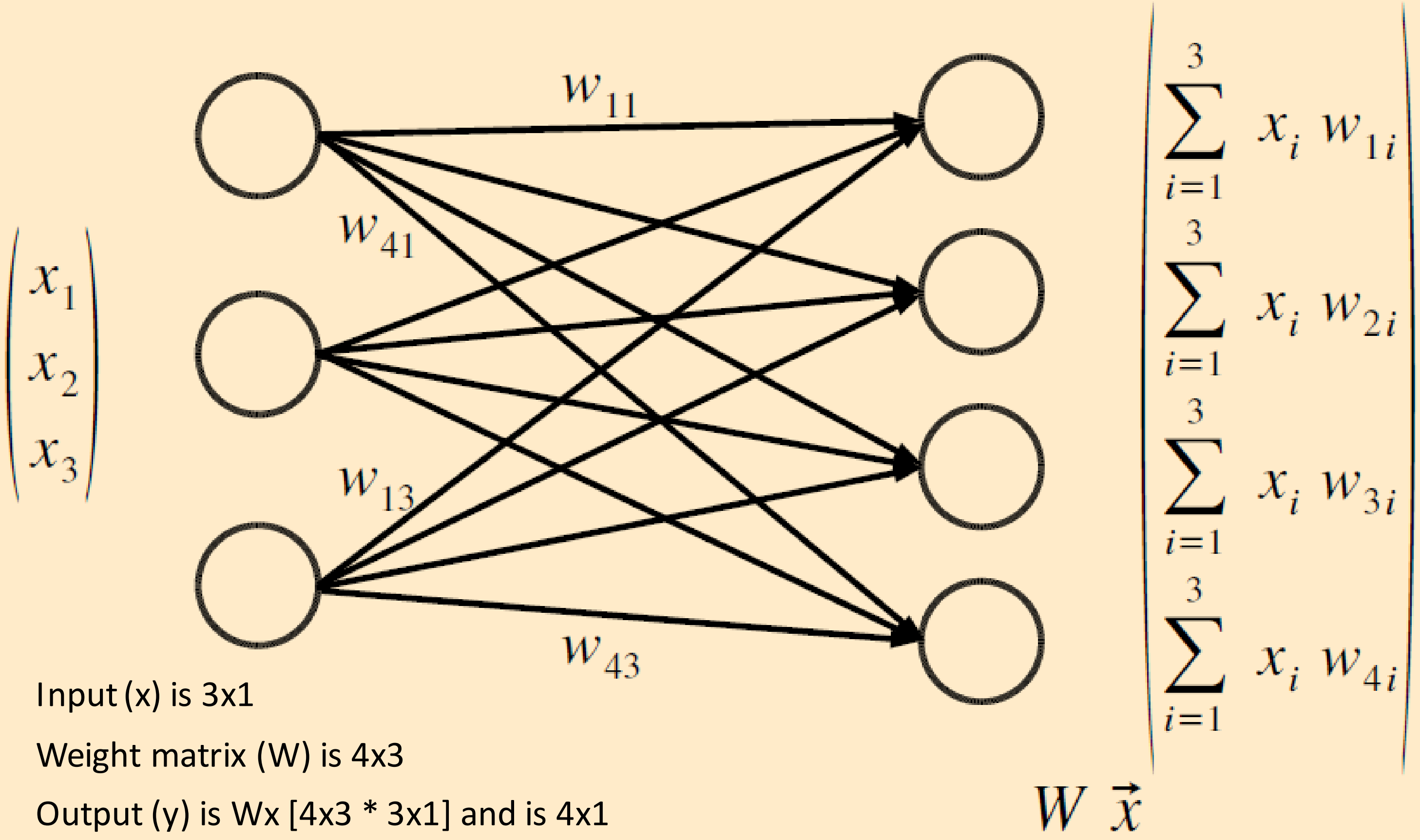
Matrices

- A **matrix** is similar to a 2-D array
- Multiplying a *vector* by a *matrix* produces another **vector**…

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix}$$

$$W \vec{x} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 w_{11} + x_2 w_{12} + x_3 w_{13} \\ x_1 w_{21} + x_2 w_{22} + x_3 w_{23} \\ x_1 w_{31} + x_2 w_{32} + x_3 w_{33} \\ x_1 w_{41} + x_2 w_{42} + x_3 w_{43} \end{pmatrix}$$

Weight Matrices



Input (x) is 3x1

Weight matrix (W) is 4x3

Output (y) is Wx [4x3 * 3x1] and is 4x1

Alternatively: $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}} = \boldsymbol{y}^{\mathrm{T}}$

Matrix Multiplication

$$AB = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

$$= \begin{pmatrix} (a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}) & (a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}) \\ (a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}) & (a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}) \\ (a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}) & (a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}) \\ (a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31}) & (a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32}) \end{pmatrix}$$

Shapes:
4x3 * 3x2 = 4x2

Special Matrices

- The **identity matrix** (**I**) is zero everywhere except along its main diagonal, where it is exactly 1. This means that for a square matrix, A:

$$\boldsymbol{AI = A}$$

- Some square matrices have an **inverse**, such that:

$$\boldsymbol{AA^{-1} = A^{-1}A = I}$$

- Every matrix has a **transpose**, which swaps the rows and columns of the original matrix:

$$\boldsymbol{A}^{\mathrm{T}}$$

- A **symmetric** matrix is any matrix which is equal to its own transpose:

$$\boldsymbol{A = A}^{\mathrm{T}}$$

The Gradient and The Hessian

- For a function of *n* variables, the **gradient** is a *vector* of the form:

$$\frac{d\, f(\vec{x})}{d\, \vec{x}} = \nabla_{\vec{x}}\, f(\vec{x}) = \left(\frac{\delta\, f}{\delta\, x_1}, \frac{\delta\, f}{\delta\, x_2}, \ldots, \frac{\delta\, f}{\delta\, x_n} \right)^T$$

- For a function of *n* variables, the **Hessian** is *matrix* with each element (i,j) is:

$$\frac{\partial}{\partial x_i} \left(\frac{\partial f(\vec{x})}{\partial x_j} \right)$$

Matrix Factorization (aka Decomposition)

- Eigen decomposition (square)
  - The nonzero vector, **x**, is an eigenvector of the matrix **A**, with corresponding eigenvalue, λ, if and only if:
$$\boldsymbol{Ax} = \lambda \boldsymbol{x}$$
  - Alternatively, Eigen decomposition factors the square matrix **A** as follows:
$$\boldsymbol{A} = \boldsymbol{Q\Lambda Q}^{\mathrm{T}}$$
  - **Q** is the square matrix of eigen vectors (one in each column), and **Λ** is the diagonal matrix of eigen values
- Singular value decomposition (rectangular)
  - The singular value decomposition of a matrix is:
$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^{\mathrm{T}}$$
  - Where **Σ** is a rectangular diagonal matrix of singular values, and **U** and **V** are the left-singular and right-singular vectors of **A**, respectively.

Probability Theory

There is often *uncertainty* in the reliability of network inputs

There is often *uncertainty* in what to do with a given set of inputs

There is often *uncertainty* about our neural network performance

We often construct networks using noisy components, introducing *uncertainty* in behavior

Probability, information, and statistical theory provide mathematical formalisms for handling *uncertainty*

Probability

- Probabilities are defined over a space of **events**.
  - Each **atomic event** has a **probability** assigned to it in the range [0,1]
  - The sum of **all** atomic event probabilities is 1
  - Events are discrete (happen / don’t happen)
  - Mutually exclusive and exhaustive
- A **discrete random variable** takes on one of a finite set of values depending on the relative probability of those values
- A **continuous random variable** takes on a real value in some (potentially infinite) range, events are defined by infinitesimal subranges.

Concepts in Probability Theory

- Probability Distribution $P(X)$
- Joint Probability Distribution $P(X_1, X_2, \ldots, X_n)$
- Unconditional Probability $P(X = x_i)$ or $P(x_i)$
- Conditional Probability $P(a \mid b) = P(a \wedge b)/P(b)$
- Bayes' Rule $P(a \mid b) = \dfrac{P(b \mid a)P(a)}{P(b)}$
- Prior Probability $P(hypothesis)$
- Posterior Probability $P(hypothesis \mid evidence)$
- Likelihood $P(evidence \mid hypothesis)$

Common Statistics

- **Expected value** of a discrete random variable (commonly known as the mean or average):

$$E[x] = \mu_x = \sum_{i=1}^{n} x_i P(x = x_i)$$

- **Variance** of such a variable:

$$\mathrm{var}[x] = \sigma_x^2 = \sum_{i=1}^{n} (x_i - E[x])^2 P(x = x_i)$$

- **Standard deviation** is the square-root of the variance

$$\sigma_x = \sqrt{\mathrm{var}[x]} = \sqrt{\sigma_x^2}$$

Covariance

- Elements within a vector may influence one another. A measure of the relationship between elements is the covariance:

$$\mathrm{cov}[x_i, x_j] = \sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)]$$

- The covariance matrix is the square matrix whose (i,j) element is $\mathrm{cov}[x_i, x_j]$.