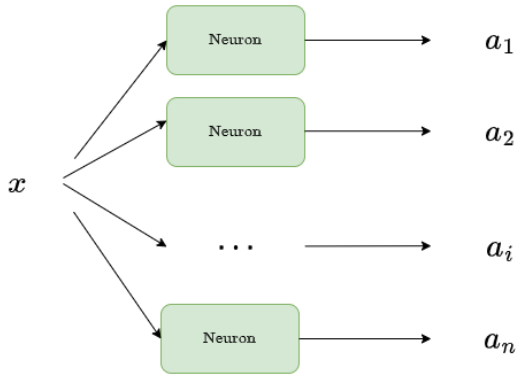
# Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2022

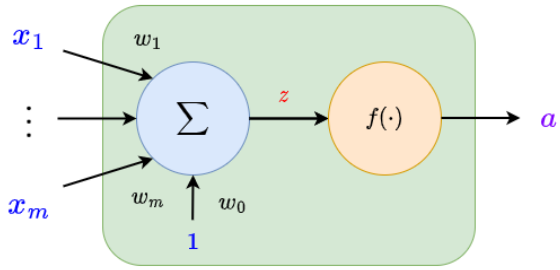
## Single Layer: Visualizing our Components

Now, rather than analyzing a single neuron, we will analyze a single **layer**.




Our first layer.

In order to **analyze** this layer, we have to open back up the **abstraction**:

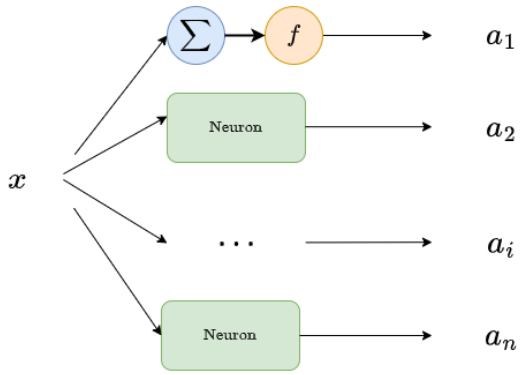



Each of those neurons looks like this.

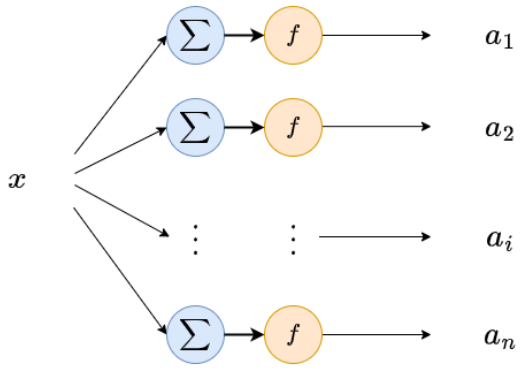
There are two important pieces of **information** we're hiding:

- We have two components inside of our neuron.
- We have many inputs $x_i$ for one neuron.

The first piece of information is easier to visualize: we just replace each neuron with the two components.



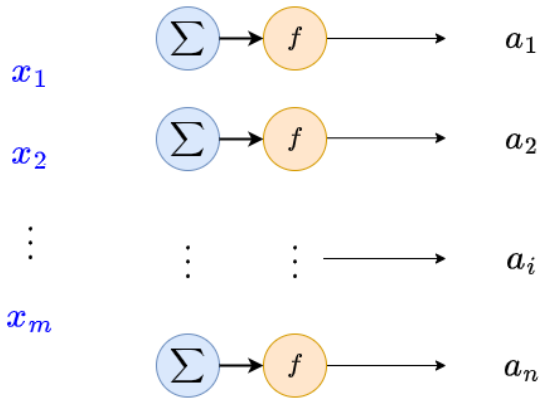

Replacing one neuron...



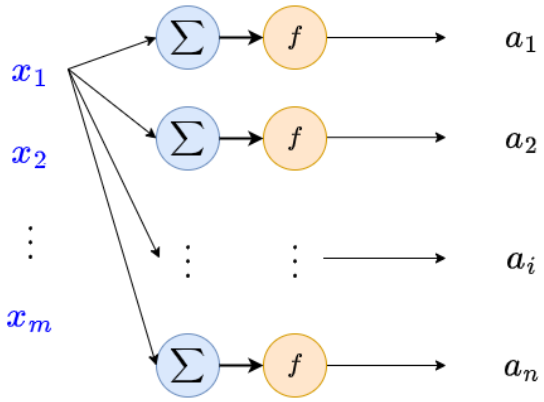

Replacing all neurons!

## Single Layer: Visualizing our Inputs

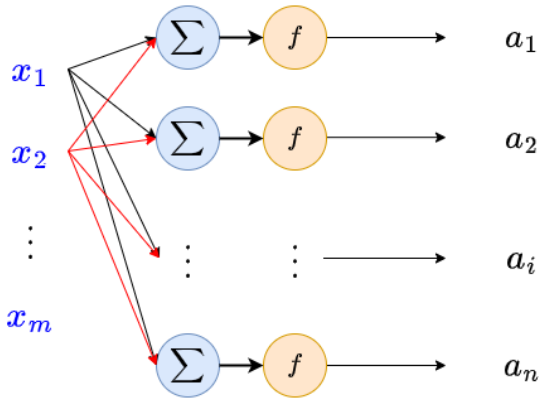
The second piece of information is much more difficult: we show all of the $x_i$ outputs.



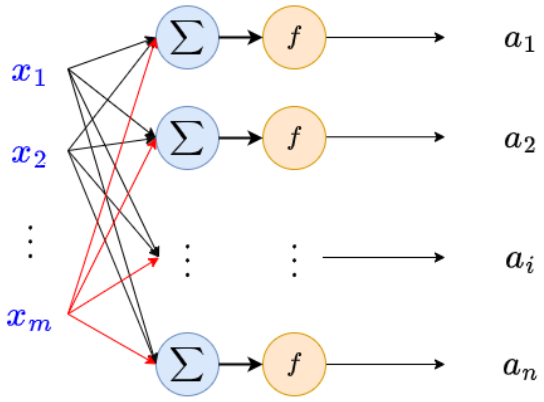

Now we have to draw the arrow for each input.




Every neuron receives the first input.




Every neuron receives the second input, too. This is getting messy...




The completed version: this is hard to look at.

Don't worry if this looks **confusing**! It's natural for it to be **hard** to read: the only thing you need to know is that we pair **every** input with **every** neuron.

This is our **final** view of this layer: because each of our $m$ inputs has to go to every of $n$ neurons, we end up with $mn$ different **weights**.

This is a ton of **information**, and its only one layer! This shows how **complex** a neural network can be, just by **combining** simple neurons.

Note that this is a **fully connected** network: not all networks are FC.

**Definition 1**
A layer is **fully connected** if every neuron has the **same input vector**.

**Example:** If one of our neurons **ignored** $x_1$, but the others did **not**, the layer would not be **fully connected**.

## Dimensions of a layer

Now that we've seen the **full** view, we can **analyze** it. Our goal is to create a more **useful** and **accurate** simplification.

Our first point: note that the input and output have a **different** dimensions!

**Clarification 2**
A **layer** can have a different **input** and **output** dimension. In fact, they are completely **separate** variables.

This is because **every** input variable is allowed to be applied to the **same** neuron:

**Example:** You can have one neuron of the form

$$z = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + w_0$$

In this case, our neuron has **one** output variable $f(z)$, but **three** inputs $x_1, x_2, x_3$.

Thus, our output dimension has been separated from our input dimension. Instead, it is the number of neurons.

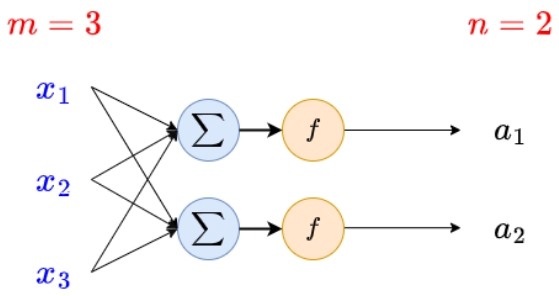
So, in general, we can say:

**Notation 3**

A **layer** has two associated **dimensions**: the **input** dimension $m$ and the **output** dimension $n$.

- The **input** dimension $m$ is based on the vector output from the **previous layer**: $x \in \mathbb{R}^m$
- The **output** dimension $n$ is equal to the **number of neurons** in the **current** layer: $A \in \mathbb{R}^n$

**Example:** Suppose you have an **input** vector $x = [x_1, x_2, x_3]$ and two **neurons**. The dimensions are $m = 3$, and $n = 2$.




The input dimension and output dimensions are **separate**.

## The known objects of our layer

So, we know we have two objects so far:

- Our **input** vector $x \in \mathbb{R}^m$
- Our **output** vector $A \in \mathbb{R}_n$

Where they each take the form

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad A = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{1}$$

But, there are a couple other things we haven't **generalized** for our entire **layer**:

- Our weights
- Our offsets
- Our preactivation

## The other variables of our layer: weights and offsets

First, our **weights**: each neuron has its own vector of weights $w \in \mathbb{R}^m$.

The dimension needs to match $x$ so we can compute $w^T x$.

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \tag{2}$$

To distinguish them from each other, we'll represent the $i^{th}$ neuron's weights as $\vec{w}_i$.

$$\vec{w}_i = \begin{bmatrix} w_{1i} \\ w_{2i} \\ \vdots \\ w_{mi} \end{bmatrix} \tag{3}$$

Each needs to be used to **compute** $a_i$, but having so many objects is annoying.

Remember that, when we had **multiple** data points $x^{(i)}$, we worked with them at the **same time** by stacking them in a **matrix**. Let's do the same here:

$$W = \overbrace{\begin{bmatrix} \vec{w}_1 & \vec{w}_2 & \cdots & \vec{w}_n \end{bmatrix}}^{\text{Each neuron has a weight vector}} \tag{4}$$

If we expand it out, we get a full matrix...

$$W = \overbrace{\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m1} & \cdots & w_{mn} \end{bmatrix}}^{n \text{ neurons}} \Bigg\} m \text{ inputs} \tag{5}$$

This is our **weight matrix** $W$: it's an $(m \times n)$ matrix. It contains all of our $mn$ weights, sorted by

- **Input variable** (row)
- **Neuron** (column)

We can do this for our **offsets** too: thankfully, there is only **one** offset per neuron, so we can write:

$$W_0 = \left.\begin{bmatrix} w_{01} \\ w_{02} \\ \vdots \\ w_{0n} \end{bmatrix}\right\} \text{Each neuron has an offset} \tag{6}$$

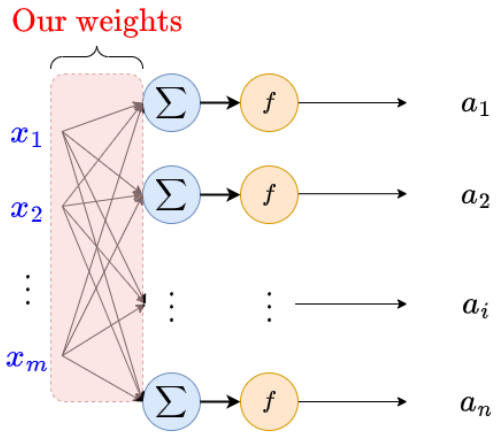
This is our offset vector, with the shape $(n \times 1)$.

**Notation 4**

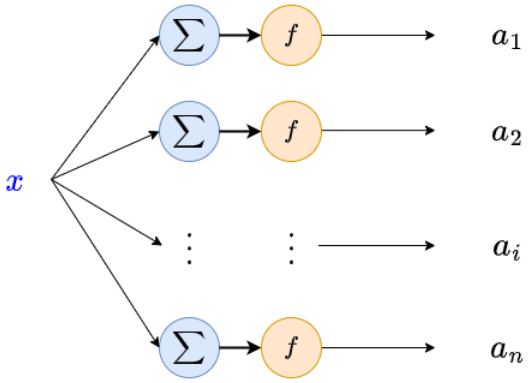
We can store our **weights** and **offsets** as **matrices**:

- **Weight** matrix $W$ has the shape $(m \times n)$
- **Offset** matrix $W_0$ has the shape $(n \times 1)$

These matrices give us a tidy way to understand all of this mess:




Now that we understand it, we'll **hide** those weights again, for readability.



## Pre-activation

Now, all that remains is the pre-activation $z$.

Before, we did

$$w^T x + w_0 = z \tag{7}$$

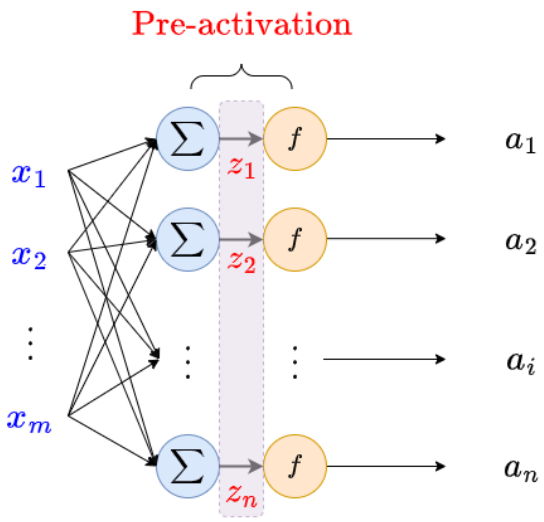
Because we so carefully kept our weights and offsets separate, we can still do this!

$$W^T x + W_0 = Z \tag{8}$$

This pre-activation vector Z contains all of the outputs of our linear components:

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \tag{9}$$

On our diagram, we can see it here:




This section is what Z details with.

And we can connect this to our activation: each $a_i$ is the result of running our function f on $z_i$:

Because we run the function on each element in Z, we call this an **element-wise** use of our function.

$$A = f(Z) = \begin{bmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_n) \end{bmatrix} \tag{10}$$

## Summary of layer

So, we can now break our our layer up into pieces:

**Notation 5**

Our **layer** is a **function** that takes in $x \in \mathbb{R}^m$, and returns $A \in \mathbb{R}^n$.

It is defined by:

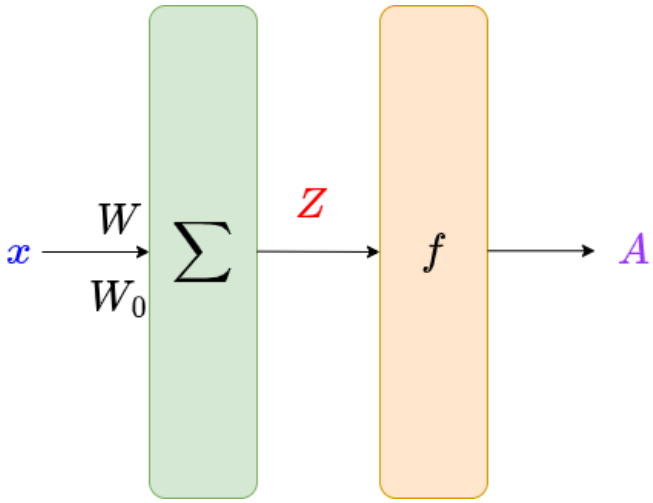
- **Dimensions**: $m$ for **input**, $n$ for **output** (number of neurons)

And our different **matrices**:

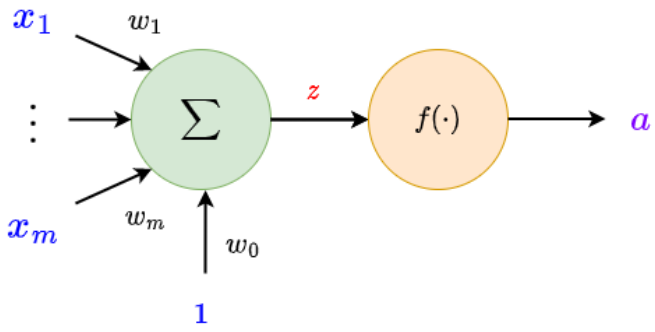
- **Input**: a **column vector** $X$ in the shape $(m \times 1)$
- **Weights**: a **matrix** $W$ in the shape $(m \times n)$
- **Offset**: a **column vector** $W_0$ in the shape $(n \times 1)$
- **Pre-activation**: a **column vector** $Z$ in the shape $(n \times 1)$
- **Activation**: a **column vector** $A$ in the shape $(n \times 1)$

We've now accomplished our goal: **simplify** the layer into its **base** components, without losing any crucial **information**.

We've can represent an entire layer like this:



Note how similar this looks to a **single** neuron: this works because the neurons in a **layer** are in **parallel**!




The math is very similar as well:

**Definition 6**

Our **layer** can be represented by

- A **linear** component that takes in x, and outputs **pre-activation** Z:

$$Z = W^T x + W_0$$

- A (potentially nonlinear) **activation** component that takes in Z, and outputs **activation** A:

$$A = f(Z)$$

When we **compose** them together, we get

$$A = f(Z) = f\left(W^T x + W_0\right)$$