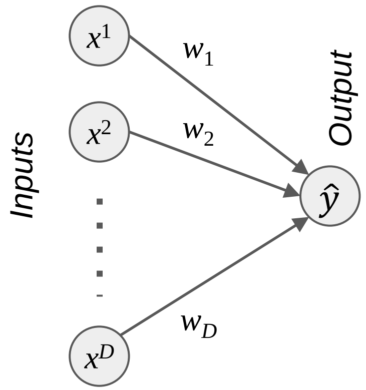
#Code

# Artificial Intelligence and Machine Learning (AIML)

2023–24

#Code

- **Last lecture**:
  - Neural networks
    - Single-layer perceptron concept (“neuron unit”)

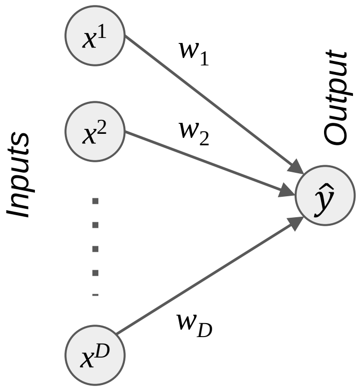



activation function (nonlinear, differentiable)

$$\hat{y} = f(w_1 x^1 + w_2 x^2 + \cdots + w_D x^D)$$

#Code

- **Last lecture**:
  - Neural networks
    - Single-layer perceptron concept ("neuron unit")



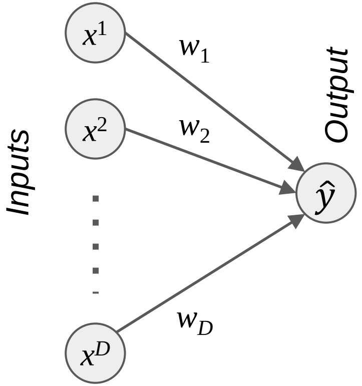

activation function (nonlinear, differentiable)

$$\hat{y} = f(w_1 x^1 + w_2 x^2 + \cdots + w_D x^D)$$

$$\hat{y} = f(w^T x)$$

#Code

- **Last lecture**:
  - Neural networks
    - Single-layer perceptron concept ("neuron unit")




activation function (nonlinear, differentiable)

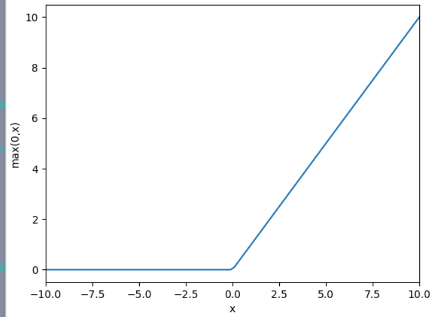
$$\hat{y} = f(w_1 x^1 + w_2 x^2 + \cdots + w_D x^D)$$

$$\hat{y} = f(w^T x) \qquad w^T = [w_1 \quad w_2 \quad \cdots \quad w_D], \;\; x = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^D \end{bmatrix}$$

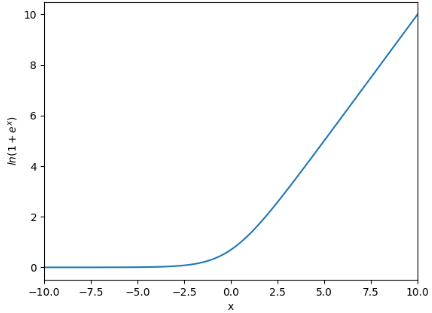
#Code

- **Last lecture**:
  - Neural networks
    - Single-layer perceptron concept (“neuron unit”)
    - Activation functions

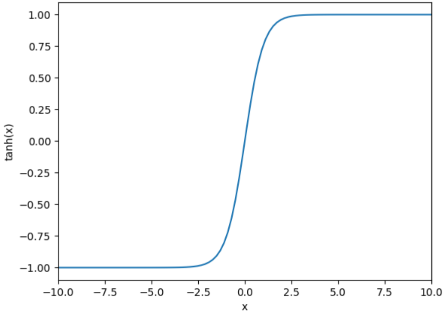
ReLU
$\left(\max\left(0, x\right)\right)$



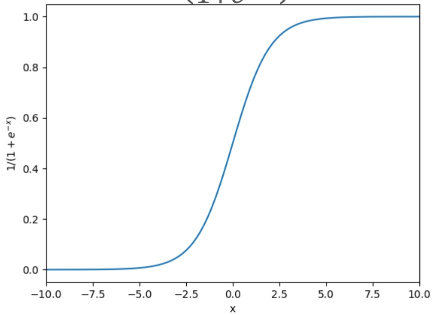

Softplus
$\left(ln(1 + e^{x})\right)$




Hyperbolic Tangent
$\left(\tanh\left(x\right)\right)$

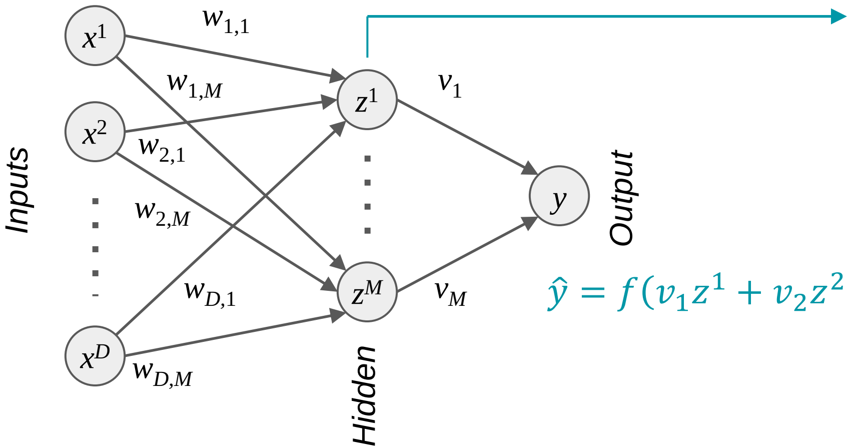



Logistic
$\left(\frac{1}{1+e^{-x}}\right)$



#Code

- **Neural networks**
  - Single-layer perceptron concept (“neuron unit”)
  - Activation functions
  - Perceptron extensions



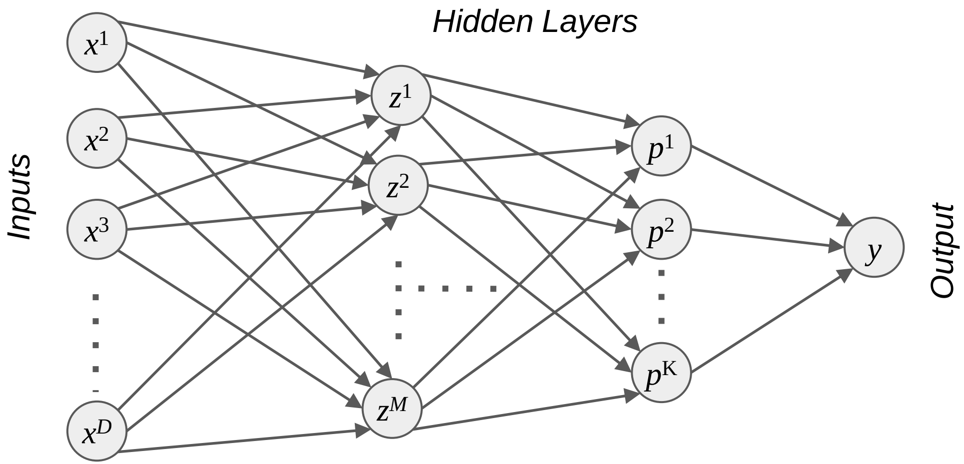

$$z^1 = f\left(w_{1,1}x^1 + w_{2,1}x^2 + \cdots + w_{D,1}x^D\right)$$
$$\vdots$$
$$z^M = f\left(w_{1,M}x^1 + w_{2,M}x^2 + \cdots + w_{D,M}x^D\right)$$

[matrix notation]
$$z = f(W^T x)$$

$$\hat{y} = f(v_1 z^1 + v_2 z^2 + \cdots + v_M z^M)$$

#Code

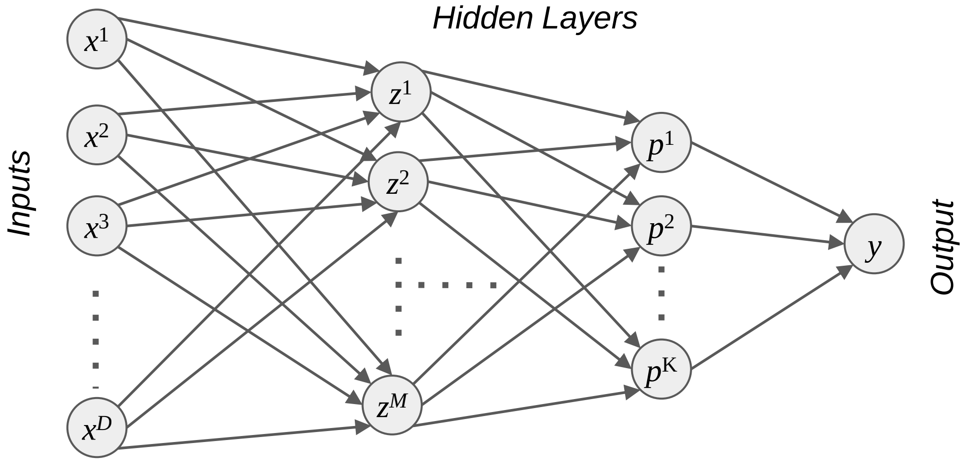
- **Neural Networks**
  - Single-layer perceptron concept (“neuron unit”)
  - Activation functions
  - Perceptron extensions and deep learning



#Code

**Neural networks**

- Single-layer perceptron concept (“neuron unit”)
- Activation functions
- Perceptron extensions and deep learning




**Fully Connected Network:**

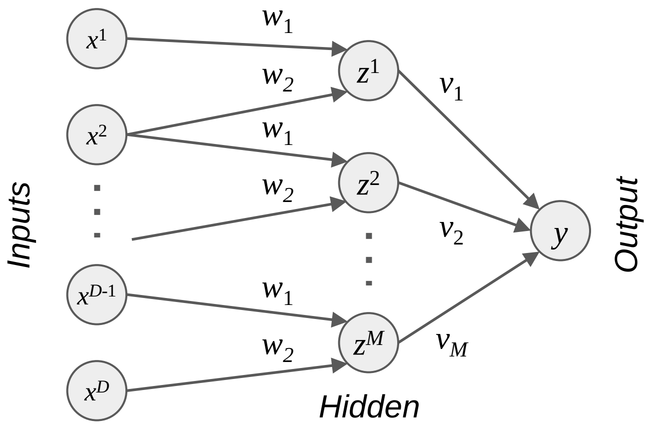
Input → Hidden Layer 1: $D \times M$ weights

HL1 → HL 2: $M \times K$ weights

Hidden Layer N: $K \times 1$ weights

#Code

- **Neural networks**
  - Single-layer perceptron concept (“neuron unit”)
  - Activation functions
  - Perceptron extensions and deep learning
    - Weights problem & weight sharing (CNN)




$$z^1 = \max(0,\ w^T[x^1\ x^2]^T)$$

$$z^2 = \max(0,\ w^T[x^2\ x^3]^T)$$

$$\ldots$$

$$z^M = \max(0,\ w^T[x^{D-1}\ x^D]^T)$$

$$y = \max(0,\ v^T z)$$

#Code

- **Neural networks**
  - Single-layer perceptron concept (“neuron unit”)
  - Activation functions
  - Perceptron extensions and deep learning
    - Weights problem & weight sharing (CNN)

- **Next:**

  How to train deep neural networks

# Training ML algorithms

**TRAINING:**
Iterative procedure for minimization of an error function, with adjustments to the weights being made at each step.

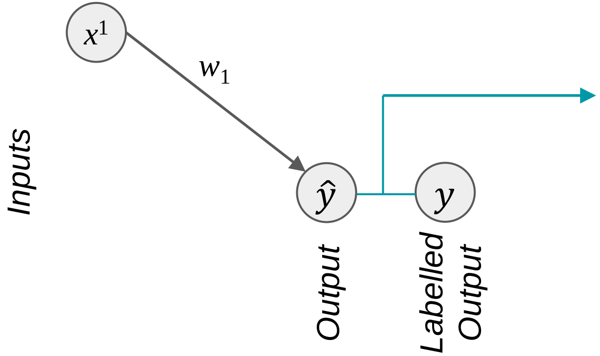
**STAGE 1:**
Evaluate derivatives of the error function wrt the weights

**STAGE 2:**
Use derivatives to compute adjustments to be made to the weights

(e.g., gradient descent)

# Training a single layer linear perceptron using gradient descent

#Code




**Error Function**

Sum of Squares:

$$F(w) = \sum_{i=1}^{N} (w_1 x_i^1 - y_i)^2$$

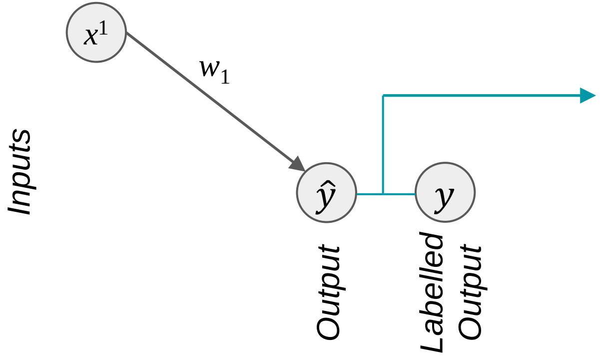
**Weight update:**

Model:

$$f(w, x) = \hat{y} = w_1 x^1$$

#Code

# Training a single layer linear perceptron using gradient descent




**Error Function**

Sum of Squares:

$$F(w) = \sum_{i=1}^{N} (w_1 x_i^1 - y_i)^2$$

**Weight update:**

$$w_{n+1} = w_n - \alpha F_w(w_n)$$

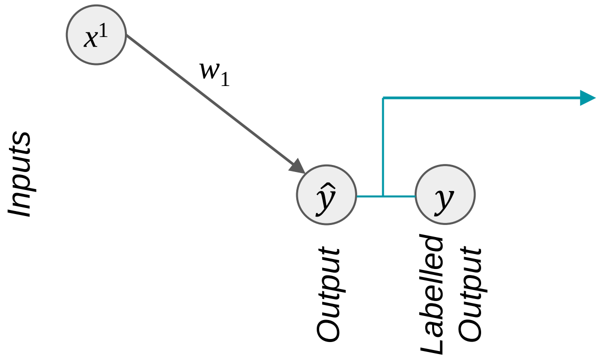
Model:

$$f(w, x) = \hat{y} = w_1 x^1$$

#Code

# Training a single layer linear perceptron using gradient descent

**Weight update:** $w_{n+1} = w_n - \alpha F_w(w_n)$




Model:

$$f(w, x) = \hat{y} = w_1 x^1$$

## Error Function

Sum of Squares:

$$F(w) = \sum_{i=1}^{N} (w_1 x_i^1 - y_i)^2$$

## Derivative

$$F_w(w) = \frac{dF}{dw_1}$$

Sum of Squares:

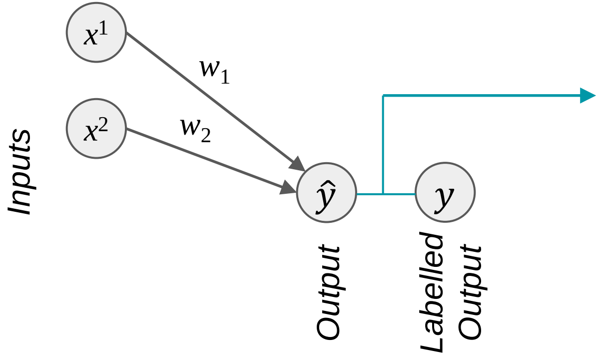
$$F_w(w) = 2 \sum_{i=1}^{N} (w_1 x_i^1 - y_i) x_i^1$$

AIML

#Code

# Training a single layer linear perceptron using gradient descent

**Weight update:** $w_{n+1} = w_n - \alpha F_w(w_n)$




Model:

$$f(w, x) = \hat{y} = w_1 x^1 + w_2 x^2$$

## Error Function

Sum of Squares:

$$F(w) = \sum_{i=1}^{N} \left( (w_1 x_i^1 + w_2 x_i^2) - y_i \right)^2$$

## Gradient

$$F_w(w) = \begin{bmatrix} \dfrac{\partial F}{\partial w_1} \\ \dfrac{\partial F}{\partial w_2} \end{bmatrix}$$

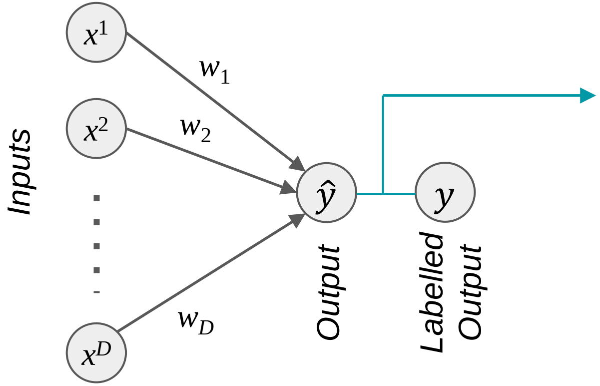
Sum of Squares:

$$F_w(w) = 2 \sum_{i=1}^{N} \left( (w_1 x_i^1 + w_2 x_i^2) - y_i \right) \begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix}$$

#Code

# Training a single layer linear perceptron using gradient descent

$$w_{n+1} = w_n - \alpha F_w(w_n)$$




**Error Function**

Sum of Squares:

$$F(w) = \sum_{i=1}^{N} (w^T x_i - y_i)^2$$

Model:

$$f(w,x) = \hat{y} = w_1 x^1 + w_2 x^2 + \cdots + w_D x^D$$

(vector notation) $\quad f(w,x) = \hat{y} = w^T x$

**Gradient**

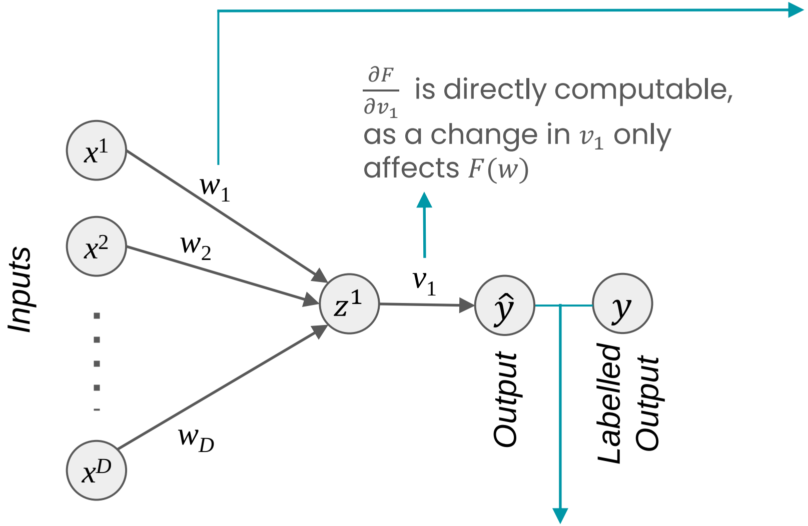
$$F_w(w) = \begin{bmatrix} \dfrac{\partial F}{\partial w_1} \\ \dfrac{\partial F}{\partial w_2} \\ \vdots \\ \dfrac{\partial F}{\partial w_D} \end{bmatrix}$$

$$F_w(w) = 2 \sum_{i=1}^{N} (w^T x_i - y_i) \begin{bmatrix} x_i^1 \\ x_i^2 \\ \vdots \\ x_i^D \end{bmatrix}$$

# Neural networks: training by gradient descent

#Code

$$w_{n+1} = w_n - \alpha F_w(w_n)$$




**Error Function:** $F(w)$

We need $\frac{\partial F}{\partial w_1}$

A change in $w_1$ will affect the output of $z^1$: $\frac{\partial z^1}{\partial w_1}$

The change in $z^1$ induced by $w_1$ will change the predicted output: $\frac{\partial F}{\partial z^1}$

The total change in the output due to a change in $w_1$ will then be:

$$\frac{\partial F}{\partial w_1} = \frac{\partial F}{\partial z^1}\frac{\partial z^1}{\partial w_1} \quad \textbf{(chain rule)}$$

## Gradient

$$F_w(w) = \begin{bmatrix} \frac{\partial F}{\partial w_1} \\ \frac{\partial F}{\partial w_2} \\ \vdots \\ \frac{\partial F}{\partial w_D} \\ \frac{\partial F}{\partial v_1} \end{bmatrix}$$

# Chain rule of Calculus

#Code

(**Gill**, Section 5.4)

- Provides a means of differentiating nested functions.
- Given two functions, $f(x)$ and $g(x)$, and the nested form

$$h = f\big(g(x)\big)$$

we need to account for the actual order of the nesting relationship to correctly differentiate $h$:

#Code

# Chain rule of Calculus

(**Gill**, Section 5.4)

- Provides a means of differentiating nested functions.
- Given two functions, $f(x)$ and $g(x)$, and the nested form

$$h = f\big(g(x)\big)$$

we need to account for the actual order of the nesting relationship to correctly differentiate $h$:

$$\frac{dh}{dx} = \frac{d}{dx} f\big(g(x)\big) = f'\big(g(x)\big)g'(x)$$

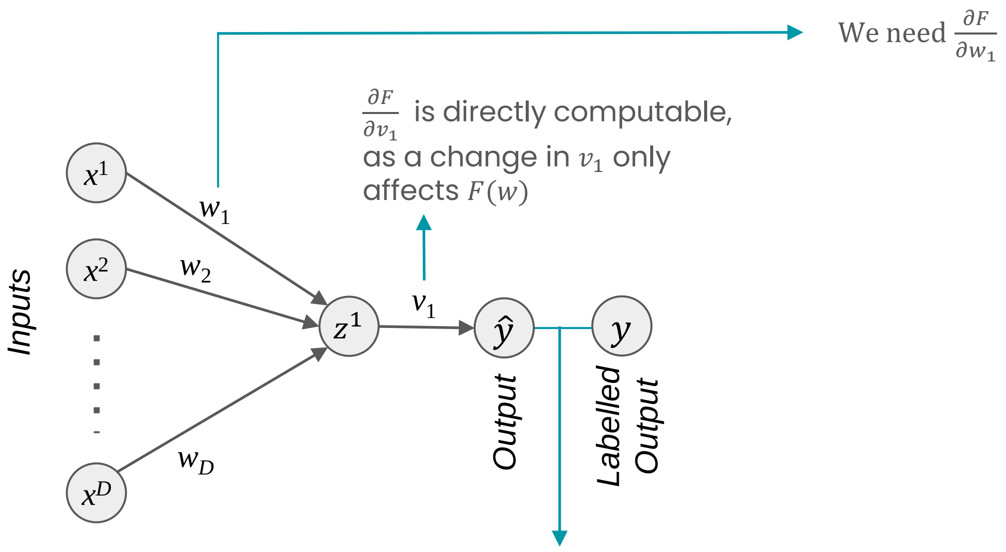
where $f'(x)$ is shorthand for $\frac{df(x)}{dx}$.

# Neural networks: Training using gradient descent

#Code

- Gradient Descent Approach

$$w_{n+1} = w_n - \alpha F_w(w_n)$$




**Error Function:** $F(w)$

**Gradient**

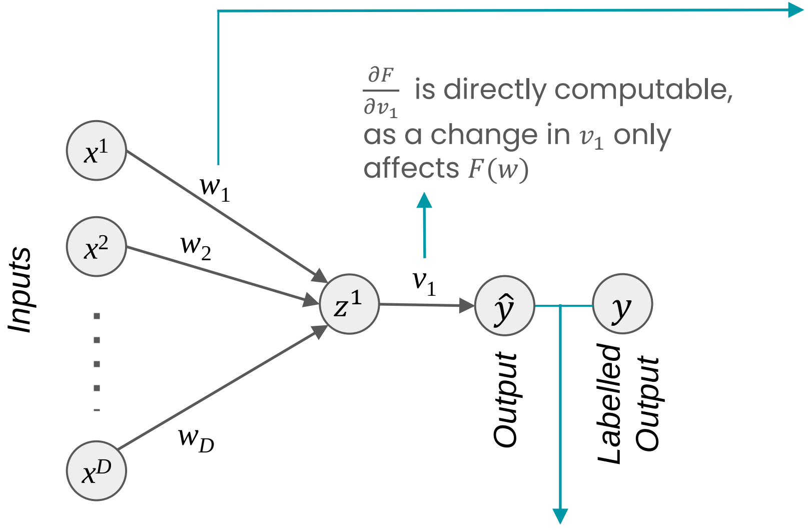
$$F_w(w) = \begin{bmatrix} \frac{\partial F}{\partial w_1} \\ \frac{\partial F}{\partial w_2} \\ \vdots \\ \frac{\partial F}{\partial w_D} \\ \frac{\partial F}{\partial v_1} \end{bmatrix}$$

# Neural networks: Training using gradient descent

#Code

- Gradient Descent Approach

$$w_{n+1} = w_n - \alpha F_w(w_n)$$




**Error Function:** $F(w)$

We need $\frac{\partial F}{\partial w_1}$

A change in $w_1$ will affect the output of $z^1$: $\frac{\partial z^1}{\partial w_1}$

**Gradient**

$$F_w(w) = \begin{bmatrix} \frac{\partial F}{\partial w_1} \\ \frac{\partial F}{\partial w_2} \\ \vdots \\ \frac{\partial F}{\partial w_D} \\ \frac{\partial F}{\partial v_1} \end{bmatrix}$$

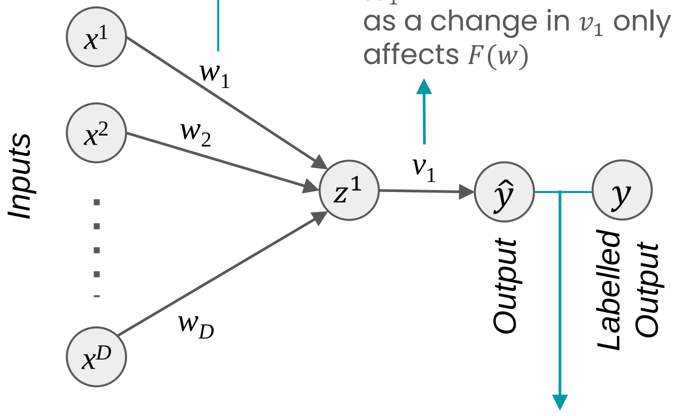
# Neural networks: Training using gradient descent

#Code

- Gradient Descent Approach

$$w_{n+1} = w_n - \alpha F_w(w_n)$$

$\frac{\partial F}{\partial v_1}$ is directly computable, as a change in $v_1$ only affects $F(w)$




**Error Function:** $F(w)$

We need $\frac{\partial F}{\partial w_1}$

A change in $w_1$ will affect the output of $z^1$: $\frac{\partial z^1}{\partial w_1}$

The change in $z^1$ induced by $w_1$ will change the predicted output: $\frac{\partial F}{\partial z^1}$

**Gradient**

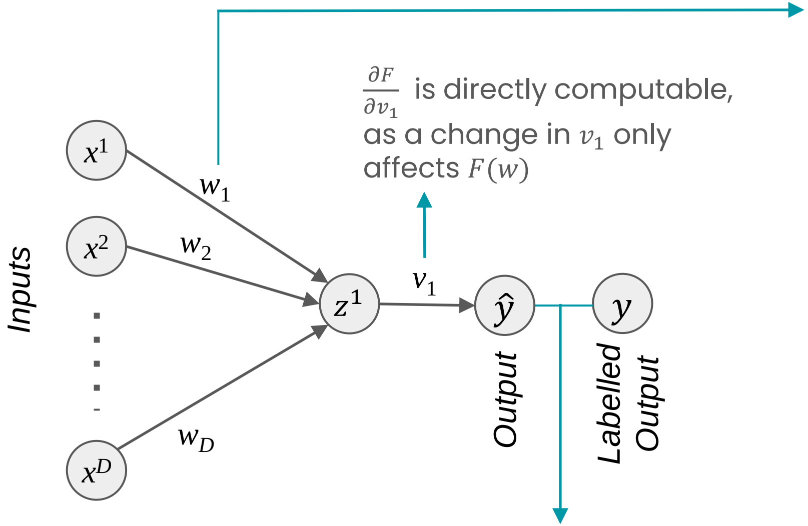
$$F_w(w) = \begin{bmatrix} \frac{\partial F}{\partial w_1} \\ \frac{\partial F}{\partial w_2} \\ \vdots \\ \frac{\partial F}{\partial w_D} \\ \frac{\partial F}{\partial v_1} \end{bmatrix}$$

#Code

# Neural networks: Training using gradient descent

- Gradient Descent Approach

$$w_{n+1} = w_n - \alpha F_w(w_n)$$




**Error Function:** $F(w)$

We need $\frac{\partial F}{\partial w_1}$

A change in $w_1$ will affect the output of $z^1$: $\frac{\partial z^1}{\partial w_1}$

The change in $z^1$ induced by $w_1$ will change the predicted output: $\frac{\partial F}{\partial z^1}$

The total change in the output due to a change in $w_1$ will then be:

$$\frac{\partial F}{\partial w_1} = \frac{\partial F}{\partial z^1}\frac{\partial z^1}{\partial w_1} \qquad \textbf{(chain rule)}$$

**Gradient**

$$F_w(w) = \begin{bmatrix} \frac{\partial F}{\partial w_1} \\ \frac{\partial F}{\partial w_2} \\ \vdots \\ \frac{\partial F}{\partial w_D} \\ \frac{\partial F}{\partial v_1} \end{bmatrix}$$

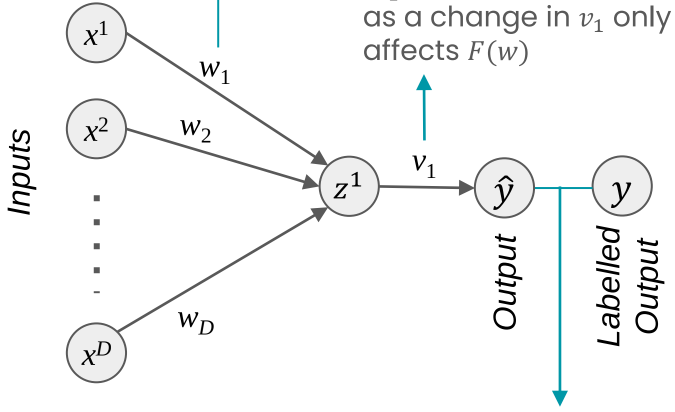
#Code

# Neural networks: Training using gradient descent

- Gradient Descent Approach

$$w_{n+1} = w_n - \alpha F_w(w_n)$$

$\frac{\partial F}{\partial v_1}$ is directly computable, as a change in $v_1$ only affects $F(w)$




**Error Function:** $F(w)$

We need $\frac{\partial F}{\partial w_1}$

A change in $w_1$ will affect the output of $z^1$: $\frac{\partial z^1}{\partial w_1}$

The change in $z^1$ induced by $w_1$ will change the predicted output: $\frac{\partial F}{\partial z^1}$

The total change in the output due to a change in $w_1$ will then be:

$$\frac{\partial F}{\partial w_1} = \frac{\partial F}{\partial z^1}\frac{\partial z^1}{\partial w_1}$$ **(chain rule)**

**Gradient**

$$F_w(w) = \begin{bmatrix} \frac{\partial F}{\partial w_1} \\ \frac{\partial F}{\partial w_2} \\ \vdots \\ \frac{\partial F}{\partial w_D} \\ \frac{\partial F}{\partial v_1} \end{bmatrix}$$

We must **propagate gradients** from the output error $F(w)$, all the way through each layer

# Training deep networks

#Code

**TRAINING:**
Iterative procedure for minimization of an error function, with adjustments to the weights being made at each step.

**STAGE 1:**
Evaluate derivatives of the error function wrt the weights

Analytical gradient expressions quickly become intractable

**STAGE 2:**
Use derivatives to compute adjustments to be made to the weights

# Training deep networks

#Code

**TRAINING:**
Iterative procedure for minimization of an error function, with adjustments to the weights being made at each step.

**STAGE 1:**
Evaluate derivatives of the error function wrt the weights

Analytical gradient expressions quickly become intractable

More sophisticated gradient calculation methods:
**backpropagation** and **automatic differentiation** (**AD**)

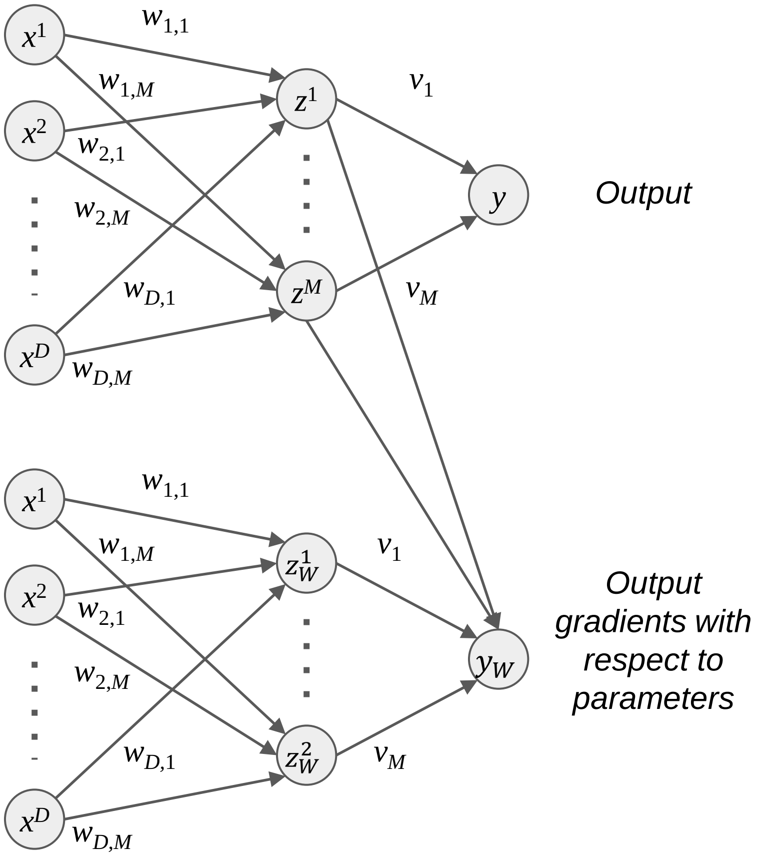
**STAGE 2:**
Use derivatives to compute adjustments to be made to the weights

(e.g., gradient descent)

#Code

# Automatic differentiation

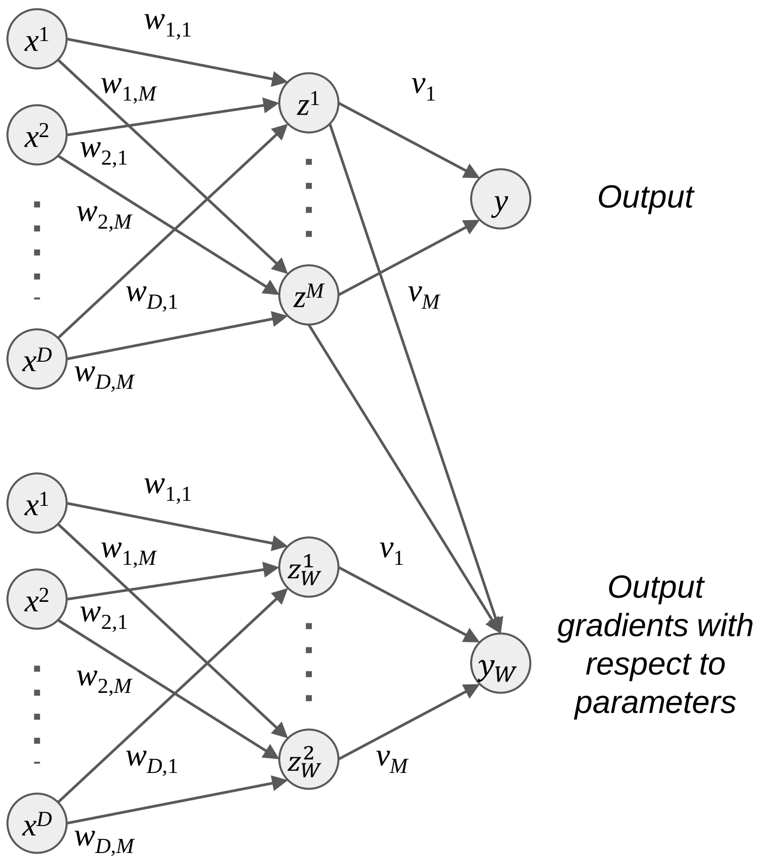
- "**meta-programming**" approach to gradient calculation
- Obtains the gradients of the output simultaneous with the output of the network



#Code

# Automatic differentiation

- "**meta-programming**" approach to gradient calculation
- Obtains the gradients of the output simultaneous with the output of the network
- Software packages such as PyTorch, JAX largely avoid the need for any hand-computed gradients in this way



#Code

# Algebra of dual numbers

- Dual numbers are an easy way to handle chained computations and automate calculations for AD

- Keeps track of current computation's value $u$ and its derivative $u'$ as a pair:

$$(u, u')$$

# Algebra of dual numbers

#Code

- Dual numbers are an easy way to handle chained computations and automate calculations for AD

- Keeps track of current computation's value $u$ and its derivative $u'$ as a pair:

$$(u, u')$$

- In dual number form, the general chain rule is,

$$f((u, u')) = (f(u), f'(u)u')$$

- Most forms of computational operations used in ML, are special cases of this rule

AIML

#Code

# Algebra of dual numbers

Computations which commonly occur in deep learning, in dual number form

- **Addition**: $f(u, v) = u + v$

**Example:** Suppose $u = 2x^1$ and $v = x^1 + x^2$.

For $x^1 = 3, x^2 = 4$, we want to compute

1. $z = u + v$

2. $\frac{dz}{dx^1}$

AIML

#Code

# Algebra of dual numbers

Computations which commonly occur in deep learning, in dual number form

- **Addition**: $f(u, v) = u + v$

**Example:** Suppose $u = 2x^1$ and $v = x^1 + x^2$.

For $x^1 = 3, x^2 = 4$, we want to compute

1. $z = u + v$

$$u = 2 \times 3 = 6$$
$$v = 3 + 4 = 7$$

$$z = 6 + 7$$

2. $\frac{dz}{dx^1} = \frac{d}{dx^1}(u + v) = \frac{du}{dx^1} + \frac{dv}{dx^1} = 2 + 1 = 3$

AIML

#Code

# Algebra of dual numbers

Computations which commonly occur in deep learning, in dual number form

- **Addition**: $f(u, v) = u + v$

$$(u, u') + (v, v') = (u + v, u' + v')$$

**Example:** Suppose $u = 2x^1$ and $v = x^1 + x^2$.

For $x^1 = 3, x^2 = 4$, we want to compute

1. $z = u + v$

$$u = 2 \times 3 = 6$$
$$v = 3 + 4 = 7$$

$$z = 6 + 7$$

2. $\frac{dz}{dx^1} = \frac{d}{dx^1}(u + v) = \frac{du}{dx^1} + \frac{dv}{dx^1} = 2 + 1 = 3$

AIML

#Code

# Algebra of dual numbers

Computations which commonly occur in deep learning, in dual number form

- **Multiplication**: $f(u, v) = uv$

**Example:** Suppose $u = 2x^1$ and $v = x^1 + x^2$.

For $x^1 = 3, x^2 = 4$, we want to compute

1. $z = uv$

2. $\frac{dz}{dx^1}$

AIML

#Code

# Algebra of dual numbers

Computations which commonly occur in deep learning, in dual number form

- **Multiplication**: $f(u, v) = uv$

**Example:** Suppose $u = 2x^1$ and $v = x^1 + x^2$.

For $x^1 = 3, x^2 = 4$, we want to compute

1. $z = uv$

$$u = 2 \times 3 = 6$$
$$v = 3 + 4 = 7$$

$$z = 6 \times 7 = 42$$

2. $\frac{dz}{dx^1} = \frac{d}{dx^1}(uv) = \frac{du}{dx^1}v + u\frac{dv}{dx^1} = 2 \times 7 + 6 \times 1 = 20$

AIML

#Code

# Algebra of dual numbers

Computations which commonly occur in deep learning, in dual number form

- **Multiplication**: $f(u, v) = uv$

$$(u, u') \times (v, v') = (uv, u'v + uv')$$

**Example:** Suppose $u = 2x^1$ and $v = x^1 + x^2$.

For $x^1 = 3, x^2 = 4$, we want to compute

1. $z = uv$

$$u = 2 \times 3 = 6$$
$$v = 3 + 4 = 7$$

$$z = 6 \times 7 = 42$$

2. $\frac{dz}{dx^1} = \frac{d}{dx^1}(uv) = \frac{du}{dx^1} v + u \frac{dv}{dx^1} = 2 \times 7 + 6 \times 1 = 20$

AIML

#Code

# Algebra of dual numbers

Computations which commonly occur in deep learning, in dual number form

- **Addition**: $(u, u') + (v, v') = (u + v, u' + v')$
- **Multiplication**: $(u, u') \times (v, v') = (uv, u'v + uv')$
- **ReLU activation**: $\mathrm{relu}((u, u')) = (\max(0, u), u' \mathbb{I}[u \geq 0])$
- **Constants** $f(u) = c$

$$f((u, u')) = (c, 0)$$

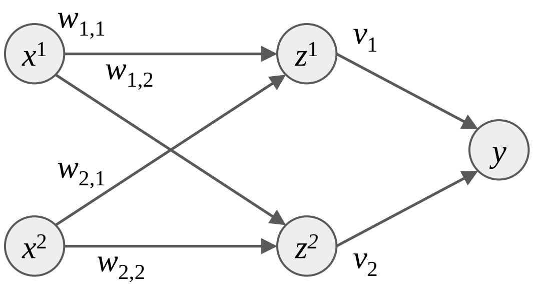
Here is an example of a chained calculation carried out using dual numbers. Given the constants $y = 3$ and $z = -1$ and variable $x = 2$, compute $u(x, y, z) = \max(yz, y + 2x)$ and its derivative, $u_x(x, y, z)$. Applying the rules above successively (and using additional symbols for intermediate computational results),

$$
\begin{aligned}
\bar{x} &= (2, 1) \\
\bar{y} &= (3, 0) \\
\bar{z} &= (-1, 0) \\
c &= (2, 0) \\
c\bar{x} &= (2, 0) \times (2, 1) = (4, 2) \\
r_1 &= \bar{y} \times \bar{z} = (3, 0) \times (-1, 0) = (-3, 0) \\
r_2 &= \bar{y} + c\bar{x} = (3, 0) + (4, 2) = (7, 2) \\
\bar{u} &= \max((-3, 0), (7, 2)) = (7, 2),
\end{aligned}
\tag{14.7}
$$

therefore $u(x, y, z) = 7$ and $u_x(x, y, z) = 2$. While it is, of course, always possible to find the symbolic derivative of the function $u(x, y, z)$, AD enables entirely 'mechanical' calculational steps which lends itself to software implementation.

# Automatic differentiation in action

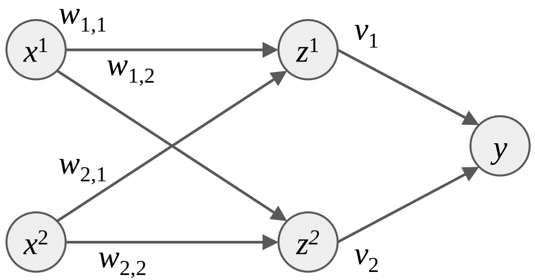
#Code




$$z^1 = \text{relu}(w_{1,1}x^1 + w_{2,1}x^2)$$

# Automatic differentiation in action

#Code

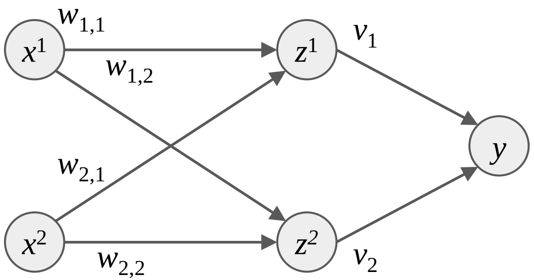



$z^1 = \mathrm{relu}(w_{1,1}x^1 + w_{2,1}x^2)$

$w_{1,1} \longrightarrow (w_{1,1}, 1), w_{2,1} \longrightarrow (w_{2,1}, 0), x^1 \longrightarrow (x^1, 0), x^2 \longrightarrow (x^2, 0)$

# Automatic differentiation in action

#Code

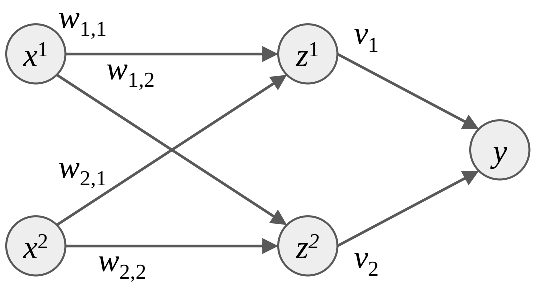



$$z^1 = \mathrm{relu}(w_{1,1}x^1 + w_{2,1}x^2)$$
$$= \mathrm{relu}((w_{1,1}, 1) \times (x^1, 0) + (w_{2,1}, 0) \times (x^2, 0))$$

$w_{1,1} \longrightarrow (w_{1,1}, 1), w_{2,1} \longrightarrow (w_{2,1}, 0), x^1 \longrightarrow (x^1, 0), x^2 \longrightarrow (x^2, 0)$

# Automatic differentiation in action

#Code




$$z^1 = \mathrm{relu}(w_{1,1}x^1 + w_{2,1}x^2)$$

$$= \mathrm{relu}((w_{1,1}, 1) \times (x^1, 0) + (w_{2,1}, 0) \times (x^2, 0))$$

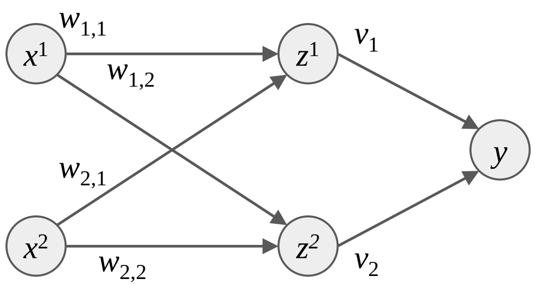
$$= \mathrm{relu}((w_{1,1}x^1, x^1) + (w_{2,1}x^2, 0))$$

$$w_{1,1} \longrightarrow (w_{1,1}, 1),\ w_{2,1} \longrightarrow (w_{2,1}, 0),\ x^1 \longrightarrow (x^1, 0),\ x^2 \longrightarrow (x^2, 0)$$

$$(u, u') \times (v, v') = (uv, u'v + v'u)$$

#Code

# Automatic differentiation in action




$$z^1 = \mathrm{relu}(w_{1,1}x^1 + w_{2,1}x^2)$$

$$= \mathrm{relu}((w_{1,1}, 1) \times (x^1, 0) + (w_{2,1}, 0) \times (x^2, 0))$$

$$= \mathrm{relu}((w_{1,1}x^1, x^1) + (w_{2,1}x^2, 0))$$

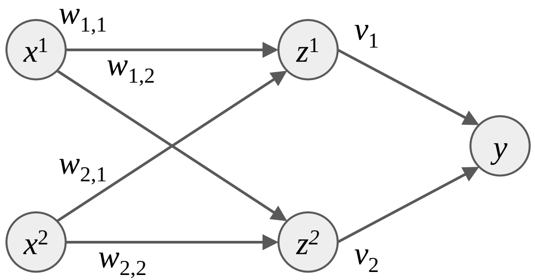
$$= \mathrm{relu}((w_{1,1}x^1 + w_{2,1}x^2, x^1))$$

$$w_{1,1} \rightarrow (w_{1,1}, 1),\ w_{2,1} \rightarrow (w_{2,1}, 0),\ x^1 \rightarrow (x^1, 0),\ x^2 \rightarrow (x^2, 0)$$

$$(u, u') \times (v, v') = (uv, u'v + v'u)$$

$$(u, u') + (v, v') = (u + v, u' + v')$$

# Automatic differentiation in action




$$z^1 = \mathrm{relu}(w_{1,1}x^1 + w_{2,1}x^2)$$
$$= \mathrm{relu}((w_{1,1}, 1) \times (x^1, 0) + (w_{2,1}, 0) \times (x^2, 0))$$
$$= \mathrm{relu}((w_{1,1}x^1, x^1) + (w_{2,1}x^2, 0))$$
$$= \mathrm{relu}((w_{1,1}x^1 + w_{2,1}x^2, x^1))$$
$$= (\max(0, w_{1,1}x^1 + w_{2,1}x^2), x^1\ 1[w_{1,1}x^1 + w_{2,1}x^2])$$

$$w_{1,1} \to (w_{1,1}, 1),\ w_{2,1} \to (w_{2,1}, 0),\ x^1 \to (x^1, 0),\ x^2 \to (x^2, 0)$$
$$(u, u') \times (v, v') = (uv, u'v + v'u)$$
$$(u, u') + (v, v') = (u + v, u' + v')$$
$$\mathrm{relu}((u, u')) = (\max(0, u), u'\ 1[u \geq 0])$$

AIML

#Code

# To recap

- We learned one approach of finding the optimal set of weights for a neural network (i.e., **train the network**)
  - Analytical gradient expressions quickly become intractable
  - Automatic differentiation computes derivatives in a single forward pass
    - We learned the *forward mode* version of AD; there is also *reverse* mode
- **Next**: Probability and probabilistic AI

# Further Reading

- **PRML**, Section 5.3
- **R&N**, Section 18.7
- **H&T**, Section 11.4, Section 11.7