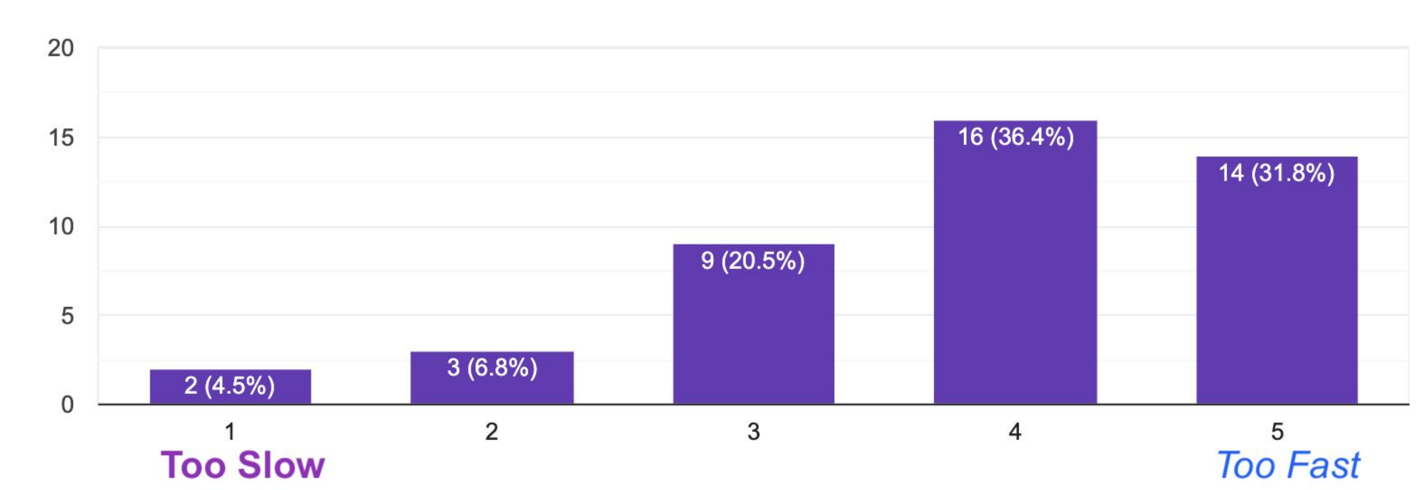
Optimization and Automatic Differentiation

Mathematics for Machine Learning

Lecturer: Matthew Wicker

Logistics!



Thank you all so much for your feedback! It is invaluable to me and I take it seriously.

Some say I use terms they don't know, so that will be the question after today's lecture!

Errata

So there were some errors especially in the last section of last week's notes. So we are going over that again today to clear things up.

Thank you so much to all who reported errata!!

New reward system for this! If you email me and point out a valid (yet unreported) mathematical error in the lecture notes, you get 1 ticket in the end of term raffle!

Prizes will include: A Imperial hoodie, a Rasberry Pi Model B, & Amazon Gift Card

Defining some terminology

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}$$

Column vectors by default

$$\mathbf{x}, \theta \in \mathbb{R}^{n \times 1}$$

Defining some terminology

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}, \qquad \theta^\top = [\theta_1, \ldots, \theta_n]$$

$$n \times 1 \qquad\qquad 1 \times n$$

Defining some terminology

$$\theta^\top = [\theta_1, \ldots, \theta_n] \qquad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}$$

$$\overbrace{1 \times \underbrace{n \qquad n}_{\text{Must match}} \times 1}^{\text{Dimension of result}}$$

Writing out our model

$\theta^T X$: $1 \times n$ $n \times 1$

$1 \times n$

$$\begin{bmatrix} \mathbf{x}^{\top(1)}, \\ \mathbf{x}^{\top(2)}, \\ \vdots, \\ \mathbf{x}^{\top(N)} \end{bmatrix}$$

$$\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

$n \times 1$

Writing out our model

$X\theta: N\times n \times n\times 1$

$$
\begin{bmatrix} \hat{y}^{(1)}, \\ \hat{y}^{(2)}, \\ \dots, \\ \hat{y}^{(N)}, \end{bmatrix}
=
\begin{bmatrix} x_1^{(1)}, \dots, x_n^{(1)} \\ x_1^{(2)}, \dots, x_n^{(2)} \\ \vdots, \ddots, \vdots \\ x_1^{(N)}, \dots, x_n^{(N)} \end{bmatrix}
\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}
=
\begin{bmatrix} x_1^{(1)}\theta_1 + x_2^{(1)}\theta_2 + \dots \\ x_1^{(2)}\theta_1 + x_2^{(2)}\theta_2 + \dots \\ \vdots \end{bmatrix}
$$

$$N \times 1 \qquad\qquad N \times n \qquad\qquad n \times 1$$

Writing out our model

$$\boxed{\hat{\mathbf{y}} = \mathbf{X}\theta}$$

$N \times 1$ $N \times n$ $n \times 1$

Writing out our model

$$\hat{\mathbf{y}} = \mathbf{X}\theta$$

actual - prediction

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_i (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$$

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_i (\mathbf{y}_i - \underbrace{\mathbf{X}_{i,j}\theta_j}_{\text{summing over } j \text{ implicitly}})^2$$

Writing out our model

$$\hat{\mathbf{y}} = \mathbf{X}\theta$$

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i}(\mathbf{y}_i - \mathbf{X}_{i,j}\theta_j)^2$$

$$\frac{1}{N}(\mathbf{X}\theta - \mathbf{y})^\top(\mathbf{X}\theta - \mathbf{y})$$

Expanding the loss

$$\frac{1}{N}(\mathbf{X}\theta - \mathbf{y})^\top(\mathbf{X}\theta - \mathbf{y}) = \frac{1}{N}\left((X\theta)^T - y^T\right)(X\theta - y)$$

$$= \frac{1}{N}\left(\theta^T X^T - y^T\right)(X\theta - y) = \frac{1}{N}\left(\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta \ldots\right.$$

$$= \frac{1}{N}\left(\theta^\top(\mathbf{X}^\top\mathbf{X})\theta - \theta^\top\mathbf{X}^\top\mathbf{y} - \mathbf{y}^\top\mathbf{X}\theta + \mathbf{y}^\top\mathbf{y}\right)$$

Expanding the loss

$y = \begin{bmatrix} \\ \\ \end{bmatrix}_{N\times 1} = \begin{bmatrix} & \\ & \end{bmatrix}_{N\times M} \begin{bmatrix} \\ \\ \end{bmatrix}_{M\times 1}$

$$\frac{1}{N}(\mathbf{X}\theta - \mathbf{y})^\top(\mathbf{X}\theta - \mathbf{y})$$

$$= \left(\theta^\top(\mathbf{X}^\top\mathbf{X})\theta - \overbrace{\underbrace{\theta^\top}_{1\times M}\underbrace{\mathbf{X}^\top}_{M\times N}\underbrace{\mathbf{y}}_{N\times 1}}^{(X\theta)^\top y}_{=1\times 1} - \underbrace{\mathbf{y}^\top}_{1\times N}\underbrace{\mathbf{X}\theta}_{N\times M\ M\times 1}{}_{=1\times 1} + \mathbf{y}^\top\mathbf{y}\right)$$

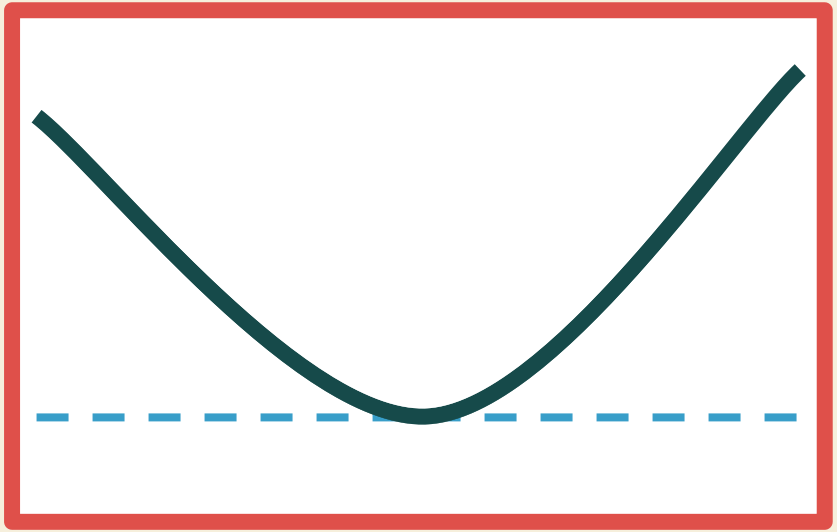
scalars

$$= \left(\theta^\top(\mathbf{X}^\top\mathbf{X})\theta - 2(\underbrace{\mathbf{y}^\top}_{1\times N}\underbrace{\mathbf{X}}_{N\times M}\underbrace{\theta}_{M\times 1})_{=1\times 1} + \mathbf{y}^\top\mathbf{y}\right)$$

Why is this step true? Take a second to convince yourself that this is correct

Using Calculus in Optimization

For a quadratic functions (formally defined in lecture 4), we know that the optimal function value will have gradient zero.



Using Calculus in Optimization

For a quadratic functions (formally defined in lecture 4), we know that the optimal function value will have gradient zero.

$$\left(\theta^\top(\mathbf{X}^\top\mathbf{X})\theta - 2(\mathbf{y}^\top\mathbf{X}\theta) + \mathbf{y}^\top\mathbf{y}\right)$$

We don't want to differentiate all of these by hand, so we will write out (and here prove) a few differentiation rules

Proving some vector calc identities

$$\nabla_\theta \mathbf{c}^\top \theta = \mathbf{c}^\top$$

vectors columns
by default

Proving some vector calc identities

Start by expanding into einstein or summation notation

$$\mathbf{c}^\top \theta = \sum_j \mathbf{c}_j \theta_j$$

Proving some vector calc identities

Start by expanding into einstein or summation notation

$$\mathbf{c}^\top \theta = \sum_j \mathbf{c}_j \theta_j$$

Now reason about what each of the partial derivatives of the einstein notation are

$$\frac{\partial \mathbf{c}^\top \theta}{\partial \theta_j} = \mathbf{c}_j$$

Proving some vector calc identities

Start by expanding into einstein or summation notation

$$\mathbf{c}^\top \theta = \sum_j \mathbf{c}_j \theta_j$$

Now reason about what each of the partial derivatives of the einstein notation are

$$\frac{\partial \mathbf{c}^\top \theta}{\partial \theta_j} = \mathbf{c}_j$$

Finally put this from einstein notation back to vector notation

$$\nabla_\theta \mathbf{c}^\top \theta = \mathbf{c}^\top$$

Proving some vector calc identities

$$\nabla_\theta(\overbrace{\theta^\top \mathbf{A}\theta}^{\text{scaler}}) = \mathbf{A}\theta + \mathbf{A}^\top\theta$$

1×N N×M M×1

row 1: cols(A) × θ

Proving some vector calc identities

Start by expanding into einstein or summation notation

$$\theta^\top \mathbf{A}\theta = \sum_i \sum_j \underbrace{\theta_i \theta_j \mathbf{A}_{i,j}}_{\text{order doesn't matter}}$$

Proving some vector calc identities

Start by expanding into einstein or summation notation

$$\theta^\top \mathbf{A}\theta = \sum_i \sum_j \theta_i \theta_j \mathbf{A}_{i,j}$$

$$\frac{\partial}{\partial \theta_k}(\theta_i \theta_j) = \begin{cases} \theta_j & , k=i \\ \theta_i & , k=j \\ 0 & \text{else} \end{cases}$$

Now reason about what each of the partial derivatives of the einstein notation are

$$\frac{\partial \theta^\top \mathbf{A}\theta}{\partial \theta_k} = \sum_i \theta_i \mathbf{A}_{i,k} + \sum_j \mathbf{A}_{k,j}\theta_j$$

$k=j$

$k=i$

Proving some vector calc identities

Start by expanding into einstein or summation notation

$$\theta^\top \mathbf{A}\theta = \sum_i \sum_j \theta_i \theta_j \mathbf{A}_{i,j}$$

Now reason about what each of the partial derivatives of the einstein notation are

$$\frac{\partial \theta^\top \mathbf{A}\theta}{\partial \theta_k} = \boxed{\sum_i \theta_i \mathbf{A}_{i,k}} + \boxed{\sum_j \mathbf{A}_{k,j}\theta_j}$$

Finally put this from einstein notation back to vector notation

Proving some vector calc identities

Start by expanding into einstein or summation notation

$$\theta^\top \mathbf{A}\theta = \sum_i \sum_j \theta_i \theta_j \mathbf{A}_{i,j}$$

Now reason about what each of the partial derivatives of the einstein notation are

$$\frac{\partial \theta^\top \mathbf{A}\theta}{\partial \theta_k} = \boxed{\sum_i \underbrace{\theta_i \mathbf{A}_{i,k}}_{\text{kth element of } A\theta}} + \boxed{\sum_j \underbrace{\mathbf{A}_{k,j}\theta_j}_{\text{kth element of } A^T\theta}}$$

Finally put this from einstein notation back to vector notation

$$\nabla_\theta(\theta^\top \mathbf{A}\theta) = \mathbf{A}\theta + \mathbf{A}^\top \theta$$

(if A is symmetric then its $2A\theta$)

Optimizing our parameter

$\nabla_\theta (c^T\theta) = c^T$

$\nabla_\theta (\theta^T A \theta) = \theta^T(A + A^T)$

$(y - Ax)^T(y - Ax)$

$x^T A^T A x - y^T A x - x^T A^T y$

$$\mathcal{L} = \frac{1}{N}(\mathbf{X}\theta - \mathbf{y})^\top(\mathbf{X}\theta - \mathbf{y})$$

$\theta^T X^T X \theta - 2y^T X\theta + y^T y \xrightarrow{\frac{\partial}{\partial\theta}} \theta^T(X^TX + (X^TX)^T) - 2y^TX = 0$

$= 2\theta^T X^T X - 2y^T X = 0$

$\theta^T = y^T X (X^T X)^{-1}$

$\theta = (X^TX)^{-1}X^Ty$

$$\nabla_\theta \mathcal{L} = \frac{2}{N}\left((\mathbf{X}^\top\mathbf{X})\theta - \mathbf{X}^\top\mathbf{y}\right)$$

$(AB)^T = B^T A^T$

$A = X^T X$

$$L = \frac{1}{N}\left(\theta^\top(\mathbf{X}^\top\mathbf{X})\theta - 2(\mathbf{y}^\top\mathbf{X}\theta) + \mathbf{y}^\top\mathbf{y}\right) = \theta^T A \theta - 2\underbrace{(X^Ty)^T\theta}_{= c^T\theta} + y^Ty$$

$[\ ]^{1\times m}\,[\ ]^{m\times m}\,[\ ]^{m\times 1}$

$\nabla_\theta L = A\theta + A^T\theta - 2X^Ty = X^TX\theta + X^TX\theta - 2X^Ty$

$= 2X^TX\theta - 2X^Ty$

Solving for the parameter

set grad of loss to 0:

$$0 = \frac{2}{N}\left((\mathbf{X}^\top\mathbf{X})\theta - \mathbf{X}^\top\mathbf{y}\right)$$

$$= (\mathbf{X}^\top\mathbf{X})\theta - \mathbf{X}^\top\mathbf{y}$$

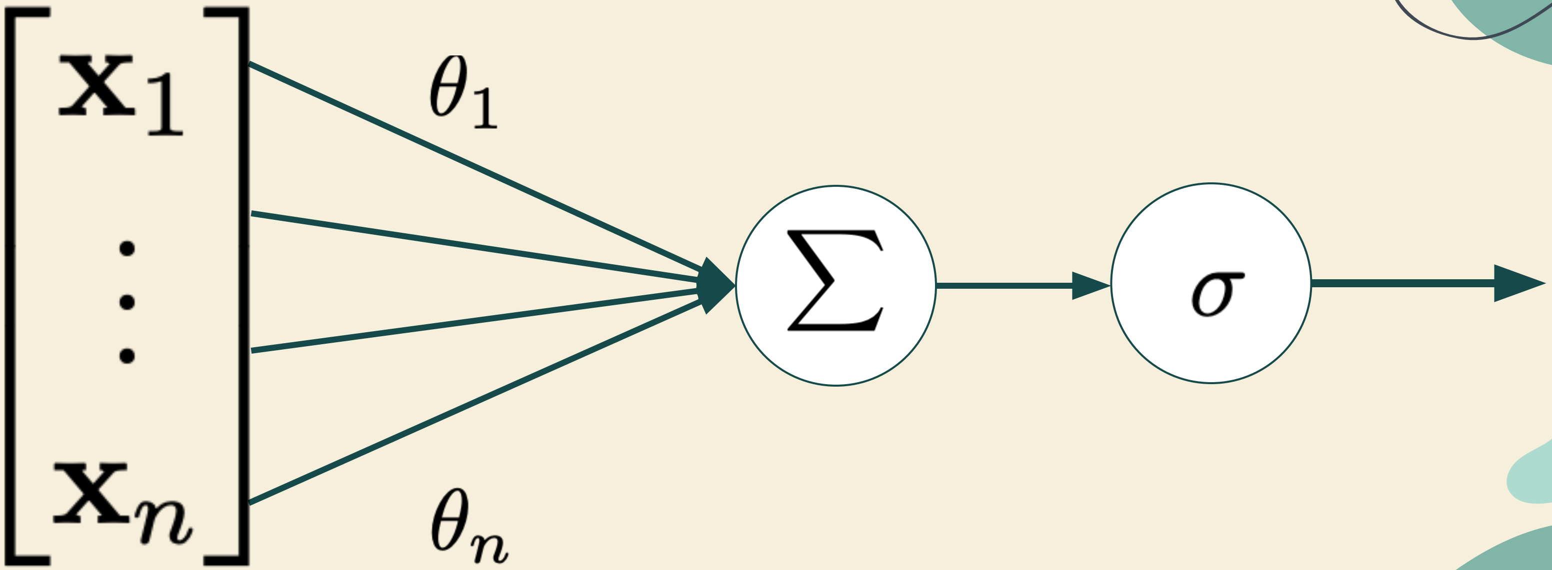
$$(\mathbf{X}^\top\mathbf{X})\theta = \mathbf{X}^\top\mathbf{y}$$

$$\theta = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

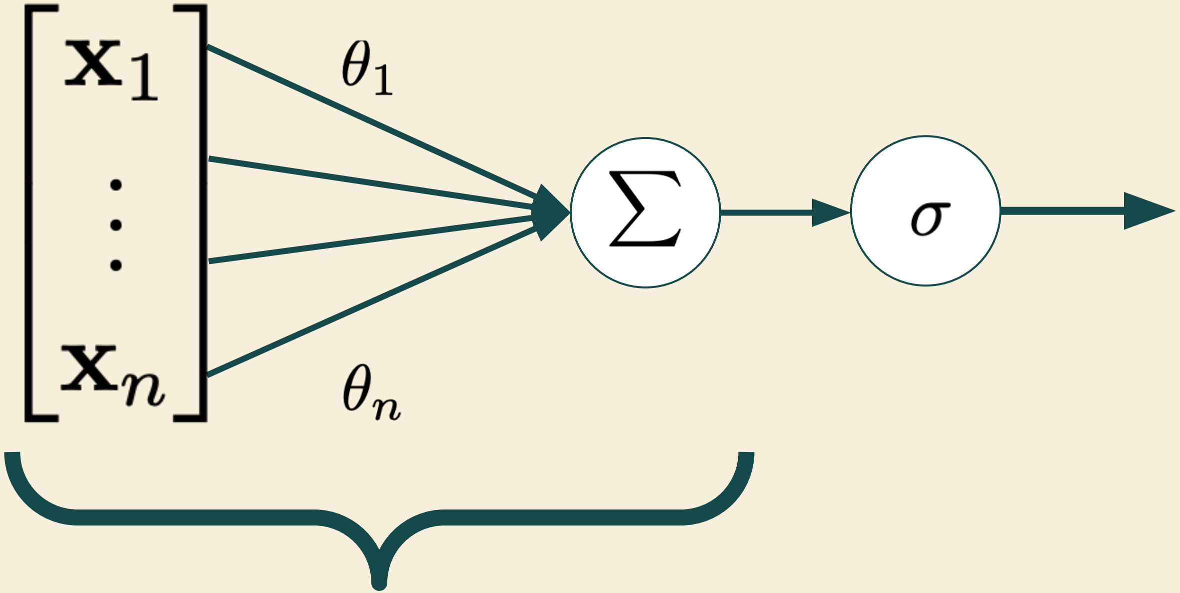
Learning Objectives

- Understand the history of deep learning
- Formulation of the multi-layer perceptron
- Deep Networks and the need for auto-diff
- Forward-mode automatic differentiation
- Reverse-mode automatic differentiation

Formulating the multilayer perceptron

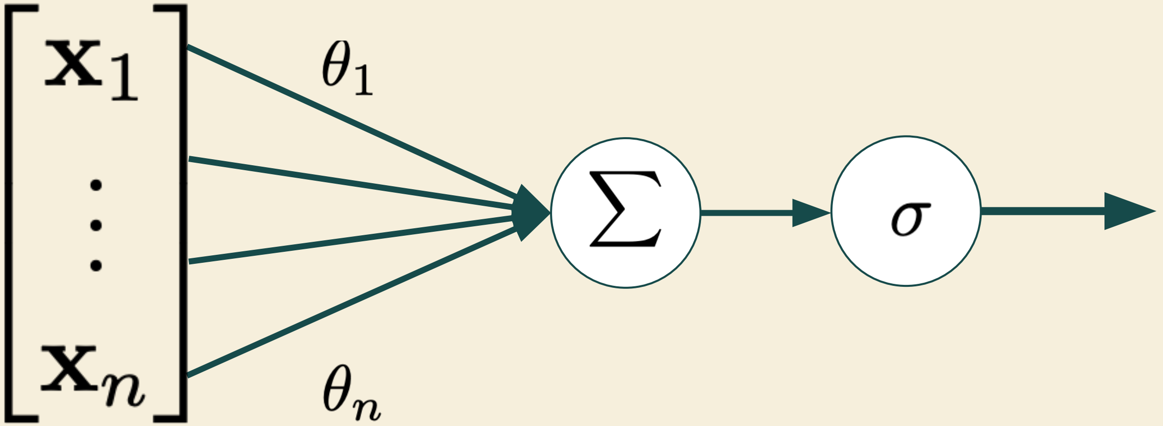


Formulating the MLP



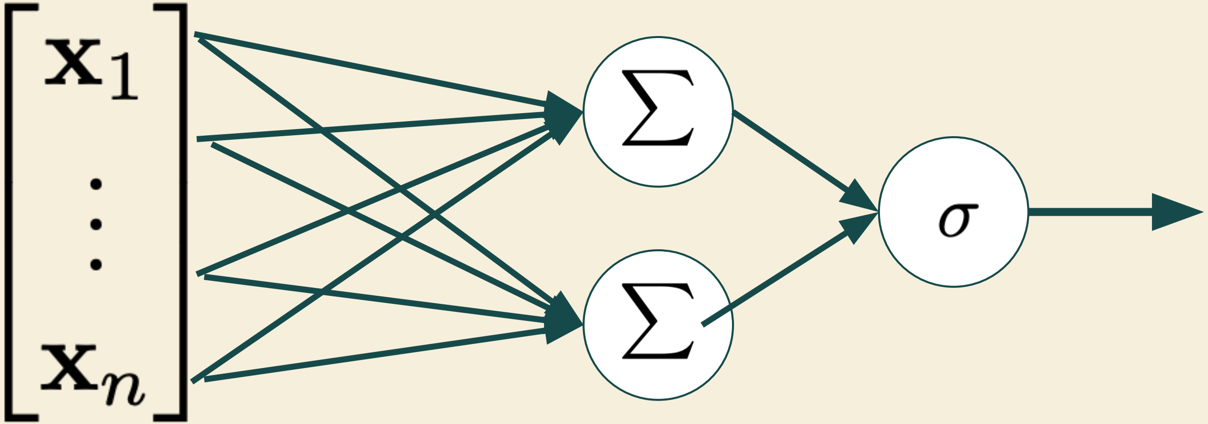
$$\underbrace{\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}}_{\sum_i \mathbf{x}_i \theta_i}$$

Formulating the MLP

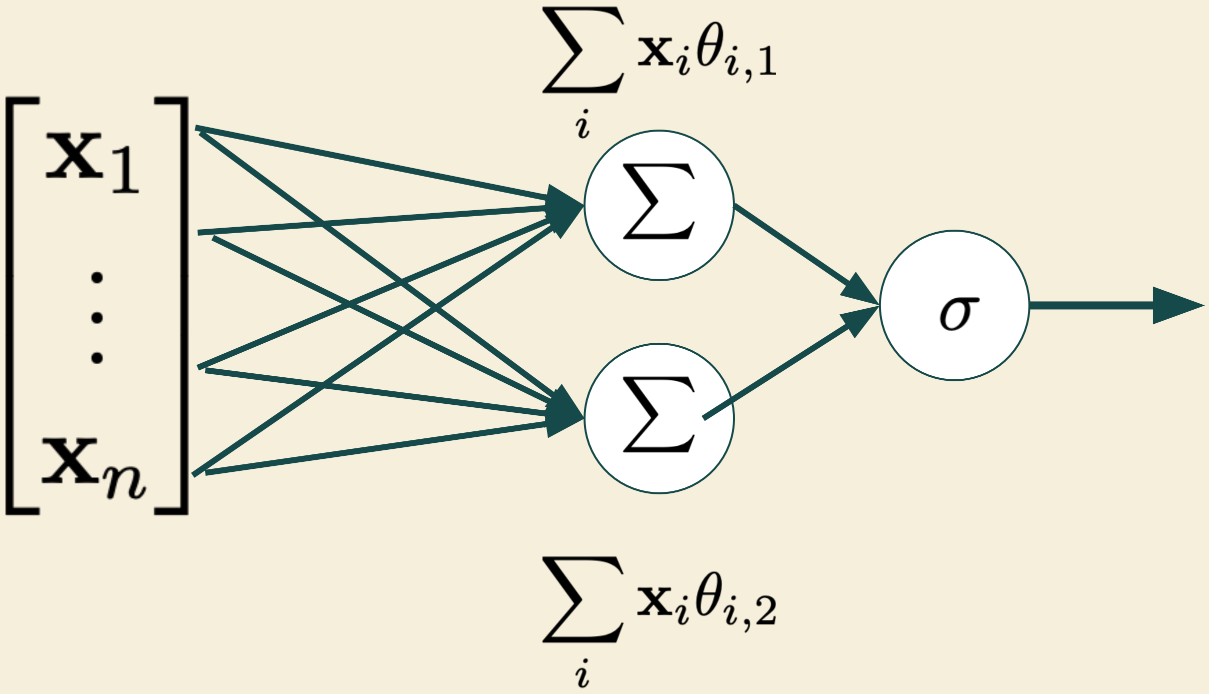


$$\sum_i \mathbf{x}_i\theta_i \qquad \sigma\left(\sum_i \mathbf{x}_i\theta_i\right)$$

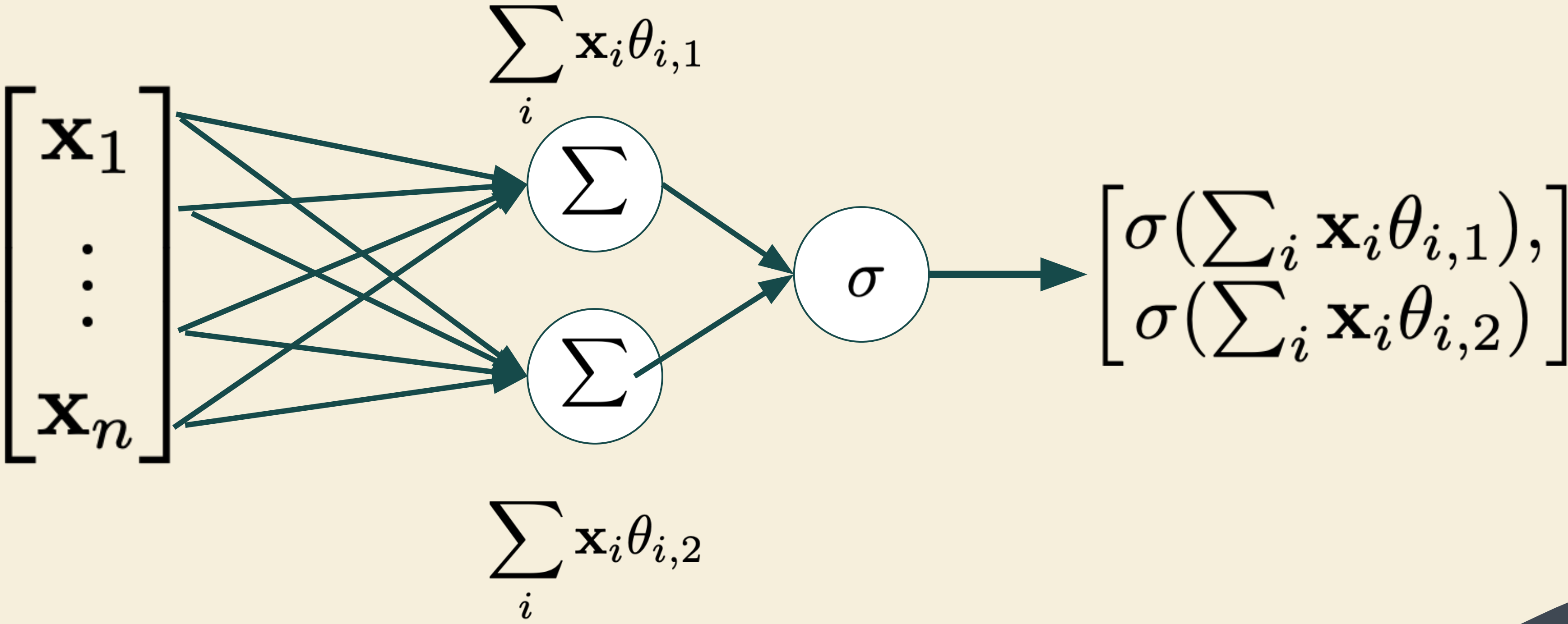
Formulating the MLP



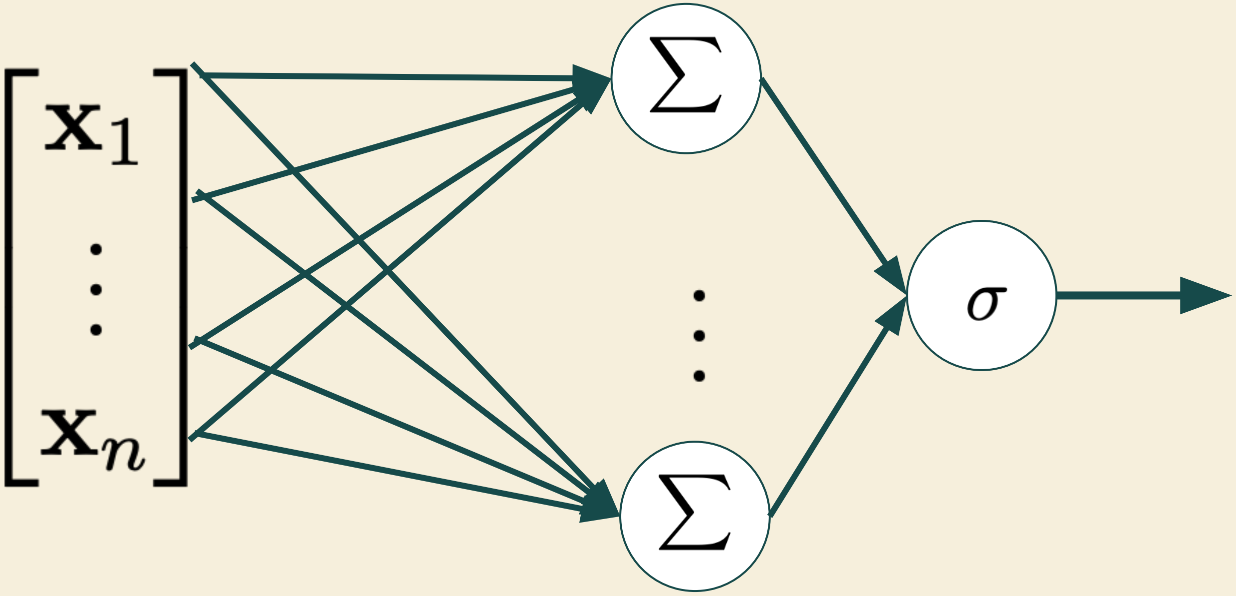
Formulating the MLP



Formulating the MLP

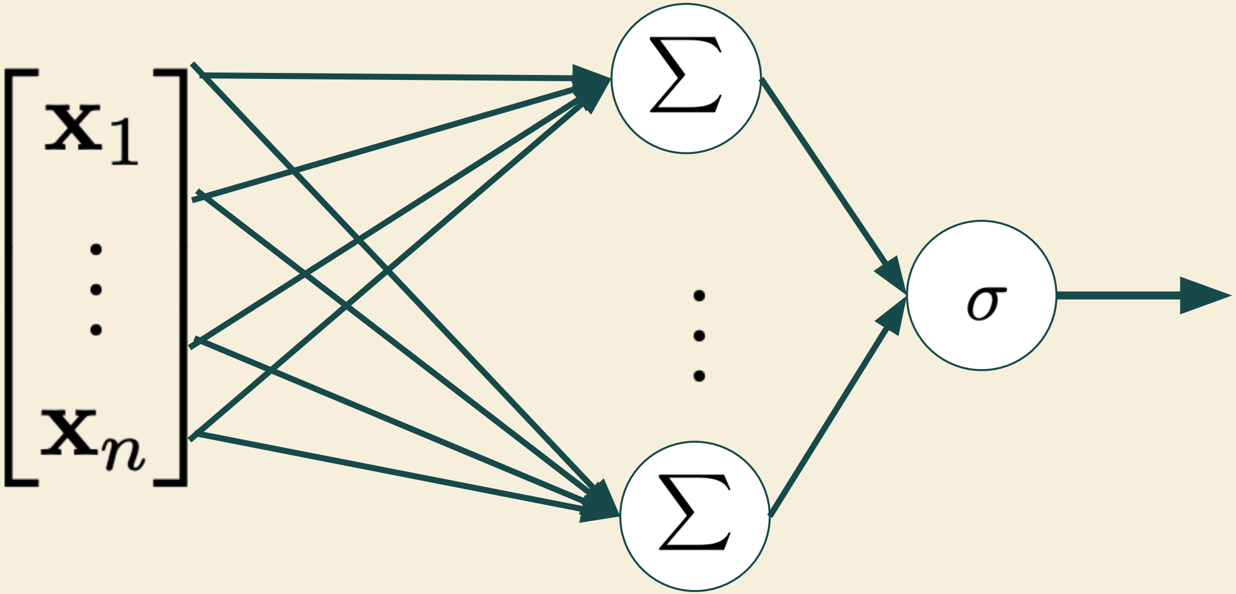


Formulating the MLP



$$\sum_i \mathbf{x}_i \theta_{i,j}$$

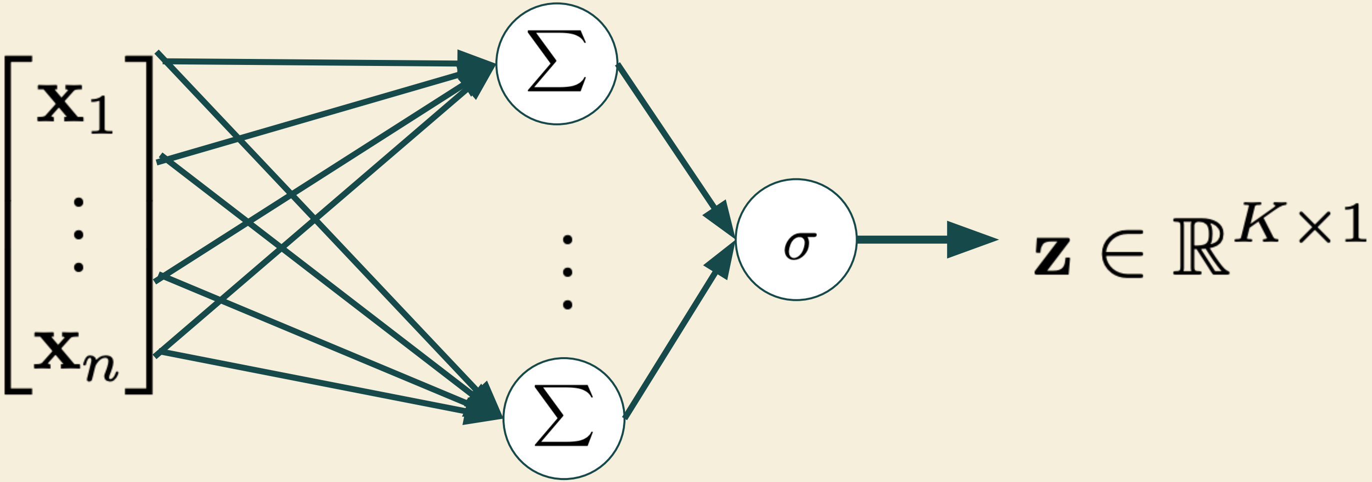
Formulating the MLP



Matrix-vector product (just need to get shapes right)

$$\sum_i \mathbf{x}_i \theta_{i,j}$$

Formulating the MLP

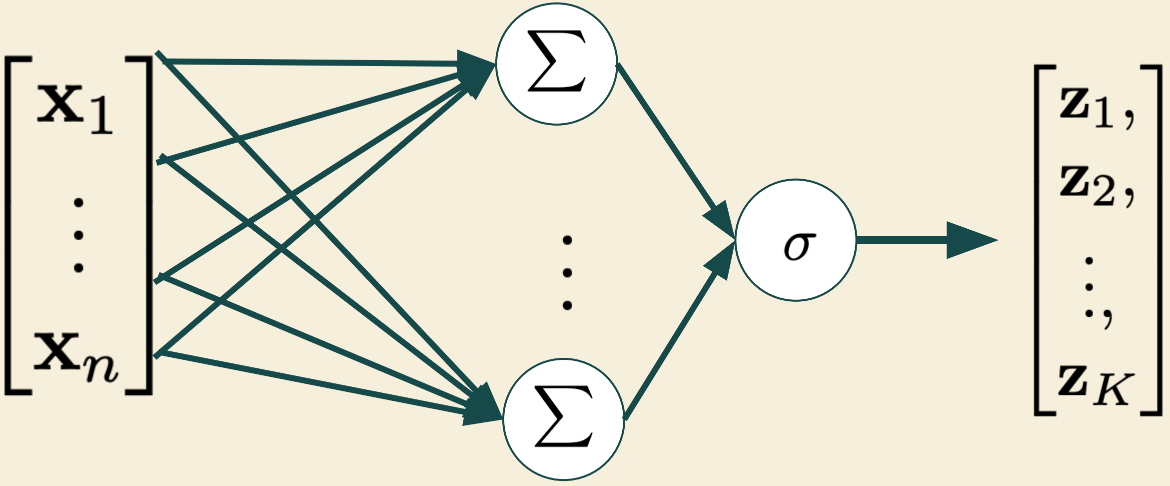


Matrix-vector product

$$\sum_i \mathbf{W}_{i,j} \mathbf{x}_i$$

$K \times n$ $\qquad$ $n \times 1$ $\qquad$ $\rightarrow K \times 1$

Formulating the MLP

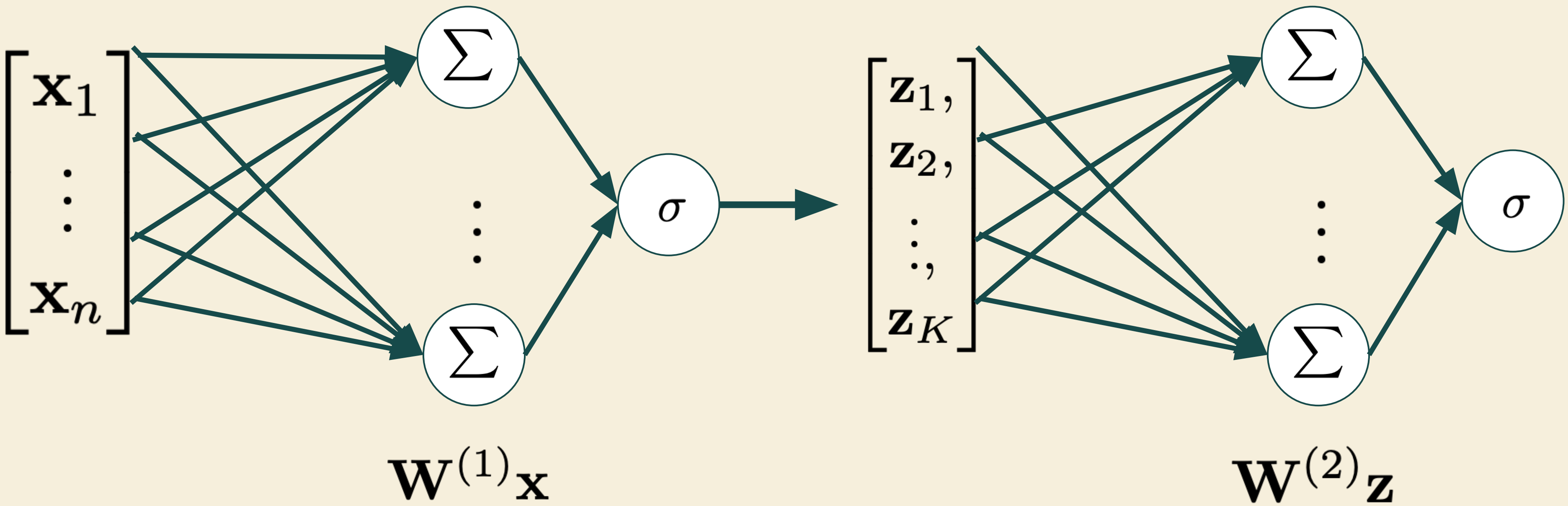


Matrix-vector product

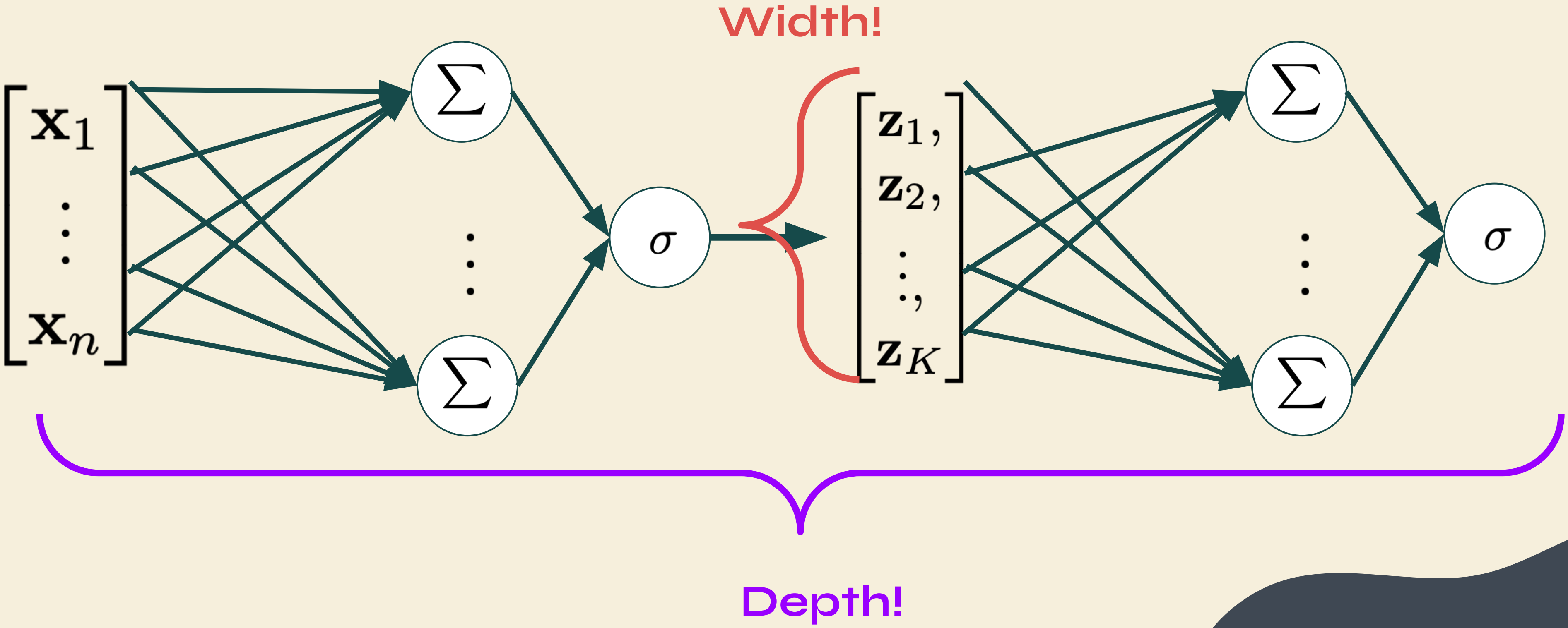
$$\sum_i \mathbf{W}_{i,j}\mathbf{x}_i$$

$$K \times n \qquad\qquad n \times 1$$

Formulating the MLP



Formulating the MLP



Mathematical abstraction

$$\mathbf{z}^{(1)} = \mathbf{x}$$

$$\mathbf{z}^{(i+1)} = \sigma(\mathbf{W}^{(i)}\mathbf{z}^{(i)} + \mathbf{b}^{(i)})$$

Learning in a 1-layer NN

$$\mathbf{z}^{(1)} = \mathbf{x}$$

$$\mathbf{z}^{(i+1)} = \sigma(\mathbf{W}^{(i)}\mathbf{z}^{(i)} + \mathbf{b}^{(i)})$$

Learning in fully connected networks

$$\hat{y} = \sigma(\mathbf{Wx})$$

$$\mathcal{L} = (\sigma(\mathbf{Wx}) - \mathbf{y})^2$$

Activation function is very generally defined and so we don't have anything analytically tractable in general

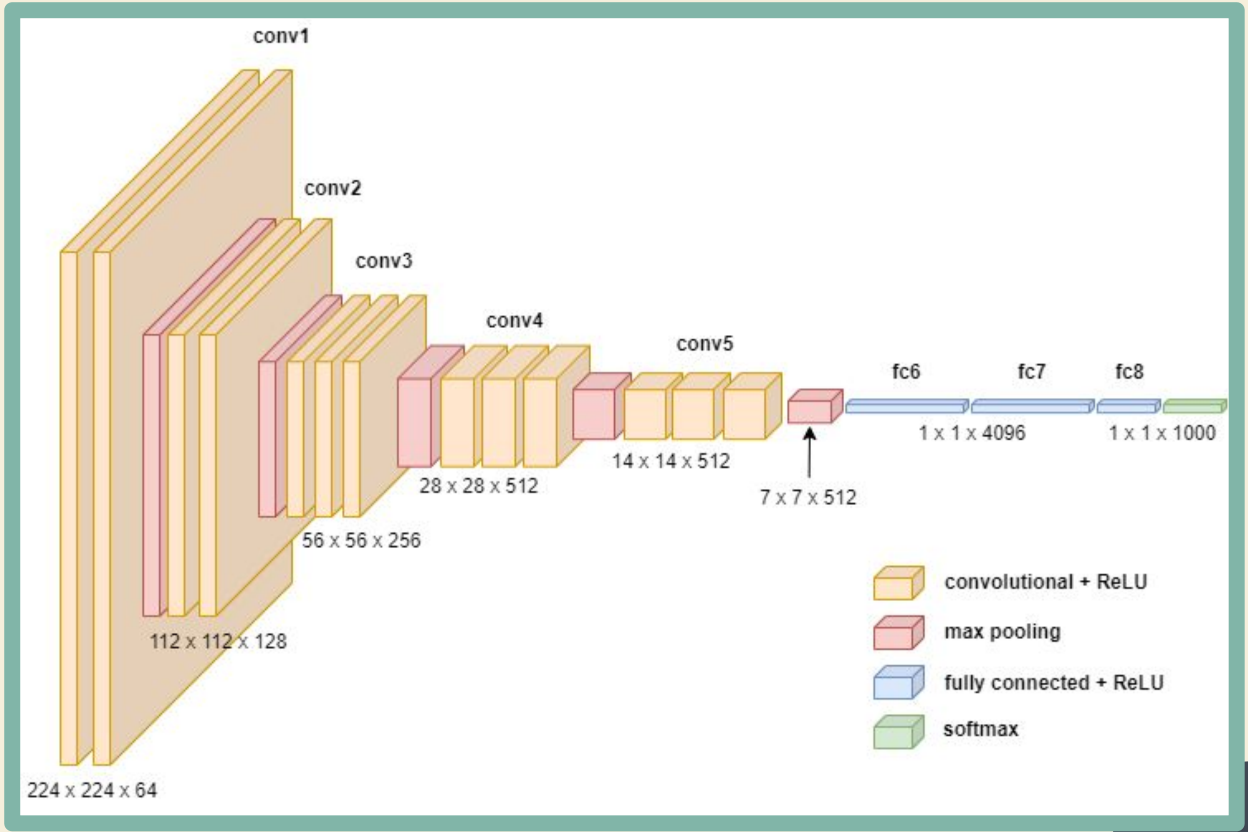
Learning in fully connected networks

$$\log(1 + \mathrm{e}^{wx+b})$$

$$\frac{e^{b_1+b_2+w_1x+w_2\log\left[1+e^{b_1+w_1x}\right]}w_2x}{\left(1+e^{b_1+w_1x}\right)\left(1+e^{b_2+w_2\log\left[1+e^{b_1+w_1x}\right]}\right)}$$

Activation function is very generally defined and so we don't have anything analytically tractable in general

Learning in more complex models



Break time!

(save questions for the end of lecture)

Automatic differentiation

We want to be able to differentiate extremely complex functions without needing to symbolically differentiate.

We call this numerical differentiation

Automatic differentiation

$$f : \mathbb{R}^n \to \mathbb{R}$$

What about the finite difference method?

$$\frac{\partial f}{\partial x_1} = \lim_{h \to 0} \frac{f(x_1 + h, x_2, \ldots, x_n) - f(x)}{h},$$

$$\frac{\partial f}{\partial x_2} = \lim_{h \to 0} \frac{f(x_1, x_2 + h, \ldots, x_n) - f(x)}{h},$$

$$\vdots$$

$$\frac{\partial f}{\partial x_n} = \lim_{h \to 0} \frac{f(x_1, x_2, \ldots, x_n + h) - f(x)}{h}$$

Automatic differentiation

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$\frac{\partial f}{\partial x_1} = \lim_{h \to 0} \frac{f(x_1 + h, x_2, \dots, x_n) - f(x)}{h},$$

$$\frac{\partial f}{\partial x_2} = \lim_{h \to 0} \frac{f(x_1, x_2 + h, \dots, x_n) - f(x)}{h},$$

$$\vdots$$

$$\frac{\partial f}{\partial x_n} = \lim_{h \to 0} \frac{f(x_1, x_2, \dots, x_n + h) - f(x)}{h}$$

What about the finite difference method?

Too slow, too approximate :(

Automatic differentiation

We want to be able to differentiate extremely complex functions without needing to symbolically differentiate.

Let's come up with a systematic way of computing numerical derivatives!

Forward Mode Auto. Diff.

Key idea: along with computing the forward pass values, also keep track of the derivative. We will see this from a programming perspective and a *mathematical one*.

Forward Mode Auto. Diff.

Consider the function:

$$f(\mathbf{x}) = h(g(\mathbf{x}))$$

Forward Mode Auto. Diff.

Consider the function:

$$f(\mathbf{x}) = h(g(\mathbf{x}))$$

$$f(\mathbf{x}) = (h \circ g)(\mathbf{x}) = h(g(x))$$

Aside: you will also see function composition this way

Forward Mode Auto. Diff.

Consider the function:

$$f(\mathbf{x}) = h(g(x))$$

Chain rule tells us that:

$$\mathbf{J}_f = \mathbf{J}_{h \circ g} = \mathbf{J}_h(g(\mathbf{x}))\mathbf{J}_g(x)$$

Jacobian

diff outside, diff inside

$f = h(g(a(x)))$

$J_f = J_h(g(a(x))) J_g(a(x)) J_a(x)$

compute $a(x)$: y_1, do $J_a(x)$

compute $g(y_1) = y_2$, do $J_g(y_1)$

compute $h(y_2)$, do $J_h(y_2)$

Forward Mode Auto. Diff.

Generalizing the above function we have:

$$f = f^{(L)} \circ f^{(L-1)} \circ \ldots \circ f^{1}$$

$f^{L}(f^{L-1}(f^{L-2}(\ldots)))$

Which in turn gives us the Jacobians:

$$\mathbf{J} = \mathbf{J}^{(L)} \cdot \mathbf{J}^{(L-1)} \cdot \ldots \cdot \mathbf{J}^{(1)}$$

diff of outside × diff of inside

Forward Mode Auto. Diff.

To compute the gradient-input product we can use the products of Jacobians formula to see:

mult both sides by x

$$\mathbf{J}\mathbf{x} = \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\dots\mathbf{J}^{(2)}(\underbrace{\mathbf{J}^{(1)}\mathbf{x}})$$

Compute the derivative of the first operation first!

Forward Mode Auto. Diff.

To compute the gradient-input product we can use the products of Jacobians formula to see:

$x^{(1)}$

$J^{(1)}$ = Jacobian w.r.t x ?

$$\mathbf{J}\mathbf{x} = \mathbf{J}^{(L)}\mathbf{J}^{(L-1)} \ldots \mathbf{J}^{(2)}(\mathbf{J}^{(1)}\mathbf{x})$$

Jacobian w.r.t $x^{(1)}$?

$$= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)} \ldots \mathbf{J}^{(3)}\underbrace{(\mathbf{J}^{(2)}\mathbf{x}^{(1)})}_{\text{Use that result to compute the next Jacobian product}}$$

$J_x = J^L J^{L-1} \ldots J^3 J^2 (J^1 x)$

$= \ldots J^3 J^2$

Keep track of deriv of f^n w.r.t input for which you calc. the deriv

Forward Mode Auto. Diff.

$$\mathbf{J}\mathbf{x} = \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(2)}(\mathbf{J}^{(1)}\mathbf{x})$$

$$= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(3)}(\mathbf{J}^{(2)}\mathbf{x}^{(1)})$$

$$= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(4)}(\mathbf{J}^{(3)}\mathbf{x}^{(2)})$$

$$\ldots$$

$$= \mathbf{J}^{L}\mathbf{x}^{(L-1)}$$

Continue until we have just the final Jacobian

Programming Perspective: Operator Overloading

```
class AdVar:
'''
 We want to keep track of both the value and
 the derivative of our variable, so it will
  hold two values one representing each
 '''
   def __init__(self,**kwargs):
       self.val = 0
       self.der = 1

       # re-assign these default values
       if 'val' in kwargs:
           self.val = kwargs['val']
       if 'der' in kwargs:
           self.der = kwargs['der']
```

allows operators like +, -, * etc. to have diff. meanings based on context

Programming Perspective: Operator Overloading

```
class AdVar:
'''
 We want to keep track of both the value and
 the derivative of our variable, so it will
  hold two values one representing each
 '''
    def __init__(self,**kwargs):
        self.val = 0
        self.der = 1

        # re-assign these default values
        if 'val' in kwargs:
            self.val = kwargs['val']
        if 'der' in kwargs:
            self.der = kwargs['der']
```

```
def add(a,b):
    # Create output evaluation and derivative object
    c = AdVar() <- val=0, der=1
    # switch to determine if a or b is a constant
    if type(a) != AdVar: ie a is a const
        c.val = a + b.val
        c.der = b.der
    elif type(b) != AdVar: ie b is const
        c.val = a.val + b
        c.der = a.der
    else:
        c.val = a.val + b.val
        c.der = a.der + b.der
    return c
```

keep track of derivative

When we update our value, we must also update our derivative!

Consider the simple neural network:

$$\mathbf{z}^{(1)} = \mathbf{x}$$ $\to$ do $J_{z^{(1)}}(1,)$

$$\mathbf{z}^{(2)} = \tanh(\mathbf{W}^{(1)}\mathbf{z}^{(1)})$$ $\to$ do $J_{z^{(2)}}(z^{(1)})$

$$\mathbf{z}^{(3)} = \mathbf{W}^{(2)}\mathbf{z}^{(2)}$$ $\to$ do $J_{z^{(3)}}(z^{(2)})$

Consider the simple neural network:

$$\mathbf{z}^{(1)} = \mathbf{x}$$

$$\mathbf{z}^{(2)} = \tanh(\mathbf{W}^{(1)}\mathbf{z}^{(1)})$$

$$\mathbf{z}^{(3)} = \mathbf{W}^{(2)}\mathbf{z}^{(2)}$$

We start by considering all of the operations that are needed to get us from the input to the loss function

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)}))\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

We start by considering all of the operations that are needed to get us from the input to the loss function

(Sorry I switched indexing schemes)

starts at $z^{(0)}$ here

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

Now, lets write out all of the derivative rules we need!

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\left(\mathbf{W}^{(1)}, \tanh(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)}))\right)$$

sig $z^{(2)}$ sig $z^{(1)}$ x

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

Operation	Value Update	Derivative Update
Addition of a constant c	$g(w) + c$	$\frac{d}{dw}(g(w) + c) = \frac{d}{dw}g(w)$
Multiplication by a constant c	$cg(w)$	$\frac{d}{dw}(cg(w)) = c\frac{d}{dw}g(w)$
Raising to a power n	$g(w)^n$	$\frac{d}{dw}(g(w)^n) = n(g(w)^{n-1})\frac{d}{dw}g(w)$
Applying Tanh	$tanh(g(w))$	$\frac{d}{dw}(tanh(g(w))) = 1 - tanh^2(g(w))\frac{d}{dw}$

Consider the simple neural network:

Operation	Value Update	Derivative Update
Addition of a constant c	$g(w)+c$	$\frac{d}{dw}(g(w)+c) = \frac{d}{dw}g(w)$
Multiplication by a constant c	$cg(w)$	$\frac{d}{dw}(cg(w)) = c\frac{d}{dw}g(w)$
Raising to a power n	$g(w)^n$	$\frac{d}{dw}(g(w)^n) = n(g(w)^{n-1})\frac{d}{dw}g(w)$
Applying Tanh	$tanh(g(w))$	$\frac{d}{dw}(tanh(g(w))) = 1 - tanh^2(g(w))\frac{d}{dw}$

Now lets go through each operation and write out how we update the value and how we update the derivative

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

Value	Derivative
$x_0 = x$ $\swarrow$ $z^{(0)} = 1$	$d_0 = \mathbf{1}$ deriv of input = 1

diff w.r.t x (x is input)

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

$[\ ][\ ] = [\ ]$

$Ax \to A_i = \sum_j a_{ij} x_j$ $\quad \frac{\partial}{\partial x_a} = \sum a_{ij} \delta_{aj}$

$= \sum a_{ia}$

Value		Derivative	
$x_0 = x$		$d_0 = \mathbf{1}$	d.y $x_0 = d0$ (= 1)
$x_1 = W^{(0)} x_0$	d.y w.r.t x_0 $\longrightarrow$	$d_1 = W^{(0)\top} d_0$	

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \mathrm{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\mathrm{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

Value	Derivative
$x_0 = x$	$d_0 = \mathbf{1}$
$x_1 = W^{(0)} x_0$	$d_1 = W^{(0)\top} d_0$
$x_2 = \tanh(x_1)$	$d_2 = 1 - \tanh^2(x_1) d_1$

diff w.r.t x_1

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

Operation	Value Update	Derivative Update
Addition of a constant c	$g(w) + c$	$\frac{d}{dw}(g(w) + c) = \frac{d}{dw}g(w)$
Multiplication by a constant c	$cg(w)$	$\frac{d}{dw}(cg(w)) = c\frac{d}{dw}g(w)$
Raising to a power n	$g(w)^n$	$\frac{d}{dw}(g(w)^n) = n(g(w)^{n-1})\frac{d}{dw}g(w)$
Applying Tanh	$tanh(g(w))$	$\frac{d}{dw}(tanh(g(w))) = 1 - tanh^2(g(w))\frac{d}{dw}$

intermediate variables

Value

$x_0 = x$

$x_1 = W^{(0)}x_0$

$x_2 = \tanh(x_1)$

$= z^{(2)}$

$x_3 = W^{(1)}x_2$

$x_4 = (x_3 - y)$ $y - x_3$

$= \text{Loss}$

$x_5 = (x_4)^2$

Derivative

$d_0 = \mathbf{1}$

transposed as d_0 is a row vector by convention

$\frac{\partial x_1}{\partial x_0} =$ $d_1 = W^{(0)\top}d_0$ $W^{(0)\top}\frac{\partial}{\partial x_1}(x_0) = W^{(0)\top}d_0$

$\frac{\partial x_2}{\partial x_1}$ $d_2 = 1 - \tanh^2(x_1)d_1$ $= 1 - \tanh^2(x_1)\frac{\partial}{\partial x}(x_1) = 1 - \tanh^2 d_1$

$d_3 = W^{(1)\top}d_2$

$\frac{\partial x_4}{\partial x_3} =$ $d_4 = d_3$

$\frac{\partial x_5}{\partial x_4}$ $d_5 = -2(x_4)d_4$ $2x_4 \frac{\partial}{\partial x}(x_4) =$

$x_3 - y$ is -ve. tbh this -ve would come out in d_4. x_4 should technically be $y - x_3 \rightarrow d_4 = -d_3$ so $d_5 = 2x_4 d_4$, where $d_4 = -d_3$

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

Value	Derivative
$x_0 = x$	$d_0 = \mathbf{1}$
$x_1 = W^{(0)} x_0$	$d_1 = W^{(0)\top} d_0$
$x_2 = \tanh(x_1)$	$d_2 = 1 - \tanh^2(x_1) d_1$
$x_3 = W^{(1)} x_2$	$d_3 = W^{(1)\top} d_2$
$x_4 = (x_3 - y)$	$d_4 = d_3$
$x_5 = (x_4)^2$	$d_4 = -2(x_4) d_4$

$$x = 1, W^{(0)} = 2, W^{(1)} = 1.2$$

diff w.r.t x

$x_0 = 1 = z^{(0)}$	$d_0 = 1$

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

$= 1$

$$x = 1, W^{(0)} = 2, W^{(1)} = 1.2$$

what is deriv. w.r.t. x at $x=1$

$x_0 = 1$	$d_0 = 1$
$x_1 = 1 * 2 = 2$	$d_1 = 2 * d_0 = 2$

Value	Derivative
$x_0 = x$	$d_0 = \mathbf{1}$
$x_1 = W^{(0)}x_0$	$d_1 = W^{(0)\top}d_0$
$x_2 = \tanh(x_1)$	$d_2 = 1 - \tanh^2(x_1)d_1$
$x_3 = W^{(1)}x_2$	$d_3 = W^{(1)\top}d_2$
$x_4 = (x_3 - y)$	$d_4 = d_3$
$x_5 = (x_4)^2$	$d_4 = -2(x_4)d_4$

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

$$x = 1, W^{(0)} = 2, W^{(1)} = 1.2$$

$x_0 = 1$	$d_0 = 1$
$x_1 = 1 * 2 = 2$	$d_1 = 2 * d_0 = 2$
$x_2 = \tanh(2) \approx 0.964$	$d_2 = (1 - \tanh^2(2))d_1 \approx 0.14$

Value	Derivative
$x_0 = x$	$d_0 = \mathbf{1}$
$x_1 = W^{(0)}x_0$	$d_1 = W^{(0)\top}d_0$
$x_2 = \tanh(x_1)$	$d_2 = 1 - \tanh^2(x_1)d_1$
$x_3 = W^{(1)}x_2$	$d_3 = W^{(1)\top}d_2$
$x_4 = (x_3 - y)$	$d_4 = d_3$
$x_5 = (x_4)^2$	$d_4 = -2(x_4)d_4$

Consider the simple neural network:

$$\mathbf{z}^{(2)} = \text{Matmul}\Big(\mathbf{W}^{(1)}, \tanh\big(\text{Matmul}(\mathbf{W}^{(0)}\mathbf{z}^{(0)})\big)\Big)$$

$$\mathcal{L} = (\mathbf{y} - \mathbf{z}^{(2)})^2$$

Value	Derivative
$x_0 = x$	$d_0 = \mathbf{1}$
$x_1 = W^{(0)} x_0$	$d_1 = W^{(0)\top} d_0$
$x_2 = \text{tanh}(x_1)$	$d_2 = 1 - \text{tanh}^2(x_1) d_1$
$x_3 = W^{(1)} x_2$	$d_3 = W^{(1)\top} d_2$
$x_4 = (x_3 - y)$	$d_4 = d_3$
$x_5 = (x_4)^2$	$d_4 = -2(x_4) d_4$

$$x = 1, W^{(0)} = 2, W^{(1)} = 1.2$$

For x_4, x_5, d_4, d_5, alternate version which makes more sense in 'or's' - $L = y - z^{(2)}$ rather than $z^{(2)} - y$

$x_0 = 1$	$d_0 = 1$
$x_1 = 1 * 2 = 2$	$d_1 = 2 * d_0 = 2$
$x_2 = \text{tanh}(2) \approx 0.964$	$d_2 = (1 - \text{tanh}^2(2)) d_1 \approx 0.14$
$x_3 = 0.964 * 1.2 \approx 1.15$	$d_3 = 1.2 * d_2 \approx 0.169$
$(x_3 - y) \rightarrow$ $x_4 = (1.15 - 2) \approx -0.85$ or $y - x_3 = 0.85$	$d_4 = d_3 = 0.169$ or $d_4 = -d_3 = -0.169$
$x_5 = (-0.85)^2 = 0.722$ or $x_5 = (x_4)^2$	$d_5 = 2(-0.85) d_4 = -0.283$ or $d_5 = 2(x_4)d_4 = 2*0.85*-0.169 = -0.283$

Taking a step back

A single pass of forward mode automatic differentiation computes the "directional derivative" with respect to x. By computing it with respect to ei, we can get one column of the Jacobian

$$
\begin{aligned}
\mathbf{J}\mathbf{x} &= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\dots\mathbf{J}^{(2)}(\mathbf{J}^{(1)}\mathbf{x}) \\
&= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\dots\mathbf{J}^{(3)}(\mathbf{J}^{(2)}\mathbf{x}^{(1)}) \\
&= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\dots\mathbf{J}^{(4)}(\mathbf{J}^{(3)}\mathbf{x}^{(2)}) \\
&\dots \\
&= \mathbf{J}^{L}\mathbf{x}^{(L-1)}
\end{aligned}
$$

Jacobian:

$$
\begin{pmatrix}
\frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \dots \\
\frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \\
\vdots & &
\end{pmatrix}
$$

Taking a step back

To get the full Jacobian we would need to compute as many forward mode sweeps as we have columns
But! In ML problems we have MANY columns and few rows, so this seems inefficient.

ML usually many inputs, few outputs so inefficient

$$
\begin{aligned}
\mathbf{J}\mathbf{x} &= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(2)}(\mathbf{J}^{(1)}\mathbf{x}) \\
&= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(3)}(\mathbf{J}^{(2)}\mathbf{x}^{(1)}) \\
&= \mathbf{J}^{(L)}\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(4)}(\mathbf{J}^{(3)}\mathbf{x}^{(2)}) \\
&\ldots \\
&= \mathbf{J}^{L}\mathbf{x}^{(L-1)}
\end{aligned}
$$

Reverse Mode Auto. Diff.

Reverse mode is also known as adjoint mode because we are interested in computing the adjoint derivatives:

grad of final output w.r.t that variable

$$\bar{x} = \frac{\partial z}{\partial x}$$

Reverse Mode Auto. Diff.

Reverse mode is also known as adjoint mode because we are interested in computing the adjoint derivatives:

$$\mathbf{J}\mathbf{x} = (\mathbf{y}\mathbf{J}^{(L)})\mathbf{J}^{(L-1)} \dots \mathbf{J}^{(1)}$$

Reverse Mode Auto. Diff.

Reverse mode is also known as adjoint mode because we are interested in computing the adjoint derivatives:

$$\begin{aligned}\mathbf{J}\mathbf{x} &= (\mathbf{y}\mathbf{J}^{(L)})\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(1)}\\ &= (\bar{\mathbf{y}}^{(L)}\mathbf{J}^{(L-1)})\mathbf{J}^{(L-2)}\ldots\mathbf{J}^{(1)}\end{aligned}$$

Reverse Mode Auto. Diff.

Reverse mode is also known as adjoint mode because we are interested in computing the adjoint derivatives:

$$
\begin{aligned}
\mathbf{J}\mathbf{x} &= (\mathbf{y}\mathbf{J}^{(L)})\mathbf{J}^{(L-1)}\ldots\mathbf{J}^{(1)} \\
&= (\bar{\mathbf{y}}^{(L)}\mathbf{J}^{(L-1)})\mathbf{J}^{(L-2)}\ldots\mathbf{J}^{(1)} \\
&\ldots \\
&= \bar{\mathbf{y}}^{(2)}\mathbf{J}^{(1)}
\end{aligned}
$$

Key Observations

$$\mathbf{Jx} = (\mathbf{yJ}^{(L)})\mathbf{J}^{(L-1)} \ldots \mathbf{J}^{(1)}$$

This is the initial co-tangent (output) that we will compute all our adjoints with respect to! But how did we get it?
-> we have to compute f(x) first

Key Observations

$$\mathbf{J}\mathbf{x} = (\mathbf{y}\mathbf{J}^{(L)})\mathbf{J}^{(L-1)} \dots \mathbf{J}^{(1)}$$

Similarly each adjoint requires us to use the intermediate computations that got us to y

This is the initial co-tangent (output) that we will compute all our adjoints with respect to! But how did we get it?
-> we have to compute f(x) first

Key Observations

- We need to complete a complete forward pass to get our initial cotangent
- We need to store all of our intermediate computations to compute the adjoints we need

Note: we have a discussion of reverse mode for NNs in the notes, but won't have time in lecture to go through that.

Computational Graphs

$$f(x) = \sqrt{x^2 + \exp(x^2)} + cos\left(x^2 + exp(x^2)\right)$$

Neural networks are actually a particularly easy/nice case for adjoint/reverse mode auto. diff. so let's look at something a little harder!

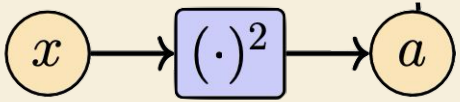
Computational Graphs

$$f(x) = \sqrt{x^2 + \exp(x^2)} + cos\left(x^2 + exp(x^2)\right)$$

1. Identify the elementary operations and look for the ones that occur most and for shared structure
2. We will have variable nodes leading to operations.
3. Start with the most common operations and then build the graph outwards

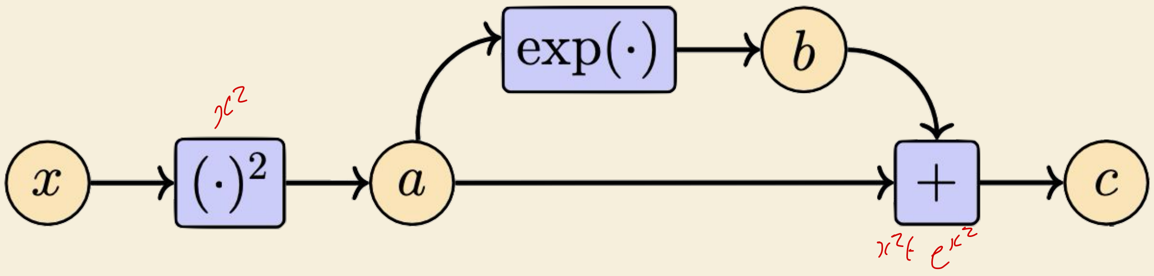
Computational Graphs

$$f(x) = \sqrt{x^2 + \exp(x^2)} + cos\left(x^2 + exp(x^2)\right)$$



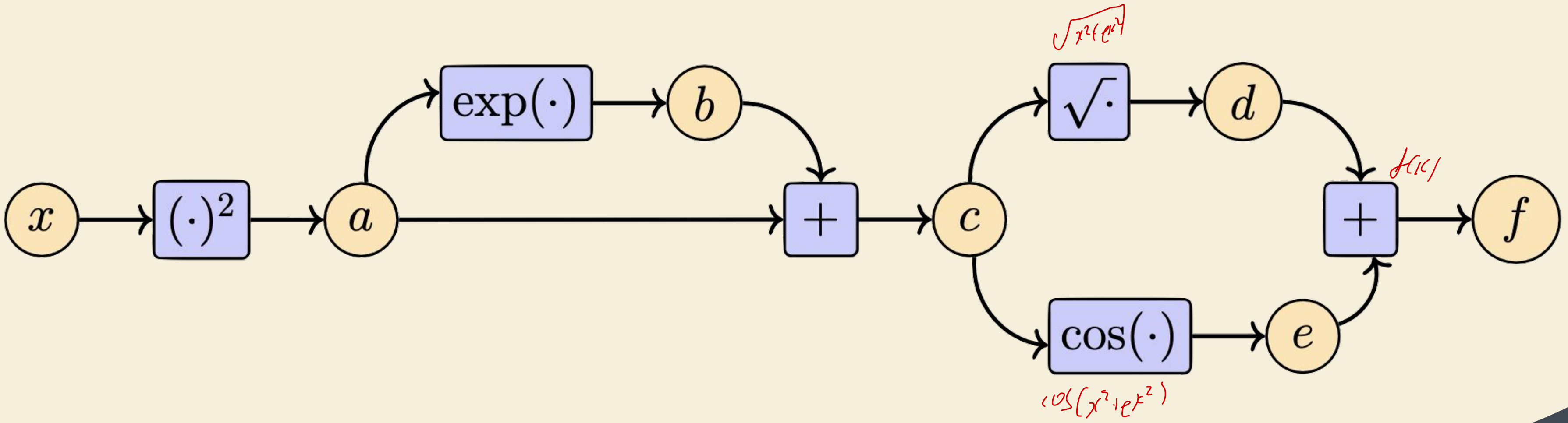
Computational Graphs

$$f(x) = \sqrt{x^2 + \exp(x^2)} + cos\left(x^2 + exp(x^2)\right)$$



Computational Graphs

$$f(x) = \sqrt{x^2 + \exp(x^2)} + cos(x^2 + exp(x^2))$$



Computational Graphs

$$f(x) = \sqrt{x^2 + \exp(x^2)} + cos(x^2 + exp(x^2))$$

$$\frac{\partial a}{\partial x} = 2x$$

$d = \sqrt{c}$ $$\frac{\partial d}{\partial c} = \frac{1}{2\sqrt{c}}$$

$b = e^a$ $$\frac{\partial b}{\partial a} = \exp(a)$$

$e = cos(c)$ $$\frac{\partial e}{\partial c} = -\sin(c)$$

$c = a+b$ $$\frac{\partial c}{\partial a} = 1 = \frac{\partial c}{\partial b}$$

$f = d+e$ $$\frac{\partial f}{\partial d} = 1 = \frac{\partial f}{\partial e}.$$

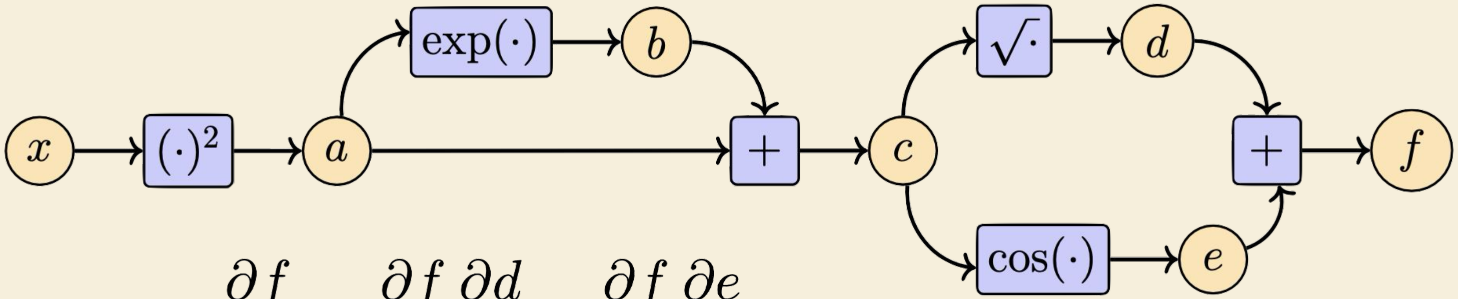
Computational Graphs

$$f(x) = \sqrt{x^2 + \exp(x^2)} + cos\left(x^2 + exp(x^2)\right)$$

To compute the derivative we want, we have to back propagate through all of the nodes in the graph using the chain rule for each incoming node and summing them together

$$\frac{\partial f}{\partial x_i} = \sum_{x_j : x_i \in \mathrm{Pa}(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i} = \sum_{x_j : x_i \in \mathrm{Pa}(x_j)} \frac{\partial f}{\partial x_j} \frac{\partial g_j}{\partial x_i}$$

Computational Graphs



$$
\begin{aligned}
\frac{\partial f}{\partial c} &= \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} \\
\frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} \\
\frac{\partial f}{\partial a} &= \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} \\
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} \,.
\end{aligned}
$$

Next Lecture:

When and why does gradient descent give us a good parameter value?