CS6421: Deep Neural Networks

Gregory Provan

Spring 2020

Lecture 4: Mathematical Foundations

Based on notes from Marc Diesenroth

Overview

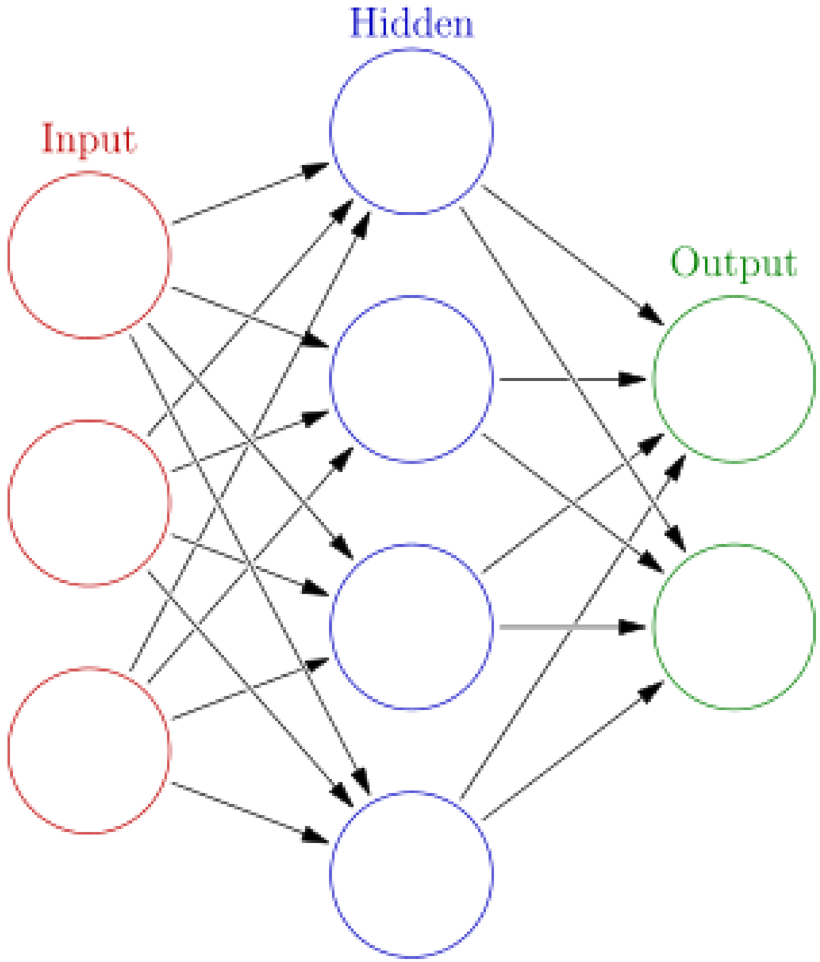
- Mathematical Basis of Deep Learning
- Linear Algebra
- Multivariate Calculus

Key Mathematical Concepts

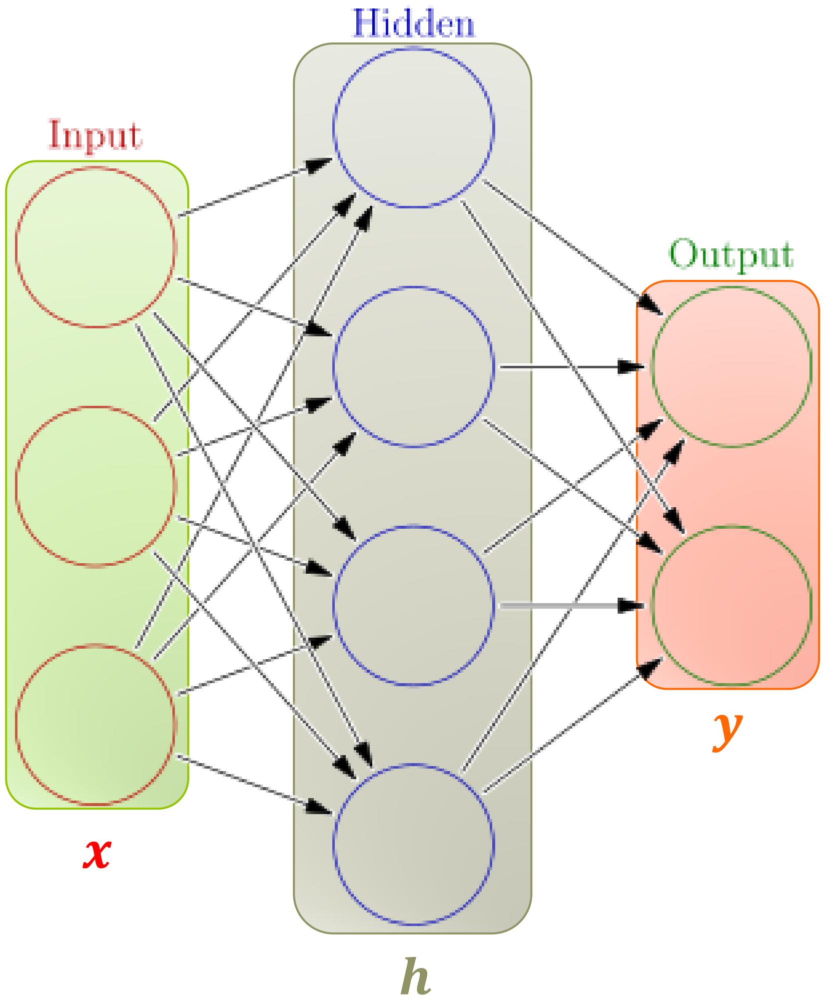
- Linear algebra and matrix decomposition
- **Differentiation**
- **Integration**
- Optimization
- Probability theory and Bayesian inference
- Functional analysis

Neural Networks Basics

- Operations
  - Forward prop: makes predictions
  - Backward prop: adjusts parameters



Neural Network: Definition



Weights

$$\boldsymbol{h} = \sigma(\mathbf{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)$$

$$\boldsymbol{y} = \sigma(\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2)$$

Activation functions

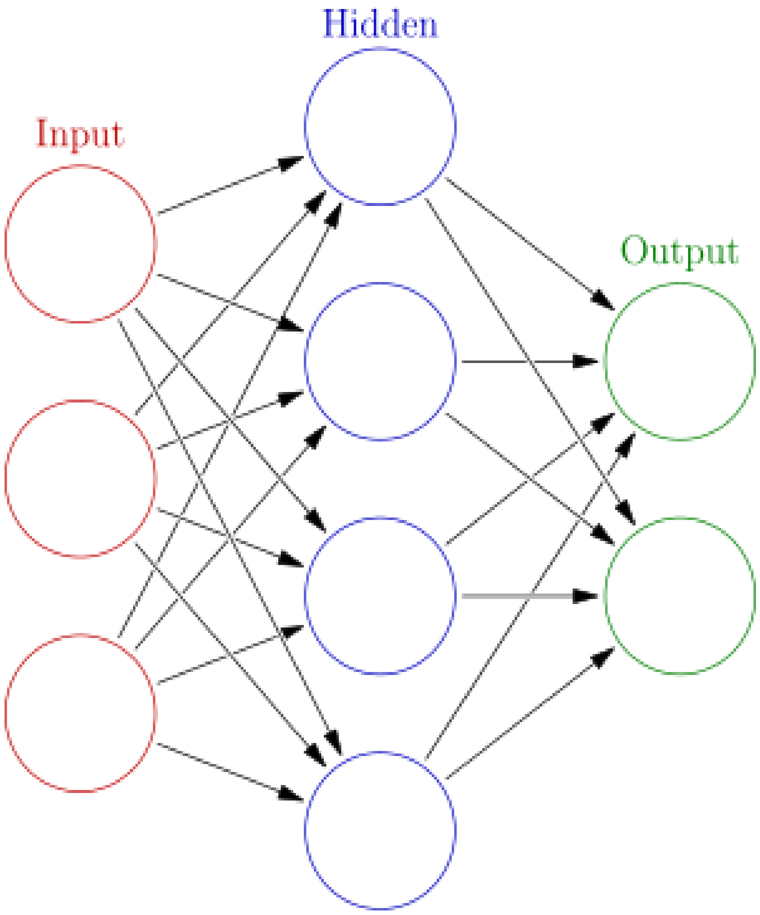
What are the underlying mathematical operations?

Forward Propagation

$$\vec{X} \rightarrow \vec{H} \rightarrow \hat{\vec{Y}}$$

$$\vec{H} = relu\left(\left([W] * \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}\right) + [b]\right)$$

$$\hat{\vec{Y}} = relu\left(\left([U] * \begin{bmatrix} H_1 \\ H_2 \\ H_3 \\ H_4 \end{bmatrix}\right) + [c]\right)$$



Forward Propagation

$$\vec{H} = relu\left(\left([W] * \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}\right) + [b]\right)$$

Matrix Multiplication

- Changes Dimensionality

$$(m * n) * (n * p) = (m * p)$$

$$(m * n) * (3 * 1) = (4 * 1)$$

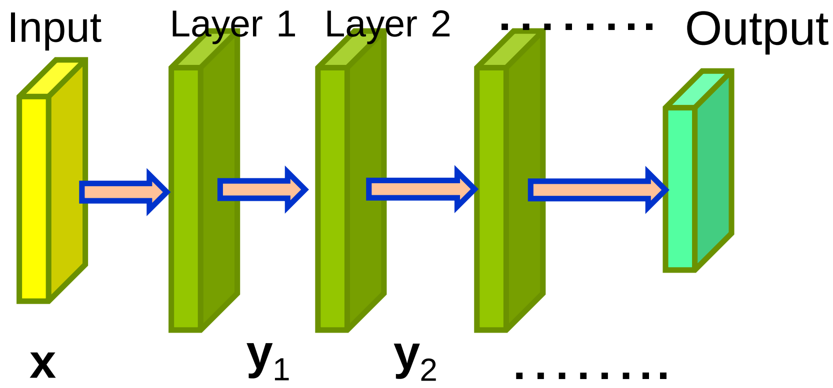
$$(4 * 3) * (3 * 1) = (4 * 1)$$

$$\vec{H} = relu\left(\left(\begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \\ W_{41} & W_{42} & W_{43} \end{bmatrix} * \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}\right) + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}\right)$$

Deep Network: Generic Definition

- A family of parametric, non-linear and hierarchical representation learning functions
  - massively optimized with stochastic gradient descent to encode domain knowledge, i.e. domain invariances, stationarity.
- $a^L(x; w^1, \ldots, w^L) = h^L(h^{L-1} \ldots h^1(x, w^1), w^{L-1}), w^L)$
  - x:input,
  - w^l: parameters for layer l,
  - $a^l = h^l(x, w^l)$: (non-) linear function
- Given training corpus $\{X, Y\}$ find optimal parameters
  - $w^* \leftarrow \text{argmin}_w \sum_{(x,y) \subseteq (X,Y)} L(y, a^L(x))$

Forward Propagation: Deep Network



$$\mathbf{y}_1 = f_1(\mathbf{x}, \theta_1)$$

$$\mathbf{y}_2 = f_2(\mathbf{y}_1, \theta_2)$$

$$\mathbf{y}_2 = f_2(f_1(\mathbf{x}, \theta_1), \theta_1)$$

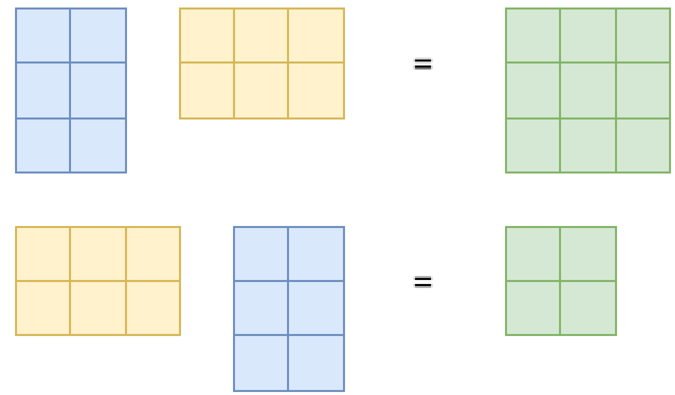
Deep Network: family of parametric, non-linear, hierarchical representations

$$\mathbf{y}_N(\mathbf{x}, \theta_1, \theta_2, \ldots \theta_N) = f_N(f_{N-1}(\ldots(f_1(\mathbf{x}, \theta_1), \theta_{N-1}), \theta_N)$$

Implemented as matrix/vector operations

Matrix Multiplication

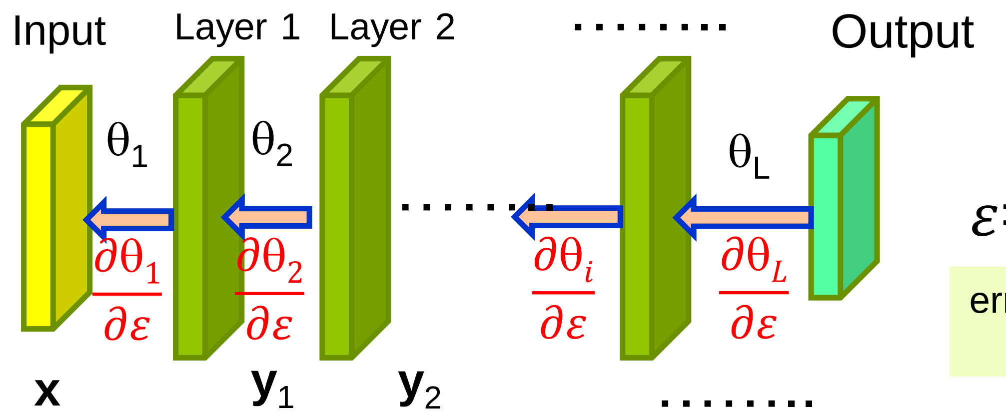
- Matrix multiplication is not commutative, i.e., **AB≠BA**
- When multiplying matrices, the “neighbouring” dimensions have to fit:



$$\underbrace{\boldsymbol{A}}_{n \times k} \underbrace{\boldsymbol{B}}_{k \times m} = \underbrace{\boldsymbol{C}}_{n \times m}$$

$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$	`y = A.dot(x)`
$y_i = \sum_j A_{ij} x_j$	`y = np.einsum('ij, j', A, x)`
$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$	`C = A.dot(B)`
$C_{ij} = \sum_k A_{ik} B_{kj}$	`C = np.einsum('ik, kj', A, B)`

Back Propagation: Deep Network



ε= t - y

error = true label – computed label

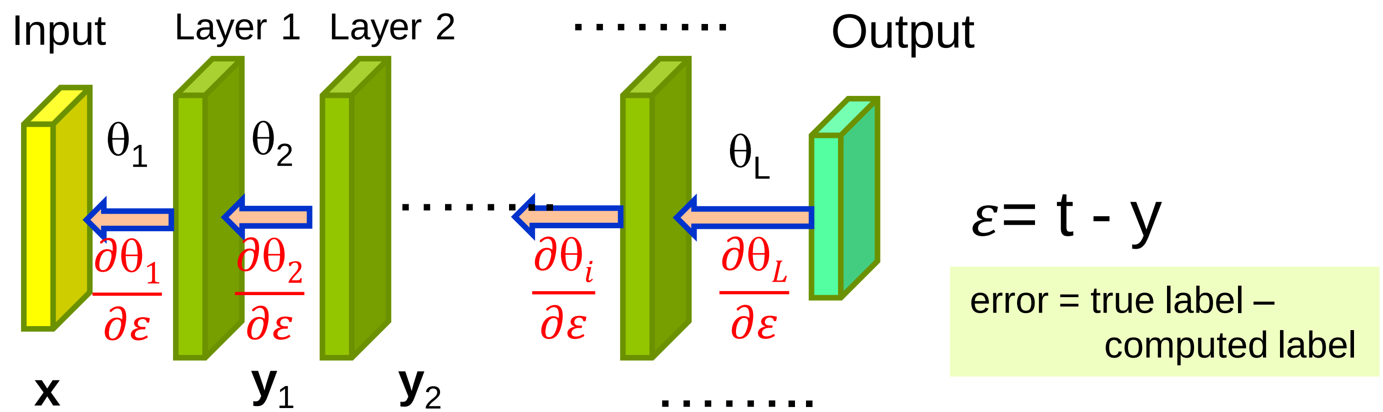
Compute how weights contribute to the error ε

$$\frac{\partial\theta_i}{\partial\varepsilon}$$

Training: optimize network parameters to minimise loss over training set

$$\theta^{*} = \arg\min \sum_{(x,y)\in(X,Y)} J[y, f^L(x, \theta_{1,\ldots L})]$$

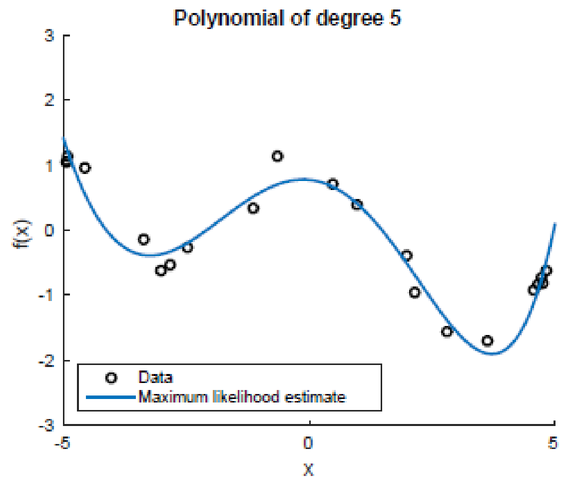
Training: Computes Derivatives



- Must use partial derivative plus chain rule
- Shows that backpropagation can be done layer by layer

Network Training

- Training data, e.g., N pairs $(\boldsymbol{x}_i, y_i)$ of inputs $\boldsymbol{x}_i$ and observations y_i
- Training the model means finding parameters $\boldsymbol{\theta}^*$, such that $f(\boldsymbol{x}_i, \boldsymbol{\theta}^*) \approx y_i$



- Define a loss function, e.g., $\sum_{i=1}^{N}(y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta}))^2$, which we want to optimize
- Typically: Optimization based on some form of gradient descent
  ▶▶ Differentiation required

Requirement for Differentiation

- Differentiate arbitrary formulae
  - Loss function: $L=\Psi(t-y)^2$
  - Activation function: tanh
  - Algebraic operation: Wx
- Rules of differentiation must be used
  - Sum, product, chain, etc.

Differentiation: Rules

- Sum Rule

$$\left(f(x)+g(x)\right)' = f'(x) + g'(x) = \frac{df}{dx} + \frac{dg}{dx}$$

- Product Rule

$$\left(f(x)g(x)\right)' = f'(x)g(x) + f(x)g'(x) = \frac{df}{dx}g(x) + f(x)\frac{dg}{dx}$$

- Chain Rule

$$(g \circ f)'(x) = \left(g(f(x))\right)' = g'(f(x))f'(x) = \frac{dg}{df}\frac{df}{dx}$$

Example

$$(g \circ f)'(x) = (g(f(x)))' = g'(f(x))f'(x) = \frac{dg}{df}\frac{df}{dx}$$

$$\begin{aligned} g(z) &= \tanh(z) \\ z &= f(x) = x^n \\ (g \circ f)'(x) &= \underbrace{(1 - \tanh^2(x^n))}_{dg/df}\underbrace{nx^{n-1}}_{df/dx} \end{aligned}$$

Partial Derivatives

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

$$y = f(\boldsymbol{x}), \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N$$

- Partial derivative (change one coordinate at a time):

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_N) - f(\boldsymbol{x})}{h}$$

- Jacobian vector (gradient) collects all partial derivatives:

$$\frac{df}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{x_N} \end{bmatrix} \in \mathbb{R}^{1 \times N}$$

Note: This is a row vector.

Example

$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R}$$

- Partial derivatives:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 x_2 + x_2^3$$
$$\frac{\partial f(x_1, x_2)}{\partial x_2} = x_1^2 + 3x_1 x_2^2$$

- Gradient:

$$\frac{df}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(x_1,x_2)}{\partial x_1} & \frac{\partial f(x_1,x_2)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 x_2 + x_2^3 & x_1^2 + 3x_1 x_2^2 \end{bmatrix} \in \mathbb{R}^{1 \times 2} .$$

Example: Chain Rule

- Consider the function

$$L(\boldsymbol{e}) = \tfrac{1}{2}\|\boldsymbol{e}\|^2 = \tfrac{1}{2}\boldsymbol{e}^\top \boldsymbol{e}$$

$$\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\,, \quad \boldsymbol{x} \in \mathbb{R}^N\,, \boldsymbol{A} \in \mathbb{R}^{M\times N}\,, \boldsymbol{e}, \boldsymbol{y} \in \mathbb{R}^M$$

- Compute $dL/d\boldsymbol{x}$. What is the dimension/size of $dL/d\boldsymbol{x}$?
- $dL/d\boldsymbol{x} \in \mathbb{R}^{1\times N}$

$$\frac{dL}{d\boldsymbol{x}} = \frac{dL}{d\boldsymbol{e}}\frac{d\boldsymbol{e}}{d\boldsymbol{x}}$$

$$\frac{dL}{d\boldsymbol{e}} = \boldsymbol{e}^\top \in \mathbb{R}^{1\times M}$$

$$\frac{d\boldsymbol{e}}{d\boldsymbol{x}} = -\boldsymbol{A} \in \mathbb{R}^{M\times N}$$

$$\Rightarrow \frac{dL}{d\boldsymbol{x}} = \boldsymbol{e}^\top(-\boldsymbol{A}) = -(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{A} \in \mathbb{R}^{1\times N}$$

Jacobian Matrix

$$f : \mathbb{R}^N \to \mathbb{R}^M$$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, \ldots, x_N) \\ \vdots \\ f_M(x_1, \ldots, x_N) \end{bmatrix}$$

- Jacobian matrix (collection of all partial derivatives)

$$\begin{bmatrix} \frac{dy_1}{d\boldsymbol{x}} \\ \vdots \\ \frac{dy_M}{d\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} \in \mathbb{R}^{M \times N}$$

Example

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}\,, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N$$

- Compute the gradient $d\boldsymbol{f}/d\boldsymbol{x}$
  - Dimension of $d\boldsymbol{f}/d\boldsymbol{x}$:
    Since $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$, it follows that $d\boldsymbol{f}/d\boldsymbol{x} \in \mathbb{R}^{M\times N}$
  - Gradient:

$$f_i = \sum_{j=1}^{N} A_{ij}x_j \qquad \Rightarrow \frac{\partial f_i}{\partial x_j} = A_{ij}$$

$$\Rightarrow \frac{d\boldsymbol{f}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MN} \end{bmatrix} = \boldsymbol{A}$$

Example

- Consider $f : \mathbb{R}^2 \to \mathbb{R}, \quad \boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$

$$f(\boldsymbol{x}) = f(x_1, x_2) = x_1^2 + 2x_2\,,$$

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}$$

- The dimensions $df/d\boldsymbol{x}$ and $d\boldsymbol{x}/dt$ are 1×2 and 2×1, respectively
- Compute the gradient df/dt using the chain rule.

$$\frac{df}{dt} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{bmatrix}$$

$$= \begin{bmatrix} 2\sin t & 2 \end{bmatrix} \begin{bmatrix} \cos t \\ -\sin t \end{bmatrix}$$

$$= 2\sin t \cos t - 2\sin t = 2\sin t(\cos t - 1)$$

Derivatives with Matrices: Summary

- Re-cap: Gradient of a function $f : \mathbb{R}^D \to \mathbb{R}^E$ is an $E \times D$-matrix:

  # target dimensions × # parameters

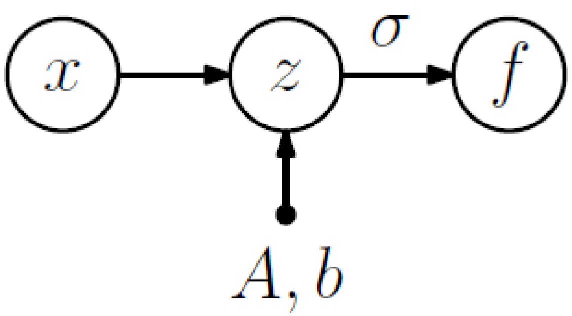
  with

$$\frac{d\boldsymbol{f}}{d\boldsymbol{x}} \in \mathbb{R}^{E \times D}, \qquad df[e,d] = \frac{\partial f_e}{\partial x_d}$$

- Generalization to cases, where the parameters (D) or targets (E) are matrices, apply immediately
- Assume $f : \mathbb{R}^{M \times N} \to \mathbb{R}^{P \times Q}$, then the gradient is a $(P \times Q) \times (M \times N)$ object (tensor) where

$$df[p,q,m,n] = \frac{\partial f_{pq}}{\partial X_{mn}}$$

Gradients of Single-Layer Network



$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z} \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M \times M} \in \mathbb{R}^{M \times M}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}}_{M \times (M \times N)} \in \mathbb{R}^{M \times (M \times N)}$$

Example

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z}\in\mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M\times M} \in \mathbb{R}^{M\times M}$$

$$\begin{aligned} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} &= \operatorname{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M\times M} \\ \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} &= \boldsymbol{I} \in \mathbb{R}^{M\times M} \end{aligned} \tag{5}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}}[i,j] = \sum_{l=1}^{M} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}[i,l] \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}[l,j]$$

```
dfdb = np.einsum('il, lj', dfdz, dzdb)
```

Example (continued)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z} \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}}_{M \times (M \times N)} \in \mathbb{R}^{M \times (M \times N)}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \operatorname{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M \times M} \tag{6}$$

$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}$ ▶▶ See (4)

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}}[i, j, k] = \sum_{l=1}^{M} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}[i, l] \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}[l, j, k]$$

```
dfdA = np.einsum('il, ljk', dfdz, dzdA)
```

Summary: Single Layer NN

- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y} = f(\boldsymbol{z}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}\,, \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$
- Train single-layer neural network with

$$\boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\theta}) = \tanh(\boldsymbol{z})\,, \quad \boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\,, \quad \boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\}$$

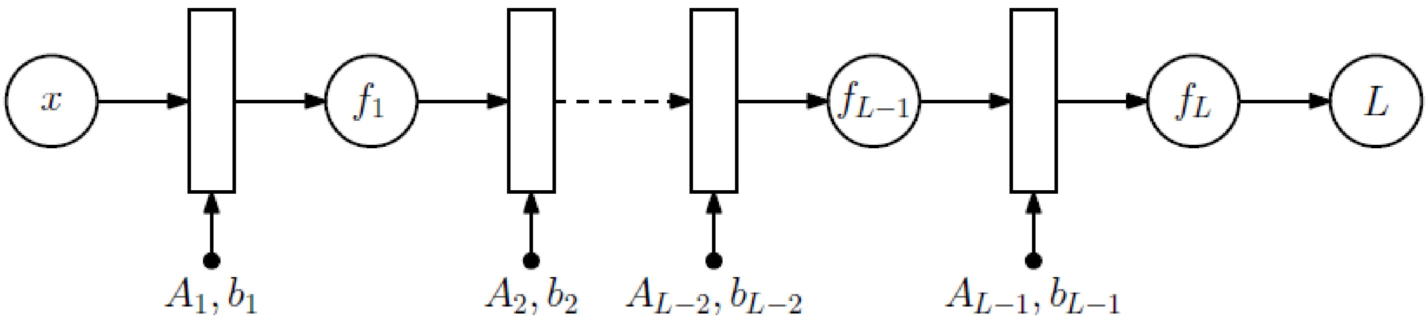
- Find $\boldsymbol{A}, \boldsymbol{b}$, such that the squared loss

$$L(\boldsymbol{\theta}) = \tfrac{1}{2}\|\boldsymbol{e}\|^2\,, \quad \boldsymbol{e} = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\theta})$$

  is minimized
- Partial derivatives:

$$\left.\begin{aligned} \frac{\partial L}{\partial \boldsymbol{A}} &= \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}} \\ \frac{\partial L}{\partial \boldsymbol{b}} &= \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} \end{aligned}\right\} \quad \begin{aligned} &\frac{\partial L}{\partial \boldsymbol{e}} \blacktriangleright\!\blacktriangleright (1) \quad \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}} \blacktriangleright\!\blacktriangleright (2), (3) \quad \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} \blacktriangleright\!\blacktriangleright (6) \\ &\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}} \blacktriangleright\!\blacktriangleright (4) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} \blacktriangleright\!\blacktriangleright (5) \end{aligned}$$

Multi-Layer NN: Gradients



- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y}$
- Train multi-layer neural network with

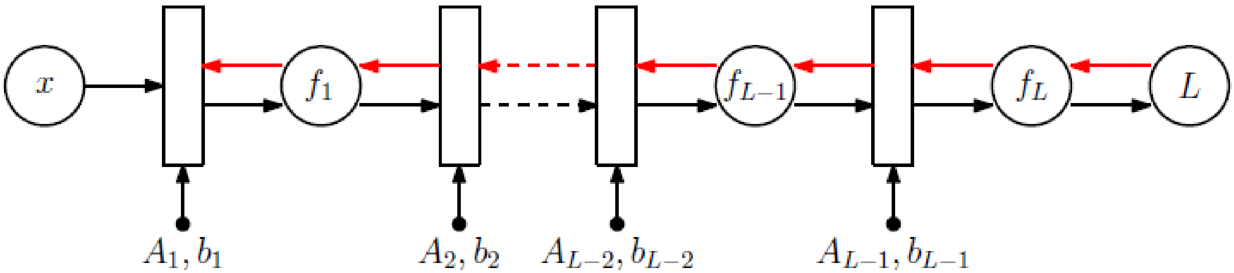
$$\boldsymbol{f}_0 = \boldsymbol{x}$$

$$\boldsymbol{f}_i = \sigma_i(\boldsymbol{A}_{i-1}\boldsymbol{f}_{i-1} + \boldsymbol{b}_{i-1}), \quad i = 1, \dots, L$$

- Find $\boldsymbol{A}_j, \boldsymbol{b}_j$ for $j = 0, \dots, L-1$, such that the squared loss

$$L(\boldsymbol{\theta}) = \|\boldsymbol{y} - \boldsymbol{f}_L(\boldsymbol{\theta}, \boldsymbol{x})\|^2$$

is minimized, where $\boldsymbol{\theta} = \{\boldsymbol{A}_j, \boldsymbol{b}_j\}, \quad j = 0, \dots, L-1$



$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{\theta}_{L-1}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-2}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \boxed{\frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{\theta}_{L-2}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-3}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \boxed{\frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{f}_{L-2}} \frac{\partial \boldsymbol{f}_{L-2}}{\partial \boldsymbol{\theta}_{L-3}}}$$

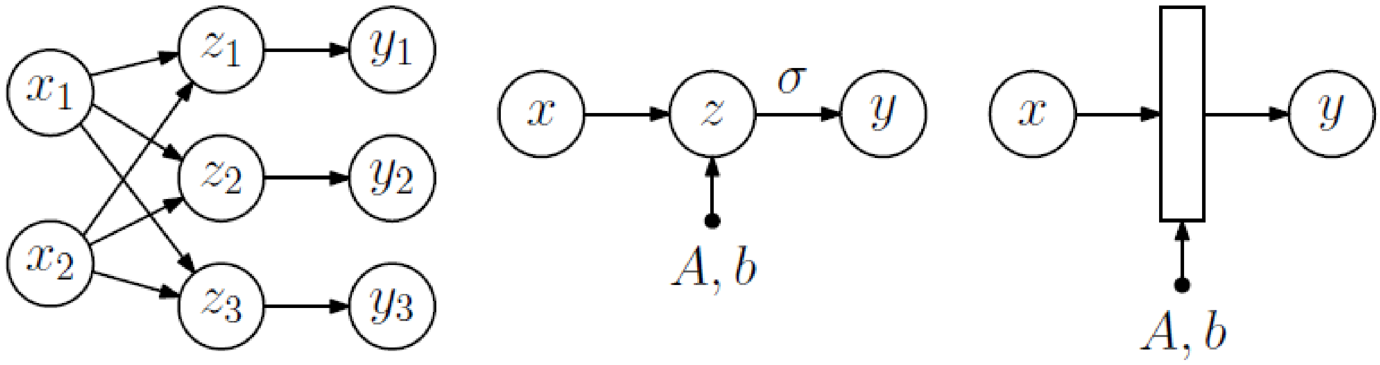
$$\frac{\partial L}{\partial \boldsymbol{\theta}_{i}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \cdots \boxed{\frac{\partial \boldsymbol{f}_{i+2}}{\partial \boldsymbol{f}_{i+1}} \frac{\partial \boldsymbol{f}_{i+1}}{\partial \boldsymbol{\theta}_{i}}}$$

Summary

- **Deep Learning is based on core mathematical principles**
- **Review of**
  - **Linear algebra**
  - **Multivariable calculus (chain rule, etc.)**

Feed-Forward Neural Network

$$\boldsymbol{y} = \sigma(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$



- Training a neural network means parameter optimization:
  - Typically via some form of gradient descent
- **Challenge 1: Differentiation.** Compute gradients of a loss function with respect to neural network parameters **A**, **b**
  - Computing statistics (e.g., means, variances) of predictions
- **Challenge 2: Integration.** Propagate uncertainty through a neural network