DSS
Decision Sciences & Systems

Technische Universität München

# Tutorial Business Analytics & Machine Learning

Tutorial 12: Gradient Descent and Neural Networks

Decision Sciences & Systems (DSS)
Department of Informatics
TU München

# Tutorial Business Analytics and Machine Learning

## Outline

Today's topics:

- Gradient Descent
  - Method
  - Convergence Guarantees
  - Variants
- Neural Networks
  - Backpropagation

# Tutorial Business Analytics and Machine Learning

## Recap – Gradient Descent

- **Goal**, given any function $f: \mathbb{R}^d \rightarrow \mathbb{R}$, find

$$x^* = \underset{x \in \mathbb{R}^d}{\operatorname{argmin}} f(x)$$

# Tutorial Business Analytics and Machine Learning

## Recap – Gradient Descent

- If $f: \mathbb{R}^d \rightarrow \mathbb{R}$ is differentiable, then its **gradient** at position $x$ is defined as

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_d} \end{pmatrix}$$

- In $d$-dimensional space, the gradient points in the direction of steepest ascent of $f$ at point $x$

- Thus $-\nabla f(x)$ is a **descent direction**, i.e. (at least) for small $\alpha > 0$:

$$f\big(x - \alpha \nabla f(x)\big) \leq f(x)$$

# Tutorial Business Analytics and Machine Learning

## Recap – Gradient Descent

- **Gradient Descent:**
  - Set $n = 0$ and start at some $x_0$
  - Calculate $\nabla f(x_n)$
  - Choose a step size (“learning rate”) $\alpha_n$
  - Take a step in the direction opposite of the gradient:

$$x_{n+1} = x_n - \alpha_n \nabla f(x_n)$$

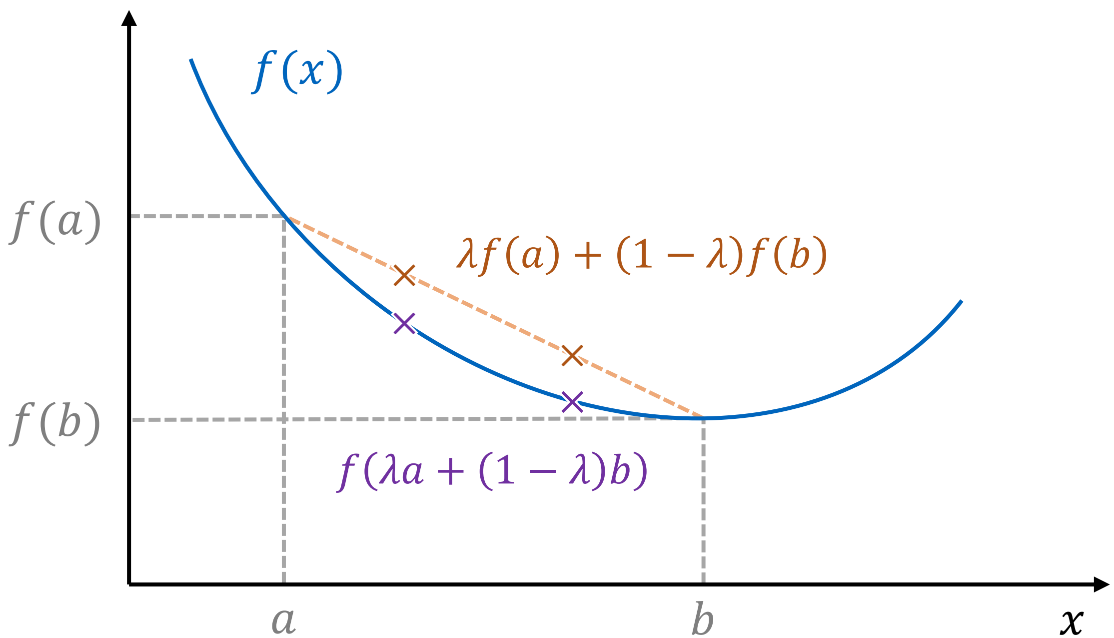
  - Repeat

# Tutorial Business Analytics and Machine Learning

## Convergence Guarantee – Convex Functions

- A function $f: \mathbb{R}^d \to \mathbb{R}$ is called **convex** iff

$$\forall \lambda \in (0,1) \text{ and } a, b \in \mathbb{R}^d: \quad f(\lambda a + (1-\lambda)b) \leq \lambda f(a) + (1-\lambda) f(b)$$



# Tutorial Business Analytics and Machine Learning

## Convergence Guarantee of Gradient Method

- Let $f: \mathbb{R}^d \to \mathbb{R}$ be (strictly) convex and let $\alpha_n$ be chosen, s.t. that they are *square summable*, but not *summable*, i.e.

$$\sum_{n=1}^{\infty} \alpha_n = \infty\,, \qquad \sum_{n=1}^{\infty} \alpha_n^2 < \infty\,,$$

  then $\lim_{n \to \infty} x_n = x^*$

- The rule $\alpha_n = \frac{1}{n}$ fulfills this criterion

# Tutorial Business Analytics and Machine Learning

**Variations of Gradient Descent: Problems with the standard method**

**Compare Exercises 12.1 a-c:**

- Step sizes with convergence guarantees are too small in practice
- Standard gradient descent can get stuck in saddle points or oscillating behavior
- Often, constant step sizes are too small in the beginning and too large towards the end of training

# Tutorial Business Analytics and Machine Learning

## Variations of Gradient Descent: Approaches to deal with these problems

- Line-search or heuristics to find optimal step size dynamically in each step (computationally expensive!)

- Learning-rate decay and elaborate learning-rate schedules

- Add ‘momentum’ to the direction (Exercise 12.1d), to make sudden changes in direction less likely, e.g.

$$d_n = \beta d_{n-1} + \alpha \nabla f(x_{n-1}), \qquad x_n = x_{n-1} - d_n$$

# Tutorial Business Analytics and Machine Learning

## Variations of Gradient Descent: Momentum

- Add ‘momentum’ to the direction, to make sudden changes in direction less likely, e.g.

$$d_n = \beta d_{n-1} + \alpha \nabla f(x_{n-1}), \qquad x_n = x_{n-1} - \ d_n$$

- Several definitions of momentum and many variations and combinations of momentum and learning rate scheduling exist, see
  - https://distill.pub/2017/momentum/ for an in-depth article about how momentum works (with interactive graphics)
  - http://ruder.io/optimizing-gradient-descent/index.html for overview of many variants

- In modern Machine Learning, most common optimization algorithms are Stochastic Gradient Descent and (stochastic versions) of momentum methods such as RMSprop, ADAM, AdaDelta, etc.

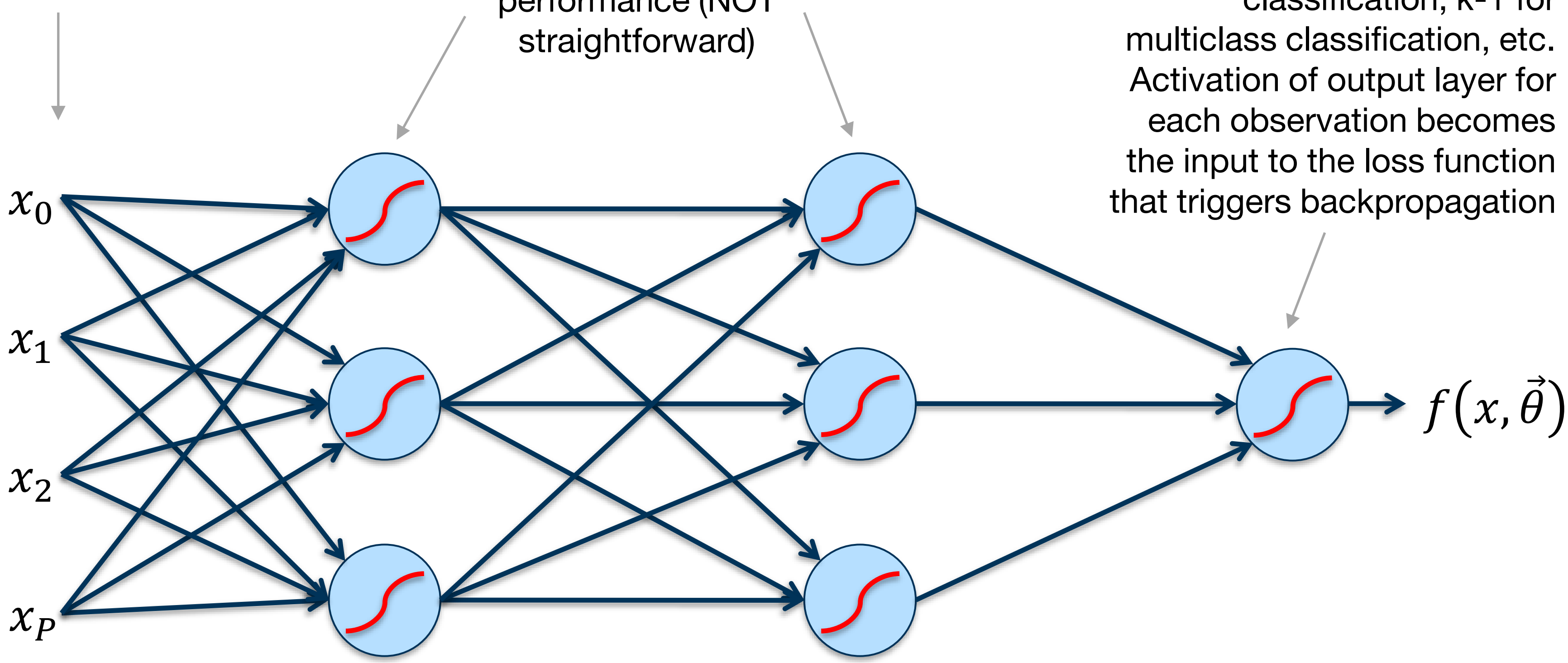
# Tutorial Business Analytics and Machine Learning

## Recap – Neural Networks

**Input Layer:**
depends on data dimension

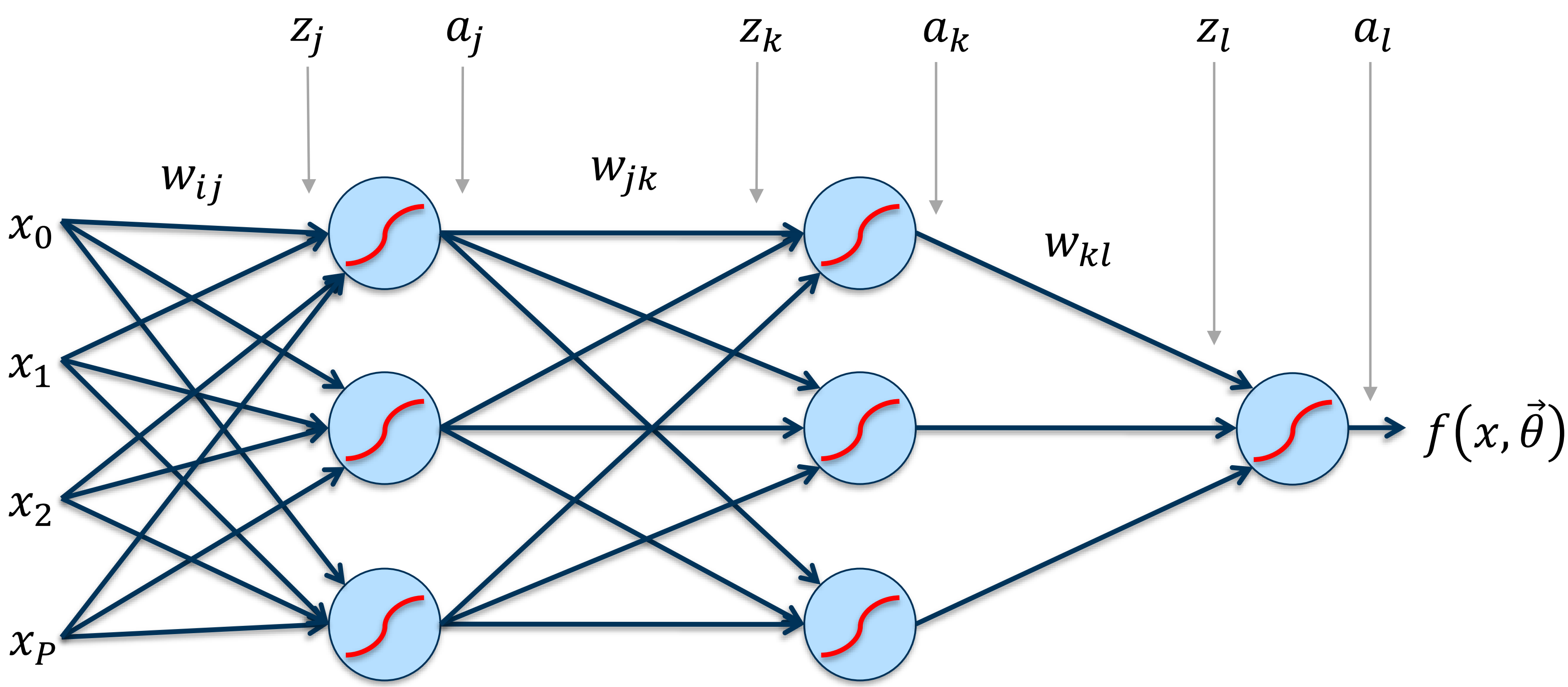
**Hidden Layers:**
should be chosen in a way that improves model performance (NOT straightforward)

**Output Layer:**
corresponds to modeling needs, i.e. 1 for binary classification, k-1 for multiclass classification, etc. Activation of output layer for each observation becomes the input to the loss function that triggers backpropagation



# Tutorial Business Analytics and Machine Learning

## Recap – Neural Networks




Not shown: “biases” $b_j,\ b_k,\ b_l$

# Tutorial Business Analytics and Machine Learning

## Recap – Neural Networks: Fully Connected Layers

The $L^{\text{th}}$ layer

Output from
previous layer

Weight matrix
Dimension:
(#nodes in $L$) $\times$ (#nodes in $L-1$)

Bias vector
(sometimes replaced
with a $0^{\text{th}}$ row in
Weight matrix and a
dummy entry $a_0^{[L-1]}$)

$$z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$$

"logits" or
"linear output"
of the layer

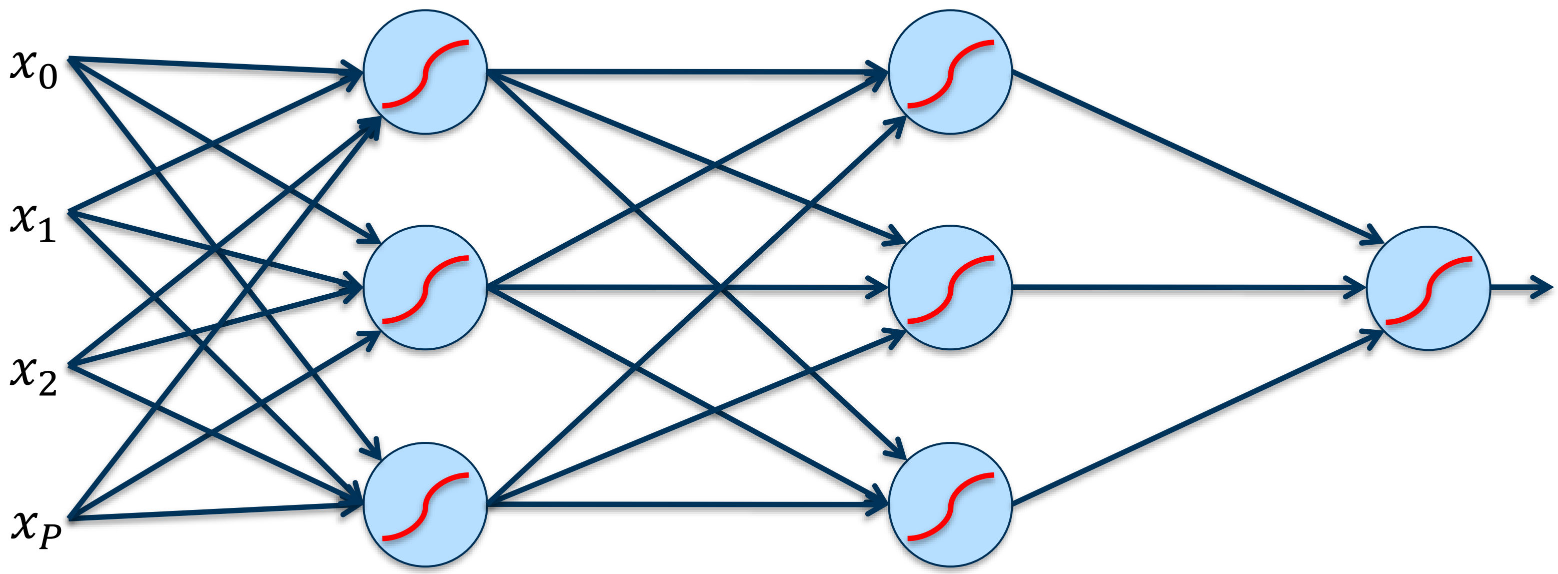
$$a^{[L]} = g^{[L]}(z^{[L]})$$

"Activation" or
output of the layer

Nonlinear, scalar "activation
function", applied element-
wise to each element of $z$

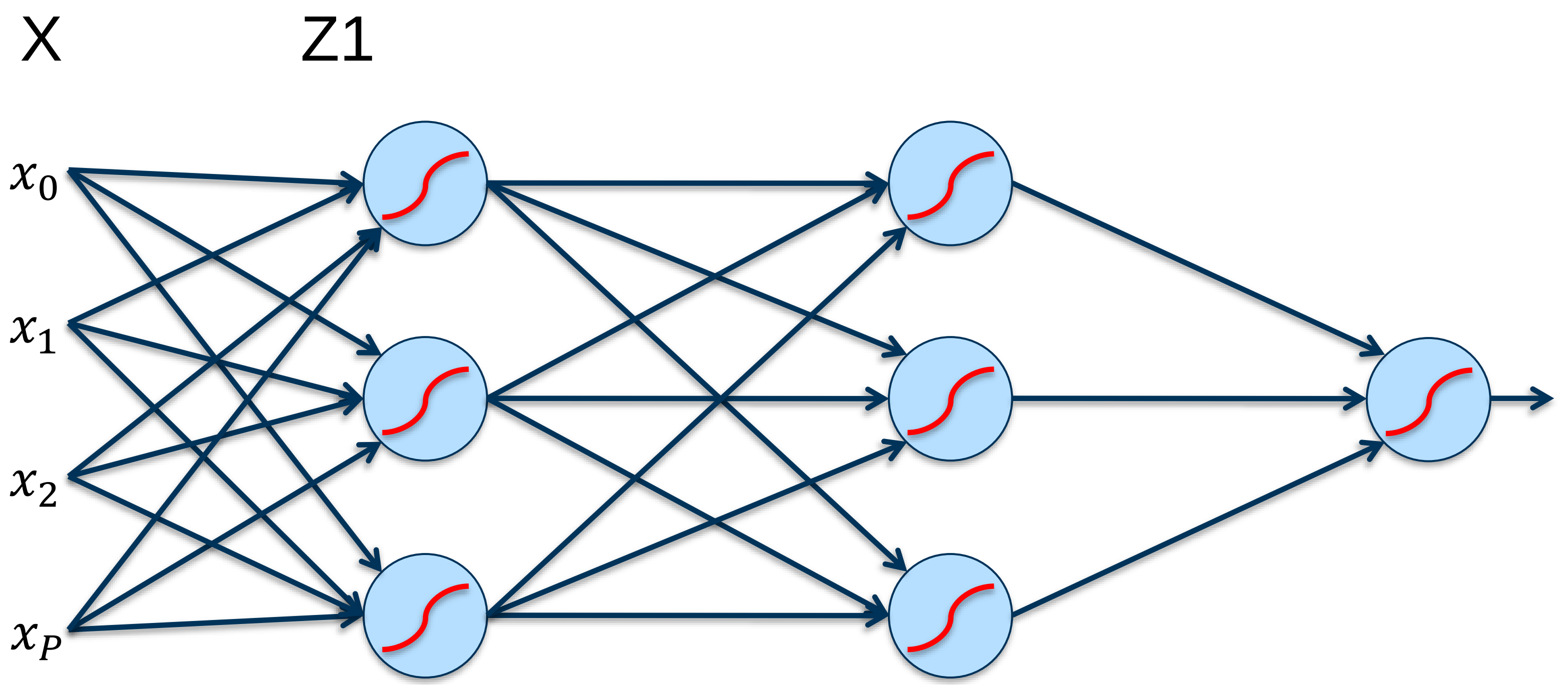
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



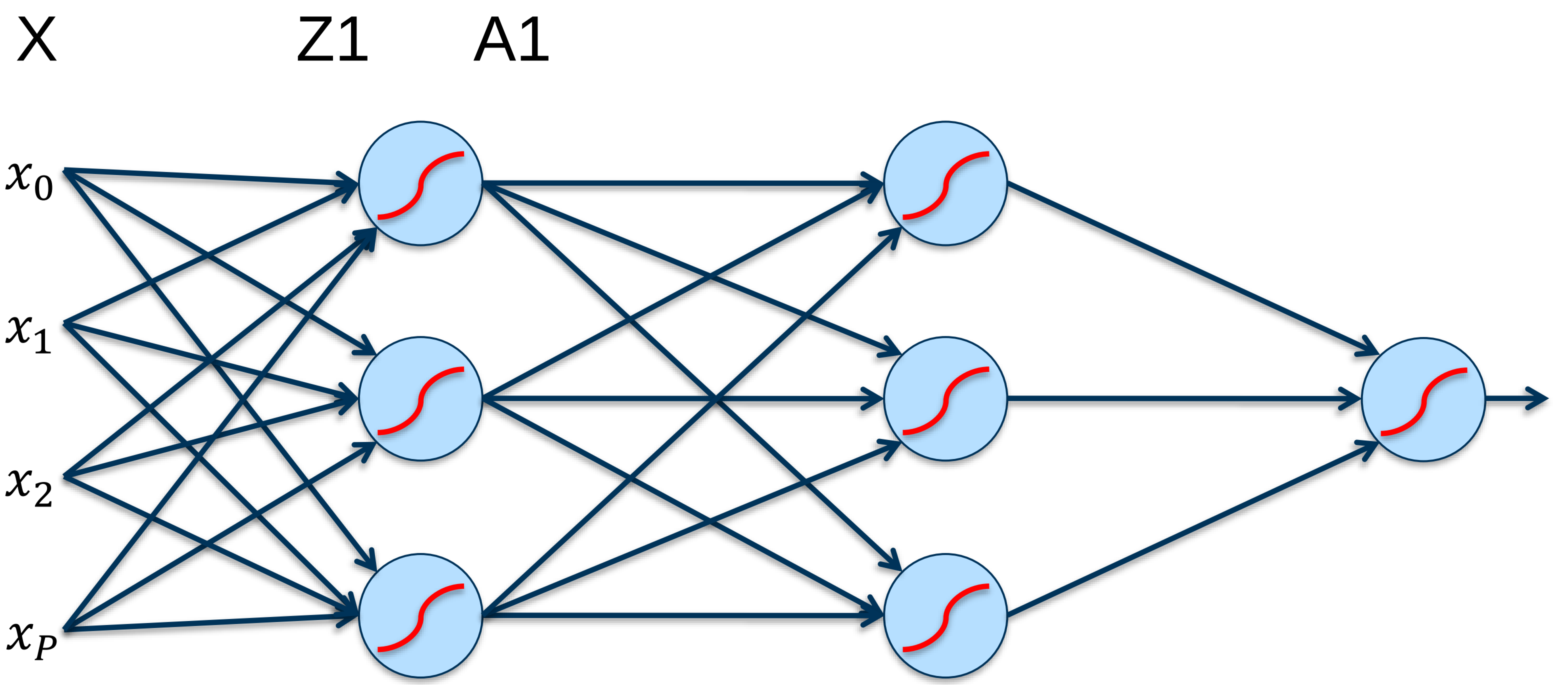
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



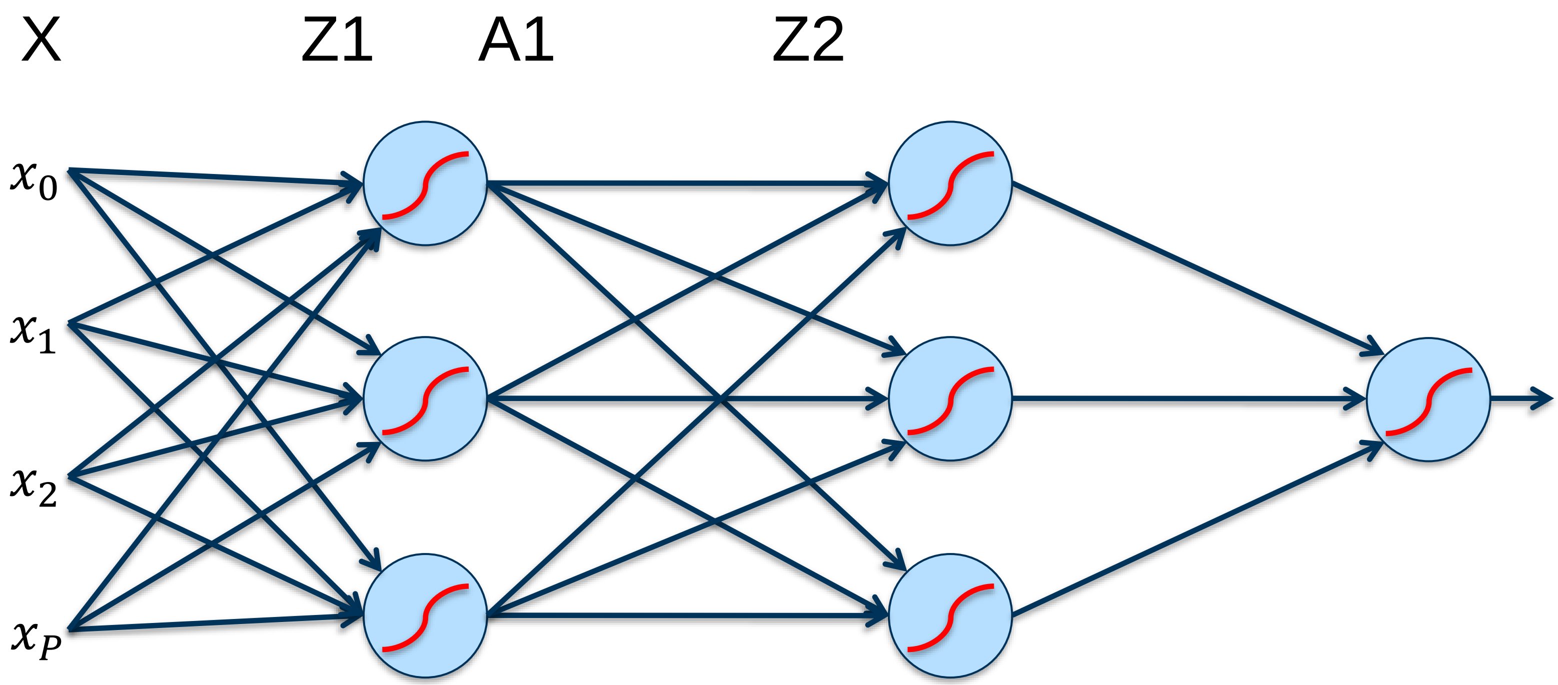
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



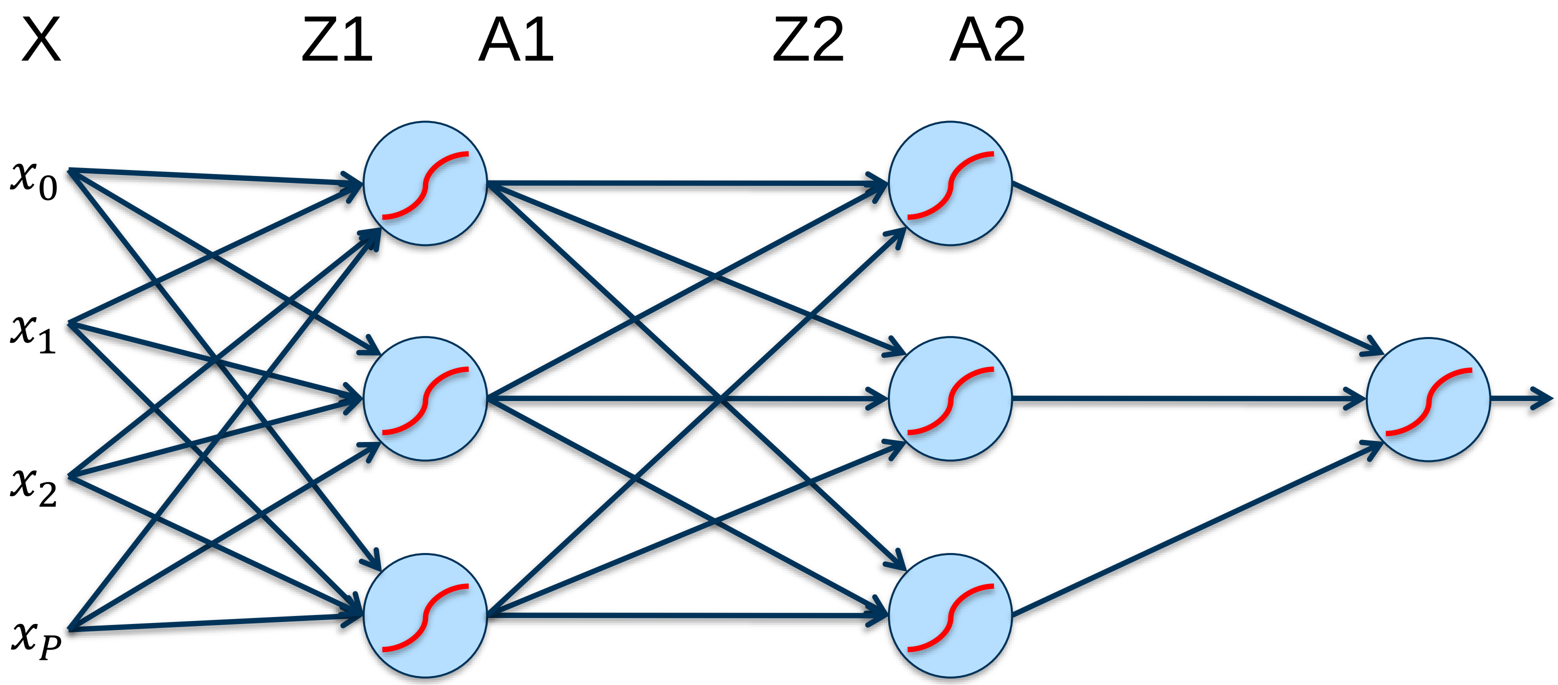
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



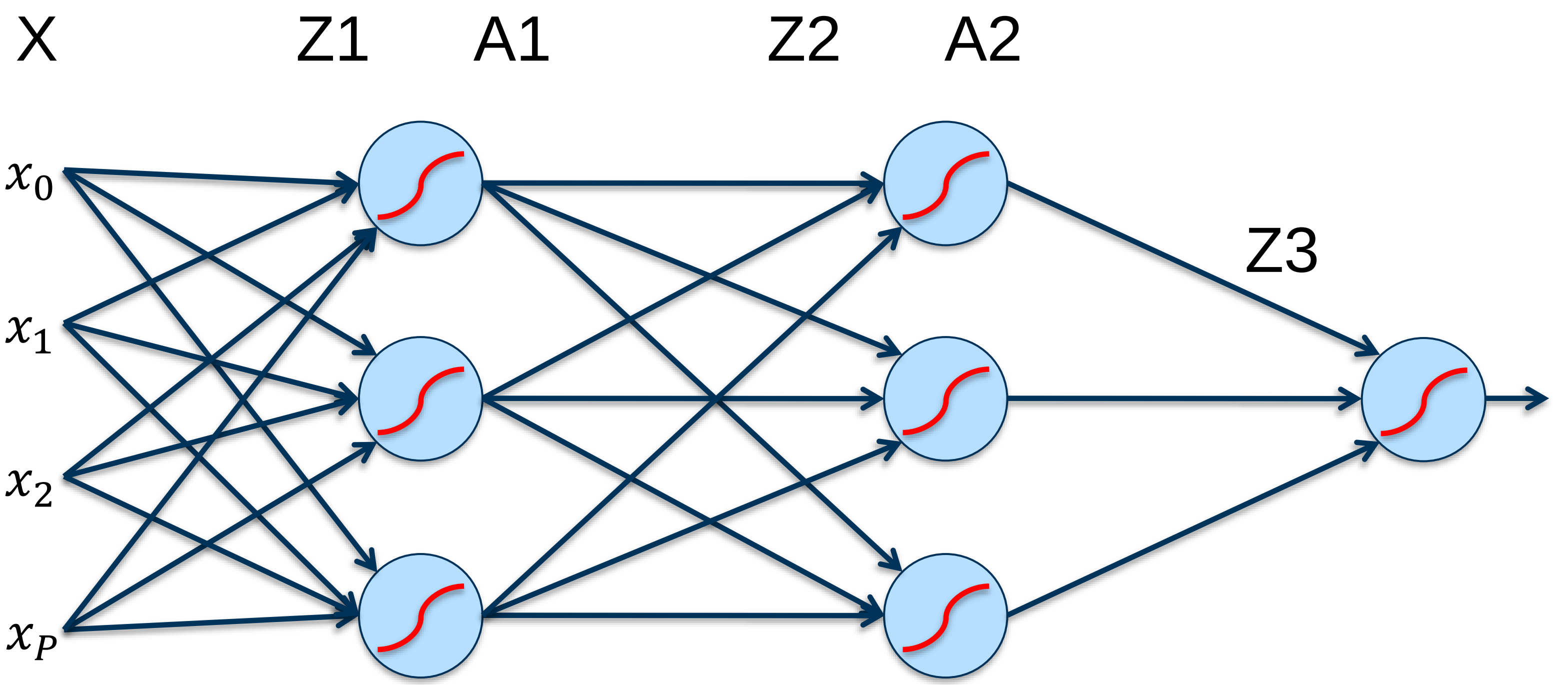
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



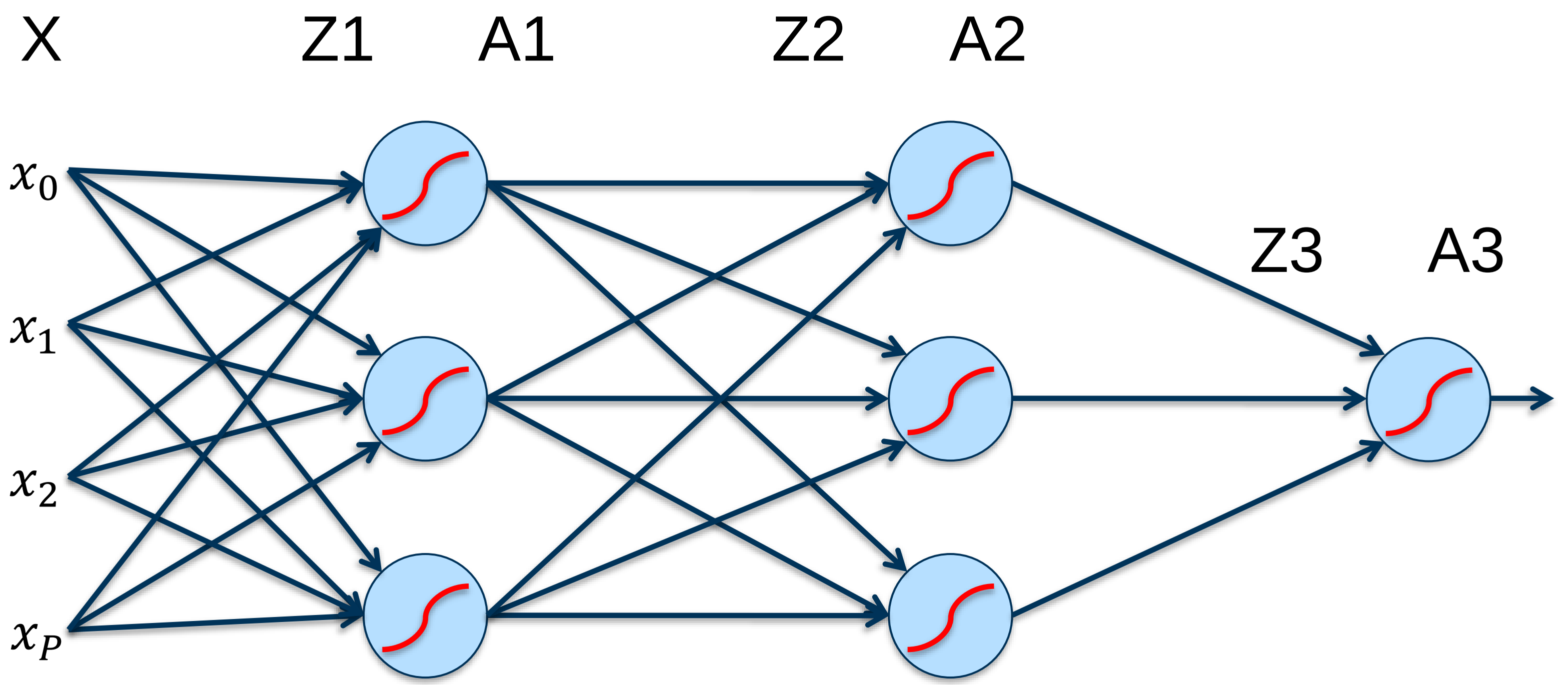
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



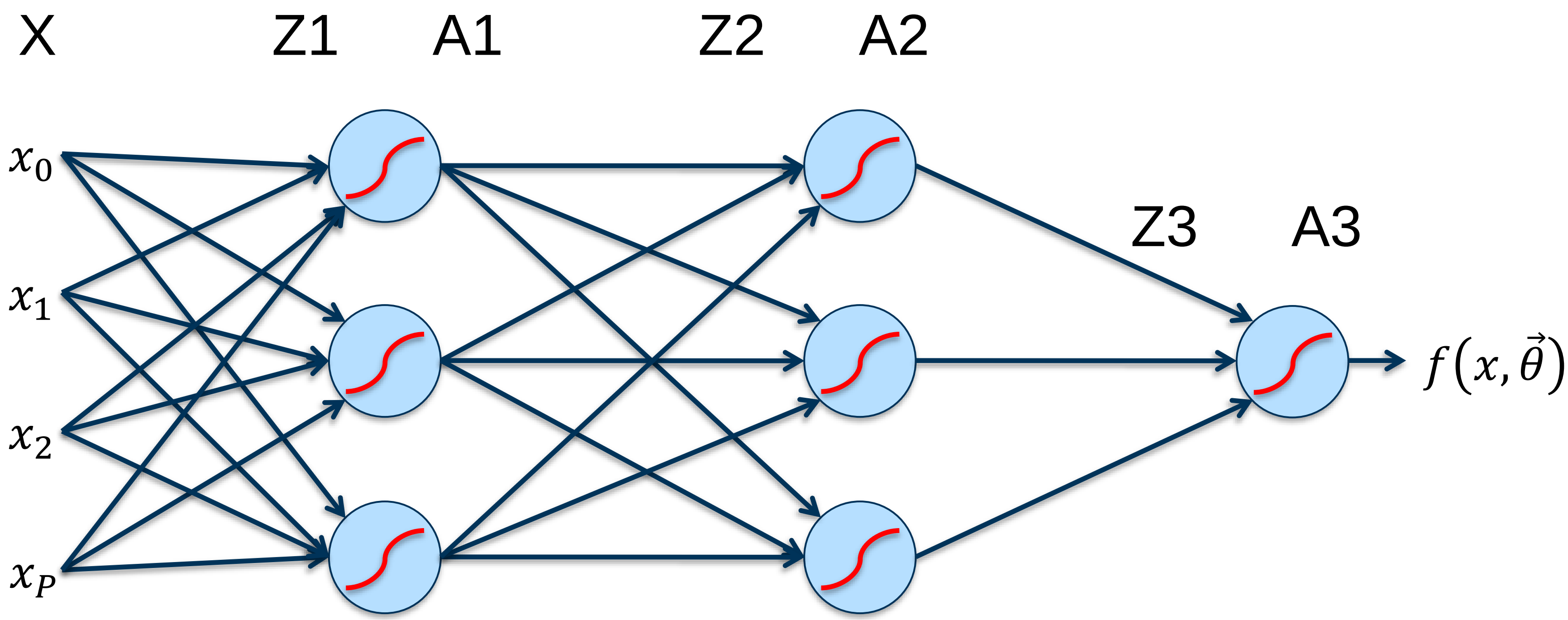
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



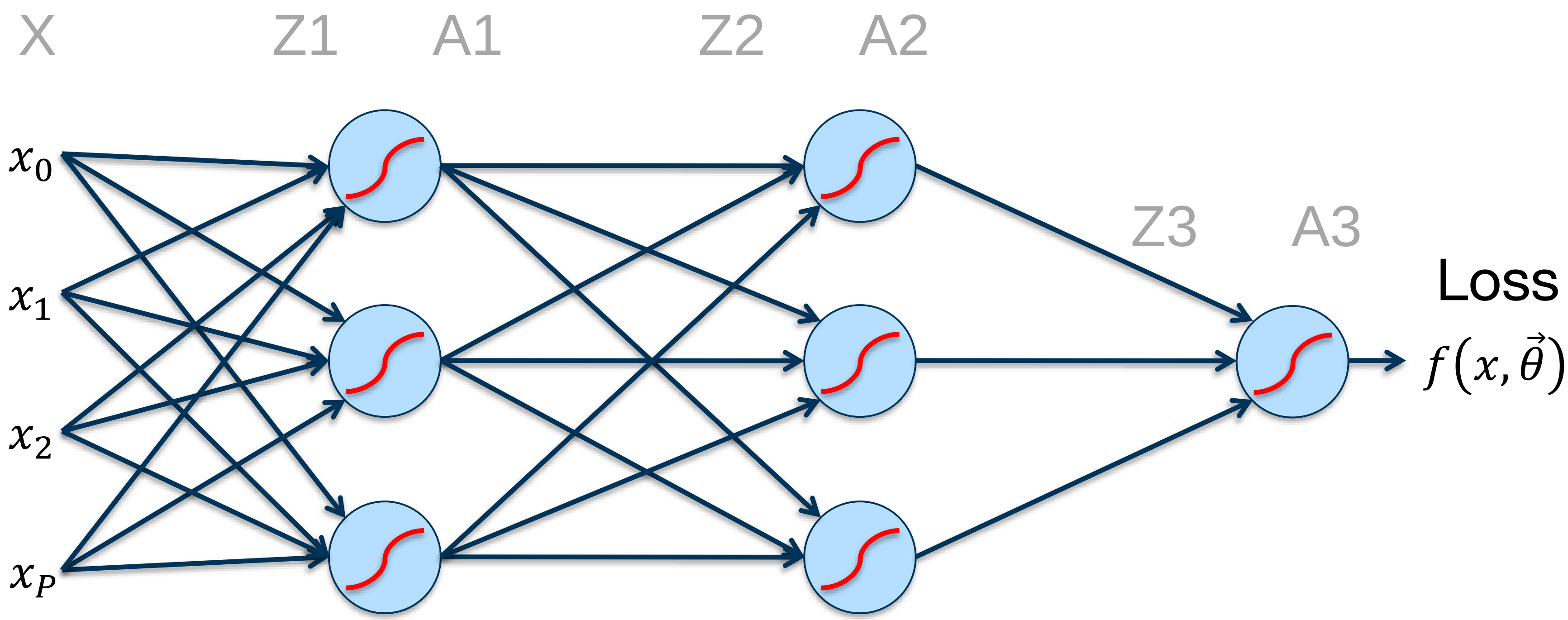
# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Forward Pass



# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Backward Pass: Apply Chain Rule

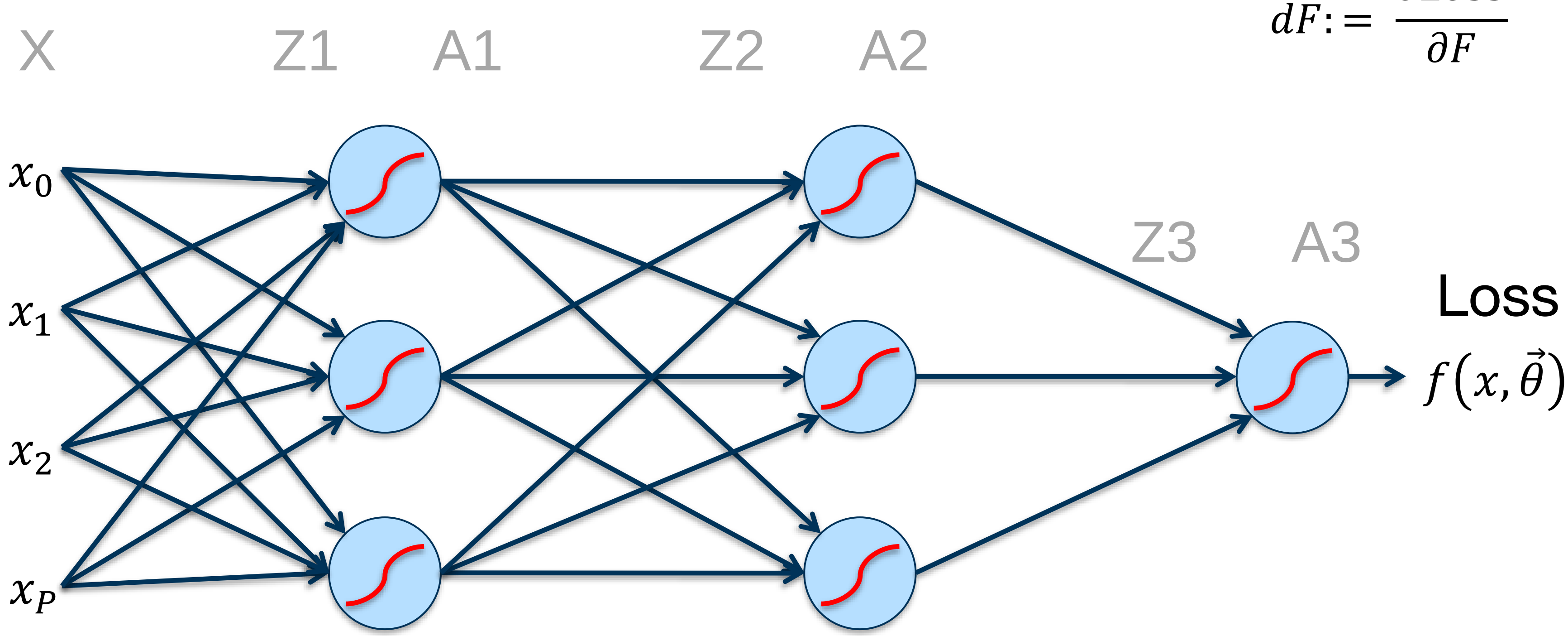


# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Backward Pass: Apply Chain Rule

Convention:

$$dF := \frac{\partial Loss}{\partial F}$$

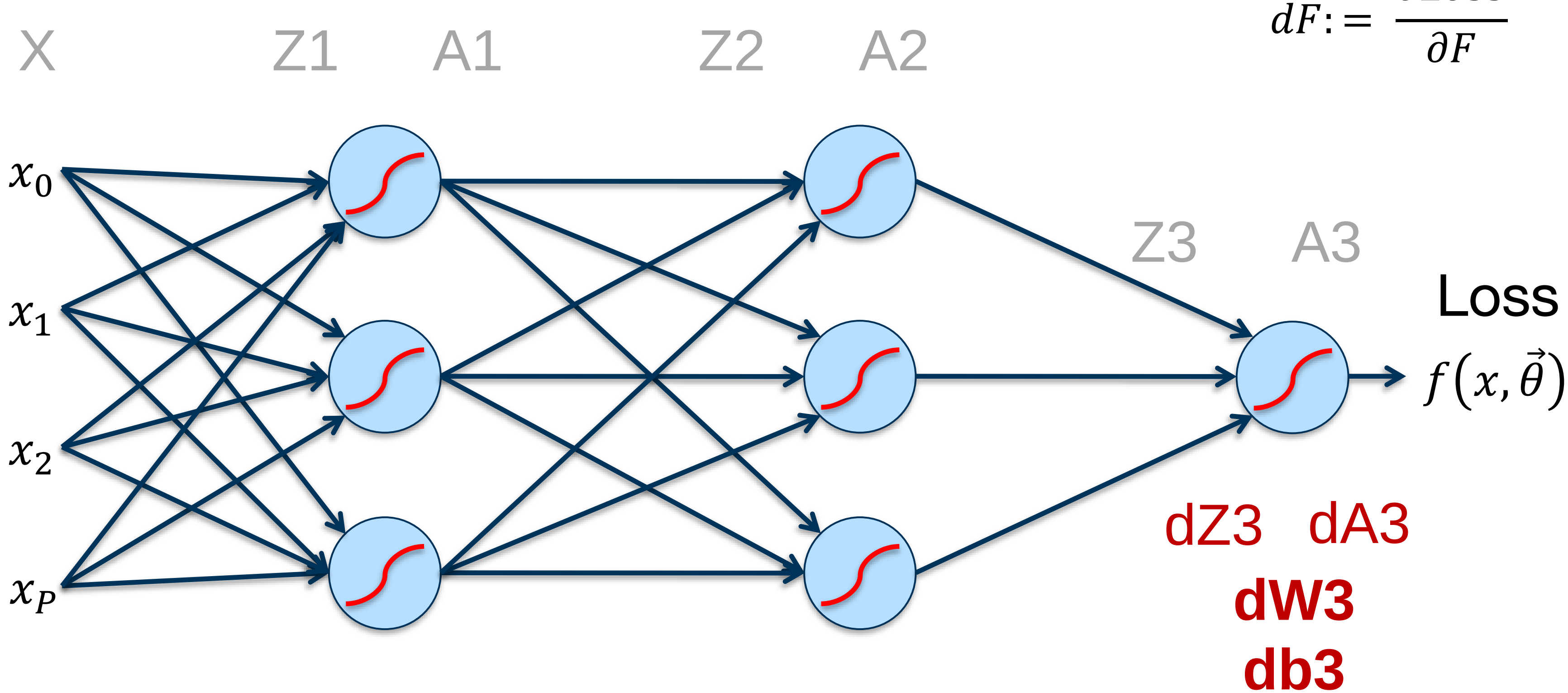


# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Backward Pass: Apply Chain Rule

Convention:

$$dF := \frac{\partial Loss}{\partial F}$$



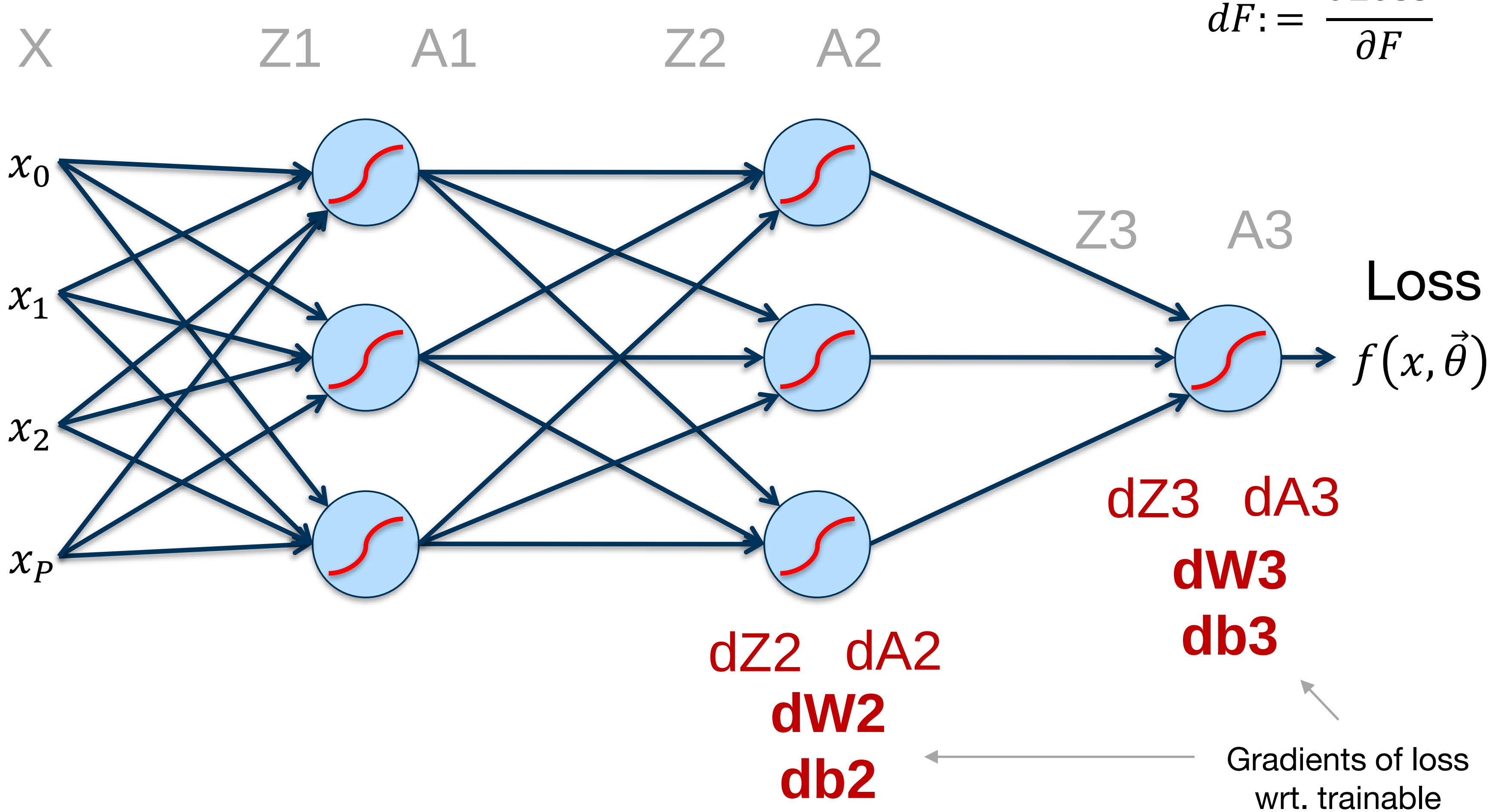

Gradients of loss wrt. trainable parameters

# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Backward Pass: Apply Chain Rule

Convention:

$$dF := \frac{\partial Loss}{\partial F}$$



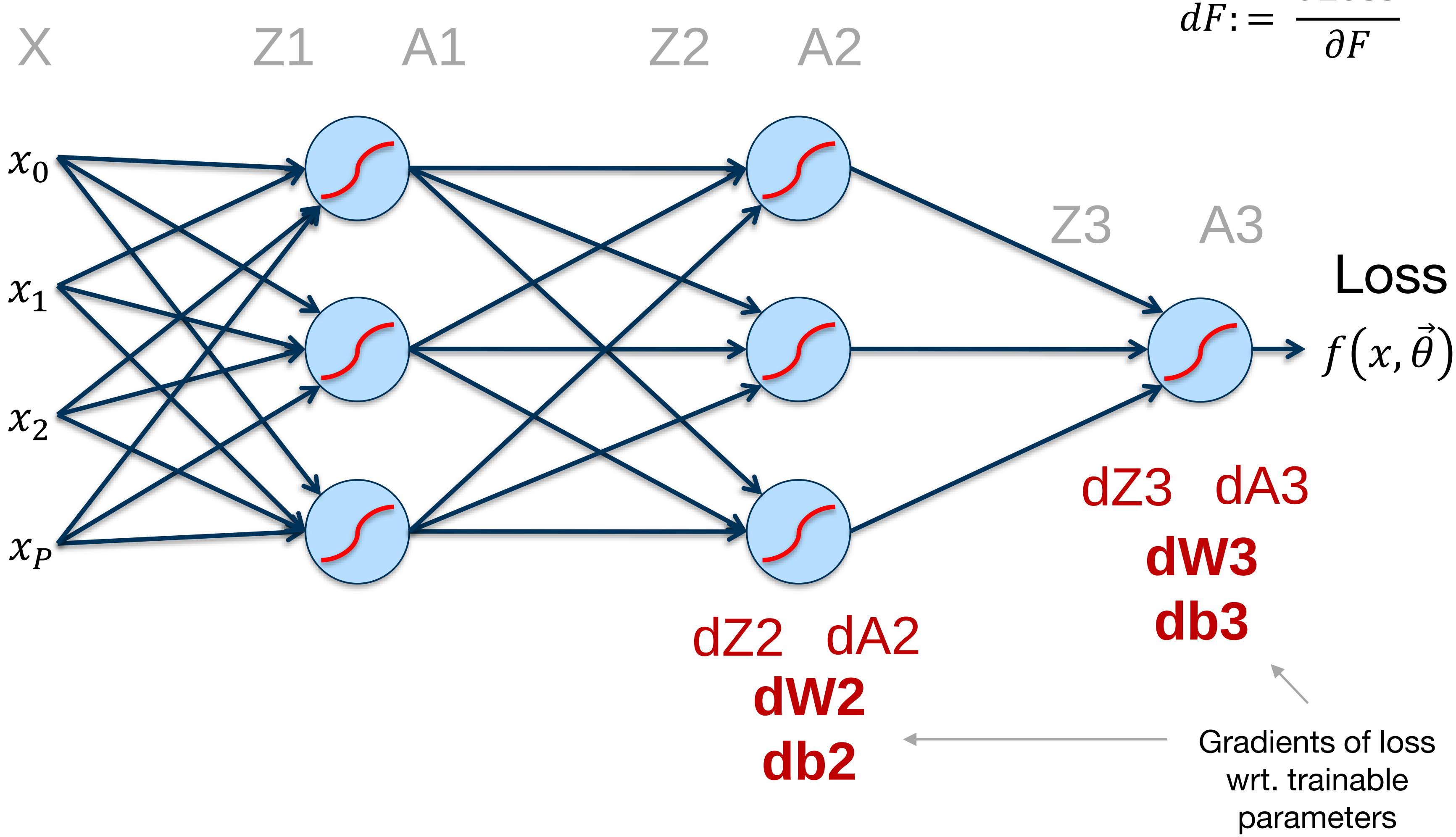

Gradients of loss wrt. trainable parameters

# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Backward Pass: Apply Chain Rule

Convention:

$$dF := \frac{\partial Loss}{\partial F}$$

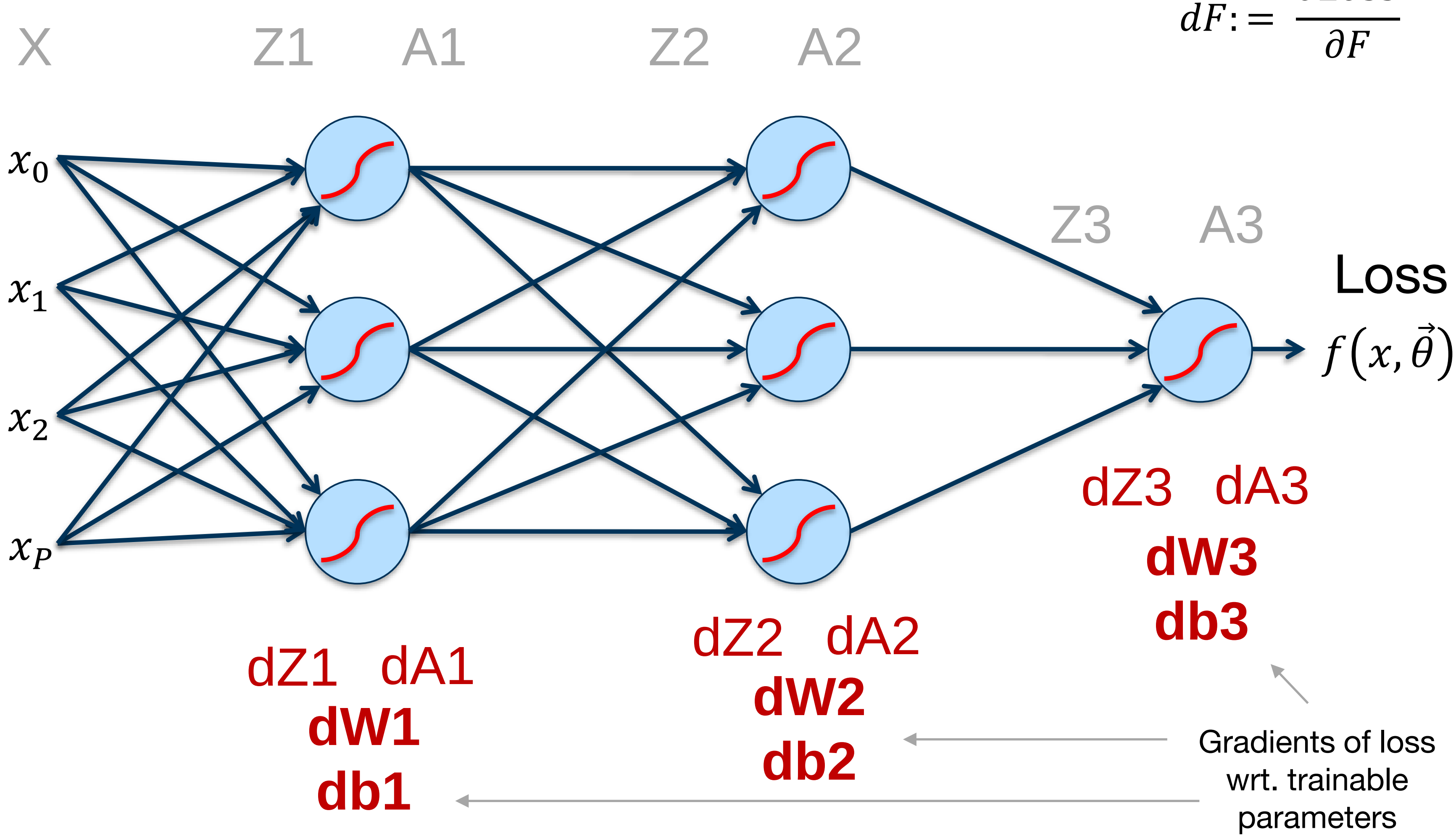


# Tutorial Business Analytics and Machine Learning

## Training and Inference in NNs – Backward Pass: Apply Chain Rule

Convention:

$$dF := \frac{\partial Loss}{\partial F}$$



# Tutorial Business Analytics

## Updating Parameters using a Gradient Step

$$W^{[L]} := W^{[L]} - \alpha \cdot dW^{[L]}$$
$$b^{[L]} := b^{[L]} - \alpha \cdot \, db^{[L]}$$

Then repeat, starting with forward pass.

# Neural Net Exercises

- 12.2: Forward Pass
- 12.3: Backpropagation
- 12.4: Programming Exercise / Practical Example