UNIVERSIDAD DE CHILE

Minería de Datos

Welcome to the Machine Learning class

Valentin Barriere

Universidad de Chile – DCC

CC5205, Fall 2025

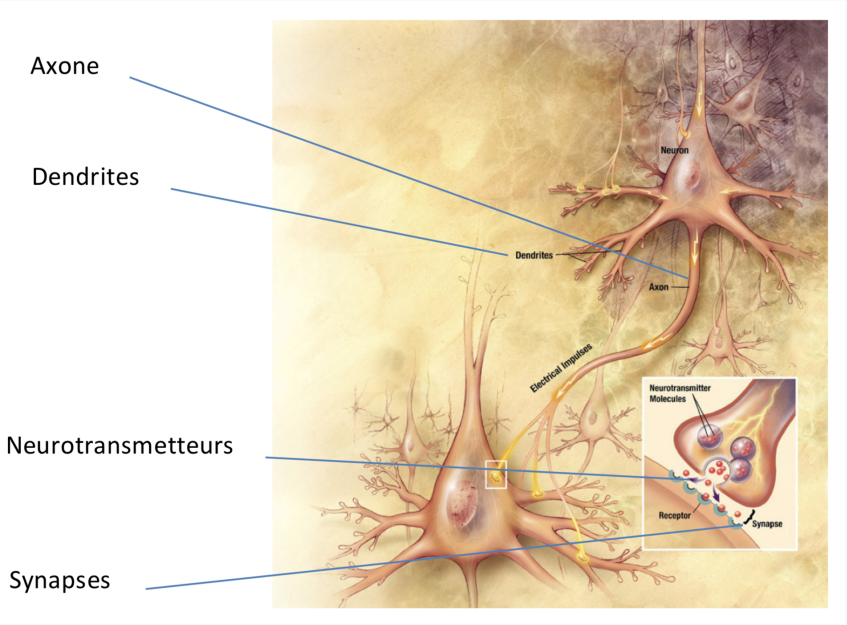
Introduction to Deep Learning

Perceptron

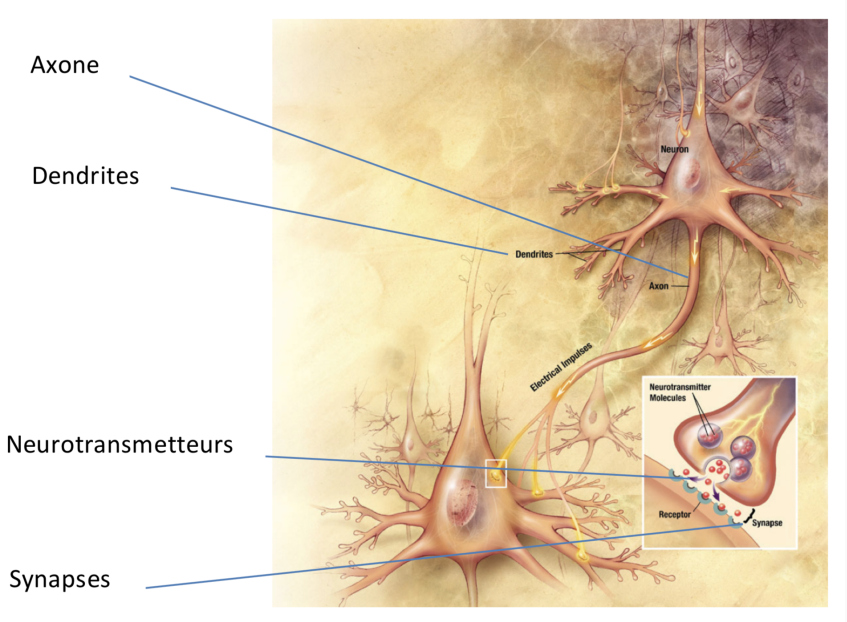
Outline : Perceptron

First, why this name?

A neuron receives multiple signals (neurotransmitters) at its **dendrites**. These neurotransmitters are released at the **synapses**. When the total input exceeds a certain **threshold**, the neuron "fires," sending an electrical impulse along its axon, which in turn allows it to release neurotransmitters through its own synapses.



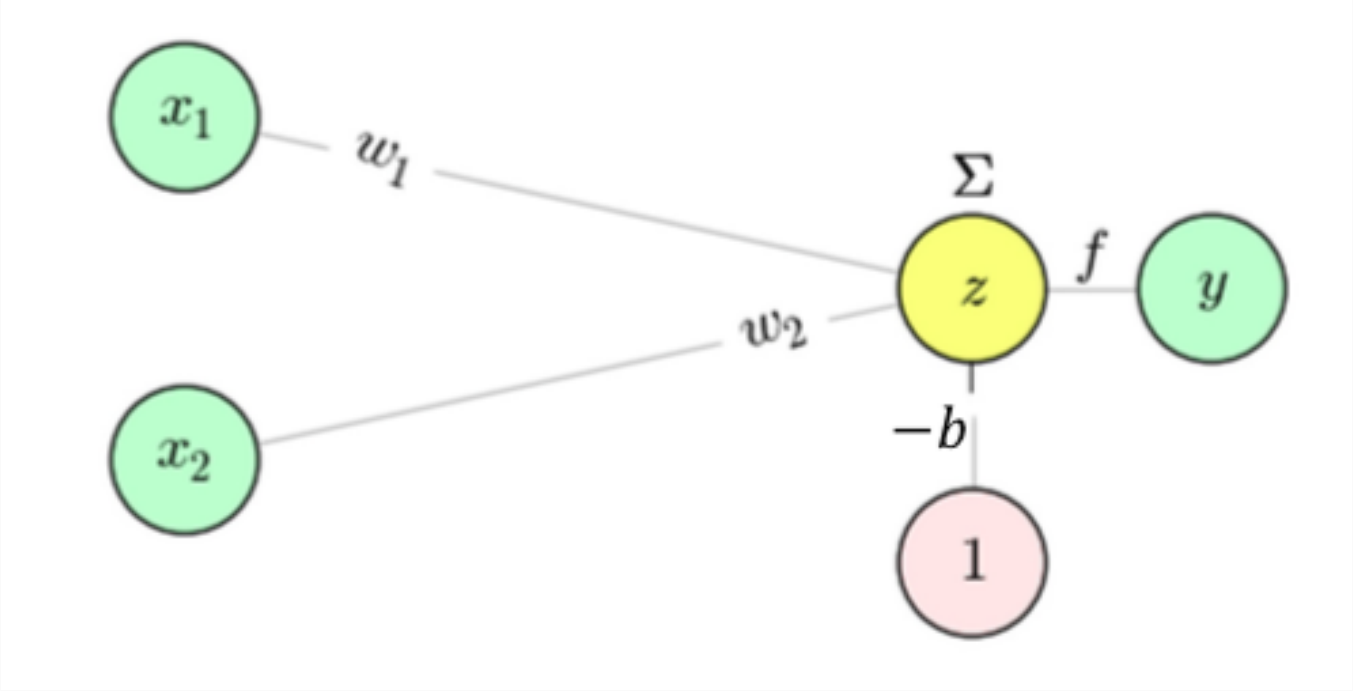
First, why this name?



To summarize:

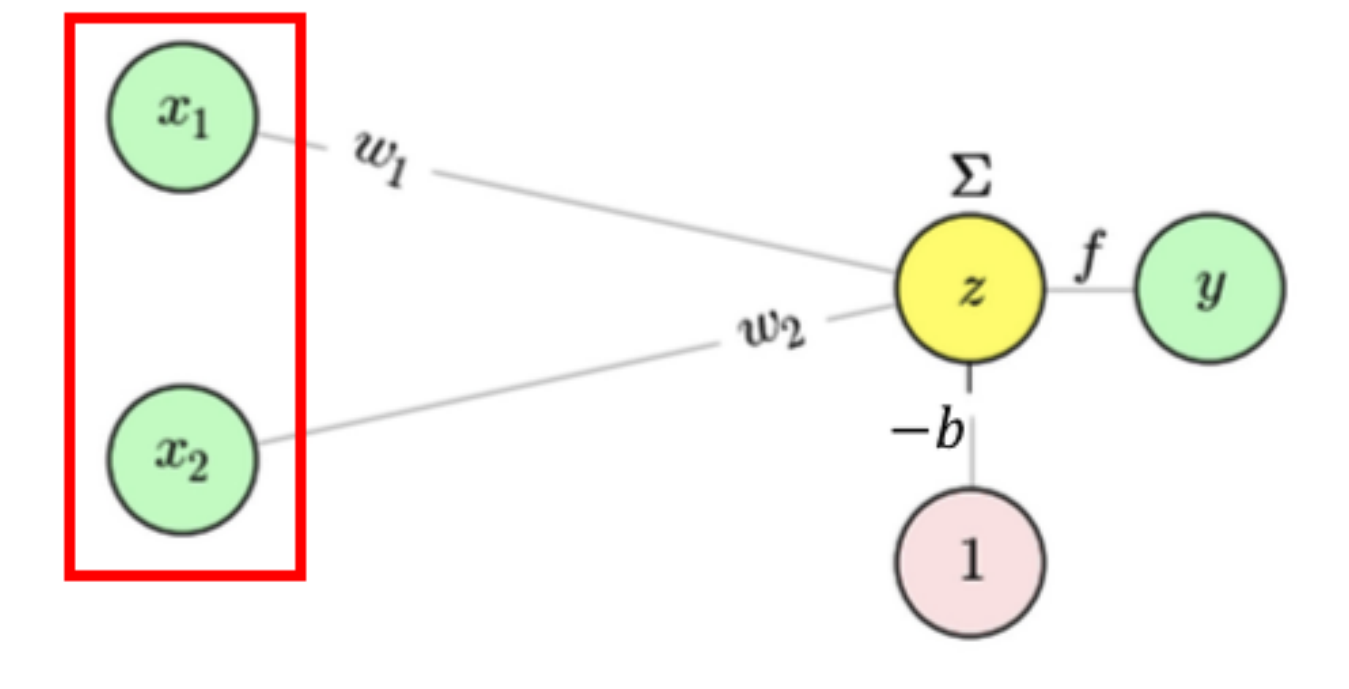
- The outputs of one neuron become the inputs of another.
- A neuron fires when it receives input exceeding a threshold.
- The strength of the signal sent to the next neuron is governed by synapses, which activate once their input crosses a threshold.

Perceptron – Overview



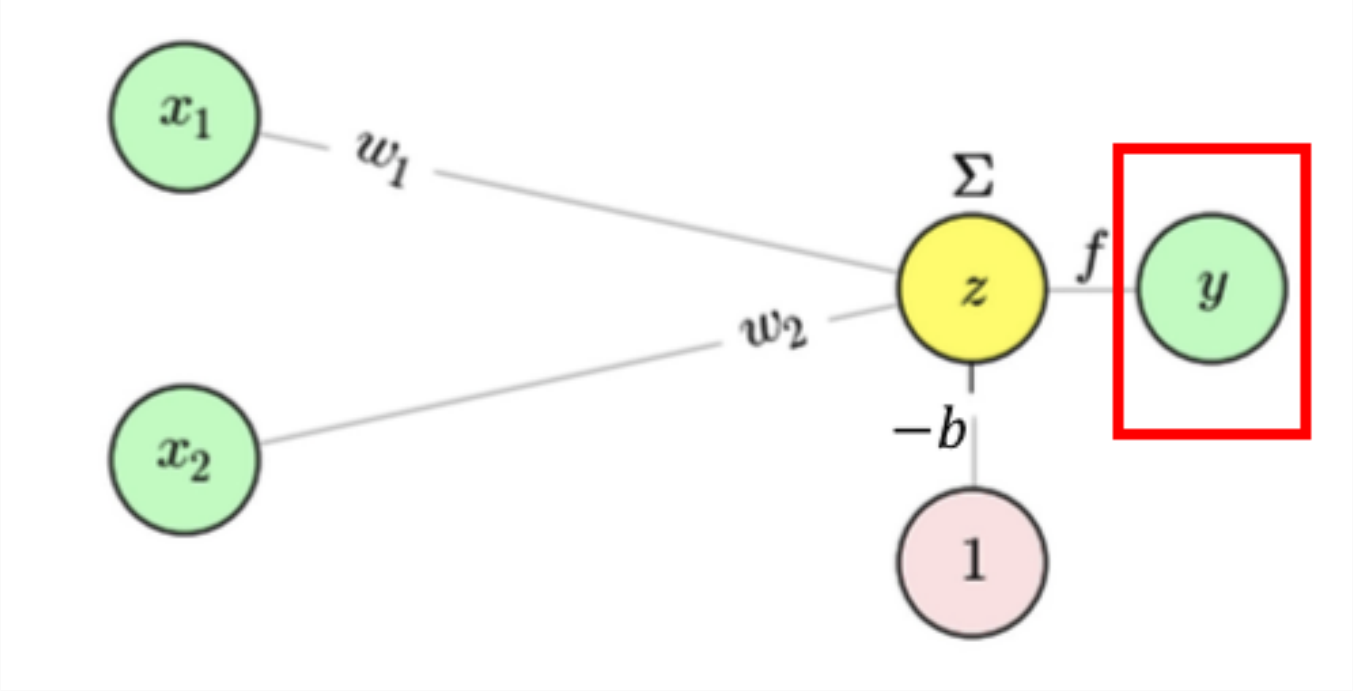
$$y = f(x_1 w_1 + x_2 w_2 - b)$$

Perceptron – Overview



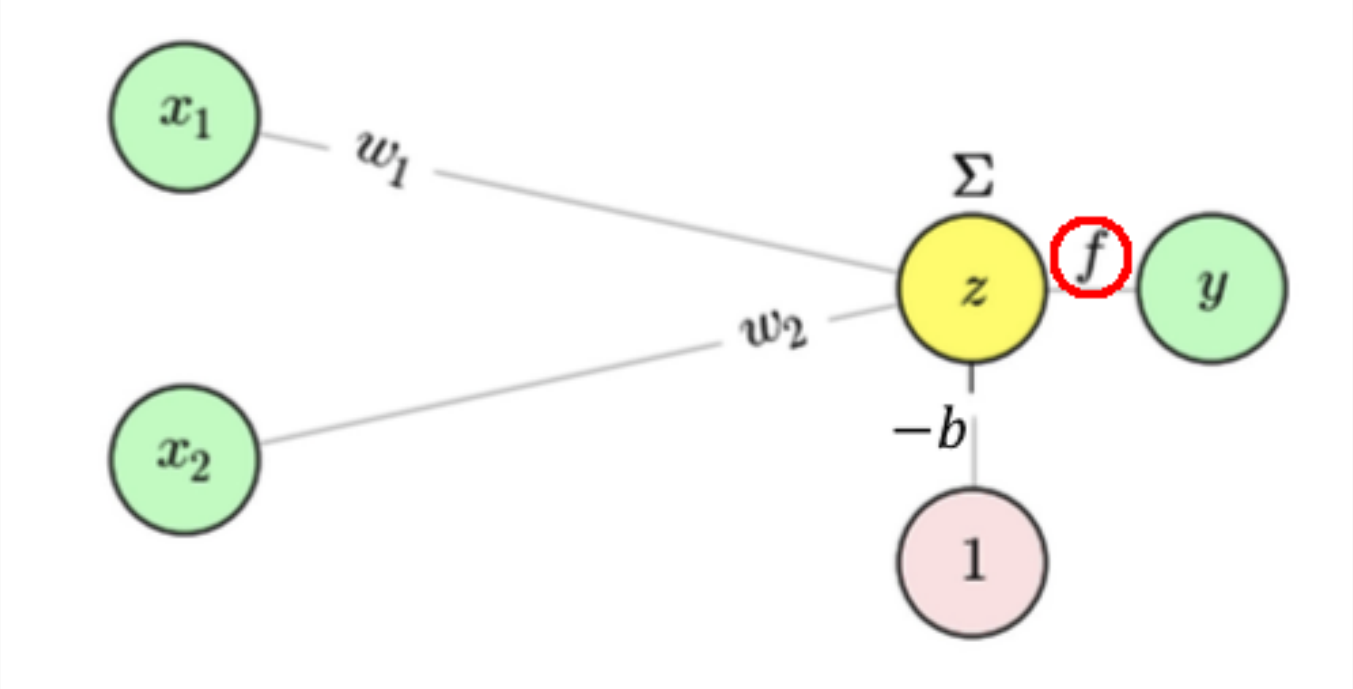
$$y = f(x_1 w_1 + x_2 w_2 - b)$$

Perceptron – Overview



$$y = f(x_1 w_1 + x_2 w_2 - b)$$

Perceptron – Overview



$$y = f(x_1 w_1 + x_2 w_2 - b)$$

In theory, f is the Heaviside step function:

$$f(z) = \mathbb{1}_{\mathbb{R}_+}(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{if } z < 0. \end{cases}$$

Multilayer Perceptron

Outline : Multilayer Perceptron

MLP – Overview

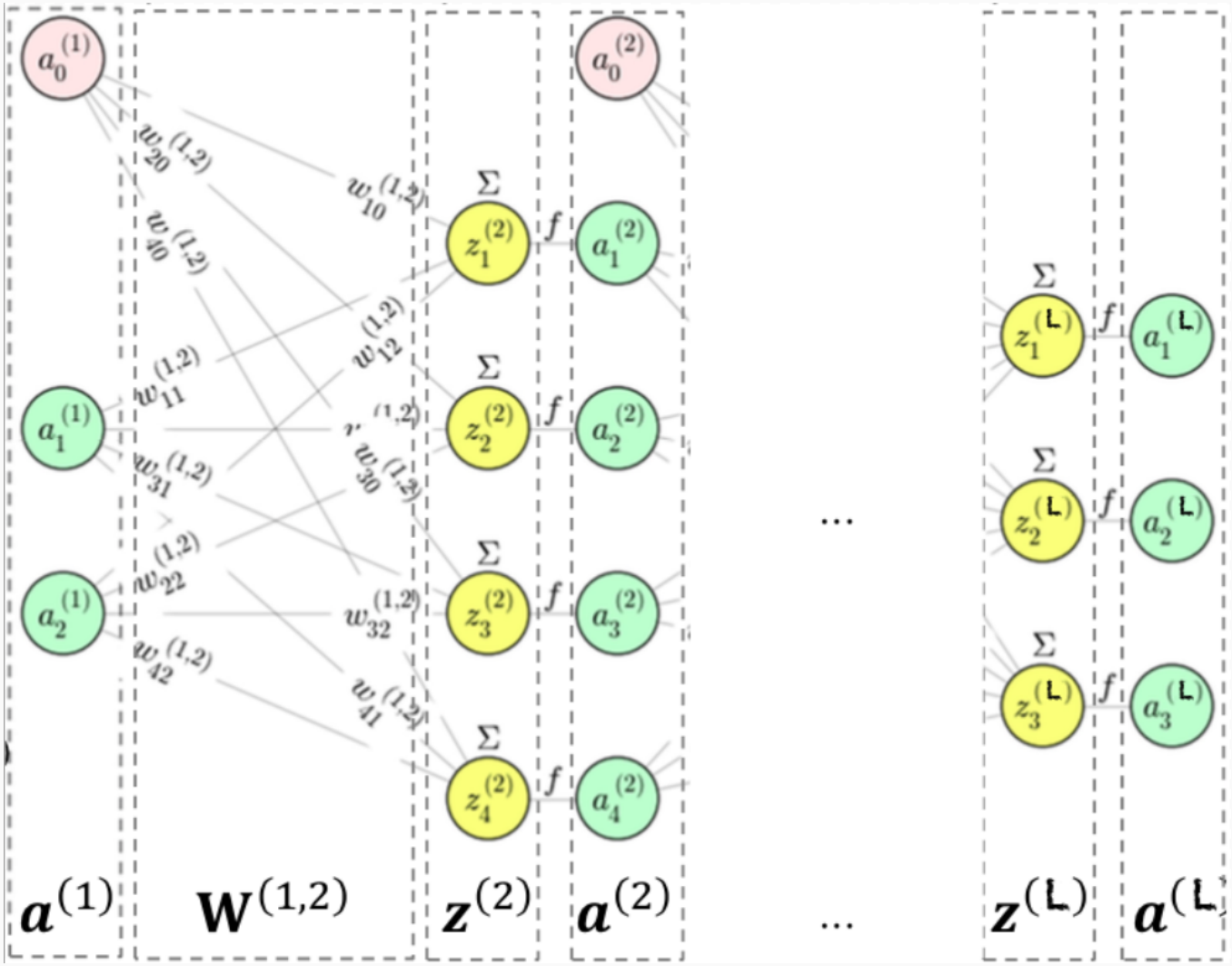


Figure 1: Forward propagation in an MLP

MLP – Overview

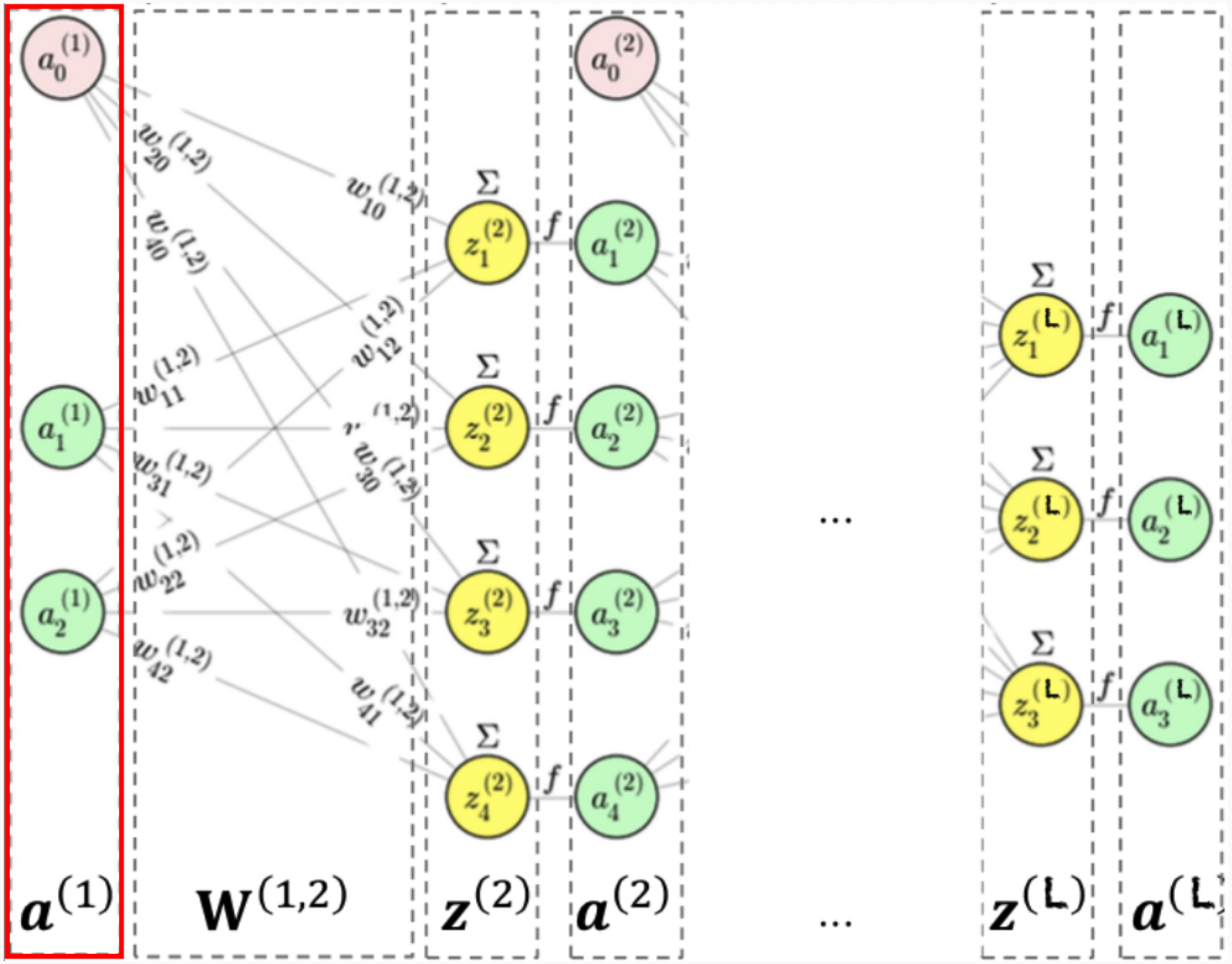


Figure 1: Forward propagation in an MLP: first layer

MLP – Overview

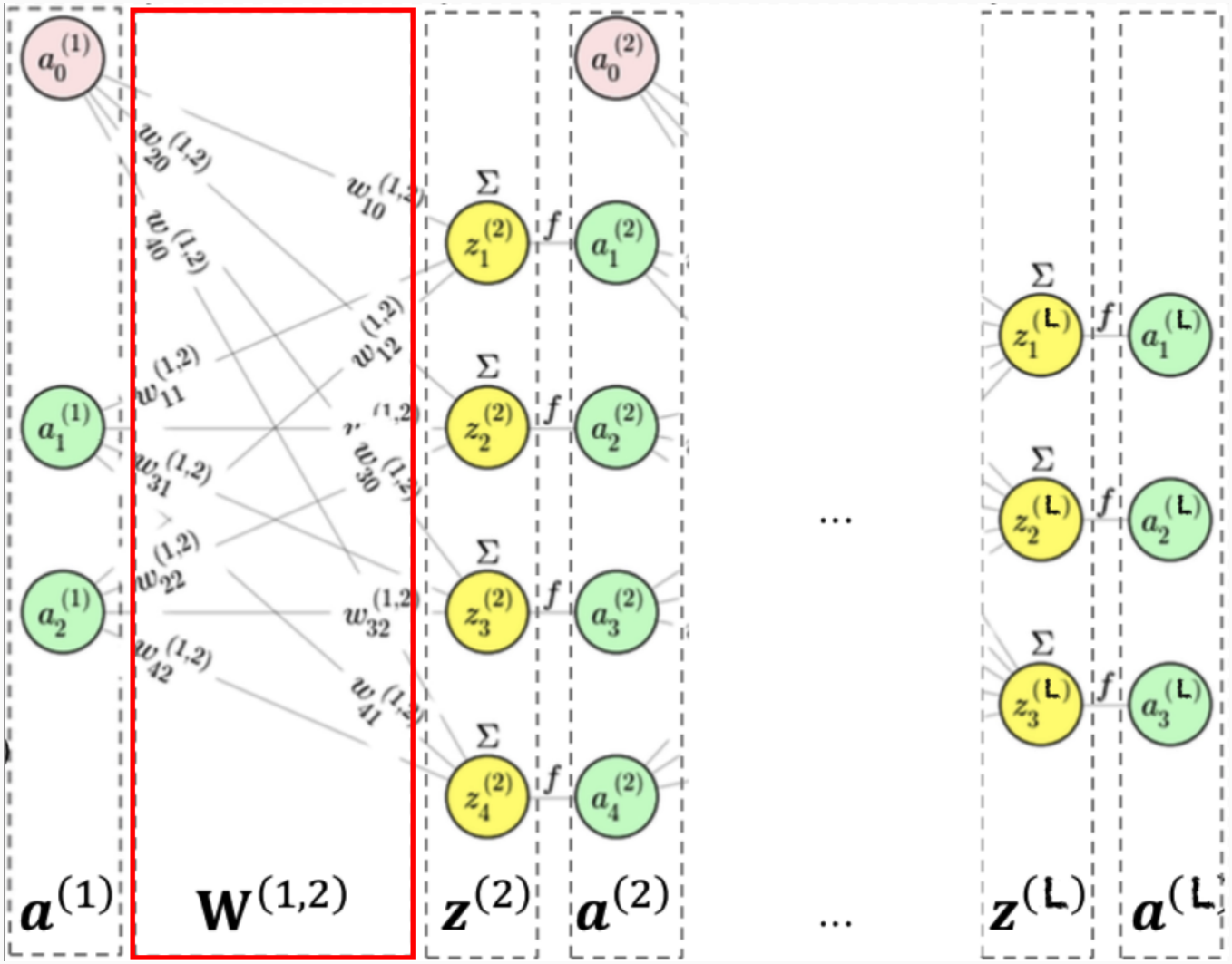


Figure 1: Forward propagation in an MLP: weight matrix

MLP – Overview

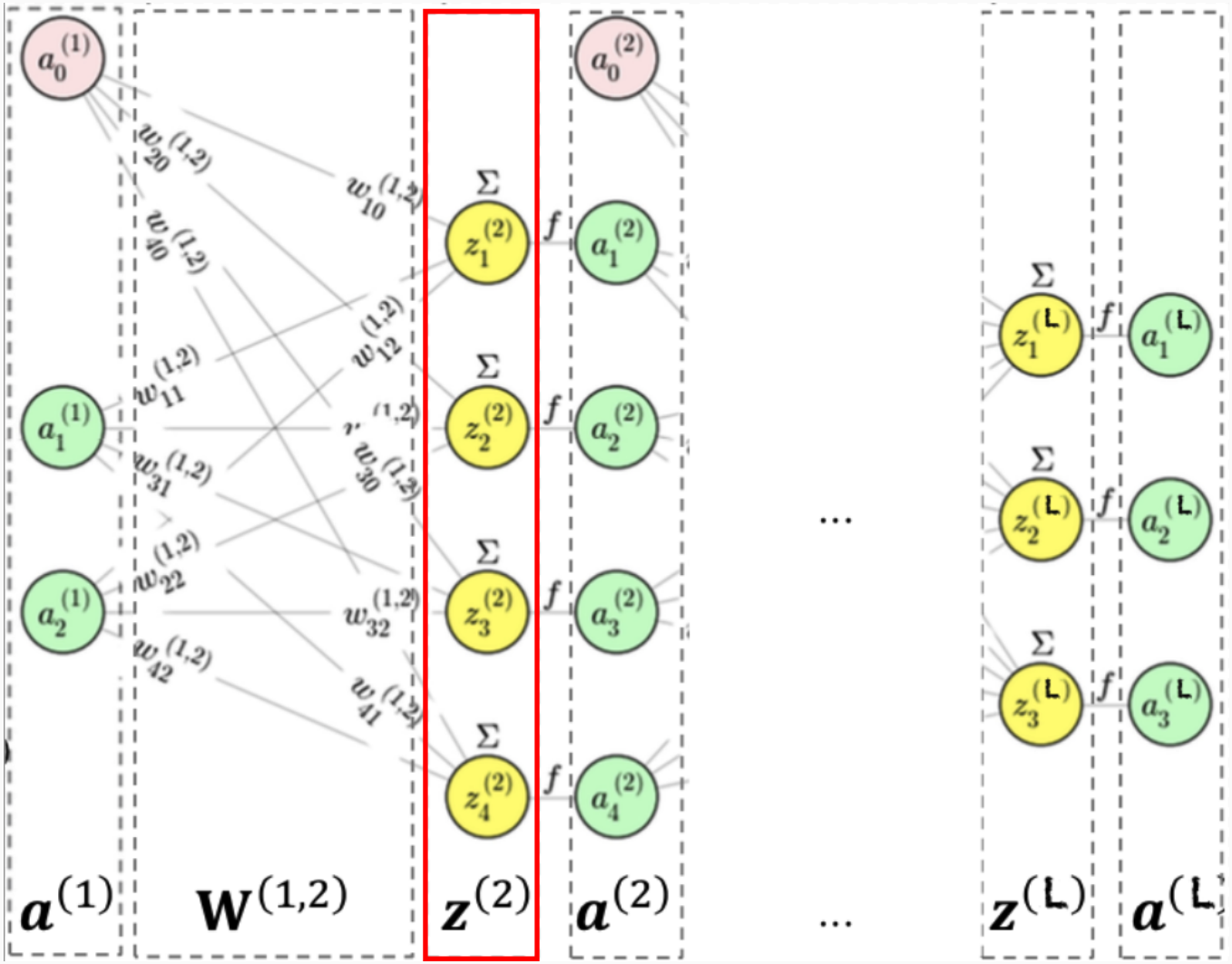


Figure 1: Forward propagation in an MLP: input to second layer

MLP – Overview

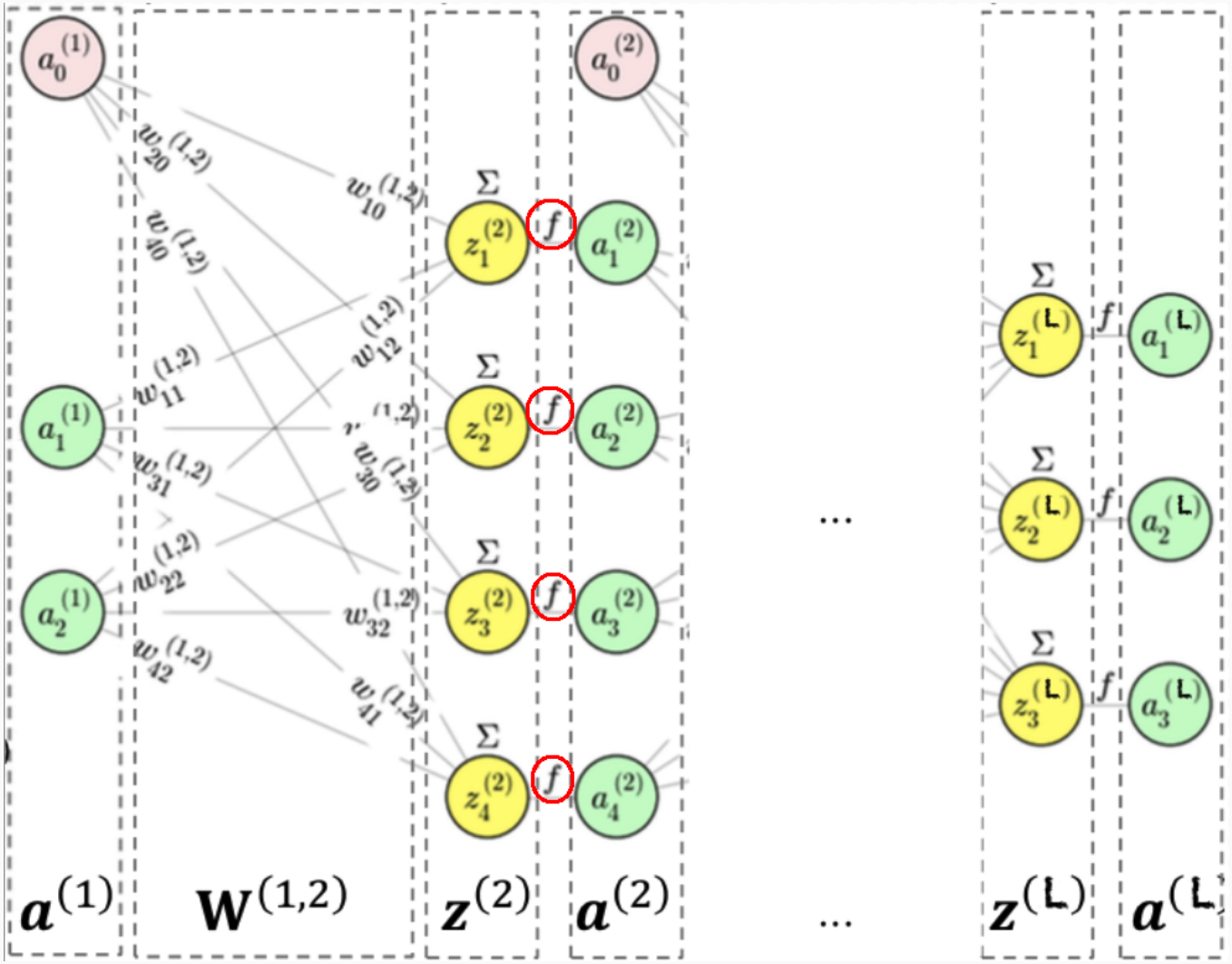


Figure 1: Forward propagation in an MLP: activation

MLP – Overview

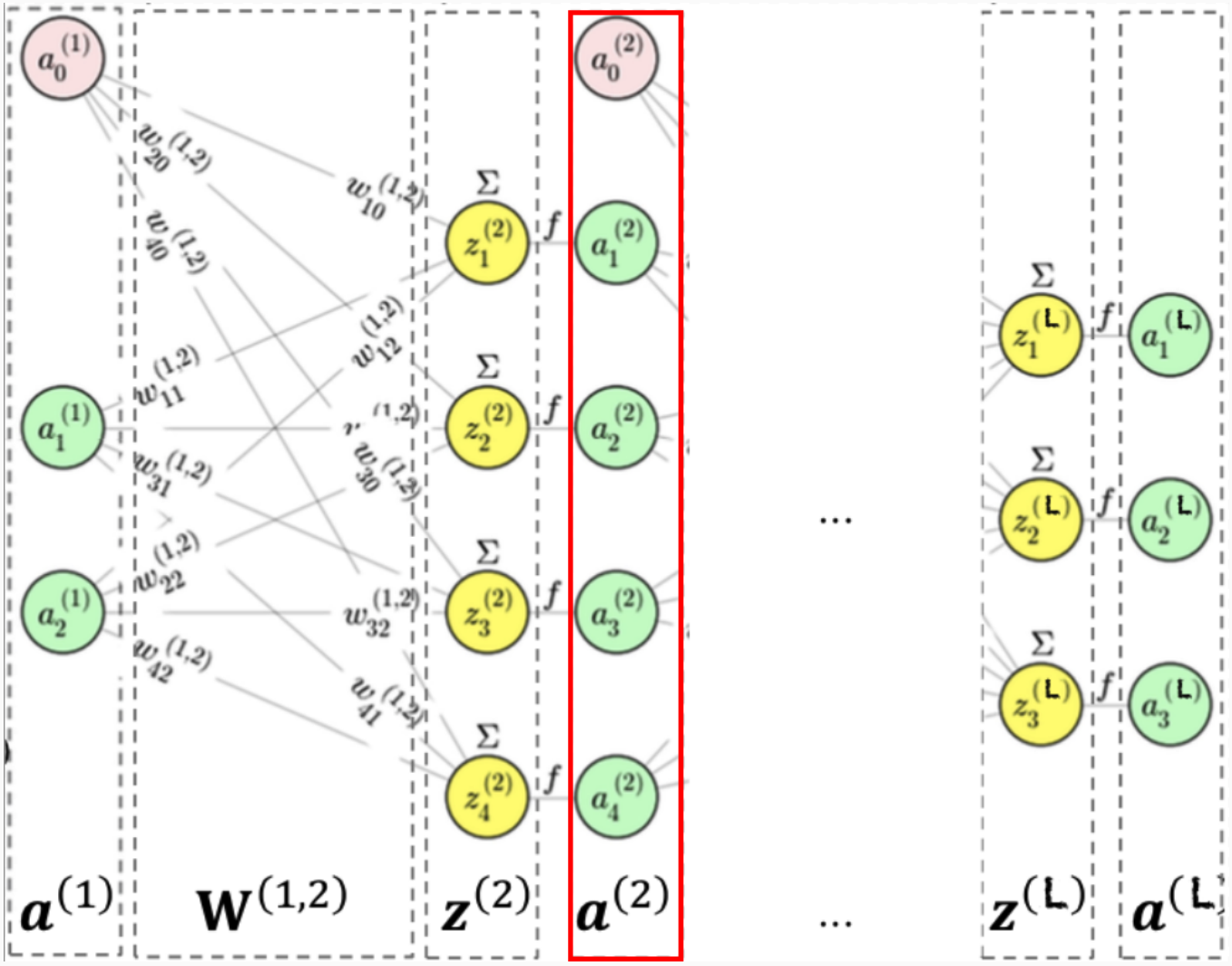


Figure 1: Forward propagation in an MLP: output of second layer

MLP – Overview

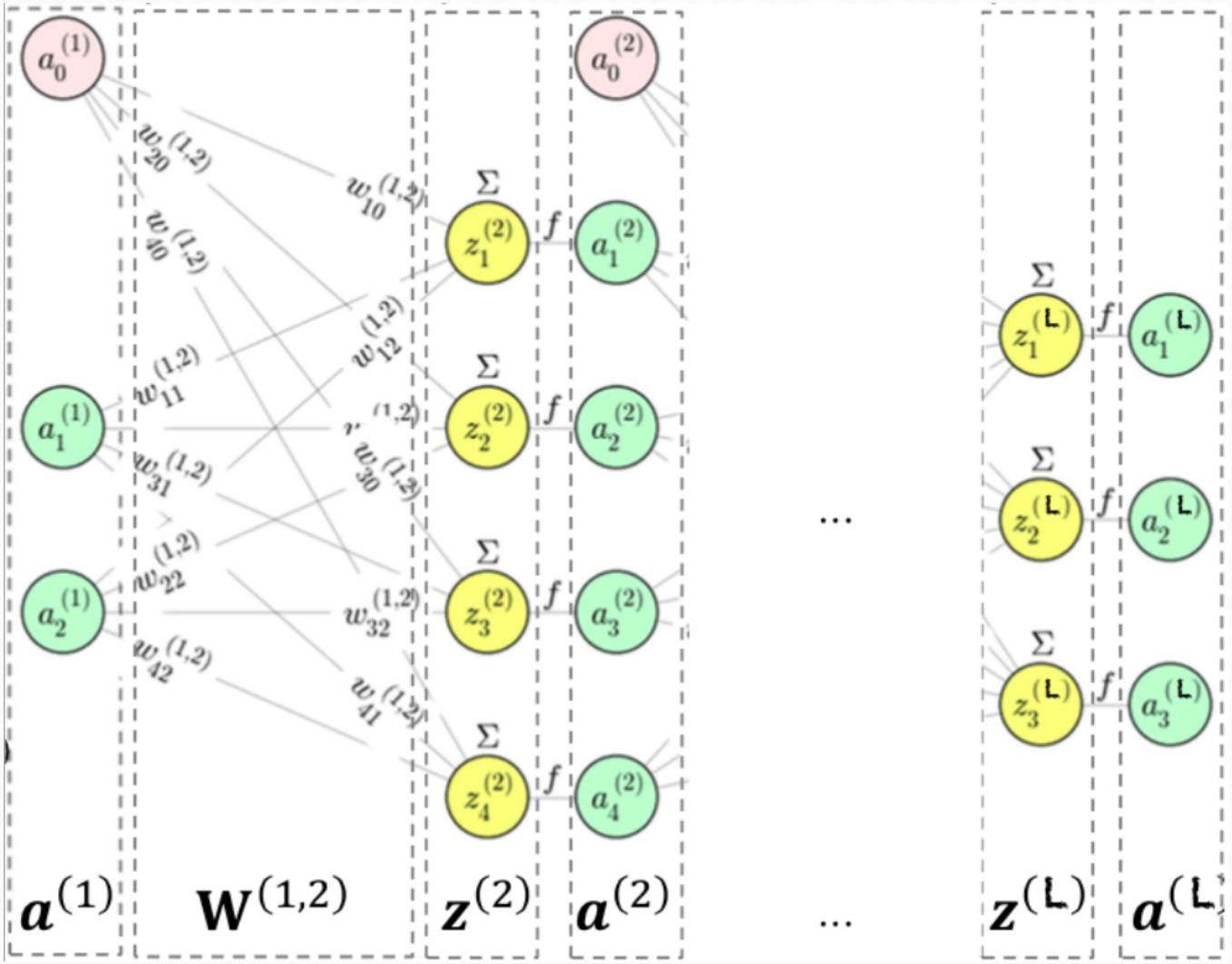
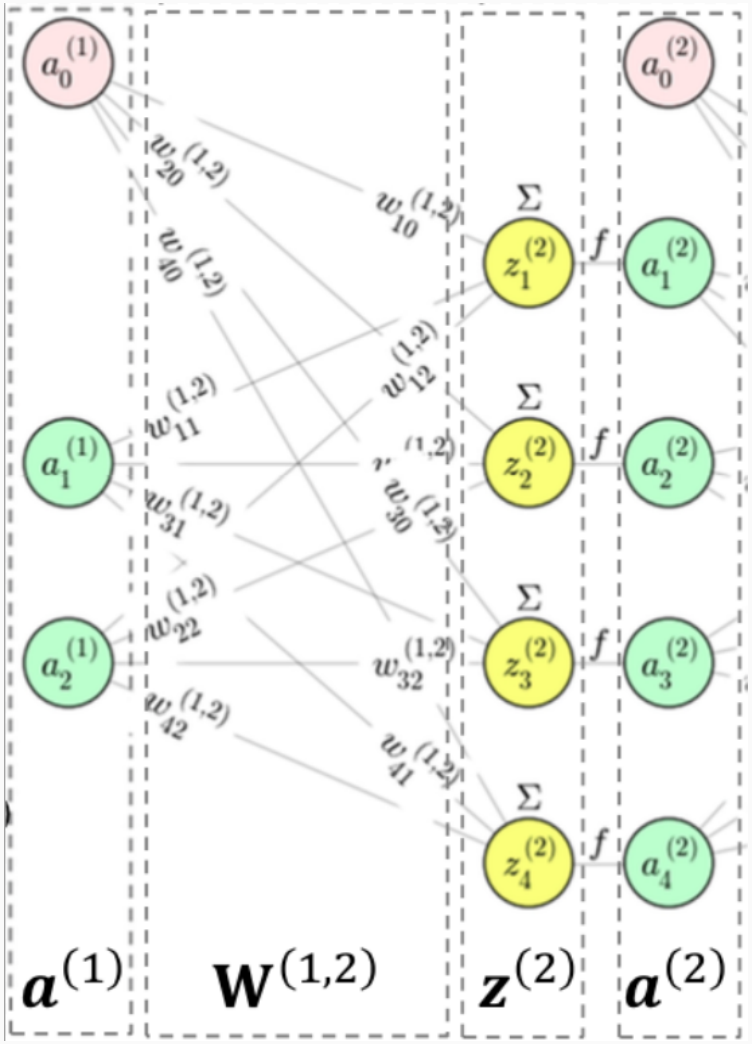


Figure 1: Forward propagation in an MLP: $\mathbf{a}^{(\ell+1)} = f\left(W^{(\ell,\ell+1)}\mathbf{a}^{(\ell)}\right)$

MLP: Single-Layer Computation



$$a_1^{(2)} = f\left(\sum_j w_{1j}^{(1,2)} a_j^{(1)}\right) = f\left(\langle \mathbf{W}_1^{(1,2)}, \mathbf{a}^{(1)} \rangle\right)$$

$$a_2^{(2)} = f\left(\sum_j w_{2j}^{(1,2)} a_j^{(1)}\right) = f\left(\langle \mathbf{W}_2^{(1,2)}, \mathbf{a}^{(1)} \rangle\right)$$

$$a_3^{(2)} = f\left(\sum_j w_{3j}^{(1,2)} a_j^{(1)}\right) = f\left(\langle \mathbf{W}_3^{(1,2)}, \mathbf{a}^{(1)} \rangle\right)$$

$$a_4^{(2)} = f\left(\sum_j w_{4j}^{(1,2)} a_j^{(1)}\right) = f\left(\langle \mathbf{W}_4^{(1,2)}, \mathbf{a}^{(1)} \rangle\right)$$

For layer ℓ with n_ℓ neurons:

$$\mathbf{a}^{(\ell)} = (a_0^{(\ell)} \cdots a_{n_\ell}^{(\ell)})$$

$$\mathbf{W}^{(\ell,\ell+1)} = (\mathbf{W}_1^{(\ell,\ell+1)} \cdots \mathbf{W}_{n_k}^{(\ell,\ell+1)})$$

Matrix computation for a layer, similar to a LogReg!

We have $\mathbf{a}^{(\ell+1)} = f(\mathbf{z}^{(\ell+1)}) = f(\mathbf{W}^{(\ell,\ell+1)}\mathbf{a}^{(\ell)})$

MLP: Principle

We stack layers: the output of layer ℓ becomes the input of layer $\ell + 1$.
Let $h^{(k,k+1)}$ denote the mapping from layer k to layer $k + 1$:

$$\begin{aligned}
\mathbf{a}^{(2)} &= h^{(1,2)}\big(\mathbf{a}^{(1)}\big), \\
\mathbf{a}^{(3)} &= h^{(2,3)}\big(\mathbf{a}^{(2)}\big), \\
&\vdots \\
\mathbf{a}^{(L)} &= h^{(L-1,L)}\big(\mathbf{a}^{(L-1)}\big).
\end{aligned}$$

Thus

$$\mathbf{a}^{(L)} = h^{(L-1,L)} \circ h^{(L-2,L-1)} \circ \cdots \circ h^{(1,2)}\big(\mathbf{a}^{(1)}\big),$$

and

$$\mathrm{MLP} = h^{(L-1,L)} \circ \cdots \circ h^{(1,2)}, \quad \text{so that} \quad \mathrm{MLP}\big(\mathbf{a}^{(1)}\big) = \mathbf{a}^{(L)}.$$

A final function as a composition of functions

The MLP can be viewed as a composition of nonlinear functions, enabling a complex function that goes from the input data to the desired output.

Activation Functions

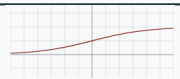
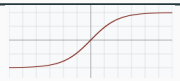
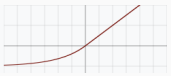
Heaviside Step Function

- In theory, f is the Heaviside step function:

$$f(z) = \mathbf{1}_{\mathbb{R}_+}(z) = \begin{cases} 0 & z < 0, \\ 1 & z \geq 0. \end{cases}$$

- Problem: this function is not differentiable.

In practice, differentiable functions are used, such as:

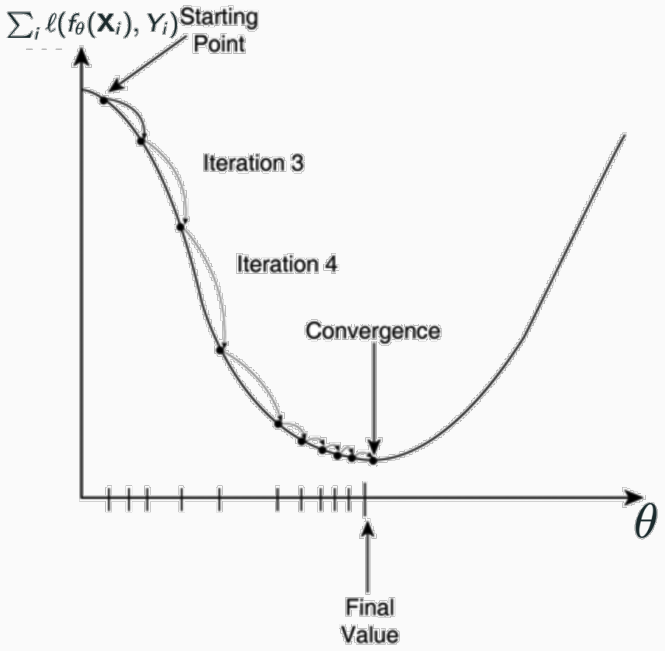
Name	Plot	Formula	Limits at $\pm\infty$	Derivative at $\pm\infty$
Sigmoid		$f(z) = \frac{1}{1+e^{-z}}$	$0; 1$	$0; 0$
Tanh		$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$-1; 1$	$0; 0$
ReLU[1]		$f(z) = \begin{cases} 0 & z < 0, \\ z & z \geq 0 \end{cases}$	$0; z$	$0; 1$
ELU[2]		$f(z) = \begin{cases} \alpha(e^z - 1) & z < 0, \\ z & z \geq 0 \end{cases}$	$-\alpha; z$	$0; 1$

[1] Rectified Linear Unit
[2] Exponential Linear Unit

Reminders and Training

Outline : Reminders and Training

Reminders: Optimization



Cost Function Optimization

- Used to converge to the minimum of the loss value in the training set
- Best case: fast and accurate
- Often approximations are made to speed up convergence

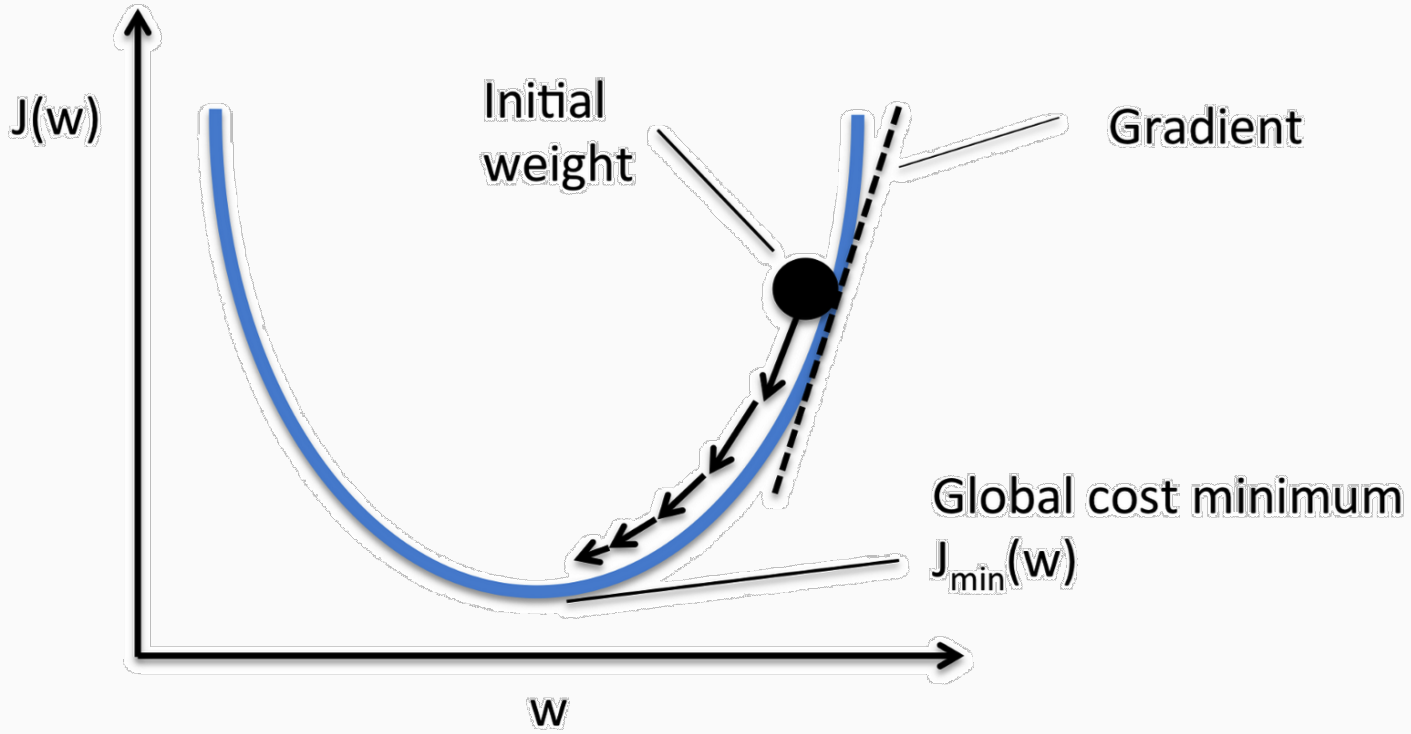
Gradient

A multivariate function $f(x)$ can be expanded via a Taylor series:

$$f(x + \delta_x) = f(x) + \nabla_x f(x)\delta_x + \mathcal{O}(||\delta_x^2||)$$

Definition

The gradient $\nabla_x f(x) = (\frac{\partial f}{\partial x_i})_{i=1..n}$ is the vector of partial derivatives. It provides a linear approximation at the local level of f at x.



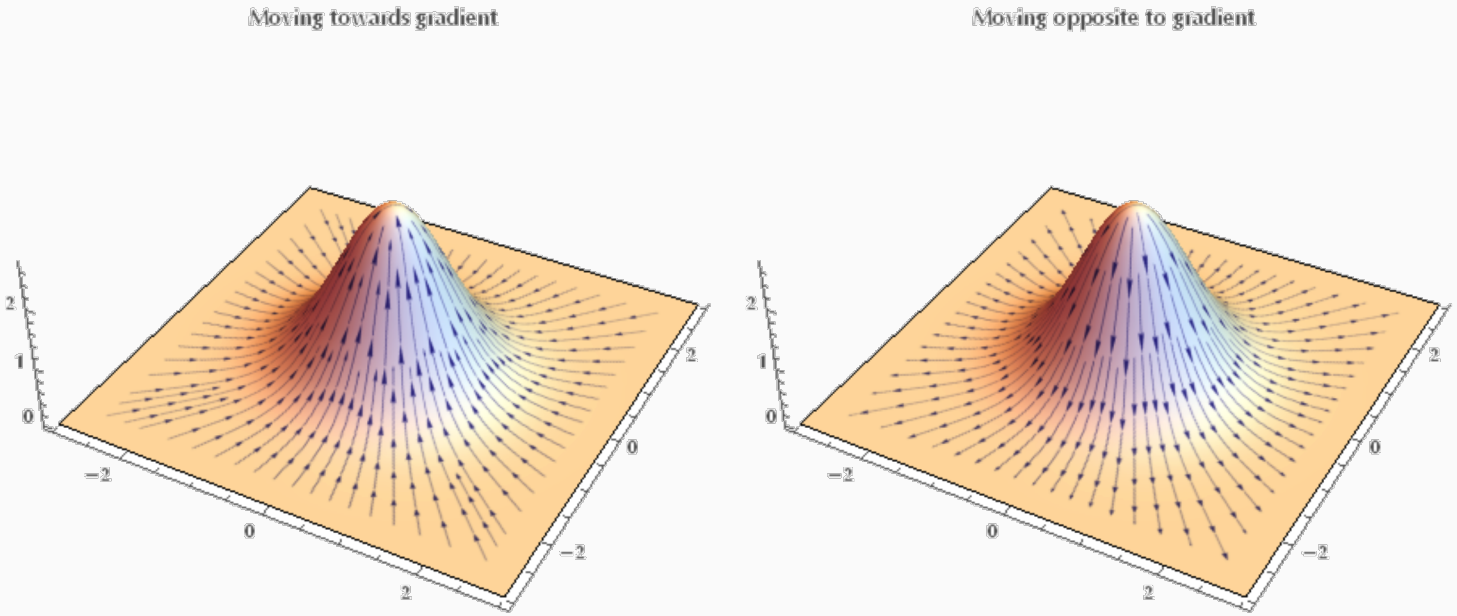
Gradient

A multivariate function $f(x)$ can be expanded via a Taylor series:

$$f(x + \delta_x) = f(x) + \nabla_x f(x)\delta_x + \mathcal{O}(||\delta_x^2||)$$

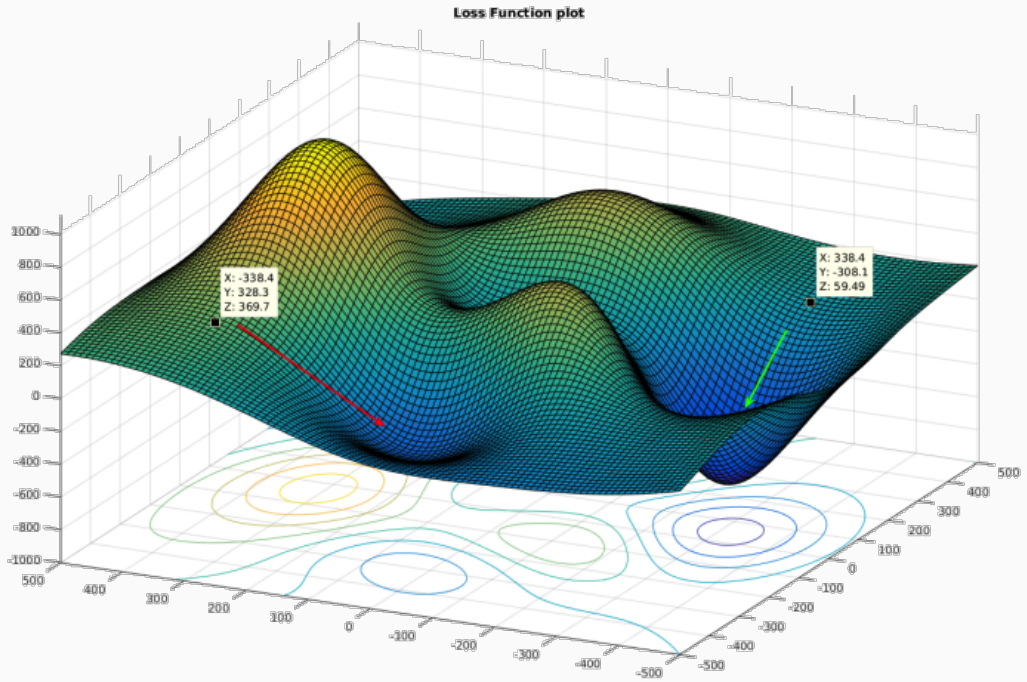
Definition

The gradient $\nabla_x f(x) = (\frac{\partial f}{\partial x_i})_{i=1..n}$ is the vector of partial derivatives. It provides a linear approximation at the local level of f at x.



More details here.

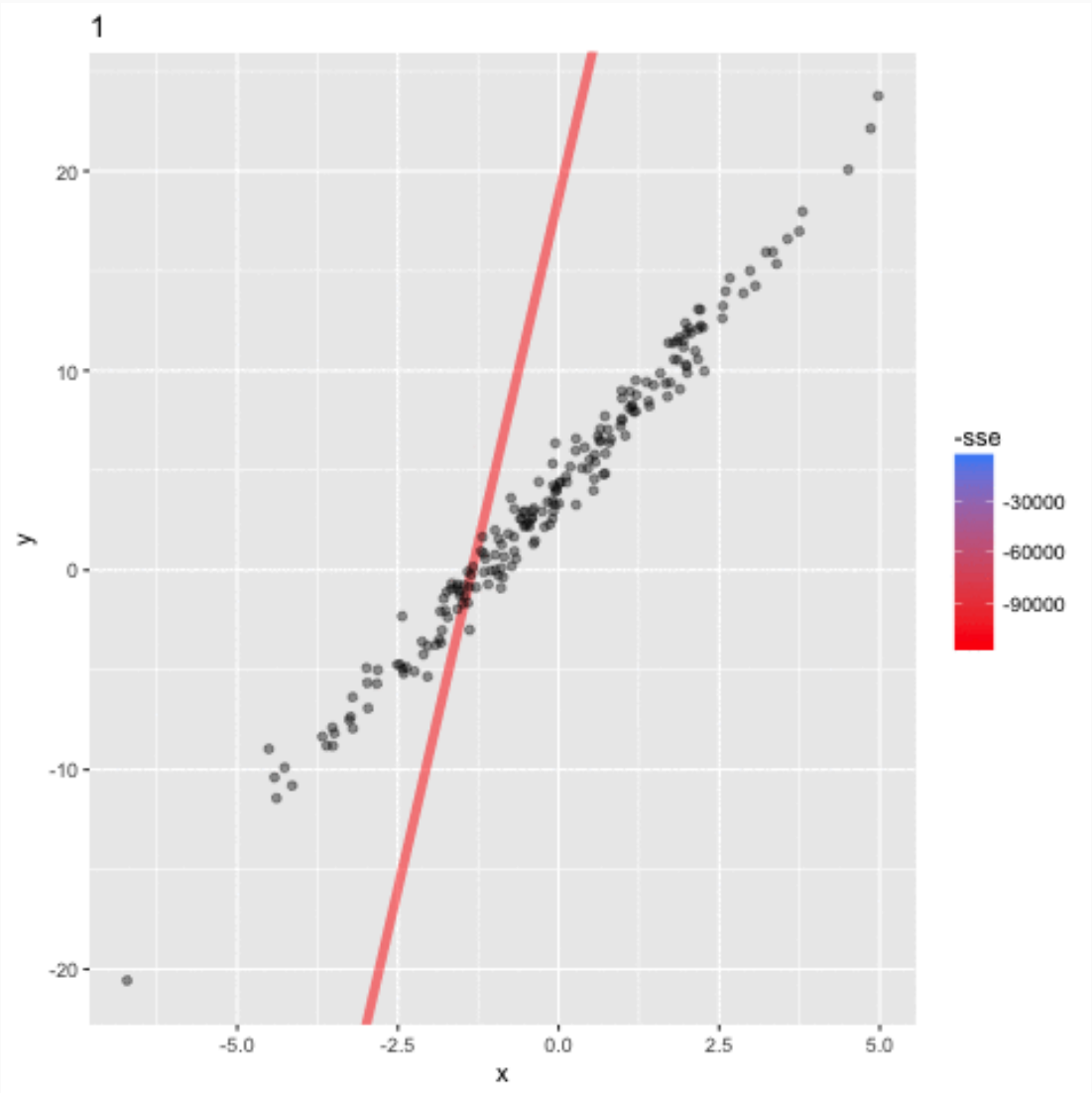
Visualizing the Loss Landscape



You can view the value of the loss function as a surface:

- Parameters θ span the plane, and the loss $\ell(\mathcal{D}_n; \theta)$ varies in height.
- Convergence occurs when encountering values of θ that land in a valley of this surface (a local or global minimum).

Optimization: Visualization



Optimization: Stochastic Gradient Descent

Gradient Descent

After each calculation of the cost function $\ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$, the gradient of this function is calculated to update the parameters θ:

$$\theta \leftarrow \theta - \alpha * \theta \ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$$

.

Example

- Let $f_\theta(\mathbf{X}) = \theta^T \mathbf{X} = \sum_k^d \theta_k X^{(k)}$ and $\ell(Y, f_\theta(\mathbf{X})) = \frac{1}{2}(Y - f_\theta(\mathbf{X}))^2$

$$\nabla_\theta \ell(Y_i, f_\theta(\mathbf{X}_i); \theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_1} \\ \vdots \\ \frac{\partial}{\partial \theta_d} \end{pmatrix} .\ell(Y_i, f_\theta(\mathbf{X}_i); \theta) = \begin{pmatrix} X_i^{(1)}(Y - f_\theta(\mathbf{X}_i)) \\ \vdots \\ X_i^{(d)}(Y_i - f_\theta(\mathbf{X}_i)) \end{pmatrix}$$

Optimization: Stochastic Gradient Descent

Gradient Descent

After each calculation of the cost function $\ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$, the gradient of this function is calculated to update the parameters θ:

$$\theta \leftarrow \theta - \alpha * \theta \ell(Y_i, f_\theta(\mathbf{X}_i); \theta)$$

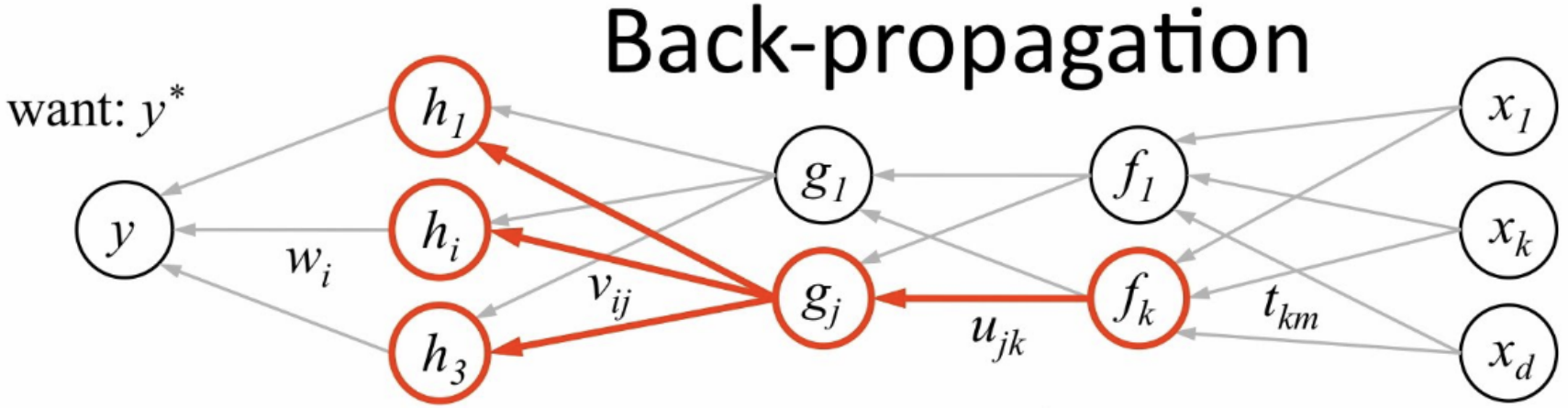
.

Example

- $f_\theta(\mathbf{X}) = \theta^T \mathbf{X} = \sum_k^d \theta_k X^{(k)}$ y $\ell(Y, f_\theta(\mathbf{X})) = \frac{1}{2}(Y - f_\theta(\mathbf{X}))^2$

$$\begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \end{pmatrix} \leftarrow \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_d \end{pmatrix} - \alpha * \begin{pmatrix} X_i^{(1)}(Y - f_\theta(\mathbf{X}_i)) \\ \vdots \\ X_i^{(d)}(Y_i - f_\theta(\mathbf{X}_i)) \end{pmatrix}$$

Backpropagation: Principle



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer 1…L
   compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y\text{-}y^*)$
4. **back-propagate error:** for each unit g_j in each layer L…1

(a) compute error on g_j

$$\frac{\partial E}{\partial g_j} = \sum_i \sigma'(h_i) v_{ij} \frac{\partial E}{\partial h_i}$$

should g_j be higher or lower? | how h_i will change as g_j changes | was h_i too high or too low?

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \sigma'(g_j) f_k$$

do we want g_j to be higher/lower | how g_j will change if u_{jk} is higher/lower

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

More on backpropagation on this video, and generalities in this blog.

Stochastic Gradient Descent

- Computations can be parallelized $\Rightarrow$ GPU!
- It is not possible to use the whole train dataset to update the model parameters, the technique of the **minibatch** is used.
- We are going to update the model weights by calculating the error on **m** examples that we are going to take randomly in the dataset
- A **larger m** means less noisy updates

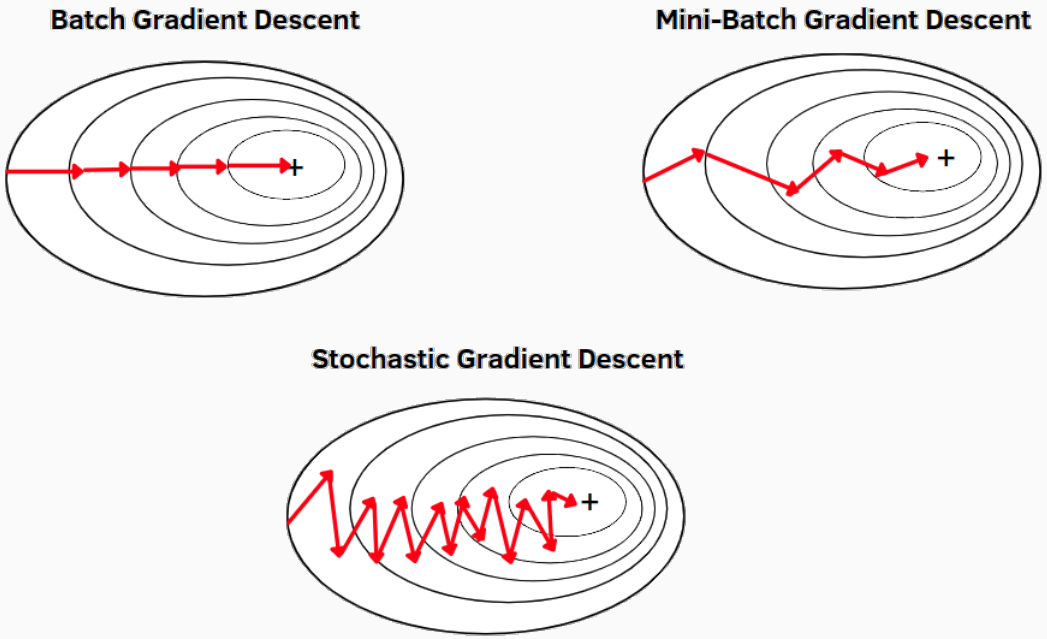


Figure 2: Comparing full-batch vs. mini-batch gradient descent

Representations

Outline : Representations

Outline : Representations and Composition

MLP – Representations

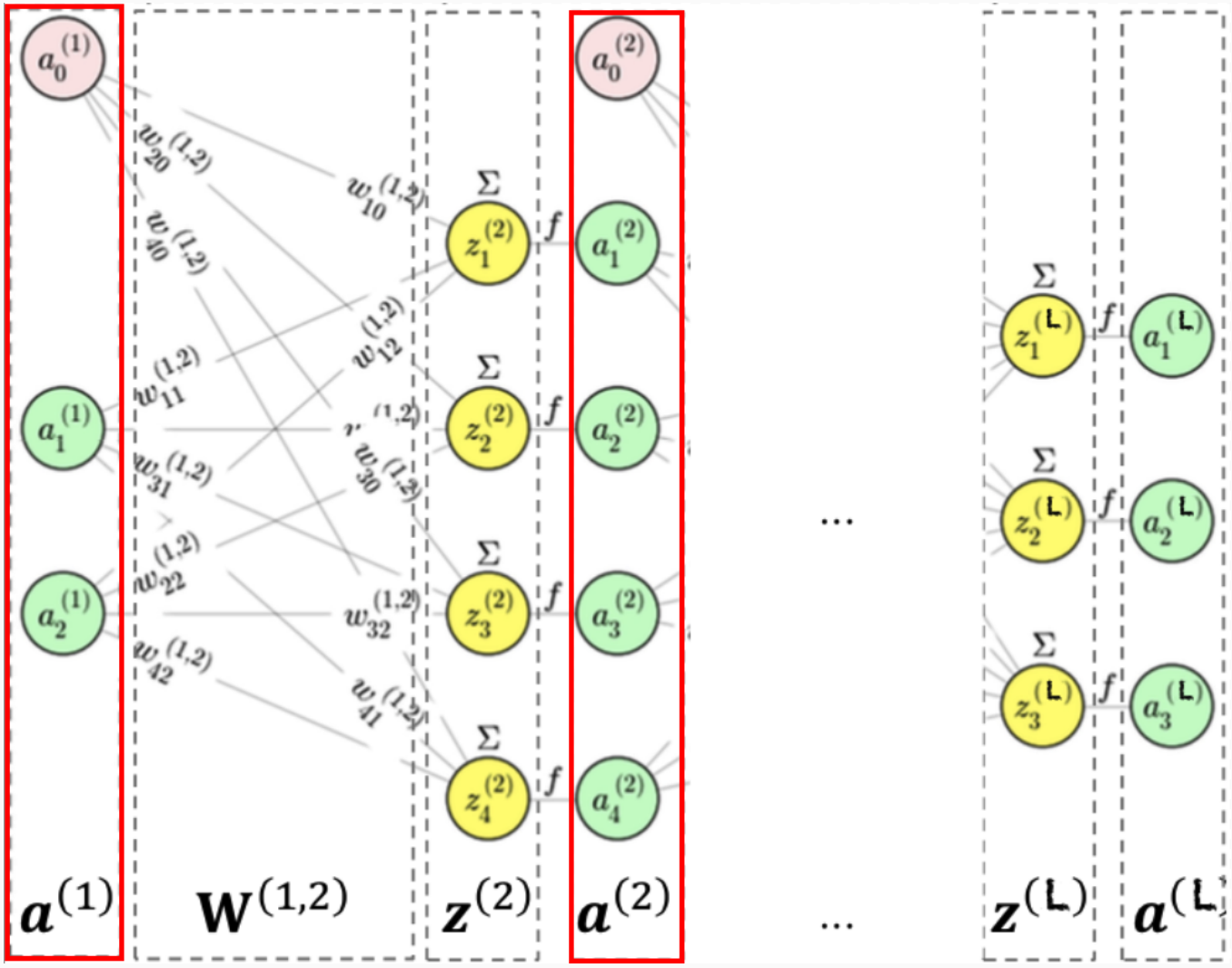


Figure 3: Each hidden layer $\mathbf{a}^{(\ell)}$ $(\ell \in \{2, \ldots, L-1\})$ is a vector representation of the input feature vector $\mathbf{a}^{(1)}$.

MLP - Complexity of Representations and Hierarchy

Complexity of Representations

- Input $\mathbf{a}^{(1)}$ is a "raw" feature vector; output $\mathbf{a}^{(L)}$ solves a complex task.
- Deeper layers (ℓ larger) produce more complex representations.
- $\mathbf{a}^{(1)}$ is low-level; $\mathbf{a}^{(L)}$ is high-level.

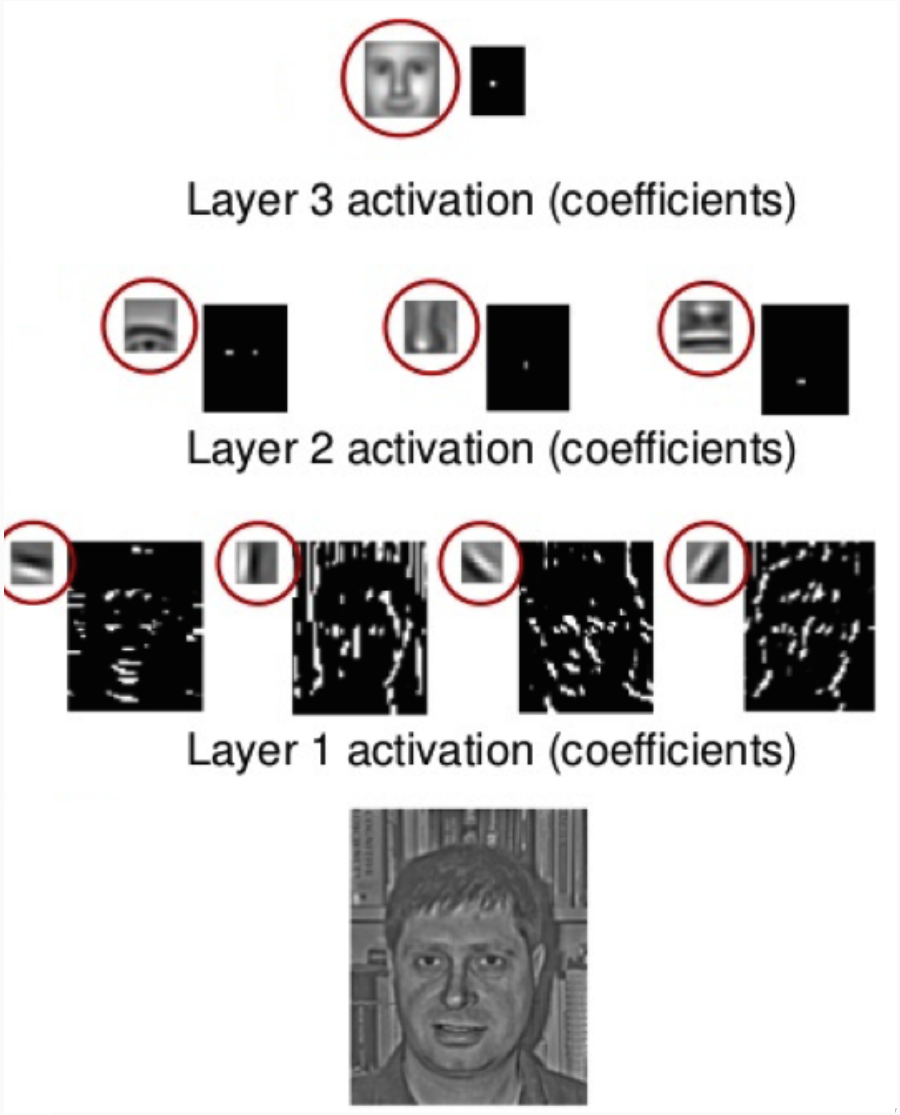
For images, layers may detect:

- I edges and contours
- II nose, mouth, eyes, etc.
- III faces and objects

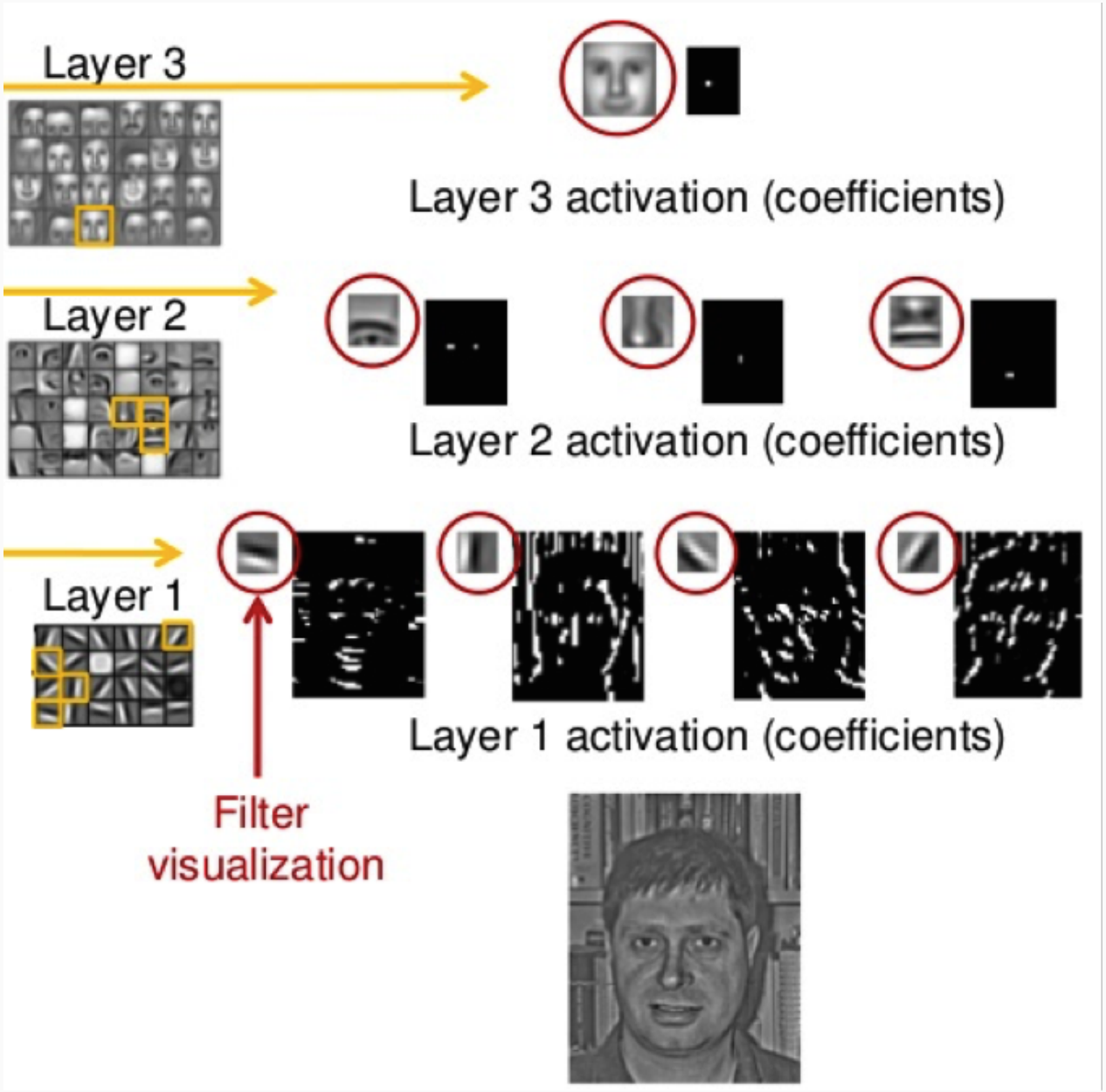
For text, layers may detect:

- I part-of-speech patterns
- II named entities
- III sentiment or other complex concepts

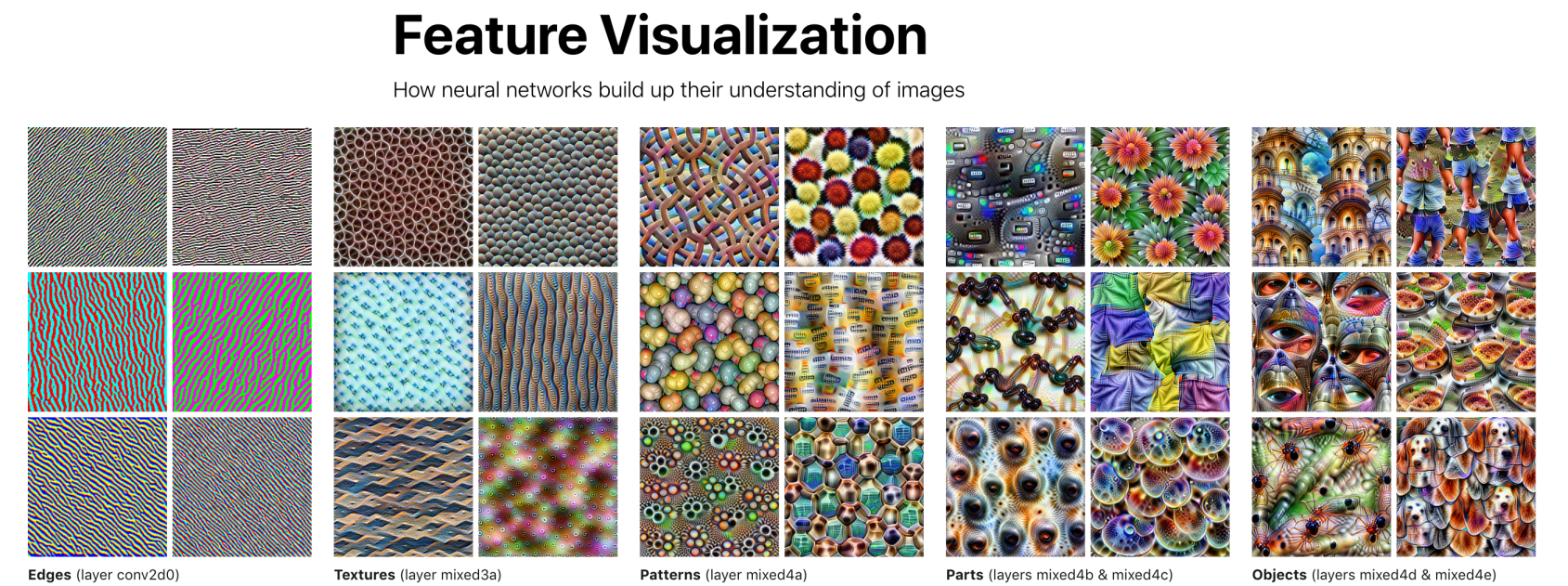
Complexity of Representations: Example



Complexity of Representations: Compositionality



Visualization of AlexNet Activations



Deep Visualization Toolbox

Distill.pub: Feature Visualization

Outline : Representation Learning for Transfer

Representation Learning

- Each layer yields a representation of the input data with lower dimensions.
- These learned representations (embeddings) learned from the raw data alone sometimes outperform sets of classical descriptors.

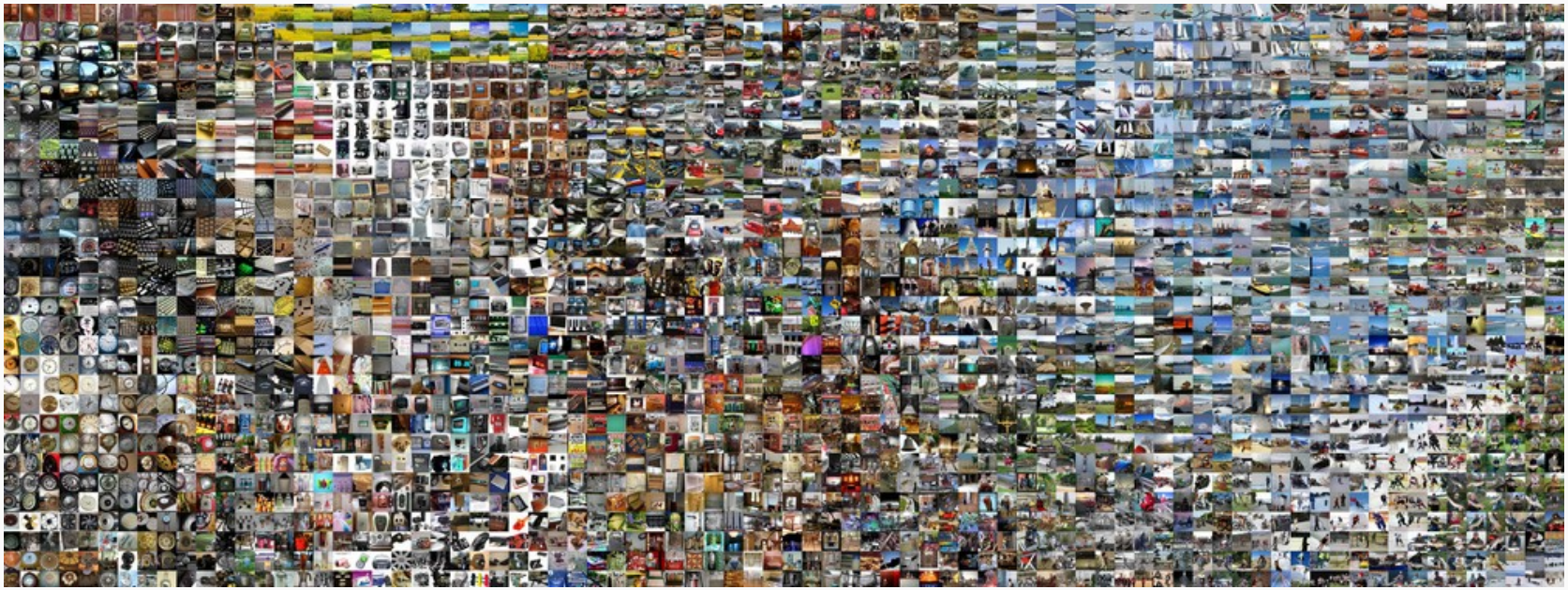
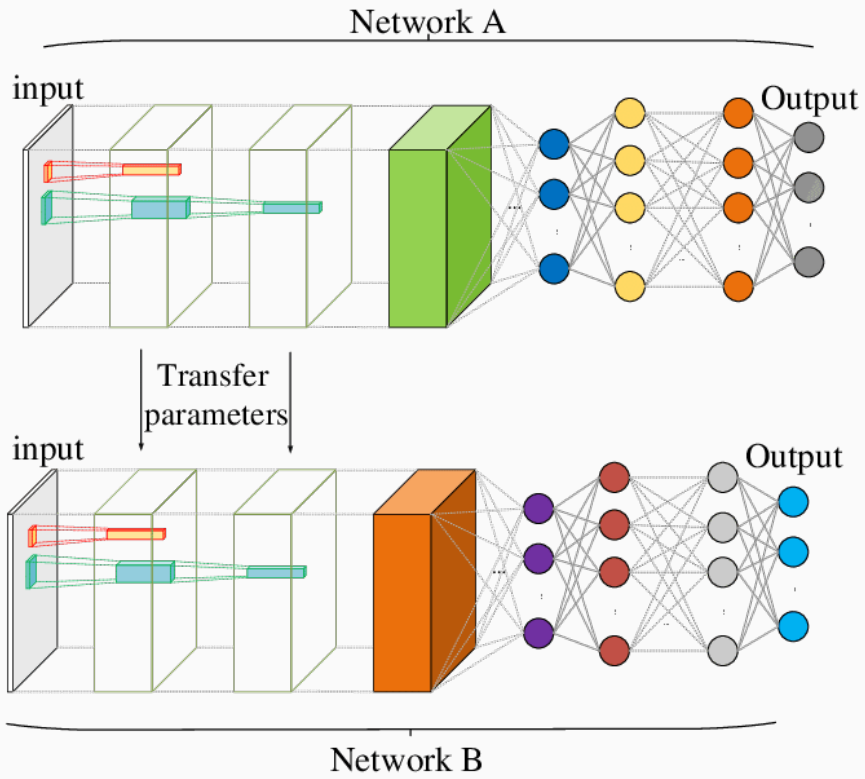


Figure 4: 4096-dimensional AlexNet feature vector projected to 2D via t-SNE

Transfer Learning and Fine-tuning

Fine-tuning

Process of fitting in the new $\mathcal{D}$ dataset. It may involve *(i)* re-training all layers, *(ii)* only the deeper layers related to specific tasks (freezing the initial layers), or *(iii)* re-training the deep layers and unfreezing the initial layers gradually.



Frameworks

Outline : Frameworks

Keras

A high-level Python library for Deep Learning, running on top of TensorFlow and other backends.

- Vision: Keras tutorial for fine-tuning a pre-trained VGG16, which can be used with: pre-trained CNNs available in Keras
- Text and audio:
  Tutorial RNN-LSTM Seq2seq for machine translation
- Text: Use of pre-trained word embeddings

PyTorch

A Python library for Deep Learning, a competitor to TensorFlow.

- Vision: Tutorial for fine-tuning a pre-trained ResNet18
- Audio: Speech Recognition with Wav2Vec2
- Text: RNN-GRU Seq2seq tutorial for machine translation

HuggingFace Transformers

A Python library for **Transformer** models (a class of deep neural networks).

Hugging Face

- Many pre-trained models for images, audio, multimodal data, text, ...
- From classical classifiers to generative models and embeddings, up to large models (e.g. Llama3-70B).
- Related libraries: `Diffusers`, `Datasets`, `Accelerate`, `PEFT`, `bitsandbytes`, `TRL`, ...

Questions?

References i