ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf Informatik Computer Science

Introduction to Machine Learning

Neural networks / “feature learning”

Dr. Kfir Levy
Learning and Adaptive Systems (las.ethz.ch)

What are good features?

- Classification of handwritten digits (e.g. MNIST data)
- What properties should good features have?
- What features would you use?
- Examples:
  - Pixels?
  - Edge Detectors?
  - Strokes?
  - Others?

Importance of features

- Success in learning crucially depends on the quality of features
- Hand-designing features requires domain-knowledge
- What about kernel methods?
  - Rich set of feature maps
  - Can fit „any function“ with infinite data*
  - Choosing the „right“ kernel can be challenging
  - Computational complexity grows with size of data

- Can we learn good features from data directly??

Learning features

- Learning with *m* hand-designed features

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell\Big(y_i; \underbrace{\sum_{j=1}^{m} w_j \phi_j(\mathbf{x}_i)\Big)}_{f_i}$$

$\ell(y_i, f_i) = (y_i - f_i)^2$

- **Key Idea**: Parameterize the feature maps, and optimize over the parameters!

$$\mathbf{w}^* = \arg\min_{\mathbf{w}, \boxed{\theta}} \sum_{i=1}^{n} \ell\Big(y_i; \sum_{j=1}^{m} w_j \phi(\mathbf{x}_i, \theta_j)\Big)$$

Parameterizing feature maps

- One possibility:

$$\phi(\mathbf{x}, \theta) = \varphi(\underbrace{\theta^T \mathbf{x}}_{z})$$

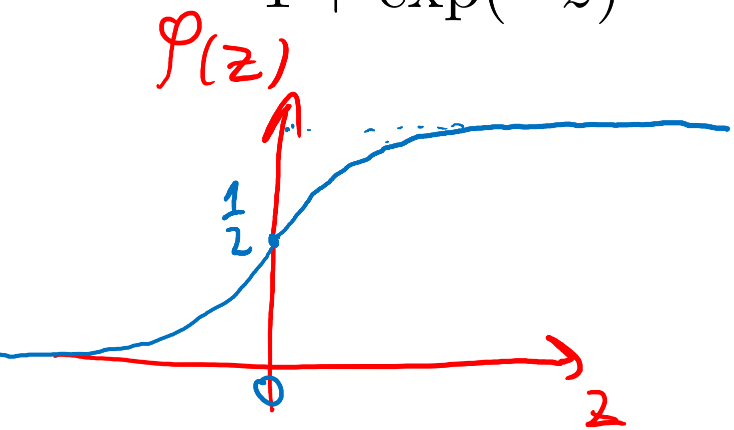
- Hereby, $\theta \in \mathbb{R}^d$ and $\varphi : \mathbb{R} \to \mathbb{R}$ is a nonlinear function, called „activation function“

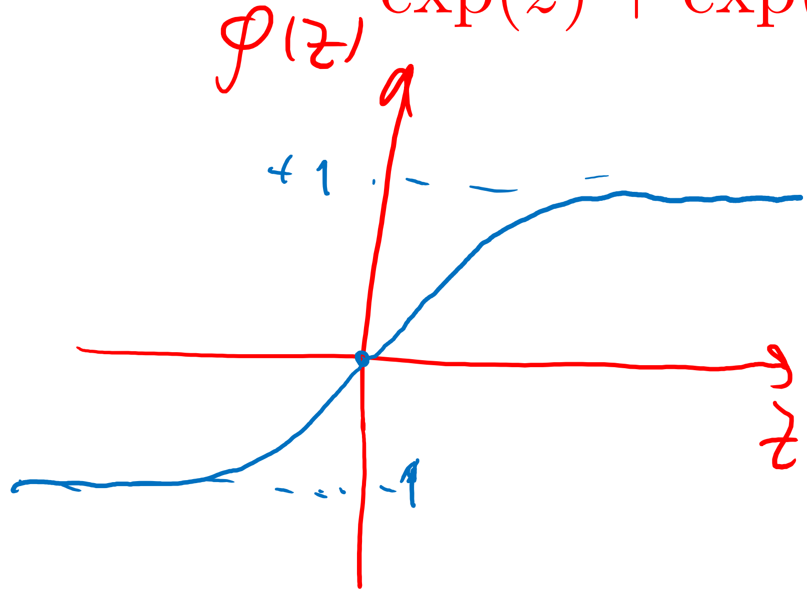
Sigmoid activation and variants

- Sigmoid and tanh activation function

$$\varphi(z) = \frac{1}{1+\exp(-z)}$$

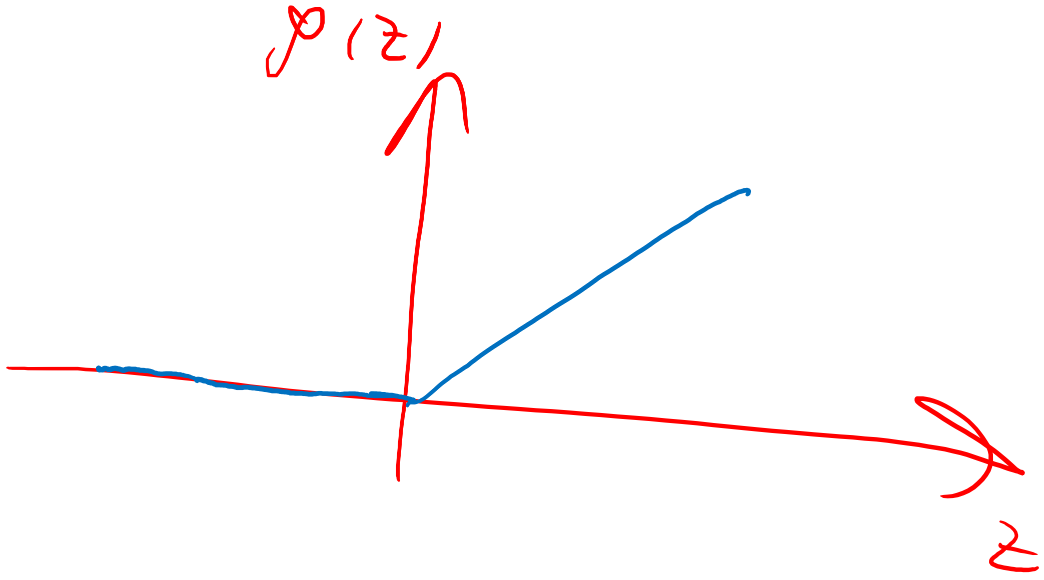
$$\varphi(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$





Rectified linear units (ReLU)

$$\varphi(z) = \max(z, 0)$$



Artificial Neural networks (ANNs)

- Functions of this form

$$f(x; w, \theta) = \sum_{j=1}^{m} w_j \underbrace{\varphi(\theta_j^T \mathbf{x})}_{v_j}$$

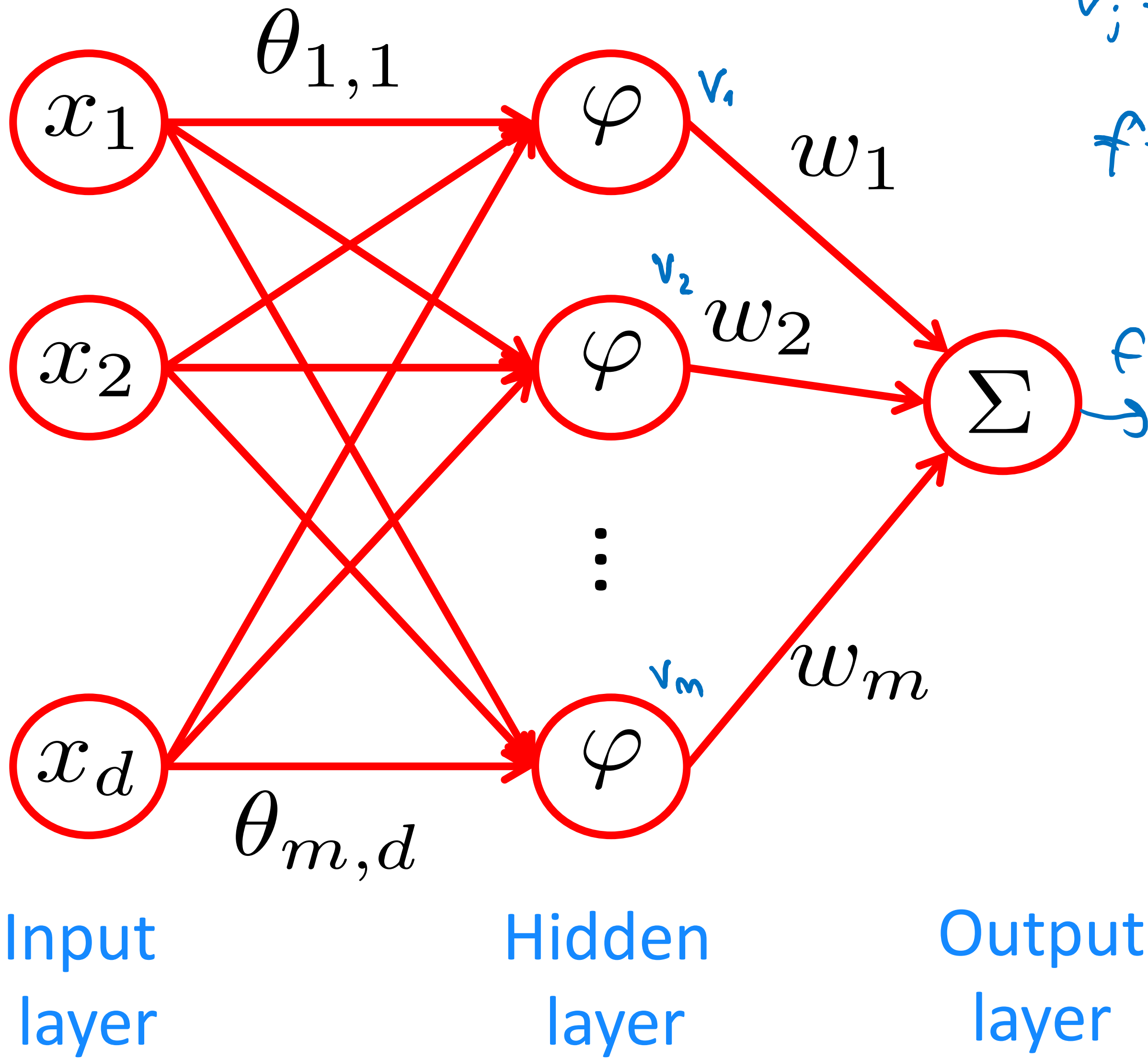
  are (examples of) artificial neural networks (ANNs) (also called Multi-layer Perceptrons)

- More generally, the term artificial neural network refers to nonlinear functions which are nested compositions of (variable) linear functions composed with (fixed) nonlinearities

Graphical illustration

$V_j = \varphi(\theta_j^T x)$

$f = \sum_{j=1}^{m} w_j V_j = w^T V$

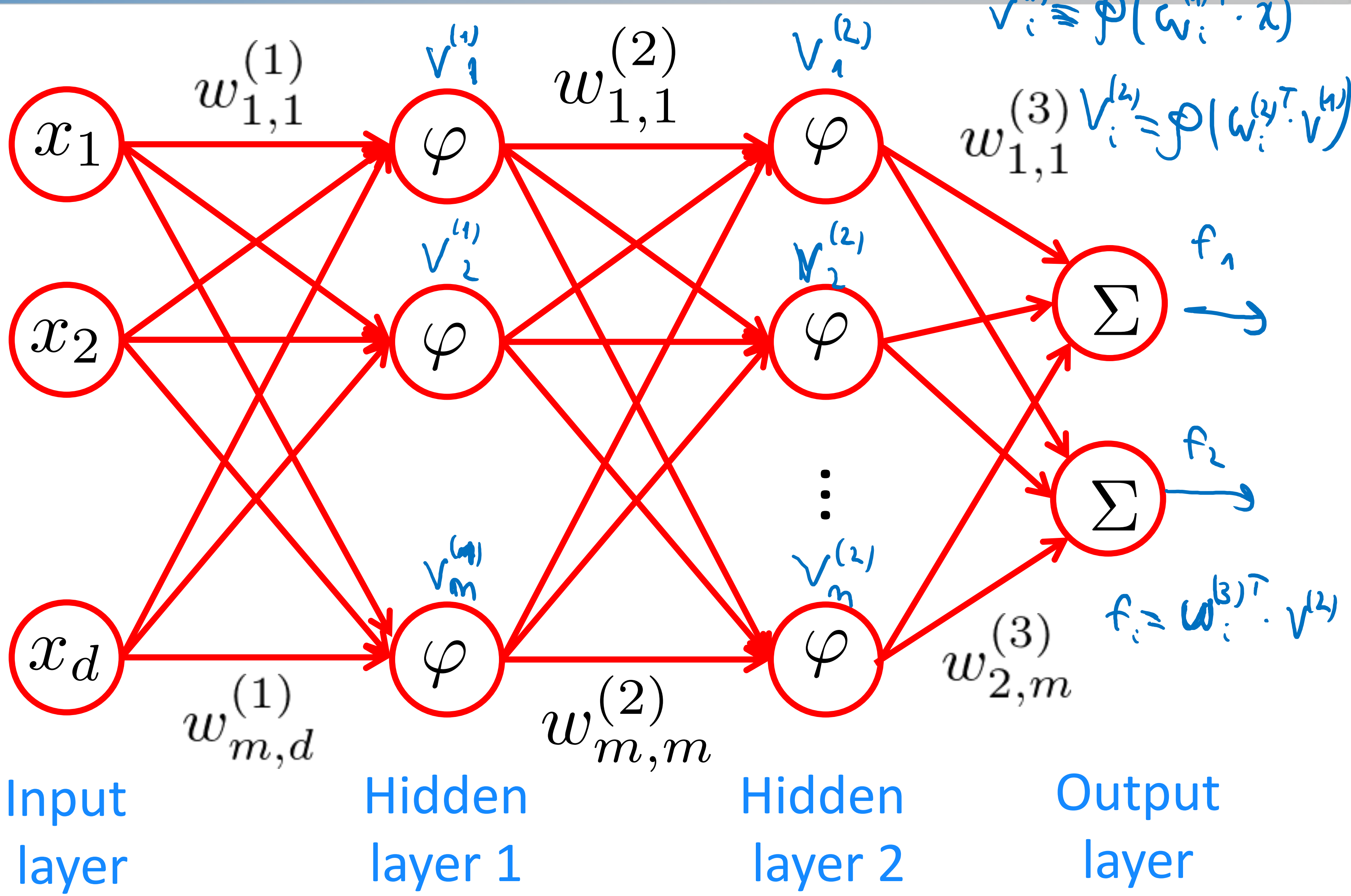


Some comments

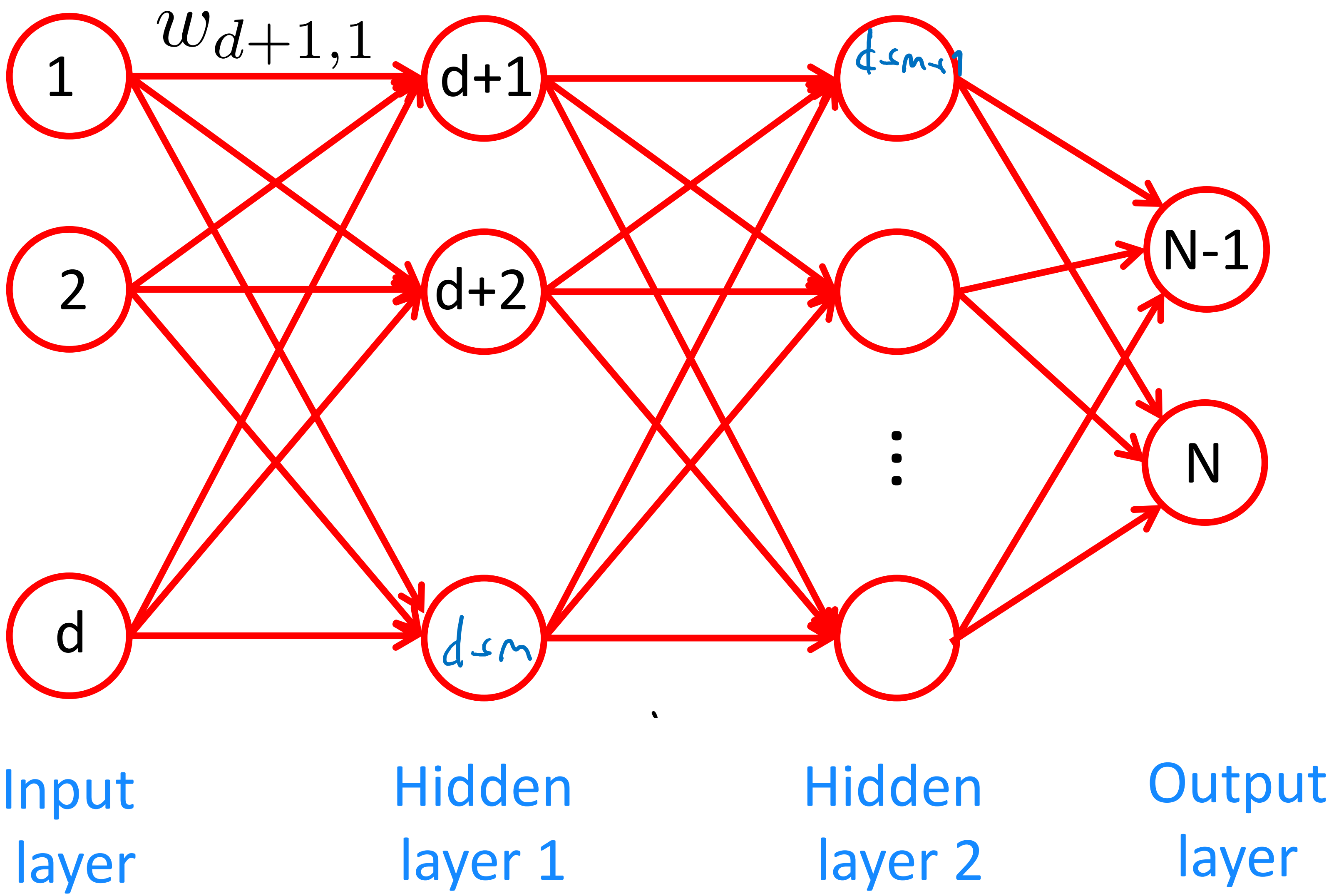
- Can have more than one output
  - Useful, e.g., for multi-class prediction (one output per class), or multi-output regression

- Can have more than one hidden layer
  - Neural networks with several hidden layers ≈ „Deep Learning“

More complex network example



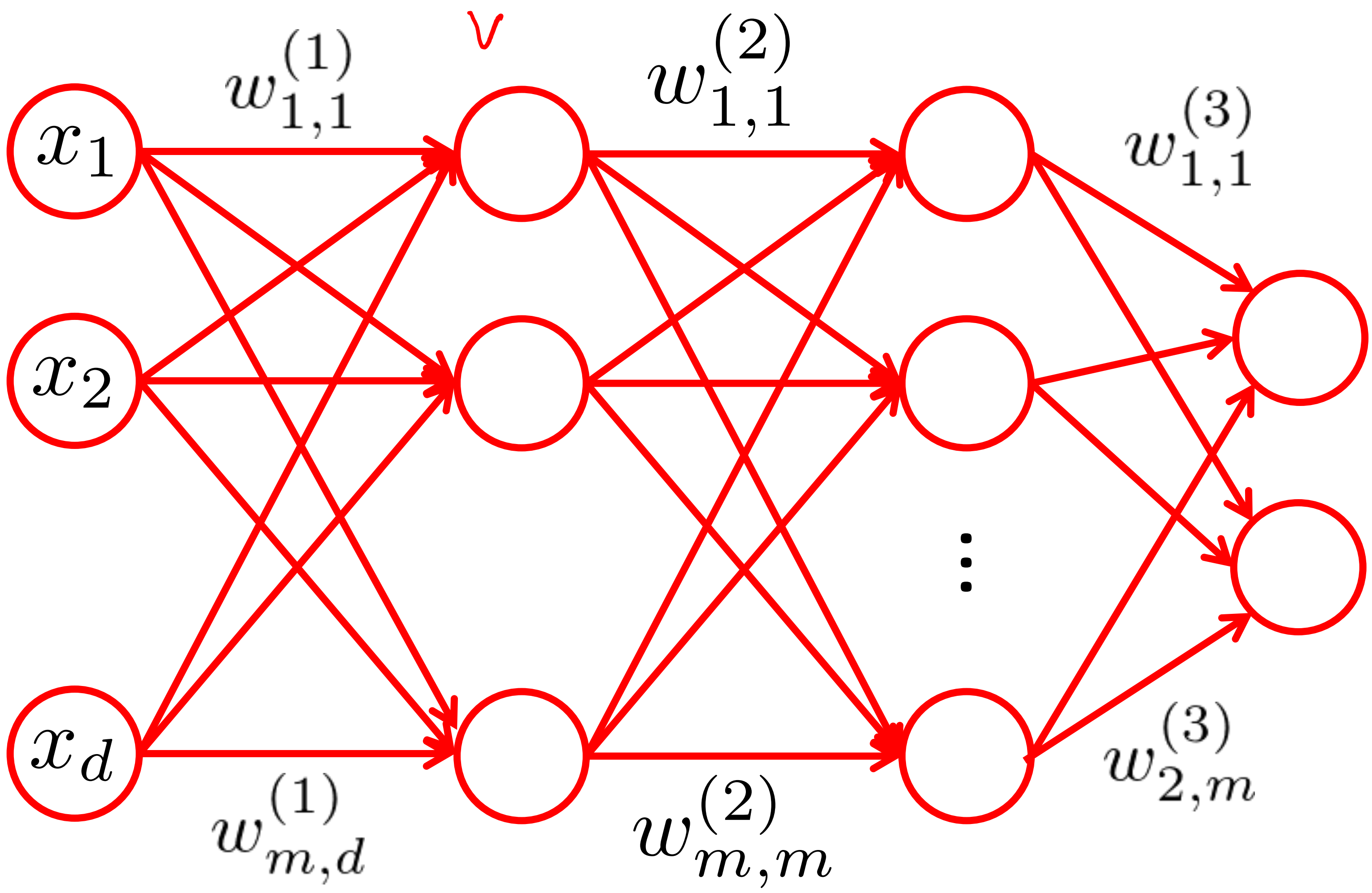
Indexing units



Making predictions

- Suppose we have learned all parameters $w_{i,j}$
- Given an input, how do we make predictions?
- Forward propagation!

Forward propagation



Input layer

Hidden layer 1

Hidden layer 2

Output layer

Forward propagation

- For each unit j on input layer, set its value $v_j = x_j$
- For each layer $\ell = 1 : L - 1$
  - For each unit j on layer ℓ set its value

$$v_j = \varphi\Big(\sum_{i \in \text{Layer}_{\ell-1}} w_{j,i} v_i\Big)$$

- For each unit j on output layer, set its value

$$f_j = \sum_{i \in \text{Layer}_{L-1}} w_{j,i} v_i$$

- Predict $y_j = f_j$ for regression,
  $y_j = \text{sign}(f_j)$ for classification

Multiclass problems

$\hat{y} = \arg\max_j f_j$

Forward propagation (short notation)

- For input layer: $\mathbf{v}^{(0)} = \mathbf{x}$
- For each hidden layer $\ell = 1 : L-1$

$$\mathbf{z}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{v}^{(\ell-1)}$$

$z^{(l)} \in R^{m^{(l)}}$

$m^{(l)}$ – number of units in the l^{th} layer

$$\mathbf{v}^{(\ell)} = \varphi(\mathbf{z}^{(\ell)})$$

$\varphi(z^{(l)}) = (\varphi(z_1^{(l)}), \varphi(z_2^{(l)}), \dots, \varphi(z_{m^{(l)}}^{(l)}))$

- For output layer: $f = \mathbf{W}^{(L)}\mathbf{v}^{(L-1)}$

- Predict: $\mathbf{y} = \mathbf{f}$ (regression) or $\mathbf{y} = \mathrm{sign}(\mathbf{f})$ (class.)

$\hat{y} = \arg\max_i f_i$

Universal Approximation Theorem

Theorem 2. *Let σ be any continuous sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^{\mathrm{T}} x + \theta_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \varepsilon \qquad \text{for all} \quad x \in I_n.$$

- Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314

➔ demo

How can we train the weights?

- Given data set $D = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$

  want to optimize weights $\mathbf{W} = (\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)})$
- How do we measure and optimize goodness of fit?

➔ Apply loss function (e.g., Perceptron loss, multi-class hinge loss, square loss, etc.) to output

example

$$\ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \ell(\mathbf{y} - f(\mathbf{x}, \mathbf{W})) = (y - f(x, W))^2$$

➔ Then optimize the weights to minimize loss over D

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{i=1}^{n} \ell(\mathbf{W}; \mathbf{y}_i, \mathbf{x}_i)$$

Side note: Losses for multi-outputs

- When predicting multiple outputs at the same time, usually define loss as sum of per-output losses:

$$\ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \sum_{i=1}^{p} \ell_i(\mathbf{W}; y_i, \mathbf{x})$$

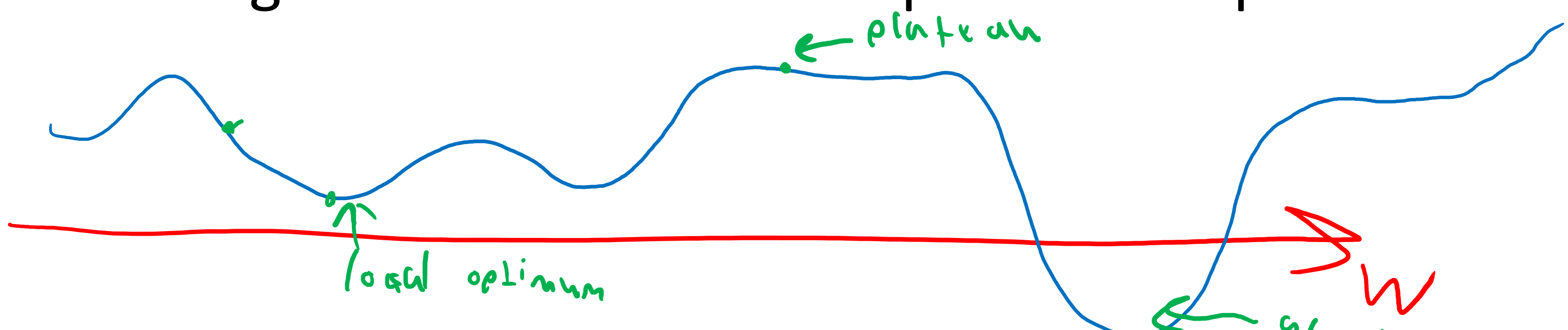
- Examples
  - For regression tasks, i.e., $y_i \in \mathbb{R}$ may use squared loss
  - For classification, may use multiclass Perceptron or hinge loss

How do we optimize over weights?

- Want to do Empirical Risk Minimization

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{i=1}^{n} \ell(\mathbf{W}; \mathbf{y}_i, \mathbf{x}_i)$$

- I.e., jointly optimize over all weights for all layers to minimize loss over the training data
- This is in general a non-convex optimization problem



- Nevertheless, can try to find a local optimum

Stochastic gradient descent for ANNs

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{i=1}^{n} \ell(\mathbf{W}; \mathbf{y}_i, \mathbf{x}_i)$$

- Initialize weights $\mathbf{W}$
- For t = 1,2,...
  - Pick data point $(\mathbf{x}, \mathbf{y}) \in D$ uniformly at random
  - Take step in negative gradient direction

$$\mathbf{W} \leftarrow \mathbf{W} - \eta_t \nabla_{\mathbf{W}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x})$$