

Introduction to Machine Learning

Neural networks / “feature learning”

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)

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

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

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

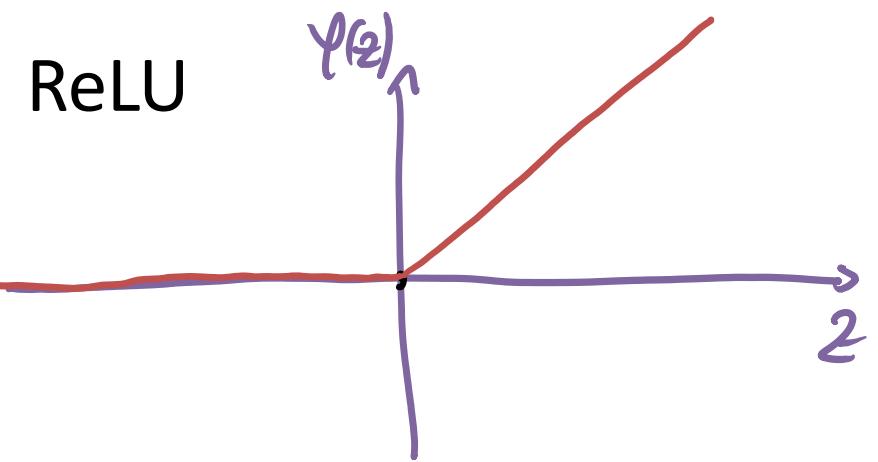
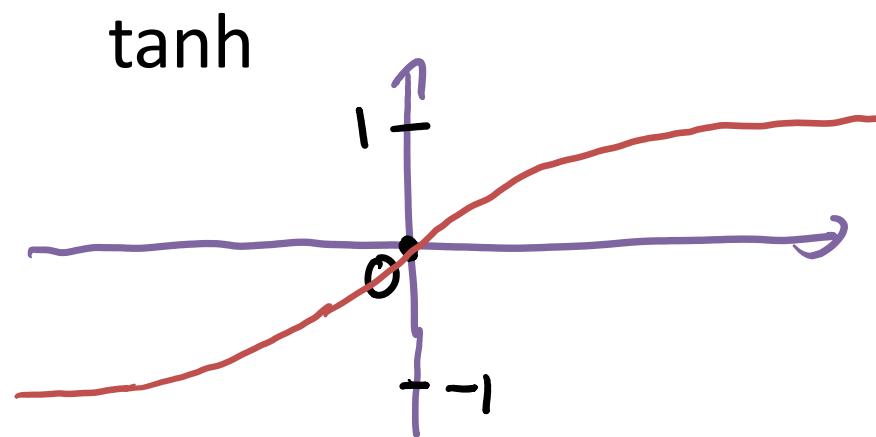
$$\mathbf{w}^* = \arg \min_{\mathbf{w}, \theta} \sum_{i=1}^n \ell\left(y_i; \sum_{j=1}^m w_j \phi(\mathbf{x}_i, \theta_j)\right)$$

Parameterizing feature maps

- One possibility:

$$\phi(\mathbf{x}, \theta) = \varphi(\underbrace{\theta^T \mathbf{x}}_2)$$

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



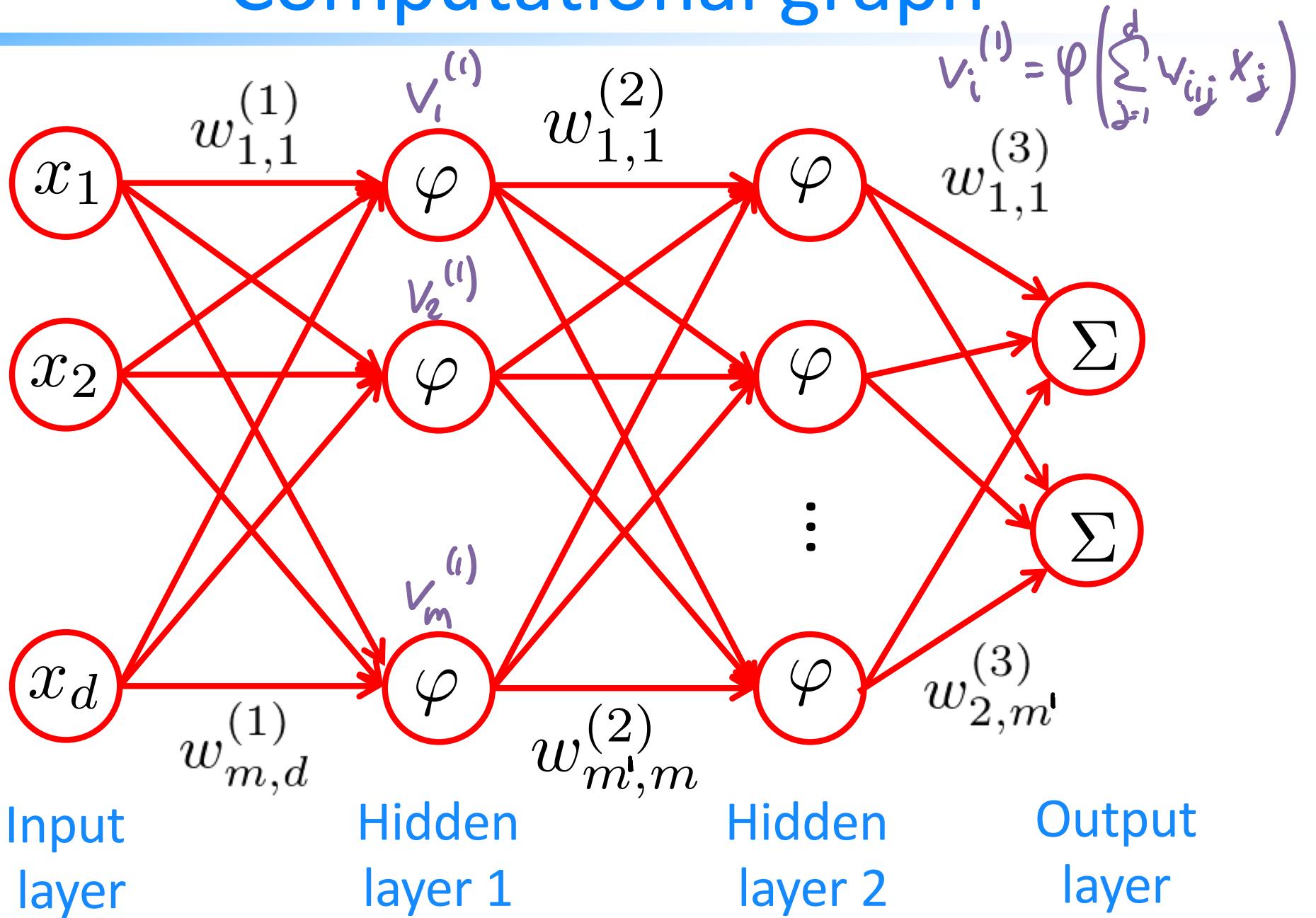
Artificial Neural networks (ANNs)

- Functions of this form

$$f(\mathbf{x}; \mathbf{w}, \theta_{1:d}) = \sum_{j=1}^m w_j \varphi(\theta_j^T \mathbf{x}) = \mathbf{w}^T \varphi(\Theta \mathbf{x})$$

- 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

Computational graph



Forward propagation (short notation)

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

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

$\varphi([z_1 \dots z_m]) = [\varphi(z_1) \dots \varphi(z_m)]$

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

- For output layer: $f = \mathbf{W}^{(L)} \mathbf{v}^{(L-1)}$
- Predict: $\mathbf{y} = f$ (regression)
or $\mathbf{y} = \text{sign}(f)$ or $y = \arg \max_i f_i$ (class.)

Universal Approximation Theorem

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

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

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

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

- 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 $D = \{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$
want to optimize weights $\mathbf{W} = (\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)})$
- 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

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

→ Then **optimize the weights** to minimize loss over D

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

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

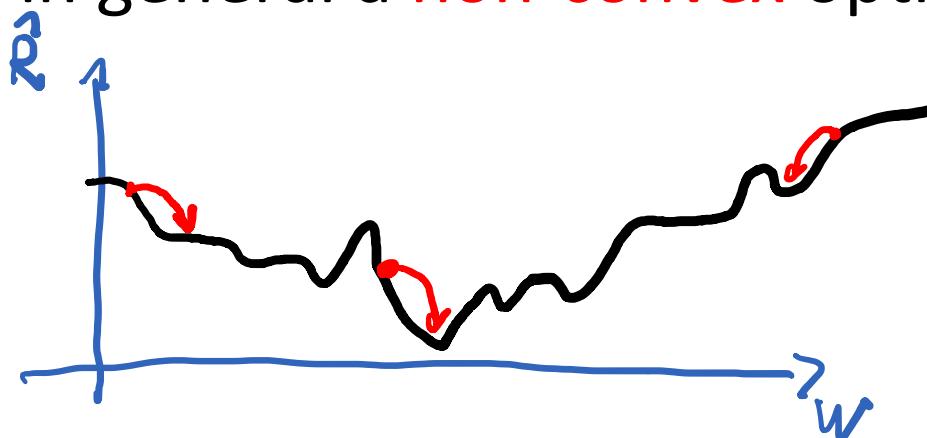
How do we optimize over weights?

- Want to do Empirical Risk Minimization

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

$\ell(w)$

- 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

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

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

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

ANN Demo

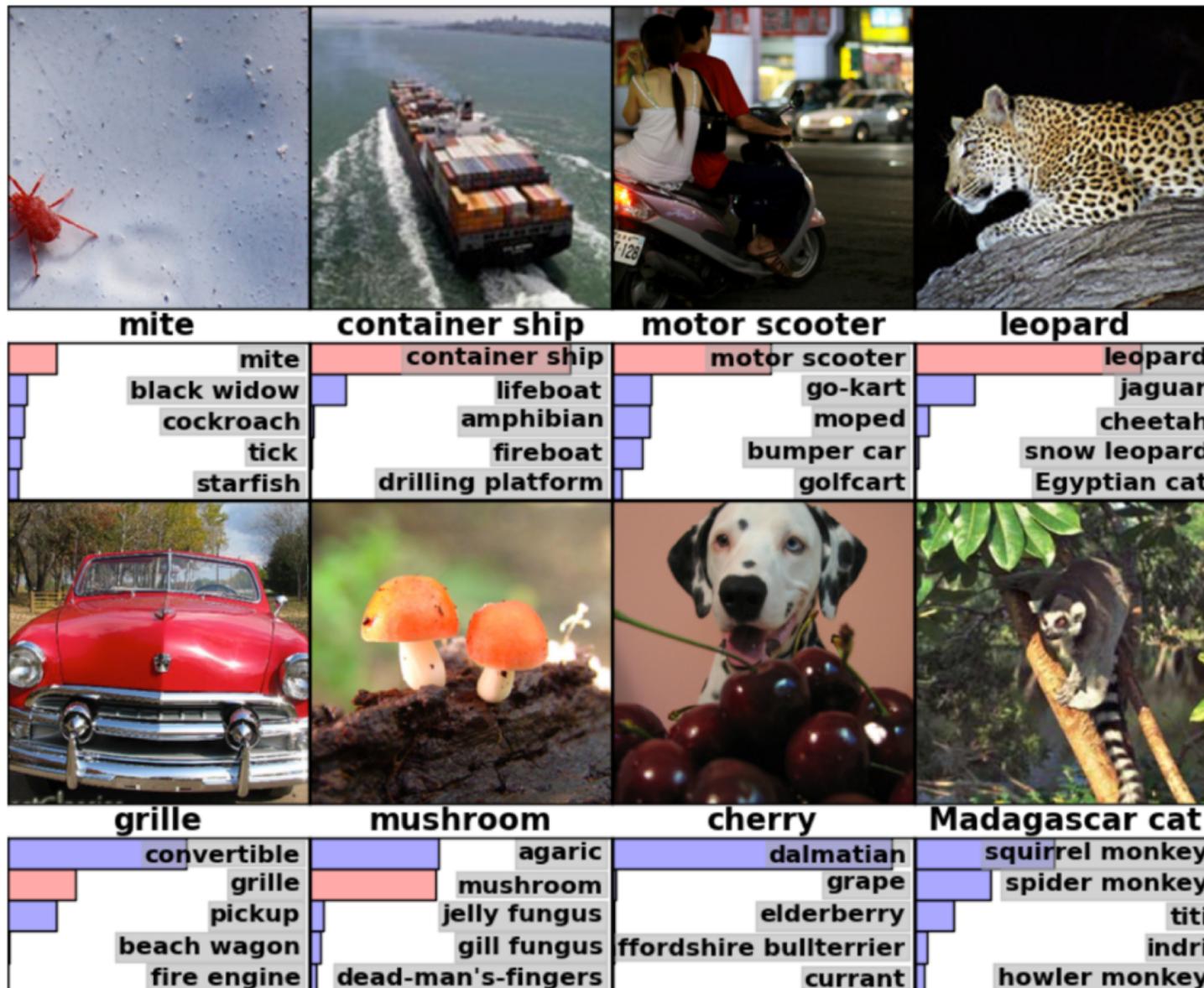
Deep Learning

- Generally refers to models with nested, layered non-linearities
- Common example:
 - Classical ANNs with multiple (many) hidden layers
 - Trained via SGD and variants
 - Some new algorithmic insights (e.g., dropout regularization) and extensions (Convnets, Resnets, RNNs, LSTMs, GRUs...)

Some deep learning success stories

- State of the art performance on some difficult classification tasks
 - Speech recognition (TIMIT)
 - Image recognition (MNIST, ImageNet)
 - Natural language processing (semantic word embeddings)
 - Speech translation
- A lot of recent work on sequential models
(Recurrent Neural Networks, LSTMs, GRUs, ...)

Example: Image Classification

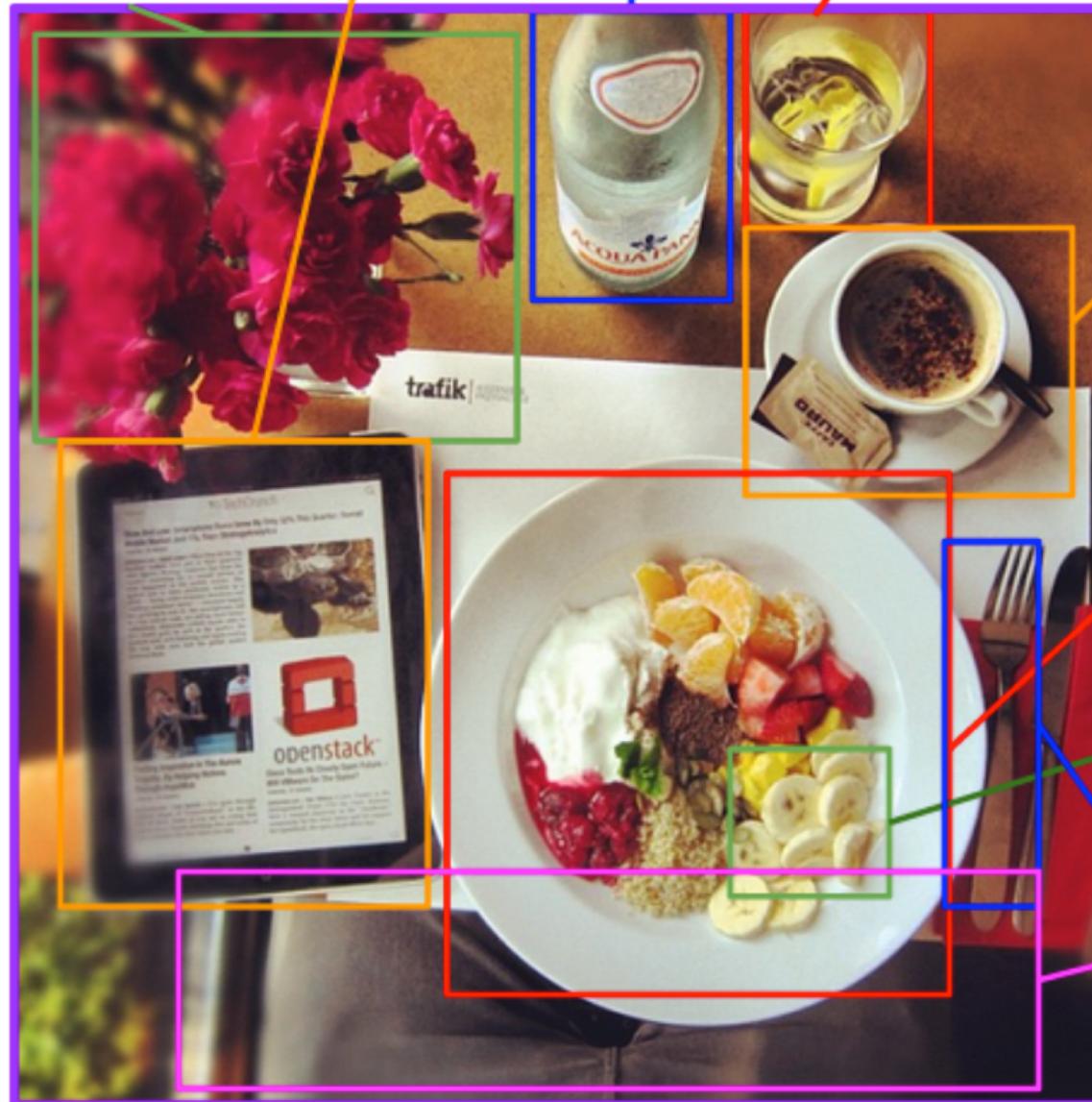


bouquet of red flowers

tablet

bottle of water

glass of water with ice and lemon



cup of coffee

dining table with breakfast items

plate of fruit

banana slices

fork

a person sitting at a table

Example: Image captioning

Y

A person riding a motorcycle on a dirt road.



X

A group of young people playing a game of frisbee.



Two dogs play in the grass.



A skateboarder does a trick on a ramp.



Two hockey players are fighting over the puck.



A little girl in a pink hat is blowing bubbles.

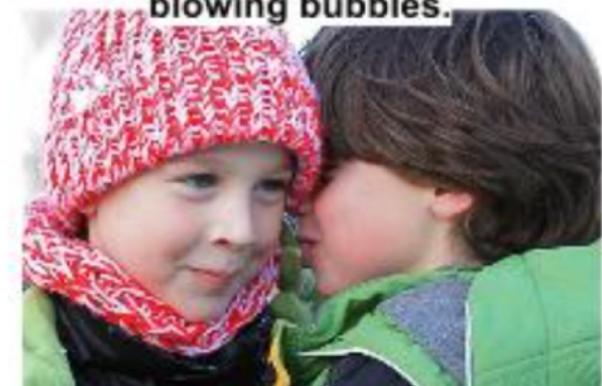
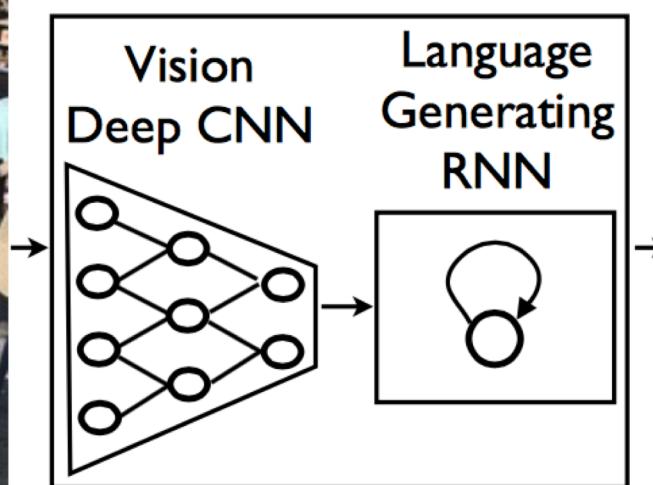


Image captioning



A group of people shopping at an outdoor market.

There are many vegetables at the fruit stand.

Example: Translation

The screenshot shows the Google Translate web interface. At the top left is the Google logo. On the right are icons for a grid, a bell, and a user profile. Below the logo, the word "Translate" is written in red. To its right are links for "Turn off instant translation" and a star icon. The main interface consists of two language selection bars. The left bar has "English" selected, followed by "Spanish", "French", "Detect language". The right bar has "English" selected, followed by "Spanish", "German". Between them is a double-headed arrow icon. To the right of the second bar is a blue "Translate" button. Below these bars are two text boxes. The left box contains the English sentence "Machine learning is getting more accurate" and has a character limit of 41/5000. The right box contains the German translation "Maschinelles Lernen wird immer genauer". Below each text box are small icons for audio, keyboard, and sharing, along with a pencil icon for editing.

X

Y

Some questions

- How can we compute the gradient?
- How should we initialize the weights?
- When should we terminate?
- How do we choose parameters (number of units/layers/activation functions/learning rate/ ...)?
- What about overfitting?

Computing the gradient

- In order to apply SGD, need to compute

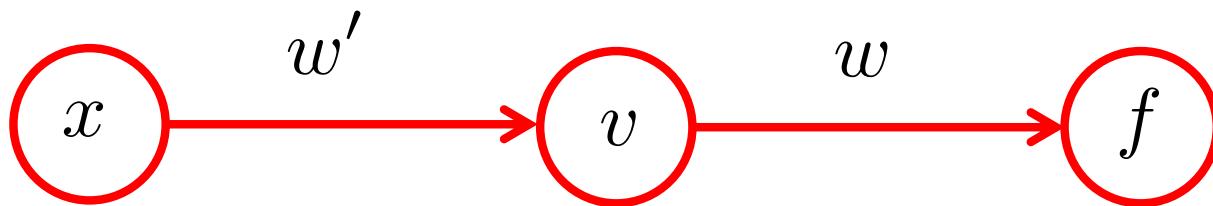
$$\nabla_{\mathbf{W}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x})$$

- I.e., for each weight between any two connected units i and j need

$$\frac{\partial}{\partial w_{i,j}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x})$$

Simple example

- ANN with 1 output, 1 hidden and 1 input unit $W = [w, w']$



Here, $f(x, \mathbf{W}) = w\varphi(w'x)$

$L(w', w) := L = \ell(f, y) = \ell_y(f)$

$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial w} = \ell'_y(f) \cdot v$

$\frac{\partial L}{\partial w'} = \underbrace{\frac{\partial L}{\partial f}}_S \underbrace{\frac{\partial f}{\partial v}}_S \underbrace{\frac{\partial v}{\partial w'}}_{\text{already computed}} = S \cdot w \cdot \varphi'(z) \cdot x$

$D = \{(x, y)\}$

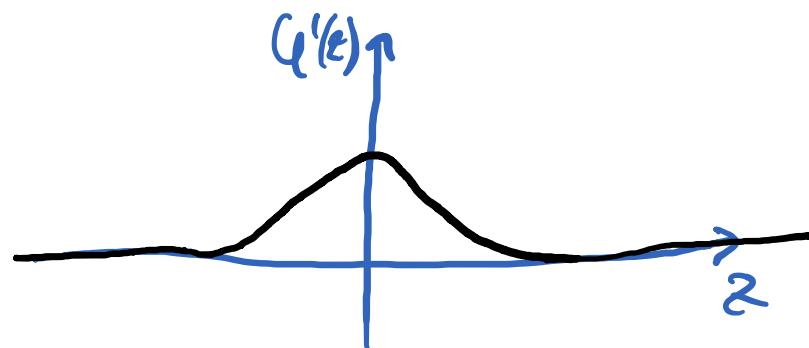
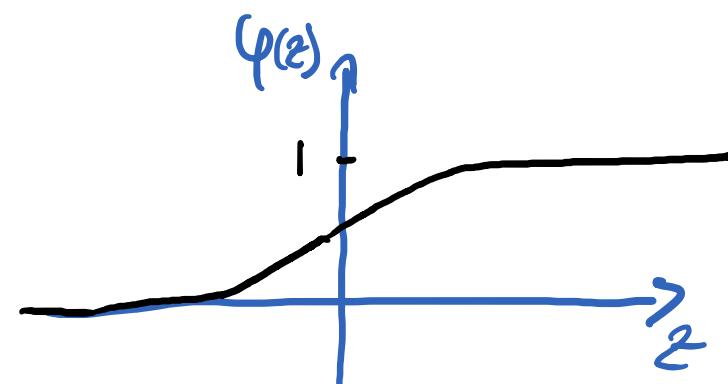
E.g.: $\ell_y(f) = (f - y)^2$
 $\Rightarrow \ell'_y(f) = 2(f - y)$

Computed by forward prop.

Derivatives of activation functions

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

$$\varphi'(z) = \frac{-1}{(1 + e^{-z})^2} \cdot e^{-z} \cdot (-1) = \underbrace{\frac{e^{-z}}{1 + e^{-z}}}_{1 - \varphi(z)} \cdot \underbrace{\frac{1}{1 + e^{-z}}}_{\varphi(z)} = \varphi(z)(1 - \varphi(z))$$



+ diff'able everywhere

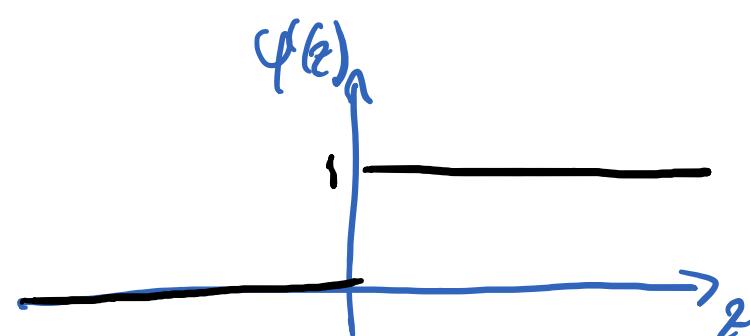
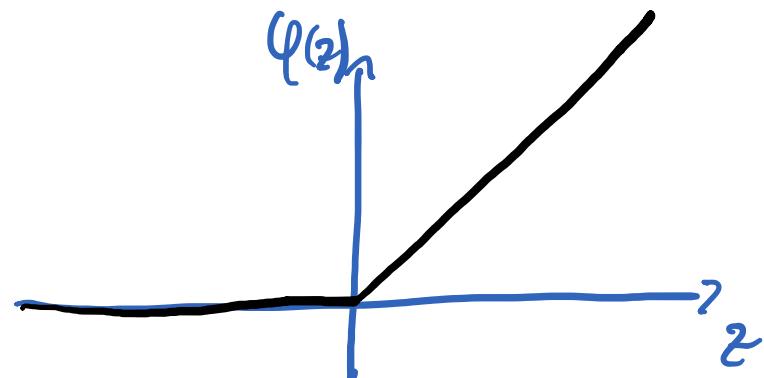
- $\varphi' \approx 0$ unless $z \approx 0$

\Rightarrow often lead to vanishing gradients for deep models

Derivatives of activation functions

- Rectified Linear Unit (ReLU): $\varphi(z) = \max(z, 0)$

$$\varphi^l(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise.} \end{cases}$$

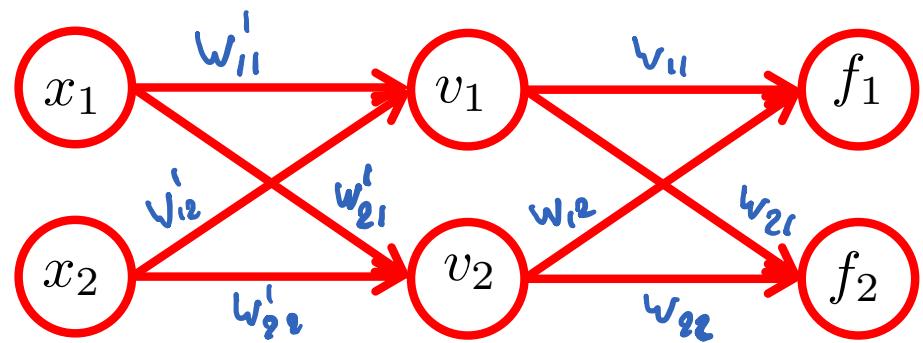


- not differentiable at 0
(in practice not a problem)

+ $\varphi' = 1$ for all $z > 0$

\Rightarrow helps with avoiding vanishing gradients

Slightly more complicated example



$$L = \sum_{i=1}^2 \ell_i \left(\sum_j w_{i,j} \varphi \left(\sum_k w'_{j,k} x_k \right) \right)$$

$$W = [(w_{ij})_{ij} , (w'_{ij})_{ij}]$$

$$\frac{\partial L}{\partial w_{ij}} = \underbrace{\frac{\partial L}{\partial f_i}}_{\delta_i} \frac{\partial f_i}{\partial w_{ij}} = \underbrace{l'(f_i)}_{\delta'_i} \cdot v_j$$

compute' by forward prop.

$$\frac{\partial L}{\partial w'_{jk}} = \underbrace{\sum_{i=1}^2 \frac{\partial L}{\partial f_i}}_{\delta_i} \underbrace{\frac{\partial f_i}{\partial v_j}}_{\delta'_i} \underbrace{\frac{\partial v_j}{\partial w'_{jk}}}_{\delta''_j} = \underbrace{\sum_{i=1}^2 \delta_i}_{\delta'_i} w_{ij} \cdot \varphi'(z_j) \cdot x_k$$

already computed

Backpropagation

- For each unit j on the output layer
 - Compute error signal $\delta_j = \ell'_j(f_j)$
 - For each unit i on layer L , $\frac{\partial}{\partial w_{j,i}} = \delta_j v_i$
- For each unit j on hidden layer $\ell = L - 1 : -1 : 1$
 - Compute error signal $\delta_j = \varphi'(z_j) \sum_{i \in \text{Layer}_{\ell+1}} w_{i,j} \delta_i$
 - For each unit i on layer $\ell - 1$, $\frac{\partial}{\partial w_{j,i}} = \delta_j v_i$

Backpropagation: Matrix form

- For the output layer

- Compute „error“ $\delta^{(L)} = \mathbf{l}'(\mathbf{f}) = [l'(f_1), \dots, l'(f_p)]$

- Gradient: $\nabla_{\mathbf{W}^{(L)}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \delta^{(L)} \mathbf{v}^{(L-1)T}$

- For each hidden layer $\ell = L - 1 : -1 : 1$

- Compute „error“ $\delta^{(\ell)} = \varphi'(\mathbf{z}^{(\ell)}) \odot (\mathbf{W}^{(\ell+1)T} \delta^{(\ell+1)})$

pointwise multiplication

- Gradient $\nabla_{\mathbf{W}^{(\ell)}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \delta^{(\ell)} \mathbf{v}^{(\ell-1)T}$