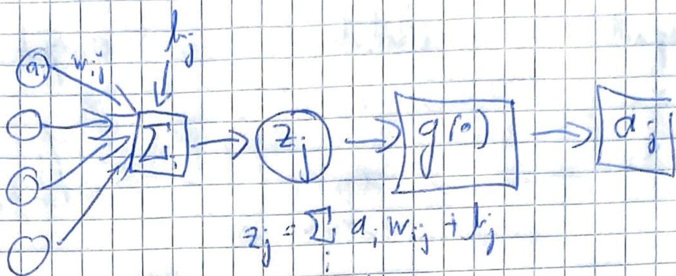
Deep Learning 1 (Structured Networks)



$$z_j = \sum_i a_i w_{ij} + b_j$$

$$a_j = g(z_j)$$

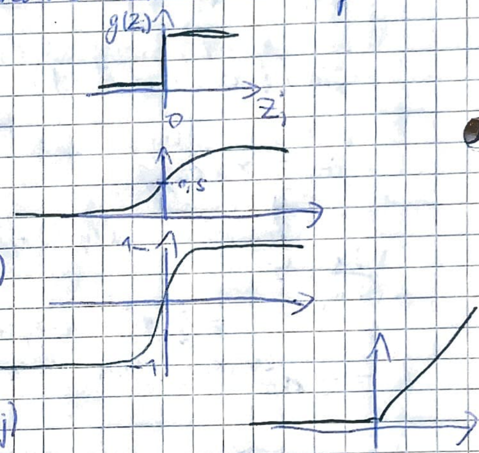
Nonlinearity through non-linear activation functions:

Threshold: $g(z_j) = 1(z_j \geq 0)$

Sigmoid: $g(z_j) = \frac{e^{z_j}}{1+e^{z_j}}$

Tanhypert. $g(z_j) = \tanh(z_j)$

ReLU $g(z_j) = \max(0, z_j)$



Learning:

Take partial derivative of objective ε with respect to w_{ij} to learn influence of w_{ij} on ε

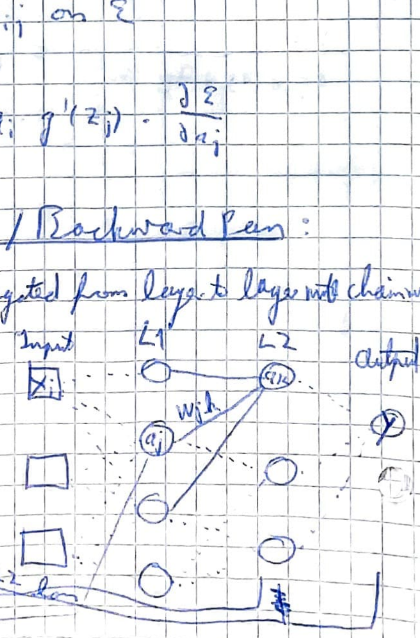
$$\frac{\partial \varepsilon}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial a_j}{\partial z_j} \frac{\partial \varepsilon}{\partial a_j} = a_i \, g'(z_j) \cdot \frac{\partial \varepsilon}{\partial a_j}$$

Backpropagation Algorithm / Backward Pass:

The gradient from the error can be propagated from layer to layer with chainrule

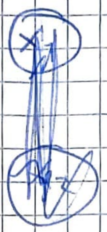
Implementation: Store intermediat $\delta_j, \delta_k, \dots$ results to save computation time!

$$\underbrace{\frac{\partial \varepsilon}{\partial a_j}}_{\delta_j} = \sum_k \underbrace{\frac{\partial a_k}{\partial a_j}}_{w_{jk} \, g'(z_k)} \underbrace{\frac{\partial \varepsilon}{\partial a_k}}_{\delta_k}$$



Contribution to loss function

Neural Networks with sufficiently many neurons can approximate any function $f(x)$ $x \in x_1, x_2, \ldots, x_d$



Disadvantages of Neural Networks:

- non-convex
- many hyperparameters (initialization, learning rate, ...)
- multiple layers can cause pathological curvature
  → optimizer might get stuck on large plateaus

Universality: CNN model recognizes most visual objects
Compactness: Representation finite dimensional at each layer
Convexity: Non-convex, but usually converges to good solutions

	Universal	Compact	Convex
Feature Engineering	✗	✓	✓
Kernels	✓	✗	✓
Neural Networks	✓	✓	✗

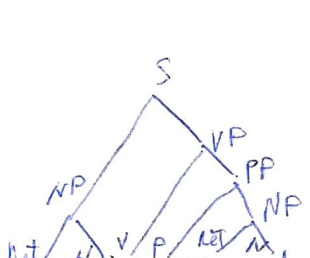
Convolutional Neural Networks

- consist of convolution and pooling layers
- backpropagation of errors
- CNNs are state-of-the-art (by wide margin) for image classification
- CNN is very resource expensive → pretrained model

Standard NN	Convolutional NN
- neurons and weights are scalars, $a_i, w_{ij} \in \mathbb{R}$	- neurons & weights are 2D arrays: $\vec{a}^{(i)} \in \mathbb{R}^{W \times H}$, $\vec{W}^{(ij)} \in \mathbb{R}^{W' \times H'}$
- mapping between two layers is a sum of products: $z_j = \Sigma_i w_{ij} a_i + b_j$	- mapping between two layers is sum of 2D cross-correlation $z_j = \Sigma_i W^{ij} \circ a^i + b^j$
	→ pooling layers to reduce spatial resolution and increase # neurons

Summary:

- NNs can learn compact representation of any task
- but more difficult optimization problem
- structured networks allow handling of real-world data (images, text, graphs)
- CNNs extract image representation by expanding semantic content while compressing spatial content
- GNNs generalize NNs to any graph-structured input (trees, lattices, ...)



Step 1: Cross-Correlation:

$$z_{t,t'} = [w * a]_{t,t'} = \sum_{s=0}^{2}\sum_{s'=0}^{2} w_{s,s'} \cdot a_{t+s,t'+s'}$$

Step 2: The Convolution Layer:

$$\vec{a}^{(j)} = g\left(\sum_i \vec{w}^{(i,j)} * \vec{a}^{(i)}\right)$$

Forward pass: $z_t = [w * a]_t = \sum_{s=-\infty}^{\infty} w_s \cdot a_{t+s}$

Backward pass: $\frac{\partial E}{\partial a_u} = \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot \frac{\partial z_t}{\partial a_u} = \sum_{t=-\infty}^{\infty}\sum_{s=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot w_s \cdot 1_{\{t+s=u\}}$

$$= \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot w_{u-t} = \left[\frac{\partial E}{\partial z} * w\right]_u$$

Parameter Gradient:

$$\frac{\partial E}{\partial w_u} = \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t}\frac{\partial z_t}{\partial w_u} = \sum_{t=-\infty}^{\infty}\sum_{s=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot a_{t+s} \cdot 1_{\{s=u\}} = \sum_{t=-\infty}^{\infty} \frac{\partial E}{\partial z_t} \cdot a_{u+t}$$

$$= \left[\frac{\partial E}{\partial z} * a\right]_u$$

Text Input: Vector Embedding of a Sentence.

1. One-hot encoding: Represent word with indicator function

$$e[w] = \begin{pmatrix} 1_{\{w='a'\}} \\ 1_{\{w='able'\}} \\ \vdots \\ 1_{w_t} \end{pmatrix} \in \mathbb{R}^{\#\text{words}}$$

2. k PCA embedding:

structured kernel measures similarity between words (co-occuring)

$$e[w] = U_{w,:d} \odot (\lambda_1^{0.5}, \dots, \lambda_d^{0.5}) \in \mathbb{R}^d$$

3. Learned Embedding: Start with random embedding $e[w] \in \mathbb{R}^d$ and learn embedding: $e[w] \leftarrow e[w] - \gamma \frac{\partial E}{\partial e[w]}$

Recursive Neural Networks: make use of parsing tree of sentence

Graph Neural Networks: - specialized for classifying graphs
- graph as input, propagation step at each layer along edges
- CNN special case of GNN where graph is 2D lattice