CS4400
DEEP REINFORCEMENT LEARNING

Lecture 4: Generalization

Wendelin Böhmer

<j.w.bohmer@tudelft.nl>

TU Delft

28th of November 2023

Content of this lecture

4.1

Generalization

Why Deep Learning?

TUDelft

Function approximation

Theorem (Universal function approximation (Funahashi, 1989))

The shallow net $f(\boldsymbol{x}) = \boldsymbol{c}^\top \sigma\big(\mathbf{A}\boldsymbol{x} + \boldsymbol{b}\big)$, *where* $\sigma : \mathbb{R} \to \mathbb{R}$ *is non-constant, bounded, monotonously increasing and continuous, can approximate any continuous function* $y : \mathcal{X} \to \mathbb{R}, \mathcal{X} \subset \mathbb{R}^m$, *arbitrarily well, i.e.,* $\forall \epsilon > 0 :$

$$\exists n < \infty \in \mathbb{N},\ \exists \mathbf{A} \in \mathbb{R}^{n \times m},\ \exists \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^n : \qquad \sup_{\boldsymbol{x} \in \mathcal{X}} \Big| f(\boldsymbol{x}) - y(\boldsymbol{x}) \Big| \ \leq\ \epsilon \,.$$

- So why “deep” learning?

Cybenko (1989) developed a similar theorem in parallel; Sonoda and Murata (2017) prove this for unbounded functions σ like ReLU

4.1 Function approximation

Theorem (Universal function approximation (Funahashi, 1989))

The shallow net $f(\boldsymbol{x}) = \boldsymbol{c}^\top \sigma\big(\mathbf{A}\boldsymbol{x} + \boldsymbol{b}\big)$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is non-constant, bounded, monotonously increasing and continuous, can approximate any continuous function $y : \mathcal{X} \to \mathbb{R}, \mathcal{X} \subset \mathbb{R}^m$, arbitrarily well, i.e., $\forall \epsilon > 0$:

$$\exists n < \infty \in \mathbb{N},\ \exists \mathbf{A} \in \mathbb{R}^{n \times m},\ \exists \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^n : \qquad \sup_{\boldsymbol{x} \in \mathcal{X}} \Big| f(\boldsymbol{x}) - y(\boldsymbol{x}) \Big| \;\leq\; \epsilon \,.$$

- So why "deep" learning?
- More layers help in practice
  - But why?

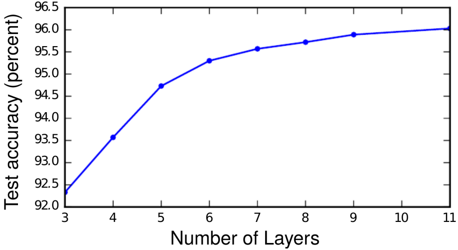


figure from classification of multi-digit numbers based on images of addresses (Goodfellow et al., 2014, 2016)

4.1 Bottom-up: representations

- Higher CNN layer encode more complex features
  - based on increasing image patches
  - similar to findings in neuroscience
- Parameter sharing increases training data
  - same kernel
  - local computation
  - translation invariant
- CNN provide better generalization to unseen inputs than linear layers
  - but why?

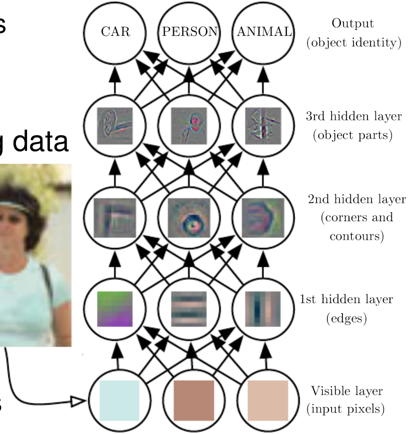
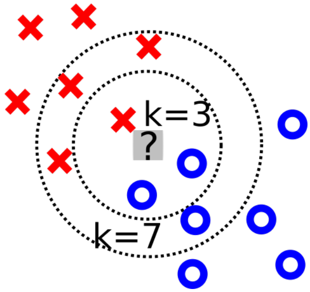


Figure from Goodfellow et al. (2016), based on data from Zeiler and Fergus (2014)

4.1 Top-down: generalization

- Generalization based on **similarity** to seen examples
  - e.g. k-nearest-neighbors classification/regression
  - inconsistent neighbors yield probability distribution
- What about neural networks?

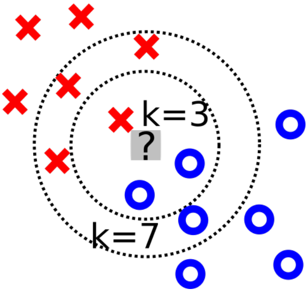


4.1 Top-down: generalization

- Generalization based on **similarity** to seen examples
  - e.g. k-nearest-neighbors classification/regression
  - inconsistent neighbors yield probability distribution
- What about neural networks?
- Cauchy-Schwarz inequality

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \;\leq\; \|\boldsymbol{a}\|_2 \, \|\boldsymbol{x} - \boldsymbol{y}\|_2$$

  - for $f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x} + b$
  - similar output for small $\|\boldsymbol{x} - \boldsymbol{y}\|_2$
  - smaller $\|\boldsymbol{a}\|_2$ “generalize” farther



(4.1) Top-down: generalization

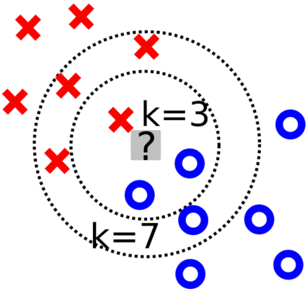
- Generalization based on **similarity** to seen examples
  - e.g. k-nearest-neighbors classification/regression
  - inconsistent neighbors yield probability distribution
- What about neural networks?
- Cauchy-Schwarz inequality

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \;\leq\; \|\boldsymbol{a}\|_2 \, \|\boldsymbol{x} - \boldsymbol{y}\|_2$$

  - for $f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{x} + b$
  - similar output for small $\|\boldsymbol{x} - \boldsymbol{y}\|_2$
  - smaller $\|\boldsymbol{a}\|_2$ "generalize" farther
- Lipschitz continuity, $\exists \ell < \infty \in \mathbb{R} : \forall \boldsymbol{x}, \boldsymbol{y}$:

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \;\leq\; \ell \, \|\boldsymbol{x} - \boldsymbol{y}\|_2$$

  - regularization lowers ℓ



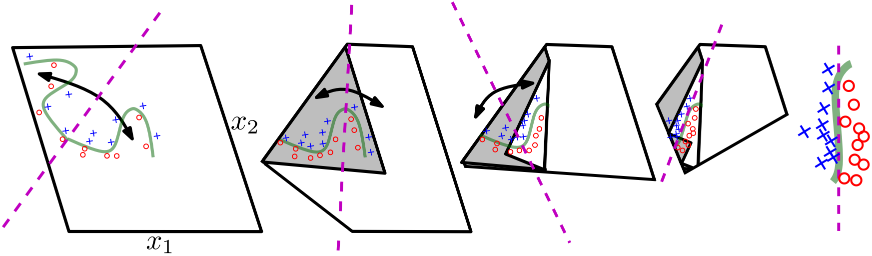
4.1 Invariances

- f is invariant to Φ iff $\quad f(\boldsymbol{x}) = f(\phi(\boldsymbol{x})),\ \forall \phi \in \Phi,\ \forall \boldsymbol{x}$

Figure modified from Montúfar et al. (2014)

4.1 Invariances

- f is invariant to Φ iff $\quad f(\boldsymbol{x}) = f(\phi(\boldsymbol{x})),\ \forall \phi \in \Phi,\ \forall \boldsymbol{x}$
- Non-linear transfer-functions induce invariances
  - e.g. ReLU zeros out half of input space
  - e.g. absolute value “folds” input space

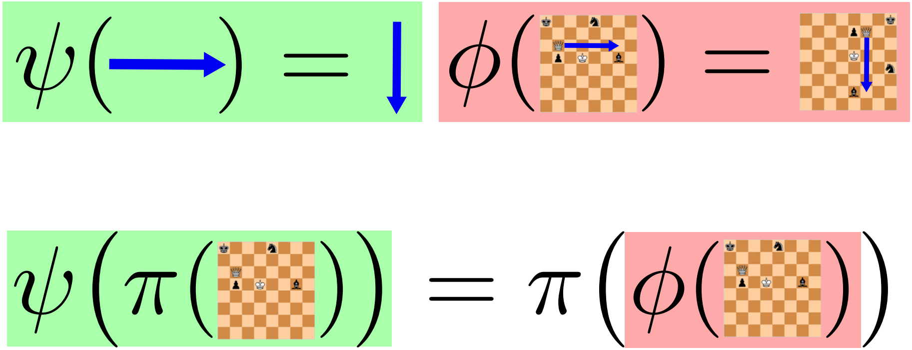


- Previous layer defines similarity $\|\boldsymbol{x} - \boldsymbol{y}\|_2$
  - deep networks increasingly generalize

Figure modified from Montúfar et al. (2014)

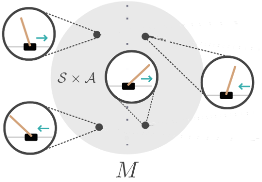
4.1 Equivariant neural networks

- f is equivariant to Φ iff $\quad \psi(f(\boldsymbol{x})) = f(\phi(\boldsymbol{x})),\ \forall(\phi,\psi) \in \Phi,\ \forall \boldsymbol{x}$



4.1 Equivariant neural networks

- f is equivariant to Φ iff $\quad \psi(f(\boldsymbol{x})) = f(\phi(\boldsymbol{x})),\ \forall(\phi,\psi)\in\Phi,\ \forall\boldsymbol{x}$
- `torch.nn.Linear` layers are most general
  - can learn "anything" (universal approximator)
  - no generalization out-of-distribution
  - e.g. $\ln\pi(a|\boldsymbol{s}) = \left(\mathbf{W}\left[\begin{smallmatrix}1\\ \boldsymbol{s}\end{smallmatrix}\right]\right)_a, \quad \boldsymbol{s}=[x,\theta,\dot{x},\dot{\theta}]^\top\in\mathbb{R}^4, \quad a\in\{L,R\}$

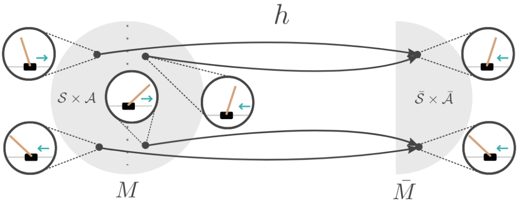


homomorphic neural networks for RL and image modified from van der Pol et al. (2020)

4.1 Equivariant neural networks

- f is equivariant to Φ iff $\quad \psi(f(\boldsymbol{x})) = f(\phi(\boldsymbol{x})),\ \forall(\phi,\psi)\in\Phi,\ \forall \boldsymbol{x}$
- `torch.nn.Linear` layers are most general
  - can learn "anything" (universal approximator)
  - no generalization out-of-distribution
  - e.g. $\ln \pi(a|\boldsymbol{s}) = \left(\mathbf{W}\left[\begin{smallmatrix}1\\ \boldsymbol{s}\end{smallmatrix}\right]\right)_a, \quad \boldsymbol{s} = [x, \theta, \dot{x}, \dot{\theta}]^\top \in \mathbb{R}^4, \quad a \in \{L, R\}$
- Equivariance symmetries are constraints on $\mathbf{W} \in \mathbb{R}^{2\times 5}$
  - e.g. $\ln \pi(L|\boldsymbol{s}) \stackrel{!}{=} \ln \pi(R|-\boldsymbol{s}), \quad \left[\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right]\mathbf{W}\left[\begin{smallmatrix}1\\ \boldsymbol{s}\end{smallmatrix}\right] \stackrel{!}{=} \mathbf{W}\left[\begin{smallmatrix}1 & \mathbf{0}^\top\\ \mathbf{0} & -\mathbf{I}\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\ \boldsymbol{s}\end{smallmatrix}\right]$

$$\mathbf{W}_h := \tfrac{1}{2}\mathbf{W} + \tfrac{1}{2}\left[\begin{smallmatrix}0 & 1\\ 1 & 0\end{smallmatrix}\right]^{-1}\mathbf{W}\left[\begin{smallmatrix}1 & \mathbf{0}^\top\\ \mathbf{0} & -\mathbf{I}\end{smallmatrix}\right] \equiv \left[\begin{smallmatrix}b & \boldsymbol{w}^\top\\ b & -\boldsymbol{w}^\top\end{smallmatrix}\right], \quad \boldsymbol{w}\in\mathbb{R}^4, \quad b \in \mathbb{R}$$



homomorphic neural networks for RL and image modified from van der Pol et al. (2020)

4.1 Equivariant neural networks

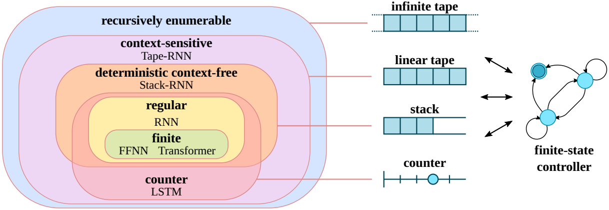
- f is equivariant to Φ iff $\quad \psi(f(\boldsymbol{x})) = f(\phi(\boldsymbol{x})),\ \forall(\phi,\psi)\in\Phi,\ \forall\boldsymbol{x}$
- `torch.nn.Linear` layers are most general
  - can learn “anything” (universal approximator)
  - no generalization out-of-distribution
  - e.g. $\ln\pi(a|\boldsymbol{s}) = \left(\mathbf{W}\left[\begin{smallmatrix}1\\ \boldsymbol{s}\end{smallmatrix}\right]\right)_a, \quad \boldsymbol{s} = [x,\theta,\dot{x},\dot{\theta}]^\top \in \mathbb{R}^4, \quad a\in\{L,R\}$
- Equivariance symmetries are constraints on $\mathbf{W}\in\mathbb{R}^{2\times 5}$
  - e.g. $\ln\pi(L|\boldsymbol{s}) \overset{!}{=} \ln\pi(R|-\boldsymbol{s}), \quad \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]\mathbf{W}\left[\begin{smallmatrix}1\\ \boldsymbol{s}\end{smallmatrix}\right] \overset{!}{=} \mathbf{W}\left[\begin{smallmatrix}1&\mathbf{0}^\top\\ \mathbf{0}&-\mathbf{I}\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\ \boldsymbol{s}\end{smallmatrix}\right]$

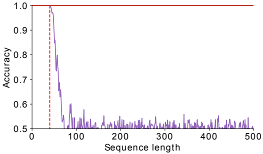
$$\mathbf{W}_h \;:=\; \tfrac{1}{2}\mathbf{W} + \tfrac{1}{2}\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]^{-1}\mathbf{W}\left[\begin{smallmatrix}1&\mathbf{0}^\top\\ \mathbf{0}&-\mathbf{I}\end{smallmatrix}\right] \;\equiv\; \left[\begin{smallmatrix}b&\boldsymbol{w}^\top\\ b&-\boldsymbol{w}^\top\end{smallmatrix}\right], \quad \boldsymbol{w}\in\mathbb{R}^4, \quad b\in\mathbb{R}$$

- Neural architecture determines **inductive bias**
  - parameter sharing = constraints
  - constraints = equivariances
  - equivariances = generalization
- Choose the *right* architecture for the *right* generalization

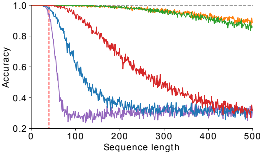
Chomsky hierarchy

- Test generalization of NN with character-sequence generation
- Train $|w| \leq 40$
- Test $|w| > 40$
- All learn $|w| \leq 40$
- But generalization limited by NN-arch!

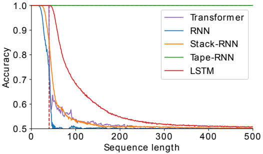




parity check
(regular language)



modular arithmetic
(deterministic context-free)



duplicate string
(context-sensitive)

see Deletang et al. (2023) for details

4.1 Summary

- Shallow networks can learn anything
- Deep networks encourage generalization
- Generalization comes from equi-/invariances
- Equi-/invariances are determined by architecture
- Architecture restricts network generalization

Learning Objectives

LO4.1: Explain how generalization and regularization/depth are related

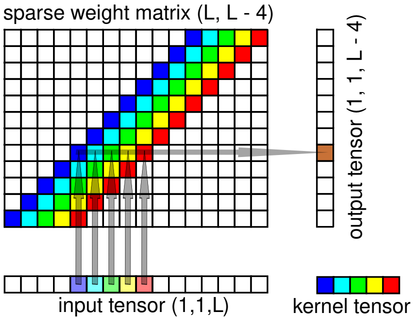
4.2

Generalization

Common Neural Layers

Simple convolutional layers

- Convolution `torch.nn.Conv1d(in_channels,out_channels, kernel_size,stride,padding)`
  - e.g. input `(N,in_channels,L)` with `kernel_size=(kx,)` output `(N, out_channels, L-kx+1)`



CNN by LeCun et al. (1989)

Question: equivalent linear layer

- Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ denote a time series of n (d-dimensional) samples
- Let $\mathbf{K} \in \mathbb{R}^{b \times d \times n_K}$ denote a given kernel
- Let $g(\mathbf{X}, \mathbf{K})$ denote a one-dimensional *convolutional layer*:

$$g(\mathbf{X}, \mathbf{K})_{k,m} \;:=\; \sum_{l=1}^{n_K} \sum_{p=1}^{d} K_{k,p,l} \, X_{p,m+l-1} \,, \qquad \begin{matrix} 1 \leq k \leq b \\ 1 \leq m \leq n - n_K + 1 \end{matrix}$$

- Define the equivalent *linear function* $f : \mathbb{R}^{\mathcal{J}} \to \mathbb{R}^{\mathcal{I}}$!
  - constructing $\boldsymbol{z}$ from $\mathbf{X}$
  - define the index sets $\mathcal{J}$ and $\mathcal{I}$
  - construct $\boldsymbol{\Theta} \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ from g's kernel $\mathbf{K}$

$$f(\boldsymbol{z})_i \;:=\; \sum_{j \in \mathcal{J}} \Theta_{i,j} \, z_j \,, \qquad \forall \boldsymbol{z} \in \mathbb{R}^{\mathcal{J}} \,, \quad \forall i \in \mathcal{I} \,,$$

assignment sheet 2

Question: equivalent linear layer

- Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ denote a time series of n (d-dimensional) samples
- Let $\mathbf{K} \in \mathbb{R}^{b \times d \times n_K}$ denote a given kernel
- Let $g(\mathbf{X}, \mathbf{K})$ denote a one-dimensional *convolutional layer*:

$$g(\mathbf{X}, \mathbf{K})_{k,m} := \sum_{l=1}^{n_K} \sum_{p=1}^{d} K_{k,p,l} \, X_{p,m+l-1} \,, \qquad \begin{smallmatrix} 1 \leq k \leq b \\ 1 \leq m \leq n - n_K + 1 \end{smallmatrix}$$

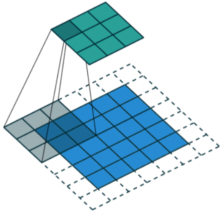
- $\mathbf{X} \equiv \boldsymbol{z} \quad \Rightarrow \quad z_{(u,v)} := X_{u,v} \,, \quad \mathcal{J} := \{(u,v) \,|\, 1 \leq u \leq d, 1 \leq v \leq n\}$
- $g(\mathbf{X}, \mathbf{K}) \equiv f(\boldsymbol{z}) \quad \Rightarrow \quad \mathcal{I} := \{(k,m) \,|\, 1 \leq k \leq b, 1 \leq m \leq n - n_K + 1\}$

$$\begin{aligned} g(\mathbf{X}, \mathbf{K})_{k,m} &= \sum_{v=m}^{m+n_K-1} \sum_{u=1}^{d} K_{k,u,v-m+1} X_{u,v} \\ &= \underbrace{\sum_{v=1}^{n} \sum_{u=1}^{d}}_{\sum\limits_{j:=(u,v) \in \mathcal{J}}} \underbrace{\underbrace{\delta(\, m \leq v < m+n_K \,) \, K_{k,u,v-m+1}}_{\Theta_{(k,m),(u,v)}}}_{\Theta_{i,j}} \underbrace{\underbrace{X_{u,v}}_{z_{(u,v)}}}_{z_j} = \underbrace{f(\boldsymbol{z})_{(k,m)}}_{f_i(\boldsymbol{z})} . \end{aligned}$$

assignment sheet 2

4.2 Higher dimensional convolutional layers

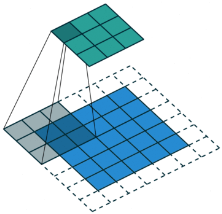
- Convolution `torch.nn.Conv`K`d(in_channels,out_channels,` `kernel_size,stride,padding)`
  - kernel-shape $K \in \{1, 2, 3\}$
  - e.g. input `(N,in_channels,W,H)` with `kernel_size=(kx,ky)` output `(N, out_channels, W-kx+1, H-ky+1)`
  - `stride` jumps pixels, `padding` allows constant output shape

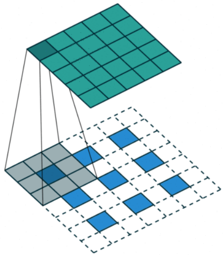


CNN by LeCun et al. (1989), images: github.com/vdumoulin/conv_arithmetic

4.2 Higher dimensional convolutional layers

- Convolution `torch.nn.Conv`K`d(in_channels,out_channels,kernel_size,stride,padding)`
  - kernel-shape $K \in \{1, 2, 3\}$
  - e.g. input `(N,in_channels,W,H)` with `kernel_size=(kx,ky)`
    output `(N, out_channels, W-kx+1, H-ky+1)`
  - `stride` jumps pixels, `padding` allows constant output shape
- Deconvolution `torch.nn.ConvTranspose`K`d`

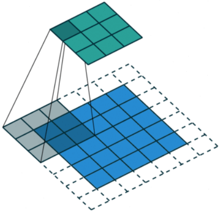




CNN by LeCun et al. (1989), deconvolution by Zeiler et al. (2010), images: github.com/vdumoulin/conv_arithmetic

(4.2) Pooling layers

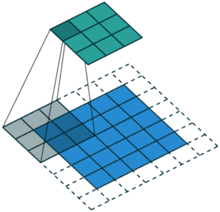
- Pooling `torch.nn.`X`Pool`K`d(kernel_size,stride,padding)`
  - kernel-shape $K \in \{1, 2, 3\}$, aggregation function $X \in \{\texttt{Max}, \texttt{Avg}\}$
  - `torch.nn.Adaptive`X`Pool`K`d(output_size)`

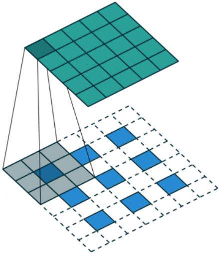


MaxPooling introduced by Zhou and Chellappa (1988), image source: github.com/vdumoulin/conv_arithmetic

Pooling layers

- Pooling `torch.nn.`X`Pool`K`d(kernel_size,stride,padding)`
  - kernel-shape $K \in \{1, 2, 3\}$, aggregation function $X \in \{$`Max,Avg`$\}$
  - `torch.nn.Adaptive`X`Pool`K`d(output_size)`
  - `MaxPool`K`d` returns indices when `return_indices=True`
- Unpooling `torch.nn.`X`Unpool`K`d(kernel_size,stride,padding)`
  - indices input to `MaxUnpool`K`d`

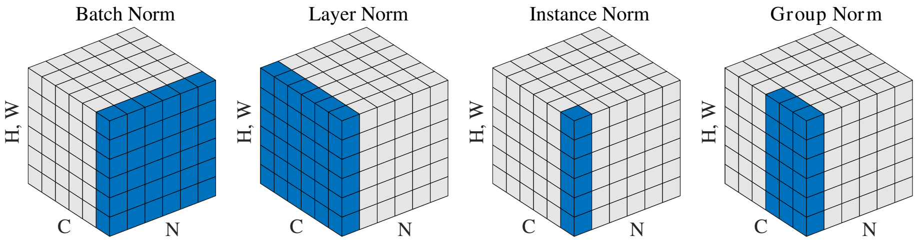




MaxPooling introduced by Zhou and Chellappa (1988), image source: `github.com/vdumoulin/conv_arithmetic`

4.2 Normalization layers

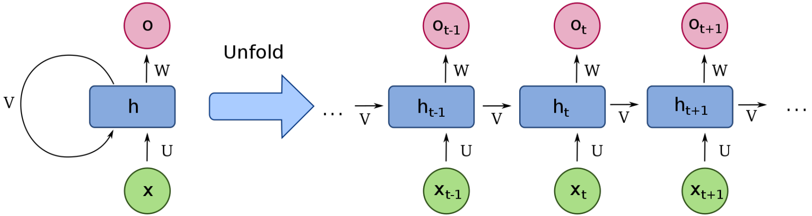
- Normalization of batch n, sample/pixel h and feature/channel c:
  - $x'_{nhc} := \frac{x_{nhc} - \mu_{nhc}}{\sqrt{\sigma^2_{nhc} + \epsilon}} \eta_c + \delta_c$
  - η_c and δ_c are learnable parameters
  - different layers average over different dimensions
  - `torch.nn.{BatchNormKd, LayerNorm, InstanceNormKd, GroupNorm}`



blue sub-tensors have same mean/variance, image from group norm paper (Wu and He, 2020)
batch norm (Ioffe and Szegedy, 2015), layer norm (Ba et al., 2016) and instance norm (Ulyanov et al., 2016)

4.2 Recurrent network layers

- `torch.nn.RNN(input_size,hidden_size)`
  - input `(L,N,input_size)` for `N` sequences of length `L`
  - `bidirectional=True`: additional end-to-beginning pass



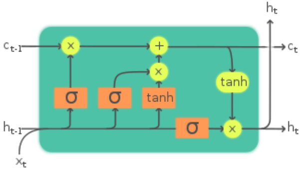
assignment sheet 2

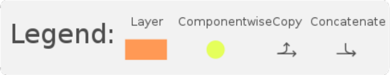
image source wikipedia.org

Recurrent network layers

- `torch.nn.RNN(input_size,hidden_size)`
  - input `(L,N,input_size)` for `N` sequences of length `L`
  - `bidirectional=True`: additional end-to-beginning pass
- `torch.nn.LSTM(input_size,hidden_size)`
  - solves vanishing gradients for long `L`
  - by introducing memory cell c_t
  - reading/writing guarded by gates $g_t^{\{i,o,f\}}$

$$
\begin{aligned}
h_t &= g_t^o \cdot \sigma(c_t) \\
c_t &= c_{t-1} \cdot g_t^f + g_t^i \cdot z_t \\
z_t &= \tanh(\mathtt{Linear}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
g_t^{\{i,o,f\}} &= \sigma\big(\mathtt{Linear}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])\big)
\end{aligned}
$$





- `torch.nn.GRU(input_size,hidden_size)`

LSTM (Hochreiter and Schmidhuber, 1997), GRU (Chung et al., 2014), image source `wikipedia.org`

4.2 Example architectures

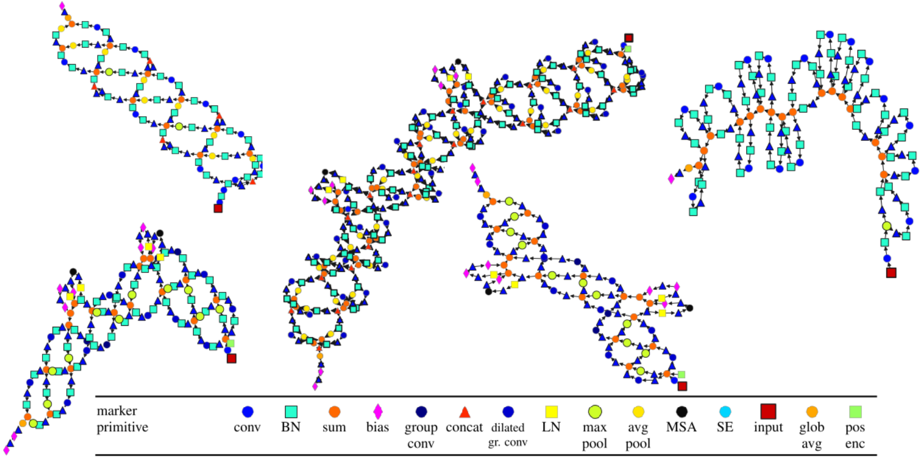


image modified from Knyazev et al. (2021)

4.2 Summary

- Deep neural nets have many specialized modules
  - CNN take images as input
  - Pooling layers reduce CNN sizes
  - RNN/LSTM take time series as input
  - Normalization layers can stabilize learning
- CNN/RNN are equivalent to restricted linear layers

Learning Objectives

LO4.2: Explain CNN, Pooling, Normalization and RNN

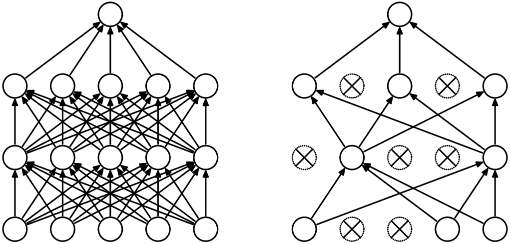
4.3

Generalization

Advanced Neural Layers

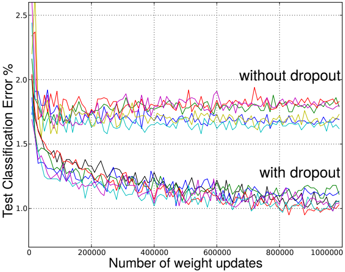
4.3 Dropout layers

- L_2 regularization: `weight_decay` parameter of optimizer
- Dropout regularization by stochastically "dropping" outputs
  - no dropping during prediction
  - network becomes robust to omission
  - less capacity $\Rightarrow$ smoother output



standard neural net

after applying dropout



Figures and results for MNIST classification from dropout paper (Srivastava et al., 2014)

4.3 Dropout layers

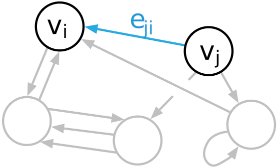
- L_2 regularization: `weight_decay` parameter of optimizer
- Dropout regularization by stochastically "dropping" outputs
  - no dropping during prediction
  - network becomes robust to omission
  - less capacity $\Rightarrow$ smoother output
- `torch.nn.Dropout` drops random outputs
  - i.i.d. with a given probability p
- `torch.nn.Dropout`K`d` drops random feature channels
  - for $K \in \{1, 2, 3\}$ dimensional data

Figures and results for MNIST classification from dropout paper (Srivastava et al., 2014)

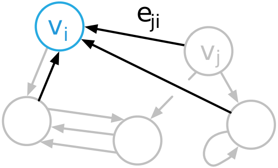
Graph neural networks (GNN)

- Given a graph $\mathcal{G} := \langle \mathcal{V}, \mathcal{E} \rangle$ and neural networks f^e, f^v:
  - nodes have annotation $\boldsymbol{v}_i \in \mathbb{R}^{m_v}$, edges $\boldsymbol{e}_{ji} \in \mathbb{R}^{m_e}$
  - learn functions f^e, f^v that update annotations

1. update edges $\boldsymbol{e}_{ji} \leftarrow f^e(\boldsymbol{e}_{ji}, \boldsymbol{v}_j, \boldsymbol{v}_i)\,, \quad \forall (j,i) \in \mathcal{E}$
2. aggregate edges $\bar{\boldsymbol{e}}_i \leftarrow \sum_{(j,i)\in\mathcal{E}} \boldsymbol{e}_{ji}\,, \quad \forall i \in \mathcal{V}$
3. update nodes $\boldsymbol{v}_i \leftarrow f^v(\boldsymbol{v}_i, \bar{\boldsymbol{e}}_i)\,, \quad \forall i \in \mathcal{V}$



(a) Edge update



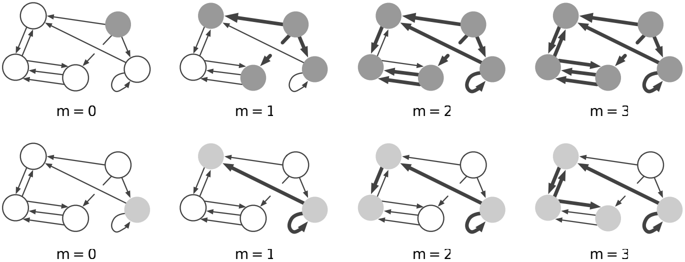
(b) Node update

images and GNN from Battaglia et al. (2018), applied in RL by Wang et al. (2018) and Huang et al. (2020)

4.3 Graph neural networks (GNN)

- Given a graph $\mathcal{G} := \langle \mathcal{V}, \mathcal{E} \rangle$ and neural networks f^e, f^v:
  - nodes have annotation $\boldsymbol{v}_i \in \mathbb{R}^{m_v}$, edges $\boldsymbol{e}_{ji} \in \mathbb{R}^{m_e}$
  - learn functions f^e, f^v that update annotations

1. update edges $\boldsymbol{e}_{ji} \leftarrow f^e(\boldsymbol{e}_{ji}, \boldsymbol{v}_j, \boldsymbol{v}_i)\,, \quad \forall (j,i) \in \mathcal{E}$
2. aggregate edges $\bar{\boldsymbol{e}}_i \leftarrow \sum_{(j,i)\in\mathcal{E}} \boldsymbol{e}_{ji}\,, \quad \forall i \in \mathcal{V}$
3. update nodes $\boldsymbol{v}_i \leftarrow f^v(\boldsymbol{v}_i, \bar{\boldsymbol{e}}_i)\,, \quad \forall i \in \mathcal{V}$



images and GNN from Battaglia et al. (2018), applied in RL by Wang et al. (2018) and Huang et al. (2020)

4.3 Graph convolutional networks (GCN)

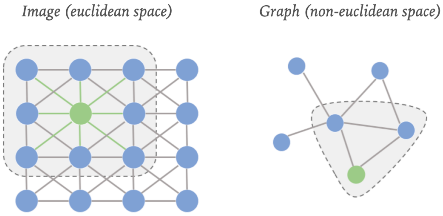
- GNN reformulated: $\mathbf{V}' \leftarrow \sigma\big(\mathbf{W}\mathbf{V}\mathbf{B} + \mathbf{V}\hat{\mathbf{B}}\big)$
  - assume no edge features (otherwise complicated)
  - $\mathbf{W} \in \mathbb{R}^{n \times n}$, $W_{ij} = 0$ iff $(j, i) \notin \mathcal{E}$
  - $\mathbf{V} := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]^\top \in \mathbb{R}^{n \times d}$
  - linear f^e and linear-ReLu f^v

assignment sheet 2

4.3 Graph convolutional networks (GCN)

- GNN reformulated: $\mathbf{V}' \leftarrow \sigma\big(\mathbf{W}\mathbf{V}\mathbf{B} + \mathbf{V}\hat{\mathbf{B}}\big)$
  - assume no edge features (otherwise complicated)
  - $\mathbf{W} \in \mathbb{R}^{n\times n}$, $W_{ij} = 0$ iff $(j,i) \notin \mathcal{E}$
  - $\mathbf{V} := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]^\top \in \mathbb{R}^{n\times d}$
  - linear f^e and linear-ReLu f^v
- Graph convolutional networks
  - $W_{ij} \in \big\{0, |\{(j,i) \in \mathcal{E}\}|^{-1}\big\}$
  - $\hat{\mathbf{B}} = \mathbf{0}\,, \quad (i,i) \in \mathcal{E}, \forall i \in \mathcal{V}$

  $\Rightarrow$ $\mathcal{E}$ determines inductive bias

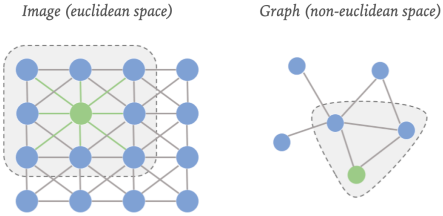


assignment sheet 2

images and GCN from Kipf and Welling (2017)

Graph convolutional networks (GCN)

- GNN reformulated: $\mathbf{V}' \leftarrow \sigma\big(\mathbf{WVB} + \mathbf{V}\hat{\mathbf{B}}\big)$
  - assume no edge features (otherwise complicated)
  - $\mathbf{W} \in \mathbb{R}^{n\times n}$, $W_{ij} = 0$ iff $(j,i) \notin \mathcal{E}$
  - $\mathbf{V} := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]^\top \in \mathbb{R}^{n\times d}$
  - linear f^e and linear-ReLu f^v
- Graph convolutional networks
  - $W_{ij} \in \big\{0, |\{(j,i) \in \mathcal{E}\}|^{-1}\big\}$
  - $\hat{\mathbf{B}} = \mathbf{0}\,, \quad (i,i) \in \mathcal{E}, \forall i \in \mathcal{V}$
  - $\Rightarrow$ $\mathcal{E}$ determines inductive bias



- Relational GCN: K topologies $\mathcal{E}^k$ propagate *different* messages
  - CNN are R-GCN: the $\mathcal{E}^k$ are pixels in kernels

$$\mathbf{V}' \leftarrow \sigma\Big(\sum_{k=1}^{K} \mathbf{W}^k \mathbf{V} \mathbf{B}^k\Big)$$

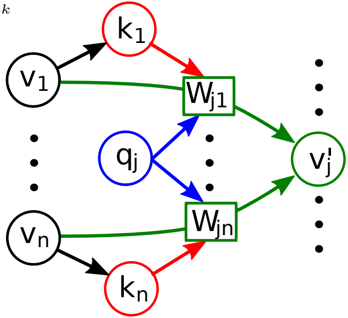
assignment sheet 2

R-GCN by Schlichtkrull et al. (2018), applied to RL by Jiang et al. (2021),

4.3 Multi-headed attention layers (MHA)

- What if we don't know the best topology?
  - every input node has key $\boldsymbol{k}_i = \mathbf{A}\boldsymbol{v}_i \in \mathbb{R}^{m_k}$
  - every output has query vector $\boldsymbol{q}_j \in \mathbb{R}^{m_k}$
  - topology matrix $W_{ji} \propto \exp(\boldsymbol{q}_j^\top \boldsymbol{k}_i)$
- Multi-headed attention: K topologies
  - often different features per topology

$$\mathbf{V}' \leftarrow \sigma\Big(\sum_{k=1}^{K} \mathbf{W}^k \mathbf{V} \mathbf{B}^k\Big)$$

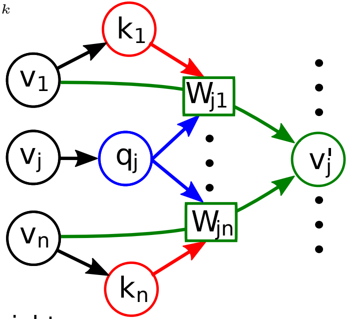


assignment sheet 2

transformer originally from NLP (Vaswani et al., 2017), applied to RL by Kurin et al. (2020)

4.3 Multi-headed attention layers (MHA)

- What if we don't know the best topology?
  - every input node has key $\boldsymbol{k}_i = \mathbf{A}\boldsymbol{v}_i \in \mathbb{R}^{m_k}$
  - every output has query vector $\boldsymbol{q}_j \in \mathbb{R}^{m_k}$
  - topology matrix $W_{ji} \propto \exp(\boldsymbol{q}_j^\top \boldsymbol{k}_i)$



- Multi-headed attention: K topologies
  - often different features per topology

$$\mathbf{V}' \leftarrow \sigma\Big(\sum_{k=1}^{K} \mathbf{W}^k \mathbf{V} \mathbf{B}^k\Big)$$

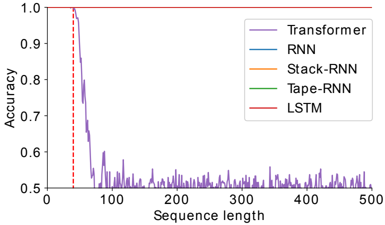
- Transformer (self-attention):
  - fully connected R-GCN with learned weights
  - queries are 'other keys': $\boldsymbol{q}_j = \bar{\mathbf{A}}\boldsymbol{v}_j \in \mathbb{R}^{m_k}$
  - softmax: $\mathbf{S}^k := \exp(\mathbf{V}\bar{\mathbf{A}}^{k\top}\mathbf{A}^k\mathbf{V}^\top)$, $\mathbf{W}^k := \mathbf{S}^k \oslash \mathbf{S}^k\mathbf{1}\mathbf{1}^\top$
  - each layer has different parameters $\mathbf{A}^k$, $\bar{\mathbf{A}}^k$ and $\mathbf{B}^k$

assignment sheet 2

transformer originally from NLP (Vaswani et al., 2017), applied to RL by Kurin et al. (2020)

Question: transformer generalization

- In the Chomsky hierarchy, transformers (MHA) do not generalize
  - cannot represent regular languages (e.g. parity check)
  - but training performance (until dashed line) looks perfect
- Which property prevents generalization?
- Why is training performance so good?
- How could we fix this problem?

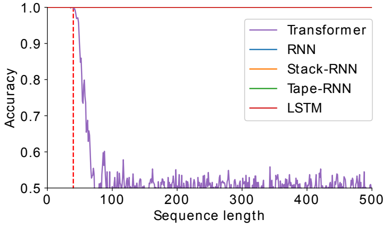


parity check
(regular language)

figure from Deletang et al. (2023)

Question: transformer generalization

- In the Chomsky hierarchy, transformers (MHA) do not generalize
  - cannot represent regular languages (e.g. parity check)
  - but training performance (until dashed line) looks perfect
- Which property prevents generalization?
- MHA are permutation invariant
  cannot represent sequences
- Why is training performance so good?
- How could we fix this problem?

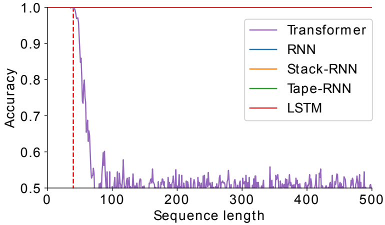


parity check
(regular language)

figure from Deletang et al. (2023)

Question: transformer generalization

- In the Chomsky hierarchy, transformers (MHA) do not generalize
  - cannot represent regular languages (e.g. parity check)
  - but training performance (until dashed line) looks perfect
- Which property prevents generalization?
- MHA are permutation invariant
  cannot represent sequences
- Why is training performance so good?
- MHA can still overfit to all
  training sequences
- How could we fix this problem?

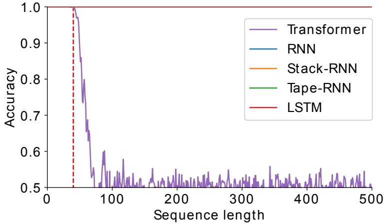


parity check
(regular language)

figure from Deletang et al. (2023)

Question: transformer generalization

- In the Chomsky hierarchy, transformers (MHA) do not generalize
  - cannot represent regular languages (e.g. parity check)
  - but training performance (until dashed line) looks perfect
- Which property prevents generalization?
- MHA are permutation invariant
  cannot represent sequences
- Why is training performance so good?
- MHA can still overfit to all
  training sequences
- How could we fix this problem?
- E.g. add trainable position k encoding $\boldsymbol{p}_k$
  to each token $\boldsymbol{v}_k \leftarrow \boldsymbol{v}_k + \boldsymbol{p}_k$



parity check
(regular language)

positional encoding was already defined in Vaswani et al. (2017)

figure from Deletang et al. (2023)

Hyper & multiplicative networks

- Neural networks can output other neural networks
  - hyper-network linear layer: $\boldsymbol{f}_\theta(\boldsymbol{x}) := \mathbf{A}\boldsymbol{x}, \quad A_{ij} := g_\theta^{ij}(\boldsymbol{x}')$
  - very useful to *merge* different input “views” $\boldsymbol{x}$ and $\boldsymbol{x}'$
  - special case of multiplicative layers

```python
import torch as th
class HyperNetwork (th.nn.Module):
    def __init__(self, in_feats, out_feats, in_alt=None, hid=512):
        super().__init__()
        in_alt = in_feats if in_alt is None else in_alt
        self.weights = th.nn.Sequential(
            th.nn.Linear(in_alt, hid), th.nn.ReLU(),
            th.nn.Linear(hid, in_feats * out_feats))

    def forward(self, x, x2=None):
        w = self.weights(x if x2 is None else x2)
        w = w.view(*x.shape, w.shape[-1] // x.shape[-1])
        return th.bmm(x.unsqueeze(dim=-2), w).squeeze(dim=-2)
```

hyper networks by Ha et al. (2017), a general discussion on multiplicative layers in Jayakumar et al. (2020)

4.3 Summary

- GNN take annotated graphs as input
- CNN are special cases of R-GCN
- MHA are GNN with learned topology
- Hypernetworks output neural network weights

Learning Objectives

LO4.3: Explain GCN, MHA and hyper-networks
LO4.4: Reduce a module to another analytically

4.3 Next lecture

- Next lecture:
  advanced Q-learning!

- New assignment 2 is out!
  - due 07-12-2023
  - also new exercise sheet 2
  - try to solve them before
    Friday's lab session

- Questions? Ask them here:
  `answers.ewi.tudelft.nl`

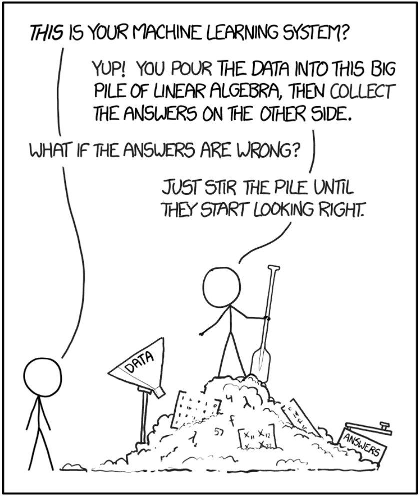


image source: `xkcd.com`

References I

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018. URL https://arxiv.org/abs/1806.01261.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Workshop on Deep Learning*, 2014. URL http://arxiv.org/abs/1412.3555.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989. ISSN 0932-4194. doi: 10.1007/BF02551274. URL http://dx.doi.org/10.1007/BF02551274.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A Ortega. Neural networks and the chomsky hierarchy. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2207.02098.

Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3): 183–192, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90003-8. URL https://www.sciencedirect.com/science/article/pii/0893608089900038.

Ian Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *International Conference of Learning Representation (ICLR)*, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618.

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representation (ICLR)*, 2017. URL https://arxiv.org/abs/1609.09106.

References II

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, 07–09 Jul 2015. URL https://proceedings.mlr.press/v37/ioffe15.html.

Siddhant M. Jayakumar, Wojciech M. Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rylnK6VtDH.

Zhengyao Jiang, Pasquale Minervini, Minqi Jiang, and Tim Rocktäschel. Grid-to-graph: Flexible spatial relational inductive biases for reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 674–682, 2021. URL https://arxiv.org/abs/2102.04220.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*. 2017.

Boris Knyazev, Michal Drozdzal, Graham W. Taylor, and Adriana Romero-Soriano. Parameter prediction for unseen deep architectures, 2021. URL https://arxiv.org/abs/2110.13100.

Vitlay Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Böhmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint 2010.01856*, 2020. URL https://arxiv.org/abs/2010.01856.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

References III

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2924–2932, 2014.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Proceedings*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 593–607, 2018.

Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268, 2017. ISSN 1063-5203. doi: 10.1016/j.acha.2015.12.005.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. URL http://arxiv.org/abs/1607.08022.

Elise van der Pol, Daniel Worrall, Herke van Hoof, Frans Oliehoek, and Max Welling. MDP homomorphic networks: Group symmetries in reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 4199–4210, 2020. URL https://arxiv.org/abs/2006.16908.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *ICLR*, 2018. URL https://openreview.net/forum?id=S1sqHMZCb.

Yuxin Wu and Kaiming He. Group normalization. *International Journal of Computer Vision*, 128(3):742–755, 2020. doi: 10.1007/s11263-019-01198-w. URL https://arxiv.org/abs/1803.08494.

References IV

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *13th European Conference of Computer Vision (ECCV)*, pages 818–833, 2014.

Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010. doi: 10.1109/CVPR.2010.5539957.

Zhou and Chellappa. Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks*, volume 2, pages 71–78, 1988. doi: 10.1109/ICNN.1988.23914.