Deep Reasoning

Hardware Accelerated Artificial Intelligence

Phillip Lippe, Stephan Schulz

DHBW Duale Hochschule Baden-Württemberg Stuttgart

Motivation

- Traditionally, search heuristics for theorem provers are hand-coded
- Problem: inefficient and not always getting optimal results
- Deep learning has shown great success in various domains for recognizing patterns and analysis of complex structures ⇒ applying for search heuristics
- Learning from existing proofs to distinguish useful and distracting selections

General architecture

- Determining a score between 0 (useful for the proof) and 1 (not useful) for each clause separately which is not changed during the proof search
- Every clause is evaluated regarding the conjecture to be proved (second input)
- Possible proofs limited to initial clauses ⇒ third input for evaluation
- Balance between accuracy/complexity and computation time of network

Clause

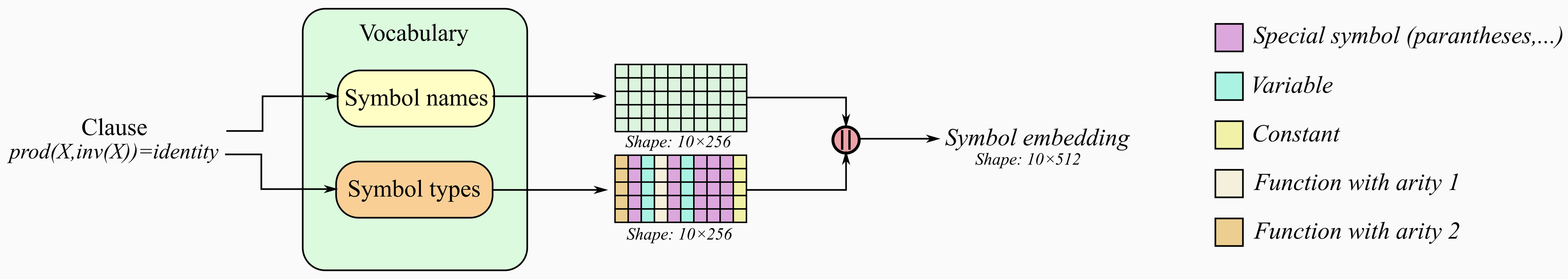
- First-order clauses to be evaluated
- Example: *prod(a, inv(a)) = identity*

Negated conjecture

- Conjecture to be proved
- Example: *prod(X, identity) = X*

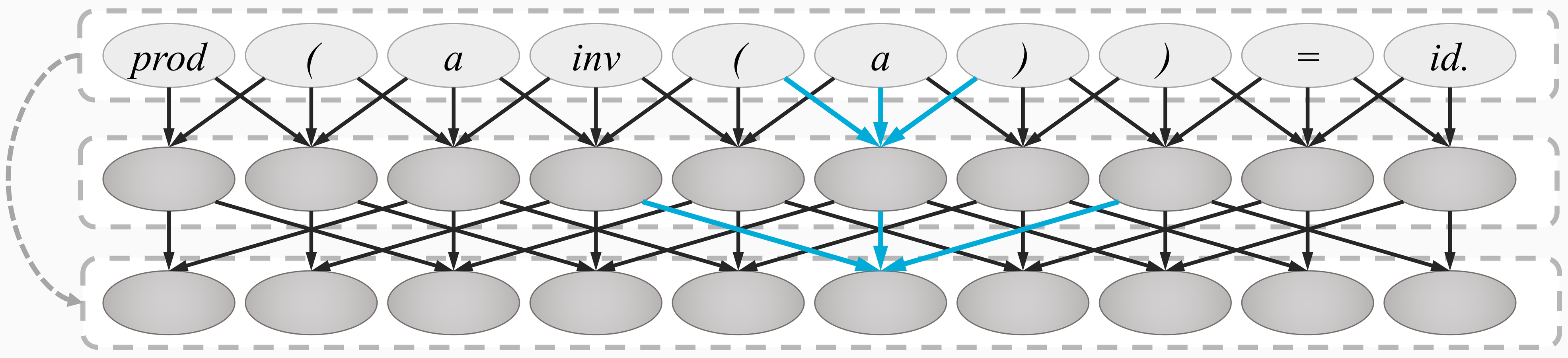
Vocabulary

- **Goal:** Convert clauses to representation suitable for NNs
- Every symbol is assigned to a feature vector
- Part of features is shared among similar symbols
- Feature vectors are learned during training
- Shared vocabulary for negated conjecture and clauses



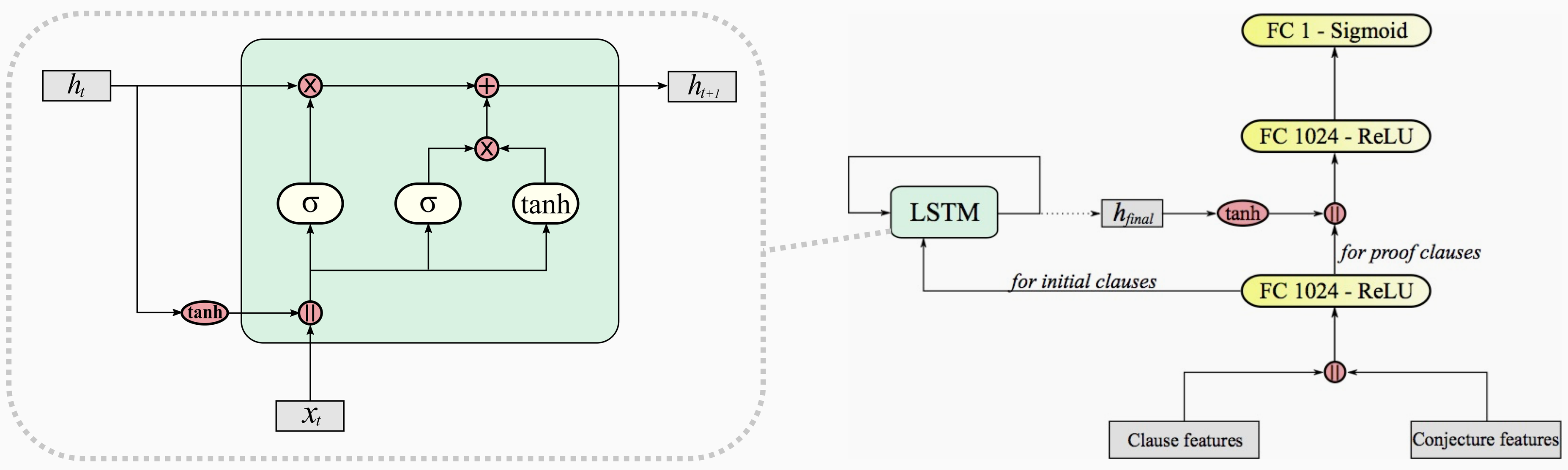
Embedding network

- **Goal:** compress an arbitrary clause into a fixed size feature vector
- Complexity is constrained by the runtime performance
- Principal component: 5-layer dense block with increasing dilation
- Processing clauses with different fields of view
- Features from all previous layers are combined as input
- Final max-pooling over features guarantee fixed size feature vector



Combiner network

- **Goal:** combine the features to predict a heuristic score
- Applying fully connected (FC) layers as feature sizes are fixed
- After first layer, the features of the initial clauses are combined by using a modified LSTM block ⇒ additional input to clauses
- Last layer compresses features to single value between 0 and 1



Performance measurement

- Testing proof deduction is expensive and not practicable during training
- Existing proofs create significantly imbalanced dataset (useful vs not useful, ...)
- Applying extensive data augmentation and sampling based on loss
- Loss function: $\mathcal{L}(p) = \begin{cases} -\alpha_1 \cdot \log\left(\frac{1+p\cdot(e-1)}{e}\right) \cdot (1-p)^\gamma & \text{for } y=1 \\ -\alpha_0 \cdot \log(1-p) \cdot p^\gamma & \text{for } y=0 \end{cases}$

Results and Outlook

- Approach tested on TPTP problem library with limited vocabulary
- Implemented in TensorFlow and integrated in the E-Prover (single GPU TitanX)
- Network could correctly classify 92.6% of all positive and 89.2% of all negative clauses from test dataset, but lacking in finding new proofs (only 1 out of 25)
- Problems to tackle in the future: partwise useful clauses | integrating proof/clause structure into network | reinforcement learning on own proofs