Latte: A Language, Compiler, and Runtime for Elegant and Efficient Deep Neural Networks

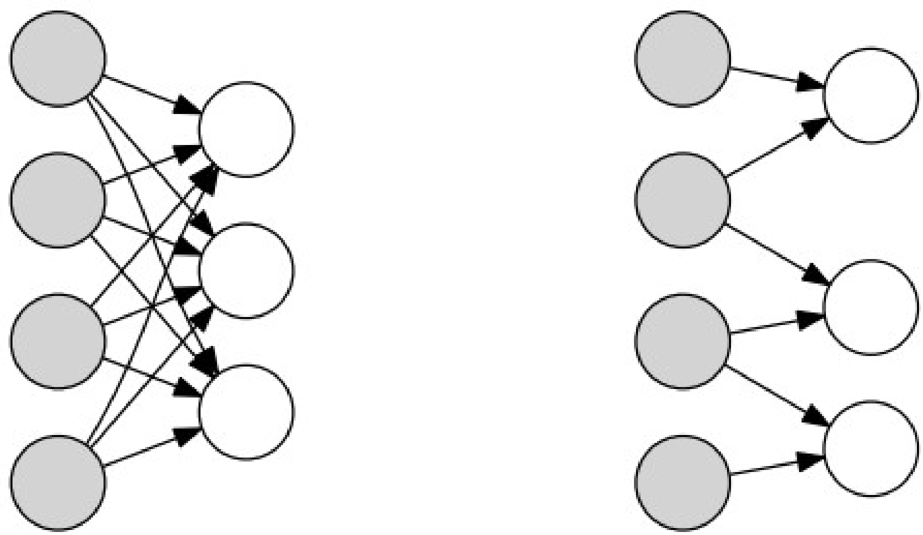
Truong et al. PLDI 2016

Neural Networks

- Family of machine learning algorithms that are inspired by the biological neural networks.
- Used to estimate the output(s) of functions that can depend on a large number of inputs.

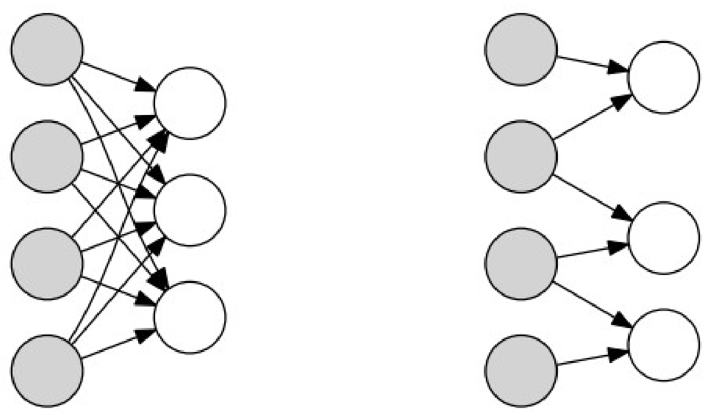
Neural Networks

- Presented as a collection of connected neurons.
- A connection between neurons indicates that messages are passed between the two neurons.



Deep Neural Networks (DNN)

- Neurons organized into layers: first=input, last=output.
- Intermediate layers = hidden layers. DNN has many.



(a) Fully connected layer (b) Sparsely connected layer

Figure 1: Simple examples of various layer types. Gray neurons represent input neurons and white neurons represent output neurons.

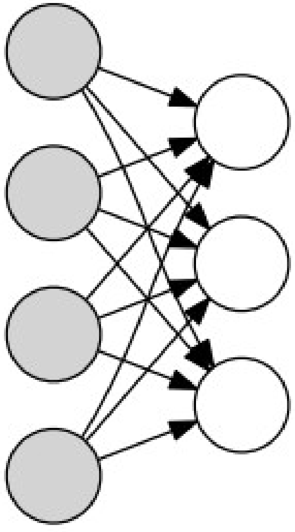
Training Neural Networks

- Forward propagation: training input fed through the network to produce output activations.
- Backward propagation: output activations are propagated backwards through the network to calculate the difference between the input and output values of each neuron.
- Differences = gradients.
- Gradients used to update weight of a neuron.
- Trained on batch of input items.

Latte programming model

- Network is a collection of connected ensembles.
- Ensemble is a collection of neurons.
- Ensembles are connected using a mapping function that specifies the connections between its neurons.

- Neurons: circles/nodes.
- Ensemble: vertical line/group of neurons.
- Mapping function: edges.
- Network: graph.



Latte programming language

- Extension to the Julia programming language.
- Neuron type: has default fields -
  - output value and its gradient,
  - a vector of vector of inputs and its gradient.
- For each neuron type, forward and backward propagation functions must be specified.
- Ensemble type: parametrized by the the number and type of neurons.

Latte compiler: IR

- Superset of the internal Julia AST (high-level).
- User-defined networks are dataflow graphs.
- Nodes (neurons) – computations.
- Edges (connections) – data dependences.
- Mapping function – implicit adjacency list (not stored).

Latte compiler: stages

1) Analysis
2) Synthesis
3) Optimization
4) Code Generation

Latte compiler: analysis

- Determines nodes that share data dependences.
- Example: neurons consuming the same input.
- Condition: when the adjacency list of neurons in an ensemble is uniform (independent of neuron).
- Action: shared buffer for shared variables.

Uses:

- Reduces memory consumption.
- Improves data locality.
- Enables pattern-matching of computation.

```
for n = 1:num_neurons
  for i = 1:num_inputs
    fc_value[n] += fc_inputs[i, n]
```

(a) Before: Each neuron has a different set of inputs demonstrated by the n used to index `fc_inputs`

```
for n = 1:num_neurons
  for i = 1:num_inputs
    fc_value[n] += fc_inputs[i]
```

(b) After: All neurons share an input vector, allowing us to drop the n when indexing `fc_inputs`.

Figure 8: FCLayer pseudo-code before and after shared variable analysis.

Latte compiler: synthesis

- Dataflow: compiler is responsible for ensuring input data is available for computation of neuron's output.
  - Latte emits data copy to copy source's output value to sink's input buffer.
- Compute: converts array-of-structs (AoS) to structure-of-arrays (SoA) layout to enable vectorization.

```
# perform dot product of weights and inputs
for i in 1:length(neuron.inputs[1])
  neuron.value +=
      neuron.weights[i] * neuron.inputs[1][i]
```

Latte compiler: optimizations

- Pattern matching: transforms synthesized code to BLAS matrix multiplication calls, when possible.
- Loop tiling.
- Cross-layer (or loop) fusion: only when there is no loop-carried dependence.