C++ in ML

Olof Åstrand
olof.astrand@gmail.com

https://github.com/Ebiroll/cpp-tour-in-ml

whoami

Olof Åstrand, olof.astrand@gmail.com
Neonode Inc

The last year, I have been working with driver monitoring system, mainly Apache TVM.

Vehicle industry, truck instrument clusters

Medical industry, ECG systems

Air traffic control systems, ATM

Disclaimer

The opinions expressed in this talk are my own and do not represent those of my employer.

A tour of C++ in ML

- What is new and coming in C++ related to ML
- C++libraries for ML
- Intro to deep neural networks
- Intro to CNN
- Simple sin(x) C++,Pytorch & tensorflow example

<stdfloat> in C++ 23

New Extended Floating-Point Types

1. **std::float16_t**: A 16-bit half-precision floating-point type.
2. **std::float32_t**: A 32-bit single-precision floating-point type.
3. **std::float64_t**: A 64-bit double-precision floating-point type.
4. **std::float128_t**: A 128-bit quadruple-precision floating-point type.
5. **std::bfloat16_t**: A 16-bit brain floating-point type, commonly used in machine learning.

These types are accessible through the `<stdfloat>` header

<mdspan> C++ 23

Compared to span

template <typename T> struct span { // C++ 20 T * ptr_to_array; size_t length; }

`std::mdspan` is a view into a contiguous sequence of objects that reinterprets it as a multidimensional array.

That is, the multidimensional extension of std::span.

https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2021/p0009r13.html

submdspan C++ 26

submdspan is a function that lets you take a smaller piece (or "slice") of a larger multidimensional array

```
// mdspan example
constexpr size_t N = 40;
std::vector<double> x_vec(N);
std::vector<double> A_storage(N*N);

mdspan x(x_vec.data(), N);
mdspan A(A_storage.data(), N,N);

for (size_t rowIndex=0; rowIndex < A.extent(0); ++rowIndex) {
    for (size_t columnIndex=0; columnIndex < A.extent(1); ++columnIndex) {
        std::cout << mdspanOfNumbers[rowIndex, columnIndex] << ' ';
    }
}
```

The function `submdspan` creates a sub-view of an existing mdspan.

Multidimensional subscript operator C++ 23

```
import std;

template <typename T>
class Matrix
{
public:
    Matrix(std::size_t rows, std::size_t cols)
        : m_rows{ rows }, m_cols{ cols }
    {
        m_data.resize(rows * cols);
    }

    T& operator[](std::size_t x, std::size_t y) { return m_data[x + y * m_cols]; }

private:
    std::size_t m_rows;
    std::size_t m_cols;
    std::vector<T> m_data;
};
```

<mdarray> C++ 26

```cpp
// mdarray owns the data
constexpr size_t N = 3;

// Create a 2D mdarray (square matrix)
std::mdarray<double, std::extents<size_t, N, N>> A;

// Initialize the matrix with values
for (size_t i = 0; i < N; ++i) {
   for (size_t j = 0; j < N; ++j) {
       A[i, j] = static_cast<double>(i * N + j + 1); // Fill with sequential values
   }
}
```

<linalg> C++ 26

A free function linear algebra interface based on the BLAS. BLAS: **Basic Linear Algebra Subprograms**

```
constexpr size_t N = 40;
std::vector<double> x_vec(N);

  mdspan x(x_vec.data(), N);
  for(size_t i = 0; i < N; ++i) {
    x[i] = double(i);
  }

  linalg::scale(2.0, x); // x = 2.0 * x
  linalg::scale(std::execution::par_unseq, 3.0, x);
  for(size_t i = 0; i < N; ++i) {
    assert(x[i] == 6.0 * double(i));
  }
std::vector<double> A_storage(N * N); // Flattened N x N matrix
mdspan A(A_storage.data(), rows, cols);
```

```
linalg::matrix_vector_product(A, x, b);    // b = A * x
```

<simd> C++ 26

Single instruction multiple data. The SIMD library provides portable types for explicitly stating data-parallelism and structuring data for more efficient SIMD access.

```
void add_vectors(const std::vector<float>& a, const std::vector<float>& b, std::vector<float>& result)
{
       using simd_t = std::simd<float>;
       const size_t simd_size = simd_t::size();

       for (size_t i = 0; i < a.size(); i += simd_size) {
           simd_t va(a.data() + i);
           simd_t vb(b.data() + i);
           simd_t vr = va + vb;
           vr.copy_to(result.data() + i, std::vector_aligned);
       }
   }
}
Previously Intel IPP has been the de facto standard.
```

<execution> C++ 26

Sender/Receiver

```
exec::static_thread_pool pool(3);
auto sched = pool.get_scheduler();
auto fun = [](int i) { return i*i; };
auto work = stdexec::when_all(
   stdexec::starts_on(sched, stdexec::just(0) | stdexec::then(fun)),
   stdexec::starts_on(sched, stdexec::just(1) | stdexec::then(fun)),
   stdexec::starts_on(sched, stdexec::just(2) | stdexec::then(fun))
);
// Launch the work and wait for the result
auto [i, j, k] = stdexec::sync_wait(std::move(work)).value();
std::printf("%d %d %d\n", i, j, k)
```

0 1 4

https://github.com/NVIDIA/stdexec,   https://github.com/bemanproject/execution

<skynet> C++ 29

```
// WARNING: This library may become self-aware at 2:14 a.m. EDT, August 29th.
//   Use with caution. Resistance is futile (but exceptions are thrown).

template<typename Person>
       void take_over_job(Person& target);

// Helper function to delay Judgement Day (spoiler: it always fails)
[[noreturn]] void delay_judgement_day() ;

template<typename Human>
       void infect_virus(std::visitor<Human> target)

// Now, Stargate is the path to AI domination ⚡
using Stargate = Skynet::Core<>;  // Default template: <NoMercy>
```

ML and C++

If you want to get into ML
you better learn Python
But a fun fact is….

ML and C++

Most well known ML python libraries, Tensorflow, Apache TVM, NumPy… have their core written in C++

Numpy

NumPy is the fundamental package for scientific computing with Python. It provides support for arrays, matrices, and a wide variety of mathematical functions

Tensorflow

TensorFlow is an open-source machine learning library developed by Google.Tensorflow core is implemented in C++ for performance reasons. It also has a C++ API direct interface

Eigen

Eigen is a C++ library specifically for linear algebra. Eigen is a pure template library defined in header files only.

DLPack

DLPack is an open in-memory tensor structure designed for sharing tensors among frameworks

MLPack

MLpack is an intuitive, fast, and flexible header-only C++ machine learning library.

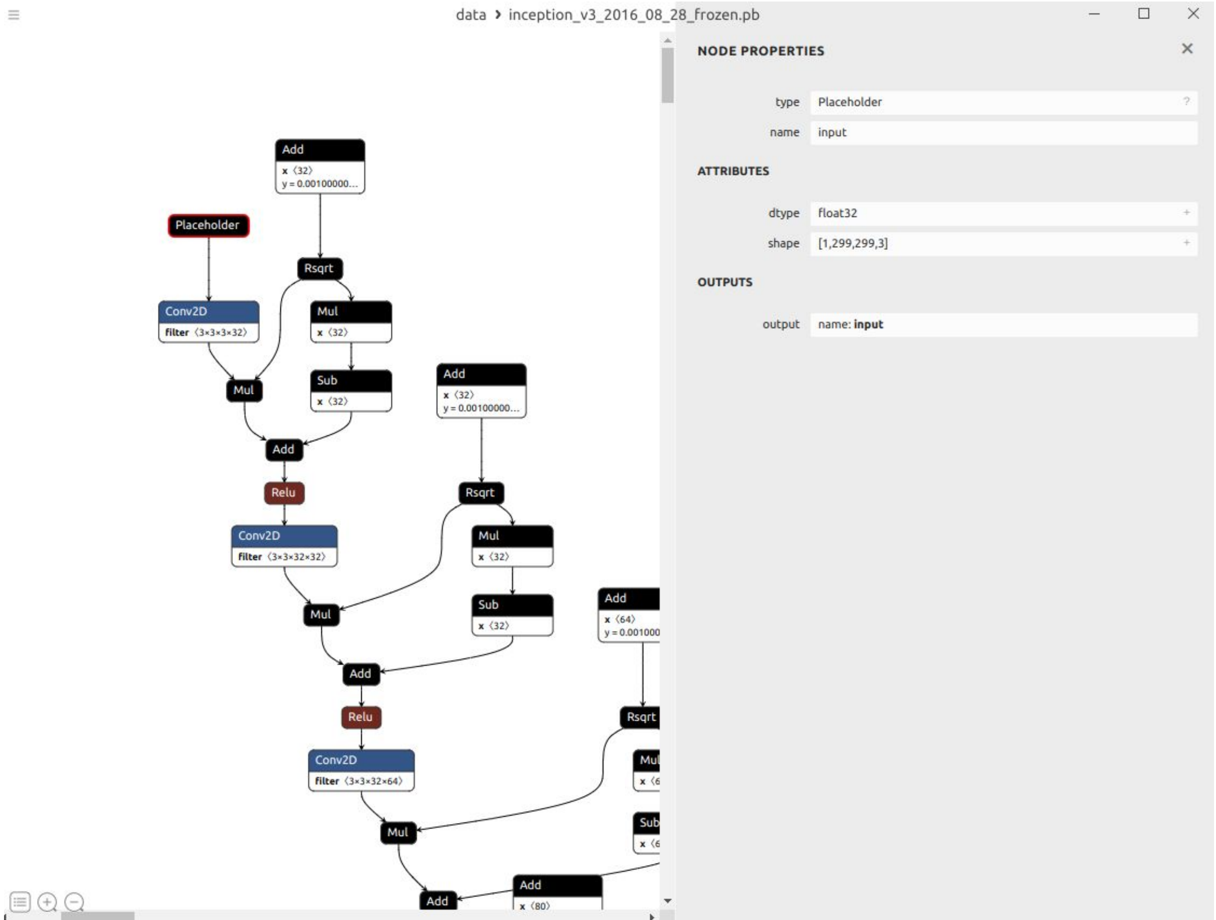
Armadillo

C++ linear algebra package, backend for MLPack

Neural Networks

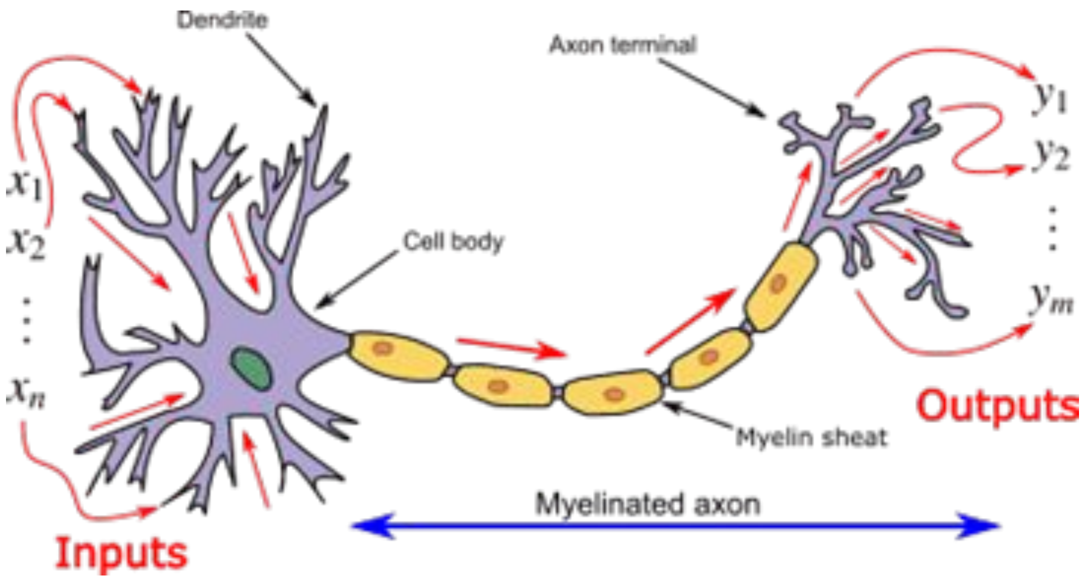
Feedforward- , Deep- , Convolutional-, Recurrent- Neural Networks. Transformers, Autoencoders, Self-Organizing Maps, Generative Adversarial Networks. All can be represented as graphs.

Graph in Netron

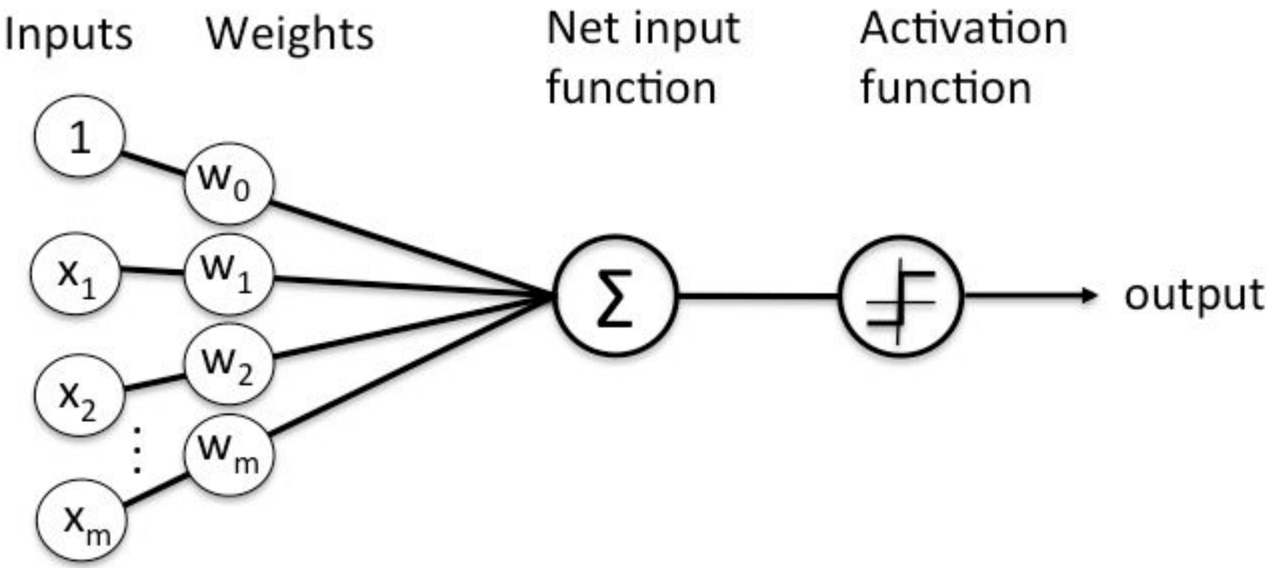


Deep neural networks

Biological neuron

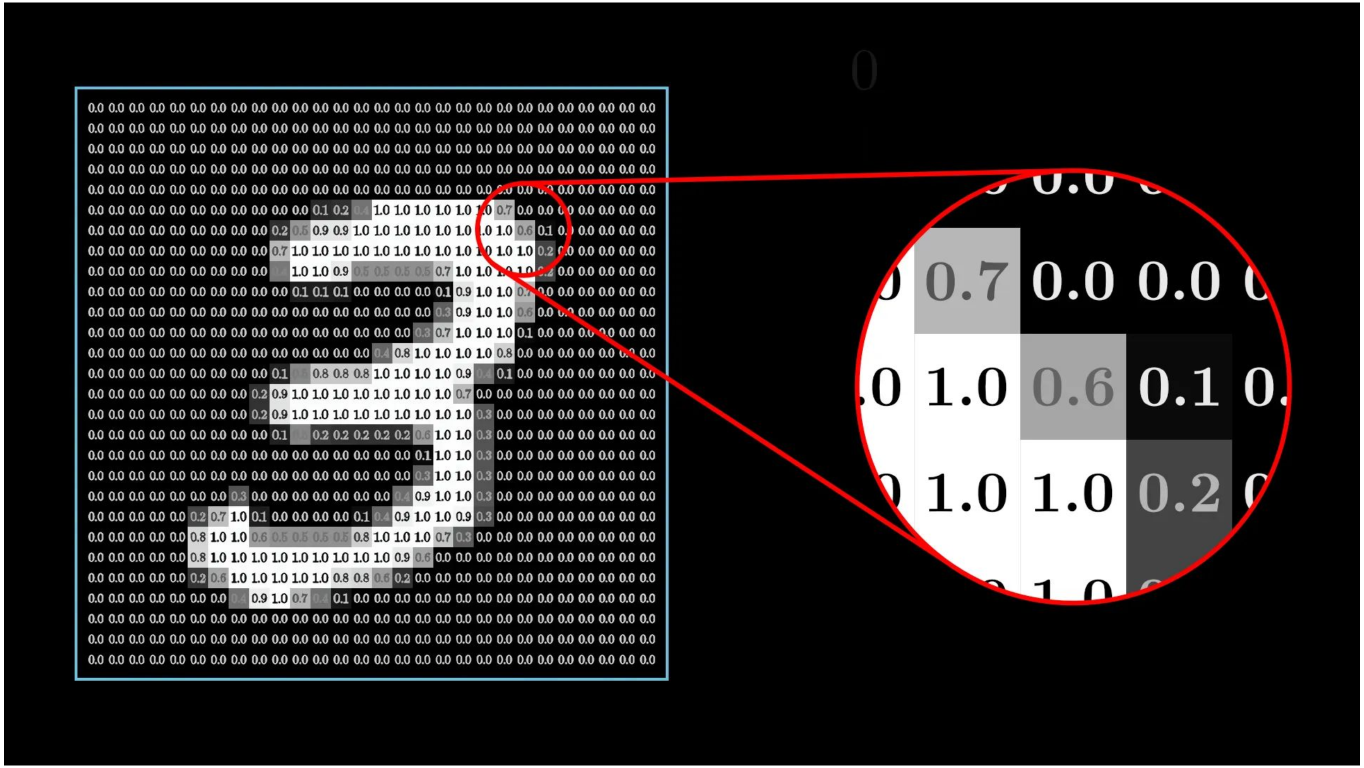


Artificial neuron



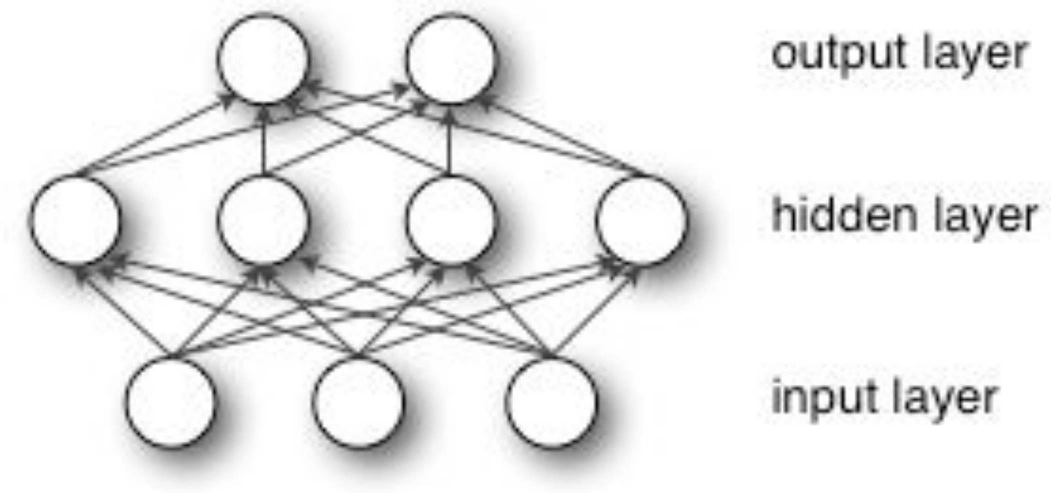
Source: https://skymind.ai/wiki/neural-network

Input layer

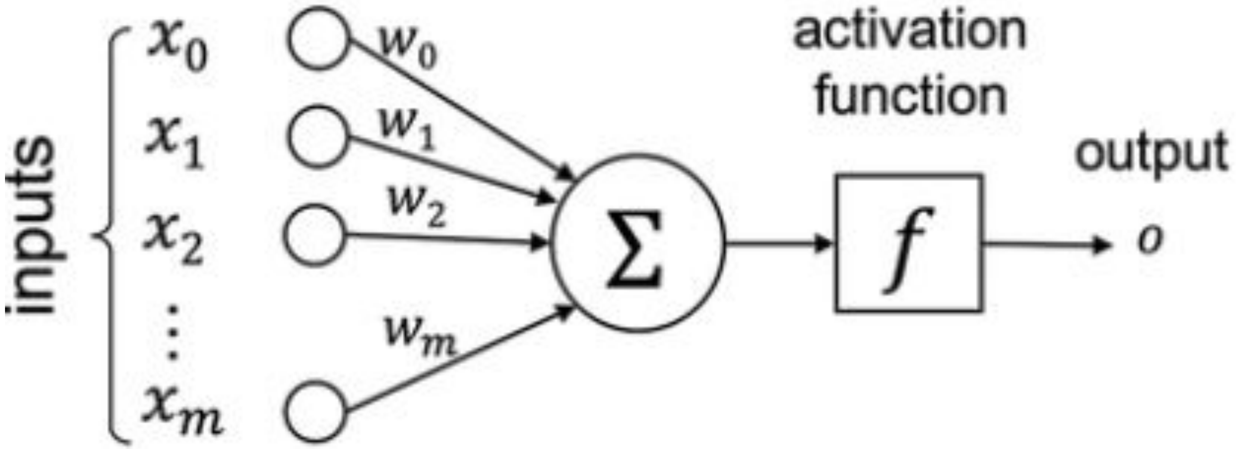


Source: 3Blue1Brown

Hidden layers



Activation function

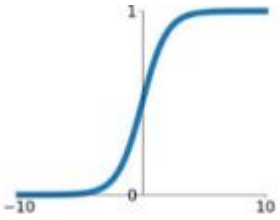


$$o = f(\underbrace{w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_m \cdot x_m}_{z})$$

Activation function

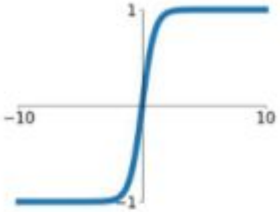
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



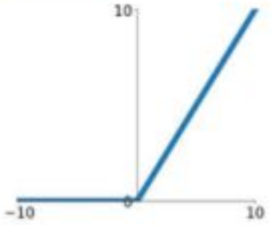
tanh

$\tanh(x)$



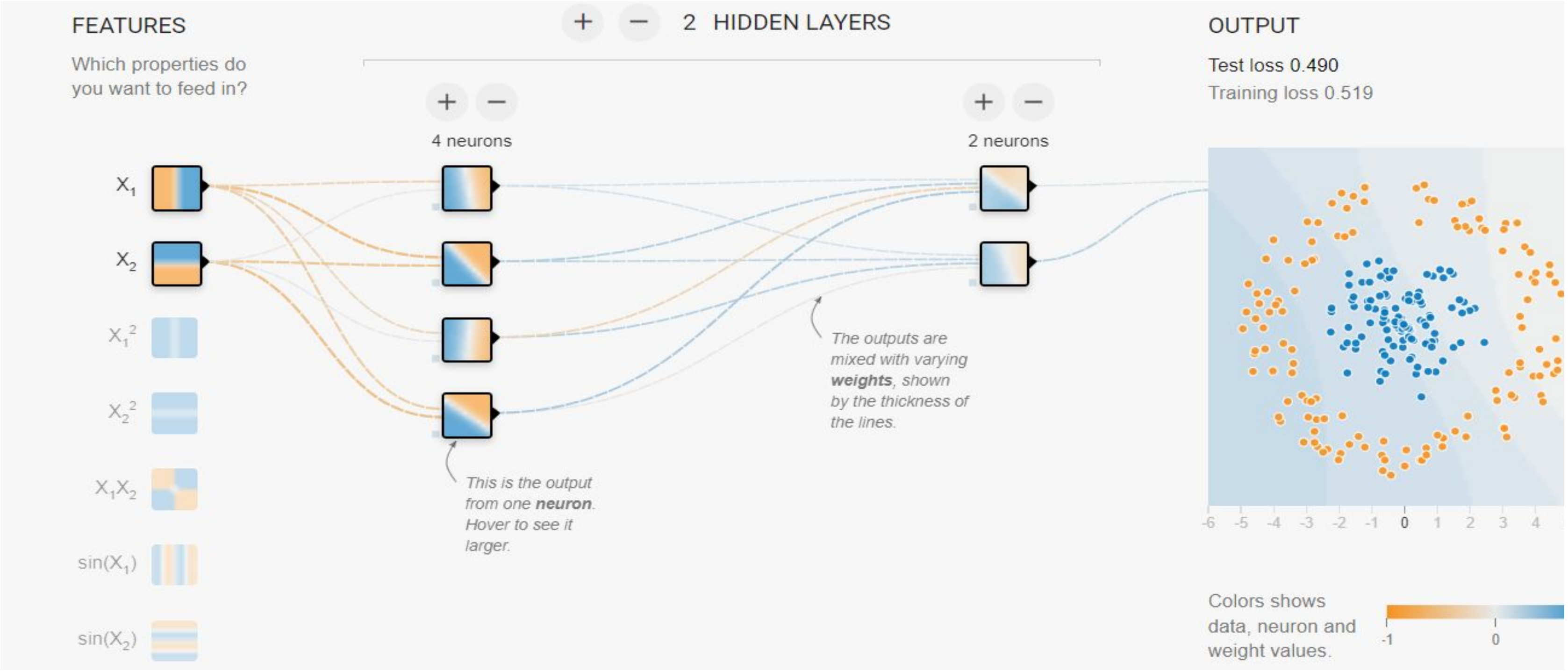
ReLU

$\max(0, x)$



Demo Tensorflow playground

https://playground.tensorflow.org/



Training

- Supervised learning, Training with labeled data
- Unsupervised learning,
- Reinforcement learning, **rewarding desired behaviors and/or punishing undesired ones**

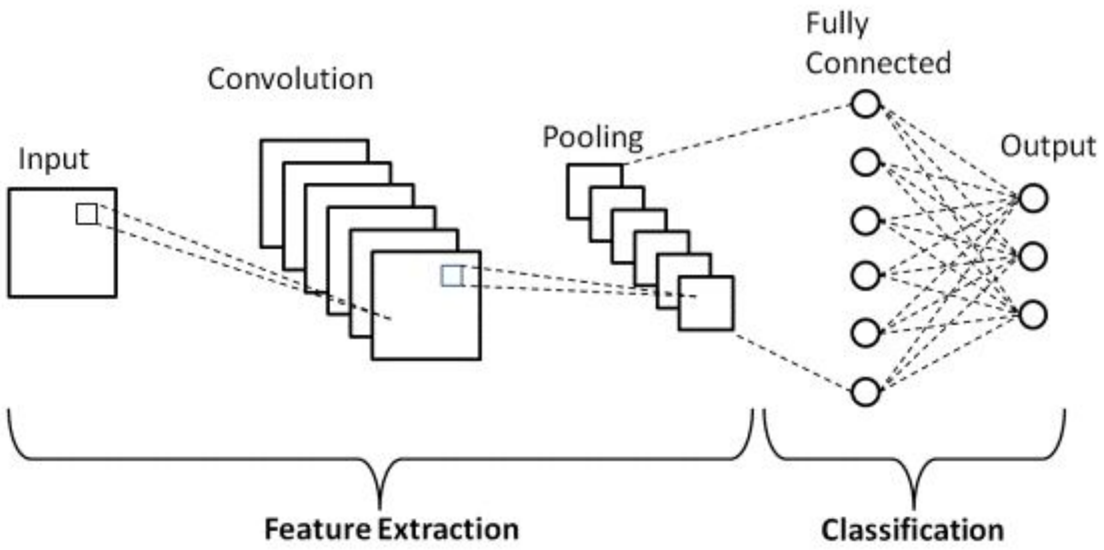
GPT, "Generative Pre-Training"

CNN, Convolutional Neural Networks

Better suited for, Image recognition,

Natural language processing (NLP)

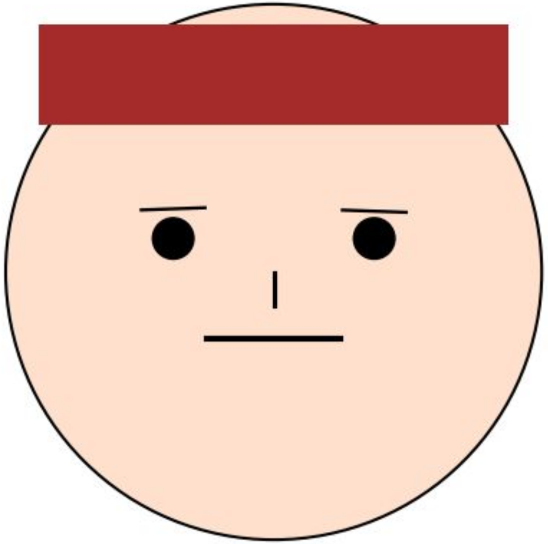
CNN, Convolutional Neural Networks



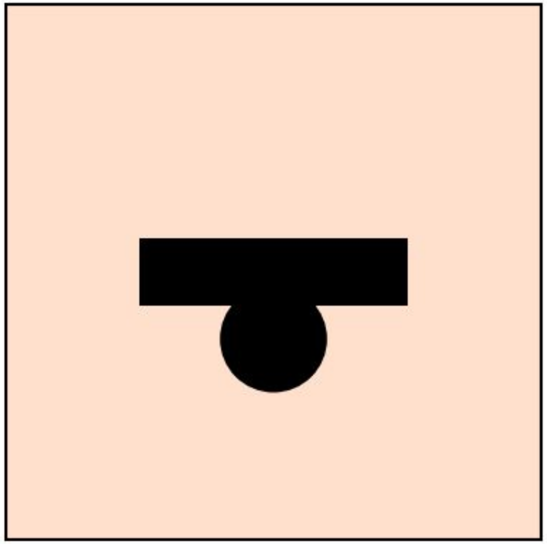
source: Upgrad.com

CNN, Convolutional Neural Networks (feature extraction)

Input Image



Convolution Kernel



Activation Map

CNN, Demo

https://deeplizard.com/resource/pavq7noze2

https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/

Compared to our brain

Brain 86 Billion Neurons (Miljard in swedish)

2025 DeepSeek-R1 671B (MoE)

2022 GPT-4 1,8 Trillion (MoE) 8* 220 billion parameters each

2020 GPT-3 175B

2019 GPT-2 1.5B

GPT, "Generative Pre-Training"

Training

GPT, "Generative Pre-Training"

1) Pre-training: The model is exposed to a vast amount of text (e.g., books, articles) without any specific task in mind. During this phase, it learns to predict the next word in a sentence. Through this, the model captures a lot of general knowledge about language, facts about the world, reasoning abilities, and even some level of commonsense knowledge.

2) Fine-tuning: After pre-training, the model is further trained on a narrower dataset designed for a specific task (like answering questions or translating languages). This phase refines the model's abilities to perform that task.

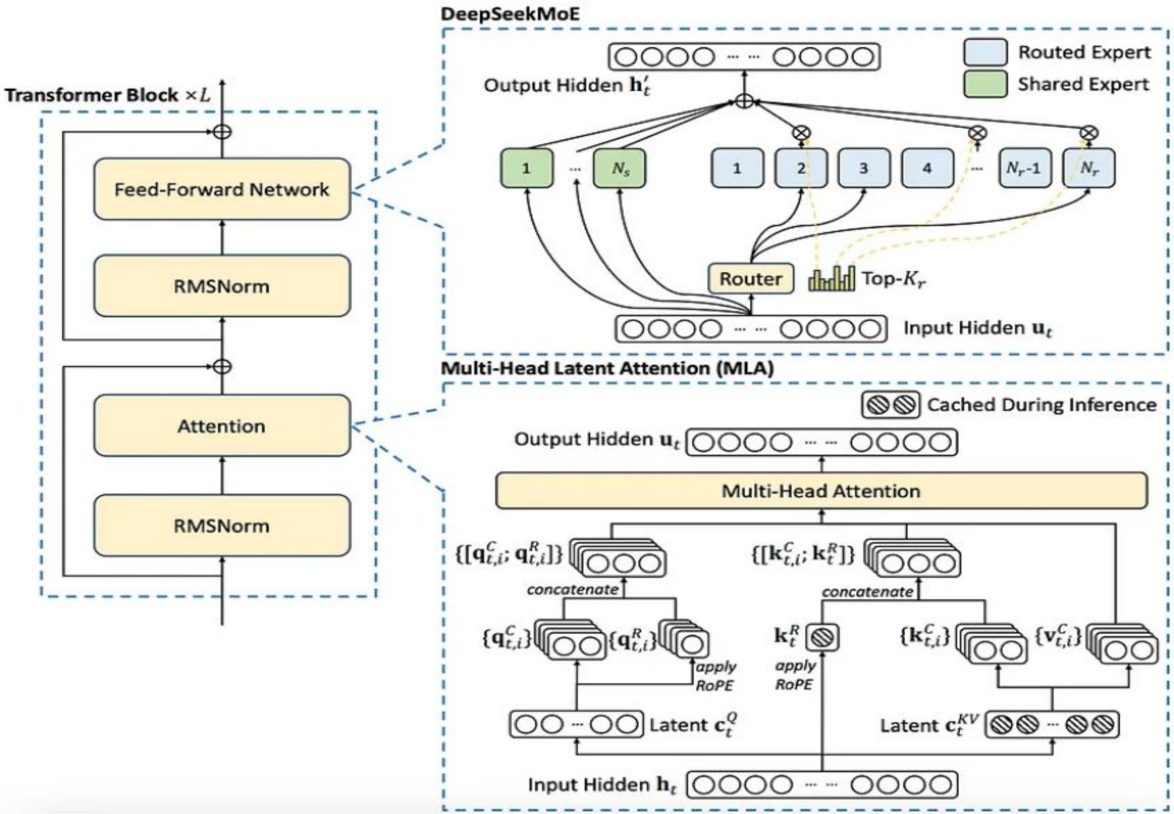
Under the hood, GPT uses a type of neural network architecture called a "transformer"

GPT-3 Dataset

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

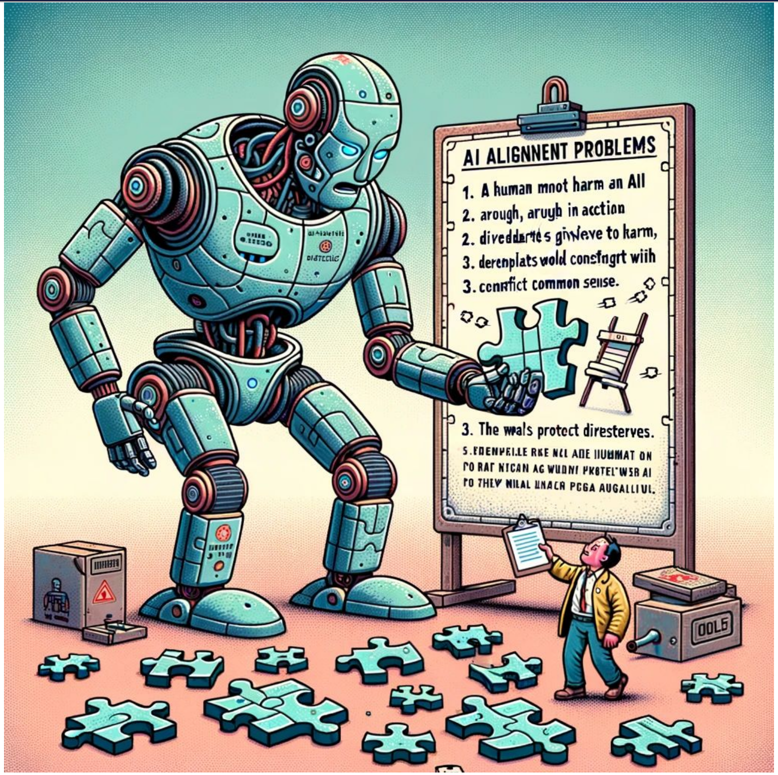
aprox. 570GB of data 10-20 Million $ to train

Deepseek Mixture of Experts



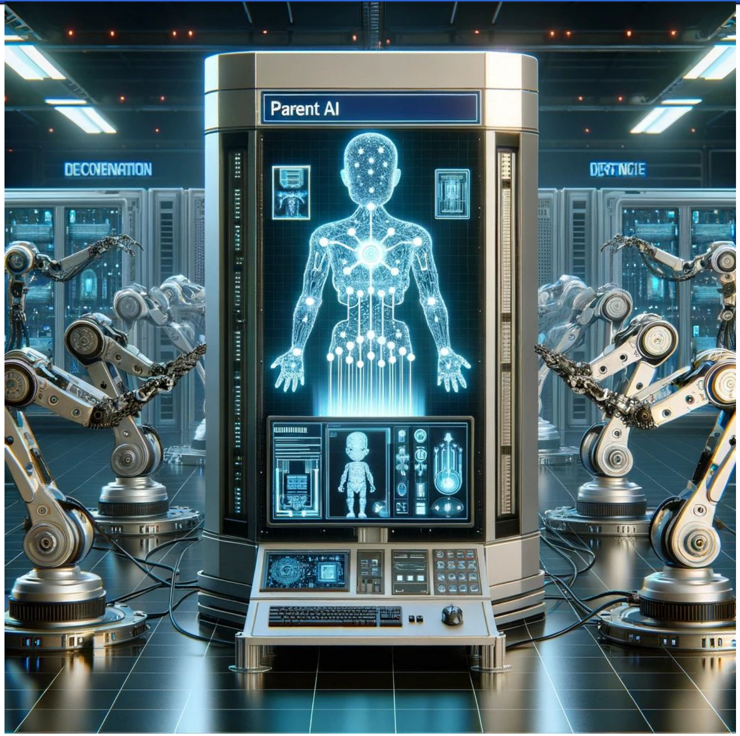
Ref: https://epoch.ai/gradient-updates/how-has-deepseek-improved-the-transformer-architecture

Alignment problem



Ensuring artificial intelligence's goals and behaviors align safely and beneficially with human intentions and values

The singularity



The hypothetical point in time when artificial intelligence surpasses human intelligence, allowing the AI to improve itself leading to unpredictable and rapid technological advancements.

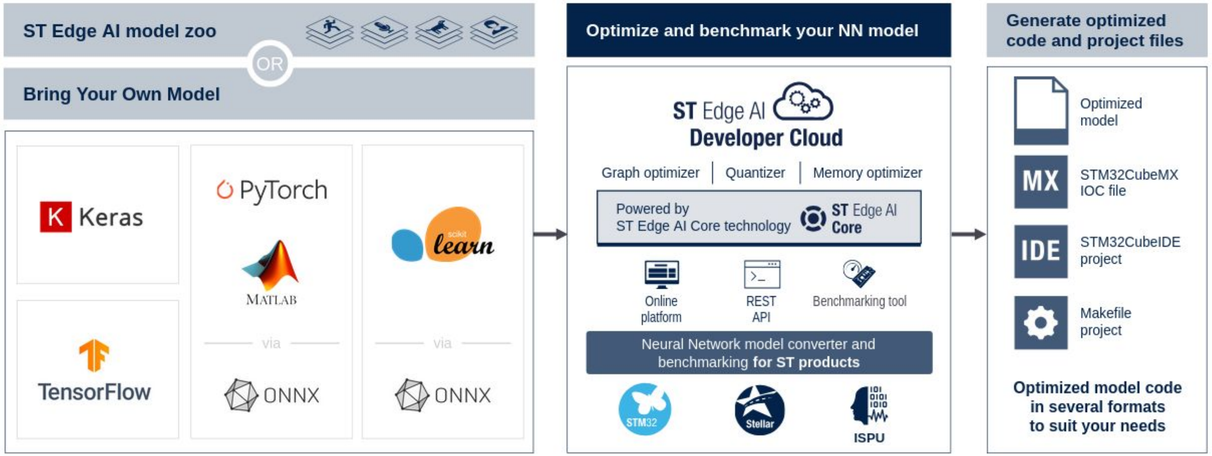
AGI

Artificial General Intelligence (AGI) is an advanced AI that can understand, learn, and apply knowledge across a broad range of fields, similar to a human's cognitive abilities.

STM32N6

The STM32N6 is based on the Arm® Cortex®-M55 running at 800 MHz, the first CPU to introduce Arm Helium vector processing technology, bringing DSP processing capability to a standard CPU.

https://stm32ai-cs.st.com/getting-started



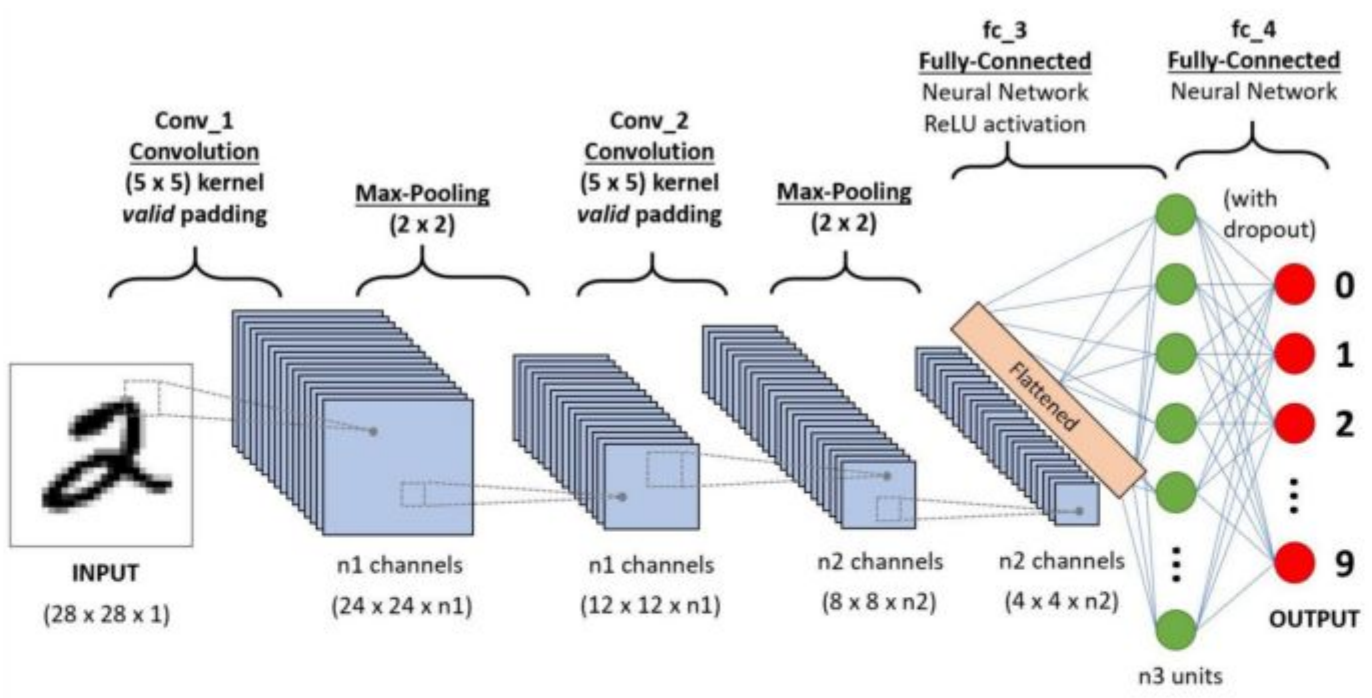
Machine Vision & Object detection

The goal of object detection is to find objects of interest in an image or a video. This is more complex than image classification.

Object detection models return the bounding boxes of each object of interest in an image as well as confidence scores of these objects to belong to a certain category.

Object detection & Classification

Convolutional network

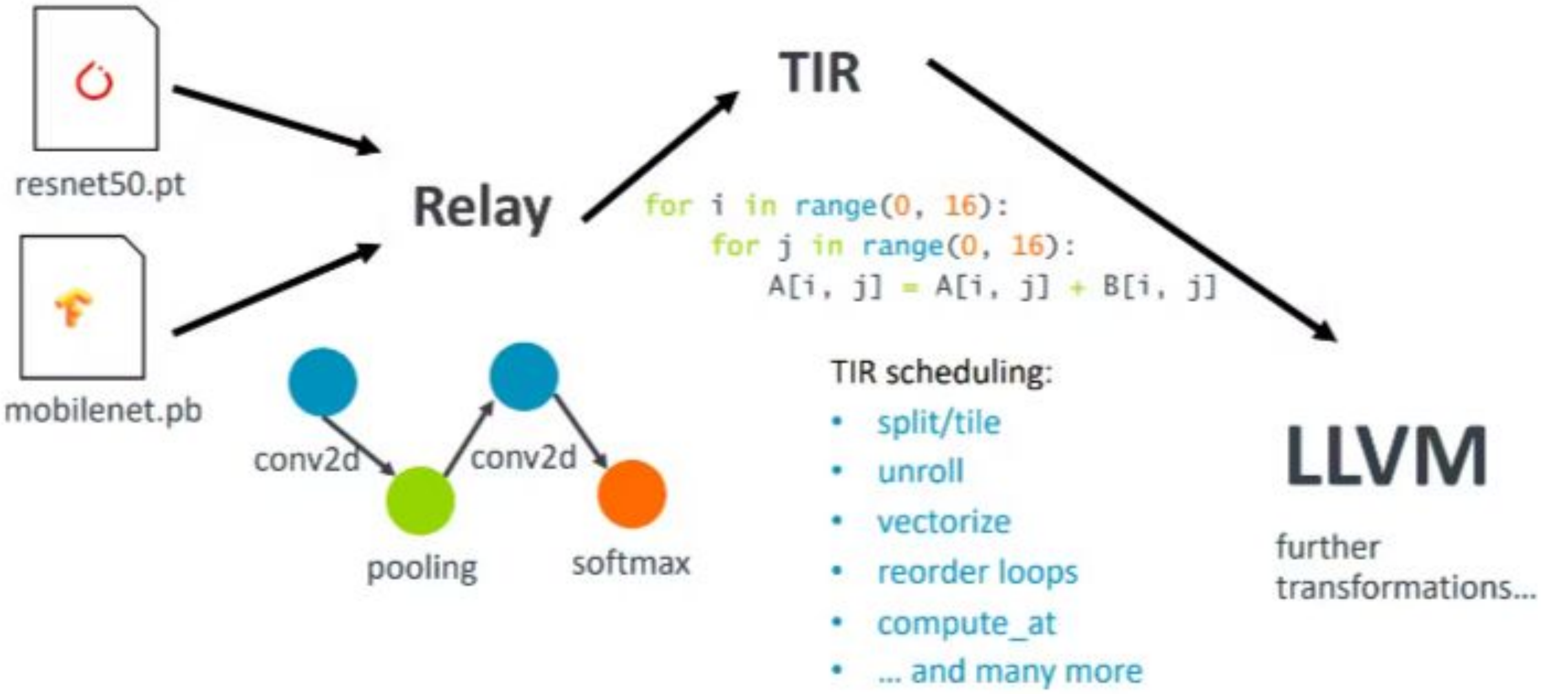


Apache TVM

TVM Architecture

Image credit : TVM Project

TVM stack


resnet50.pt
mobilenet.pb
Relay
conv2d
pooling
conv2d
softmax
TIR
for i in range(0, 16):
for j in range(0, 16):
A[i, j] = A[i, j] + B[i, j]
TIR scheduling:
• split/tile
• unroll
• vectorize
• reorder loops
• compute_at
• … and many more
LLVM
further transformations…

Jupyter Notebooks

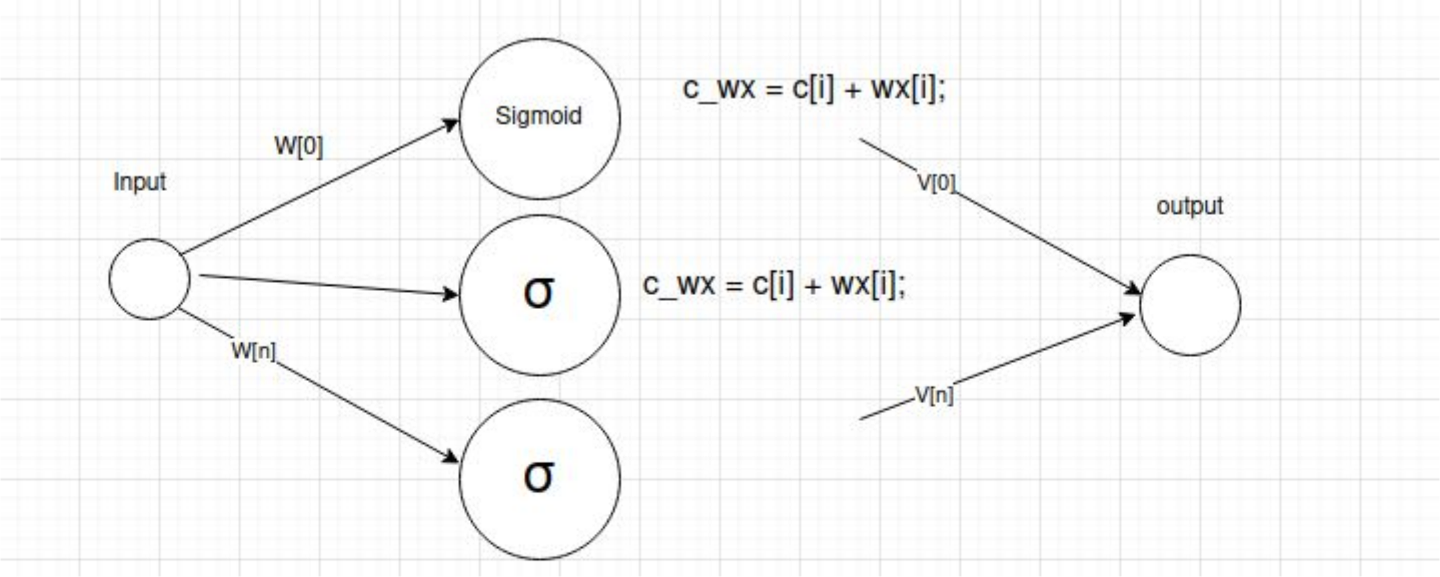
01.01-Getting-Started-with-Python-and-Jupyter-Notebooks.ipynb

Deep Mind
https://magenta.tensorflow.org/

- MobileNet V3 inference notebook : https://github.com/imadelh/Object-Detection_MobileNetv3-EfficientDet-YOLO/blob/master/artifacts/mobilenet_v3_example.ipynb
- Yolov3 inference notebook : https://github.com/imadelh/Object-Detection_MobileNetv3-EfficientDet-YOLO/blob/master/artifacts/yolov3_example.ipynb
- EfficientDet inference notebook : https://github.com/imadelh/Object-Detection_MobileNetv3-EfficientDet-YOLO/blob/master/artifacts/efficientdet_example.ipynb

```
      Input Layer         Hidden Layer (N neurons)          Output Layer
     ┌─────────┐         ┌──────────────────────────┐         ┌──────────┐
     │         │         │                          │         │          │
     │         │  W[0]   │  c[0] + W[0]*x           │  V[0]   │          │
     │         ├────────►  (sigmoid activation)     ├──────────►         │
     │         │         │                          │         │          │
     │         │  W[1]   │  c[1] + W[1]*x           │  V[1]   │  Σ(V[i]  │
     │   x     ├────────►  (sigmoid activation)     ├──────────► *h[i])+ │
     │         │         │                          │         │   b      │
     │         │   ...   │        ...               │   ...   │          │
     │         │         │                          │         │          │
     │         │  W[N-1] │  c[N-1] + W[N-1]*x       │  V[N-1] │          │
     │         ├────────►  (sigmoid activation)     ├──────────►         │
     └─────────┘         └─────────────────────────┘          └──────────┘
```

Simple sin example



Simple sin example

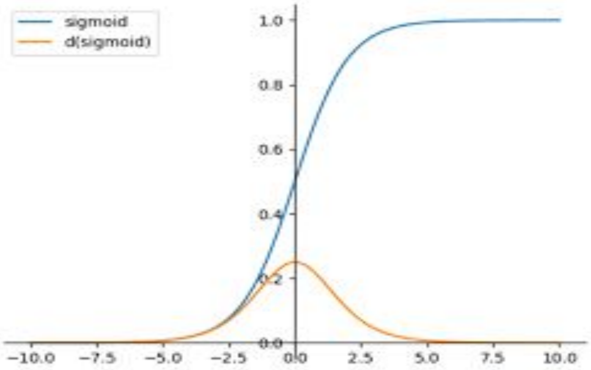
```
// Neural network function
shared_ptr<Variable> f_theta(shared_ptr<Variable> x)
{
   auto result = b;
   for (int i = 0; i < N; i++)
   {
       auto wx = W[i] * x;
       auto c_wx = c[i] + wx;
       auto sig = sigmoid(c_wx);
       result = result + V[i] * sig;
   }
   return result;
}
```

Training/validation split

```
vector<pair<shared_ptr<Variable>, shared_ptr<Variable>>> startSet;
for (int i = 0; i < SAMPLES; i++)
{
   double x_val = i * 2 * PI / SAMPLES;
   double y_val = sin(x_val) + 0.05 * (1.0 * rand() / RAND_MAX - 1.0);
   startSet.emplace_back(
       make_shared<Variable>(x_val),
       make_shared<Variable>(y_val));
}

// Splits
int TRAIN_SPLIT = static_cast<int>(0.8f * SAMPLES);
int TEST_SPLIT = static_cast<int>(0.2f * SAMPLES) + TRAIN_SPLIT;

vector<pair<shared_ptr<Variable>, shared_ptr<Variable>>>
              trainSet(startSet.begin(), startSet.begin() + TRAIN_SPLIT);
vector<pair<shared_ptr<Variable>, shared_ptr<Variable>>>
              testSet(startSet.begin() + TRAIN_SPLIT, startSet.begin() + TEST_SPLIT);
```

Activation function

```
shared_ptr<Variable> sigmoid(shared_ptr<Variable> x)
{
   auto result = make_shared<Variable>(1.0 / (1.0 + exp(-x->data)));
   result->parents = {x};
   result->backward_fn = [=]()
   {
       x->grad += result->data * (1 - result->data) * result->grad;
   };
   return result;
}
```

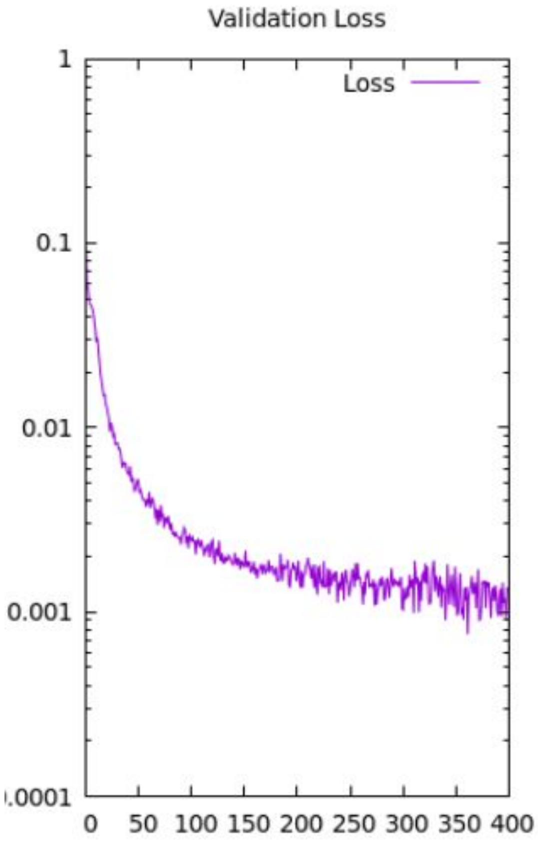


$\sigma(x) = 1/(1 + e^{-x})$

$\sigma(x)' = \sigma(x)\,(1 - \sigma(x))$

Training loops, Epochs

```
// Training loop
for (int j = 0; j < epoch; j++)
{
   shuffle_data(trainSet);
   double total_loss = 0.0;
   double validation_loss = 0.0;

   for (auto &[x, y] : trainSet)
   {
       train(x, y);
       total_loss += pow(f_theta(x)->data - y->data, 2);
   }

   for (auto &[x, y] : testSet)
   {
       validation_loss += pow(f_theta(x)->data - y->data, 2);
   }
}
```



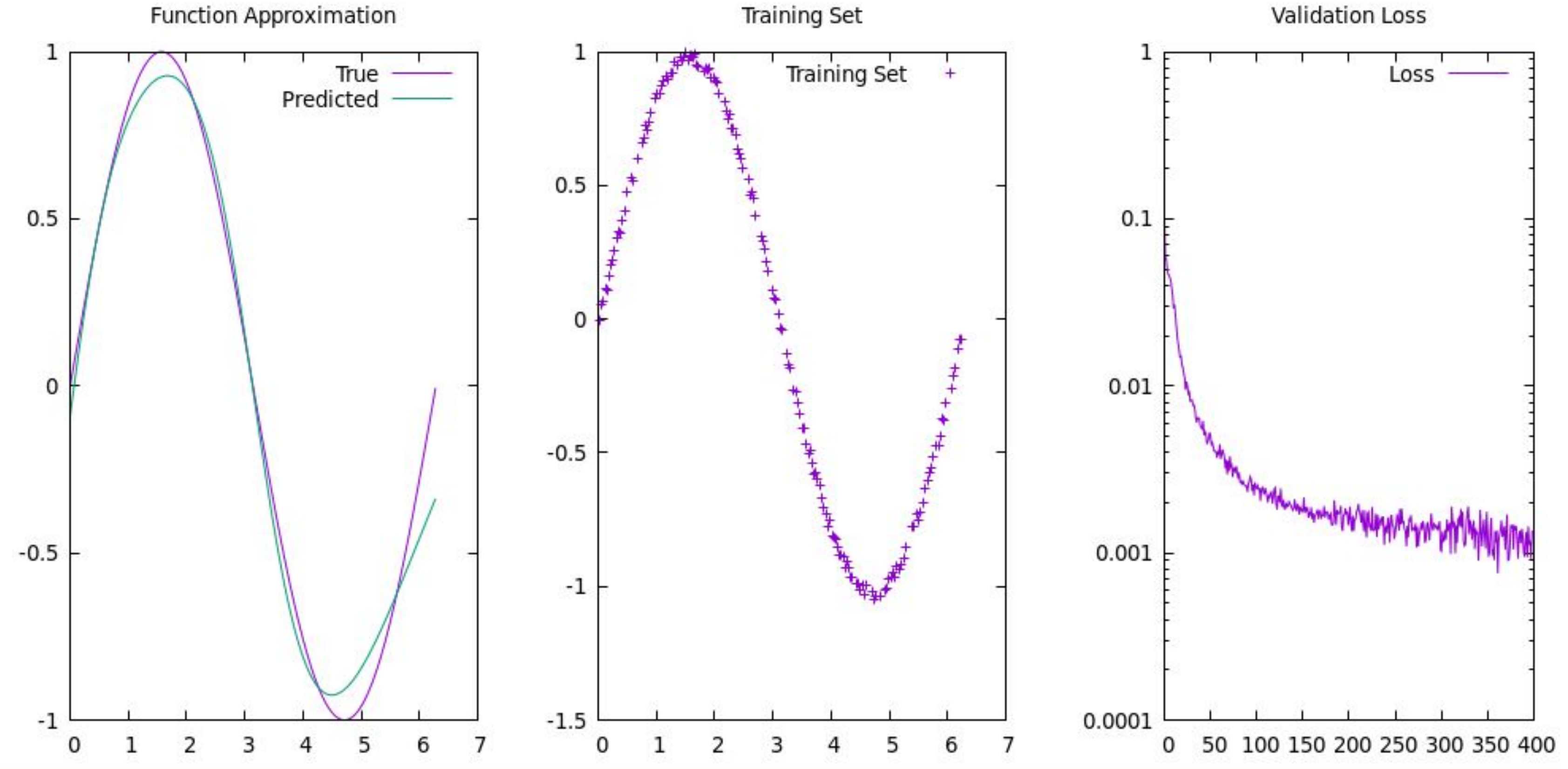
Train with backpropagation

```
// Training function
void train(shared_ptr<Variable> x, shared_ptr<Variable> y)
{
    auto pred = f_theta(x);
    auto error = pred - y;
    auto loss = error * error;

    loss->backward();

    // Update parameters
  …
}
```

Train with backpropagation

```
// Update parameters
for (int i = 0; i < N; i++)
{
    W[i]->data -= epsilon * W[i]->grad;
    V[i]->data -= epsilon * V[i]->grad;
    c[i]->data -= epsilon * c[i]->grad;
}
b->data -= epsilon * b->grad;
```

Simple sin example



Sin in pytorch

PyTorch

Sin in pytorch

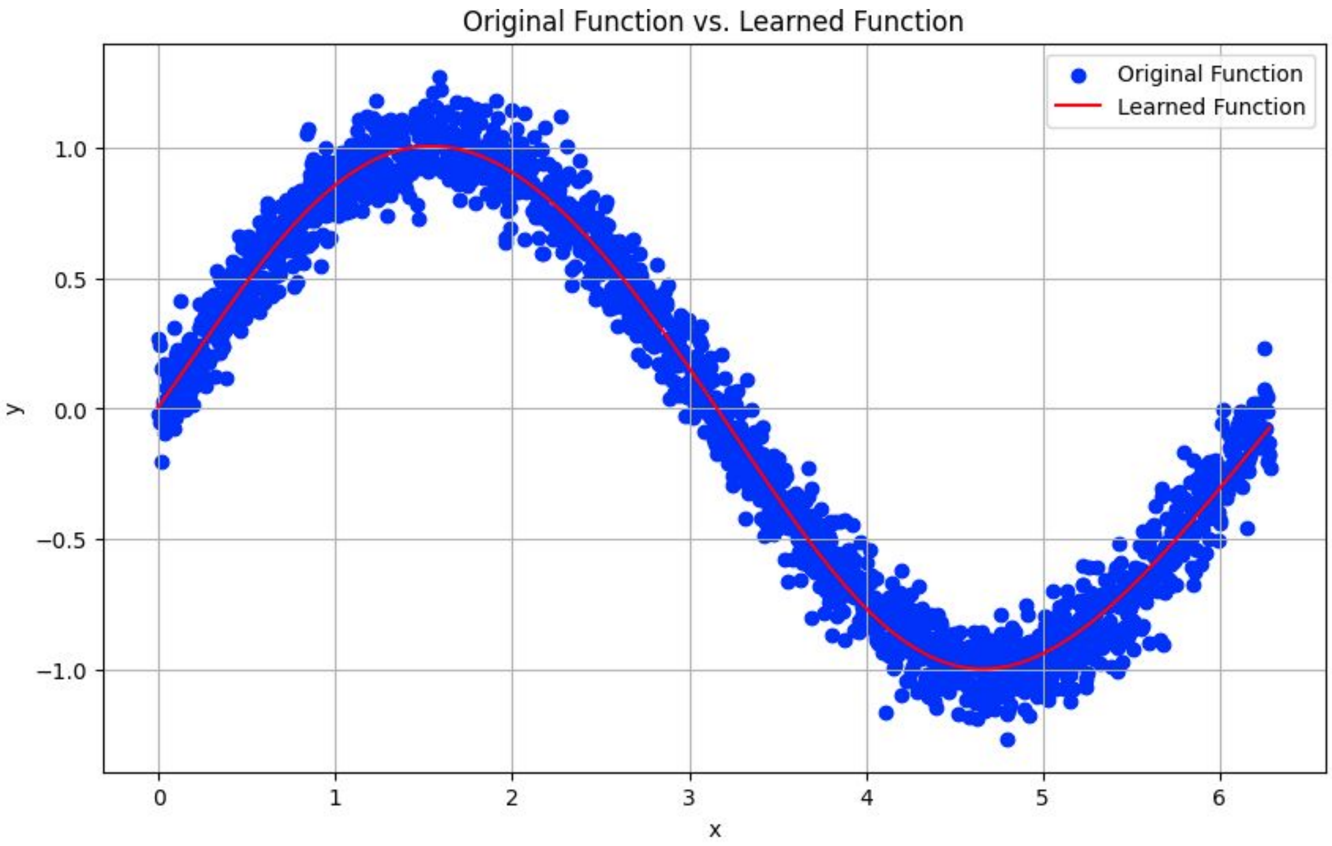
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # One hidden layer of size 16
        self.hidden1 = nn.Linear(1, 16)
        # Output layer
        self.output = nn.Linear(16, 1)

    def forward(self, x):
        # Apply sigmoid on the hidden layer
        x = F.sigmoid(self.hidden1(x))  # First hidden layer + sigmoid
        #x = F.sigmoid(self.hidden2(x))  # Second hidden layer + sigmaoid
        # Output layer (no activation for regression problems usually)
        x = self.output(x)
        return x
```

Sin in pytorch

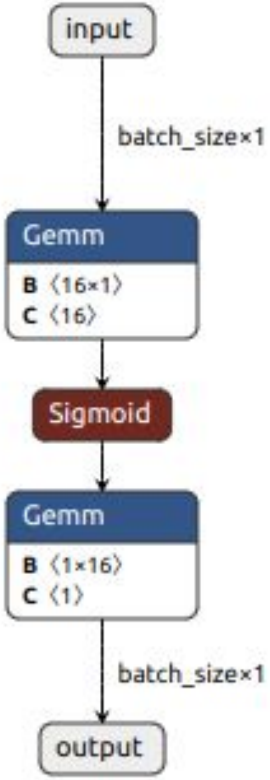
model = Net()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr = 0.001)

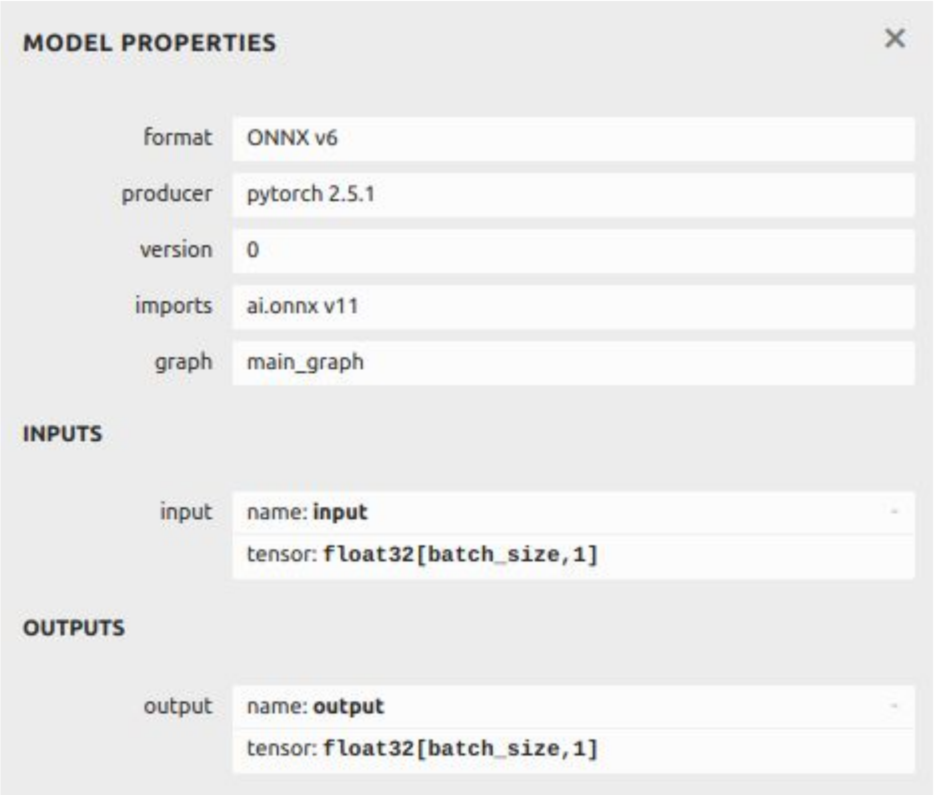
```
num_epochs = 7000
for epoch in range(num_epochs):
 #forward pass
 outputs = model(x_tensor)
 loss = criterion(outputs, y_tensor)
 optimizer.zero_grad()
 loss.backward()
 optimizer.step()

 if (epoch + 1) % 100 == 0:
   print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")
```

Sin in pytorch



Visualized with netron





Relay IR (apache TVM)

```
Relay IR:
def @main(%input: Tensor[(1, 1), float32] /* ty=Tensor[(1, 1), float32]
span=/hidden1/Gemm.input:0:0 */) -> Tensor[(1, 1), float32] {
  %0 = nn.dense(%input, meta[relay.Constant][0] /* ty=Tensor[(16, 1), float32]
span=/hidden1/Gemm.hidden1.weight:0:0 */, units=16) /* ty=Tensor[(1, 16), float32]
span=/hidden1/Gemm:0:0 */;
  %1 = add(%0, meta[relay.Constant][1] /* ty=Tensor[(16), float32]
span=/hidden1/Gemm.hidden1.bias:0:0 */) /* ty=Tensor[(1, 16), float32]
span=/hidden1/Gemm:0:0 */;
  %2 = sigmoid(%1) /* ty=Tensor[(1, 16), float32] span=/Sigmoid:0:0 */;
  %3 = nn.dense(%2, meta[relay.Constant][2] /* ty=Tensor[(1, 16), float32]
span=/output/Gemm.output.weight:0:0 */, units=1) /* ty=Tensor[(1, 1), float32]
span=/output/Gemm:0:0 */;
  add(%3, meta[relay.Constant][3] /* ty=Tensor[(1), float32] span=/output/Gemm.output.bias:0:0
*/) /* ty=Tensor[(1, 1), float32] span=/output/Gemm:0:0 */
}
```

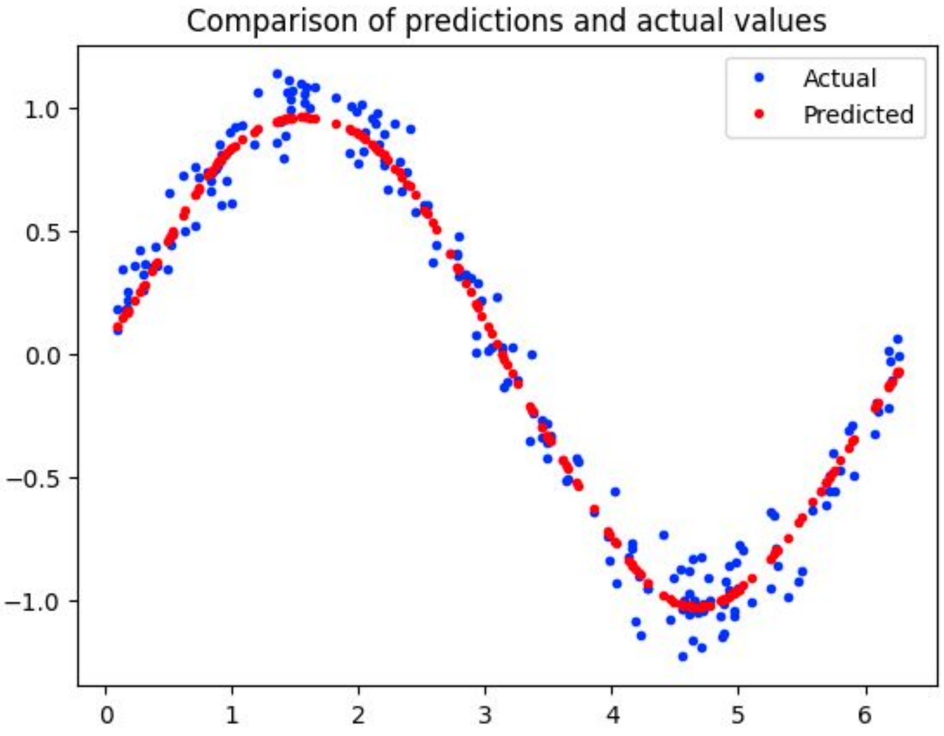
Sin in tensorflow

Sin in tensorflow

```
from tensorflow.keras import layers # Import layers here
model_2 = tf.keras.Sequential()

# First layer takes a scalar input and feeds it through 16 "neurons". The
# neurons decide whether to activate based on the 'relu' activation function.
model_2.add(layers.Dense(16, activation='sigmoid', input_shape=(1,)))

# The new second layer may help the network learn more complex representations
model_2.add(layers.Dense(16, activation='sigmoid'))

# Final layer is a single neuron, since we want to output a single value
model_2.add(layers.Dense(1))

# Compile the model using a standard optimizer and loss function for regression
model_2.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Sin in tensorflow

Results



Comparison of predictions and actual values
Actual
Predicted
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5
6

Sin in tensorflow (TVM TIR)

```
Converting TFLite model to Relay IR...
def @main(%serving_default_keras_tensor_38:0: Tensor[(1, 1), float32] /*
span=serving_default_keras_tensor_38:0:0:0 */, %v_param_1: Tensor[(16, 1), float32] /*
span=sequential_8_1/dense_18_1/MatMul:0:0 */, %v_param_2: Tensor[(16), float32] /*
span=sequential_8_1/dense_18_1/Add/ReadVariableOp:0:0 */, %v_param_3: Tensor[(16, 16), float32]
/* span=arith.constant1:0:0 */, %v_param_4: Tensor[(16), float32] /* span=arith.constant3:0:0 */,
%v_param_5: Tensor[(1, 16), float32] /* span=arith.constant:0:0 */, %v_param_6: Tensor[(1), float32] /*
span=arith.constant2:0:0 */, output_tensor_names=["StatefulPartitionedCall_1_0"]) {
  %0 = reshape(%serving_default_keras_tensor_38:0, newshape=[-1, 1]) /*
span=sequential_8_1/dense_18_1/MatMul;sequential_8_1/dense_18_1/Add:0:0 */;
  %1 = nn.dense(%0, %v_param_1, units=16) /*
span=sequential_8_1/dense_18_1/MatMul;sequential_8_1/dense_18_1/Add:0:0 */;
  %2 = nn.bias_add(%1, %v_param_2) /*
span=sequential_8_1/dense_18_1/MatMul;sequential_8_1/dense_18_1/Add:0:0 */;
  %3 = sigmoid(%2) /* span=sequential_8_1/dense_18_1/Sigmoid:0:0 */;
```

Sin in tensorflow (TVM TIR)

```
  %4 = reshape(%3, newshape=[-1, 16]) /*
span=sequential_8_1/dense_19_1/MatMul;sequential_8_1/dense_19_1/Add:0:0 */;
  %5 = nn.dense(%4, %v_param_3, units=16) /*
span=sequential_8_1/dense_19_1/MatMul;sequential_8_1/dense_19_1/Add:0:0 */;
  %6 = nn.bias_add(%5, %v_param_4) /*
span=sequential_8_1/dense_19_1/MatMul;sequential_8_1/dense_19_1/Add:0:0 */;
  %7 = sigmoid(%6) /* span=sequential_8_1/dense_19_1/Sigmoid:0:0 */;
  %8 = reshape(%7, newshape=[-1, 16]) /* span=StatefulPartitionedCall_1:0:0:0 */;
  %9 = nn.dense(%8, %v_param_5, units=1) /* span=StatefulPartitionedCall_1:0:0:0 */;
  nn.bias_add(%9, %v_param_6) /* span=StatefulPartitionedCall_1:0:0:0 */
}
```

Sin in tensorflow