Valhalla - Flattened type is a primary requirement for compact representation of tensors and vectors.



primitive class Element {

float elem;

}

primitive class Vector512 {

Element lane0;

Element lane1;

….

Element lane15;

}

primitive class Tensor2x512 {

Vector512 row0;

Vector512 row1;

}

Either as bundle of scalars or as one Tensor unit if target supports tiles.

List various operations in tensorflow and its equivalent implementation using Java Vector API.

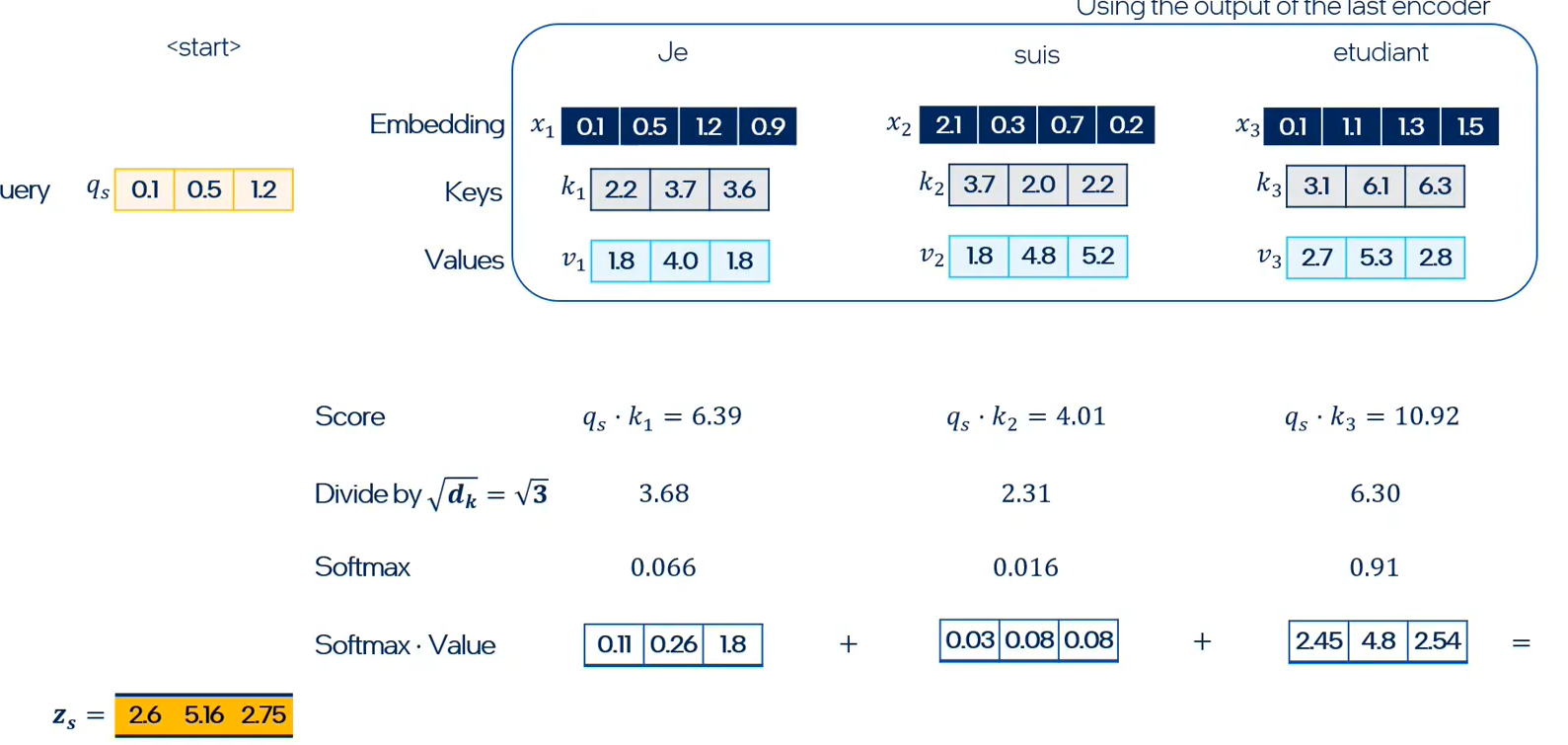
Define a model using Vector API, check tribuo.

<https://github.com/oracle/tribuo>

<https://github.com/PaulSandoz/blis-matrix>

Transformer Notes:-

Self-Attention



Transformer operations in various layers.

* Dot product
* Divide by constant.
* Vector Addition
* Cross entropy – transdental log function
* Matrix multiplication (linear layer)
* Normalization (clippings)
* Vector multiplication

Parameters: Weight matrices ( Query, Key, Value)

Embeddings -> Position Encoding -> Time annotated Embedding ->

Encoder Stack (Self Attention -> Contextual embedding -> FF N/W -> Normalization)

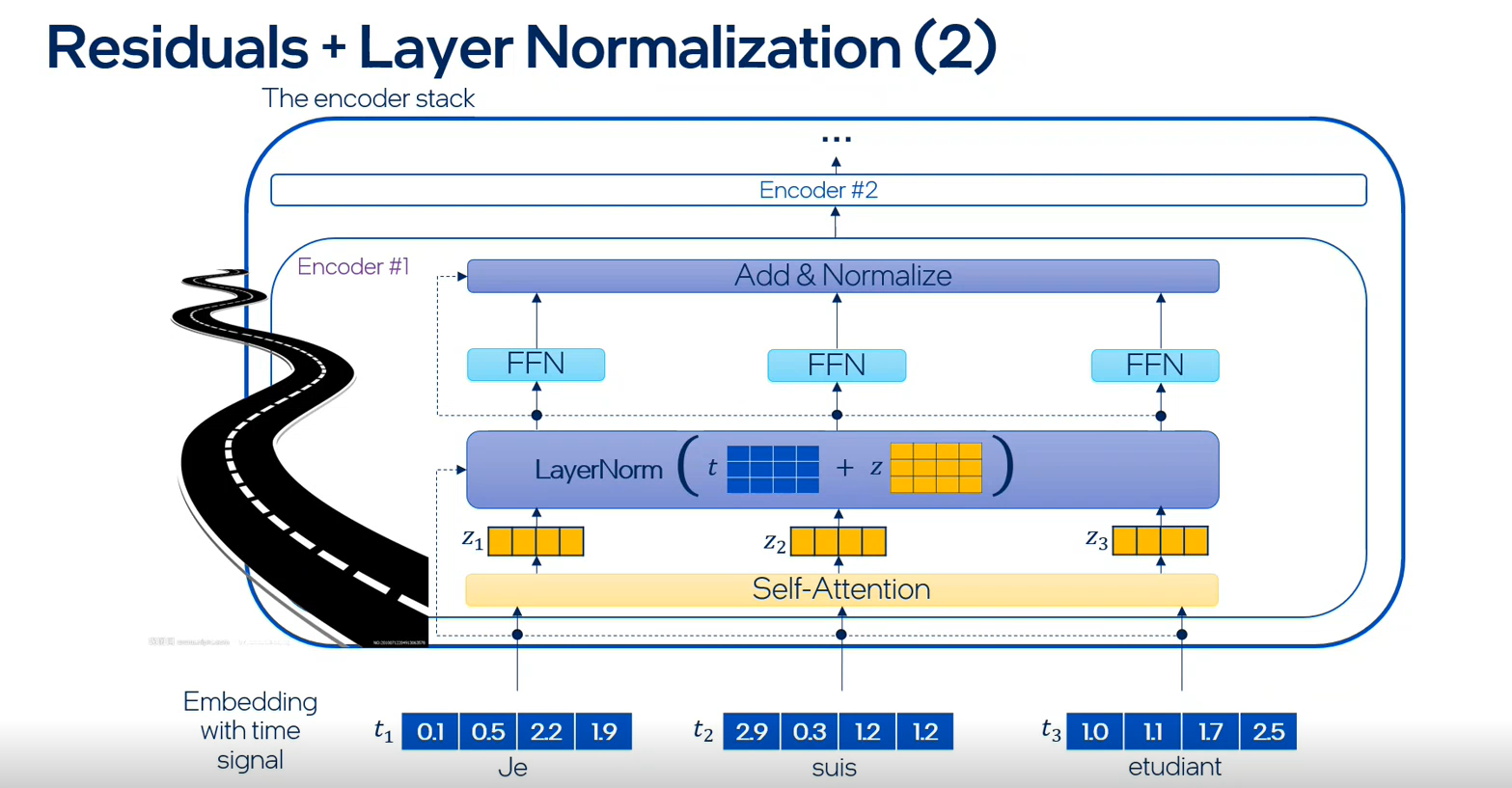
Decode Stack (Encoder – Decode Attention (Input Embedding + previous output) )

Linear layer (matmul) -> softmax (probabilities) -> Loss function - > Gradients (Back propagation)

Vanishing gradients problem similar to RNNs , fixed using skip layer routes, in RNNs LSTM / GRU counters this.

A computer screen shot of a diagram

Description automatically generated



A computer screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Java Vector API: -

Exercise 1:-

Identify various primitive operations in CNN / RNN / Transformers and develop a JMH benchmark for each and compare its performance against equivalent Tensorflow operators.

Exercise 2:-

Write gradient descent and stochastic gradient descent with varying batch size algorithms in Java.

Exercise 3 : -

1. Develop each CNN layer (Input, Hidden (Convolution, Pooling, Fully connected) Soft max-Output) using Java Vector API for MNIST data set.
2. Compare its performance against equivalent Tensor flow implementation.
3. Extend vector API to support custom APIs for matrix multiplication (AMX\_FP16 GNR, AMX\_BF16 SPR, AMX\_INT8)

Low precision math support for INT4 INT8, FP8, FP16 in Java Vector API.

PPML – Privacy protected ML

Training: -

Federated machine learning – Horizontal vs Vertical

Horizontal – Gradient computation done locally and propagated to parameter servers. Parameter / gradients are stored in secured enclaves using SGX. Communication b/w edge devices and servers is encrypted.

<https://cczoo.readthedocs.io/en/latest/index.html>

Inferencing: -

Accept pre-trained models (ONNX formats)

Edge inferencing acceleration using [graph compilers](https://deci.ai/blog/graph-compilers/) (quantization, various backends CPU / GPU/ NPU / VPU).

Runtime dispatches optimized models / split operations to various XPUs for inferencing.

Deeplearning4j -> nd4j -> BaseNDArray.java

<https://github.com/deeplearning4j/deeplearning4j/blob/master/nd4j/nd4j-backends/nd4j-api-parent/nd4j-api/src/main/java/org/nd4j/linalg/api/ndarray/BaseNDArray.java>

Optimize with Vector API MemorySegment backing storage.

Interesting Conversation (Adam Pocock Oracle, Tribuo)

[https://matrix.to/#/#tensorflow\_sig-jvm:gitter.im](https://matrix.to/#/)

[https://matrix.to/#/#tensorflow\_sig-jvm:gitter.im](https://matrix.to/#/)

<https://medium.com/apache-mxnet/speed-up-your-bert-inference-by-3x-on-cpus-using-apache-tvm-9cf7776cd7f8>

<https://d2l.djl.ai/>

RNN types

1 : N -

N : 1 – sentiment analysis

N : M – text generation

Prominent vector databases for AI

* JVector:- Java
  + integrated into AstraDB.
* Quadrant – Rust
* Milvus - Go

A screenshot of a computer

Description automatically generated

A screenshot of a software

Description automatically generated

Frameworks : Tensorflow , pytorch, mxnet, dlj (under the hood uses various compute engines using native bindings to existing popular frameworks), deeplearning4j,

Tensorflow

Tf.constant -> creates an immutable tensor instance, and push these through the operator graphs which modifies these as per the operator semantics. Tf.numpy() transforms these immutable tensors to mutable numpy arrays.

Notes on LLMs, VectorDB, RAG, Inferencing Engines etc..: -

* JVector is a vector search engine integrated into AstaraDB and Cassendra.
* AstaraDB and Cassendra community have added a new vector type, using which user can added a DB column for vector embeddings.

VectorDB augments the KB (knowledge base) of existing LLMs which are initially trained on specific set of inputs, this allows extending LLMs based predictions over foreign knowledge bases.

VectorDB is data base of vector embeddings, Vector search is different form regular keyword search which looks for an exact match in the indexed database, this can be called a shallow search. Vector search on the other hand perform similarity / deep search by computing the distance between query vector embedding and vector DB. Distance computation may employ different techniques like DOT product, Euclidean distance, cosine distance.

This framework where pre-trained LLMs uses Vector Databases to augment the predictions is know is retrieval augmented generation or RAG.

**TODO: Develop a RAG model using Langchain which uses Jlama (Java based inferencing engine) and AstraDB / Cassendra as a vector database. Both these databases rely on JVector for deep searching.**

Both **Jvector and Jlama are making heavy use of Java VectorAPI for accelerating distance computation, product quantization and inferencing**.

Jlama accept pretrained model form Hugging Face in SafeTensor format.

SafeTensor is a format to capture tensorflow graph which represents the data flow of tensors though various operators. These operators could either be activation function, LSTM / GRU (for deep neural networks) , or high level layers (convolution, FC /Dense). LLVM MLIR is another popular serialized IR for model computation graph.

ONNX is model exchange format.

Inferencing engines which embed AI compilers (Apache TVM , Intel ngraph, XLA etc) receive these pre-trained models, apply target independent and dependent optimization over it and then generates optimized code for various XPUs (CPU, GPU, NPU, VPU).

Inferencing engines make direct use of IHV optimized libraries like Intel’s oneDNN, Nvidia cuDNN to generate optimized code for various operators.

Advanced Deep learning models

* CNN
  + Layered model with input, series of hidden layer and an output layer.
  + Graph based CNN a variation of CNN.
* RNN
  + Stateful learning, counters vanishing gradients by inserting LSTM / GRU layer with forget / gating function.
* Reinforcement learning
  + Agent interacts with environment, selects an action to maximize rewards.
  + Q-function vs Policy gradient.

Generative AI Models

1 ) VAE

Encoder -> Latent space -> Decoder

Used for generating vector embeddings used in Vector data bases.

2) GAN

Generator -> [ ] -> Disseminator

Generator generates new random data and discriminator discriminates b/w real and fake data.

Both are indipendnt models, fed with mixed real / fake date.

Eventual goal is to remove discriminatory post training and allow generator to generate new real like samples.

3 ) Diffusion models

Two stage process

* Noising – introduce noise in initial input in each successive time step.
* De-noising – Takes noisy data at various timestep and inject VAE b/w them to generate original sample.

1. Transformers
   * Self-attention, stacked encoder .

Supervised

* Regression – Single vs multivariate regression, optimizes SSE (sum of squared errors) .
  + Advanced regressions include regularization function to prevent overfitting by capping the parameter. Lasso and ridge regularization.
* Classification
  + Advanced classification using Support vector machines.
    - Efficiently capture non-linearity in the inputs, consist of SVM kernel to transform non-linear input space into linear space.

Unsupervised

* Clustering.

Ensemble and decision trees.

* XGBoost

ANN, CNN , RNN and deep neural N/W models as discussed above….

**DSE AI Seminar series:;**

Michels’ and Sergay’s presentation on Ollama and RAG

**Meeting notes**

Q By taking a manual embedding route we can try with different seed embeddings which may eventually impact the inference time.

Q Is it a good practice to use same embedding model for training and inferencing.

A. Obviously yes, because model parameters are learned i.e. BPTT is sensitive to embeddings, to compute the gradiant both input and outputs are translated into vector embeddings and the loss is computed using vector embeddings of actual and predicted output