Multivariable chain rule:

In general:

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

**t1= 0.01**

**w5 = 0.40**

**o1**

**h1**

**i1**

**i1= 0.05**

**w1 = 0.15**

**w2 = 0.20**

**w7 = 0.50**

**w6 = 0.45**

**i2= 0.10**

**t2= 0.99**

**w3 = 0.25**

**o2**

**h2**

**i2**

**w8 = 0.55**

**W4 = 0.30**

**b2b**

**b1**

**b1 = 0.35**

**b2 = 0.60**

Activation function:

Derivative of sigmoid:

Loss function:

Forward pass:

Input matrix: A = **[i1 i2]**

Weight matrix: X =

Bias Matrix : B1 = **[b1 b1]**

**[h1\_in h2\_in]** = A . X + B

**[h1\_out h2\_out] = [(h1\_in) (h2\_in)]**

Weight matrix: Y =

Bias Matrix : B2 = **[b2 b2]**

**[o1\_in o2\_in] = [h1\_out h2\_out] . Y + B2**

**[o1\_out o2\_out] = [(o1\_in) (o2\_in)]**

Backward Pass:

Target = **[t1 t2]**

d(MSE)/d(o1\_out) = o1\_out – t1

d(MSE)/d(o2\_out) = o2\_out – t2

**[d(MSE)/d(o1\_out) d(MSE)/d(o2\_out)] = [o1\_out o2\_out] – [t1 t2]**

d(MSE)/d(o1\_in) = (d(MSE)/d(o1\_out)) \* (d(o1\_out)/d(o1\_in))

= (d(MSE)/d(o1\_out)) \* (o1\_out) \* (o1\_out - 1)

d(MSE)/d(o2\_in) = (d(MSE)/d(o2\_out)) \* (d(o2\_out)/d(o2\_in))

= (d(MSE)/d(o2\_out)) \* (o2\_out) \* (o2\_out - 1)

**[d(MSE)/d(o1\_in) d(MSE)/d(o2\_in)] = [d(MSE)/d(o1\_out) \* o1\_out \* (o1\_out-1) d(MSE)/d(o2\_out)\* o2\_out \* (o2\_out-1)]**

d(MSE)/d(w5) = (d(MSE)/d(o1\_in)) \* (d(o1\_in)/d(w5)) = (d(MSE)/d(o1\_in)) \* h1\_out

d(MSE)/d(w6) = (d(MSE)/d(o1\_in)) \* (d(o1\_in)/d(w6)) = (d(MSE)/d(o1\_in)) \* h2\_out

d(MSE)/d(w7) = (d(MSE)/d(o2\_in)) \* (d(o2\_in)/d(w7) = (d(MSE)/d(o2\_in)) \* h1\_out

d(MSE)/d(w8) = (d(MSE)/d(o2\_in)) \* (d(o2\_in)/d(w8)) = (d(MSE)/d(o2\_in)) \* h2\_out

Multivariable Chain Rule

d(MSE)/d(h1\_out) = d(MSE)/d(o1\_in) \* (d(o1\_in)/d(h1\_out)) + d(MSE)/d(o2\_in) \* (d(o2\_in)/d(h1\_out))

= d(MSE)/d(o1\_in) \* w5 + d(MSE)/d(o2\_in) \* w7

d(MSE)/d(h2\_out) = d(MSE)/d(o1\_in) \* (d(o1\_in)/d(h2\_out)) + d(MSE)/d(o2\_in) \* (d(o2\_in)/d(h2\_out))

= d(MSE)/d(o1\_in) \* w6 + d(MSE)/d(o2\_in) \* w8

Weight matrix: Y =

**[d(MSE)/d(h1\_out) d(MSE)/d(h2\_out)] = [d(MSE)/d(o1\_in) d(MSE)/d(o2\_in)] \* YT**

d(MSE)/d(h1\_in) = (d(MSE)/d(h1\_out)) \* (d(h1\_out)/d(h1\_in)) = (d(MSE)/d(h1\_out)) \* (h1\_out) \* (h1\_out - 1)

d(MSE)/d(h2\_in) = (d(MSE)/d(h2\_out)) \* (d(h2\_out)/d(h2\_in)) = (d(MSE)/d(h2\_out)) \* (h2\_out) \* (h2\_out - 1)

**[d(MSE)/d(h1\_in) d(MSE)/d(h2\_in)] = [d(MSE)/d(h1\_out) \* h1\_out \* (h1\_out-1) d(MSE)/d(h2\_out)\* h2\_out \* (h2\_out-1)]**

d(MSE)/d(i1\_out) = d(MSE)/d(h1\_in) \* (d(h1\_in)/d(i1\_out)) + d(MSE)/d(h2\_in) \* (d(h2\_in)/d(i1\_out))

= d(MSE)/d(h1\_in) \* w1 + d(MSE)/d(h2\_in) \* w3

d(MSE)/d(i2\_out) = d(MSE)/d(h1\_in) \* (d(h1\_in)/d(i2\_out)) + d(MSE)/d(h2\_in) \* (d(h2\_in)/d(i2\_out))

= d(MSE)/d(h1\_in) \* w2 + d(MSE)/d(h2\_in) \* w4

Weight matrix: X =

**[d(MSE)/d(i1\_out) d(MSE)/d(i2\_out)] = [d(MSE)/d(h1\_in) d(MSE)/d(h2\_in)] \* XT**

How to train a DNN model with billions of parameters and trillions of training examples.

Use thousands of machines. Each machine contains multiple high-end GPUs.

How to compute in parallel.

1. Data Parallelism
   1. Each worker has a local copy of the model.
   2. Data is partitioned among all workers.
   3. Each worker computes its local gradients.
   4. The local gradients are sent to a parameter server.
   5. The parameter server aggregates the results and sends them back to workers.
   6. Workers update their weights and go to step c.
2. Model Parallelism
   1. GPT4 uses 1.76 trillion parameters. If each parameter is 4 bytes, the size of the model is 1639 GB. The maximum storage of Nvidia A100 is 80 GB. Therefore, the entire model can’t fit into the GPU.
   2. Partition the model layer-wise, e.g., each GPU/worker owns a given partition, and the data is processed in a pipelined manner.
      1. Challenges: A GPU would be idle unless the GPU responsible for the previous layer finishes its execution. Divide the data into chunks. After a chunk is processed by layer i, it is ready to be processed by layer i+1. Spilt the model so that each split roughly takes the same time. The parallelism would be similar to hardware pipelining you might have studied in the computer organization course.
   3. Another technique is to partition the model horizontally, e.g., split all layers into multiple halves. Each worker is assigned a partition. Challenges: Additional aggregate operation may be required to combine the outputs of the partitioned layers. Split the layer wisely to minimize the overhead of aggregation.

Other challenges include optimizing the bandwidth and time required for communication. For example, O(n) bandwidth is required for the parameter server model, where n is the number of workers.

Dataflow programming

A dataflow program is described by a directed graph where the nodes denote operations and arcs denote data dependencies.

The data is encapsulated into a token.

Token <dest\_addr, value, port>

A node may execute when a token is available on each input arc. After execution, the token is removed from the input arc, and the resulting token is placed on the output arc.

Arcs implements FIFO queues to store the tokens.

x = a \* b

y = 4 \* c

return (x+y) \* (x-y)/c

Imperative style vs. dataflow.

Dataflow advantage:

1. Nodes may execute in parallel unless there is an explicit data dependence between them
2. The result doesn’t depend on the order in which the potentially parallel nodes execute

A dataflow graph is well-behaved if, for a given sequence of input, the output is produced in the same sequence.

Conditional and loop constructs can be implemented using switch and merge operators.

The switch operation takes one value token and a Boolean input token. It has two output ports, True and False. The input token is forwarded to the True or False port depending on whether the condition is true or false.

The merge has two input ports for data, True and False, and it also takes on a Boolean input token. It has one output port. If the Boolean input is true, the token at the True port is forwarded to the output. Otherwise, the token at the False port is forwarded to the output.

if (x < y)

r = x + y

else

r = x – y