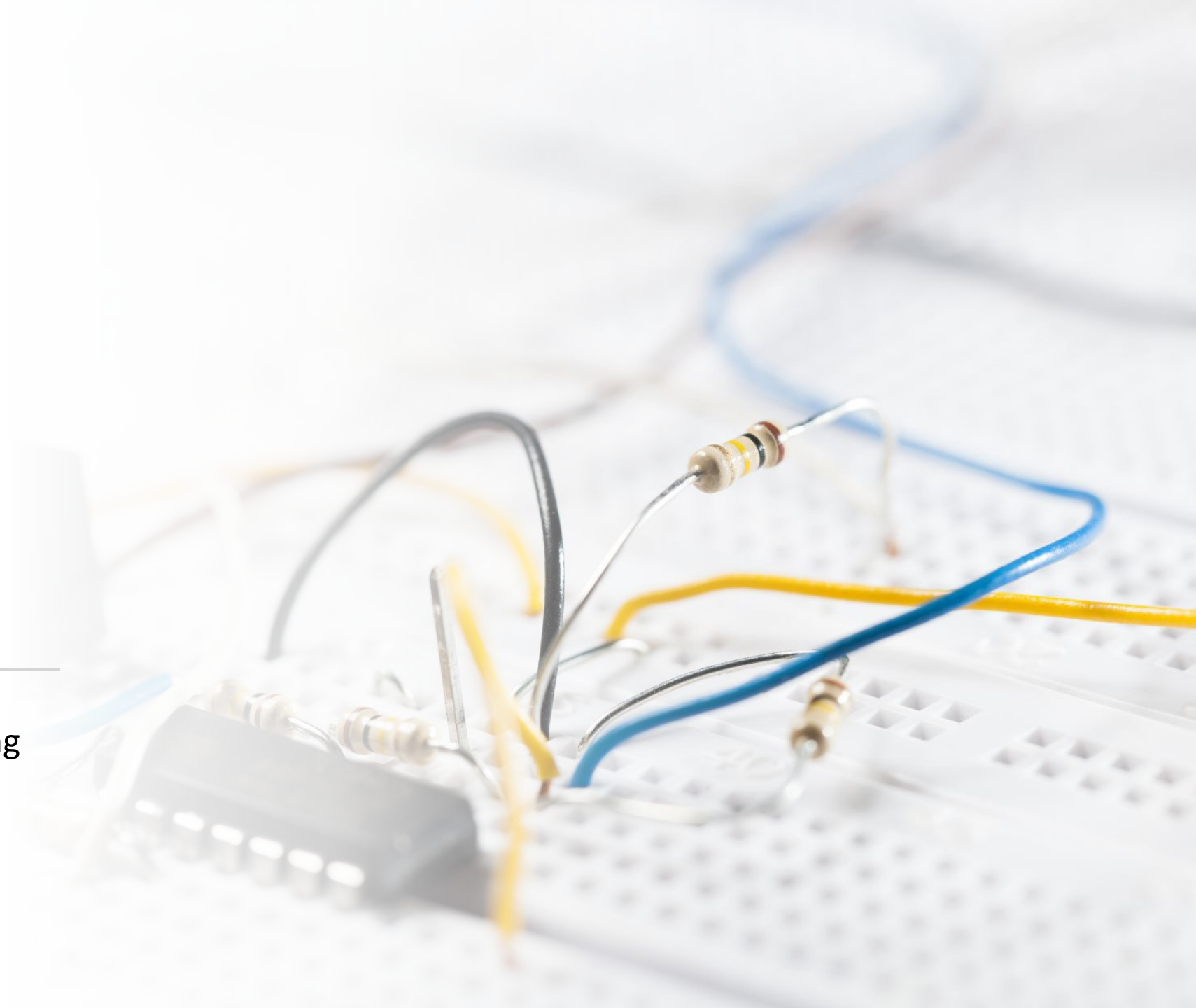
Using Analog Computer for Neural Network

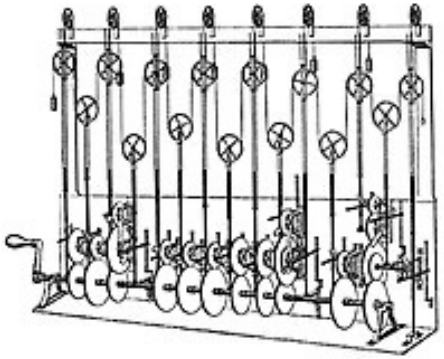


--Use case: simulating with training resnet18 on ImageNet data

Binwei Yan

Basic concept

- Digital computers represent varying quantities symbolically and by discrete values of both time and amplitude.
- **Analog computer** use continuous variation aspect of physical phenomena such as electrical



A tide-predicting machine invented by Lord Kelvin in 1872

EAI 8800 Analog computer in 1986

.

Pros and Cons

LIMITATION	BENEFIT
Noise!	Energy efficient
Not general	Fast
Obsolete?	Computing in Memory

Suitable for machine learning!

General Architecture

Analog computation units (integrators, differentiators, adders, multipliers, etc.)

Analog storage units (long-term storage oxide layer charge traps, short-term storage potential wells)

Interconnections between storage and computation units (initial hard connections via poly, rewritable soft connections via floating gate MOS)

Restructuring units (real-time alteration of soft connections between storage and computation units based on a reward mechanism)

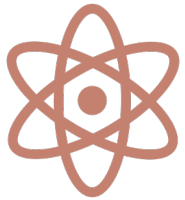
Image and audio input/output units (analog acquisition, storage, indexing)

Initialization units (digital storage and translation, hard connection generation, initial reward mechanism generation, operation guarantee system)

Computing Architecture!

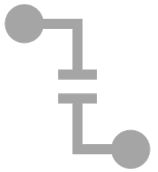
Analog computation units

(integrators, differentiators, adders, multipliers, etc.)



Analog storage units

(long-term storage oxide layer charge traps, short-term storage potential wells)

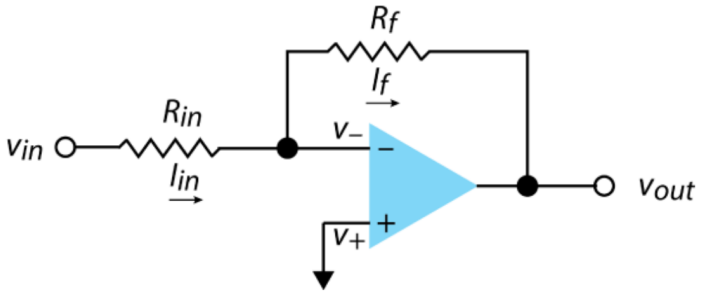


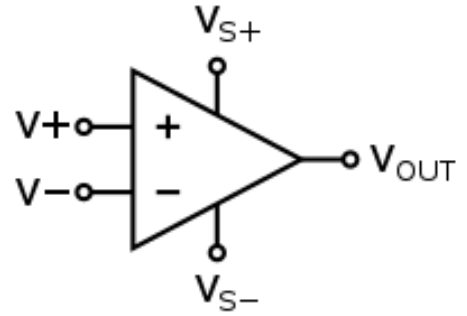
Interconnections between storage and computation units

(initial hard connections via poly, rewritable soft connections via floating gate MOS)

Analog computation units

- Op-amp: A is large, around 1e5
- Can be used as addition, subtraction, Integration and differentiation, log, and exponent





$$V_{\text{out}} = A_{\text{OL}}(V_+ - V_-),$$

- Multiplication and division of input are second primitive operations

Basic Unit of Analog NN

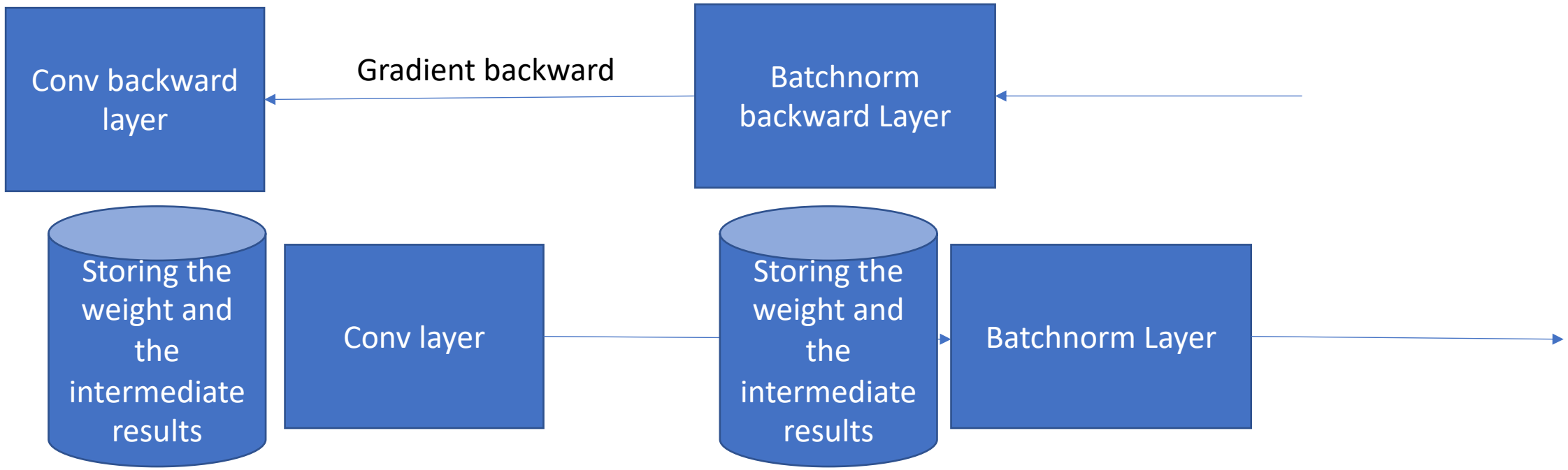
- A layer of Neural network will be considered as a block
- Mostly not very complicated operations
- Fully Connected Layer
- Batchnorm Layer
- Convolution Layer
- Activation Layer
- Attention Layer
- Neural Network is basically linear!

Compute Unit with memory stored together

- Intuition: Compute in memory with tensor operations
- Each layer operation is associated with some memory (capacitance) to store the intermediate result
- Paging the memory by index (Don’t need too complex paging!)
- Each Layer is equipped with a back propagation layer
- Before training, wire the modules and deploy all the parameters! (Harvard Architecture)

Compute Unit with memory stored together

- Before training, wire the layers in the sequential order
- During training, save the intermediate result to the capacitances



Change of algorithm

- Error rate of at most 1e-3 for all the parameters!
- Ways to accelerate:
- Bigger batch size, the calculation is more effective by sequential computing:
- Accelerate even faster!
- Don’t wait for all the loss of a batch!
- The gradient descent step comes from batch to be in the middle of the training
- (i.e. when the time the gradient is calculated to a block, gradient descent is done simultaneously)
- No waiting time for gradient descent!

Julia code!

```
function ResidualNetwork(model_size = 18; in_channels = 3, classes = 1000)
    config = Dict(
        18=>((2, 2, 2, 2), BasicBlock, 1),
        34=>((3, 4, 6, 3), BasicBlock, 1),
        50=>((3, 4, 6, 3), Bottleneck, 4),
        101=>((3, 4, 23, 3), Bottleneck, 4),
        152=>((3, 8, 36, 3), Bottleneck, 4))

    model_size in keys(config) || throw(
        "Invalid mode size [$model_size]. Supported sizes are $(keys(config)).")

    repeats, block, expansion = config[model_size]
    stages_channels = _get_stages_channels(expansion)

    entry = Chain(
        Conv((7, 7), in_channels=>64, pad=(3, 3), stride=(2, 2), bias=false),
        BatchNorm(64, relu))
    pooling = MaxPool((3, 3), pad=(1, 1), stride=(2, 2))

    head = nothing
    if classes ?~I? nothing
        head = Chain(MeanPool((7, 7)), Flux.flatten, Dense(512 * expansion, clas
ses))
    end

    in_channels = 64
    channels = (64, 128, 256, 512)
    strides = (1, 2, 2, 2)

    layers = []
    for (out_channels, repeat, stride) in zip(channels, repeats, strides)
        push!(layers, make_layer(
            block, in_channels=>out_channels, repeat, expansion, stride))
        in_channels = out_channels * expansion
    end
```

Change of performance using Resnet18 and ImageNet as an example

- With SGD:
- Score for original training: top class: 0.6930 5:0.8894
- Score for training with error: top class: 0.7078 top class 5:0.9013