Lecture 1: Introduction

Puneet Gupta

Some slides from Lei He

What does this course plan to cover ?

- Part I: Introduction to Neural Networks, operators, dataflows and exemplar ML accelerator architectures. (3 weeks)
- Part II: Neural network compression (2.5 weeks)
- Part III: Model training (distributed training, hardware-aware training, etc) (2 weeks)
- Part IV: Emerging computing models for AI (2.5 weeks)

- “TA”: Shurui Li
- We will be using Piazza and Gradescope

What this course is NOT

- Not a ML software/algorithms class (though we will briefly talk about it)
- Not a computer architecture class (though we will briefly talk about it in context of ML)
- Not a circuits class (though we will be assessing ML from a circuit standpoint)
- Focus is on ML-hardware interactions
- Expected background:
  - ECE216A or ECE201A or equivalent
  - Familiarity with Python and Pytorch/TensorFlow
  - Familiarity with Machine learning/neural networks (e.g., ECEM147)
  - ECEM116C (Computer Architecture) or equivalent

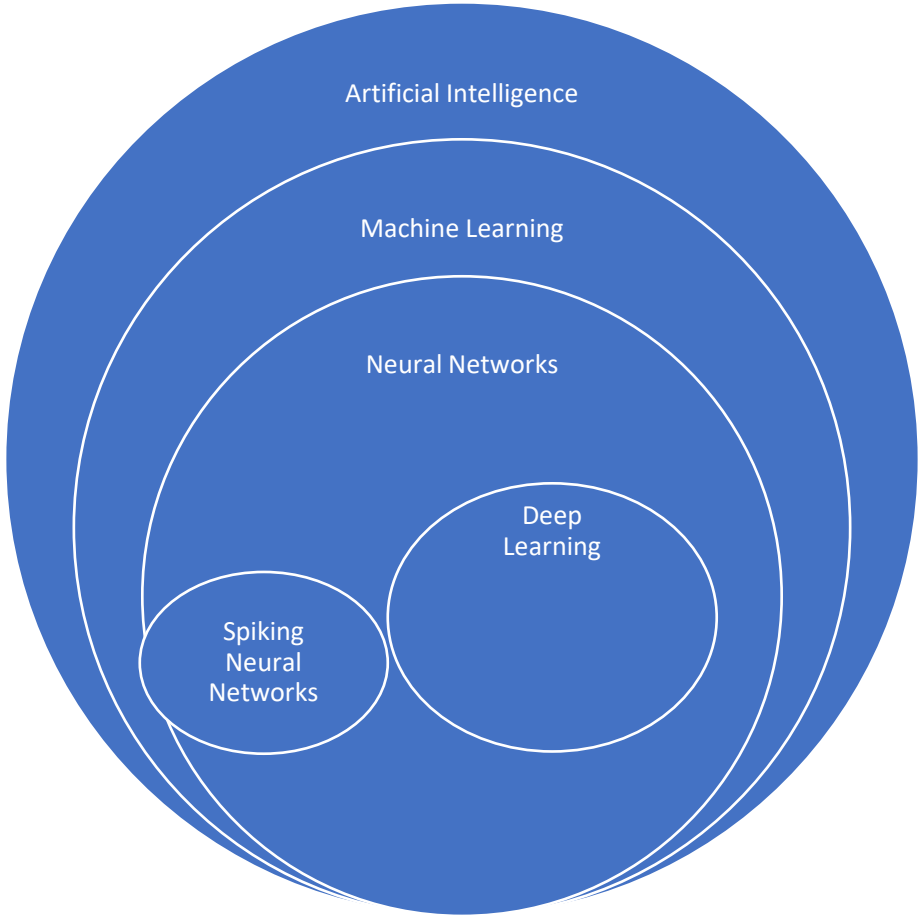
Grading

- Project 1 on NN operator synthesis tradeoffs: 25% (Verilog + SP&R)
- Project 2 on network compression: 25%
- Project 3 on hardware impact modeling of distributed training: 25%
- Paper presentations: 25%
  - 15+5= 20 minutes per presentation
  - 3 presentations per lecture starting week 2
  - Select a paper from an assigned reading list for every Part.
  - Student to week assignment: done randomly
- Projects in groups of two. Paper presentations individual.
  - Solo project bonus: 2% per project.
- Grading policy:
  - 90%+: A+
  - 80%+: A
  - 75%+: A-
  - 70%+: B+
  - 65%+: B
  - 60%+: B-

PTEs/ Class Enrollment

- Please make sure you have the requisite background
  - Please take the pre-requisite quiz
- Class is oversubscribed
- PTE policy:
  - Ph.D. students with needed background: get a PTE
  - M.S. students:
    - CES strongly preferred
    - Must know Verilog and Python (no exceptions)
    - Must have taken at least an undergraduate class in ML
    - Do not ask for a PTE unless you have taken most if not all suggested background classes (201A/216A/M147/M116C)

The Bigger Picture



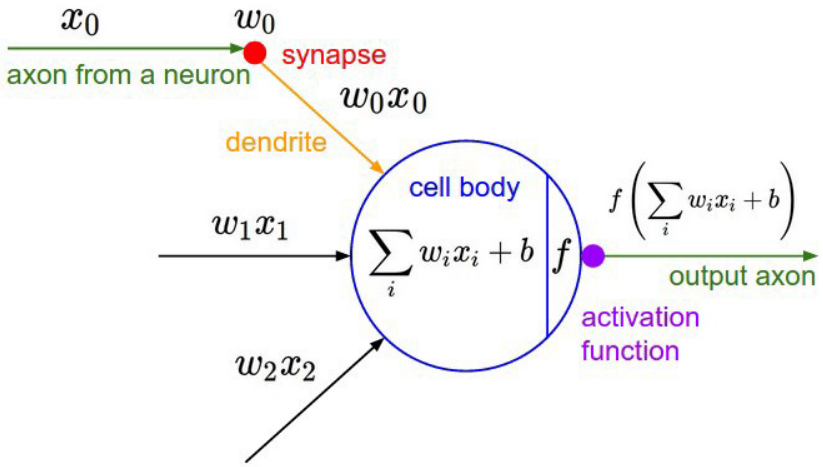
- AI: intelligent machines that behave like humans
- ML: computers that can “learn” without being explicitly “programmed” to do certain tasks
- NNs: Brain-inspired computational networks to do decision/classification/regression/generation tasks
- Spiking networks: special class of NNs where nodes integrate and fire asynchronously
- Deep learning: basis of modern ML where large, deep NNs are “trained” to do tasks

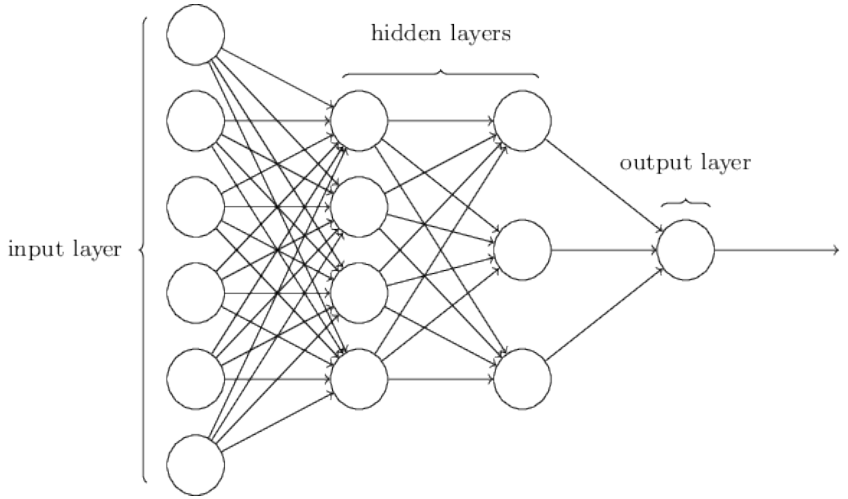
A Perceptron

- A typical decision problem = perceptron

$$y = \begin{cases} 0, & x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 < \text{threshold} \\ 1, & x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \geq \text{threshold} \end{cases}$$

  - Here “activation” function would be a step function (not really used anymore)
- Can extend to a multi-layer perceptron (MLP) neural network
  - A fully connected NN
- Training a NN:
  - Figure out weights w_i using optimization and a dataset of inputs
- Inference:
  - Use trained weights with new inputs





Training a Neural Network

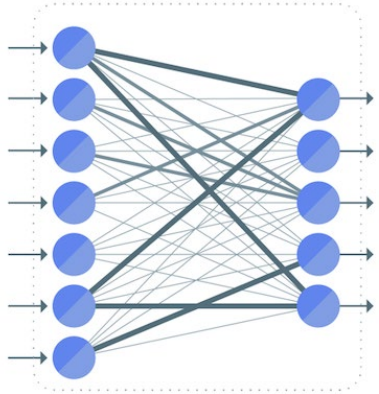
- Define a “loss” function (i.e., error of the neural network)
  - Mean square error
    $$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$
  - Cross-entropy loss
    $$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$
- Use gradient descent to optimize the weights
  - Calculate $\frac{\partial Loss}{\partial w_i}$ partial derivatives
    - On a single input sample or a “batch” of input samples averaged
    - Backpropagation (leverage calculus chain rule) to efficiently calculate the derivatives
  - Update the weights $w^{t+1} = w^t - \alpha \frac{\partial Loss}{\partial w_i}$
  - Iterate to minimize Loss
  - Wide variety of efficient algorithm implementations in ML frameworks such as Pytorch, TensorFlow, etc

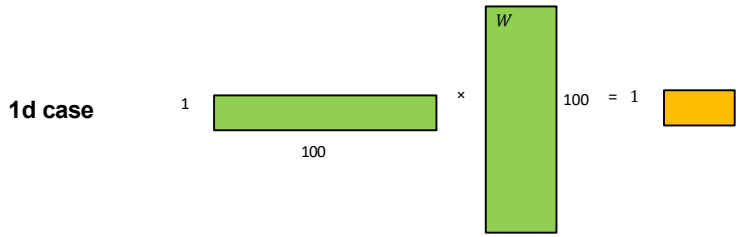
Common Layer Types in a DNN

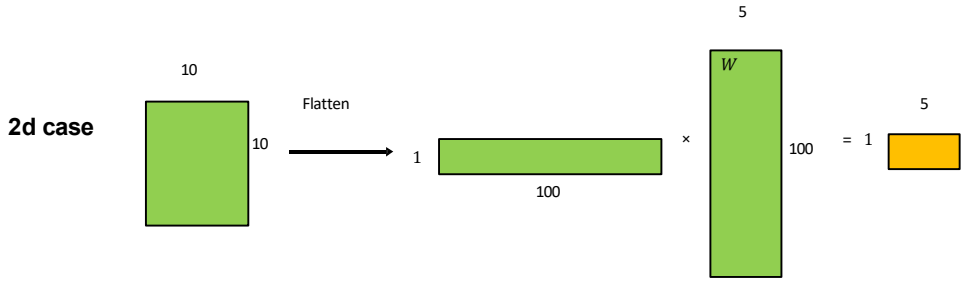
- Fully connected or FC layer (essentially a perceptron-like layer)
- Convolutional or CONV layer (1D/2D/3D filtering operations)
- Activation layer (provides the non-linearity)
- Pooling layer (downsampling number of outputs to the next layer by combining them)
- Normalization layer (to make sure inputs to next layer are normalized in magnitude)
- Residual layer (adds skip forward connections)
- Recurrent layer (provide memory; output depends on a previous output)
- Deconvolution layer (used to upsample)
- Attention layer (used to “focus” on a specific region in transformer networks)

FC Layers

- All values in a kernel have connections to all values in the feature map
  - Single neuron: $\sum_1^n w_i \; x_i \; + b$
  - A vector-vector dot-product
- A full layer (multiple neurons)
  - A vector-matrix multiply (VMM)
  - If there are m inputs and n outputs in a hidden layer, a fully connected layer would require O(mn) operations to compute the output.
- Note: every weight is multiplied to exactly one input! → no “reuse” of weight
- Imagine a CIFAR10 image dataset
  - Input 32x32x3 = 3072 → 3072 weight values need *per* neuron just for input layer → quickly becomes untenable due to memory requirements



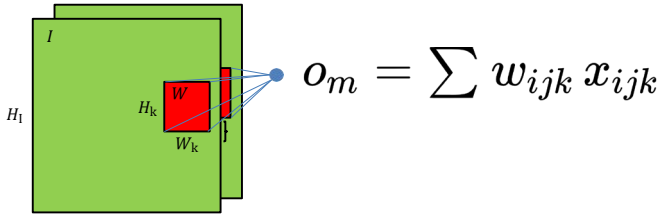




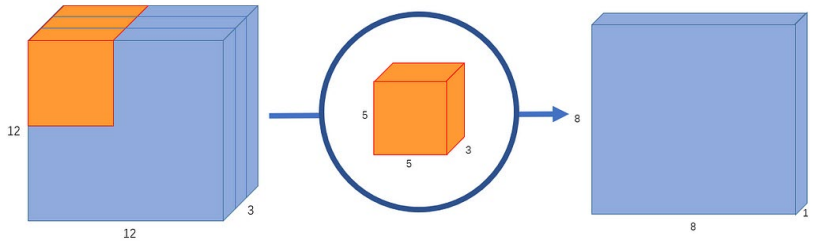
Quick question

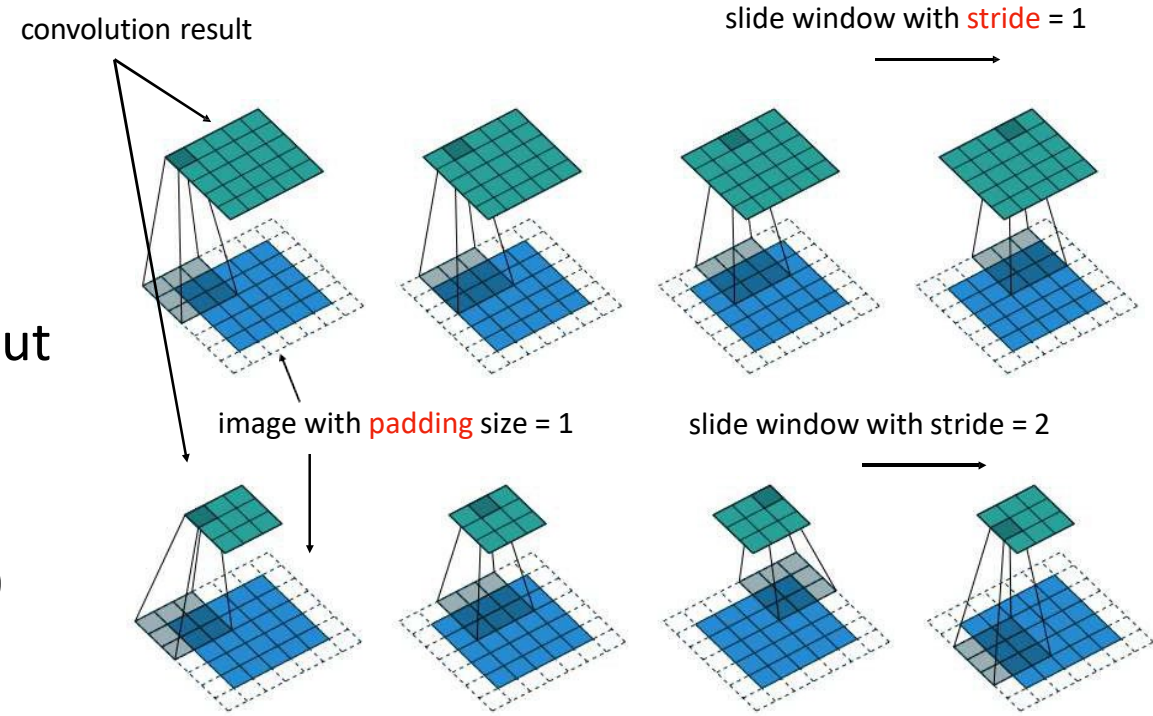
- A FC layer with *m* inputs and *n* outputs
  - What is the total # parameters ?
  - What is the total # of multiply-accumulate (MAC) operations done ?

CONV Layers



- Convolve the filter with the feature map
  - “slide over the image spatially, computing dot products”
  - Filters are usually “3D” but the z (channel) dimension is collapsed using a 3D dot product
  - Typically edges of input are “padded” (e.g., with zeroes)
- CONV layers need far fewer parameters to process same number of inputs
  - Filter size is much smaller (e.g., 5x5) than the input size (e.g., 200x200) → Every weight is “reused” several times (= number of outputs)
  - If each of n outputs is connected to only k inputs (w x h x d =k) then the layer would require O (kn) operations to compute the output.



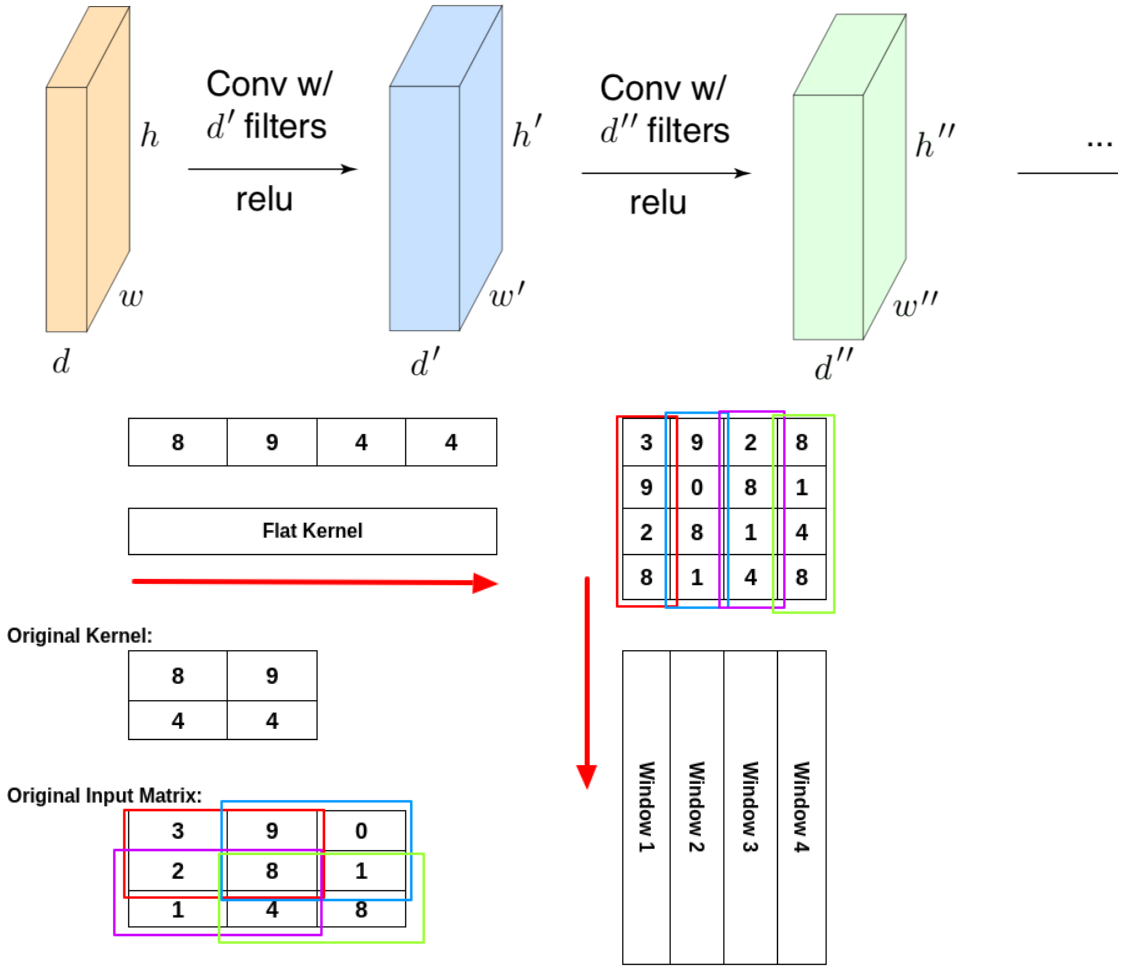


Quick questions

- A CONV layer with a *n x n* input and one *k x k* filter with stride 1
  - What will be the size of output if there is no padding ? With padding ?
  - What is the total number of trainable parameters ?
  - What is the total number of MACs (in the padding case) needed to compute all outputs ?

Multi-layer CONV structure

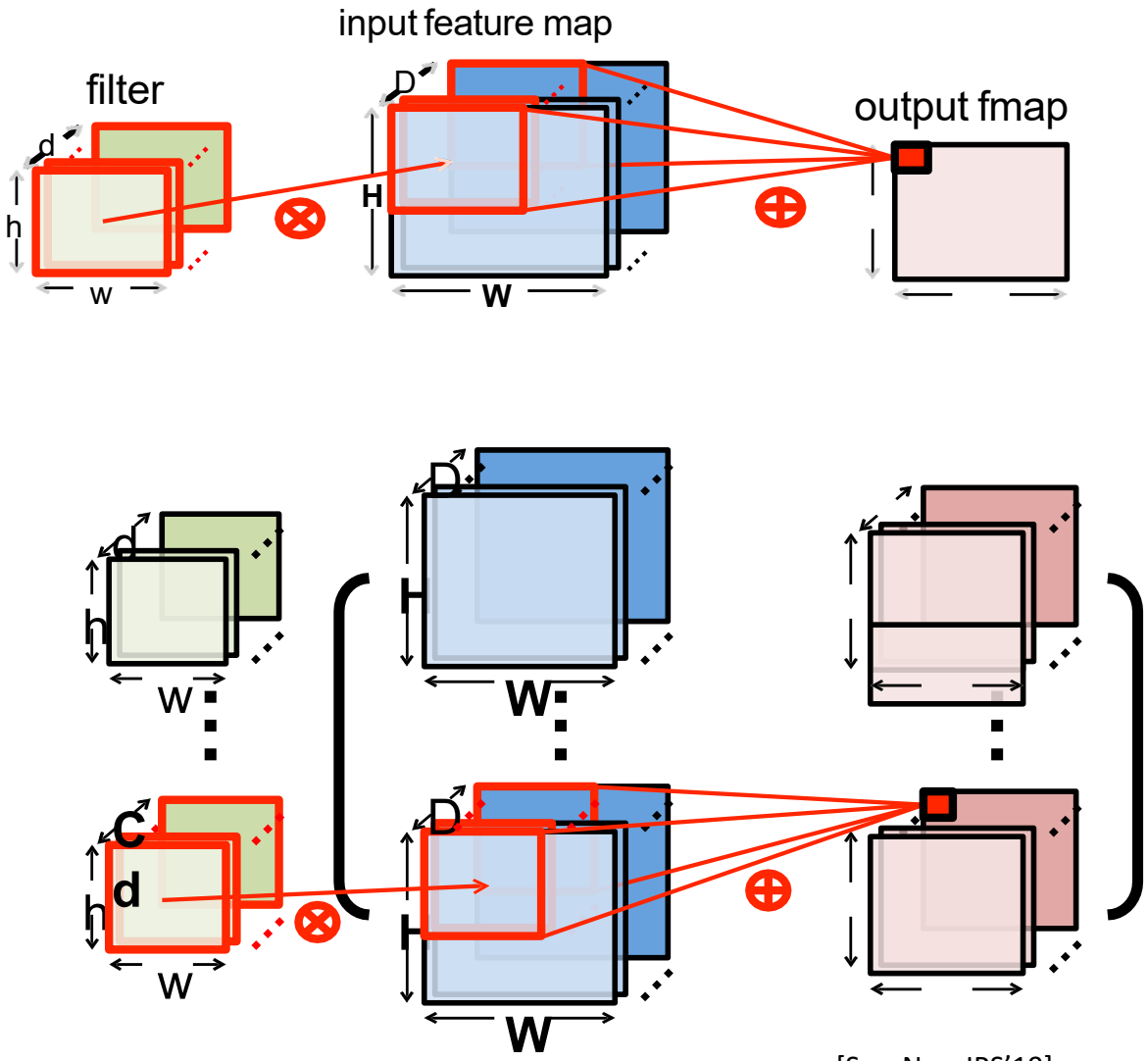
- Number of filters in current layers become number of input channels in the next layer
- Small filters → less memory, less compute → 3x3 filters are very popular
- How to get a large receptive field with small filter ?
  - Use more CONV layers
  - Other techniques such as dilated CONV
- "Im2Col" converts the CONV operation into a VMM
  - More input storage memory but far better parallelization
- Other ways of doing CONV
  - Explicit sliding window
  - FFT
  - Winograd transform
  - Most software libraries will pick the best implementation depending on input/filter size.



[Different implementations of the ubiquitous convolution | by Sundar Raman P | Medium]

High Dimensional CONV Processing

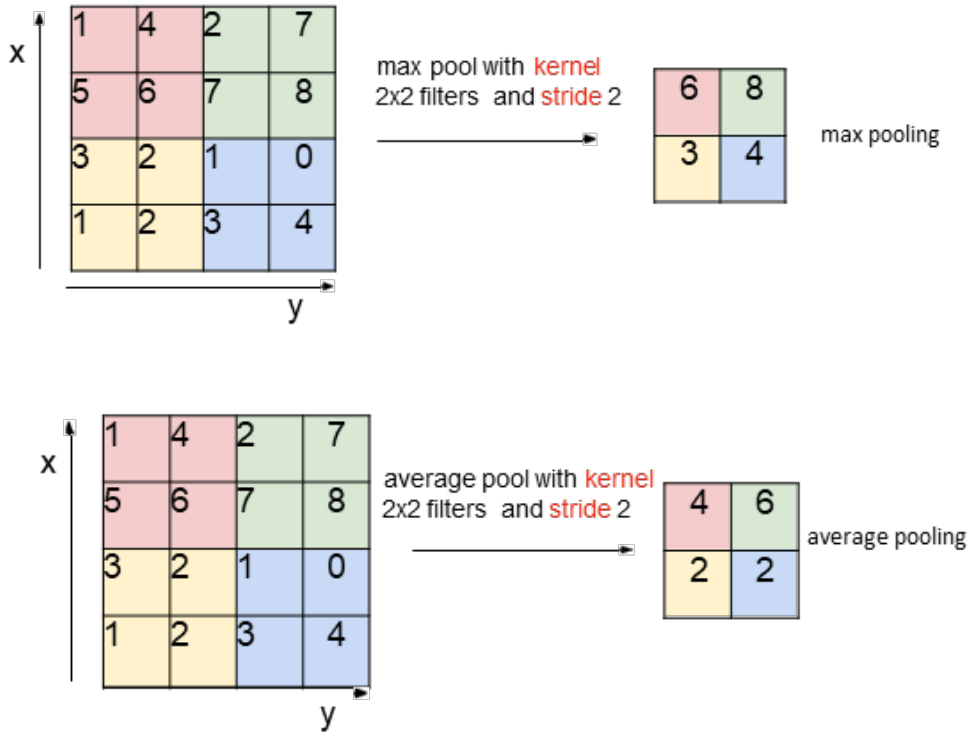
- Inputs/filter have many channels (z dimension), as high as 256 for some layers
- A CONV layer will have many filters
  - #filters = #output channels
- Input may have a “batch size” > 1 → multiple output feature maps are created.
- A FC layer is essentially a CONV layer with filter dimensions same as input dimensions



[Sze, NeurIPS’19]

Pooling/Downsampling

- Provide an approach to down sample feature maps by summarizing the presence of features in patches of the feature map.
- Two common approaches
  - Max-pool: subsampling by picking the largest
  - Average-pool: subsampling by computing the mean



Quick questions

- A CONV layer with 100 5x5x3 filters, 20x20x3 input feature map (assume padding) followed by a 2x2 max pool layer
  - What is the size and shape of the output ?

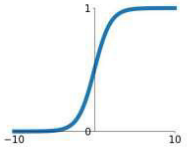
5 min break

Common Activation Functions

- Recall: f(output) = activation → goes to the next layer in the NN
- Key to make the network non-linear.
  - Otherwise no difference from linear regression!
  - ReLU is strictly not differentiable at 0 but its derivative assumed to be 0 at 0 (not a big issue)
- ReLU is very popular
  - Simple and fast to calculate derivative
  - Results in sparse, positive only activations which can help in hardware acceleration

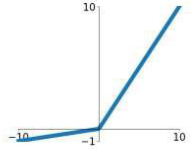
Sigmoid

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



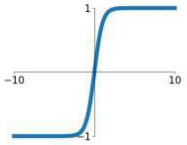
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

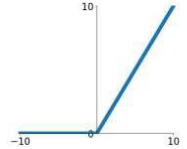


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

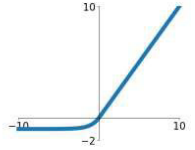
ReLU

$$\max(0, x)$$



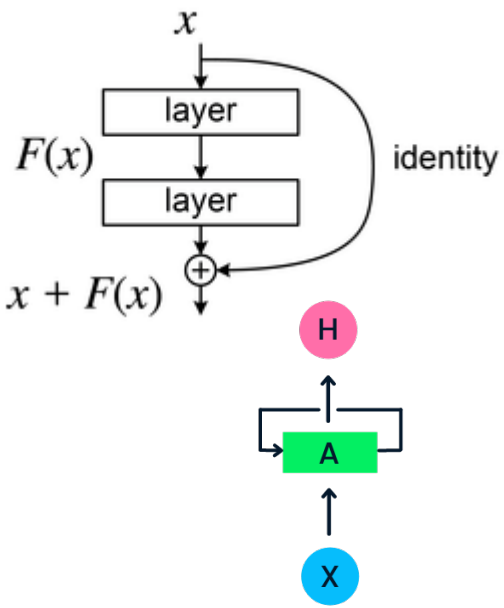
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Other layer types

- Residual block (used in ResNet): allows deeper networks to train better
- Recurrent layer: usually used for temporal inputs (e.g., natural language processing)
  - Have “memory”: take information from prior inputs to influence the current input and output.
- Attention layers: used in transformer models
- Batch normalization:
  - To improve training process by keeping inputs to all layers similarly distributed
- Softmax classification: $\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
  - an activation function that scales numbers into probabilities

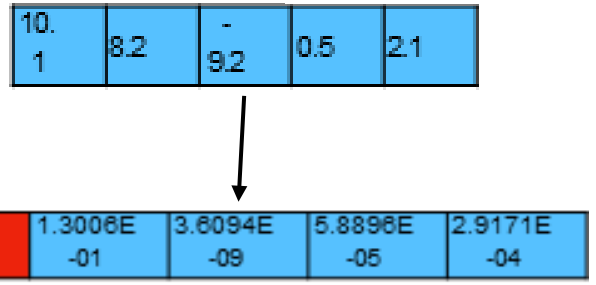


$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{Batch mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \quad \text{Batch variance}$$

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

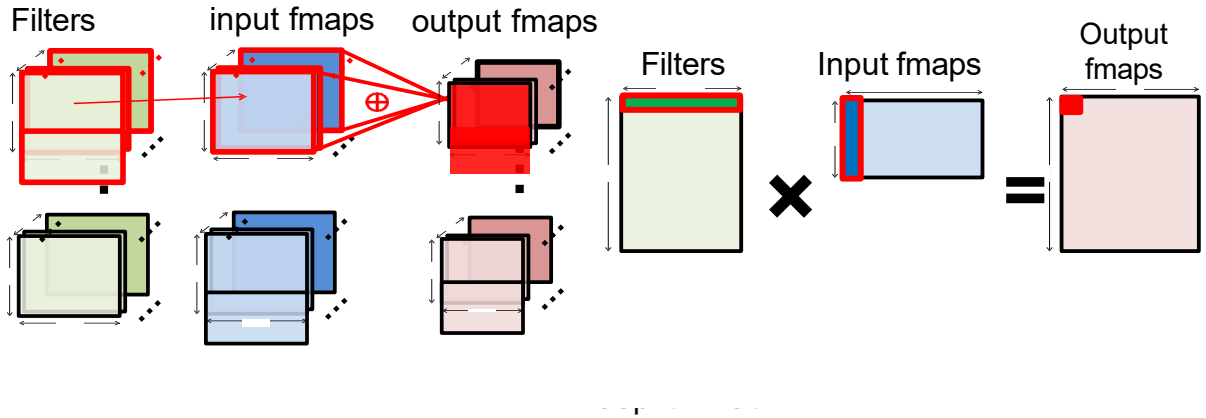
$$y_i = \gamma\overline{x_i} + \beta$$



Application: Classification from Cat, dog, bird, human, sheep

Heart of ML Compute: GEMM

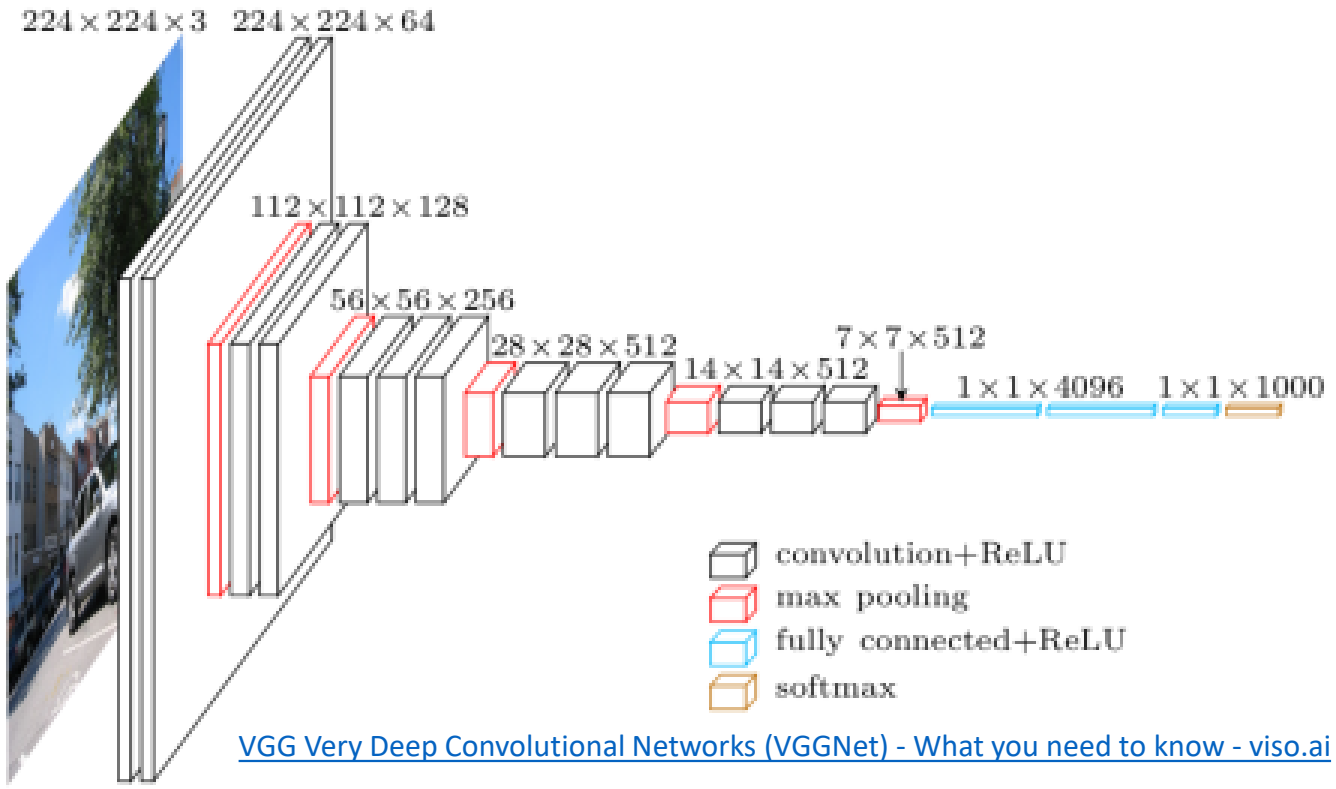
- GEMM: General Matrix to Matrix Multiply
- Each filter operation can be mapped to a VMM → multiple filters or multiple inputs (batch size > 1) → GEMM
- GEMM works for CONV, FC, Attention, Recurrent, etc most layers
  - CONV requires reshaping (Im2col)
  - Note that the input data is repeated
- Most ML accelerators and software try to speed up GEMM



Quick question

- A CONV layer with 100 5x5x3 filters, 20x20x3 input feature map (assume padding)
  - What is the size of GEMM to be done ?

Example DNN: VGG16



VGG Very Deep Convolutional Networks (VGGNet) - What you need to know - viso.ai

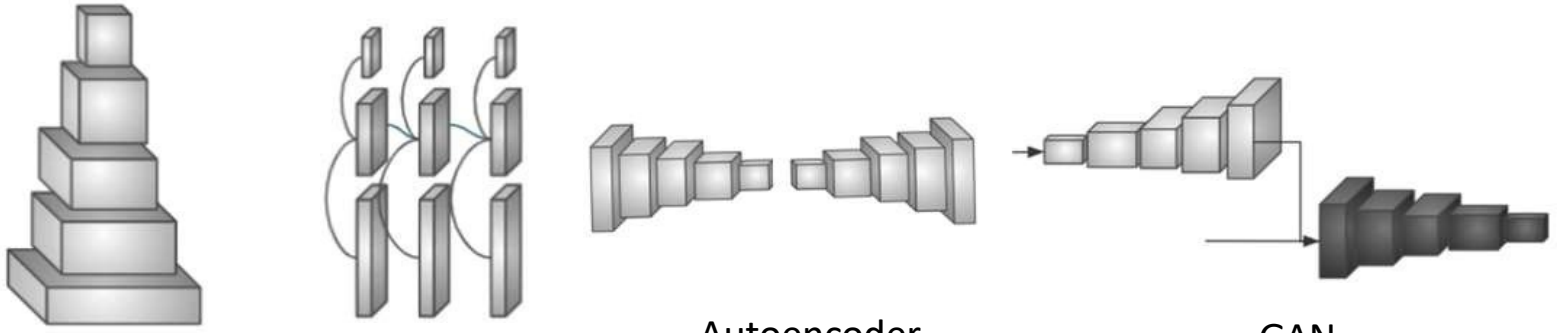
- Total 138M parameters [[1409.1556v6] Very Deep Convolutional Networks for Large-Scale Image Recognition (arxiv.org)]

Types of Deep Learning

- Supervised Learning:
  - learning a function that maps an input to an output from datasets consisting of input-output pairs
  - Example: prediction of house price based on location
- Unsupervised Learning:
  - … input data without labeled responses.
  - Example: recommend channels based on history
- Semi-supervised Learning
  - … a small amount of labeled data with a large amount of unlabeled data
  - Example: detection of CAR from photos with a few labels
- Reinforcement Learning
  - learning in an interactive environment using feedback from its own actions and experiences
  - Example: game AI

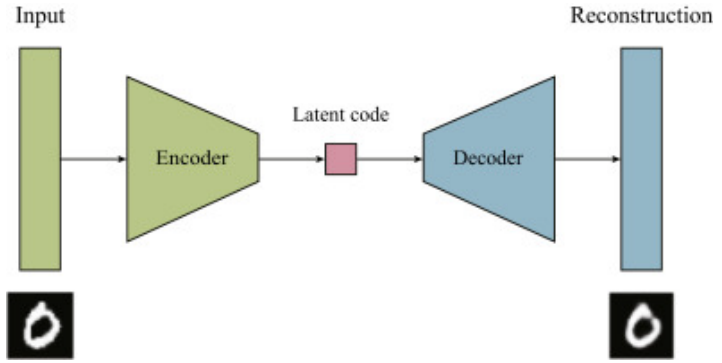
Model architectures

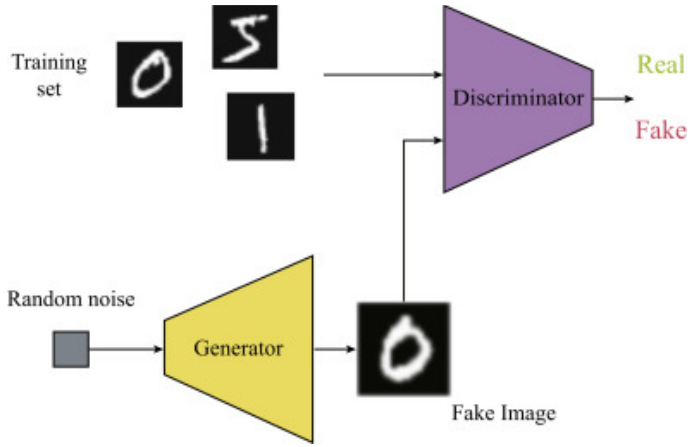
- Convolutional Neural Network (CNN):
  - feed-forward structure consists of a stack of layers
- Recurrent Neural Network (RNN):
  - transform an input sequence into an output sequence
- Transformers:
  - Input to output sequence. Leverage “attention” module
- Autoencoder:
  - learn a low-dimensional representation of a high-dimensional data set
- Generative Adversarial Network (GAN):
  - consist of two (CNN) models competing against each other in a two-player game framework
  - create new fake data by learning through the real data



Example: An Autoencoder

- Two NNs: encoder generating a compressed ”latent code” so that reconstruction loss is minimized
  - The latent code can serve as a reduced dimension representation (similar to PCA)
- Generative Adversarial Networks (GANs) use to the encoder component but with a different loss function
  - To generate “realistic” fake inputs
- Related concept: “embedding layers”: maps input information from a high-dimensional to a lower-dimensional space
  - Encoders can generate an embedding (of an image, text, graph, …)
  - Embedding can be as simple as a one-hot encoding of words in a vocabulary





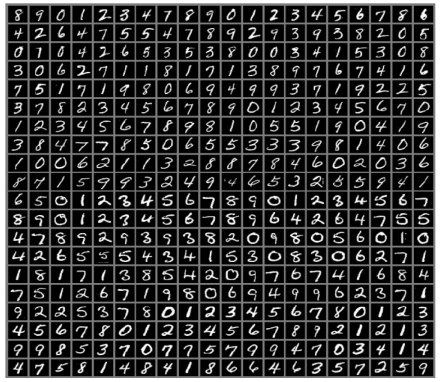
[Autoencoders – ScienceDirect]

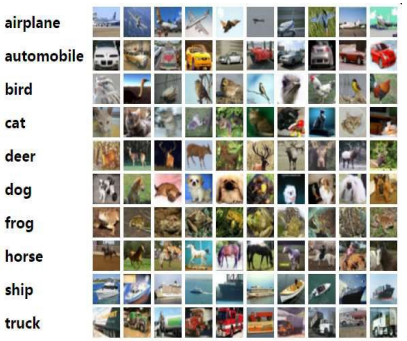
A Full Cycle of DL in Practice

- Training:
  - Data Cleaning
  - Data Input
  - Forward Computation
  - Loss Calculation
  - Backward Computation
  - Parameter Update
  - Converge Check / Stop Criteria
  - Save Model (Parameters)
- Testing/Inference:
  - Data Cleaning
  - Data Input
  - Load Saved Model
  - Forward Computation
  - Results Interpretation

Common Computer Vision Datasets

- MNIST (1998)
  - scans of handwritten digits and associated labels describing which digit 0–9 is contained in each 28x28 image.
  - The training set contains 60000 examples, and the test set 10000 examples.
  - A simple dataset by modern standards
- CIFAR10 (2009)
  - 60000 32x32 color images in 10 classes, with 6000 images per class.
  - 50000 training images and 10000 test images.
  - Has a 100 class CIFAR100 variant
- IMAGENET (2014)
  - ~14 million labeled images, 20k classes
  - Average image size 469x387 usually downsampled to 224x224
  - Images gathered from Internet
  - Human labels via Amazon MTurk





Logistics

- Make sure your BruinLearn, Gradescope, Piazza accounts are all set up.
  - Use Piazza for any questions. No direct emails please.
- Make sure your account on SEAS machines is setup to use Matlab, Cadence, Python
- Set up account on Google Colab
  - We will use this for ML software projects
- Expect hiccups!
  - First time teaching this class.
  - All projects are new and setup from scratch.
  - I apologize upfront for any issues throughout the quarter

How to present a research paper ?

- What problem are they solving ?
- Why is it an important problem (or if you think it is not, why not) ?
- What are the key ideas to the approach ?
  - Explain without equations, nitty gritty details..
- Briefly: what is the experimental setup ?
  - Is it reasonable ?
  - What do they compare against ?
- What did you think of the paper ?
  - Obvious problems ?
  - Good nuggets ?
- Do not just copy paste figures/tables/charts from the paper unless they are clear and explain things well. You may need to redraw some figures.
  - Figures and charts are much better in slides than text and tables
- Spell, grammar check your slides
- Do NOT use long paragraphs in slides!
- 15 minutes → < 15 slides

Papers for Week 2

- Lecture 3
  1. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).
  2. Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
  3. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
- Lecture 4
  1. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
  2. Jorda, Marc, Pedro Valero-Lara, and Antonio J. Pena. "Performance evaluation of cudnn convolution algorithms on nvidia volta gpus." *IEEE Access* 7 (2019): 70461-70473.
  3. Chen, Yu-Hsin, Joel Emer, and Vivienne Sze. "Using dataflow to optimize energy efficiency of deep neural network accelerators." *IEEE Micro* 37.3 (2017): 12-21.
- Paper-student assignment will be posted on Piazza.
  - NOT in alphabetical order
  - No requests for delays/change would be entertained unless medical reasons