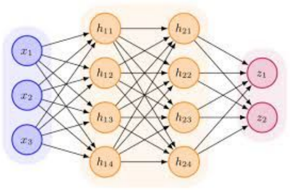
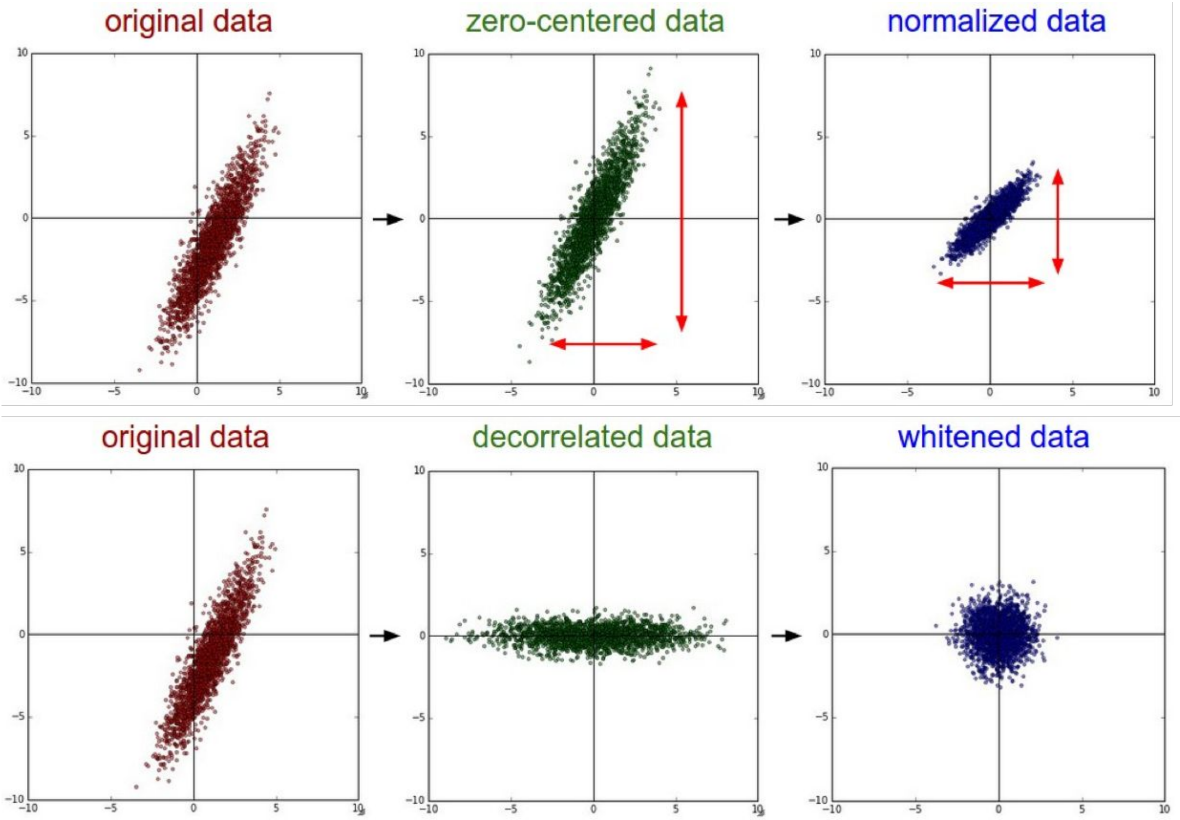
Deep Learning Programming: The Finer Details



Girish Varma

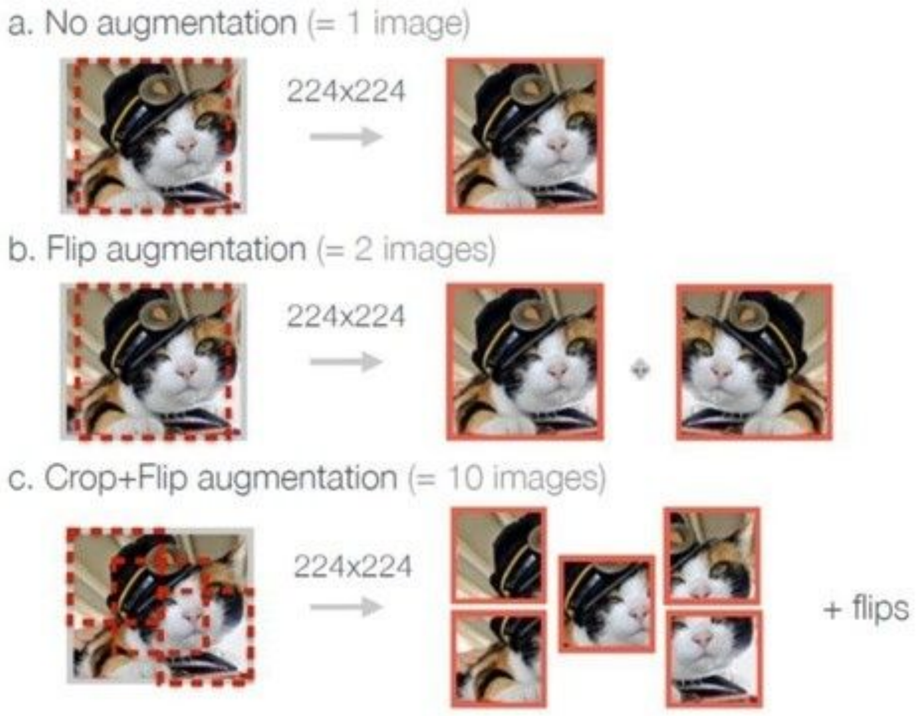
Step 1: Data Loading

Data Normalization



Data Augmentation or Jittering

A trick to increase the training data



Step 2: Model Definition

Weight Initialization

Need to pick a starting point for gradient descent: an initial set of weights

Zero is a very **bad idea**!

- Zero is a **critical point**
- Error signal will not propagate
- Gradients will be zero: no progress

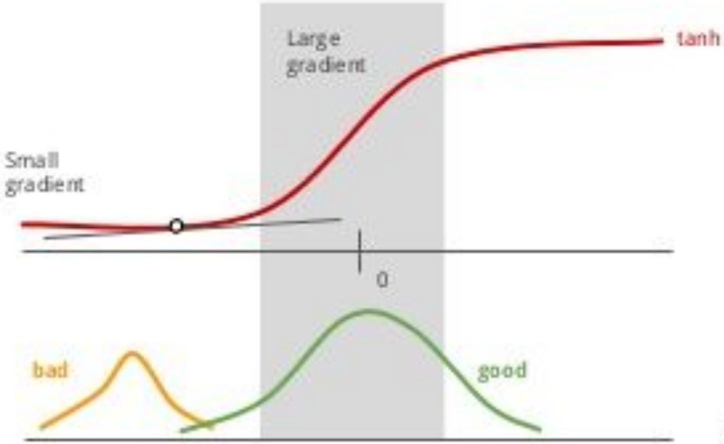
Constant value also bad idea:

- Need to break symmetry

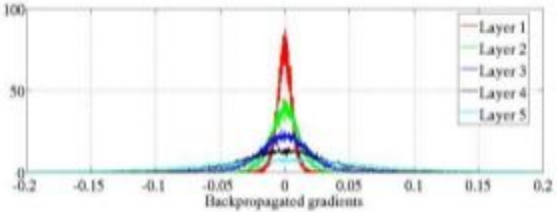
Use **small random values:**

- E.g. zero mean Gaussian noise with constant variance

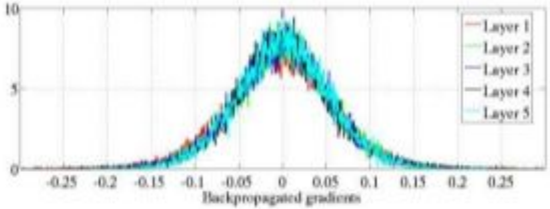
Ideally we want inputs to activation functions (e.g. sigmoid, tanh, ReLU) to be mostly **in the linear area** to allow larger gradients to propagate and converge faster.



Weight Initialization



standard



"Xavier"

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right]$$

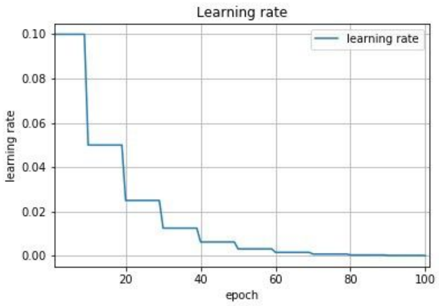
Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *International conference on artificial intelligence and statistics*. 2010.

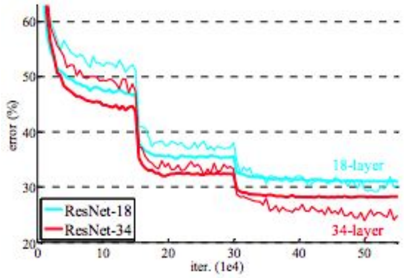
Step 3: Specify Loss and Training Algo

Optimization Algorithms

SGD

Learning_rate (learning schedule)



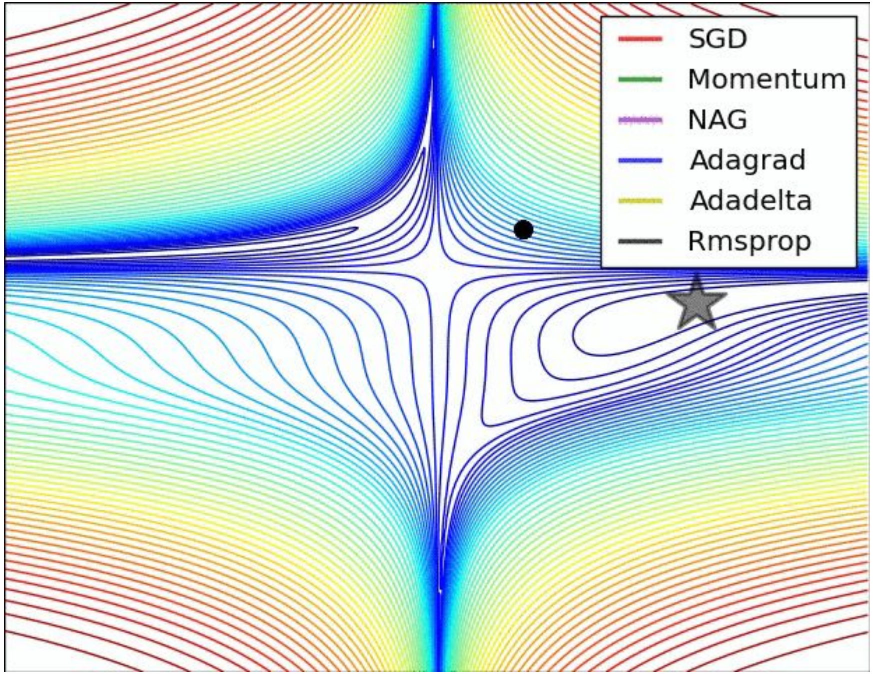


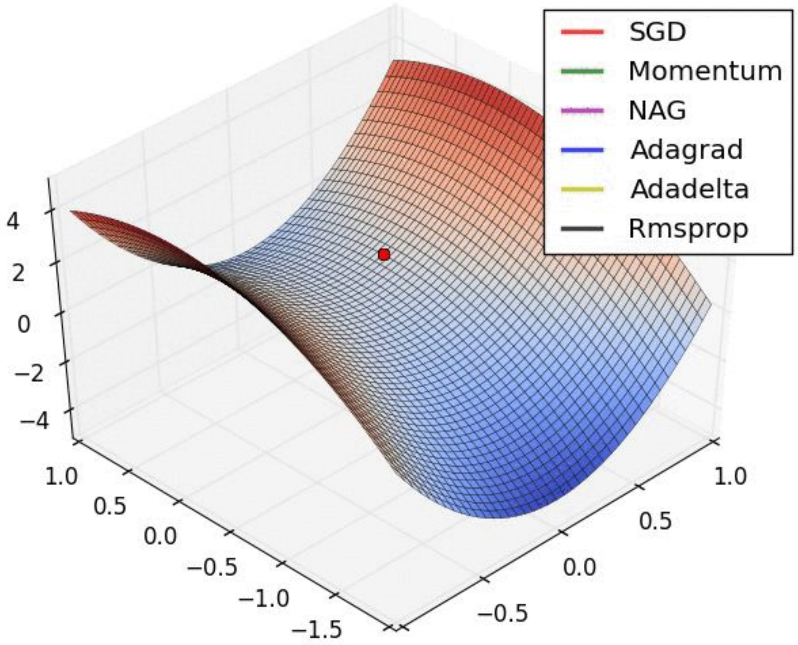
momentum

https://distill.pub/2017/momentum/

Adam

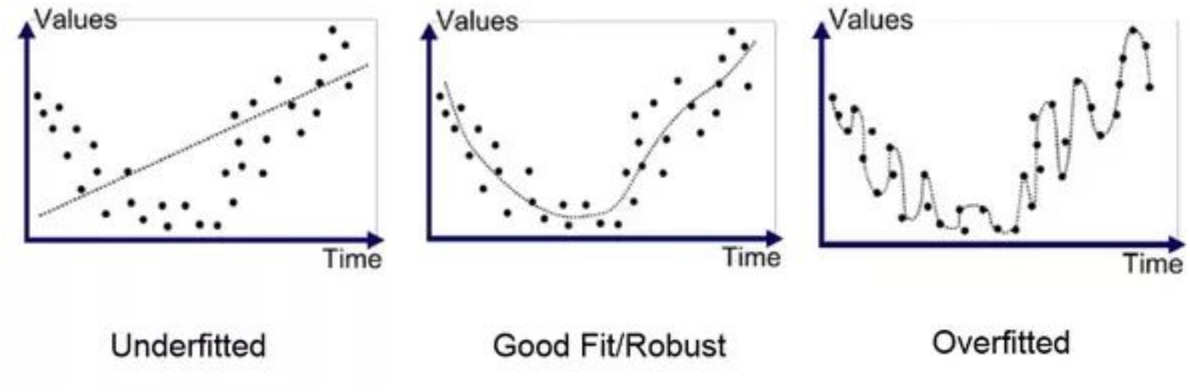
Optimization Algorithms





Regularizer

If neural network weights are unconstrained, it can over fit the data.



Regularizers

Regularization ensures that the weights take only a small range of values

L1 Regularization

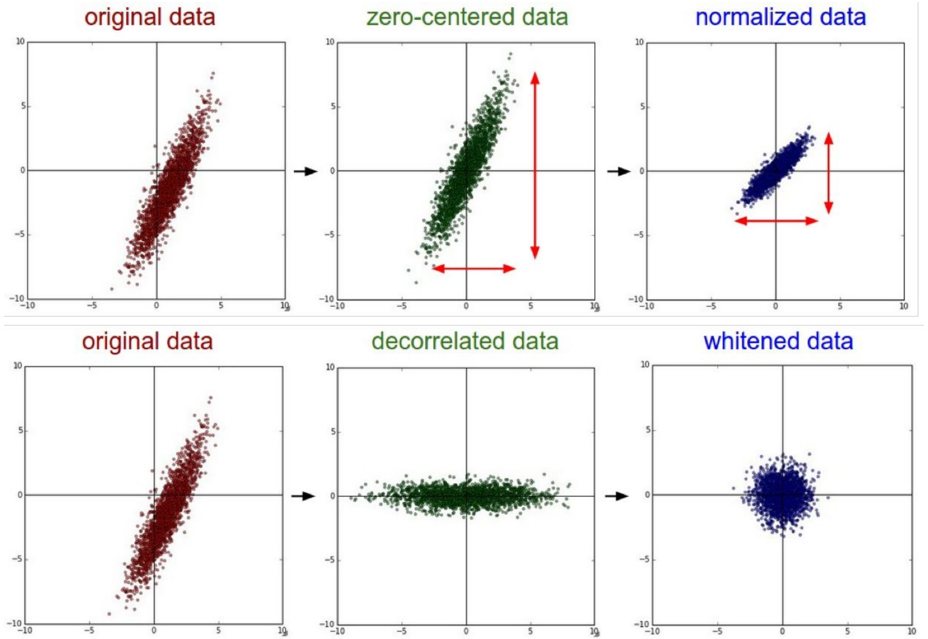
$$\text{Cost} = \sum_{i=0}^{N}\Big(y_i - \sum_{j=0}^{M} x_{ij}W_j\Big)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}\Big(y_i - \sum_{j=0}^{M} x_{ij}W_j\Big)^2}_{\text{Loss function}} + \lambda \underbrace{\sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

Batch Norm Layer

Apply normalization to hidden space



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

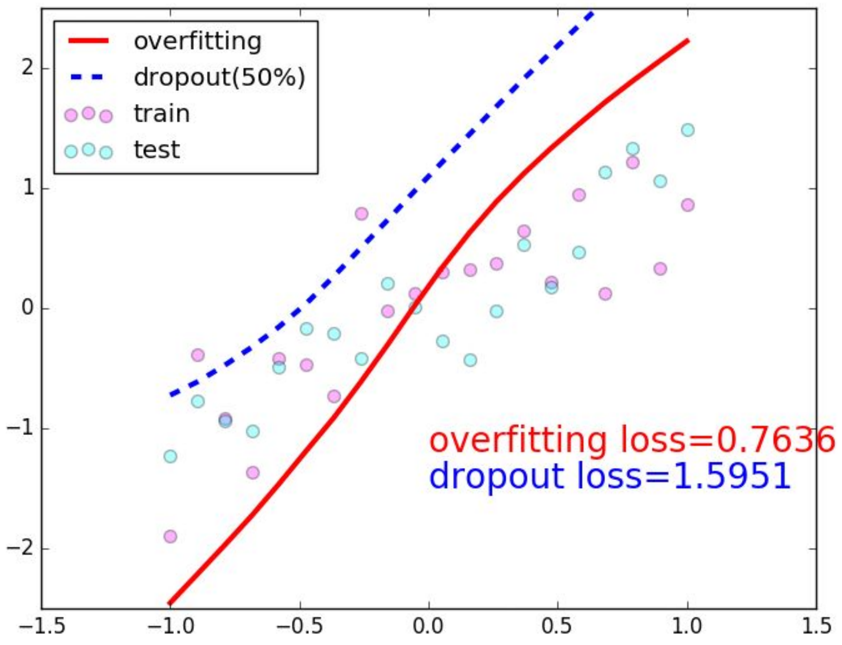
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Dropout

Another way of preventing overfitting.

Cons:

Training can take longer.

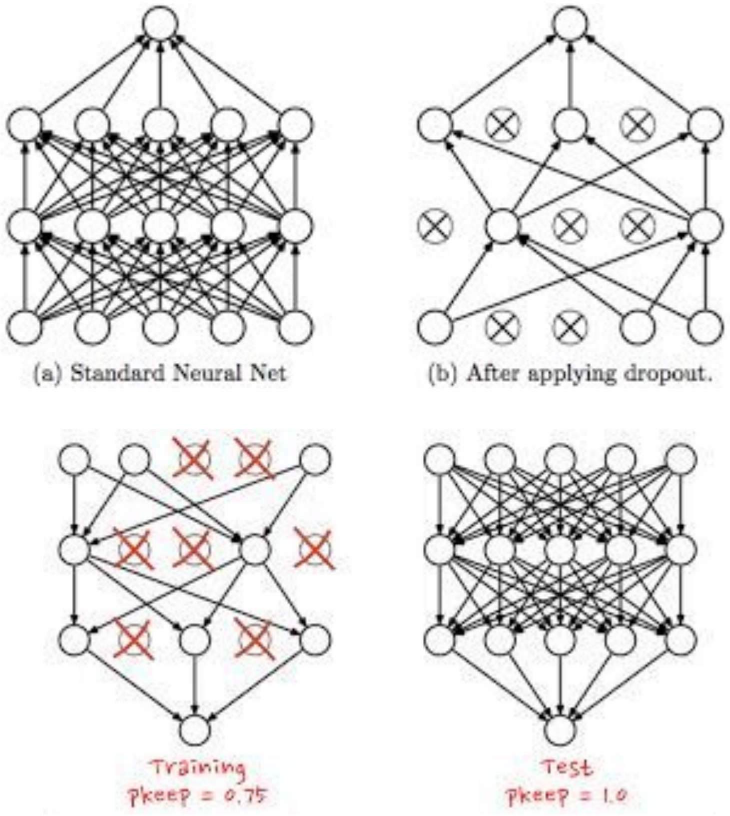


Dropout

Another way of preventing overfitting.

Cons:

Training can take longer.



Summary

- Data Normalization
- Data Augmentation
- Weight Initialization
- Optimization Algorithms
- Regularizer
- Batch Norm (Layer)
- Dropout (Layer)

GPU and Deep Learning

CPU vs GPU

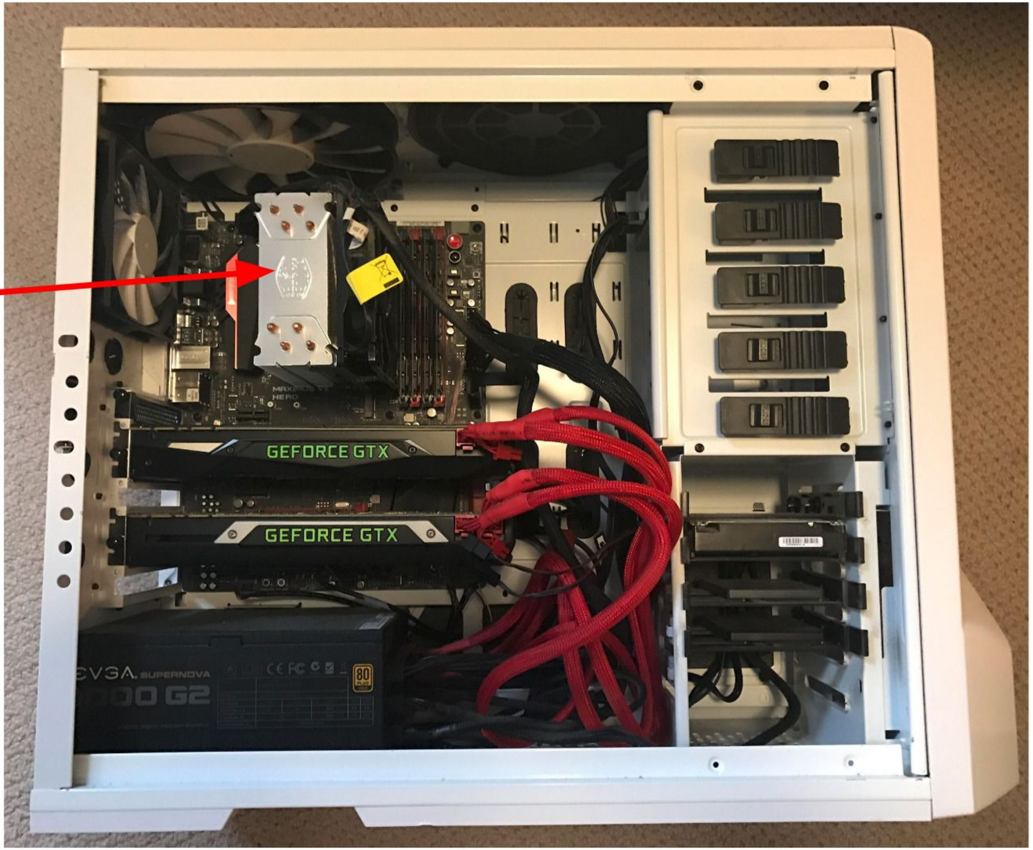
GPU and Deep Learning

My computer

GPU and Deep Learning

Spot the CPU!

(central processing unit)



GPU and Deep Learning

Spot the GPUs!

(graphics processing unit)

This image is in the public domain

GPU and Deep Learning

AIML

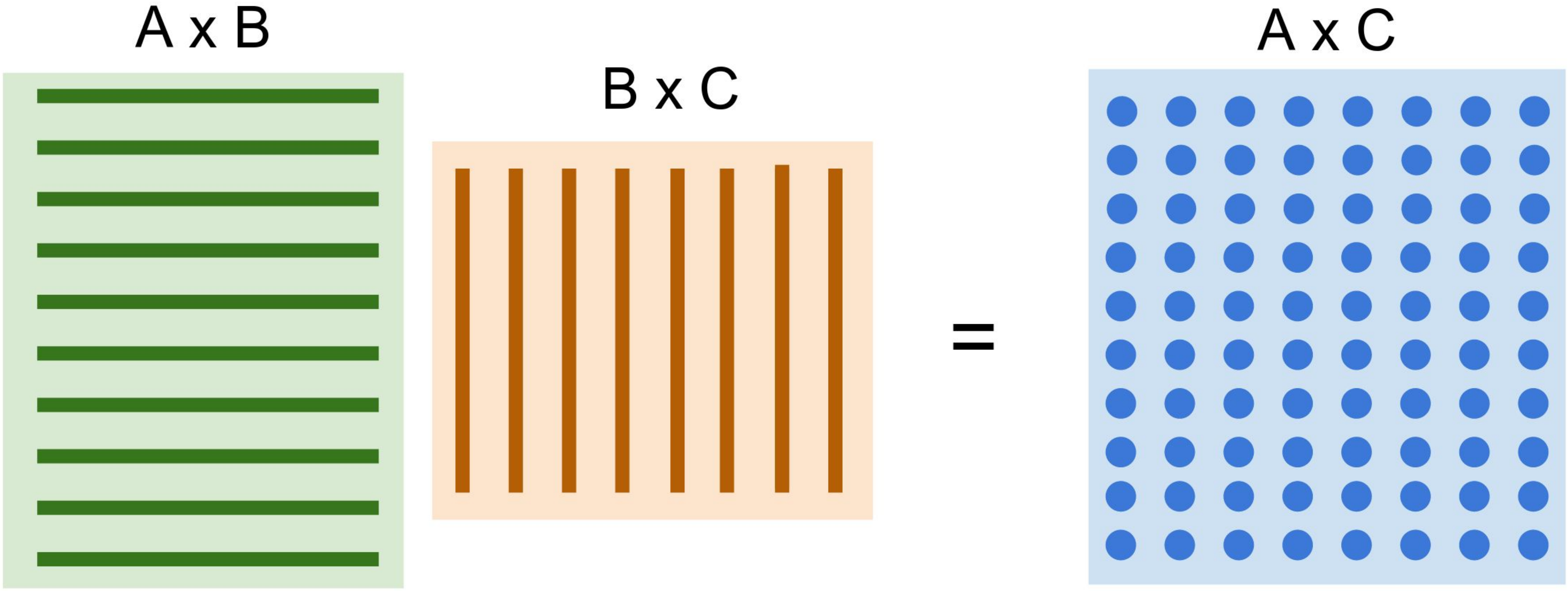
CPU vs GPU

	# Cores	Clock Speed	Memory	Price
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.4 GHz	Shared with system	$339
CPU (Intel Core i7-6950X)	10 (20 threads with hyperthreading)	3.5 GHz	Shared with system	$1723
GPU (NVIDIA Titan Xp)	3840	1.6 GHz	12 GB GDDR5X	$1200
GPU (NVIDIA GTX 1070)	1920	1.68 GHz	8 GB GDDR5	$399

CPU: Fewer cores, but each core is much faster and much more capable; great at sequential tasks

GPU: More cores, but each core is much slower and “dumber”; great for parallel tasks

Example: Matrix Multiplication



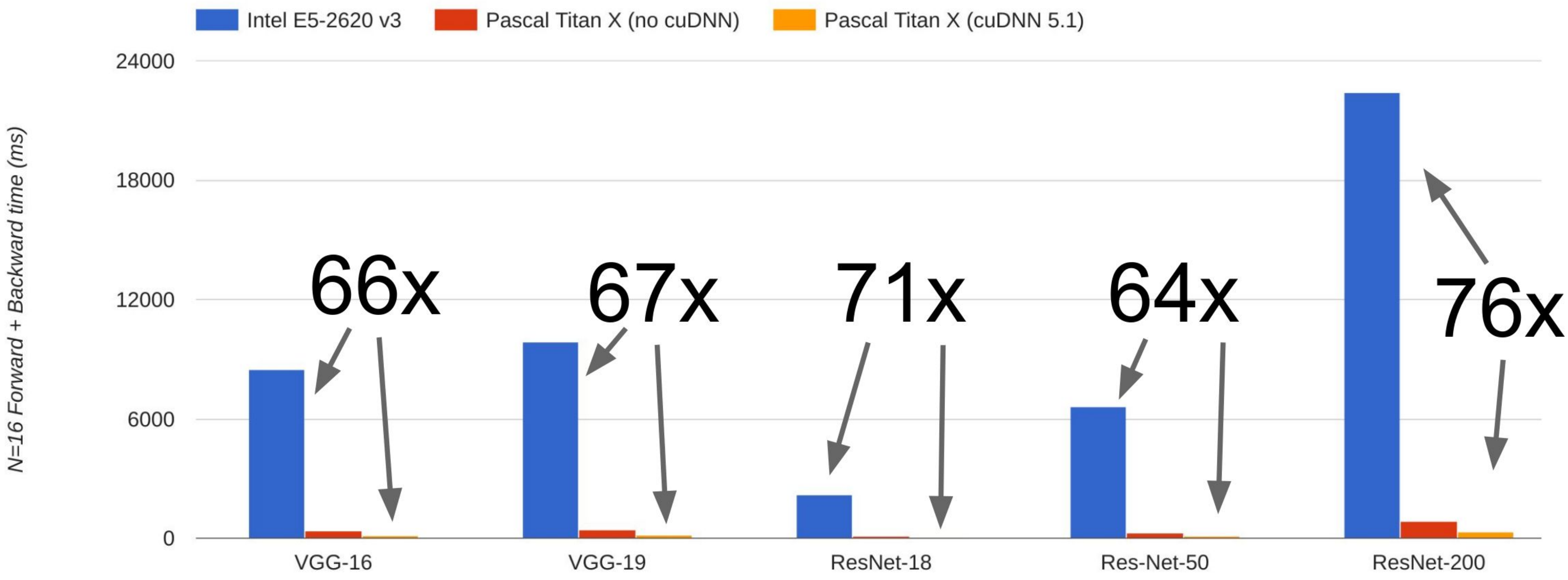
Programming GPUs

- CUDA (NVIDIA only)
  - Write C-like code that runs directly on the GPU
  - Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - Similar to CUDA, but runs on anything
  - Usually slower :(

GPU and Deep Learning

CPU vs GPU in practice

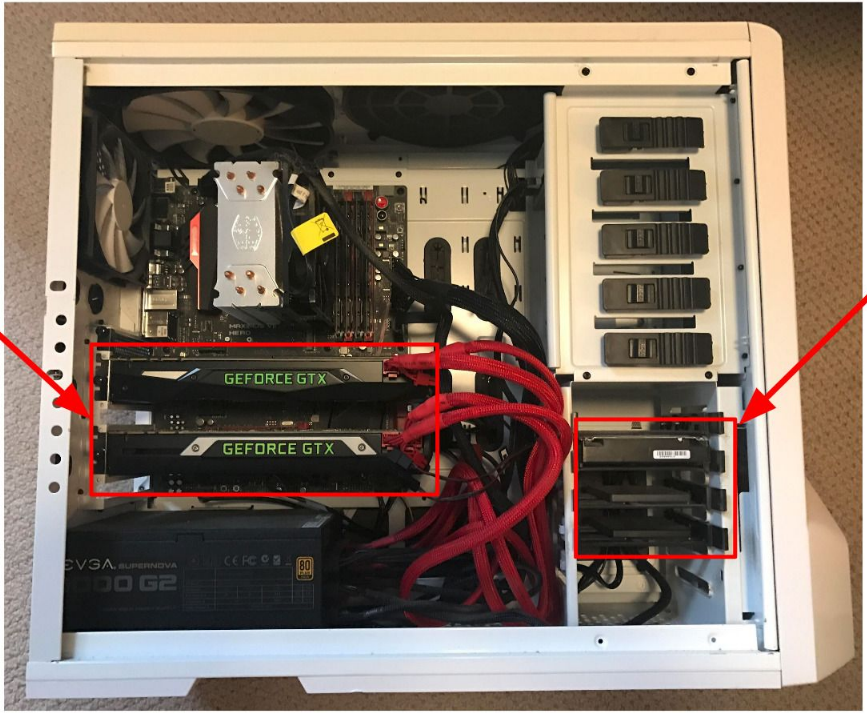
(CPU performance not well-optimized, a little unfair)



GPU and Deep Learning

CPU / GPU Communication

Model is here



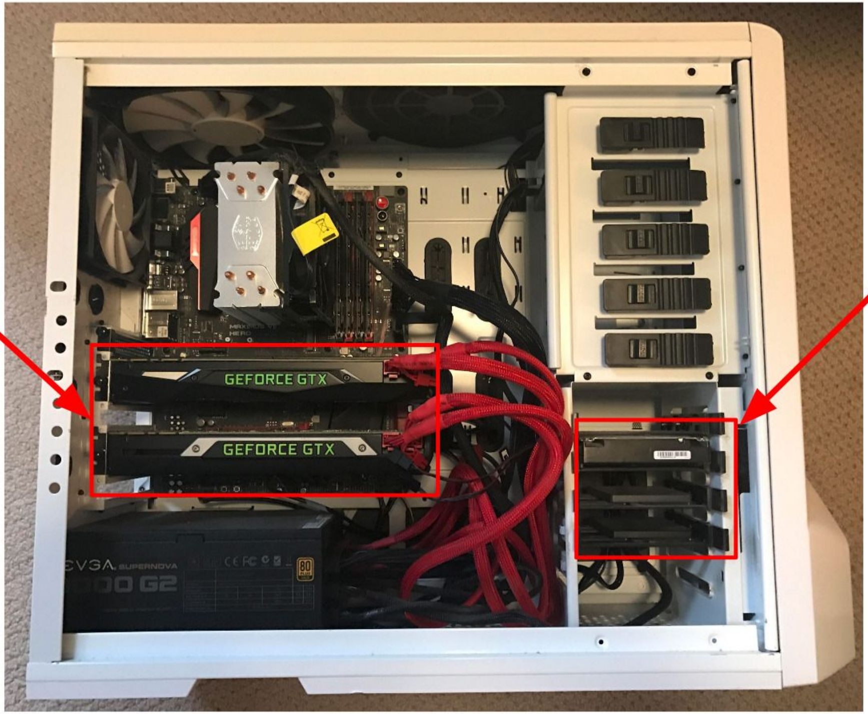
Data is here

GPU and Deep Learning

AIML

CPU / GPU Communication

Model
is here



Data is here

If you aren't careful, training can bottleneck on reading data and transferring to GPU!

Solutions:

- Read all data into RAM
- Use SSD instead of HDD
- Use multiple CPU threads to prefetch data