Deep Learning & Convolution Neural Networks (CNNs)

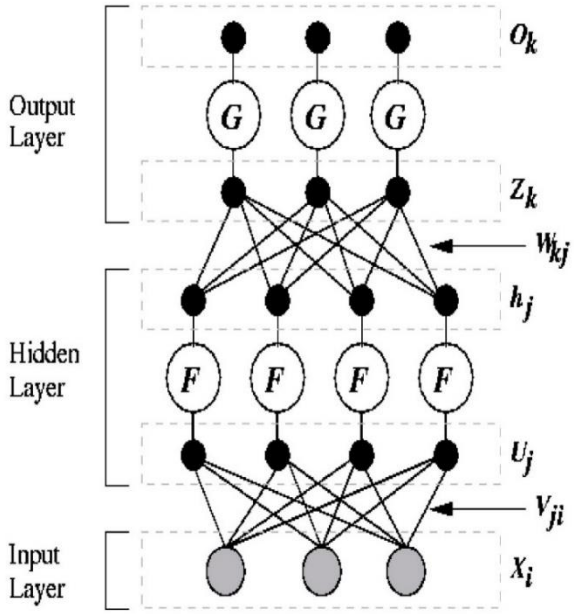
Jenq-Neng Hwang, Professor

Department of Electrical & Computer Engineering

University of Washington, Seattle WA

hwang@uw.edu

Different Loss for BP



- For classification, if it is a binary (2-class) problem, then cross-entropy error function often does better

$$E = -\sum_{n=1}^{N} t^{(n)} \log o^{(n)} + (1 - t^{(n)}) \log(1 - o^{(n)})$$

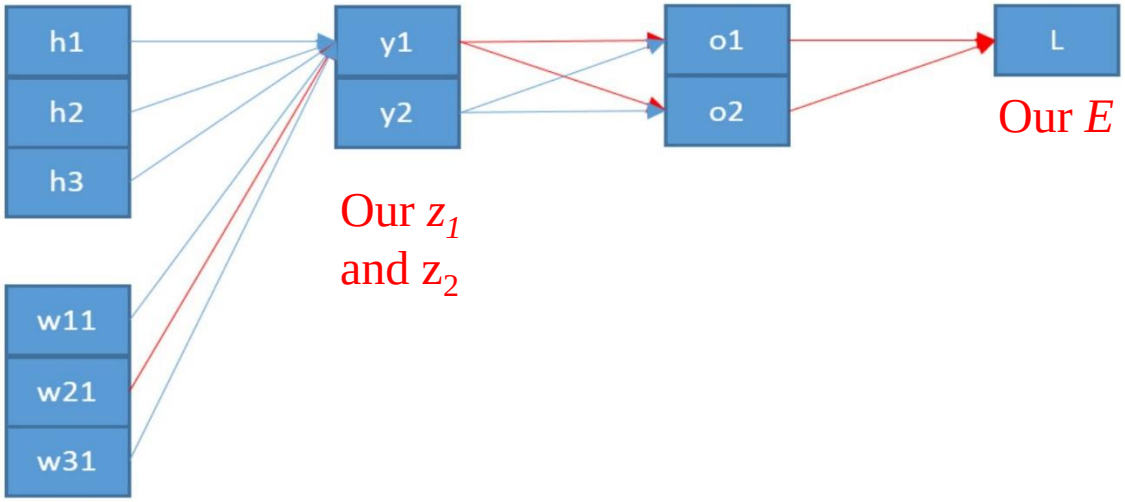
$$o^{(n)} = (1 + \exp(-z^{(n)})^{-1} \quad \text{sigmoid}$$

- For multi-class classification problems, use the softmax activation (prob.)

$$E = -\sum_{n} \sum_{k} t_k^{(n)} \log o_k^{(n)}$$

$$o_k^{(n)} = \frac{\exp(z_k^{(n)})}{\sum_j \exp(z_j^{(n)})}$$

Gradient for Cross-Entropy



$$L = -t_1 \log o_1 - t_2 \log o_2$$

$$o_1 = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}$$

$$o_2 = \frac{\exp(y_2)}{\exp(y_1) + \exp(y_2)}$$

$$y_1 = w_{11}h_1 + w_{21}h_2 + w_{31}h_3$$

$$y_2 = w_{12}h_1 + w_{22}h_2 + w_{32}h_3$$

$$\frac{\partial L}{\partial o_1} = -\frac{t_1}{o_1}$$

$$\frac{\partial L}{\partial o_2} = -\frac{t_2}{o_2}$$

$$\frac{\partial o_1}{\partial y_1} = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)} - \left(\frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}\right)^2 = o_1(1 - o_1)$$

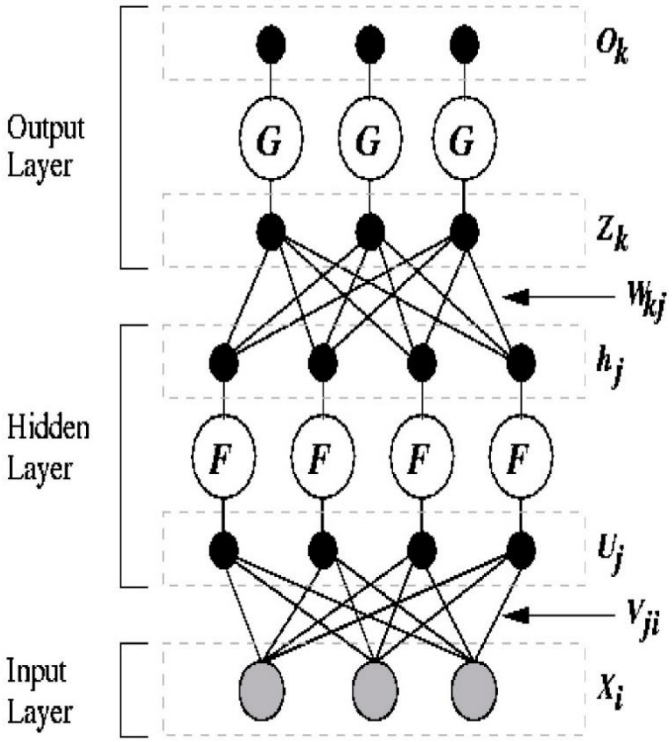
$$\frac{\partial o_2}{\partial y_1} = \frac{-\exp(y_2)\exp(y_1)}{(\exp(y_1) + \exp(y_2))^2} = -o_2 o_1$$

$$\frac{\partial y_1}{\partial w_{21}} = h_2$$

$$\begin{aligned}
\frac{\partial L}{\partial w_{21}} &= \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} \\
&= \frac{-t_1}{o_1}[o_1(1 - o_1)]h_2 + \frac{-t_2}{o_2}(-o_2 o_1)h_2 \\
&= h_2(t_2 o_1 - t_1 + t_1 o_1) \\
&= h_2(o_1(t_1 + t_2) - t_1) \\
&= h_2(o_1 - t_1)
\end{aligned}$$

$t_1+t_2=1$, because the vector $\boldsymbol{t}$ is a one-hot vector

Gradient Descent Updates



- How often to update
  - After each training sample ($N=1$) or
  - After a mini-batch of N training patterns (e.g., N=32, 64, 128, 256, etc)

$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}} = w_{ki} - \eta \sum_{n=1}^{N} (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)}) x_i^{(n)}$$

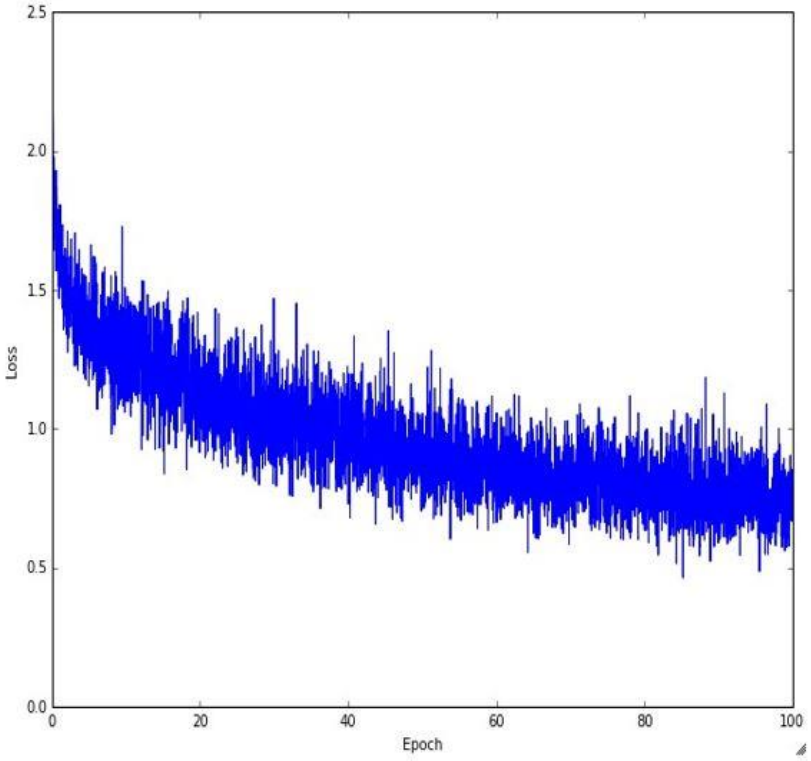
- How much to update
  - Use a fixed or an adaptive learning rate
  - Add a momentum term with leakage

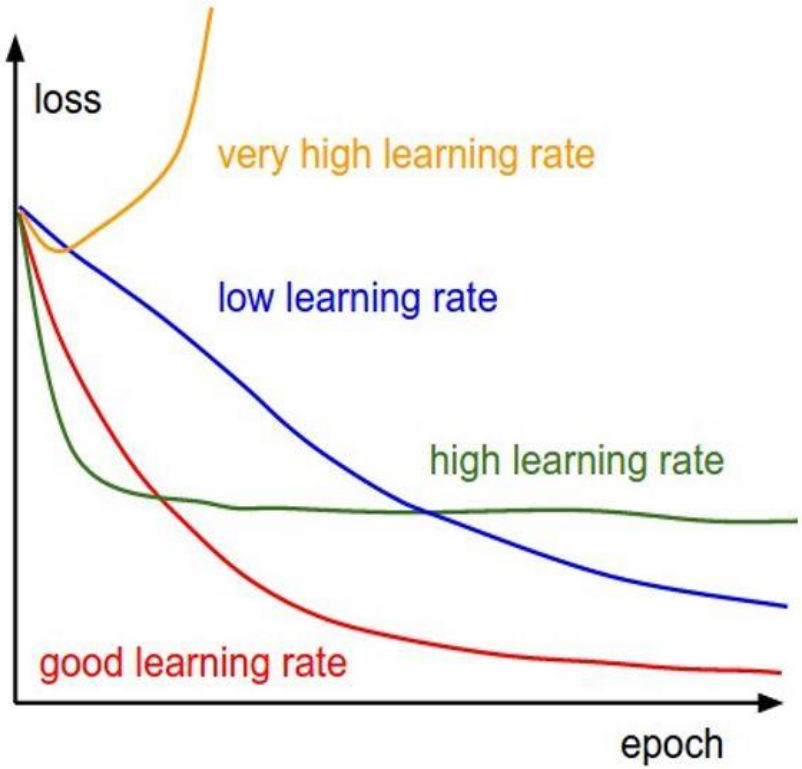
$$V_t = \mu V_{t-1} - \alpha \nabla L_t(W_{t-1})$$
$$W_t = W_{t-1} + V_t$$

- $\alpha > 0$ – *learning rate* (typical choices: 0.01, 0.1)
- $\mu \in [0, 1)$ – *momentum* (typical choices: 0.9, 0.95, 0.99)

Momentum smooths updates, enhancing stability and speed.

Choice of Learning Rate





More Variants of Updates

- Adaptive Gradient (AdaGrad)

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{\sum_{t'=1}^{t} \nabla L_{t'}(W_{t'-1})^2}}$$

- Root Mean Square Propagation (RMSProp)

$$\begin{aligned} R_t &= \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1})^2 \\ W_t &= W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{R_t}} \end{aligned}$$

Adaptive Moment (Adam) Estimation Updates

- Combine the advantages of:
  - AdaGrad – works well with sparse gradients
  - RMSProp – works well in non-stationary settings
- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

$M_0 = \mathbf{0}, R_0 = \mathbf{0}$ (Initialization)

For $t = 1, \ldots, T$:

$M_t = \beta_1 M_{t-1} + (1-\beta_1)\nabla L_t(W_{t-1})$ (1st moment estimate)

$R_t = \beta_2 R_{t-1} + (1-\beta_2)\nabla L_t(W_{t-1})^2$ (2nd moment estimate)

$\hat{M}_t = M_t / (1-(\beta_1)^t)$ (1st moment bias correction)

$\hat{R}_t = R_t / (1-(\beta_2)^t)$ (2nd moment bias correction)

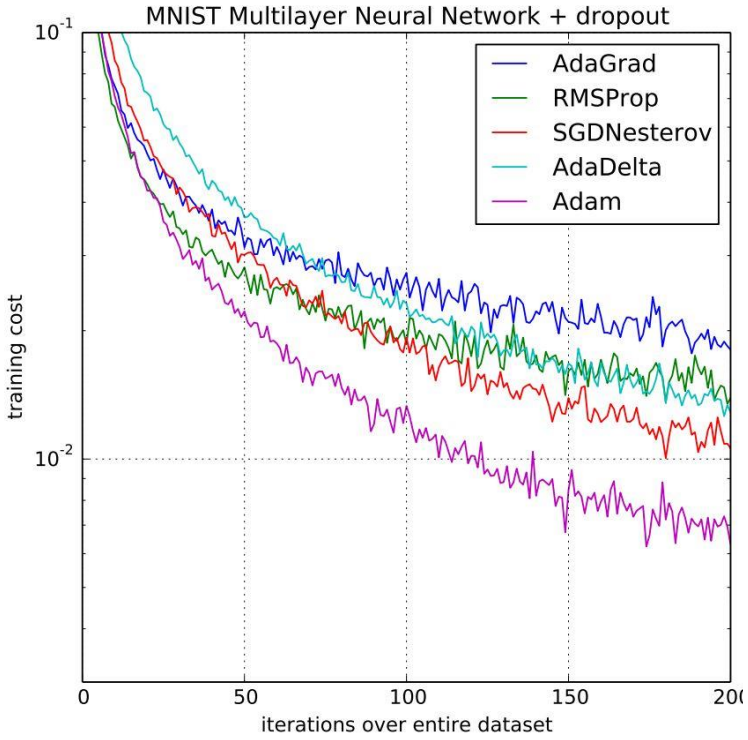
$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t + \epsilon}}$ (Update)

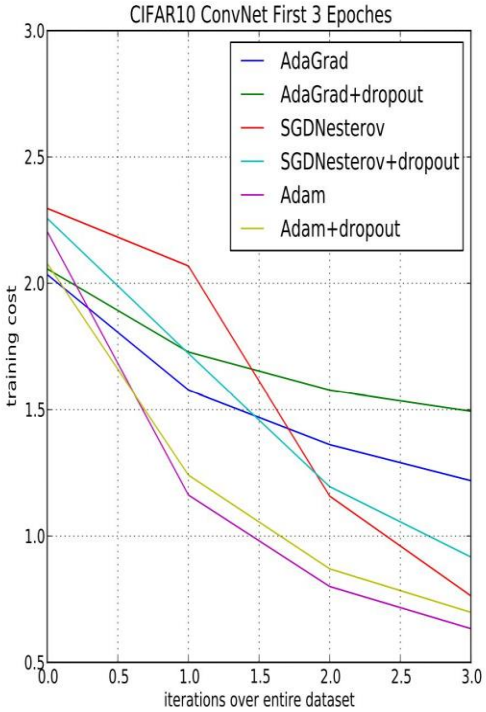
Return W_T

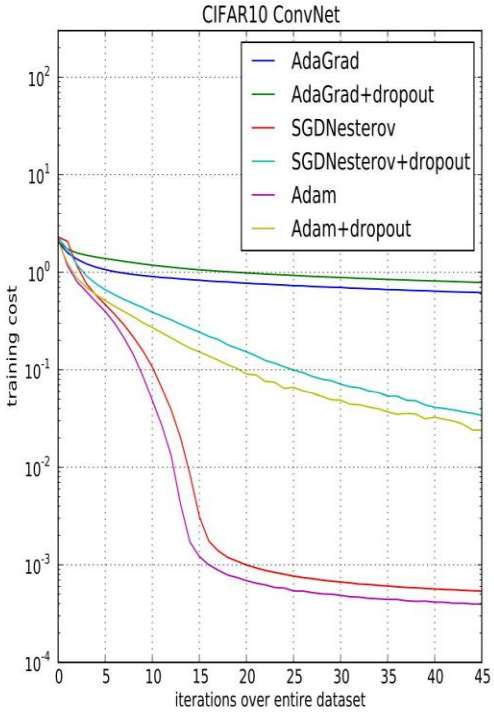
Hyper-parameters:

- $\alpha > 0$ – learning rate (typical choice: 0.001)
- $\beta_1 \in [0,1)$ – 1st moment decay rate (typical choice: 0.9)
- $\beta_2 \in [0,1)$ – 2nd moment decay rate (typical choice: 0.999)
- $\epsilon > 0$ – numerical term (typical choice: 10^{-8})

Adam: A Method for Stochastic Optimization



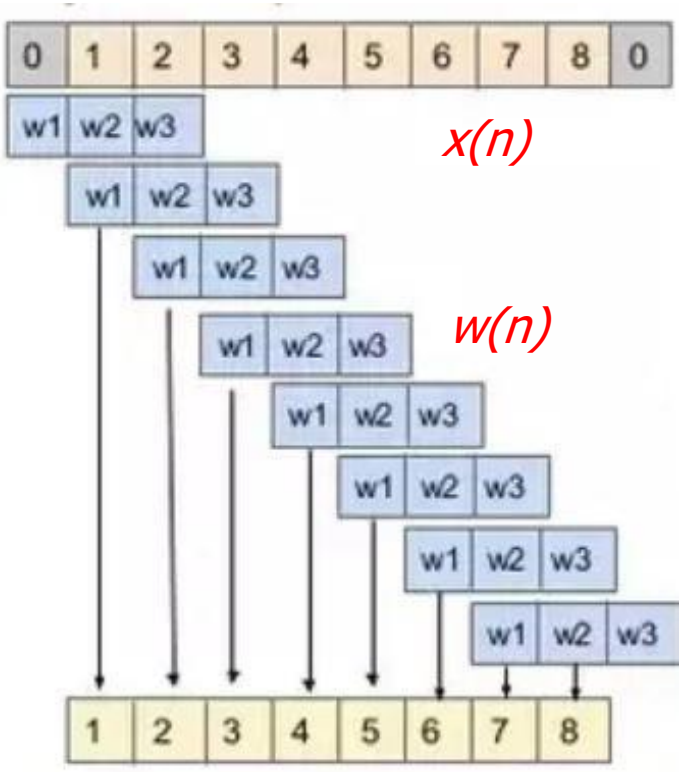


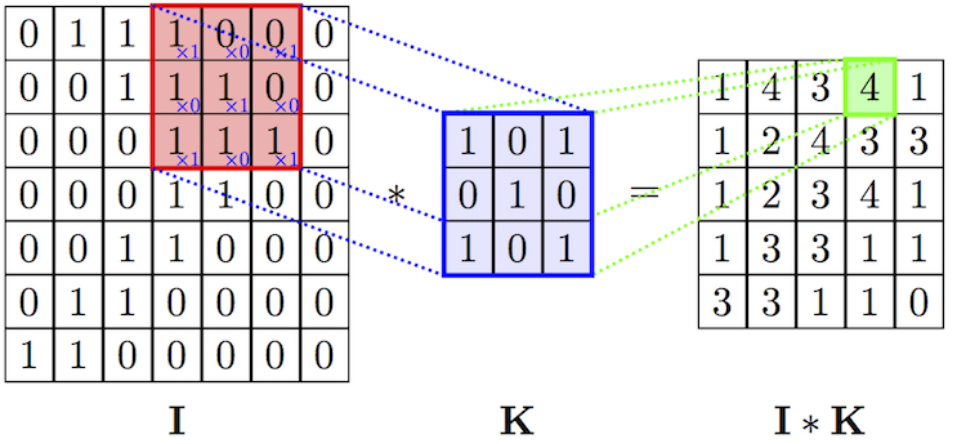


Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization," ICLR 2015, https://arxiv.org/abs/1412.6980

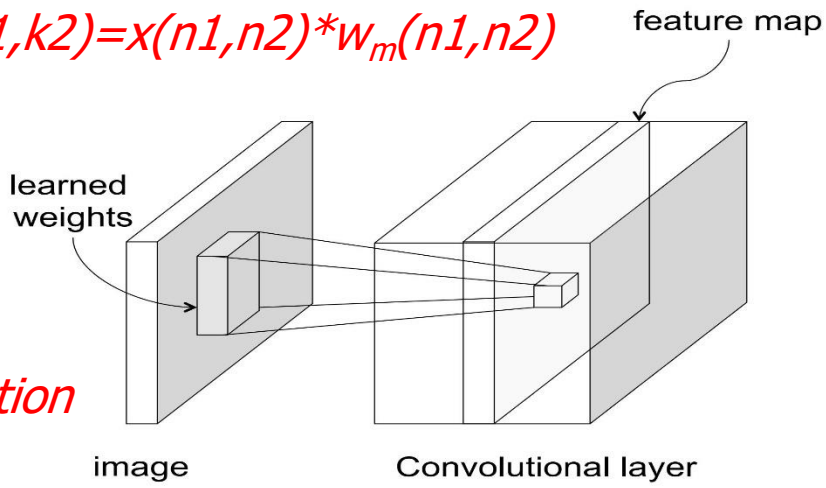
Deep Convolution Neural Networks (CNNs)

1-D and 2-D Convolution



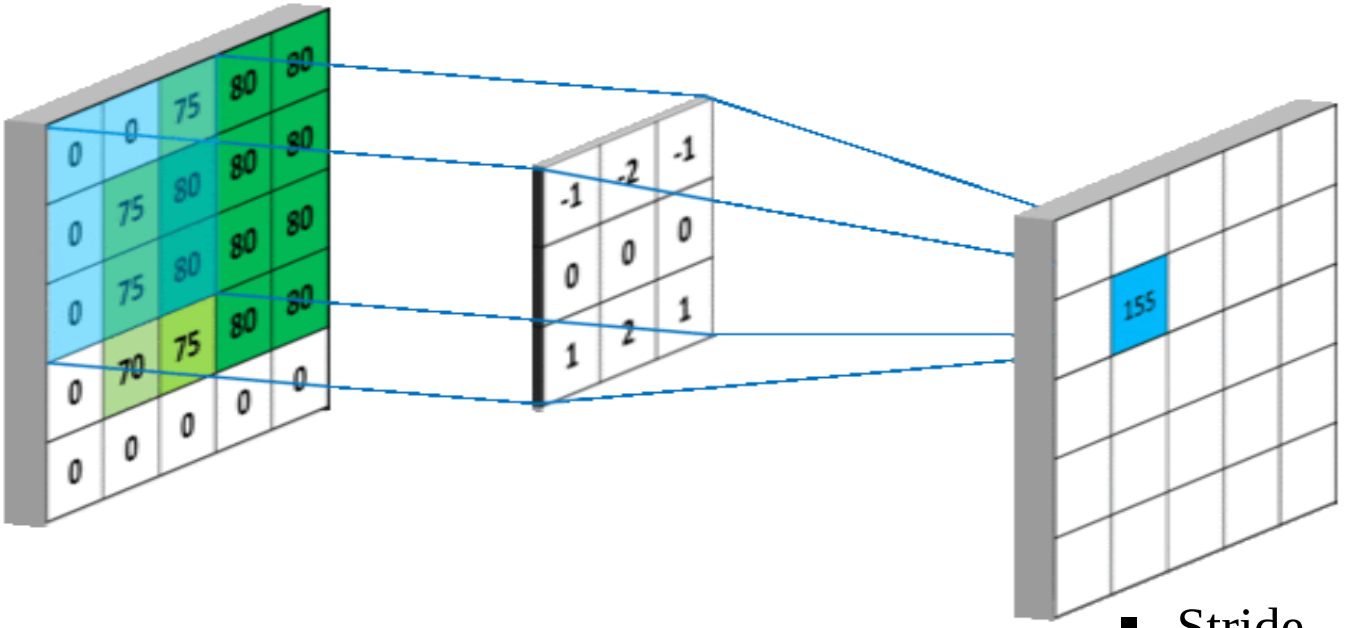


$F_m(k1,k2)=x(n1,n2)*w_m(n1,n2)$



$y(k)=x(n)*w(n)=sum_n\ x(n+k)w(k)$ correlation
should be $x(n-k)w(k)$ mathematically

2D Convolution Example



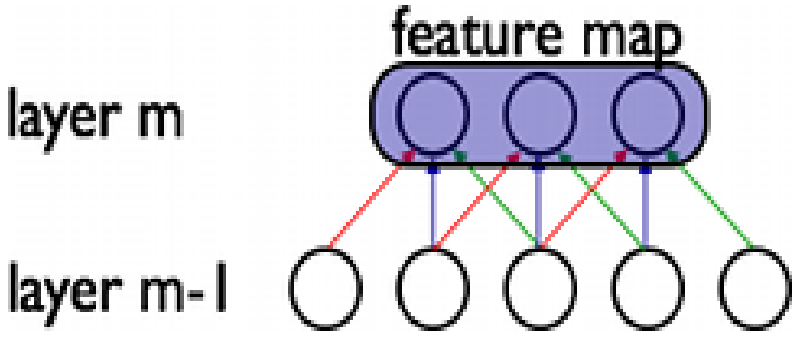
$F_m(k1,k2)=x(n1,n2)*w_m(n1,n2)$

- Stride
- Padding
- Pooling (Subsampling)

Let the convolution kernels $w_m(n1, n2)$ learnable in an MLP

Shared Convolutional Kernels

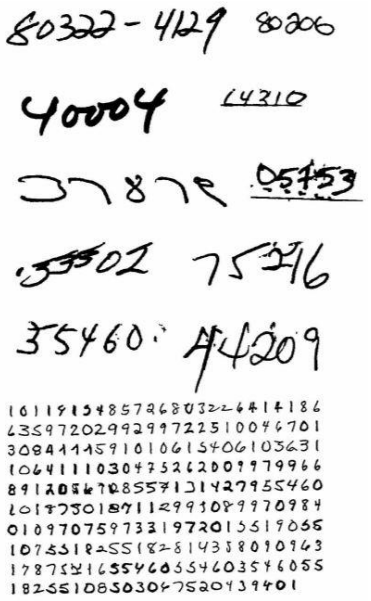
- Replicating units in this way allows for features to be "detected" regardless of their position in the visual field.
- Additionally, weight sharing increases learning efficiency by greatly reducing the number of learned free parameters.
- The constraints on the model enable CNNs to achieve better generalization on vision problems.

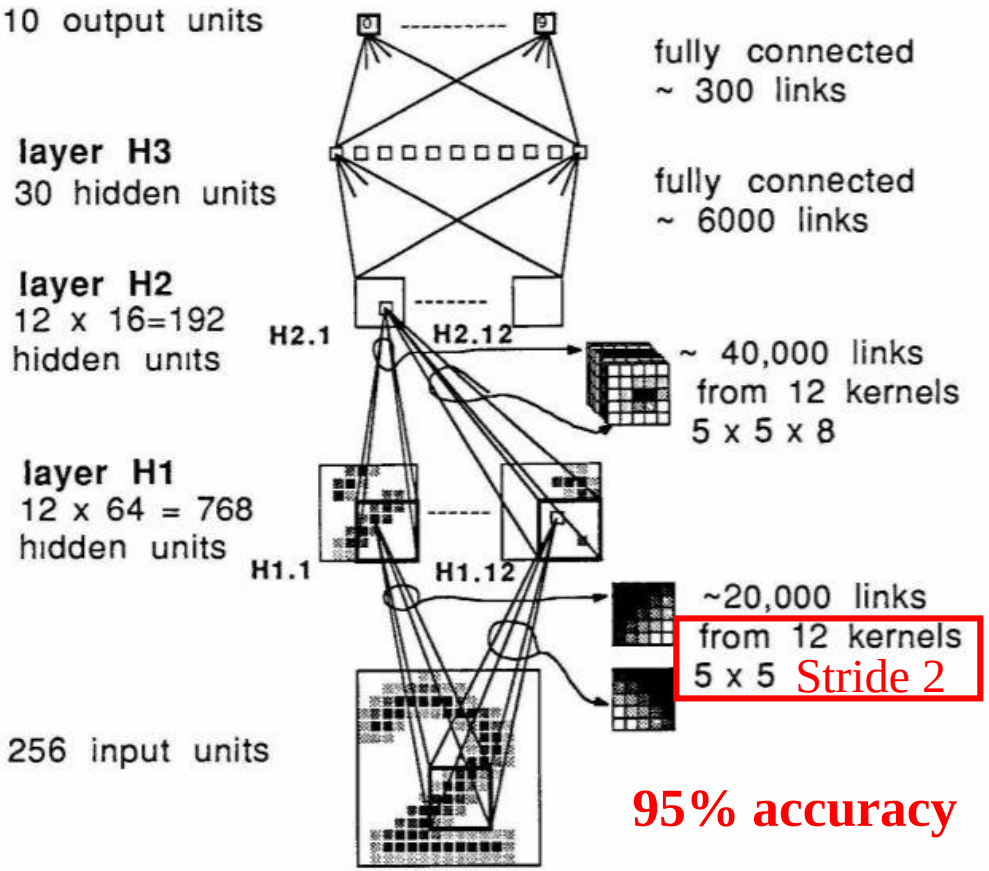


Convolution & Pooling

Shared weights and simplified nonlinearity [LeCun 1989]

LeNet for Handwritten Digits (1989)



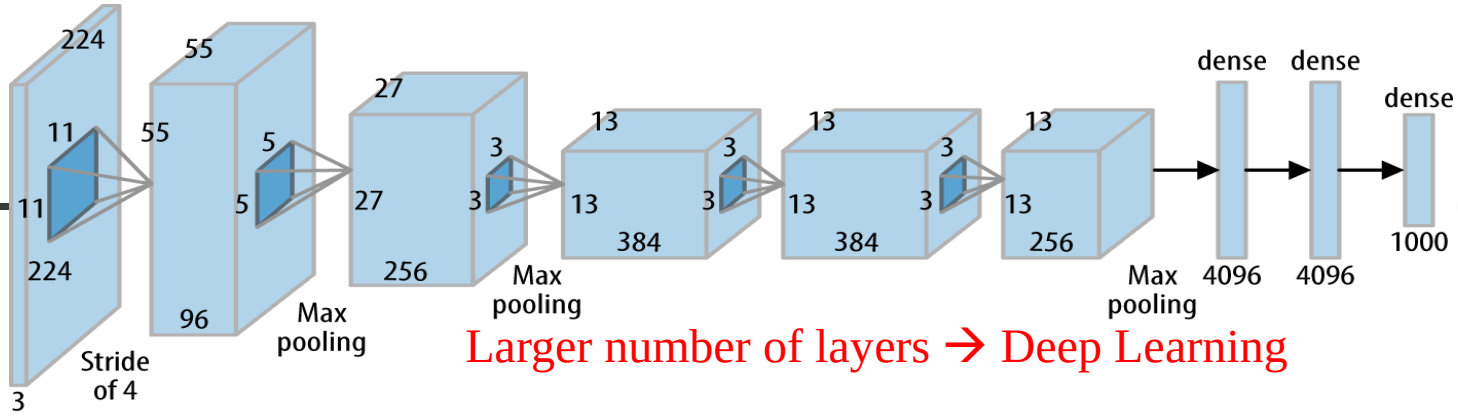


- For layer H1: 768 hidden units, 768x256=199608 connections, but only 1068 trainable weights (25x12+768 biases)

Y. LeCun, et al, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, Winter 1989.

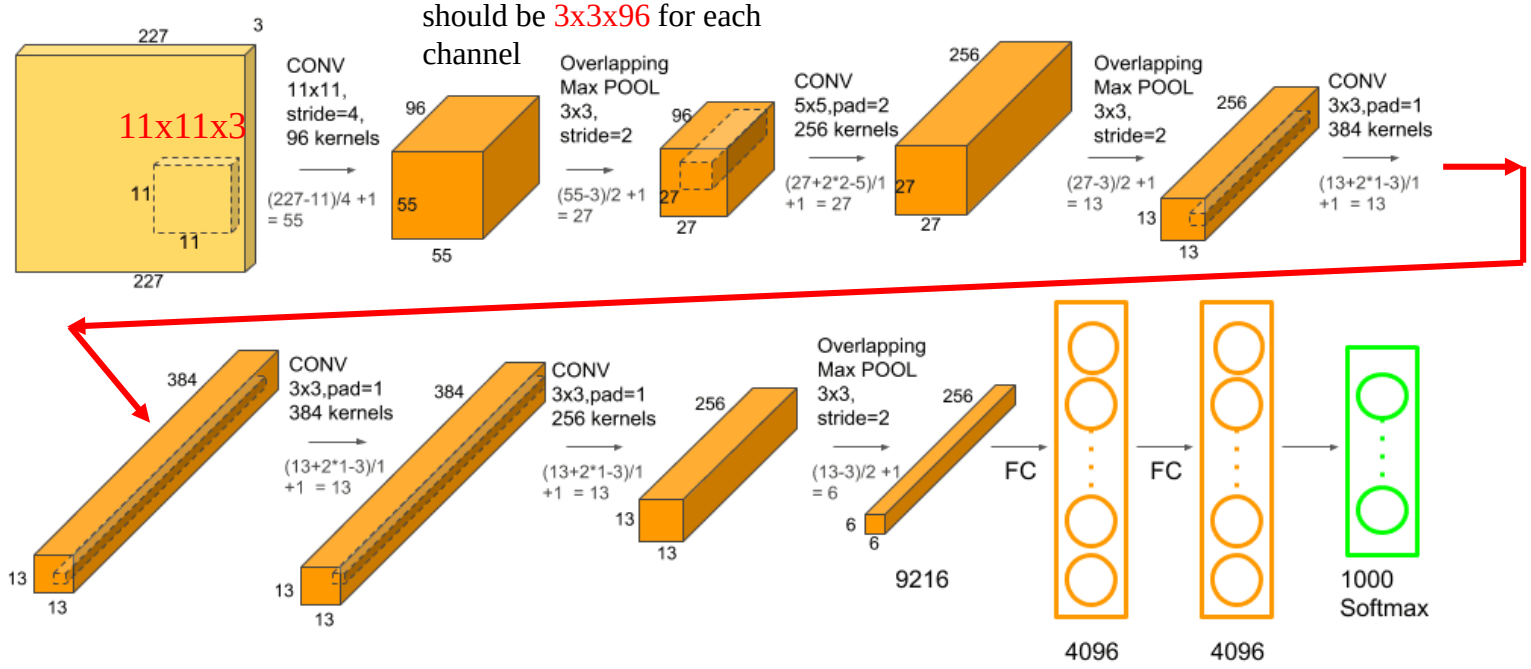
Ice Age of Neural Network based Learning 1992-2012

AlexNet for ImageNet



Larger number of layers → Deep Learning

Note that the 2D convolution should be 3x3x96 for each channel



Alex Krizhevsky, et al. "ImageNet Classification with Deep Convolutional Neural Networks," NIPS 2012.

AlexNet Parameters

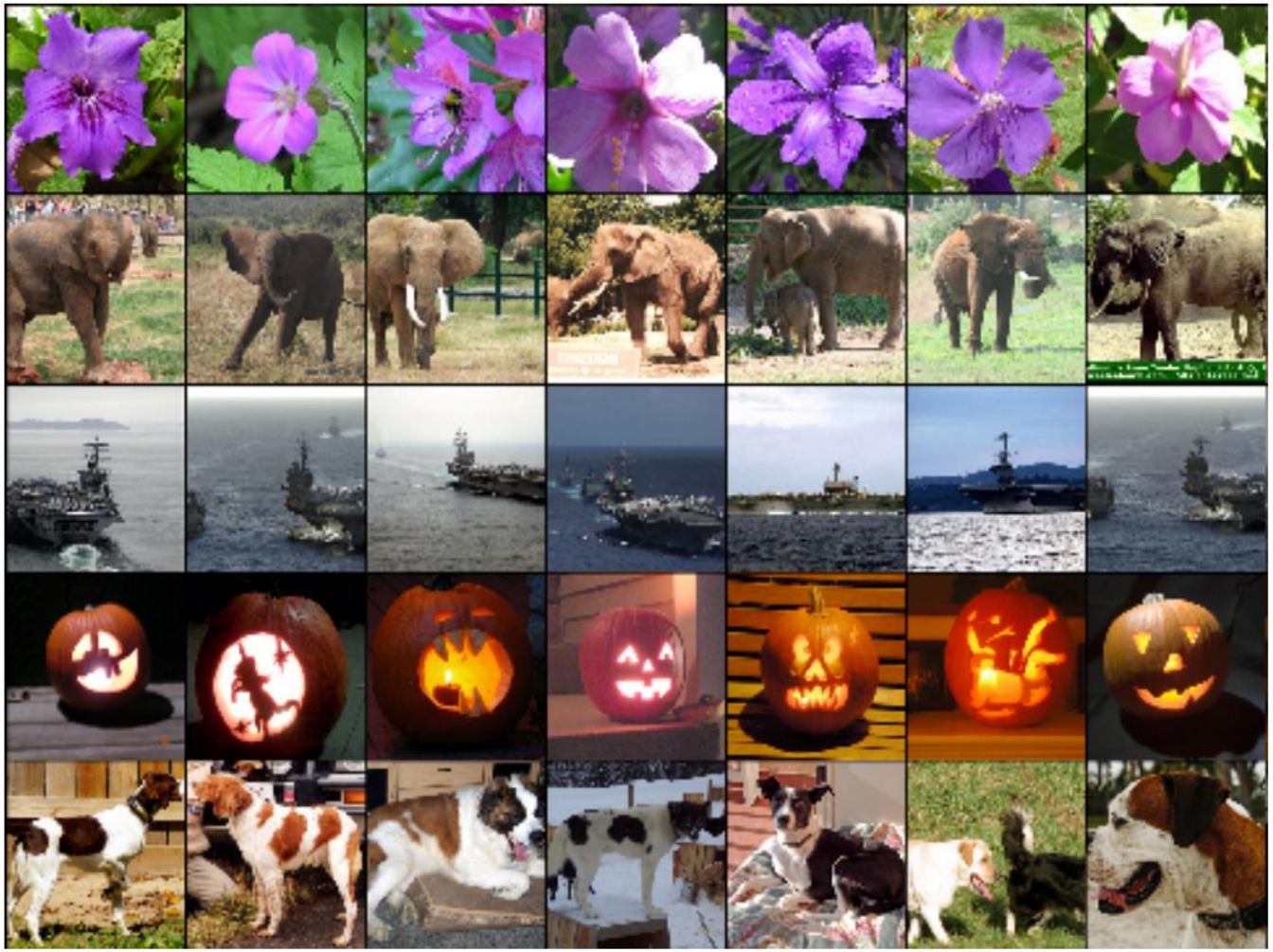
Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
-	-	-	-	-	-	-
-	-	-	-	-	-	-
-	-	-	-	-	-	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

ImageNet and ILSVRC

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



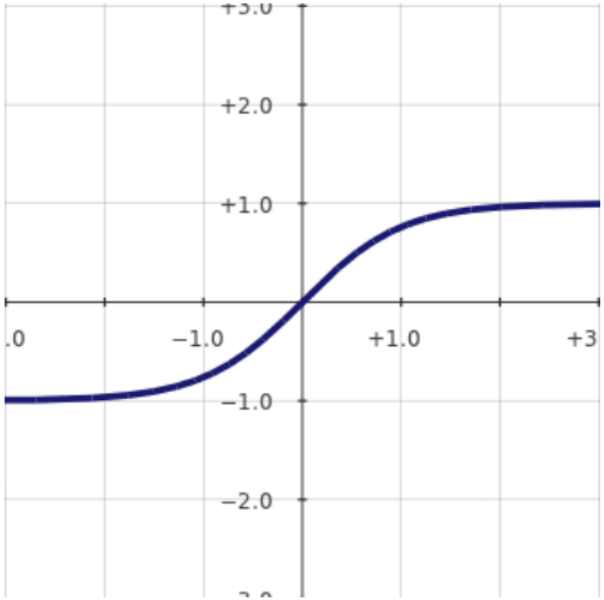
Deep Learning for AlexNet

- Deep Convolutional Neural Network
  - 5 convolutional and 3 fully connected layers
  - 650,000 neurons, 60 million parameters, 630 millions connections

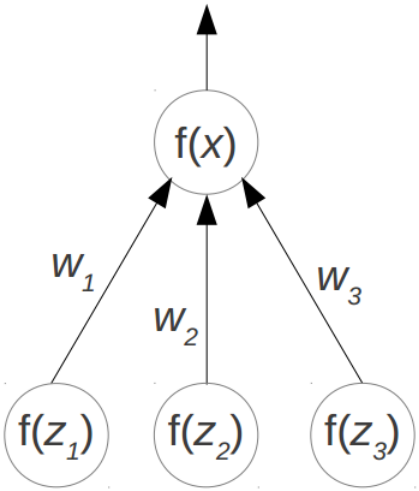
- Some techniques for boosting up performance (top 5)
  - SoftMax and Cross-Entropy Loss
  - ReLU nonlinearity for convolution layers (6x faster)
  - Overlapping Max pooling (0.5% error reduced)
  - Batch Normalization (1.2% error)
  - Data augmentation (spatial random crop, 4 corners+center & reflection, RGB perturbation) → x2048 (1% error reduced)
  - Dropout (less local minimum)

Rectified Linear Units (ReLU)

$f(x) = \tanh(x)$



Very bad (slow to train)



$x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3)$

x is called the total input to the neuron, and $f(x)$ is its output

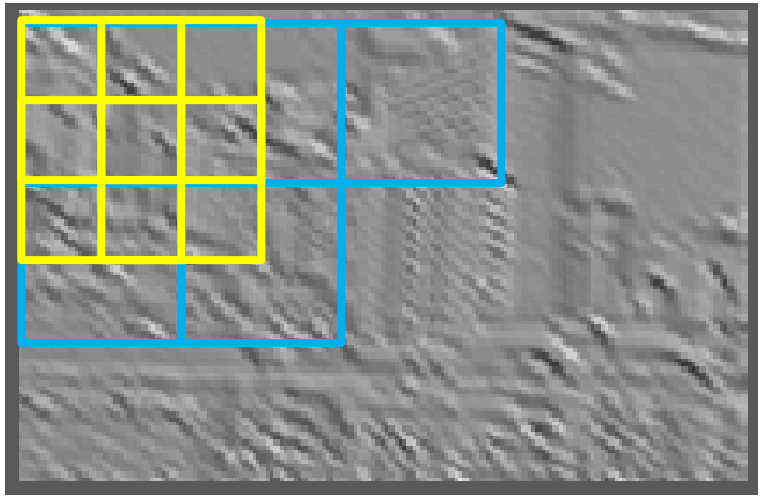
$f(x) = \max(0, x)$

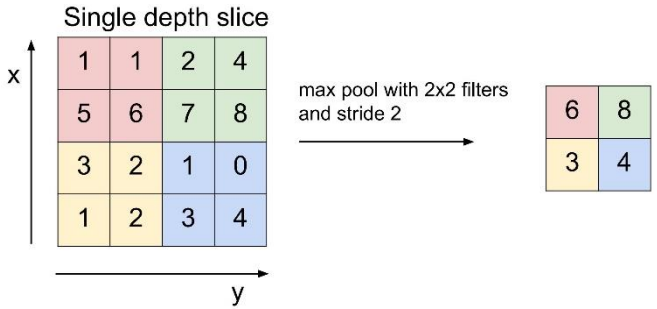
Very good (quick to train)

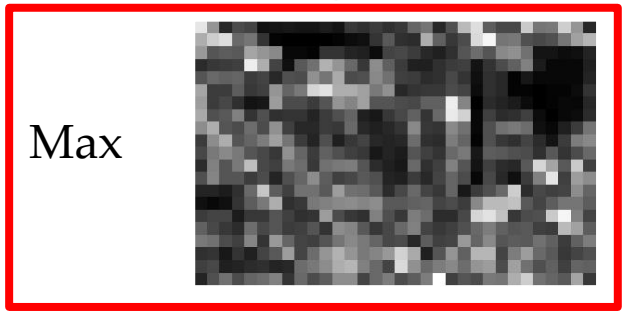
reference : http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf

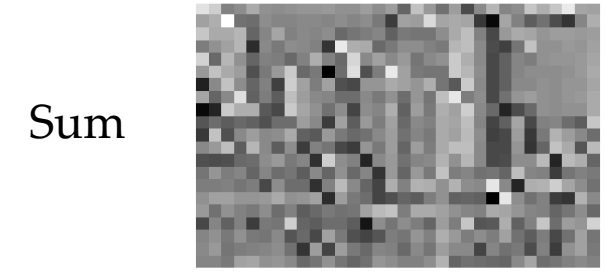
Maximum Pooling

- Spatial Pooling
  - Non-overlapping/overlapping
  - Sum or max (3x3 size)





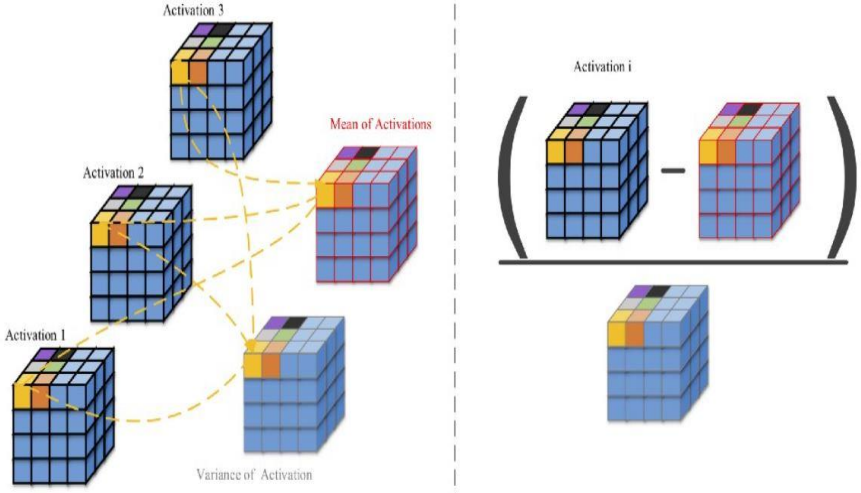




reference : http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

Batch Normalization (BN)

- To limit the unbounded activation from increasing the output layer values, normalization is used just before the activation function (e.g., ReLU) → accelerates training, better generalization.
- The normalization is carried out for each pixel across all the activations in a batch.



e.g., Batch of 3

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

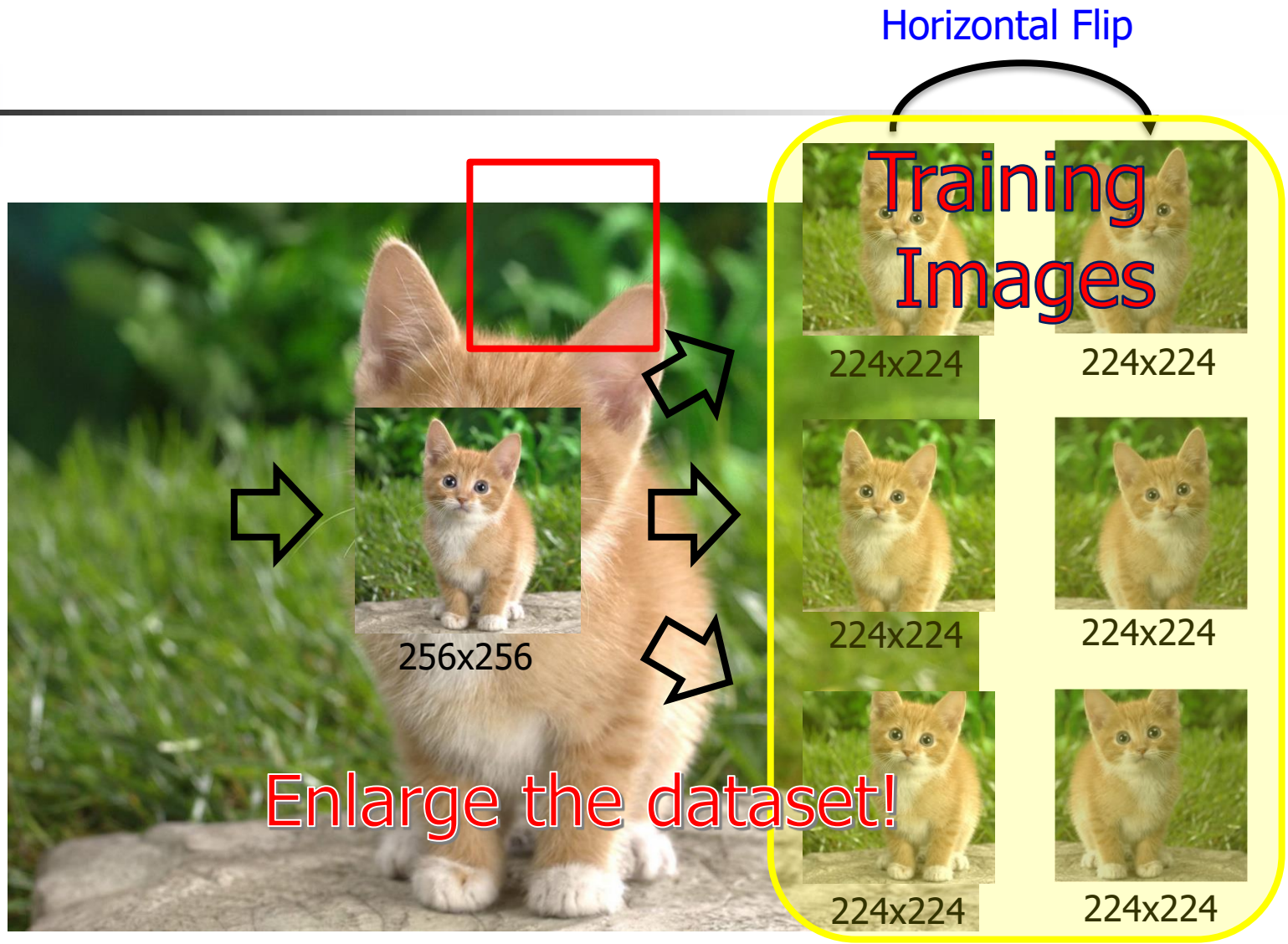
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

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

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

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

Data Augmentation



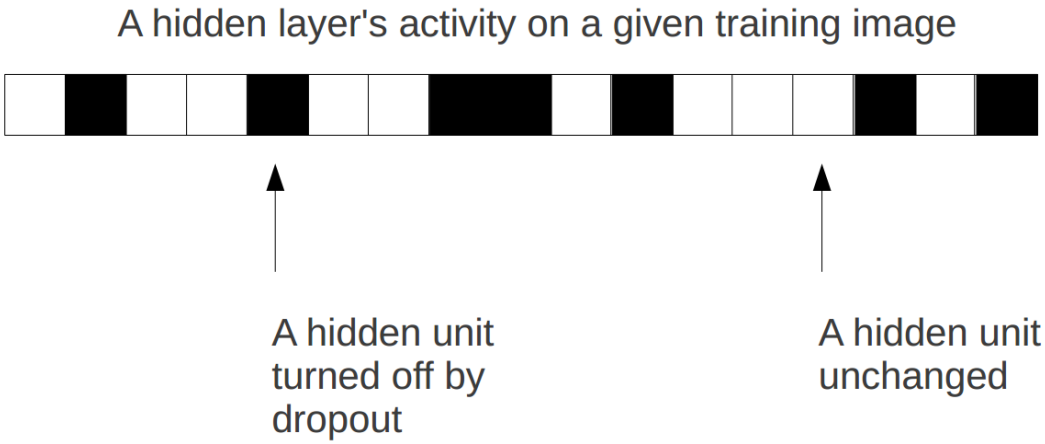
Color Data Augmentation

- Alter the intensities of the RGB channels in training images
- Perform PCA on the set of RGB pixel values
- To each training image, add multiples of the found 3 principal components to each RGB image pixel $I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]^T$ with the following quantity $[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$

- $\mathbf{p}_i$, λ_i : i-th eigenvector and eigenvalue
- α_i : random variable drawn from a Gaussian with mean 0 and standard deviation 0.1
- This reduces top-1 error rate by over 1%

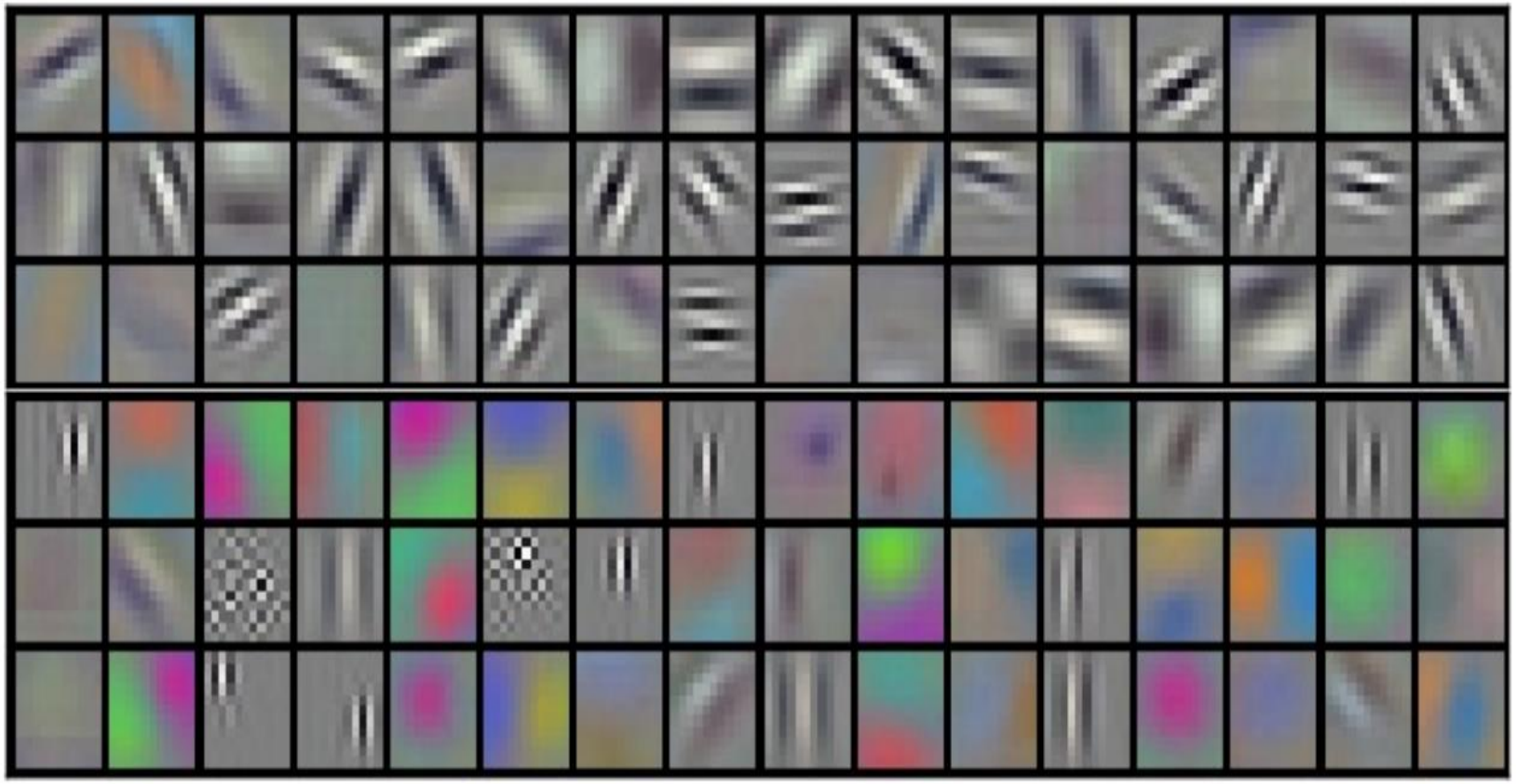
Dropout

- Independently set each hidden unit learning activity to zero with 0.5 probability for every training image
- Used in the two globally-connected hidden layers at the network's output



Exemplar Kernel Filters

- 96 learned low-level (1st layer) *11x11x3* kernels



Convolution Outputs

Input

Feature Map

reference :http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/fergus_dl_tutorial_final.pptx

Details of CNN Learning

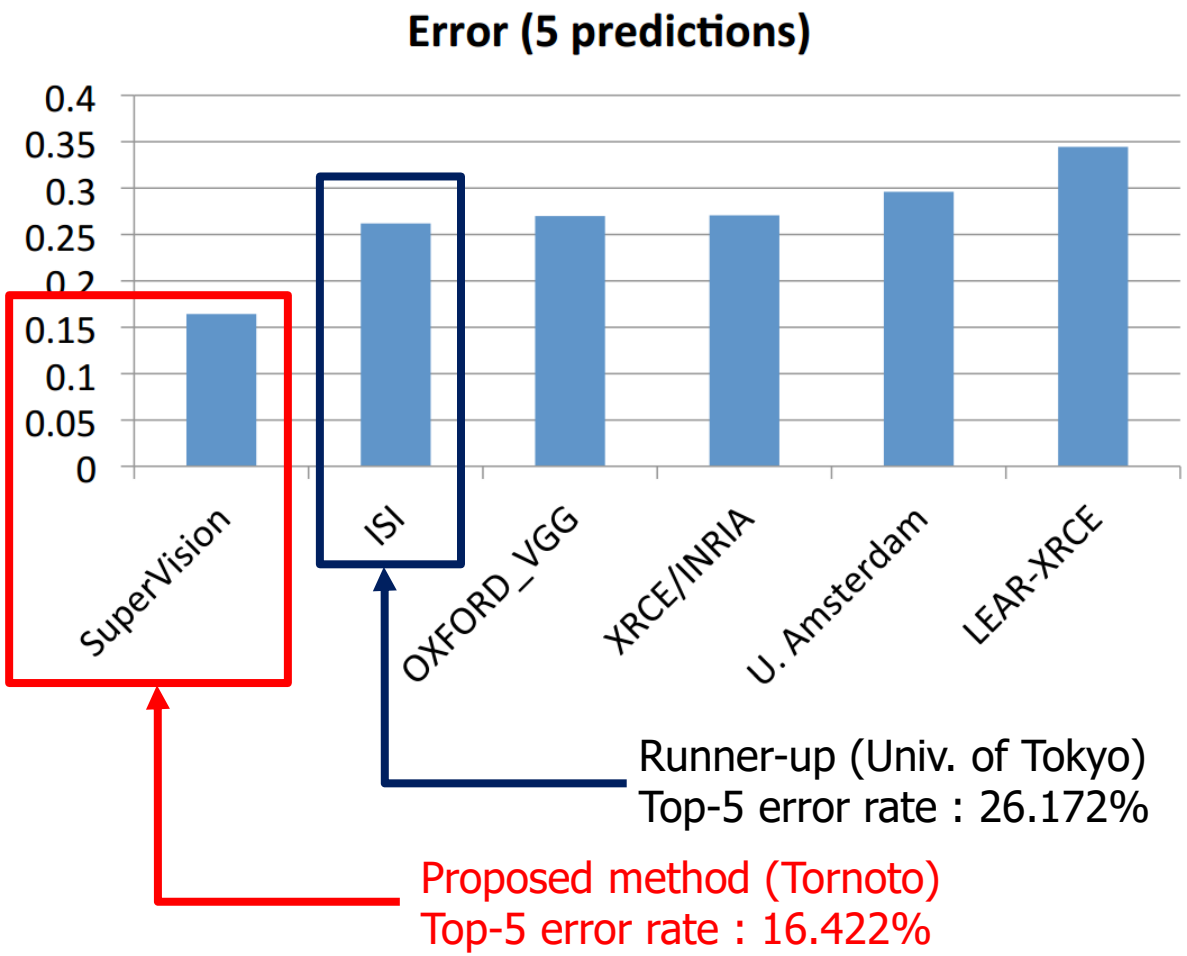
- Use stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weigh decay of 0.0005
- The update rule for weight *w* was

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

- Train for 90 cycles through the training set of 1.2 million images

ILSVRC 2012 Results



reference : http://image-net.org/challenges/LSVRC/2012/ilsvrc2012.pdf

mite

mite
black widow
cockroach
tick
starfish

container ship

container ship
lifeboat
amphibian
fireboat
drilling platform

motor scooter

motor scooter
go-kart
moped
bumper car
golfcart

leopard

leopard
jaguar
cheetah
snow leopard
Egyptian cat

grille

convertible
grille
pickup
beach wagon
fire engine

mushroom

agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers

cherry

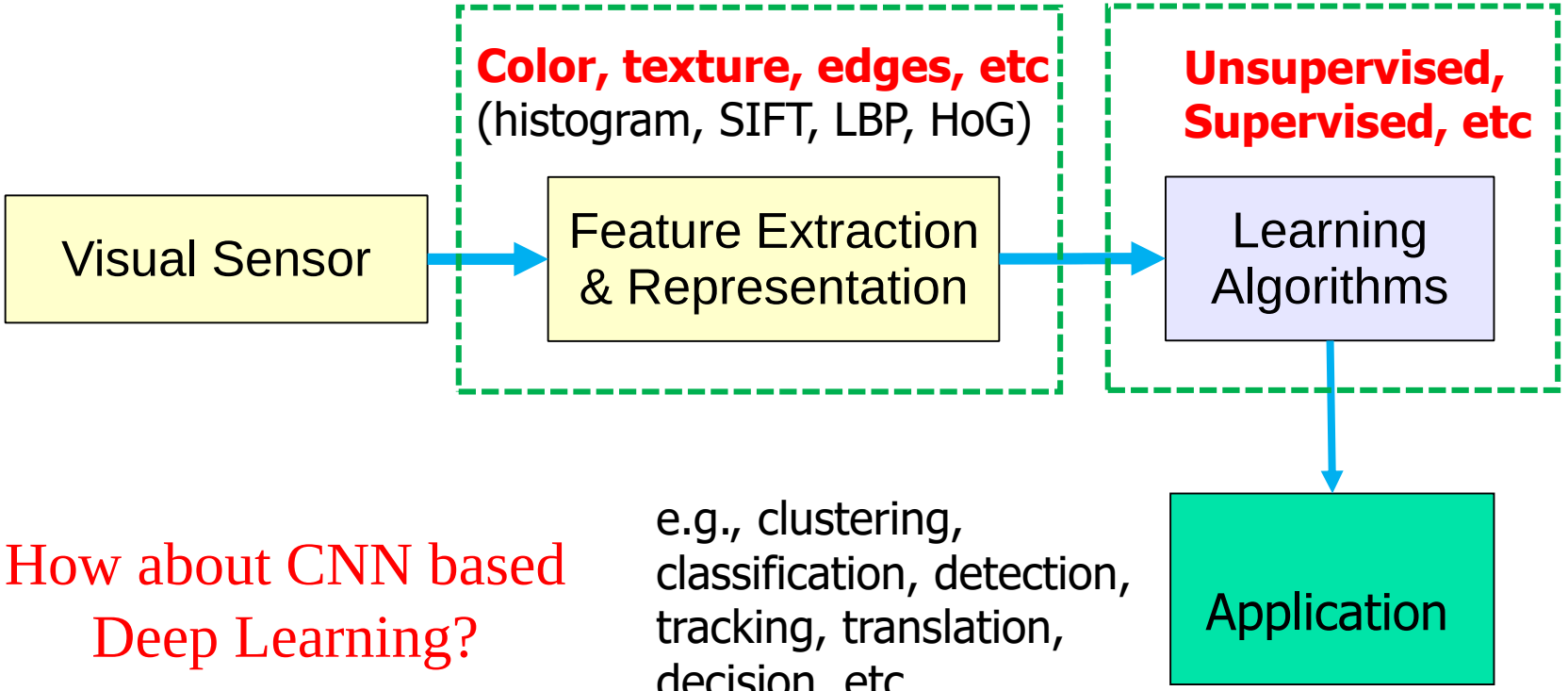
dalmatian
grape
elderberry
ffordshire bullterrier
currant

Madagascar cat

squirrel monkey
spider monkey
titi
indri
howler monkey

Recall Traditional ML

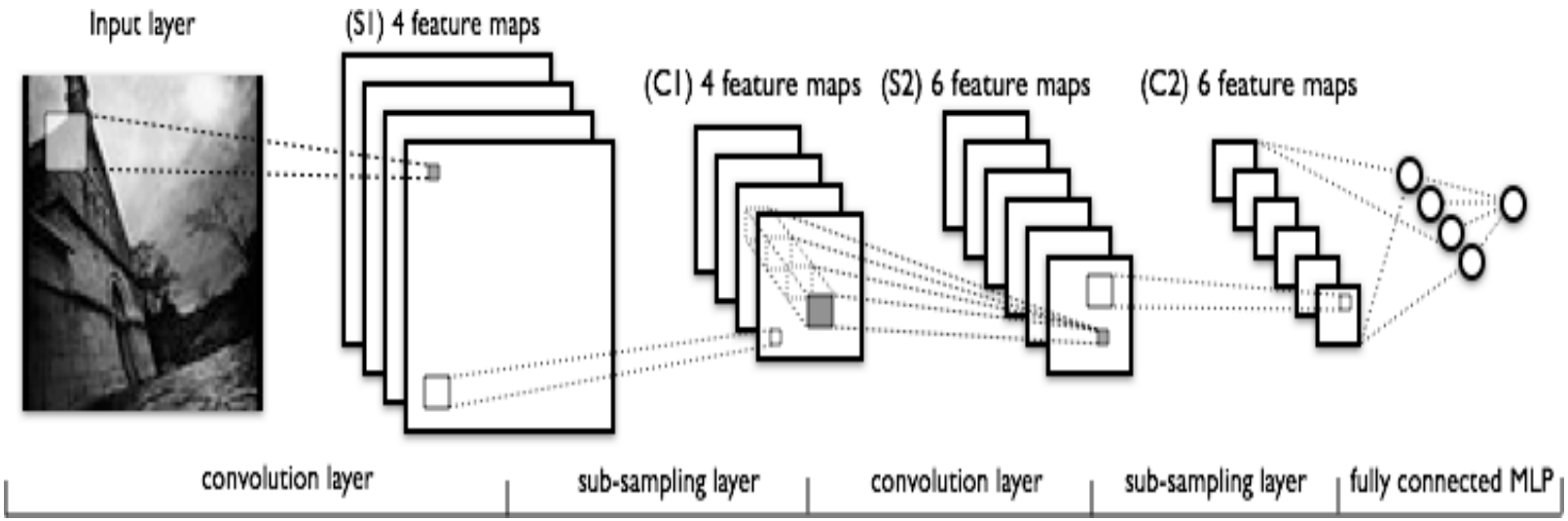
Recall the two-stage sequential machine learning



How about CNN based Deep Learning?

e.g., clustering, classification, detection, tracking, translation, decision, etc

Why CNNs Are Better?



A single-stage (end-to-end) joint machine learning (enough big training data, powerful computing resources)

Deep Learning Frameworks

Caffe
(UC Berkeley)

Caffe2
(Facebook)

Torch
(NYU / Facebook)

PyTorch
(Facebook)

Theano
(U Montreal)

TensorFlow
(Google)

Paddle
(Baidu)

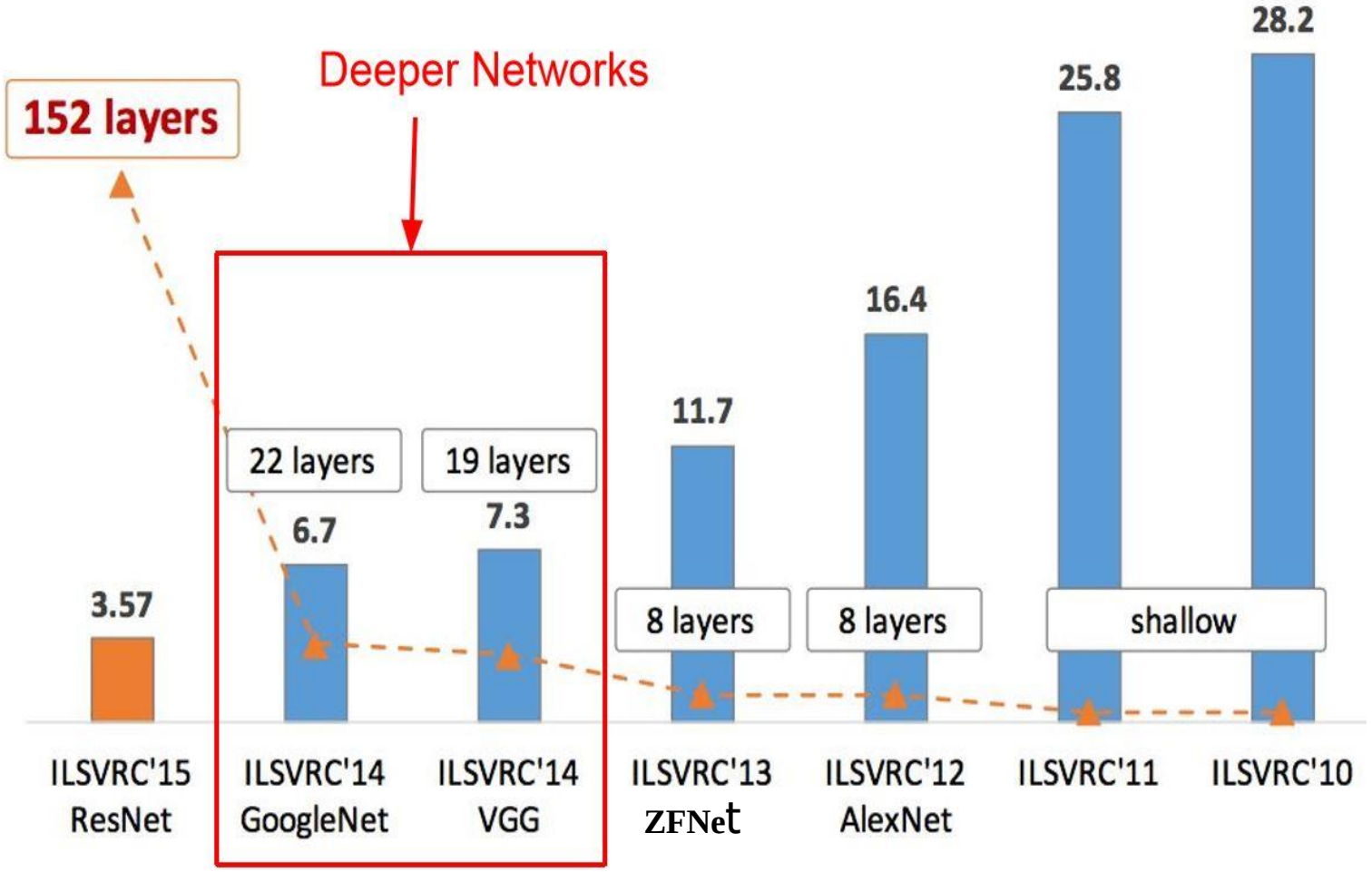
CNTK
(Microsoft)

MXNet
(Amazon)

Developed by U Washington, CMU, MIT, Hong Kong U, etc but main framework of choice at AWS

And others...

Success of CNNs for ILSVRC

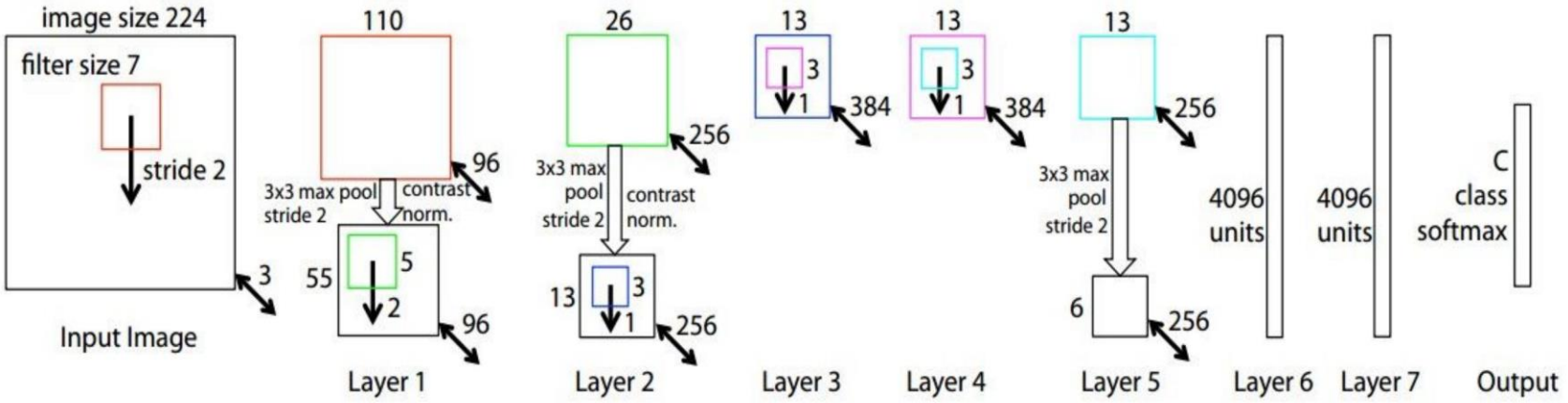


ZFNet, ILSVRC 2013 Winner

- Similar to 8-layer AlexNet but:
  - CONV1: change from (11x11 stride 4) to (7x7 stride 2)
  - CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ZFNet

[Zeiler and Fergus, 2013] NYU



Oxford VGG Net

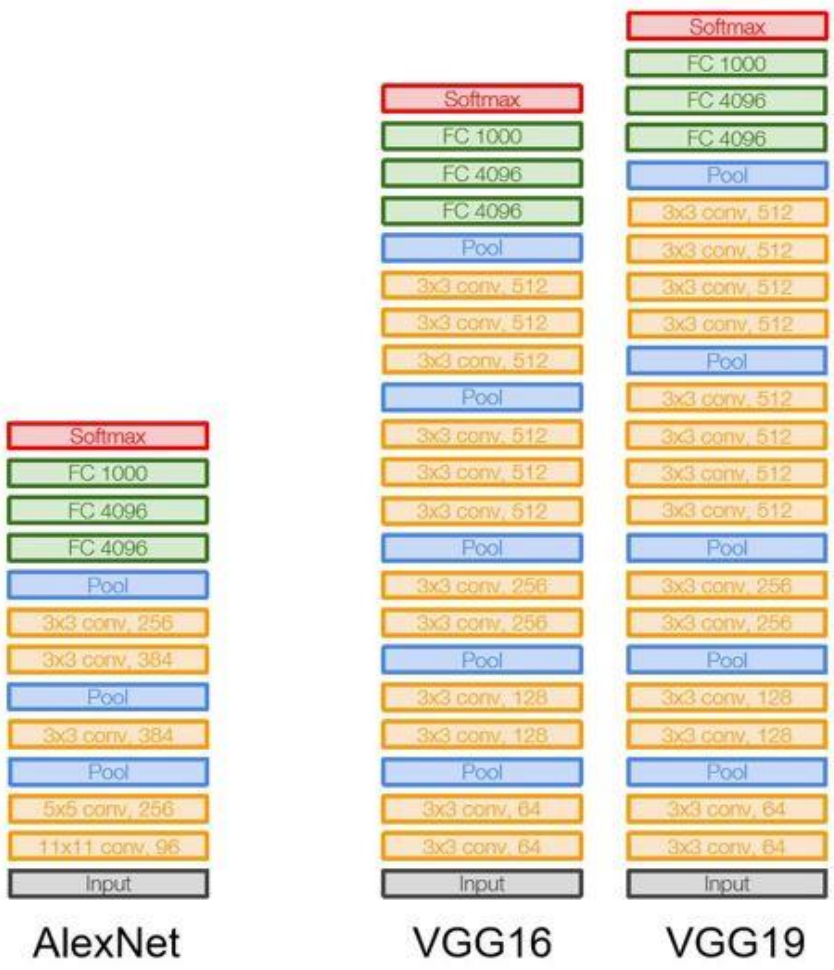
Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

[Simonyan and Zisserman, 2014]

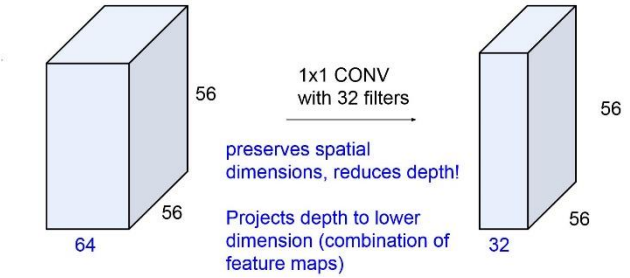


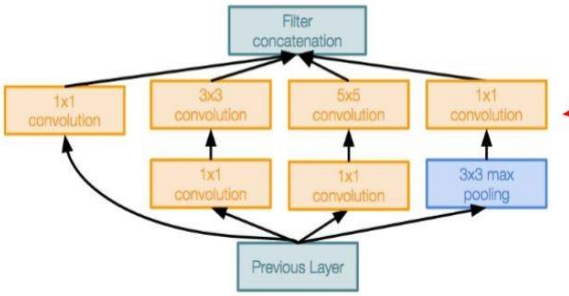
GoogLeNet

Deeper networks, with computational efficiency

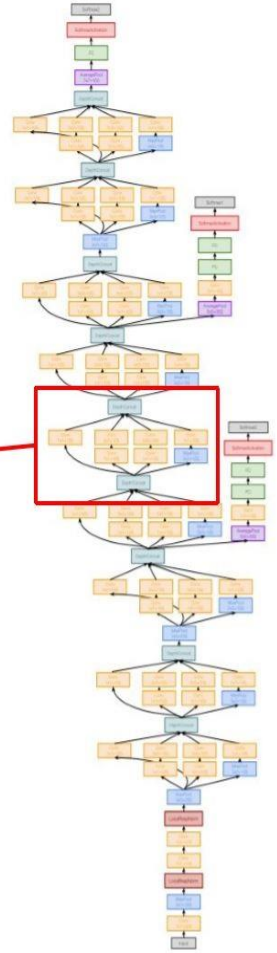
- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC'14 classification winner
  (6.7% top 5 error)

[Szegedy et al., 2014]





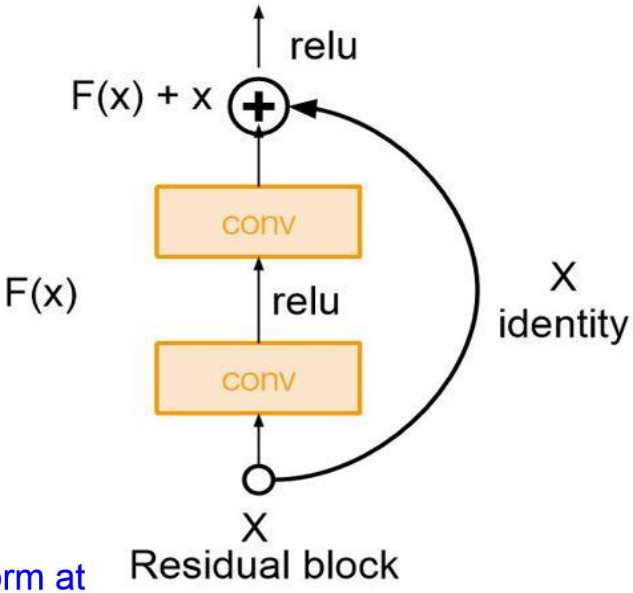
Inception module



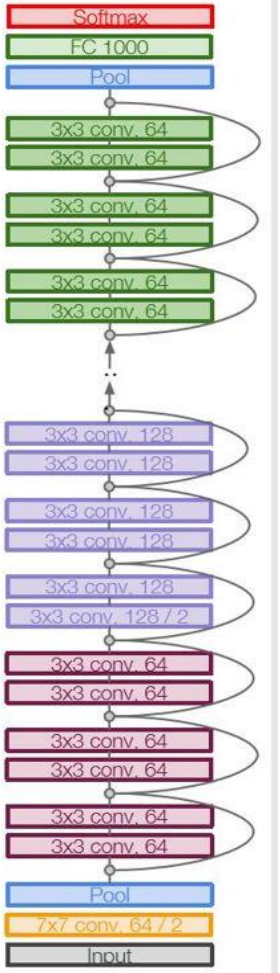
Microsoft ResNet

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



[He et al., 2015]



The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Visual Machine Learning

Classification

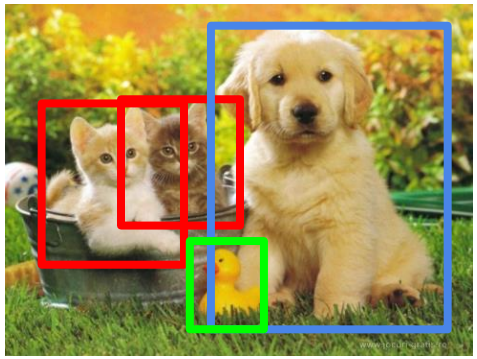
CAT

Classification + Localization

CAT

Single object

Object Detection



CAT, DOG, DUCK

Instance Segmentation

CAT, DOG, DUCK

Multiple objects

Classification + Localization Task

Classification: C classes

Input: Image

Output: Class label

Evaluation metric: Accuracy

Loss Function: cross entropy

CAT

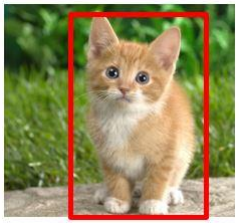
Localization:

Input: Image

Output: Box in the image (x, y, w, h)

Evaluation metric: Intersection over Union

Loss Function: L2

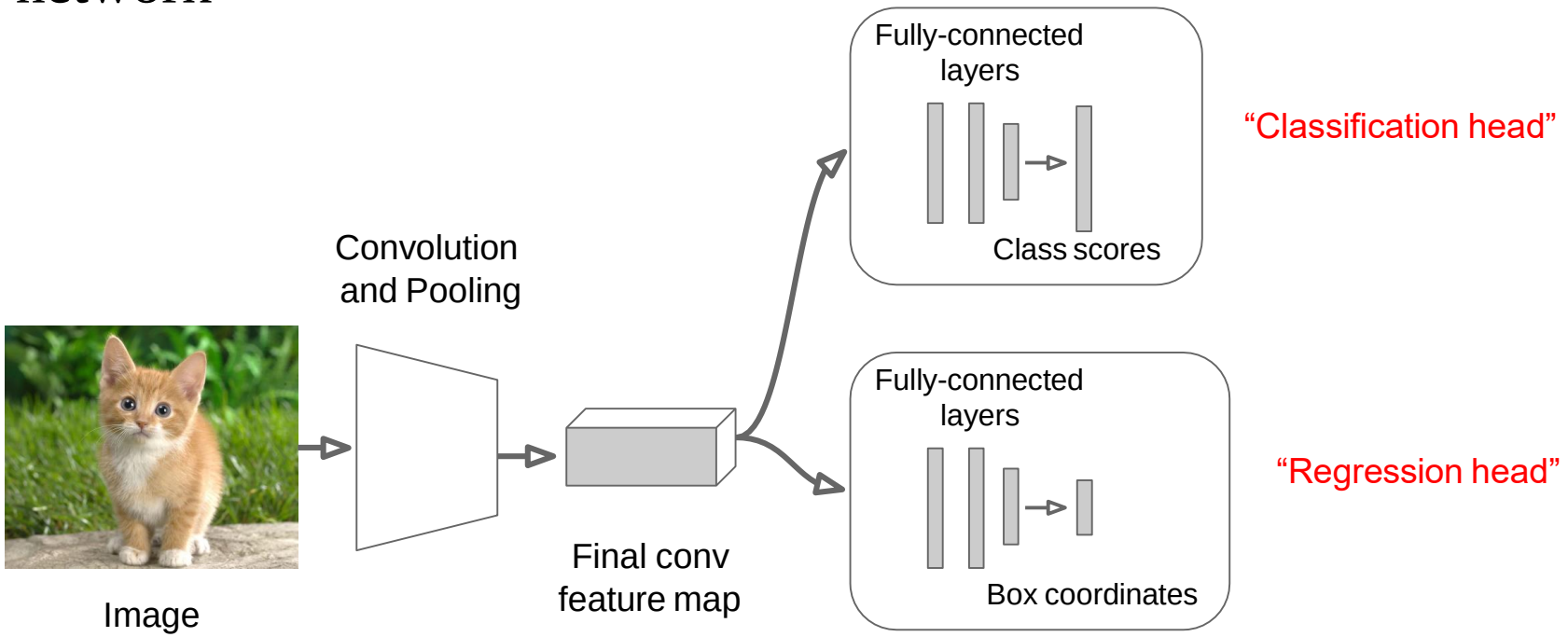


(x, y, w, h)

Classification + Localization: Do both

Training for Classification + Localization

Step 2: Attach new fully-connected "regression head" to the network

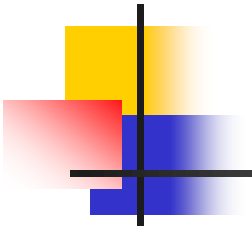


Detection as Regression?

CAT, (x, y, w, h)
CAT, (x, y, w, h)
....
CAT (x, y, w, h)

How many?

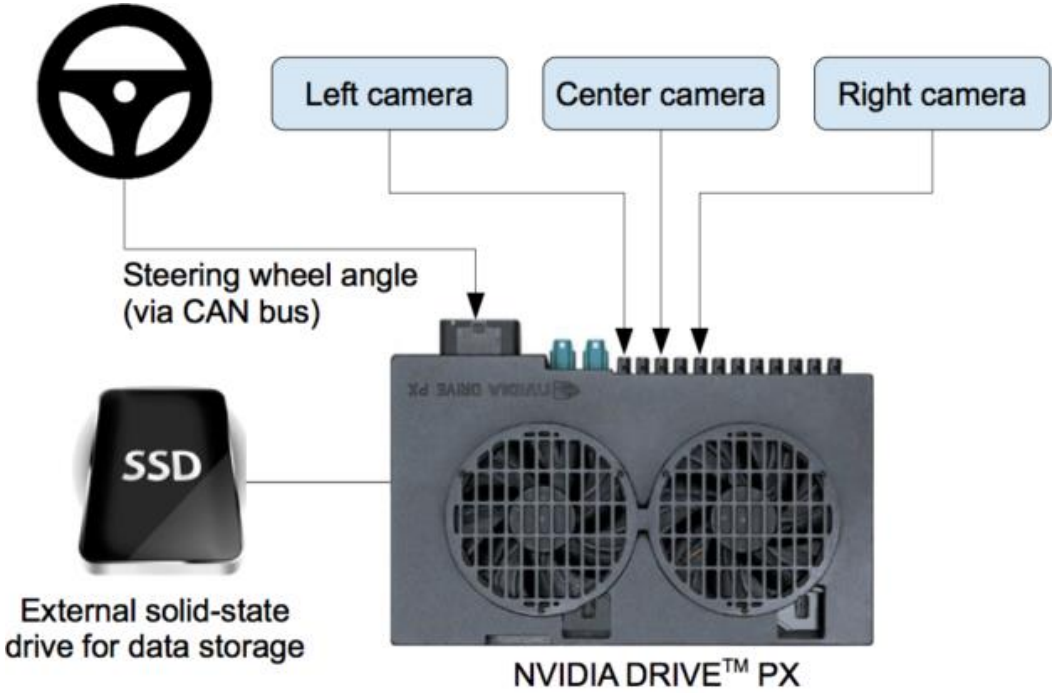
Need variable sized outputs → New Architectures



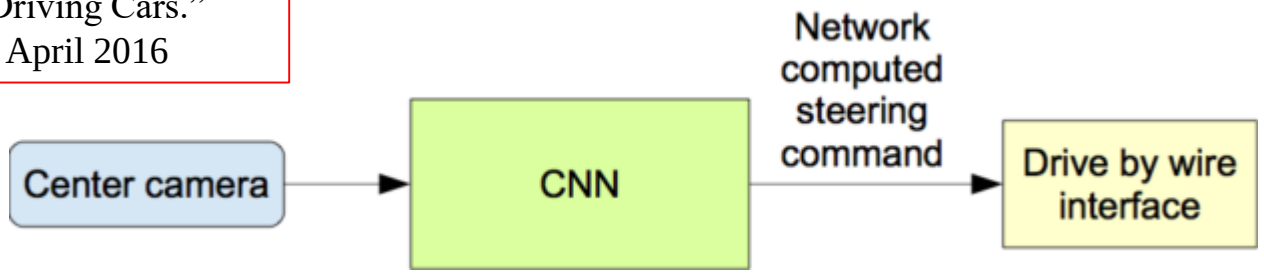
Deep Learning Regression for Autonomous Driving

Nvidia DAVE-2 (DARPA Autonomous Vehicle)

CAN bus allows electronic control units and devices to communicate with each other.

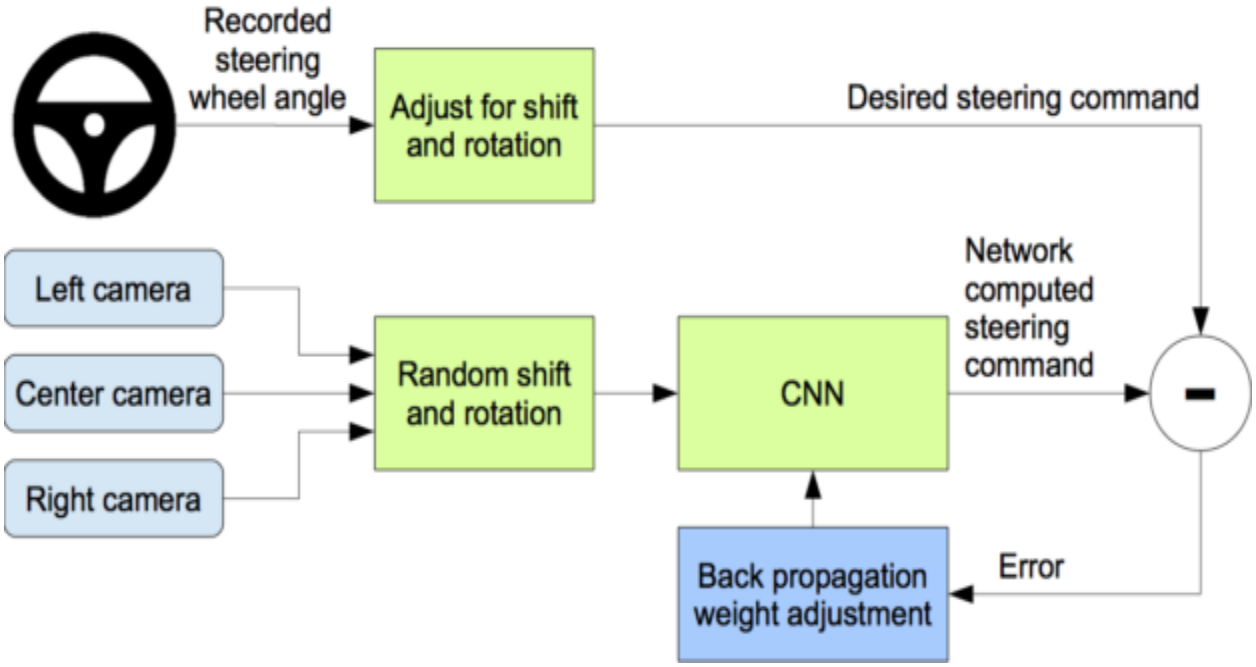


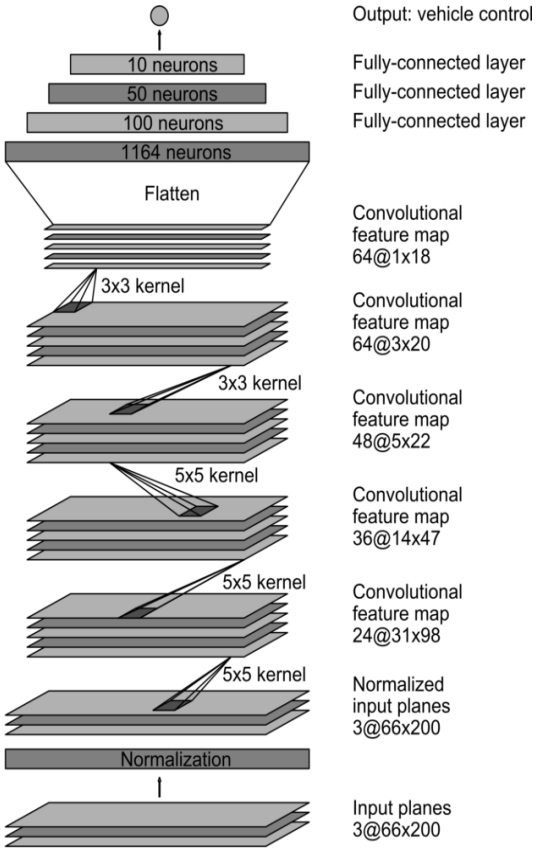
Mariusz Bojarski, et al. "End to End Learning for Self-Driving Cars." arXiv:1604.07316, April 2016



Training the CNN

- To learn how to recover from mistakes, the training data is augmented with additional images that show the car in different shifts from the center of the lane and rotations from the direction of the road (72 hours).



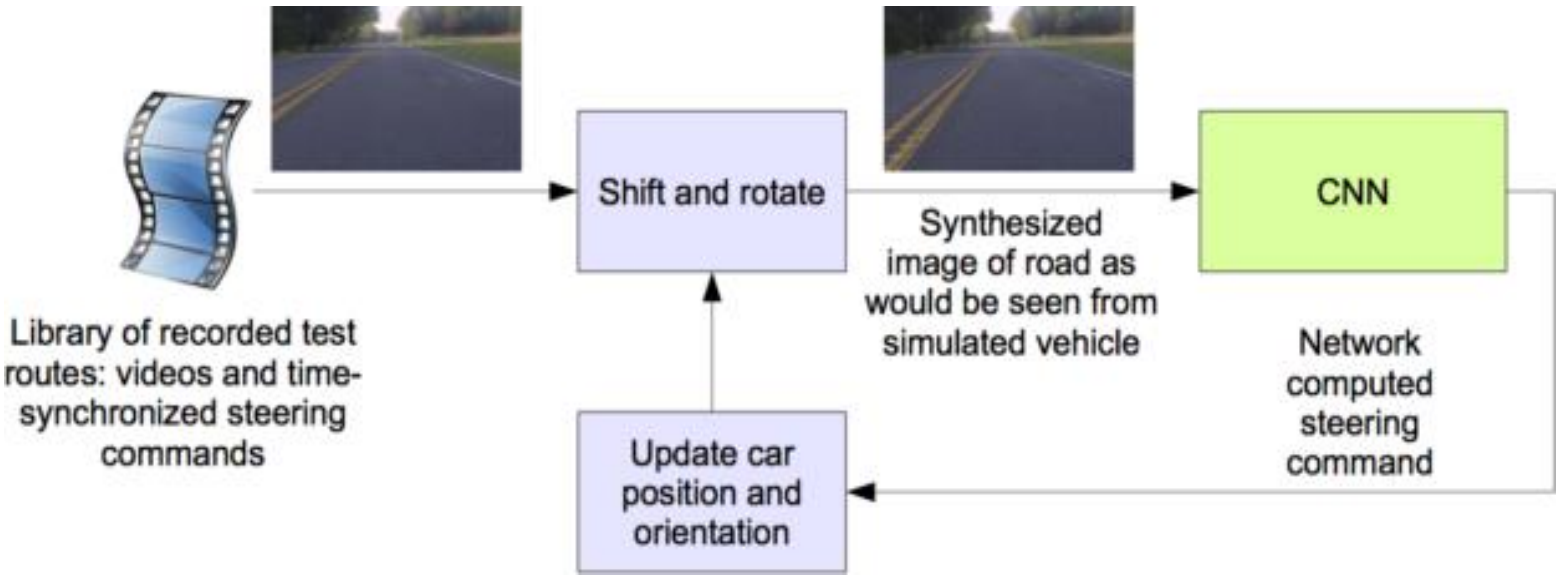


Training Data Collections

- The collected data is labeled with road type, weather condition, and the driver's activity (staying in a lane, switching lanes, turning, and so forth).
- Activity Dependent Training: to train a CNN to do lane following we only select data where the driver was staying in a lane and discard the rest.
- Video are sampled at 10 FPS, as a good compromise.

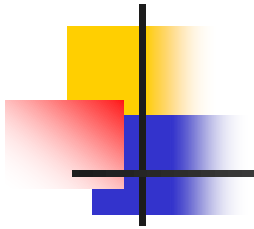
Data Augmentation

- Artificial shifts and rotations images are added to teach the network how to recover from a poor position or orientation.
- The random magnitude has zero mean, with standard deviation twice the standard deviation measured with human drivers.




Dave-2 A Neural Network Drives A Car
0:24
1:27 / 14:13


Dave-2 A Neural Network Drives A Car
In-vehicle camera
2:40 / 14:13



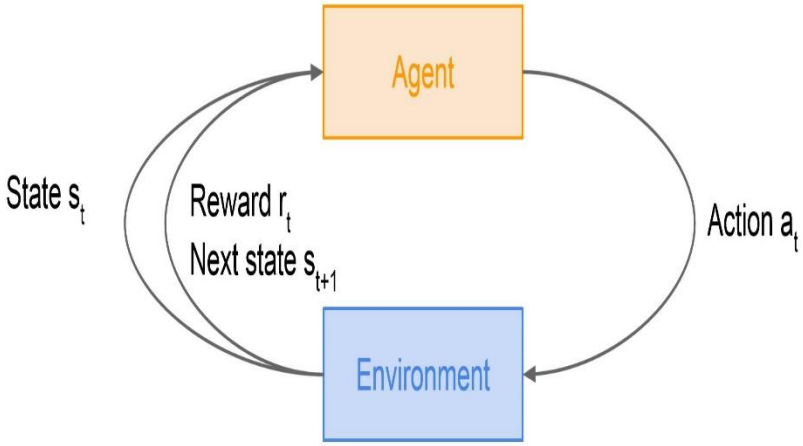
Deep Reinforcement Learning and AlphaGo

(a combination of classification and regression tasks)

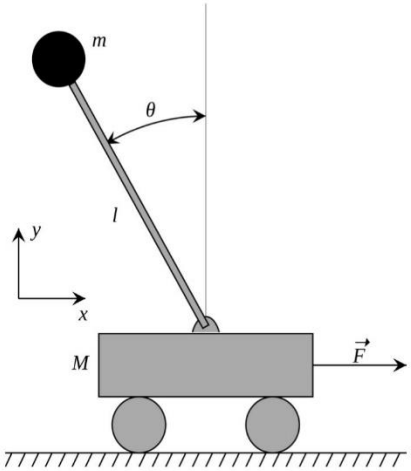
Reinforcement Learning (RL)

- How can an agent learn behaviors when it doesn't have a teacher to tell it how to perform, except a delayed scalar feedback (a number called reward).
- The goal is to get the agent to act, by learning from interaction with environment, to maximize its future rewards

- Potential applications
  - Backgammon/chess playing
  - Game playing (Atari)
  - job or shop scheduling
  - Car-pole control



Examples of RL



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Atari 2600 games

Markov Decision Process

- * At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
  Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(\,.\,|\,s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\,.\,|\,s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}
- Find optimal policy π that maximizes the expected sum of rewards!

$$\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right] \quad \text{with} \quad s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

- The value function at "state s", is the expected cumulative reward

$$V^\pi(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

γ : discount factor

Q-Learning & Bellman Equation

- The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a joint "(state, action)" pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

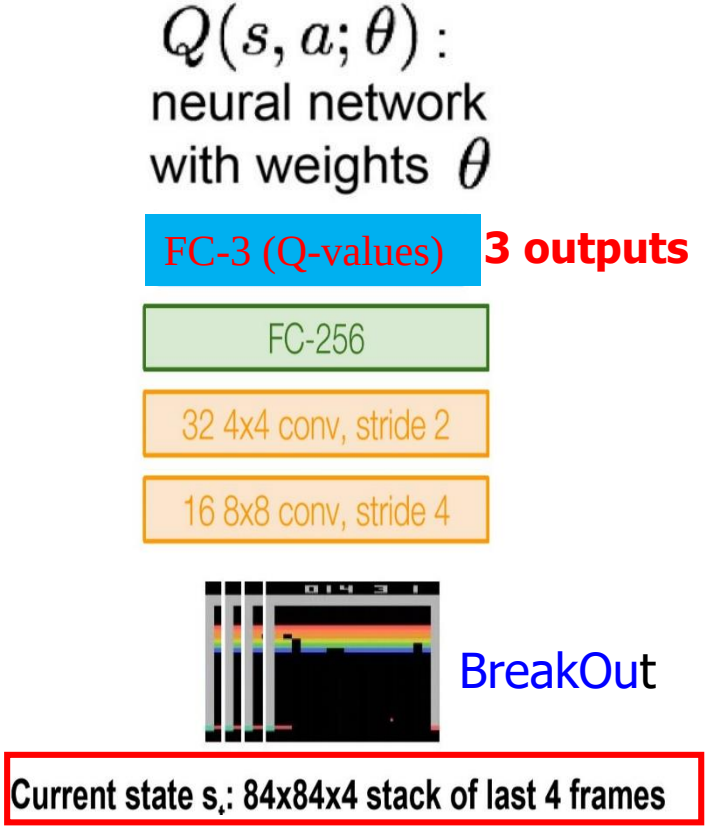
- Q^* satisfies the following Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

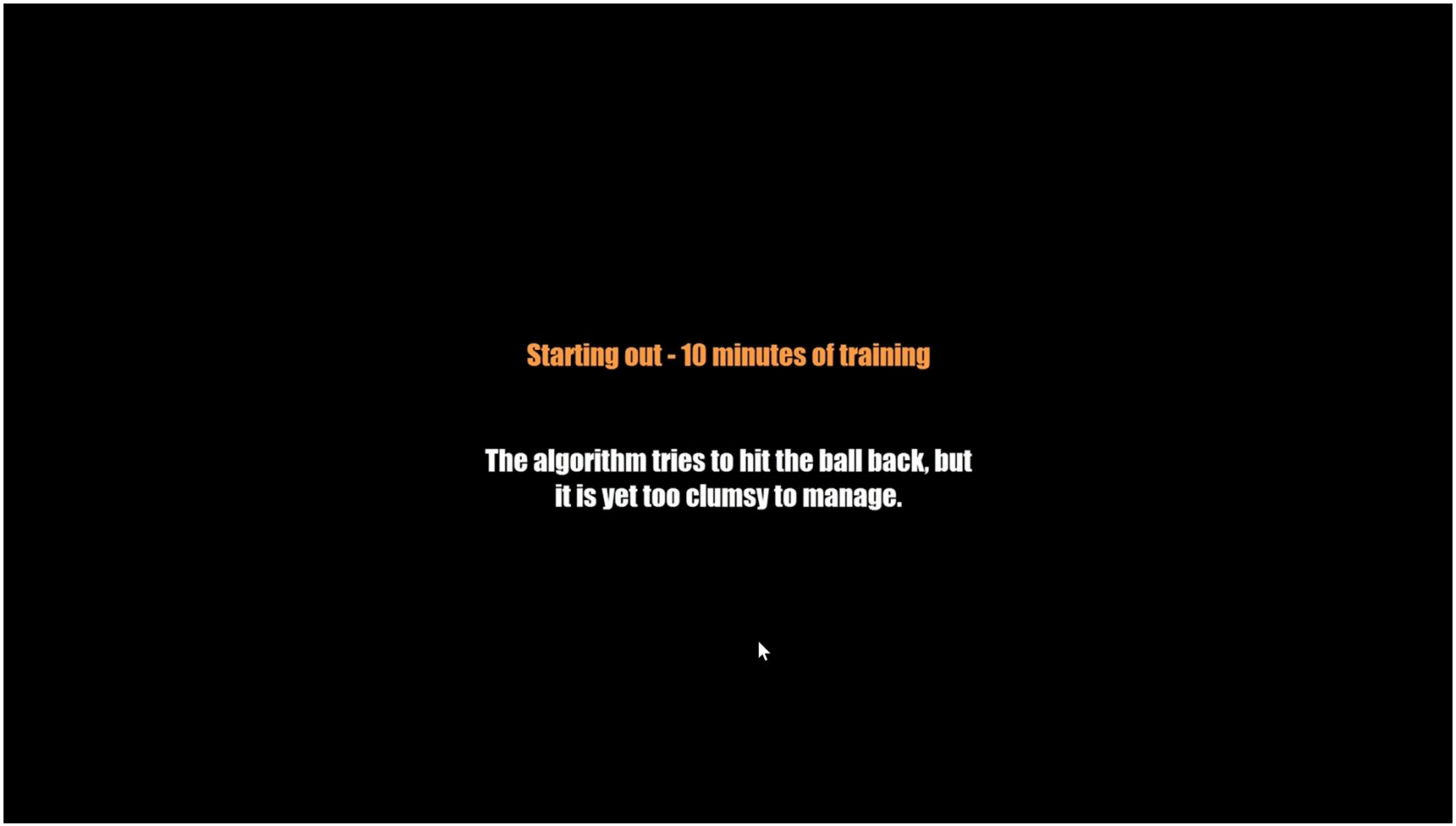
- If the optimal state-action values for the next time-step $Q^*(s',a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$

Deep CNN based Q-Learning

- Atari game: Complete the game with the highest score
  - State: Raw pixel inputs of the game state
  - Action: Game controls e.g., Left, Right, Still
  - Reward: Score increase/decrease at each time step
- Maximum expected cumulative reward Q(s, a, Θ)

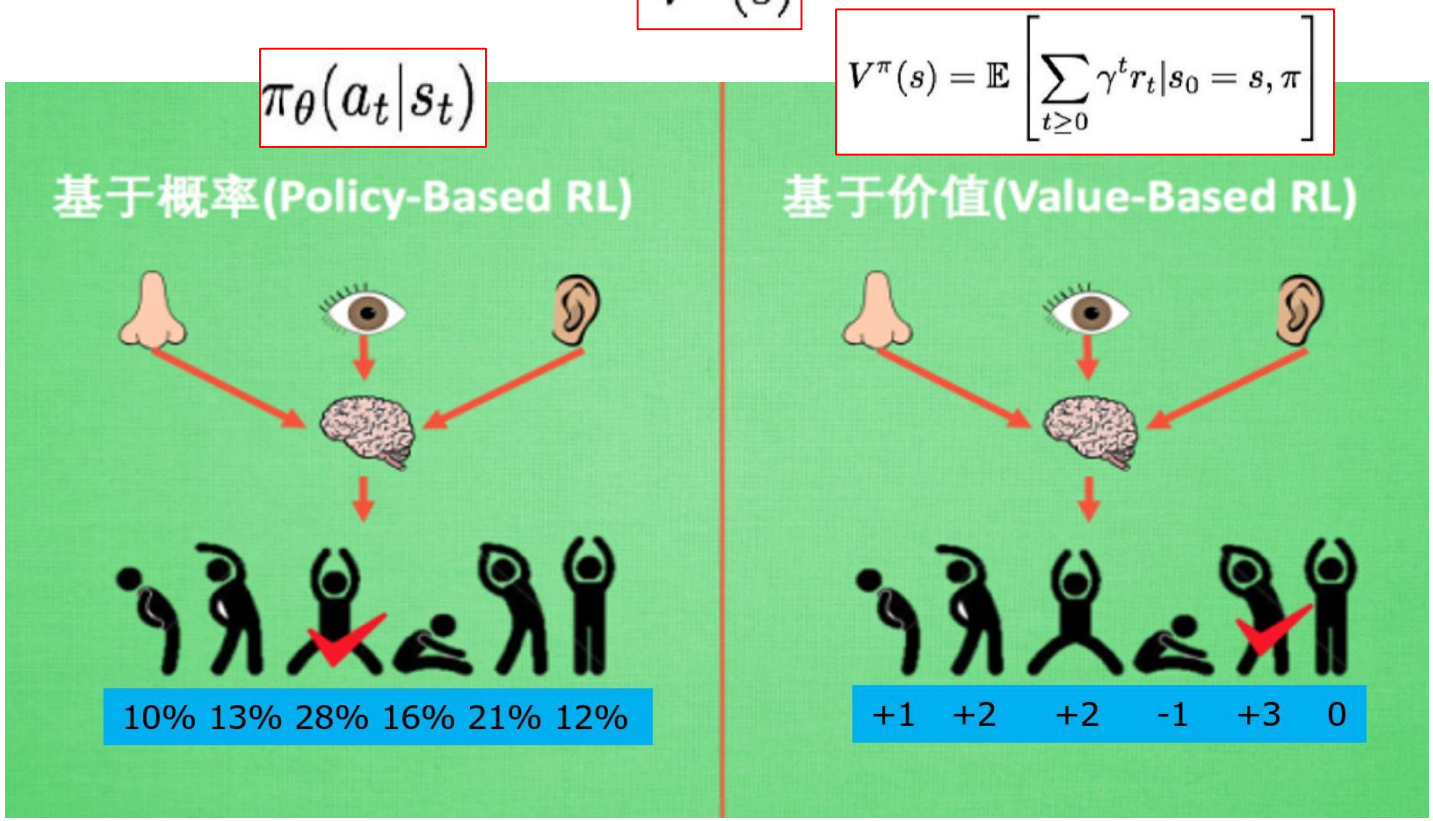


DeepMind Q-Learning CNN



Policy-based and Value-based RL

- Learn a policy function $\pi_\theta(a_t|s_t)$, take action "a" given state "s" and a value function $V^\pi(s)$



Computer Based Go Games

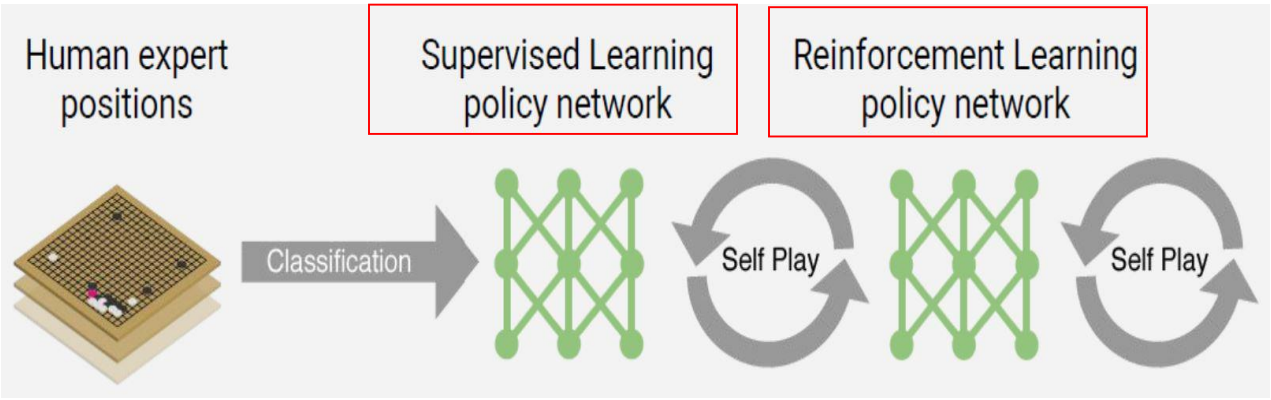
- Brute force search intractable:
  - Search space is huge
  - "Impossible" for computers to evaluate

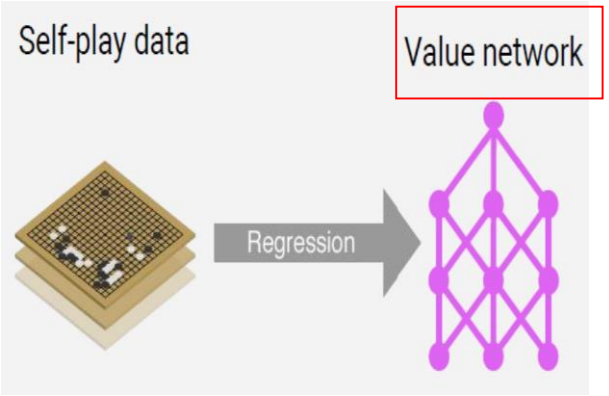
AlphaGo

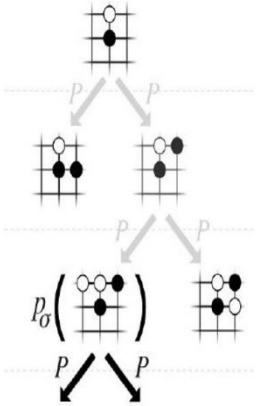
- AlphaGo, developed by Google DeepMind in London, the first Computer Go program to beat a professional human Go player, 2-dan Hui Fan, in a 5-0 match, without handicaps on a full-sized 19×19 board in Oct. 2015
- In March 2016, it beat 9-dan Sedol Lee, in a 4-1 match.

David Silver, et al., "Mastering the game of Go with deep neural networks and tree search," *Nature* (529):484–489, 2016

Supervised Learning (SL) + Reinforcement Learning (RL)



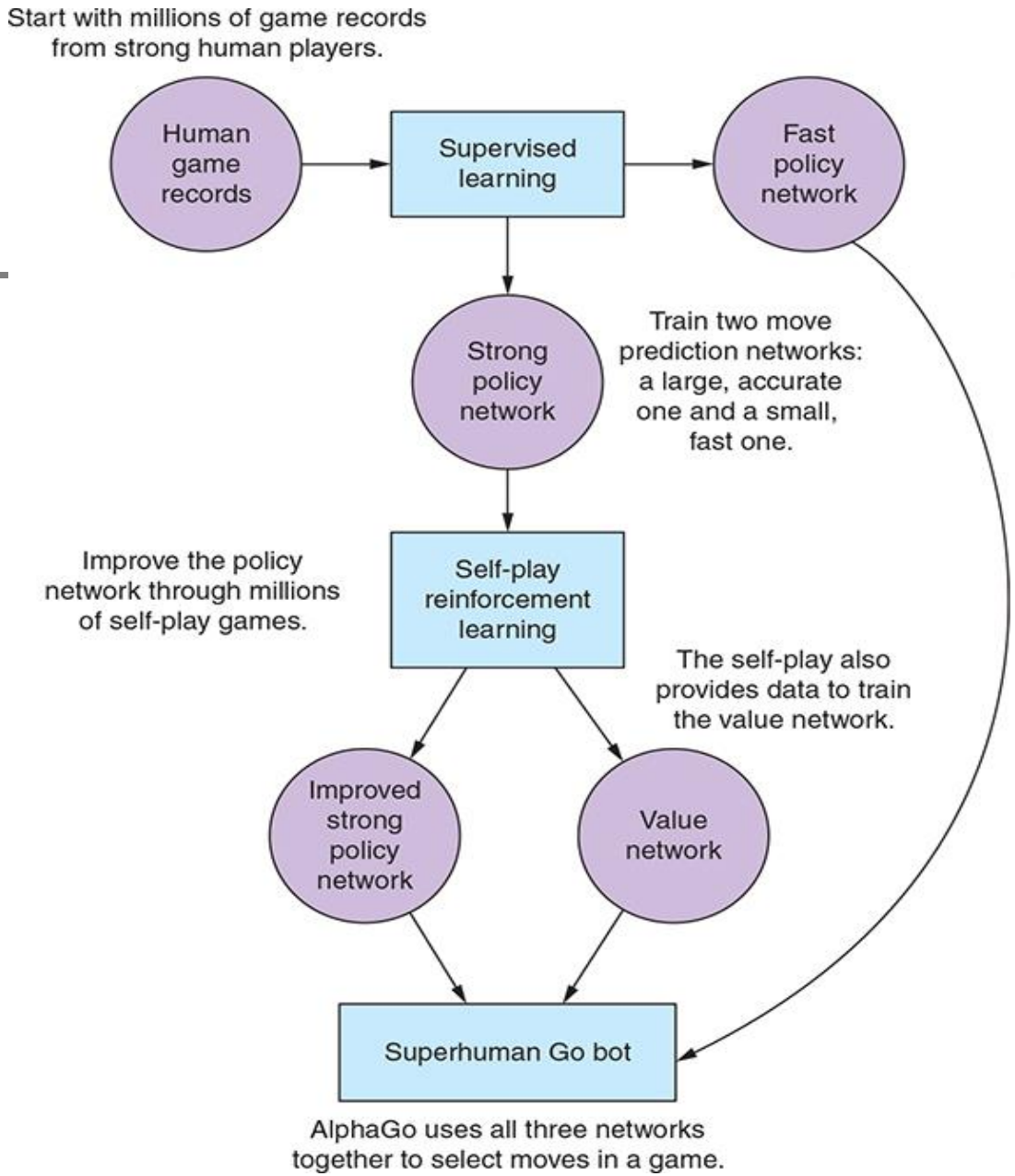




Monte Carol Tree Search (MCTS)

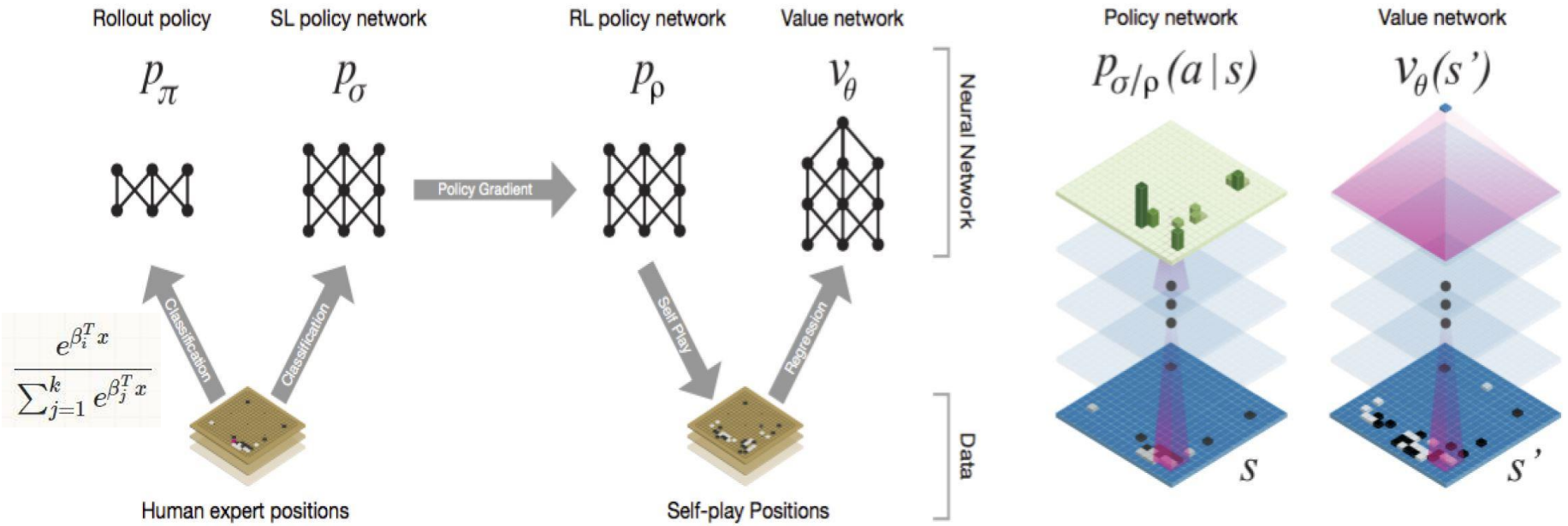
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016).

How AlphaGo Works



Two CNNs

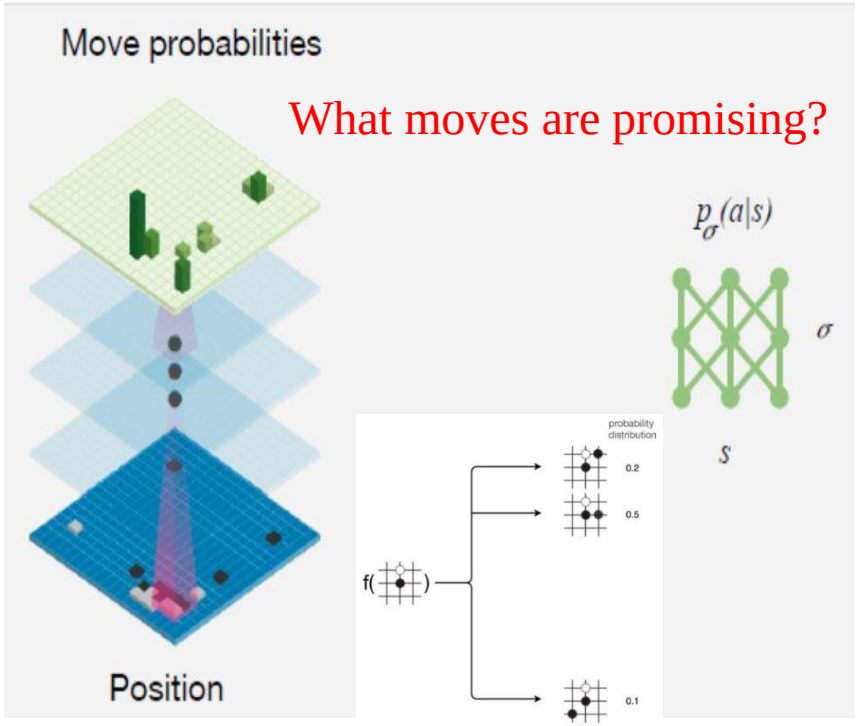
- One SL-trained policy CNN , P_σ , further RL-trained policy CNN, P_ρ ,and one RL value CNN V_θ

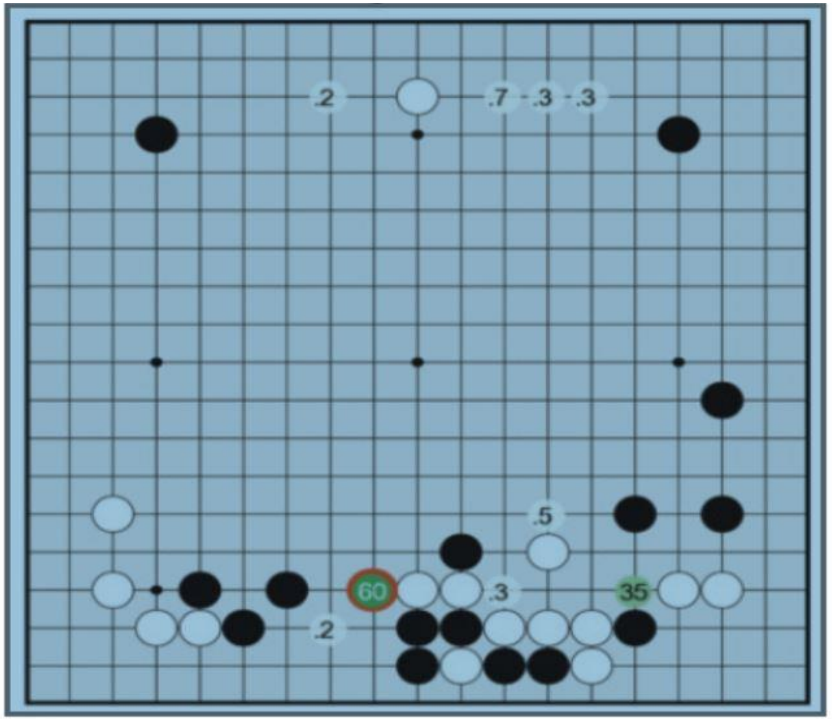


Single-layer Rollout policy P_Π for fast $p(a|s)$ based on linear softmax

A Policy Network

- A SL Policy Network to serve as a "move picker"
- 19x19x2 outputs, one for white and one for black



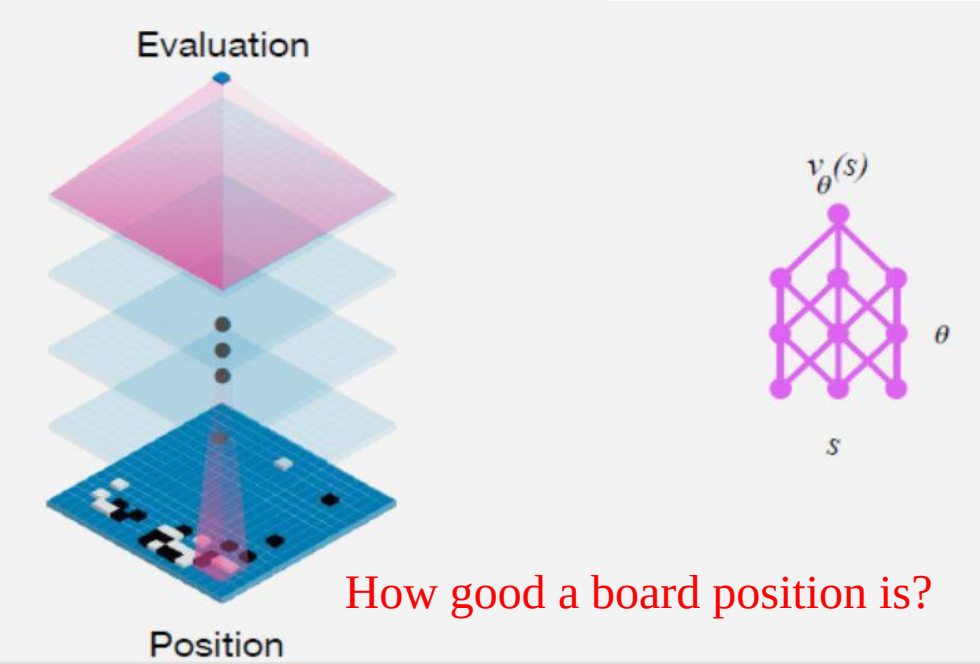


A Value Network

- A RF Value Network as a "position evaluator" – chance of winning

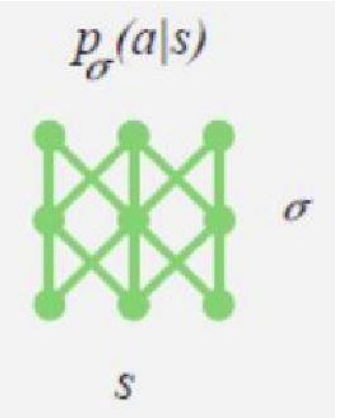
$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \ldots T} \sim p]$$

policy



How good a board position is?

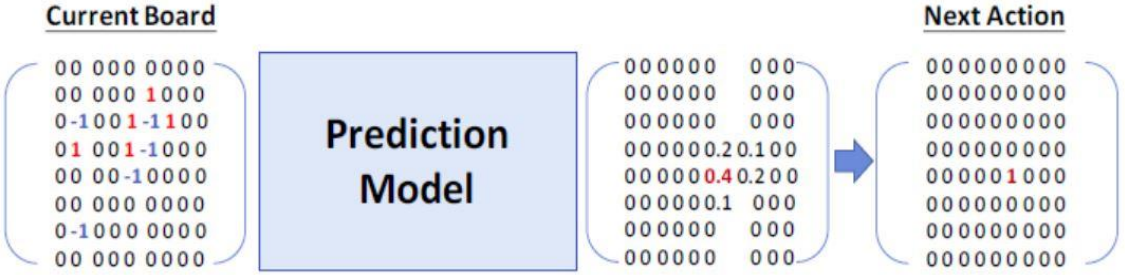
Supervised Learning of Policy Network



- Policy network: 13-layer convolutional neural network
- Training data: 160K games, 39M positions from human expert games (KGS, 5+ dan)
- Training algorithm: maximize likelihood (softmax output)

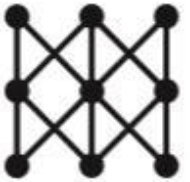
$$\Delta\sigma \propto \frac{\partial \log_\sigma p_\sigma(a \mid s_t)}{\partial \sigma} \quad \text{stochastic gradient ascent}$$

- 4-weeks training on 50 GPUs (TPUs) using Google Cloud

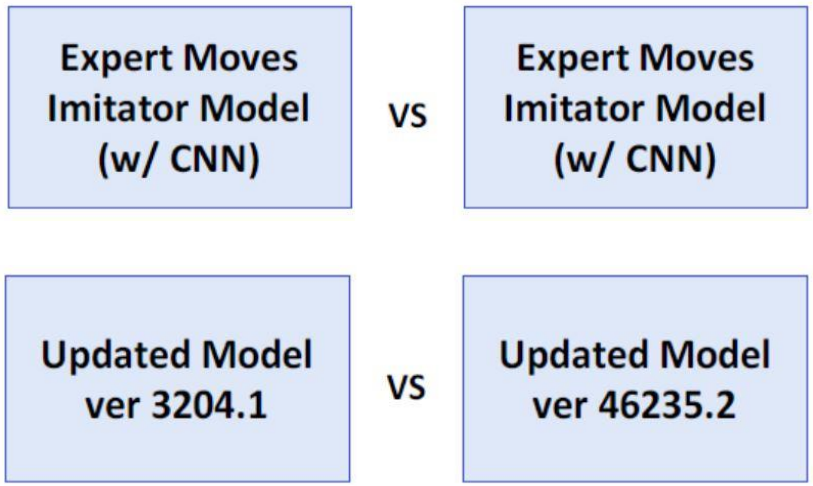


There are 19x19=361 possible actions (with different probabilities)

Reinforcement Learning of Policy Network



- Policy network: 13 layer convolutional neural network
- Training data: games of self-play between policy nets, different versions of updated policy nets
- 1 week training on 50 GPUs using Google Cloud

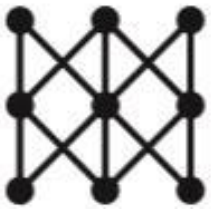


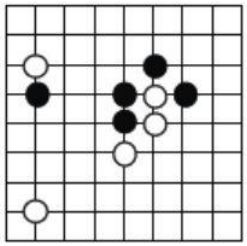
Return: board positions, win/lose info

30 million games

Reinforcement Learning of Policy Network

p_ρ





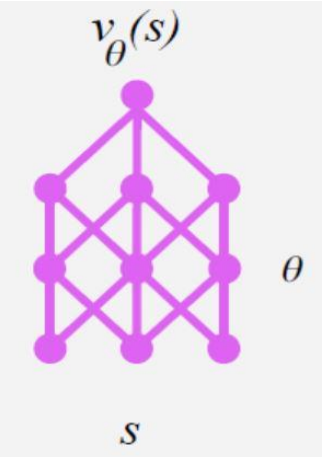
Mainly used to create "z" for training value network

- Training algorithm: maximize wins *z* by policy gradient reinforcement learning, a reward function *r(s)* that is zero (p_t=0) for all non-terminal time-steps $t < T$

$$\rho \Leftarrow \rho - \eta \frac{\partial E}{\partial \rho} = \rho + \eta \sum_{t=1}^{m} (z - p_t) \frac{\partial p_t}{\partial \rho} \Rightarrow \quad \Delta\sigma \propto \frac{\partial \log_\sigma p_\sigma(a_t \mid s_t)}{\partial \sigma} z_t$$

- The final model wins 80% of the time when playing against the first SL model

Reinforcement Learning of Value Network



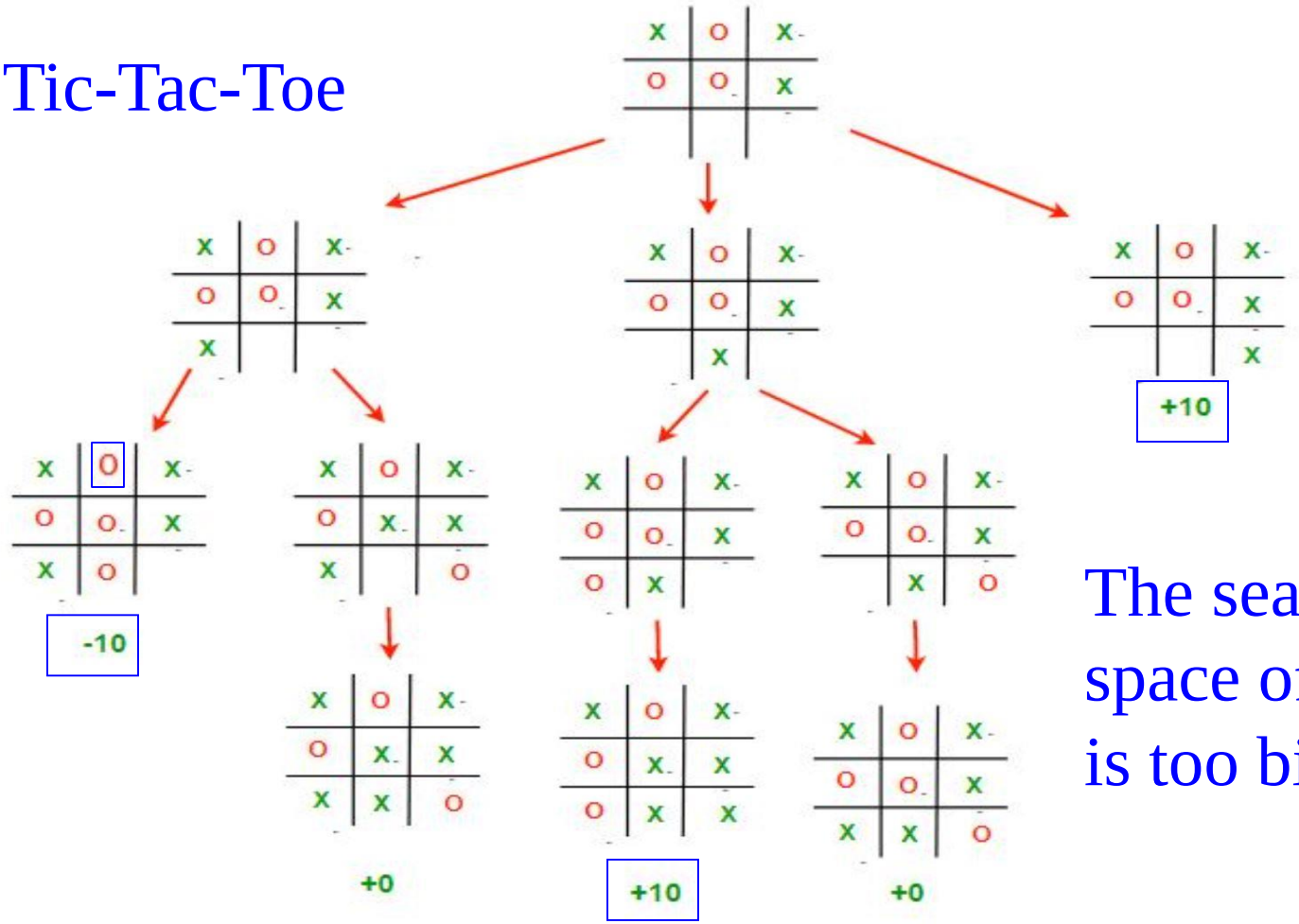
- Value network: 12-layer convolutional neural network
- Training data: 30-million games of self-play (p_ρ).
- Training algorithm: minimize MSE by stochastic gradient descent (z_t=*1* or *-1*), one random "s" per game

$$\Delta\theta \propto -\eta \frac{\partial E}{\partial \theta}, \text{ where } \mathrm{E} = \frac{1}{2}(\mathrm{z_t} - \mathrm{v}_\theta(s_t))^2 \Rightarrow \boxed{\Delta\theta \propto \frac{\partial v_\theta(s_t)}{\partial \theta}(z_t - v_\theta(s_t))}$$

- 1 week training on 50 GPUs using Google Cloud
- Results: First strong position evaluation function - previously thought impossible

Search for Best Move: MinMax

Tic-Tac-Toe

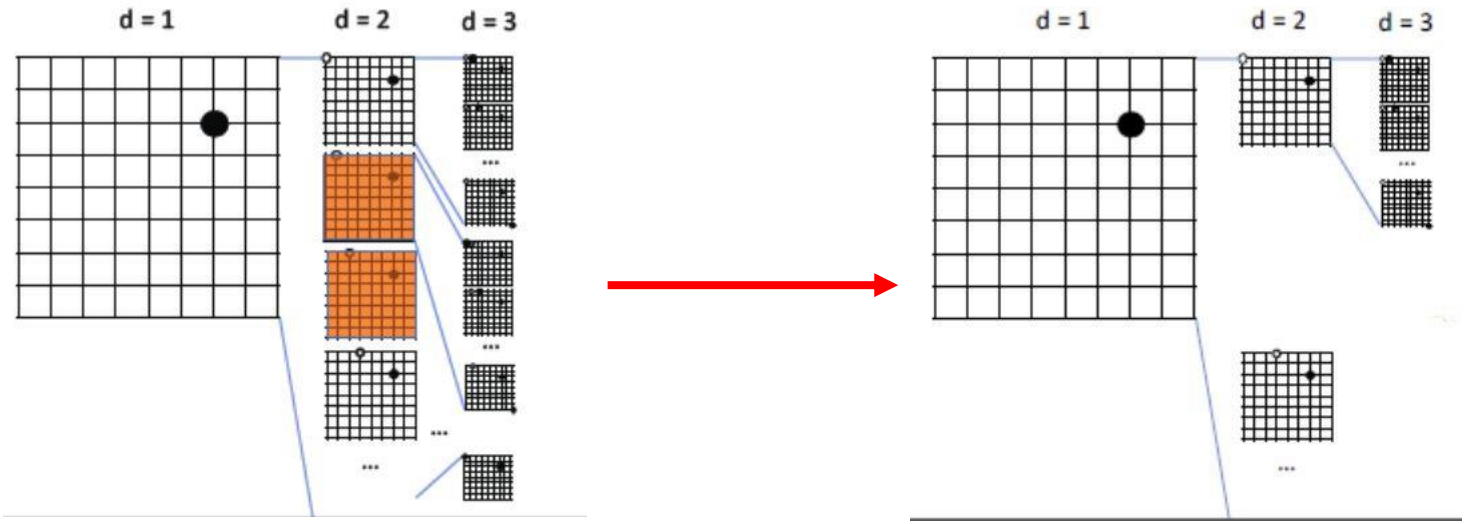


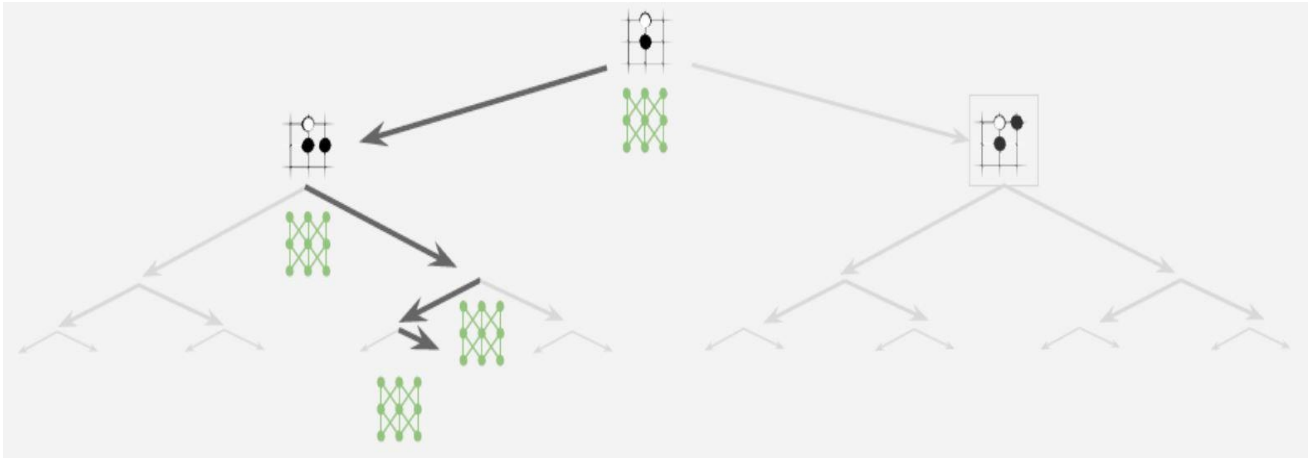
The search space of Go is too big

Monte Carlo Tree Search

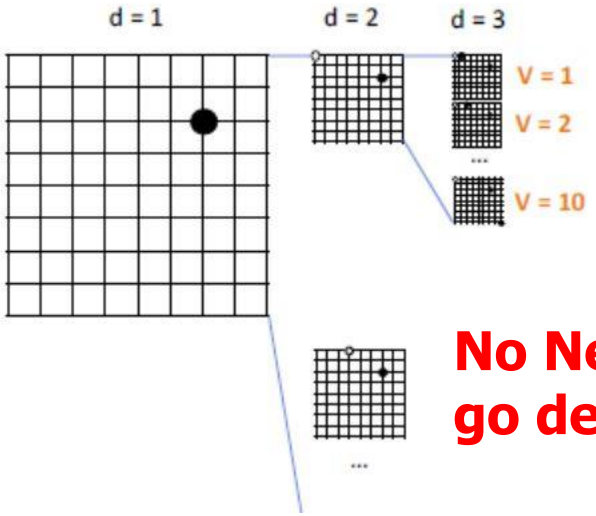
- Simulate games starting from the current board position.
- Search for moves and play simulated games until the end.
  - If enough games are simulated, will get an idea which move likely gets the most wins.
  - A search tree recording sequences (how many wins for each move) of simulated moves is built
- To avoid the huge search space, need to prioritize searches based on information provided by the policy and the value networks.
- Should explore moves that we know little in the simulations so far (exploration), on the other hand, if we win a simulated game, try the corresponding moves more often (exploitation).

Policy Network for Search Breadth Reduction



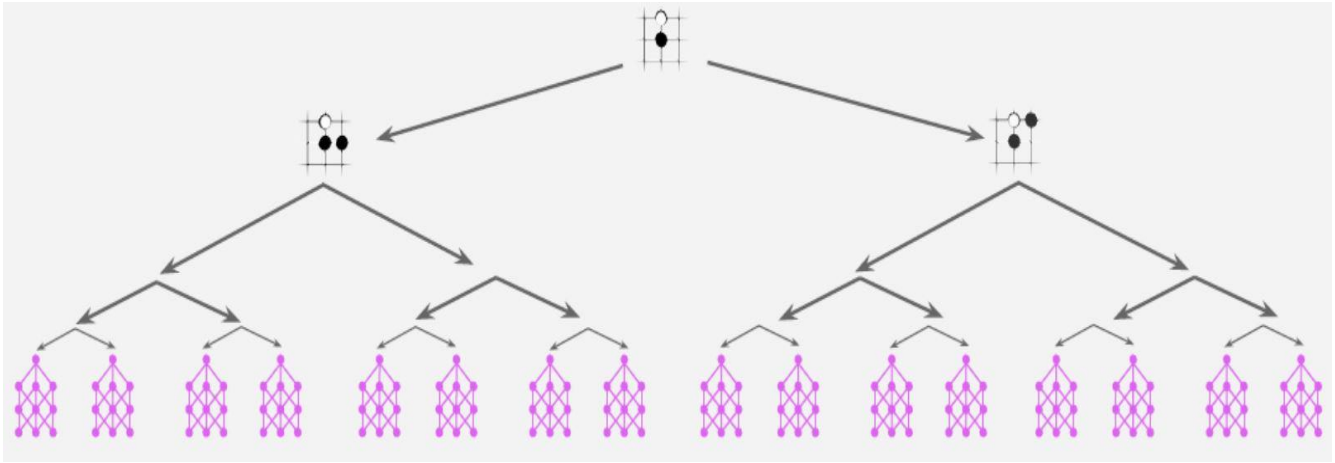


Value Network for Search Depth Reduction



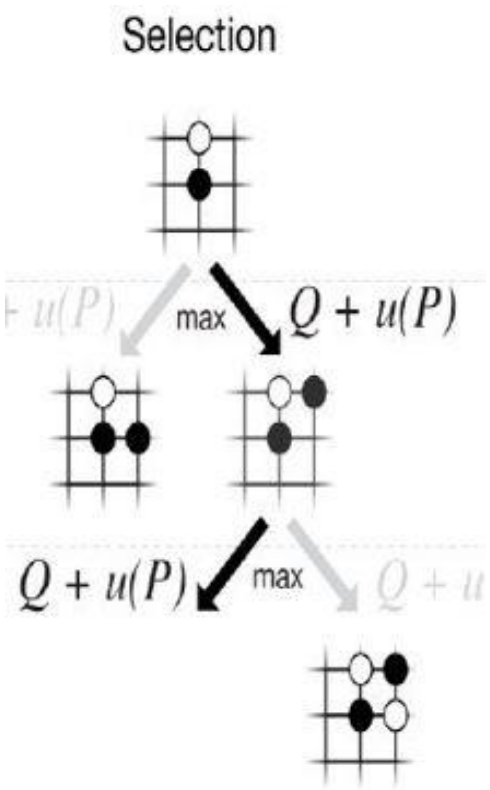
No Need to go deeper

RollOut: when we reach the leaf node, we randomly choose an action at each step and simulate this action to receive an average reward when the game is over.



Selection in MCTS

- ***Q*** is for the exploitation and ***u*** is for the exploration.



$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t, a) + u(s_t, a)\right)$$

$$N(s,a) = \sum_{i=1}^{n} \mathbf{1}(s,a,i)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbf{1}(s,a,i) V(s_L^i)$$

$$V(s_L) = (1-\lambda) v_\theta(s_L) + \lambda z_L$$

$$u(s,a) \propto \frac{P(s,a)}{1+N(s,a)}$$

$$P(s,a) = p_\sigma(a|s)$$

$p_\sigma(a|s)$ - From the policy network: how good to take action a.

$v_\theta(s_L)$ - From the value network: how good to be in positions s_L.

$N(s,a)$ - How many times have we select action a so far.

z_L - the previous simulated game result.