Introduction to data science

Patrick Shafto

Department of Math and Computer Science

Plan for today

- Tensorflow!
- Intro to deep learning (CNNs, Deep reinforcement learning)

Presentations

- Start Weds!
- 5 minutes, plus 2 for questions
- If you do not see your name on the list, please contact me ASAP

Rate this class!

- Students can still respond until 11:59 pm (EST) on Friday, December 15 at
- https://sakai.rutgers.edu/portal/site/sirs

Install

- In terminal, using virtual environments
  - conda create --name tensorflow
  - source activate tensor flow
  - conda install -c conda-forge tensorflow

MNIST: Image recognition

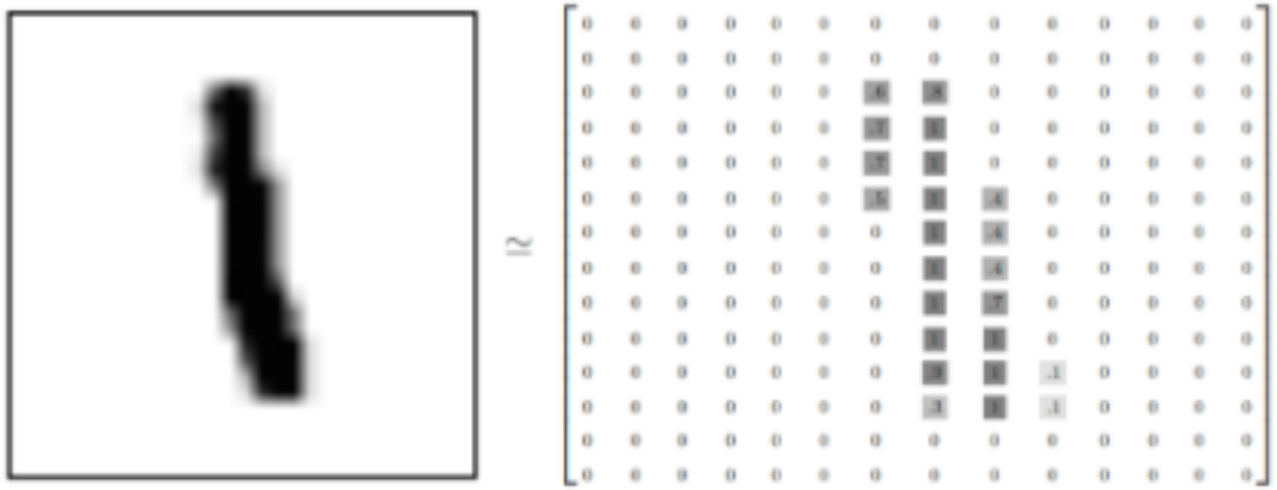
MNIST is a simple computer vision dataset. It consists of images of handwritten digits like these:

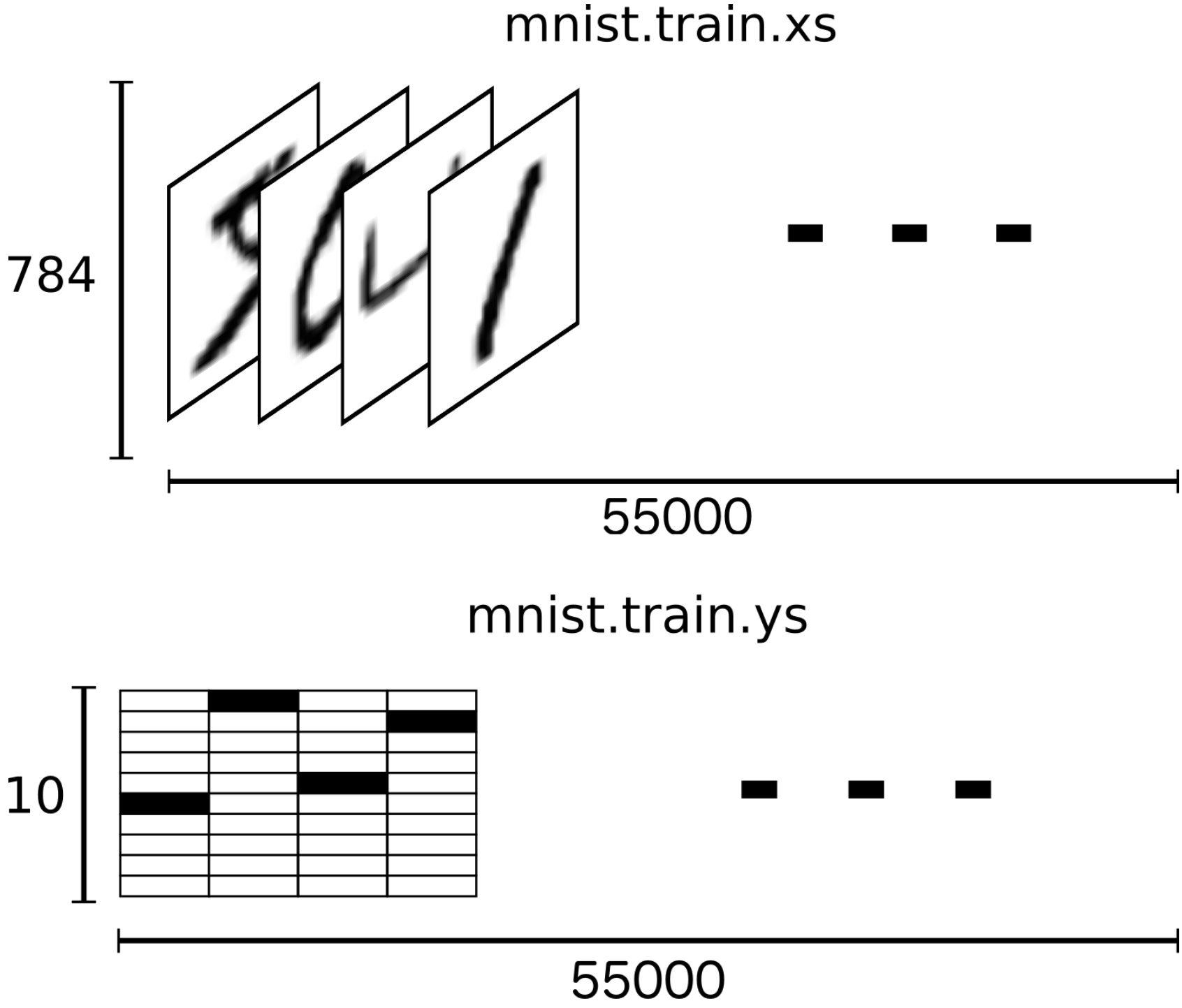
It also includes labels for each image, telling us which digit it is. For example, the labels for the above images are 5, 0, 4, and 1.

mnist_softmax.py code on tensor flow website

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

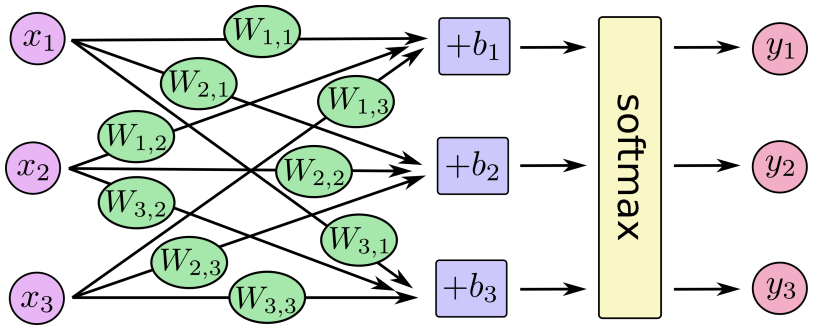
Each image is 28 pixels by 28 pixels. We can interpret this as a big array of numbers:




mnist.train.xs
784
55000
mnist.train.ys
10
55000

Softmax regression (multinomial regression)

$$\text{evidence}_i = \sum_j W_{i,\,j} x_j + b_i \qquad \text{softmax}(evidence)_i = \frac{\exp(evidence_i)}{\sum_j \exp(evidence_j)}$$



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$y = \text{softmax}(Wx + b)$$

Convolutional neural nets (CNN)

- Convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology.
- time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals,
- image data,which can be thought of as a 2-D grid of pixels.

Goodfellow, Bengio, Courville, 2016

Convolutional neural nets (CNN)

- The name indicates that the network employs a mathematical operation called convolution.
- Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers

Goodfellow, Bengio, Courville, 2016

What is a convolution?

- Convolution is an operation on two functions of a real-valued argument.

- Suppose we are tracking the location of a spaceship with a laser sensor. Our laser sensor provides a single output x(t), the position of the spaceship at time t. Both x and t are real valued, that is, we can get a different reading from the laser sensor at any instant in time

Goodfellow, Bengio, Courville, 2016

What is a convolution?

- Now suppose that our laser sensor is somewhat noisy.
- To obtain a less noisy estimate of the spaceship's position, we would like to average several measurements.
- We can do this with a weighting function w(a). If we apply such a weighted average operation at every moment, we obtain a new function s providing a smoothed estimate of the position of the spaceship:

Goodfellow, Bengio, Courville, 2016

What is a convolution?

$$s(t) = \int x(a)w(t-a)da. \tag{9.1}$$

This operation is called **convolution**. The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t). \tag{9.2}$$

Goodfellow, Bengio, Courville, 2016

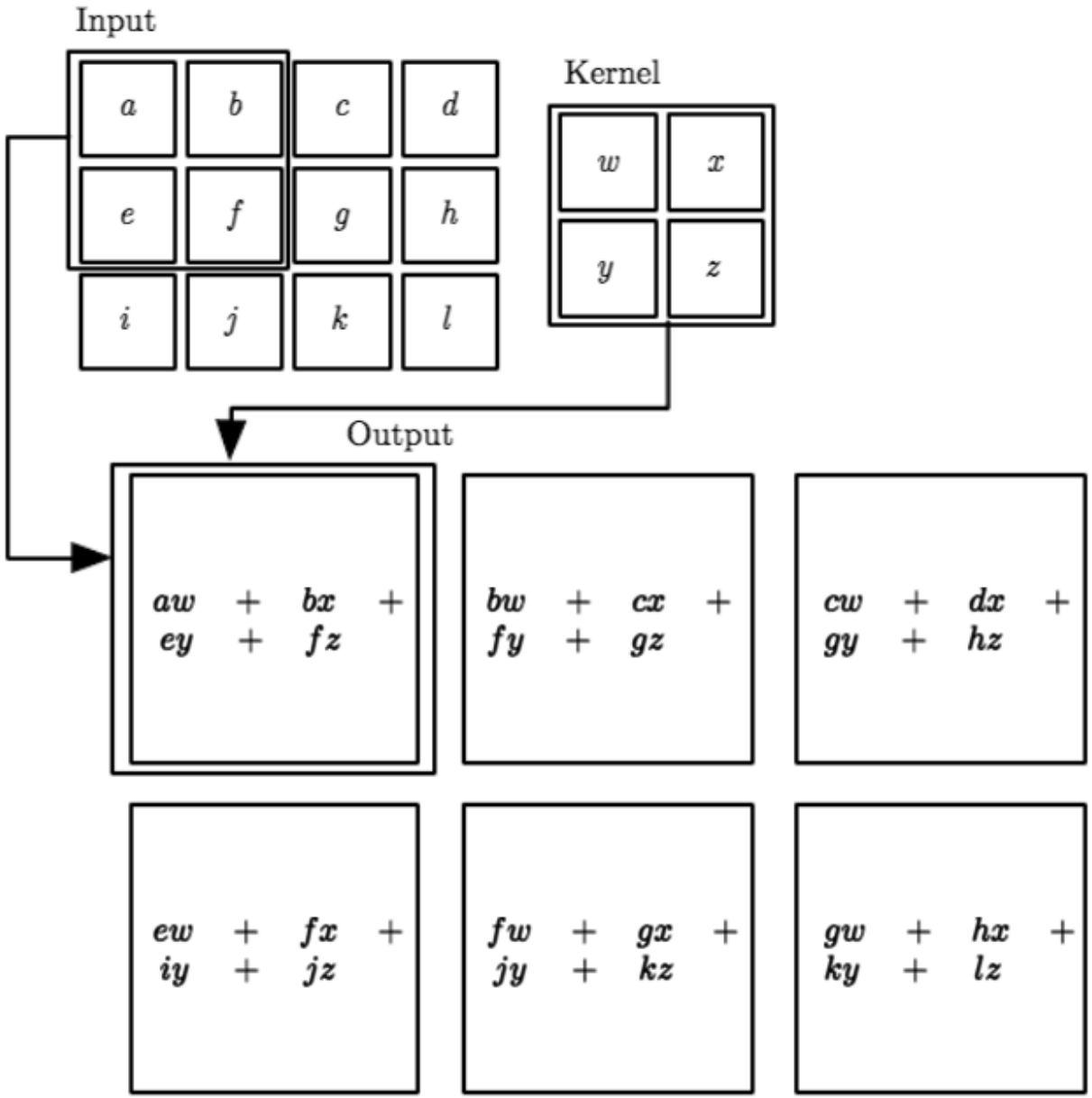


Figure 9.1: An example of 2-D convolution without kernel flipping. We restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

Pooling

- A typical layer of a convolutional network consists of three stages
  - In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations.
  - In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage.
  - In the third stage, we use a pooling function to modify the output of the layer further

Goodfellow, Bengio, Courville, 2016

Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
    - For example, the max pooling (Zhou and Chellappa, 1988) operation reports the maximum output within a rectangular neighborhood
- Pooling helps to make the representation approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.
- Invariance to local translation can be a useful property if we care more about whether some feature is present than exactly where it is

Goodfellow, Bengio, Courville, 2016

Convolutional neural nets (CNN)

- https://www.tensorflow.org/tutorials/deep_cnn
- CIFAR-10 classification is a common benchmark problem in machine learning. The problem is to classify RGB 32x32 pixel images across 10 categories:
  - airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

Convolutional neural nets (CNN)

- https://www.tensorflow.org/tutorials/deep_cnn
- CIFAR-10 classification is a common benchmark problem in machine learning. The problem is to classify RGB 32x32 pixel images across 10 categories:
  - airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
- The model is a multi-layer architecture consisting of alternating convolutions and nonlinearities. These layers are followed by fully connected layers leading into a softmax classifier.
- This model achieves a
  - peak performance of about 86% accuracy
  - within a few hours of training time on a GPU
  - 1,068,298 learnable parameters
  - requires about 19.5M multiply-add operations to compute inference on a single image.

Convolutional neural nets (CNN)

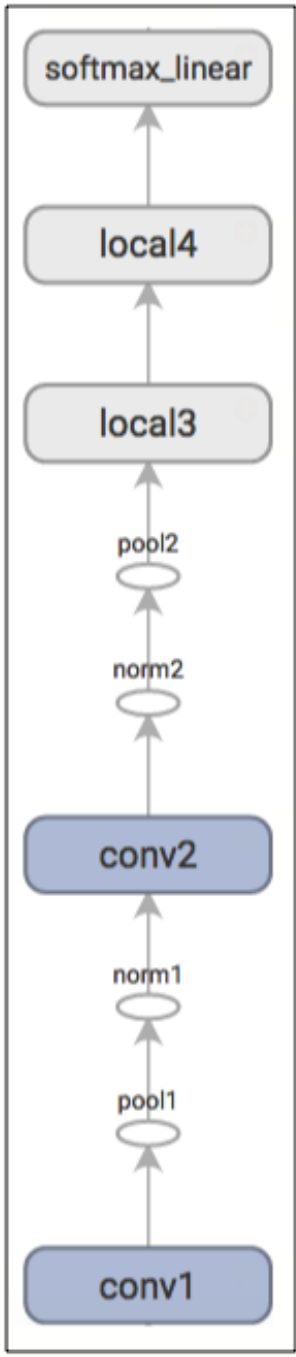
- The CIFAR-10 tutorial demonstrates several important constructs for designing larger and more sophisticated models in TensorFlow:

- Core mathematical components including convolution (wiki), rectified linear activations (wiki), max pooling (wiki) and local response normalization (Chapter 3.3 in AlexNet paper).

- Visualization of network activities during training, including input images, losses and distributions of activations and gradients.

Convolutional neural nets

Layer Name	Description
conv1	convolution and rectified linear activation.
pool1	max pooling.
norm1	local response normalization.
conv2	convolution and rectified linear activation.
norm2	local response normalization.
pool2	max pooling.
local3	fully connected layer with rectified linear activation.
local4	fully connected layer with rectified linear activation.
softmax_linear	linear transformation to produce logits.



Deep reinforcement learning

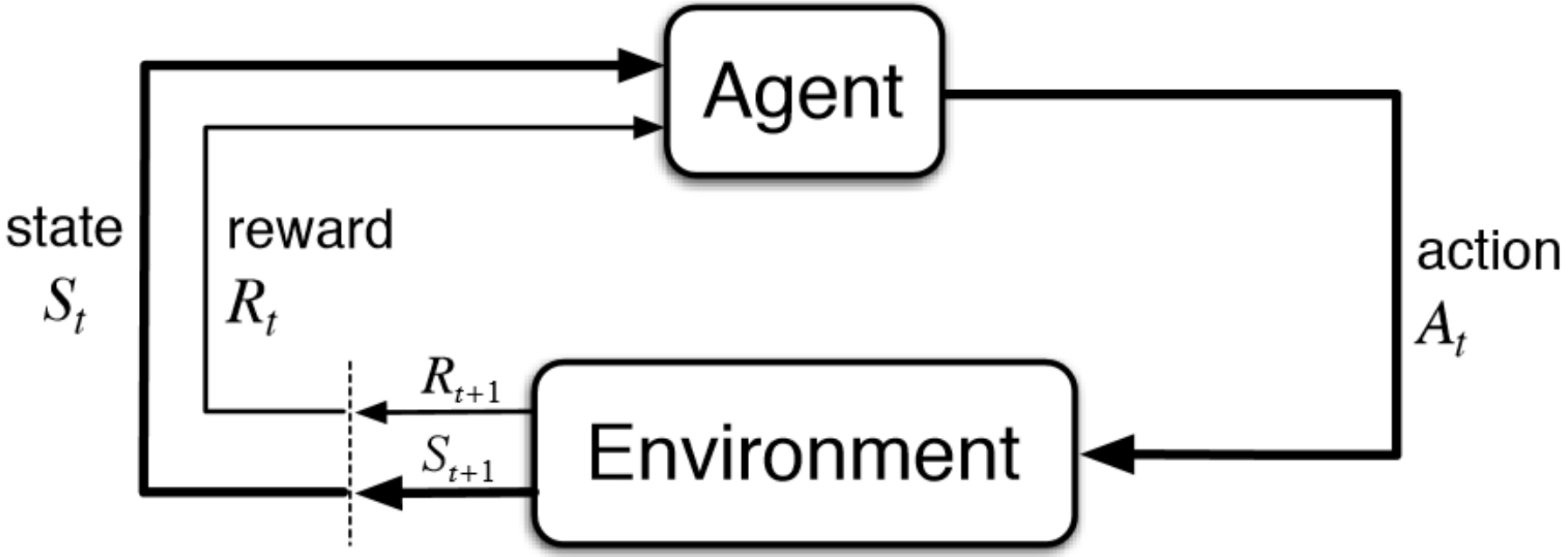
ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

Reinforcement learning



Goal is to infer a policy:
a mapping from states to actions that maximize rewards

Deep reinforcement learning

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses ‘value networks’ to evaluate board positions and ‘policy networks’ to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state- of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

- All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players.
- These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length).
- In large games, such as chess ($b \approx 35$, $d \approx 80$)[1] and especially Go ($b \approx 250$, $d \approx 150$)[1], exhaustive search is infeasible, but the effective search space can be reduced by two general principles.
- First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess, checkers and othello, but it was believed to be intractable in Go due to the complexity of the game.
- Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s.
- For example, Monte Carlo rollouts search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon and Scrabble, and weak amateur level play in Go

- Monte Carlo tree search (MCTS) uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate.
- The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function.
- The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves. These policies are used to narrow the search to a beam of high-probability actions, and to sample actions during rollouts. This approach has achieved strong amateur play.

We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1).

- We begin by training a supervised learning (SL) policy network p_σ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients.
- We also train a fast policy p_π that can rapidly sample actions during rollouts.
- Next, we train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self- play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy.
- Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program AlphaGo efficiently combines the policy and value networks with MCTS.

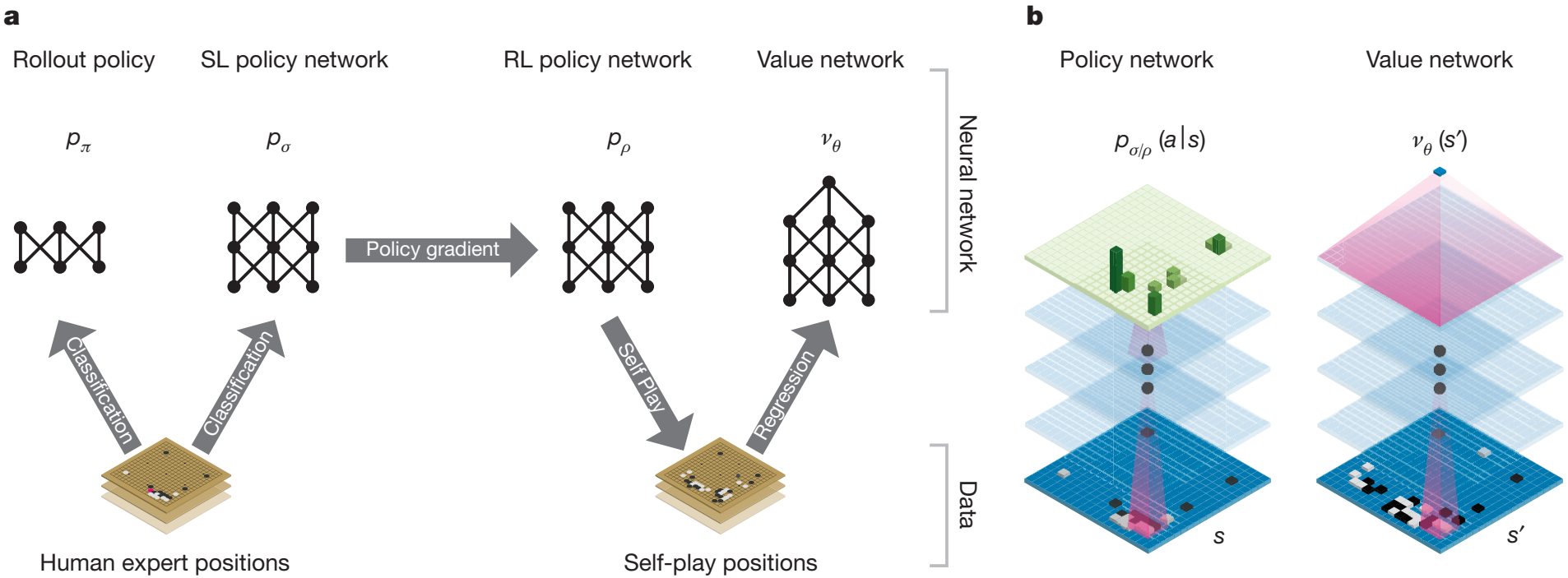


Figure 1 | Neural network training pipeline and architecture. a, A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set. **b**, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

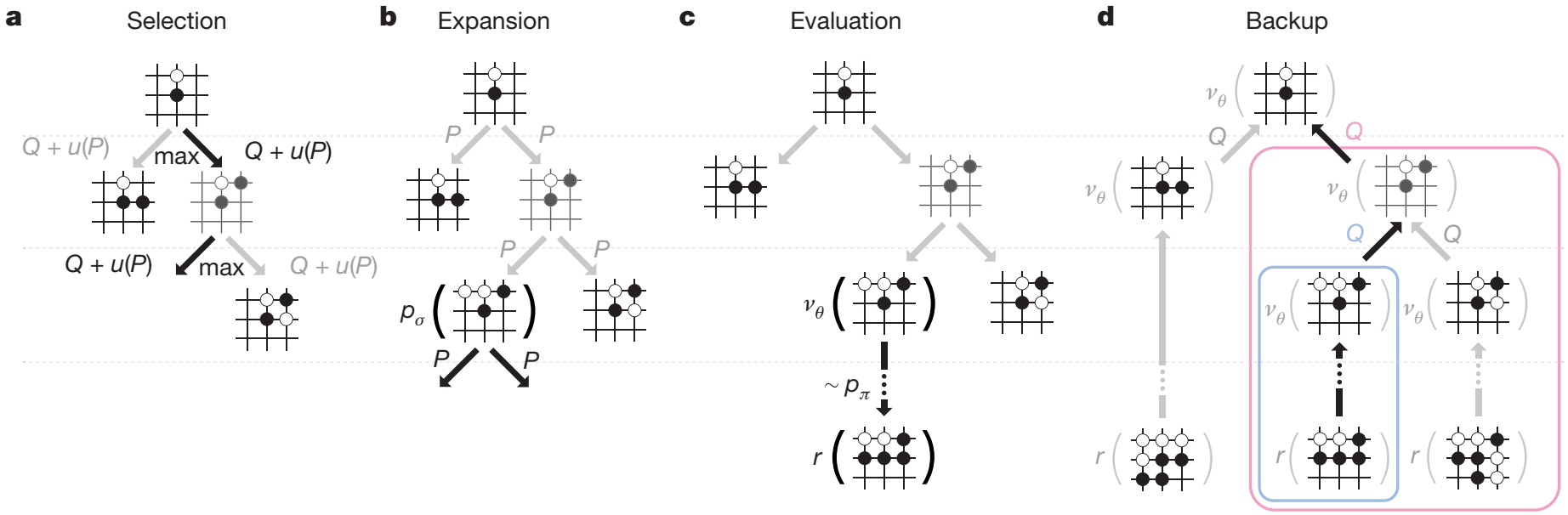


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

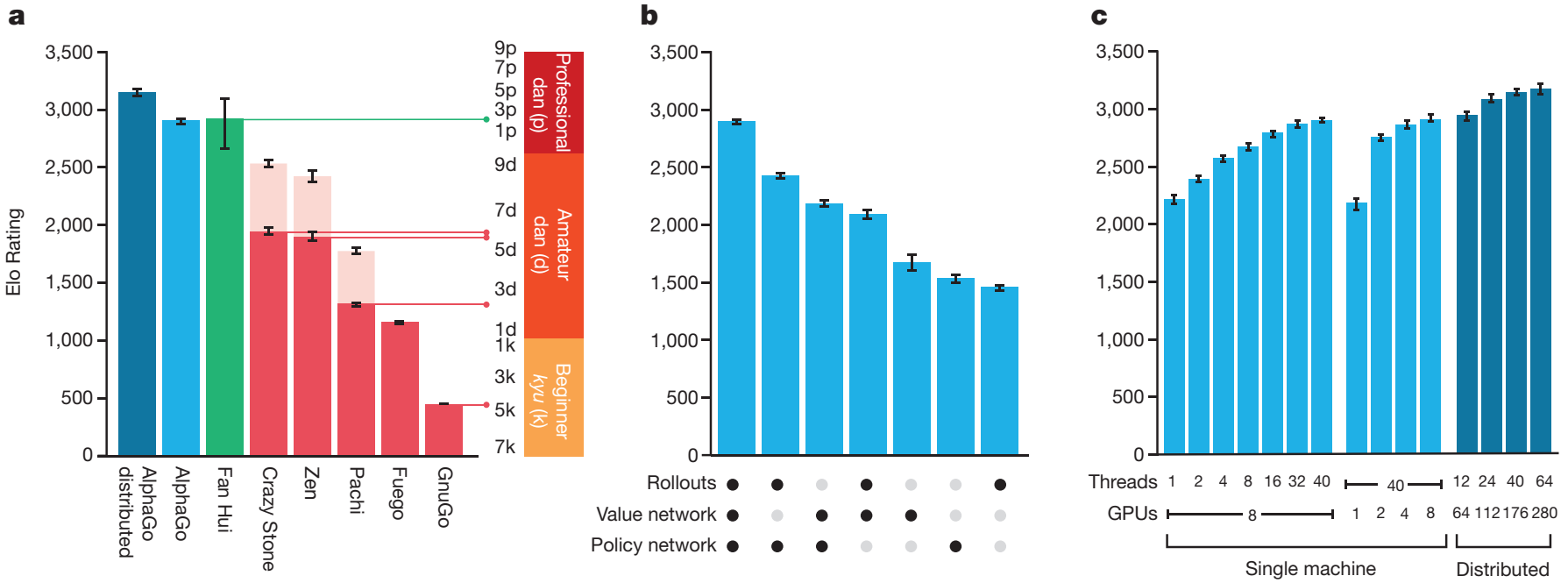


Figure 4 | Tournament evaluation of AlphaGo. a, Results of a tournament between different Go programs (see Extended Data Tables 6–11). Each program used approximately 5 s computation time per move. To provide a greater challenge to AlphaGo, some programs (pale upper bars) were given four handicap stones (that is, free moves at the start of every game) against all opponents. Programs were evaluated on an Elo scale[37]: a 230 point gap corresponds to a 79% probability of winning, which roughly corresponds to one amateur *dan* rank advantage on KGS[38]; an approximate correspondence to human ranks is also shown, horizontal lines show KGS ranks achieved online by that program. Games against the human European champion Fan Hui were also included; these games used longer time controls. 95% confidence intervals are shown. **b**, Performance of AlphaGo, on a single machine, for different combinations of components. The version solely using the policy network does not perform any search. **c**, Scalability study of MCTS in AlphaGo with search threads and GPUs, using asynchronous search (light blue) or distributed search (dark blue), for 2 s per move.

Presentations

- Start Weds!
- 5 minutes, plus 2 for questions