Ian's ideas for research projects

Tea talk Jan 16

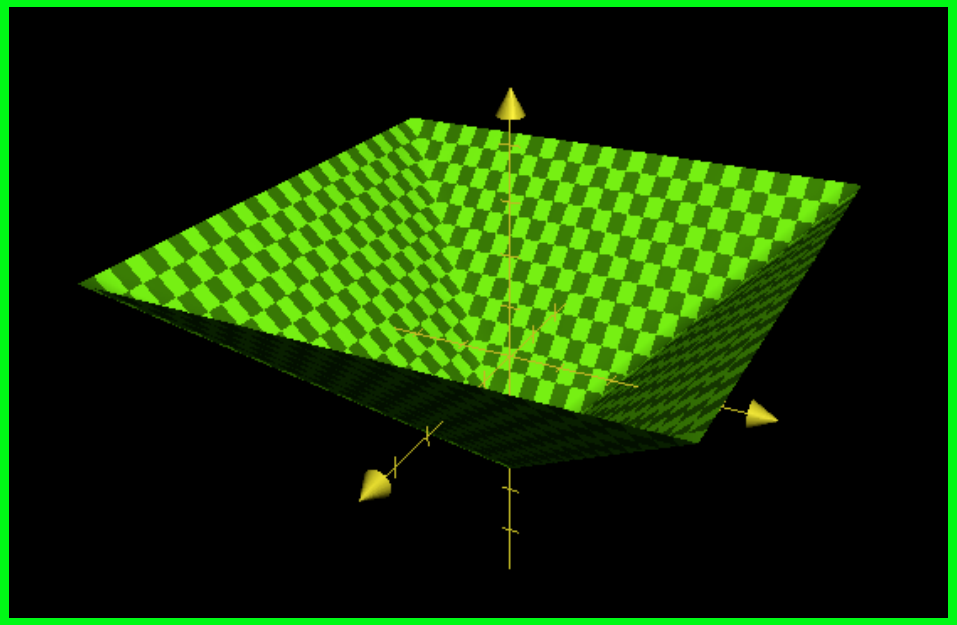
Research ideas

- Restricted maxout units
- Big, sparsely connected nets
- Piece change regularization
- Trajectory optimization
- Cognitive agency

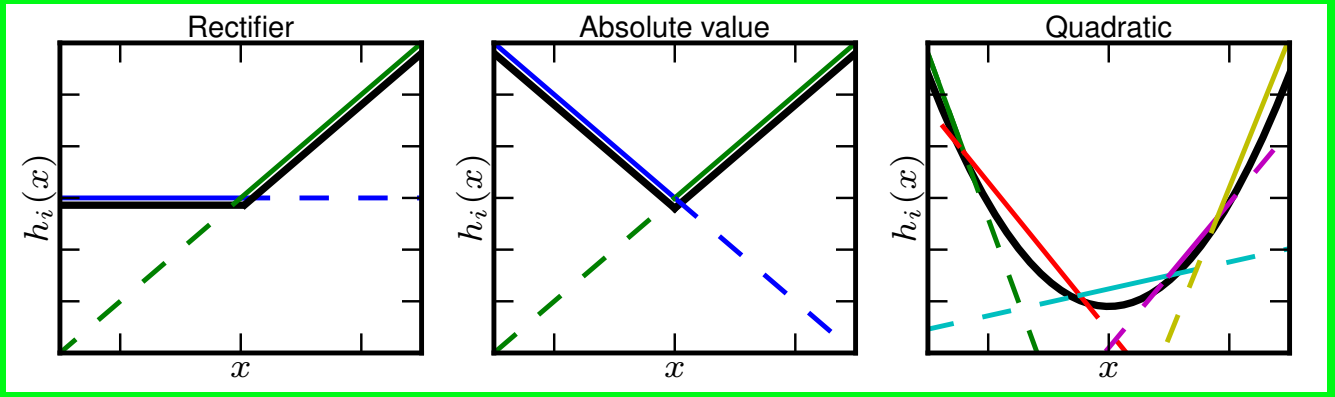
Restricted maxout units

- Meng Cai
  - Maxout works without dropout
  - Best # of filters per unit without dropout is 2

Restricted maxout units



VS



Works on MNIST

- Permutation invariant test error:
  - Maxout + dropout: 0.94%
  - ReMUs + dropout: 0.90%
  - Benefit is from the ReMU itself, not the (direction, slope) parameterization

What's left to do

- Get it working somewhere else
  - I had trouble getting it to work on CIFAR-10, but haven't tried hard yet
- Show that it reduces overfitting (maybe works better without dropout?)
- Show that it reduces memory consumption
- Show that it reduces runtime

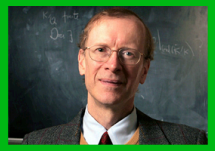
Big, sparsely connected nets

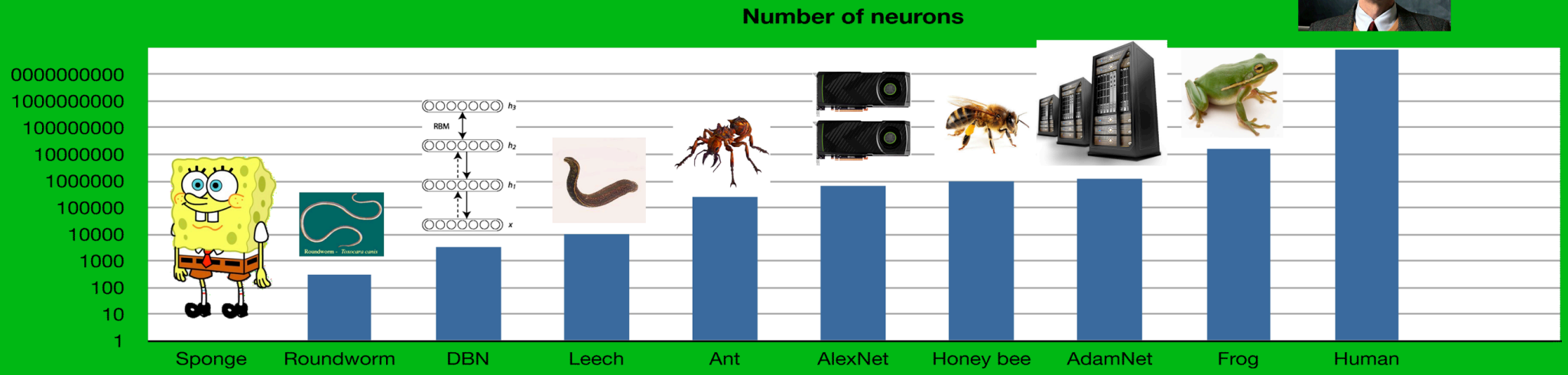
- Neural nets are suddenly big in applications:
  - ImageNet-level object rec
  - Object detection
  - Speech rec
  - House number transcription

What changed?

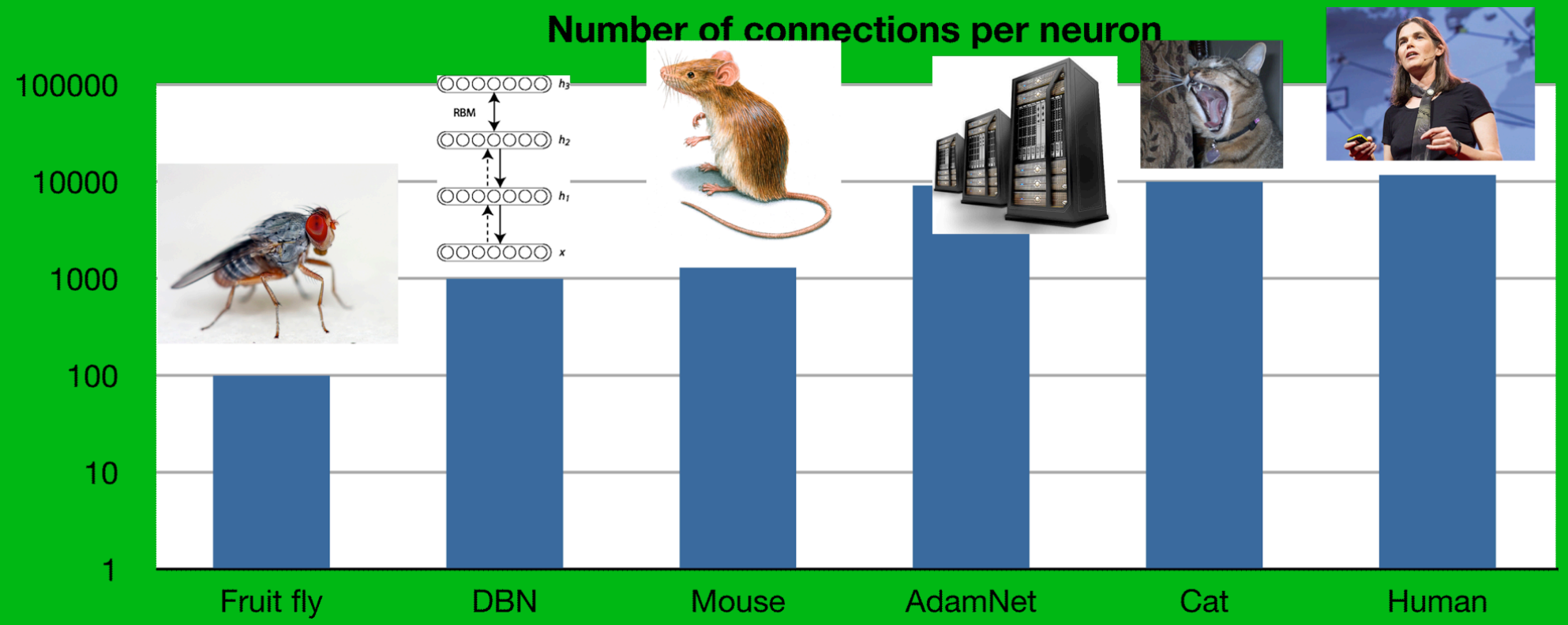
- ~~Greedy layerwise pretraining~~
- ~~Probabilistic semantics~~
- ~~Manifold semantics~~
- ~~Semi-supervised learning~~
- Rectified linear units
- Large datasets
- Large models

Model size




Number of neurons
0000000000
1000000000
100000000
10000000
1000000
100000
10000
1000
100
10
1
RBM
Roundworm - Toxocara canis
Sponge
Roundworm
DBN
Leech
Ant
AlexNet
Honey bee
AdamNet
Frog
Human

Model connectivity



Sparse connectivity for other tasks

- Language modeling (Mehdi)
  - Yoshua pushed this idea years ago; Joseph Turian was too busy winning $1M
- Recurrent nets tend to underfit (Vincent?)
  - Ilya Sutskever recommends working on Schmidhuber's benchmark

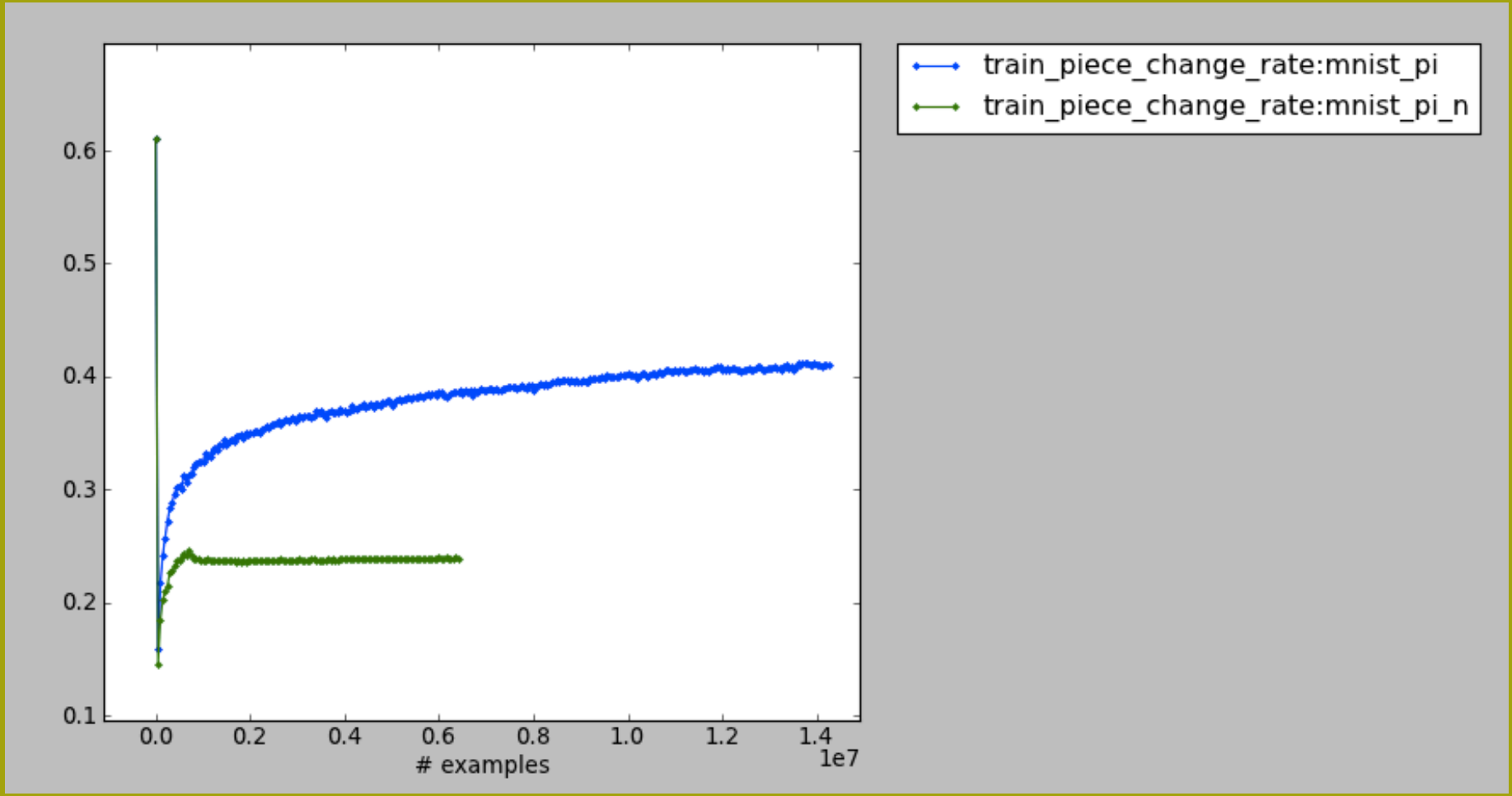
How to do it

- First hidden layer is densely connected
- first_hidden_layer.reshape(square)
- apply cuda_convnet
- Other options:
  - Sparse multiplication on CPU
  - for loop and subtensor
  - CUBLAS block matrix multiply (Razvan)

Piece change regularization

- Dropout model averaging is perfect for ReLUs/maxout if dropping units doesn't move you from one linear piece to another
- Hypothesis that we thought might explain why dropout is better for maxout / ReLUs than curvy activation functions:
  - Maybe dropout training makes you change linear pieces less often

Reality


train_piece_change_rate:mnist_pi
train_piece_change_rate:mnist_pi_n
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e7
examples

Dropout doesn't do what we thought

- So the hypothesis was wrong
- This is an opportunity to improve
- Could design an objective function term to encourage low piece change rate
- Many possible surrogate functions

Trajectory optimization

- Technique from reinforcement learning
- Used in conjunction with variational policy optimization:
  - E-step: figure out best sequence of actions for multiple simulated scenarios
  - M-step: update policy to increase likelihood of these optimal "trajectories"

Trajectory optimization for recurrent nets

- Make the states of the net optimization variables themselves
- Alternate between searching between better sequences of hidden states and better model params
- Mitigates vanishing gradient by facilitating larger search steps
- No explosion problem anymore:
  - Model gradients only propagated 1 step
  - States are bounded

Cognitive agency

- Idea: use reinforcement learning to learn how to think
- *Agents* that can take *cognitive* actions

Possible applications

- Discrete-valued units (Yoshua and others have already explored this a lot)
  - Possibly a strong regularizer
  - "Stickier" memory units in recurrent nets (think digital versus analog)
- Gating (Yoshua already has Mehdi + Nicholas exploring this)
  - Turn off sections of a net to make it cheaper

Possible applications

- Routing
  - Let layer i have incoming weights W
  - Choose which block of layer i-1 is connected to it dynamically
  - The action is a switch that chooses which block to send
  - Olshausen did this with 1-layer models where blocks are spatial locations of image
  - Implements attention; speeds up inference
  - Routing with learned blocks = attention to abstract properties

Possible applications

- Interact with non-neural computation objects
  - Example: stack
    - Cognitive agent learns to send actions PUSH and POP
    - Can store/retrieve states of a specific hidden layer
    - Better memory for recurrent net (e.g., PUSH every time you see '(', POP every time you see ')', predict ')' only if something is on the stack)
    - Combine with attention to have net learn to do structured parsing tasks
  - Could also do tapes, hash maps, etc.

Possible ways of getting there

- Model expected reward of each action
- Select action with highest expected reward
- Learning might need to be based on *causal modeling* (see Judea Pearl's tutorial) rather than *statistical modeling*
  - We are making *intervention queries*
  - We are not inferring *conditional distributions*