Hyper-Parameters for Deep Learning

ICML'2014
AutoML Workshop

Yoshua Bengio

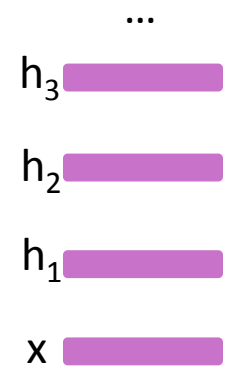
June 26, 2014

Université de Montréal

LISA

Deep Representation Learning

Learn multiple levels of representation of increasing complexity/abstraction

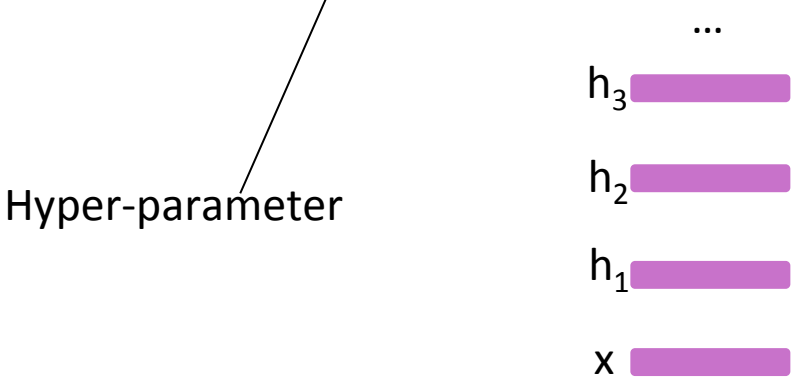


- theory: exponential gain
- brains are deep
- cognition is compositional
- Better mixing (Bengio et al, ICML 2013)

- **They work! SOTA on industrial-scale AI tasks (object recognition, speech recognition, language modeling, music modeling)**

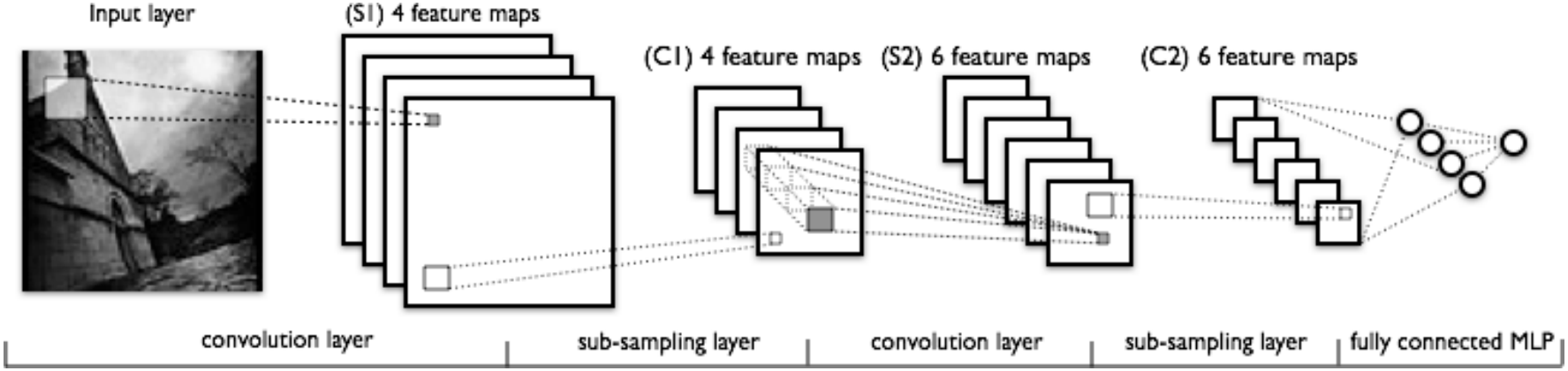
Deep Learning

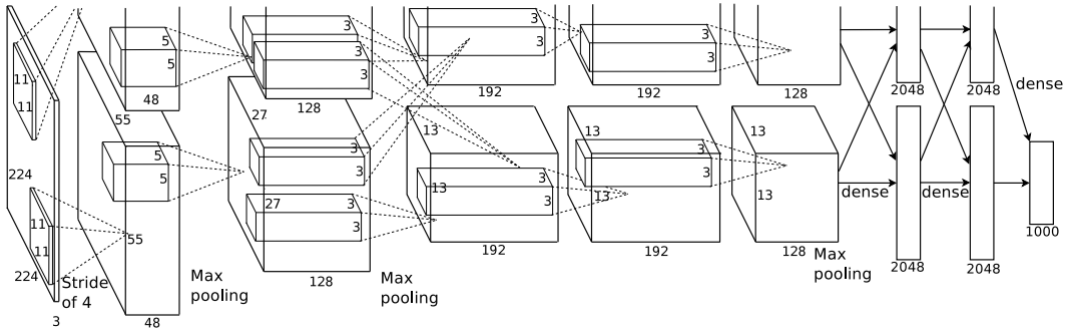
When the number of levels can be ***data-selected****, this is a deep architecture*

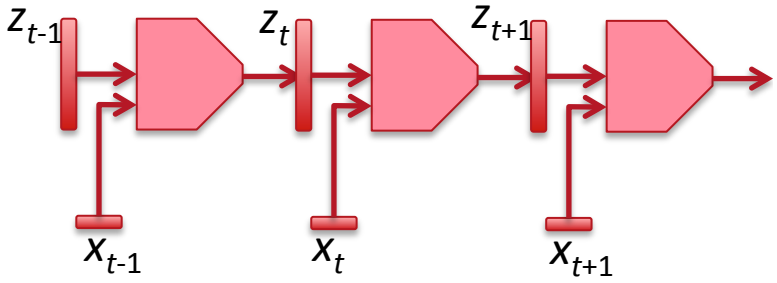


Temporal & Spatial Inputs: Convolutional & Recurrent Nets

- Local connectivity across time/space
- Sharing weights across time/space (translation equivariance)
- Pooling (translation invariance, cross-channel pooling for learned invariances)





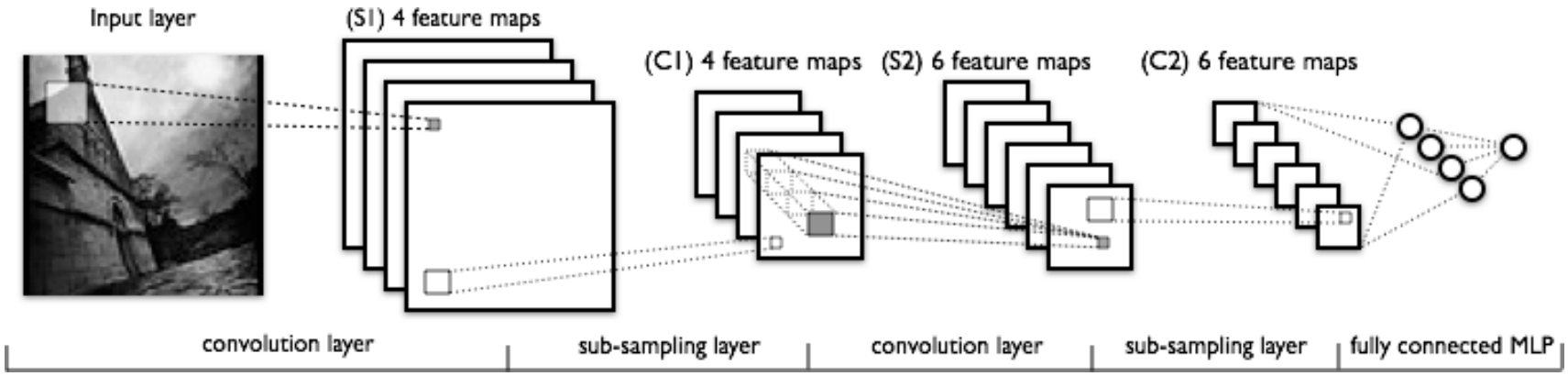


Recurrent nets (RNNs) can summarize information from the past

Bidirectional RNNs also summarize information from the future

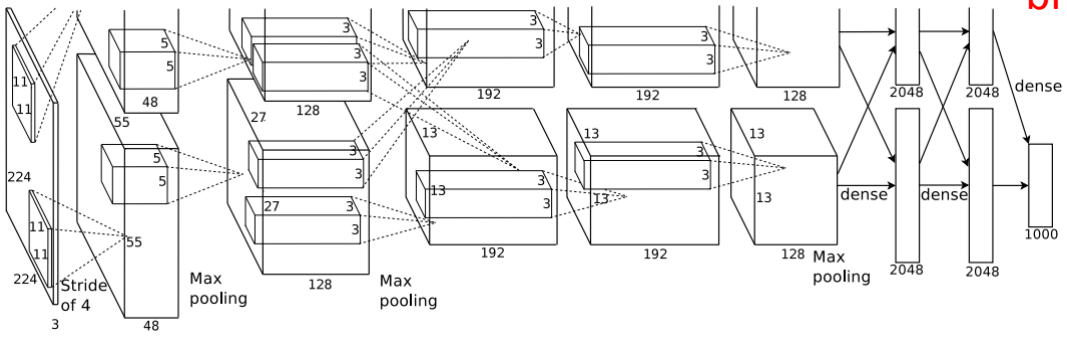
Alternating convolutions & pooling

- Inspired by visual cortex, idea from Fukushima's Neocognitron, combined with back-prop and developped by **LeCun** since 1989



- Increasing number of features, decreasing spatial resolution
- Top layers are fully connected

Krizhevsky, Sutskever & Hinton 2012
breakthrough in object recognition



Understanding the difficulty of training deep feedforward supervised neural networks

(Glorot & Bengio, AISTATS 2010)


Study the activations and gradients

- wrt depth
- as training progresses
- for different initializations → big difference
- for different non-linearities → big difference

First demonstration that deep supervised nets can be successfully trained almost as well as with unsupervised pre-training, by setting up the optimization problem appropriately…

Training RBMs

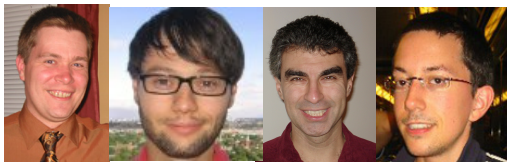
Contrastive Divergence: (CD-k)	start negative Gibbs chain at observed x, run k Gibbs steps
SML/Persistent CD: (PCD)	run negative Gibbs chain in background while weights slowly change
Fast PCD:	two sets of weights, one with a large learning rate only used for negative phase, quickly exploring modes
Herding:	Deterministic near-chaos dynamical system defines both learning and sampling
Tempered MCMC:	use higher temperature to escape modes

Some RBM Variants

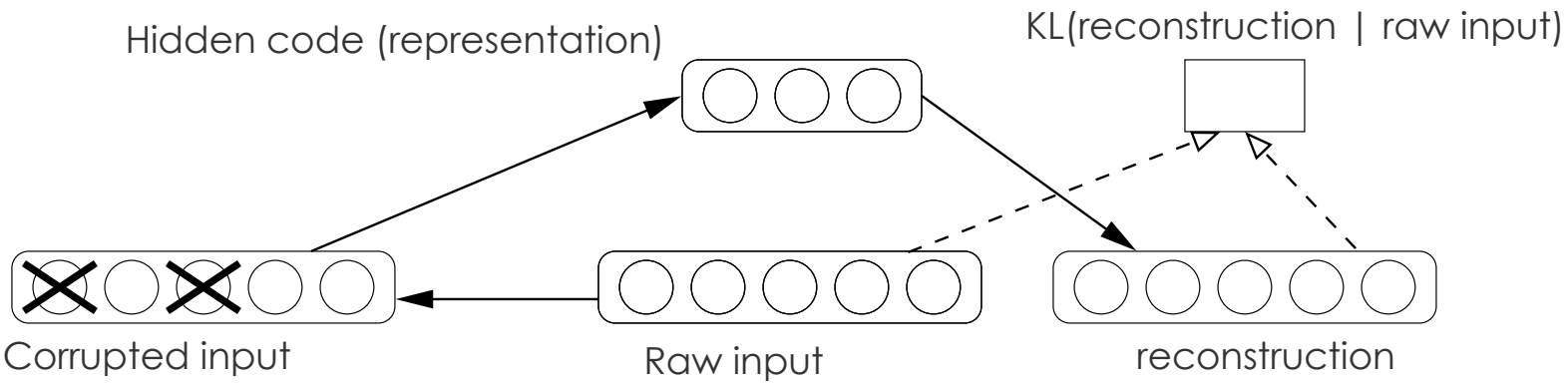
- Different energy functions and allowed values for the hidden and visible units:
  - Hinton et al 2006: binary-binary RBMs
  - Welling NIPS'2004: exponential family units
  - Ranzato & Hinton CVPR'2010: Gaussian RBM weaknesses (no conditional covariance), propose mcRBM
  - Ranzato et al NIPS'2010: mPoT, similar energy function
  - Courville et al ICML'2011: spike-and-slab RBM

Denoising Auto-Encoder

(Vincent et al 2008)



- Corrupt the input during training only
- Train to reconstruct the uncorrupted input



- Encoder & decoder: any parametrization
- As good or better than RBMs for unsupervised pre-training

Level-Local Learning is Important

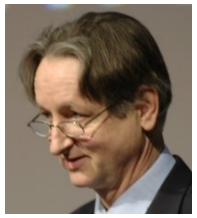
- Initializing each layer of an unsupervised deep Boltzmann machine helps a lot
- Initializing each layer of a supervised neural network as an RBM, auto-encoder, denoising auto-encoder, etc can help a lot
- Helps most the layers further away from the target
- Not just an effect of the unsupervised prior
- Jointly training all the levels of a deep architecture is difficult because of the increased non-linearity / non-smoothness
- Initializing using **a level-local learning algorithm** is a useful trick
- Providing intermediate-level targets can help tremendously (Gulcehre & Bengio ICLR 2013)

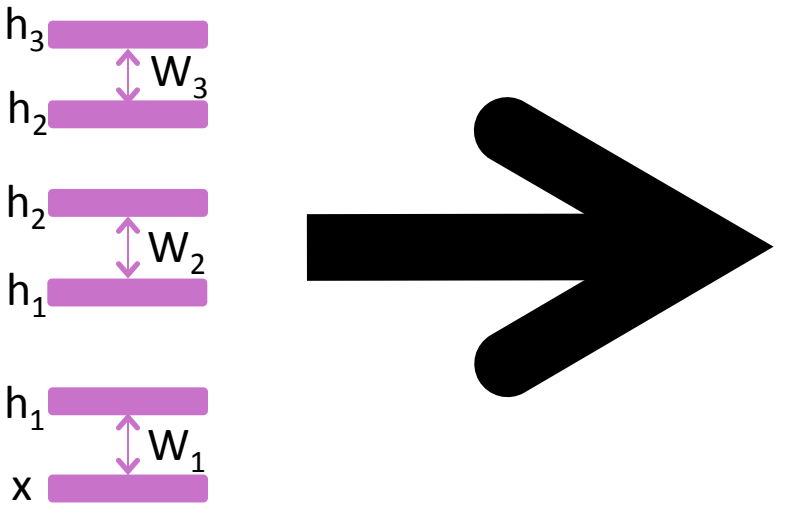
Different hyper-parameters for different stages of learning

- Hyper-parameters for pre-training 1st layer
- Hyper-parameters for pre-training 2nd layer
- …
- Hyper-parameter for supervised fine-tuning

Stack of RBMs / AEs → Deep MLP

- Encoder or P($h|v$) becomes MLP layer

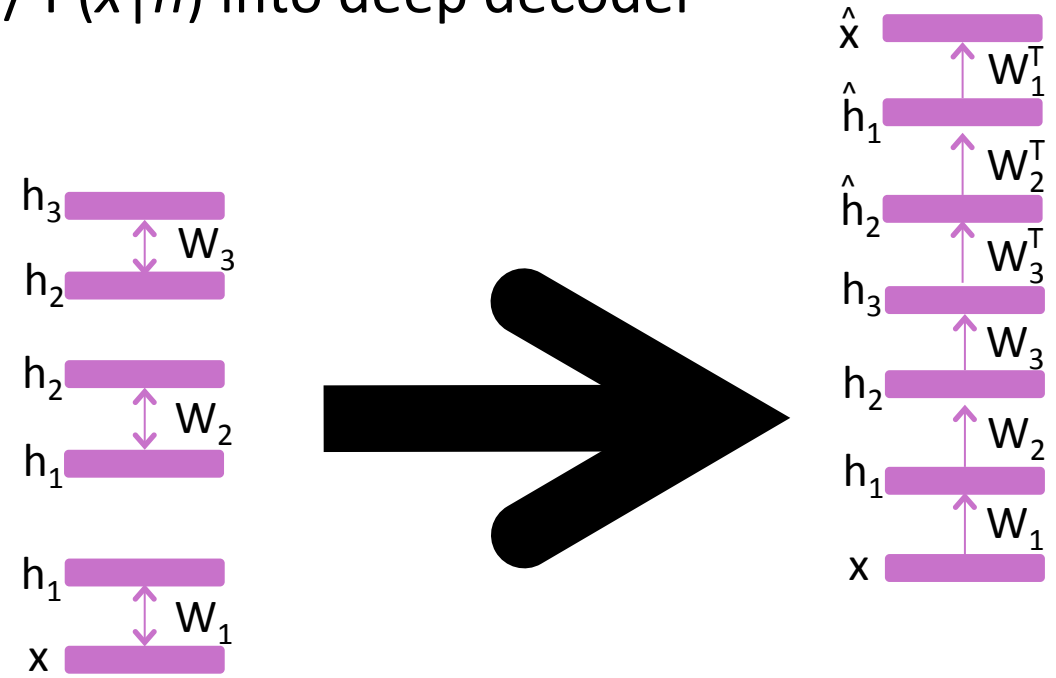




Stack of RBMs / AEs → Deep Auto-Encoder

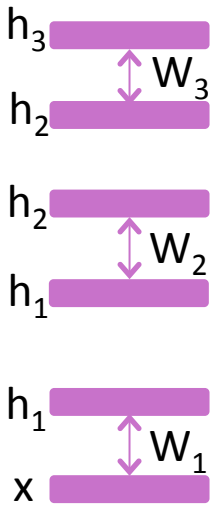
(Hinton & Salakhutdinov 2006)

- Stack encoders / P($h|x$) into deep encoder
- Stack decoders / P($x|h$) into deep decoder

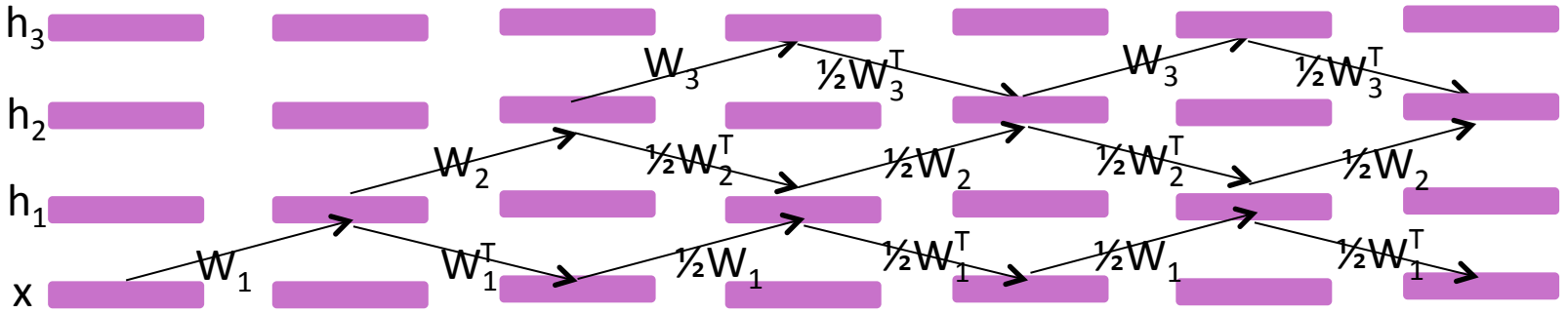


Stack of RBMs / AEs → Deep Recurrent Autoencoder, GSN

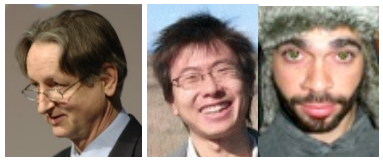
(Savard 2011)  (Bengio & Laufer, arxiv 2013)  



- Each hidden layer receives input from below and above
- Deterministic (mean-field) recurrent computation (Savard 2011)
- Stochastic (injecting noise) recurrent computation: Deep Generative Stochastic Networks (GSNs)
  (Bengio & Laufer arxiv 2013)

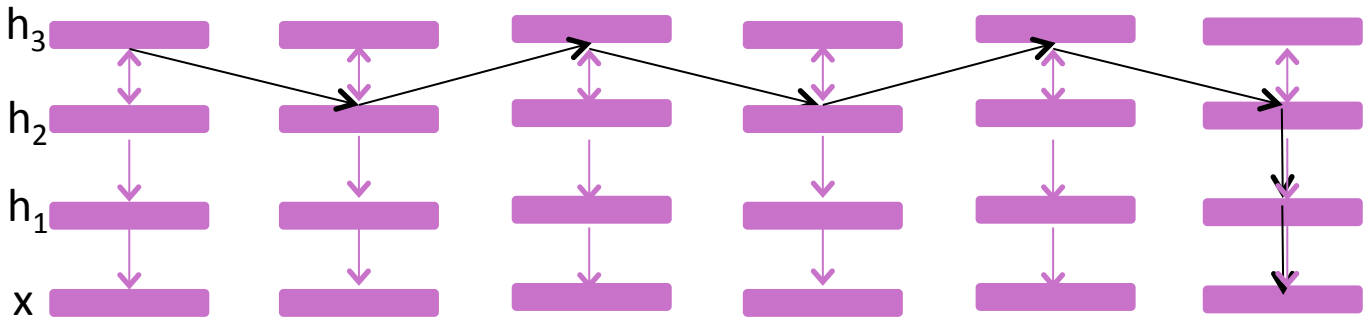


Stack of RBMs → Deep Belief Net



(Hinton et al 2006)

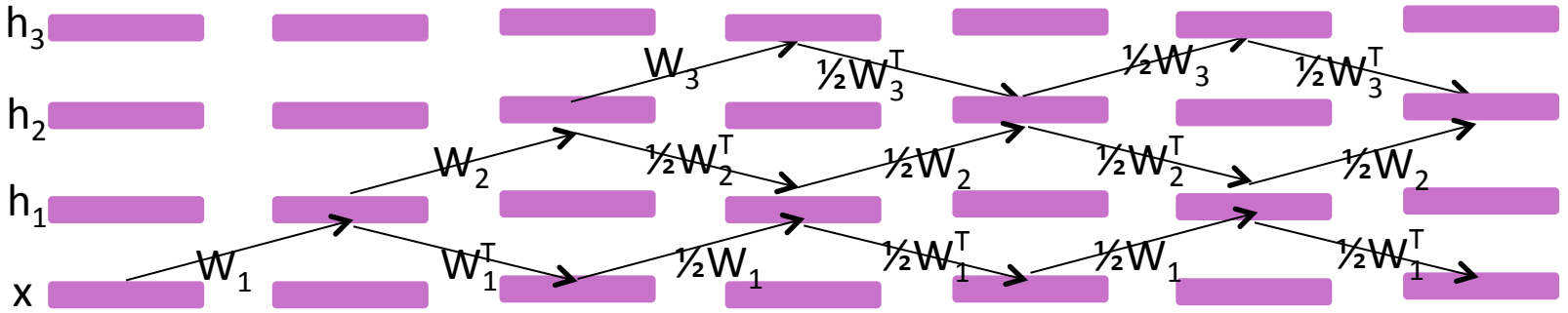
- Stack lower levels RBMs' P($x|h$) along with top-level RBM
- $P(x, h_1, h_2, h_3) = P(h_2, h_3)\, P(h_1|h_2)\, P(x \mid h_1)$
- Sample: Gibbs on top RBM, propagate down



Stack of RBMs → Deep Boltzmann Machine

(Salakhutdinov & Hinton AISTATS 2009)

- Halve the RBM weights because each layer now has inputs from below and from above
- Positive phase: (mean-field) variational inference = recurrent AE
- Negative phase: Gibbs sampling (stochastic units)
- train by SML/PCD

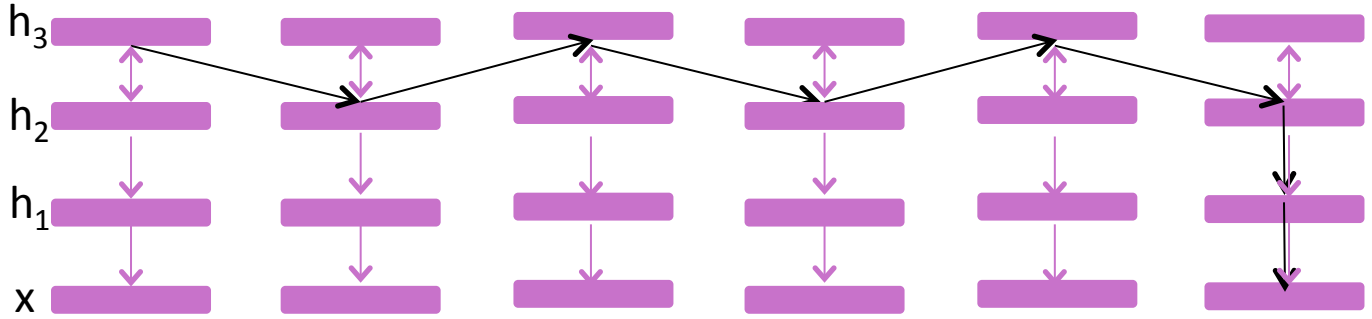


Stack of Auto-Encoders → Deep Generative Auto-Encoder

(Rifai et al ICML 2012)

- MCMC on top-level auto-encoder
  - h_{t+1} = encode(decode(h_t))+σ noise

  where noise is Normal(0, d/dh encode(decode(h_t)))
- Then deterministically propagate down with decoders



Deep Learning Tricks of the Trade

- Y. Bengio (2013), "Practical Recommendations for Gradient-Based Training of Deep Architectures"

  - Unsupervised pre-training
  - Stochastic gradient descent and setting learning rates
  - Main hyper-parameters
    - Learning rate schedule
    - Early stopping
    - Minibatches
    - Parameter initialization
    - Number of hidden units
    - L1 and L2 weight decay
    - Sparsity regularization
  - Debugging
  - How to efficiently search for hyper-parameter configurations

Learning Rates

- Simplest recipe: keep it fixed and use the same for all parameters.
- Collobert scales them by the inverse of square root of the fan-in of each neuron
- Better results can generally be obtained by allowing learning rates to decrease, typically in O(1/t) because of theoretical convergence guarantees, e.g.,

$$\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}$$

with hyper-parameters ε_0 and τ.

- New papers on adaptive learning rates procedures (Schaul 2012, 2013), Adagrad (Duchi et al 2011), ADADELTA (Zeiler 2012)

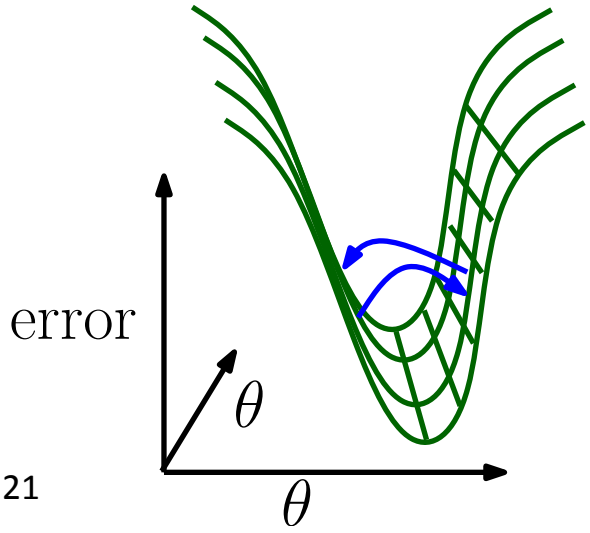
Early Stopping

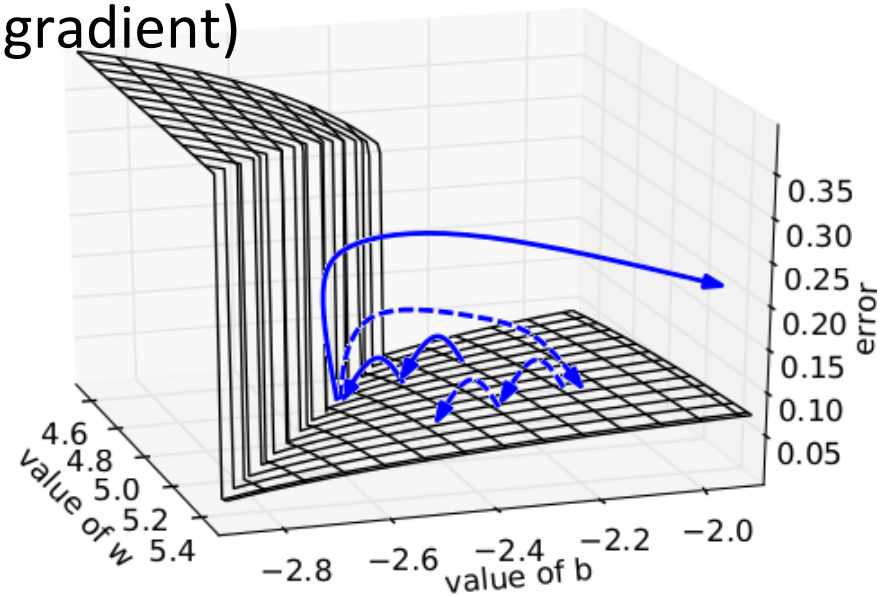
- Beautiful **FREE LUNCH** (no need to launch many different training runs for each value of hyper-parameter for #iterations)
- Monitor validation error during training (after visiting # of training examples = a multiple of validation set size)
- Keep track of parameters with best validation error and report them at the end
- If error does not improve enough (with some patience), stop.

RNN Tricks

(Pascanu, Mikolov, Bengio, ICML 2013; Bengio, Boulanger & Pascanu, ICASSP 2013)

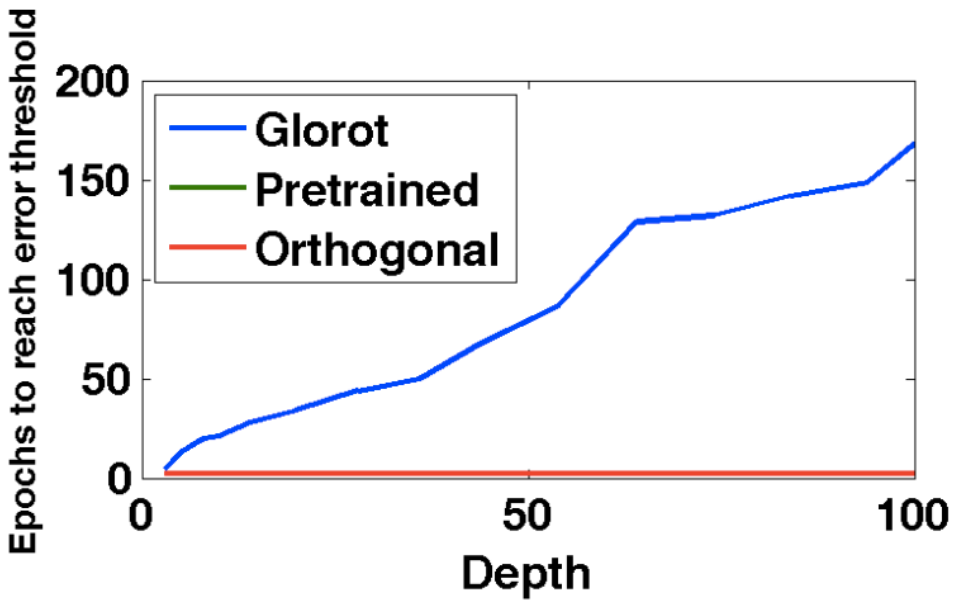
- Clipping gradients (avoid exploding gradients)
- Leaky integration (propagate long-term dependencies)
- Momentum (cheap 2nd order)
- Initialization (start in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid vanishing gradient)
- LSTM self-loops (avoid vanishing gradient)





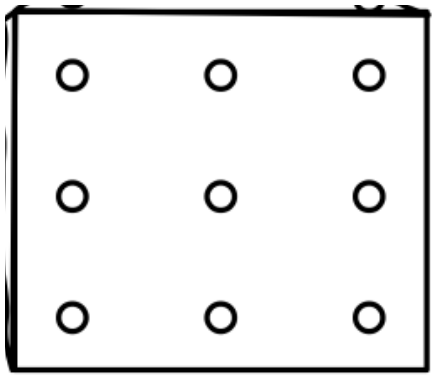
Orthogonal Initialization Works Even Better

- Auto-encoder pre-training tends to yield orthogonal W
- (Saxe, McClelland & Ganguli ICLR 2014) showed that **very deep** nets initialized with random orthogonal weights are much easier to train
- All singular values = 1



Grid Search for Hyper-Parameters

- Discretize hyper-parameter values
- Form cross-product of values across all hyper-parameters: the grid
- Launch a trial training + validation error measurement for each element of the grid
- Can be parallelized on a cluster, but may need to redo failed experiments, until all grid is filled
- Exponential in # of hyper-parameters!



Examples of hyper-parameters in DL

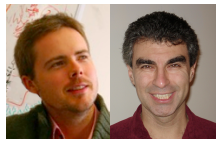
- Initial learning rate
- Learning rate decrease rate
- Number of layers
- Layer size
- Non-linear activation function
- Output non-linearity
- Output cost function
- Minibatch size
- Skip connections
- Dropout probability
- L1 regularizer, L2 regularizer
- Max weight vector norm
- Pre-training algorithm

Other hyper-parameters:

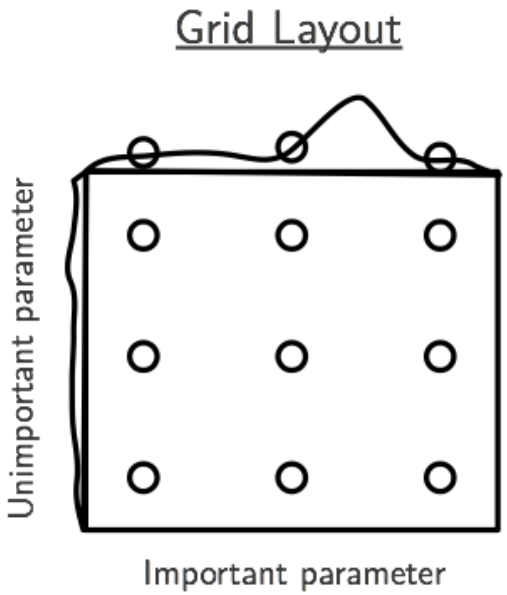
- Pre-training hyper-parameters
- Momentum
- Gradient clipping norm
- Early stopping patience
- Input normalization
- Input dimensionality reduction
- Convolution kernels widths
- Convolutions stride
- Pooling windows sizes
- Pooling strides
- Number of shared layers in multi-task settings
- Output layer regularizer
- Embeddings dimension

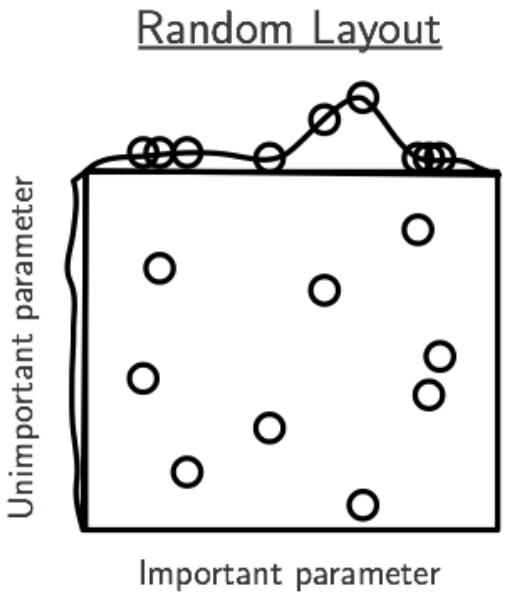
Random Sampling of Hyperparameters

(Bergstra & Bengio 2012)



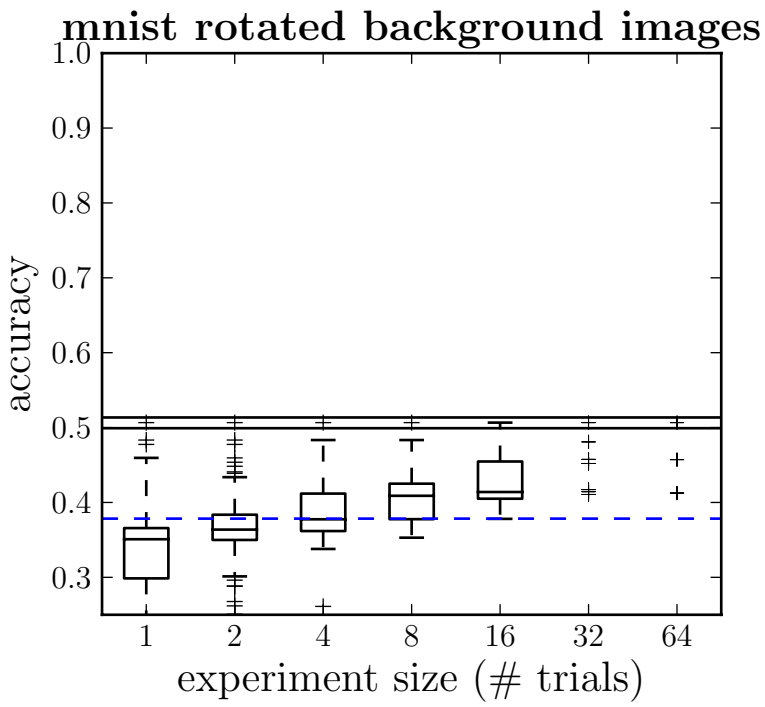
- Random search: simple & efficient
  - Independently sample each HP, e.g. l.rate~exp(U[log(.1),log(.0001)])
  - Each training trial is iid
  - If a HP is irrelevant grid search is wasteful
  - More convenient: ok to early-stop, continue further, etc.

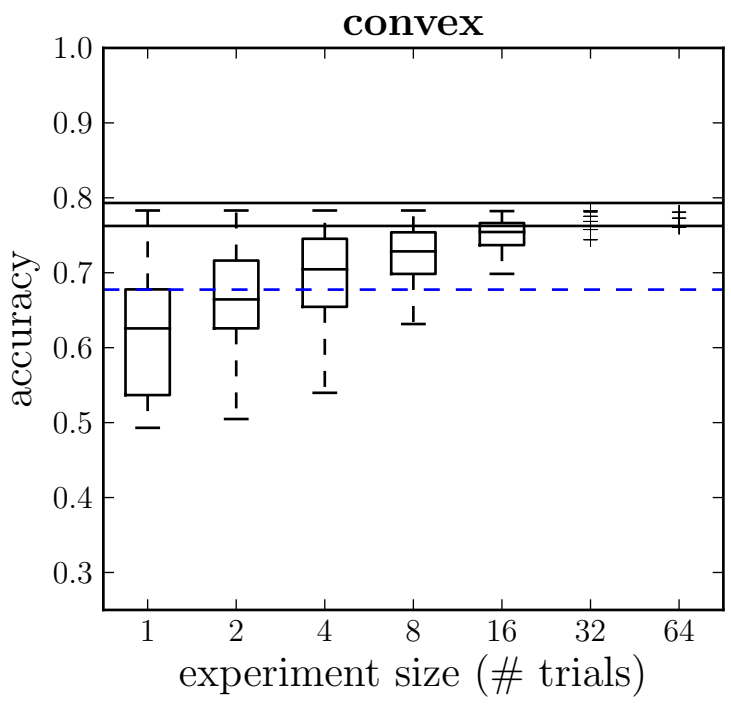




Random Search Learning Curves

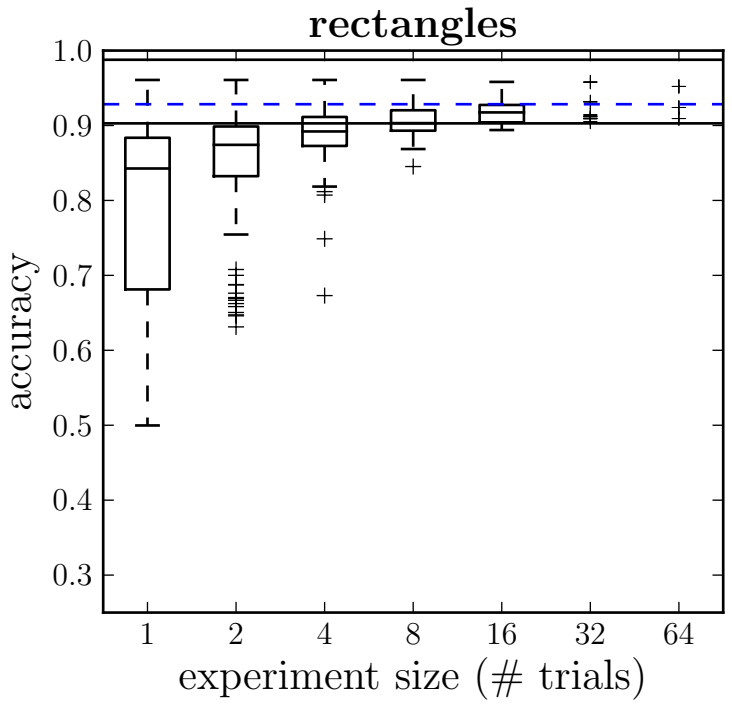
- Blue dotted line = grid search with 100 trials

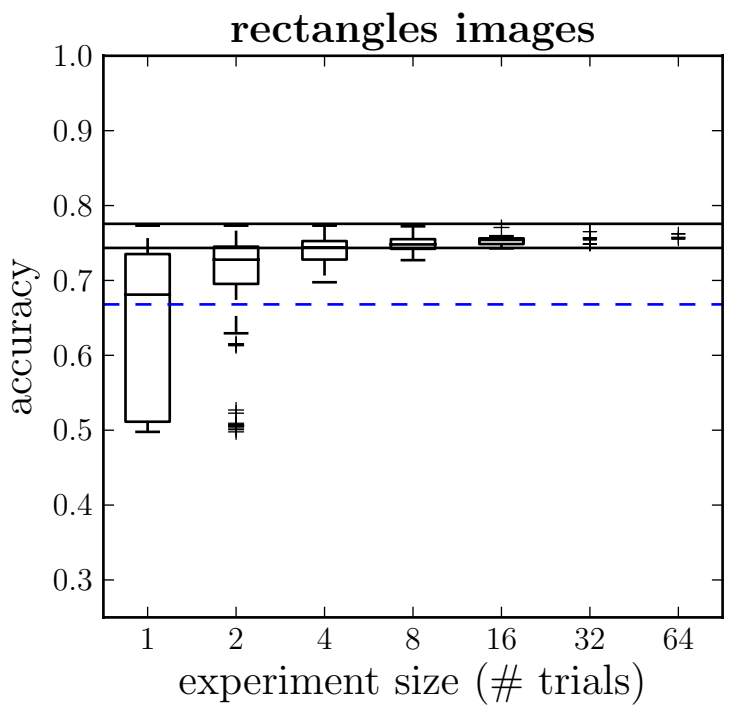




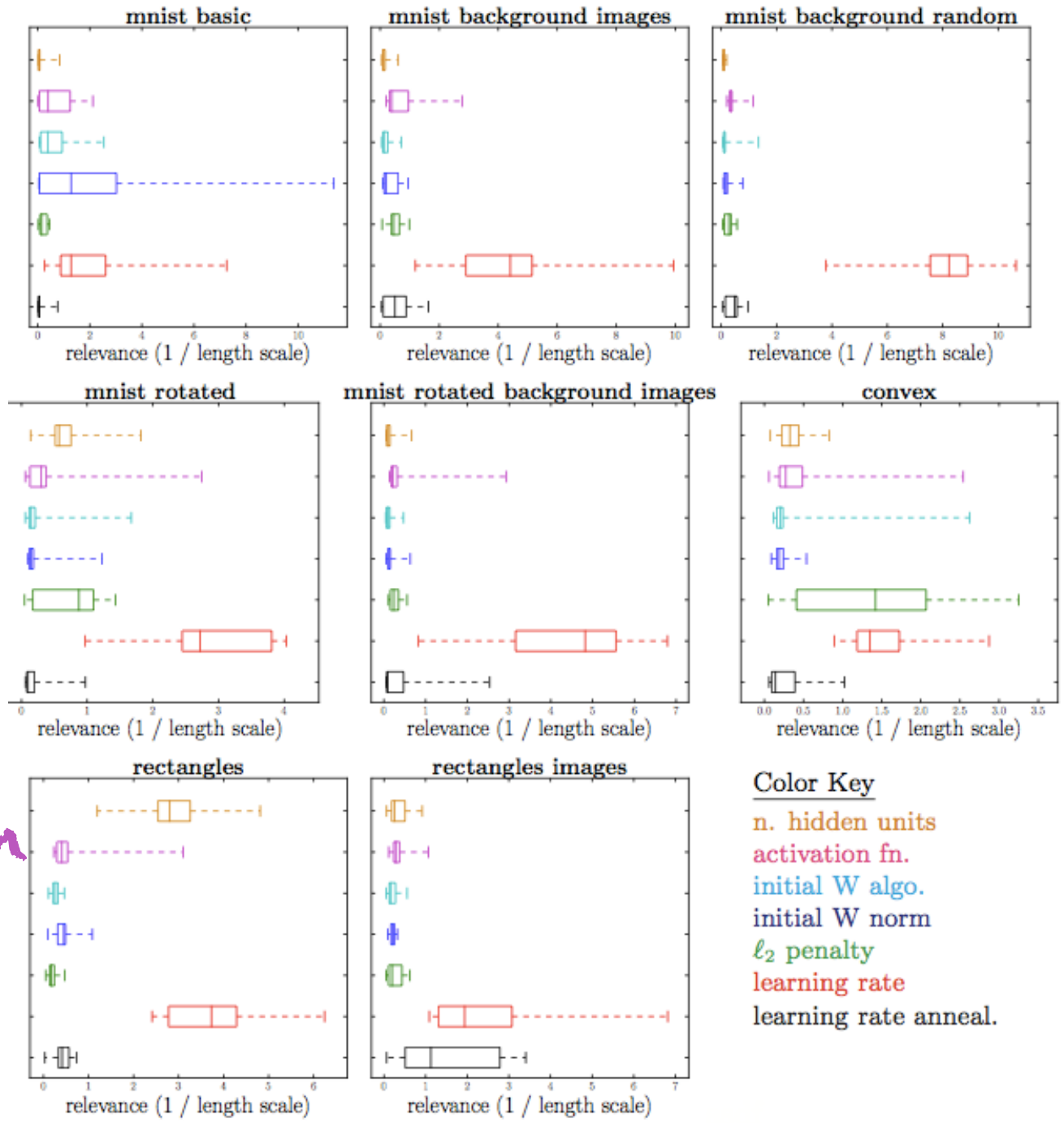
Random Search Learning Curves

- Blue dotted line = grid search with 100 trials



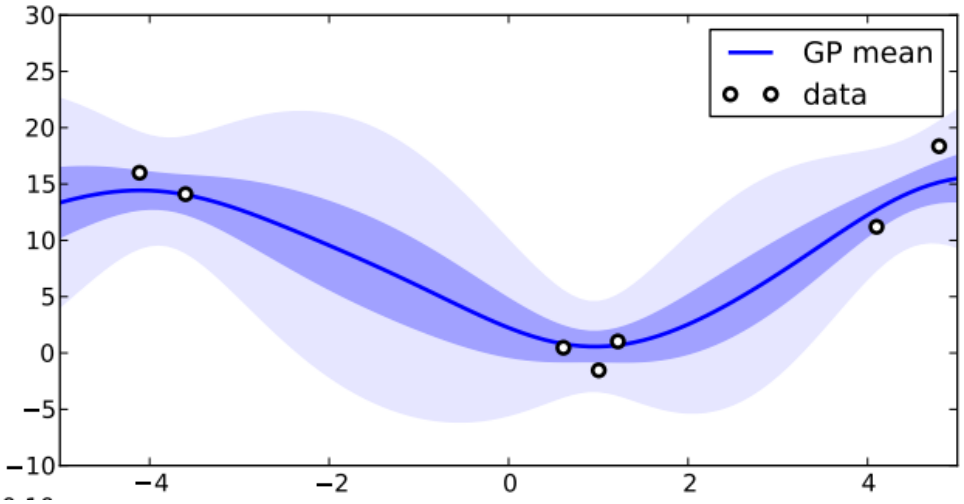


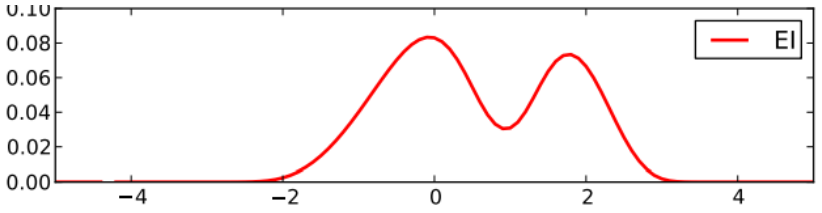
The Low Effective Dimension of Hyper-Optimization



Sequential Model-Based Optimization of Hyper-Parameters

- (Hutter et al JAIR 2009; Bergstra et al NIPS 2011; Thornton et al arXiv 2012; Snoek et al NIPS 2012)
- Iterate
- Estimate P(valid. err | hyper-params config x, D)
- choose optimistic x, e.g. $\max_x$ P(valid. err < current min. err | x)
- train with config x, observe valid. err. v, D ← D U {(x,v)}





Is Sequential Model-Based Optimization of Hyperparameters Already Commonly Used in my Lab?

No, not yet.

Why?

- Need handling conditional variables (that exist only if some variables take some values)
- Need automatic stopping of training trials that will fail
- Need parallelized version (what are the next best N trials to run in parallel?)
- It needs to work substantially better than random search, since the latter is so simple. For this to happen probably need to learn from more than a single sequence of runs, across datasets, etc.

Conclusions

- Deep Learning and neural nets have scared novice practitioners in the past because of the many hyper-parameters
- Hyper-parameter optimization is crucial to make deep learning easier to use
- Random sampling + manual selection of ranges + iterate remains the method most commonly used
- Need more R&D in hyper-parameter optimization, and easy to use robust software