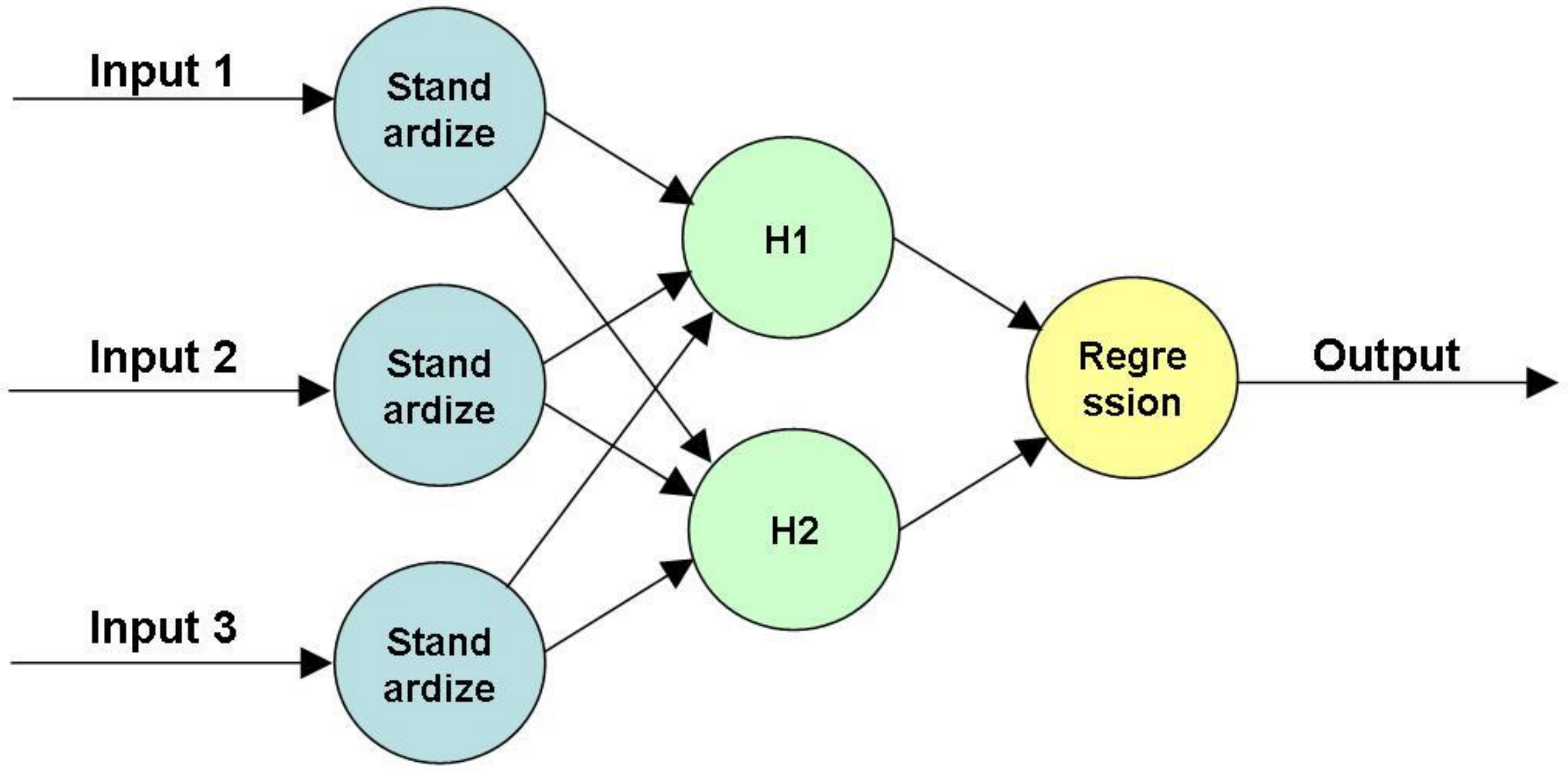
Inspire…Educate…Transform.

Inspire…Educate…Transform.

Deep Learning

Dr. K. V. Dakshinamurthy
President, International School of Engineering


Input 1
Input 2
Input 3
Stand ardize
Stand ardize
Stand ardize
H1
H2
Regre ssion
Output

Demos of neural net’s non-linearity power

- http://image.baidu.com/?fr=shitu
- http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html
- http://cs.stanford.edu/people/karpathy/convnetjs/demo/regression.html

Which is better?

- Shallow layers and many nodes
- Many layers and fewer nodes?
  - Technical
  - Business
  - Philosophical

Technical issues

- Computationally multi-layers are better

- Vanishing gradients: as we add more and more hidden layers, backpropagation becomes less and less useful in passing information to the lower layers. In effect, as information is passed back, the gradients begin to vanish and become small relative to the weights of the networks.

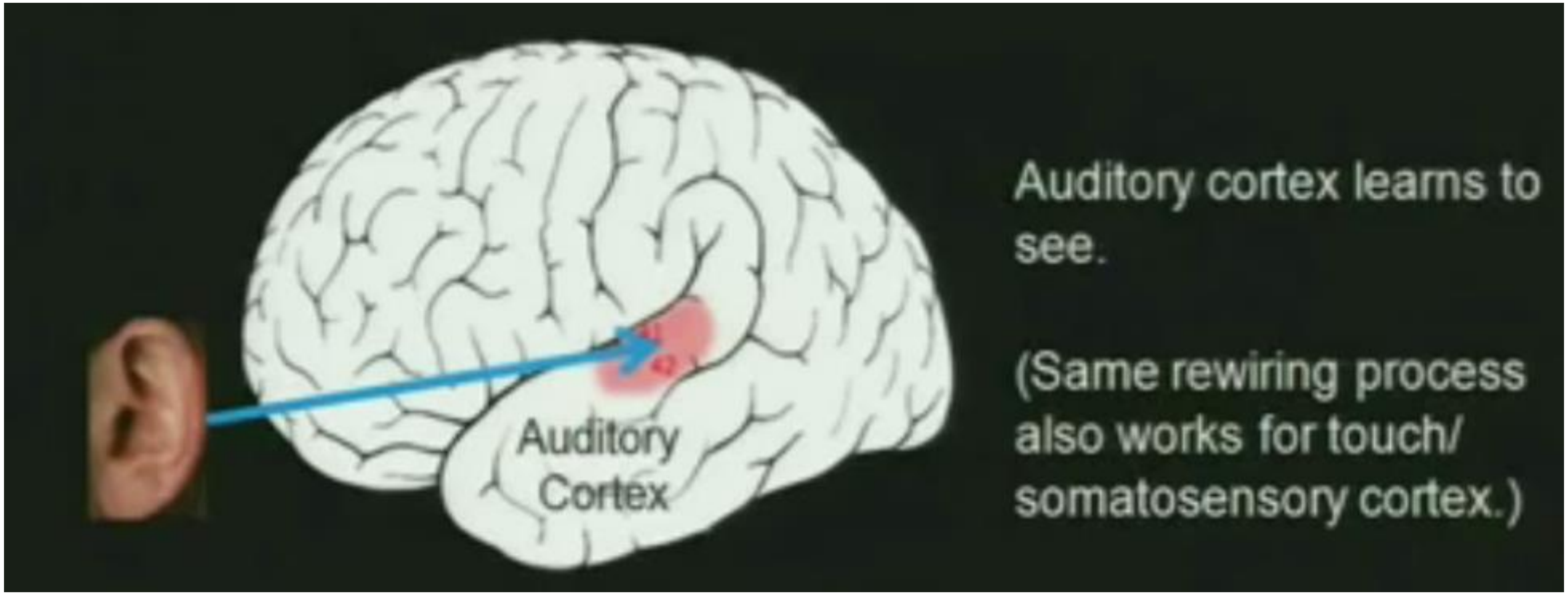
- Overfitting: perhaps the central problem in Machine Learning. Briefly, overfitting describes the phenomenon of fitting the training data *too* closely, maybe with hypotheses that are *too* complex. In such a case, your learner ends up fitting the training data really well, but will perform much, much more poorly on real examples.

Business issues: Feature engineering

Taking inputs, transforming inputs and feeding the transformed inputs to an algorithm — is called feature engineering. Feature engineering is very difficult, and there are little resources which help you to learn this skill. In consequence, there are very few people which can apply feature engineering skillfully to a wide range of tasks.

Feature engineering is — hands down — the most important skill to score well in Kaggle competitions. Feature engineering is so difficult because for each type of data and each type of problem, different features do well: Knowledge of feature engineering for image tasks will be quite useless for time series data; and even if we have two similar image tasks, it will not be easy to engineer good features because the objects in the images also determine what will work and what will not. It takes a lot of experience to get all of this right.

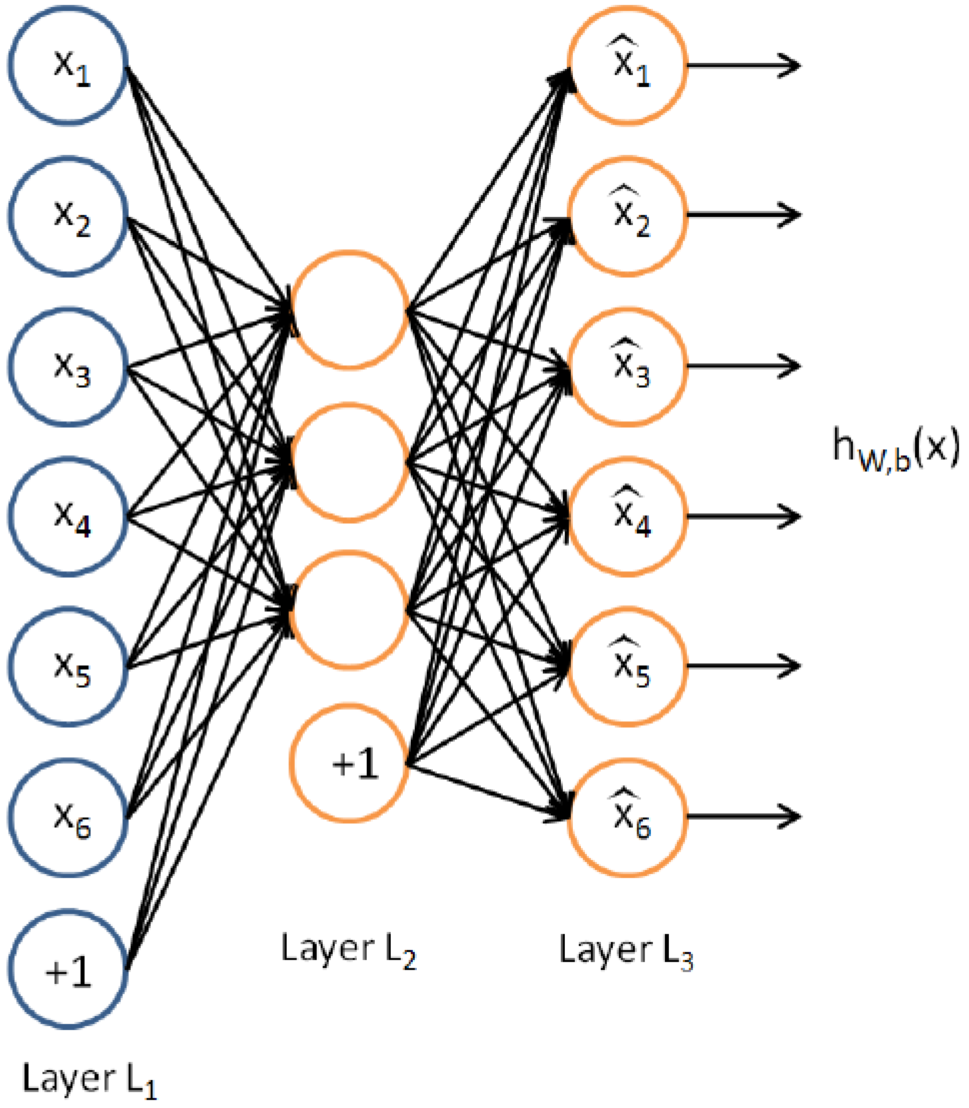
Philosophical issues



Auditory cortex learns to see.

(Same rewiring process also works for touch/ somatosensory cortex.)

Autoencoders



The network is trained to "recreate" the input. An autoencoder is a feed forward neural network which aims to *learn a compressed, distributed representation (encoding) of a dataset.*

8

Autoencoding

- Can have more interesting architectures
- One layer can be used (with equal nodes) just to denoise and then compress (smaller nodes) and then un-denoise and expand

Autoencoders

- The intuition behind this architecture is that the network will not learn a “mapping” between the training data and its labels, but will instead learn the *internal structure* and features of the data itself.

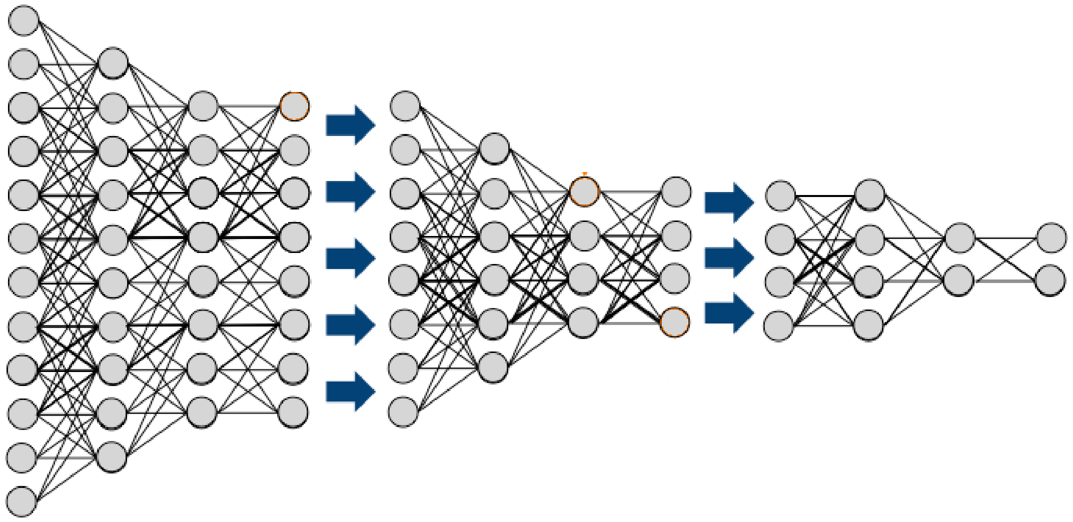
- Usually, the number of hidden units is smaller than the input/output layers, which forces the network to learn only the most important features and achieves a dimensionality reduction.

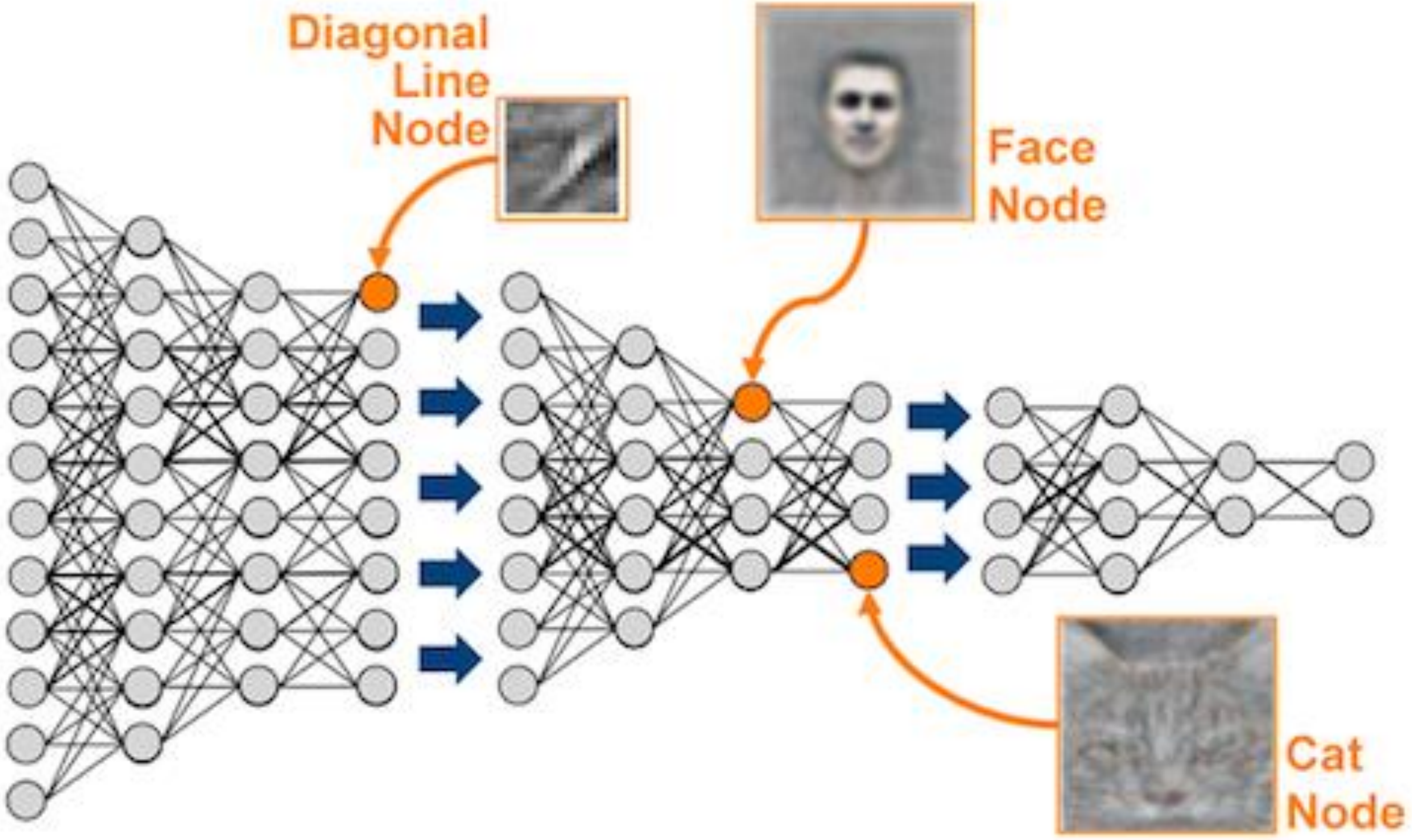
Learning concise features demo

- http://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html

Stack them

- Hidden layer 1 becomes visible layer for the hidden layer 2 and one can keep stacking




Diagonal Line Node
Face Node
Cat Node

Deep learners as predictive machines

- Build a stacked autoencoders
- Add desired output as last layer
- Learn weights through one network level back propagation using already computed weights as initial weights

Committing the same mistake

- http://ai.stanford.edu/~joni/papers/LasersonXRDS2011.pdf

To avoid overfitting

REGULARIZATION

Information theory metrics: While comparing multiple models

$$\mathrm{AIC} = 2k - 2\ln(L)$$

$$\mathrm{BIC} = -2 \cdot \ln \hat{L} + k \cdot \ln(n).$$

Ridge regression

$$\min_w \frac{1}{n}(\hat{X}w - \hat{Y})^2 + \lambda\|w\|_2^2$$

$$\nabla = \frac{2}{n}\hat{X}^T(\hat{X}w - \hat{Y}) + 2\lambda w$$

$$0 = \hat{X}^T(\hat{X}w - \hat{Y}) + n\lambda w$$

$$w = (\hat{X}^T\hat{X} + \lambda n I)^{-1}(\hat{X}^T\hat{Y})$$

Lasso and Elastic Net

$$\min_{w \in \mathbb{R}^p} \frac{1}{n} \|\hat{X}w - \hat{Y}\|^2 + \lambda \|w\|_1$$

Lasso: Sparsity

$$\min_{w \in \mathbb{R}^p} \frac{1}{n} \|\hat{X}w - \hat{Y}\|^2 + \lambda(\alpha \|w\|_1 + (1-\alpha)\|w\|_2^2), \alpha \in [0, 1]$$

Elastic net: Grouping

Why does Lasso lead to sparsity

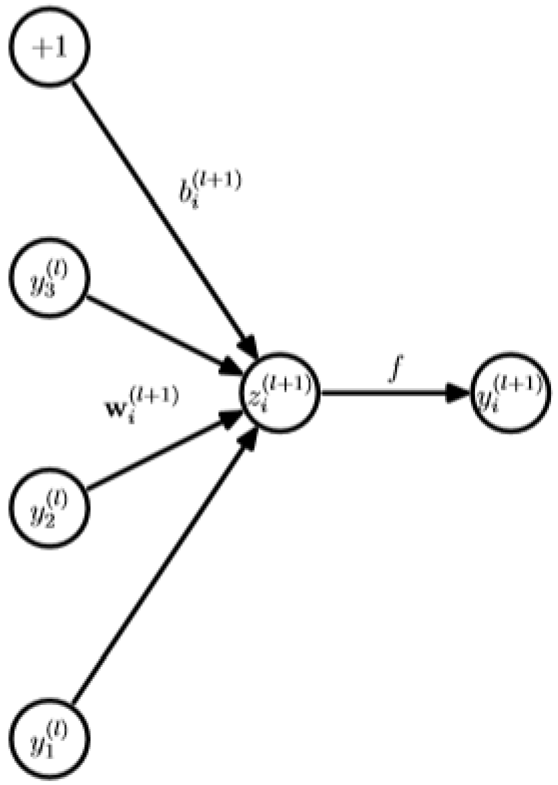
In particular, consider the vectors $a = (0.5, 0.5)$ and $b = (-1, 0)$. We can compute two possible norms:

- $||a||_1 = |0.5| + |0.5| = 1$ and $||b||_1 = |-1| + |0| = 1$
- $||a||_2 = \sqrt{0.5^2 + 0.5^2} = 1/\sqrt{2} < 1$ and $||b||_2 = \sqrt{(-1)^2 + (0)^2} = 1$

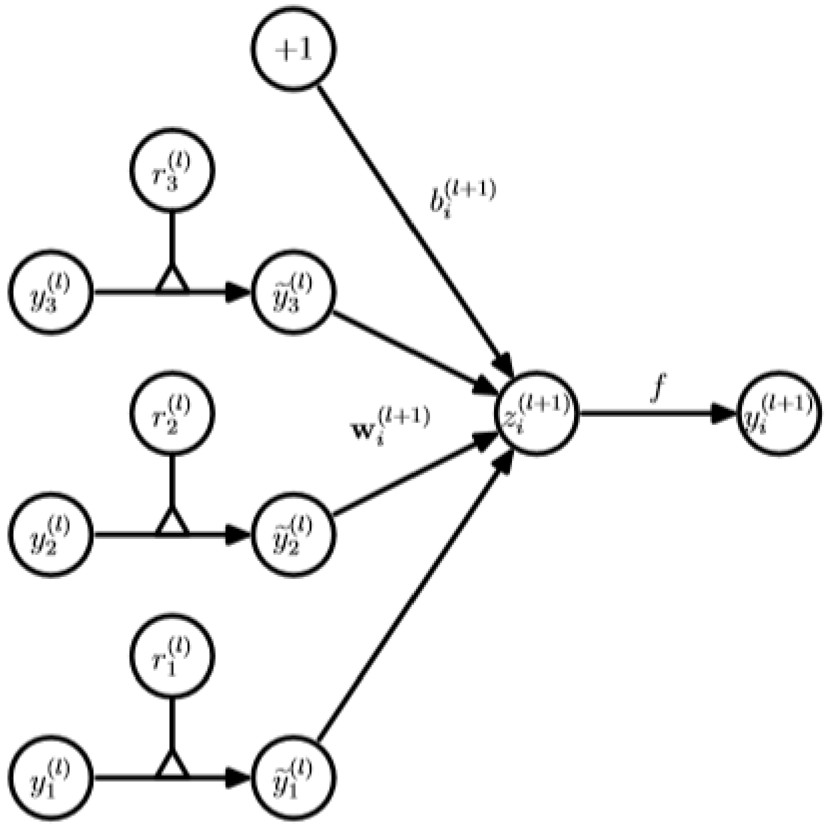
So, the two vectors are equivalent with respect to the L1 norm but different with respect to the L2 norm. This is because **squaring a number punishes large values more than it punishes small values**.

Keep in mind that ridge regression can't zero out coefficients; thus, you either end up including all the coefficients in the model, or none of them. In contrast, the LASSO does both parameter shrinkage and variable selection automatically. If some of your covariates are highly correlated, you may want to look at the Elastic Net [3] instead of the LASSO.

If you have both implemented, use subsets of your data to find the ridge and the lasso and compare how well they work on the left out data. The errors should give you an idea of which to use.



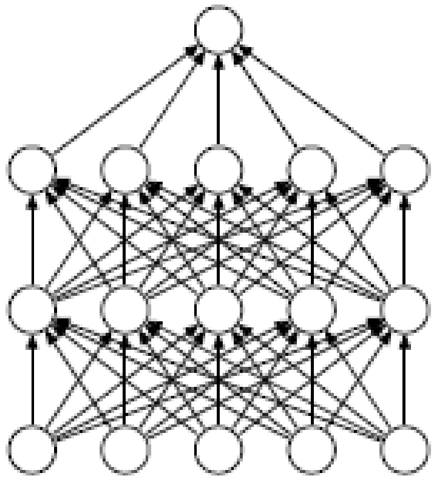
(a) Standard network

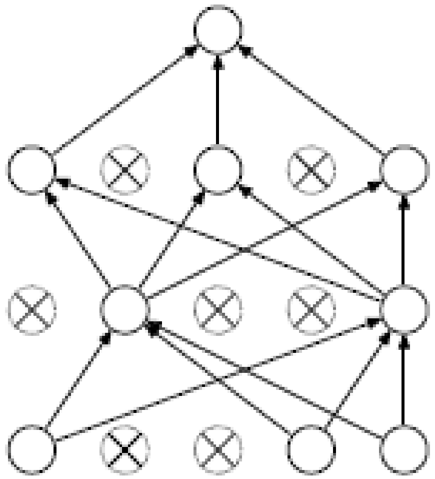


(b) Dropout network

Drop outs

http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf


(a) Standard Neural Net


(b) After applying dropout.

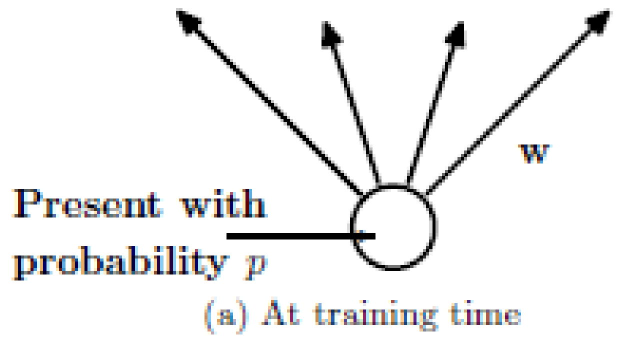
Overfitting

- Model combination nearly always improves the performance of machine learning methods. With large neural networks, however, the obvious idea of averaging the outputs of many separately trained nets is prohibitively expensive. Combining several models is most helpful when the individual models are different from each other and in order to make neural net models different, they should either have different architectures or be trained on different data.

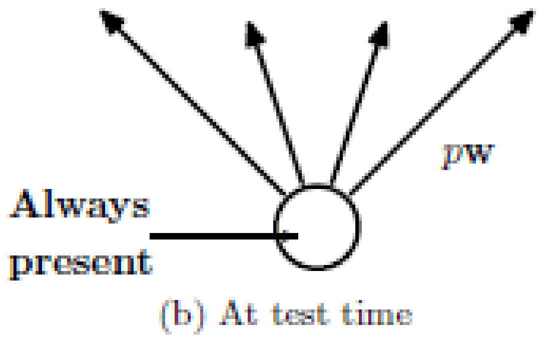
Dropout

- The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure 1. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5.

Drop out



(a) At training time



(b) At test time

Motivation

- Reproduction: Nature produces offspring by combining distinct genes rather than strengthening co-adopting them.

Ensembling in dropout

- Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout. A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. These networks all share weights so that the total number of parameters is still $O(n^2)$, or less. For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

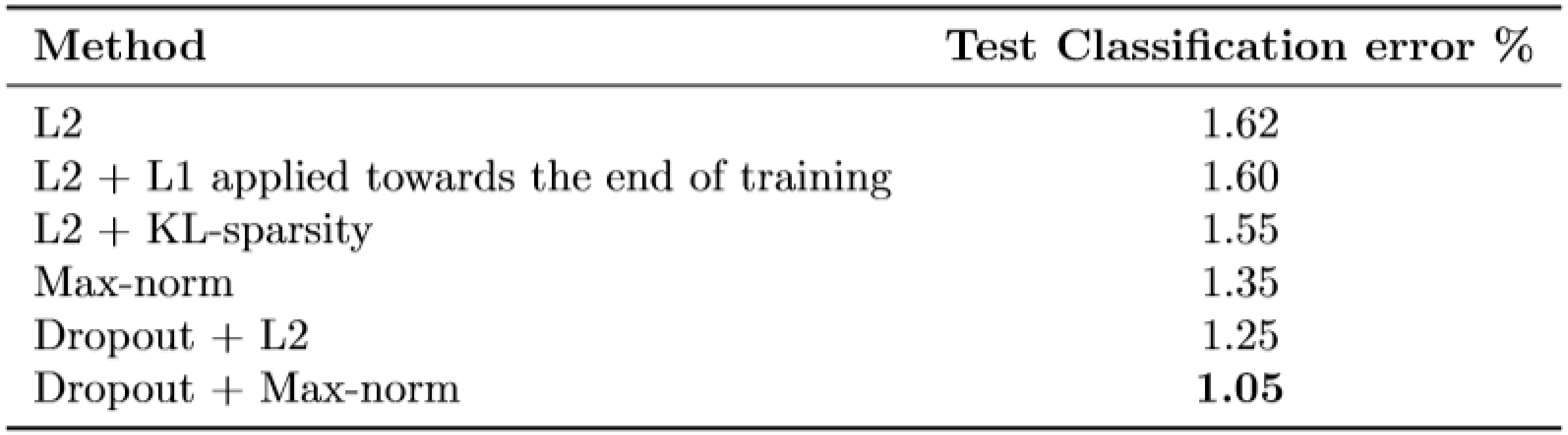
- The idea of dropout is not limited to feed-forward neural nets. It can be more generally applied to graphical models such as Autoencoders. Dropout RBMs are found to be better than standard RBMs in certain respects.

Add-ons

- One particular form of regularization was found to be especially useful for dropout—constraining the norm of the incoming weight vector at each hidden unit to be upper bounded by a fixed constant c. In other words, if w represents the vector of weights incident on any hidden unit, the neural network was optimized under the constraint $||w||^2 \leq c$. This is also called max-norm regularization since it implies that the maximum value that the norm of any weight can take is c.

- Although dropout alone gives significant improvements, using dropout along with maxnorm regularization, large decaying learning rates and high momentum provides a significant boost over just using dropout.
- A possible justification is that constraining weight vectors to lie inside a ball of fixed radius makes it possible to use a huge learning rate without the possibility of weights blowing up. The noise provided by dropout then allows the optimization process to explore different regions of the weight space that would have otherwise been difficult to reach. As the learning rate decays, the optimization takes shorter steps, thereby doing less exploration and eventually settles into a minimum.

Method	**Unit Type**	**Architecture**	**Error %**
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5×240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**



Method	Test Classification error %
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

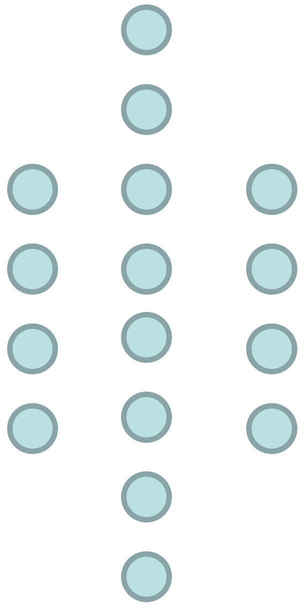
Practical tips

- Network size: n/p
- Learning and momentum: As there is significant noise we need to compensate. 10-100 times the learning rate of normal nets and 0.95-0.99 momentum.
- Max norm regularization
- Keep rate of 0.5 to 0.8 for hidden and 0.8-0.9 for inputs

Sparse autoencoders

- What are the ideal properties of machine generated features
  - An example must be explained primarily by a few features
  - A feature must belong to only a few features
  - All features should have similar activities

Sparse autoencoders



$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^{(2)}(x^{(i)})\right]$$

be the average activation of hidden unit j (averaged over the training set). We would like to (approximately) enforce the constraint

$$\hat{\rho}_j = \rho,$$

where ρ is a **sparsity parameter**, typically a small value close to zero (say $\rho = 0.05$). In other words, we would like the average activation of each hidden neuron j to be close to 0.05 (say). To satisfy this constraint, the hidden unit's activations must mostly be near 0.

HYPER PARAMETER TUNING

Why is it important

- Linear and logistic regression do not have any parameters to tune. Of course, they have weights to learn
- Regularization has a parameter to tune (α)
- ANNs have learning rate, number of layers, number of nodes per layer to tune

```
h2o.deeplearning(x, y, training_frame, model_id = "", overwrite_with_best_model, validation_frame =
NULL, checkpoint = NULL, autoencoder = FALSE, pretrained_autoencoder = NULL,
use_all_factor_levels = TRUE, standardize = TRUE, activation = c("Rectifier", "Tanh", "TanhWithDropout",
"RectifierWithDropout", "Maxout", "MaxoutWithDropout"), hidden = c(200, 200), epochs = 10,
train_samples_per_iteration = -2, target_ratio_comm_to_comp = 0.05, seed, adaptive_rate = TRUE, rho
= 0.99, epsilon = 1e-08, rate = 0.005, rate_annealing = 1e-06, rate_decay = 1, momentum_start = 0,
momentum_ramp = 1e+06, momentum_stable = 0, nesterov_accelerated_gradient = TRUE,
input_dropout_ratio = 0, hidden_dropout_ratios, l1 = 0, l2 = 0, max_w2 = Inf, initial_weight_distribution =
c("UniformAdaptive", "Uniform", "Normal"), initial_weight_scale = 1, loss = c("Automatic", "CrossEntropy",
"Quadratic", "Absolute", "Huber"), distribution = c("AUTO", "gaussian", "bernoulli", "multinomial",
"poisson", "gamma", "tweedie", "laplace", "huber", "quantile"), quantile_alpha = 0.5, tweedie_power = 1.5,
score_interval = 5, score_training_samples, score_validation_samples, score_duty_cycle,
classification_stop, regression_stop, stopping_rounds = 5, stopping_metric = c("AUTO", "deviance",
"logloss", "MSE", "AUC", "r2", "misclassification"), stopping_tolerance = 0, max_runtime_secs = 0,
quiet_mode, max_confusion_matrix_size, max_hit_ratio_k, balance_classes = FALSE,
class_sampling_factors, max_after_balance_size, score_validation_sampling, missing_values_handling =
c("MeanImputation", "Skip"), diagnostics, variable_importances, fast_mode, ignore_const_cols,
force_load_balance, replicate_training_data, single_node_mode, shuffle_training_data, sparse,
col_major, average_activation, sparsity_beta, max_categorical_features, reproducible = FALSE,
export_weights_and_biases = FALSE, offset_column = NULL, weights_column = NULL, nfolds = 0,
fold_column = NULL, fold_assignment = c("AUTO", "Random", "Modulo"),
keep_cross_validation_predictions = FALSE)
```

It is important to tune

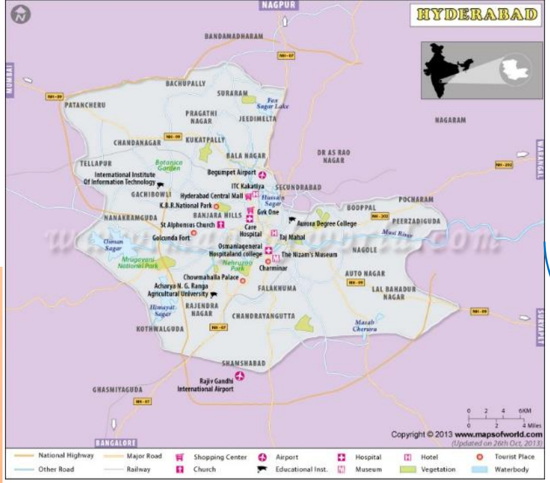
- The parameters can drastically impact the performance

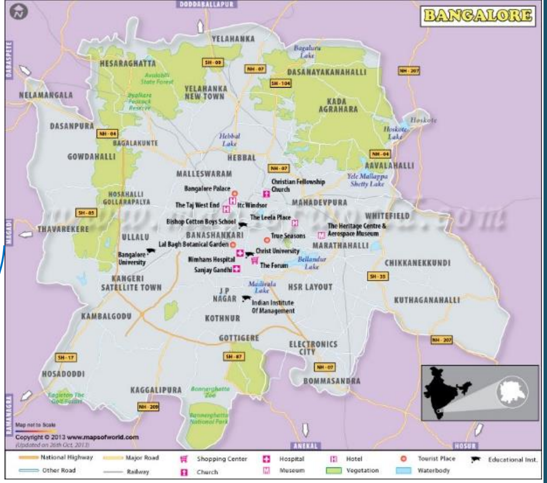
- Tuning is receiving a lot of attention
  - Grid or exhaustive search
  - Random or MonteCarlo simulation
  - Bayesian inference or simulated annealing
  - Regression of performance on parameters

- A recommended process
  - Use Montecarlo and select 100 combinations of hyper parameters
  - Compute the performance of each of those combinations
  - Build a regressor (RF or any other technique) where performance is regressed on hyper parameters
  - Run GA (you need not run the NN each time, you can use the regressor to predict the performance for various combinations)

Deciding the hyper parameters

- Initialize random weights
- Try various architectures and compute the best hyper parameters using any of the above four methods for these weights
- Pick top 3 and then train them to learn the weights for the given architecture





HYDERABAD

Office and Classrooms
Plot 63/A, Floors 1&2, Road # 13, Film Nagar,
Jubilee Hills, Hyderabad - 500 033
+91-9701685511 (Individuals)
+91-9618483483 (Corporates)

BENGALURU

Office
Incubex, #728, Grace Platina, 4th Floor, CMH Road,
Indira Nagar, 1st Stage, Bengaluru – 560038
+91-9502334561 (Individuals)
+91-9502799088 (Corporates)

Classroom
KnowledgeHut Solutions Pvt. Ltd., Reliable Plaza,
Jakkasandra Main Road, Teacher's Colony, 14th Main
Road, Sector – 5, HSR Layout, Bengaluru - 560102

Social Media

Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering

This presentation may contain references to findings of various reports available in the public domain. INSOFE makes no representation as to their accuracy or that the organization subscribes to those findings.