Deep Learning

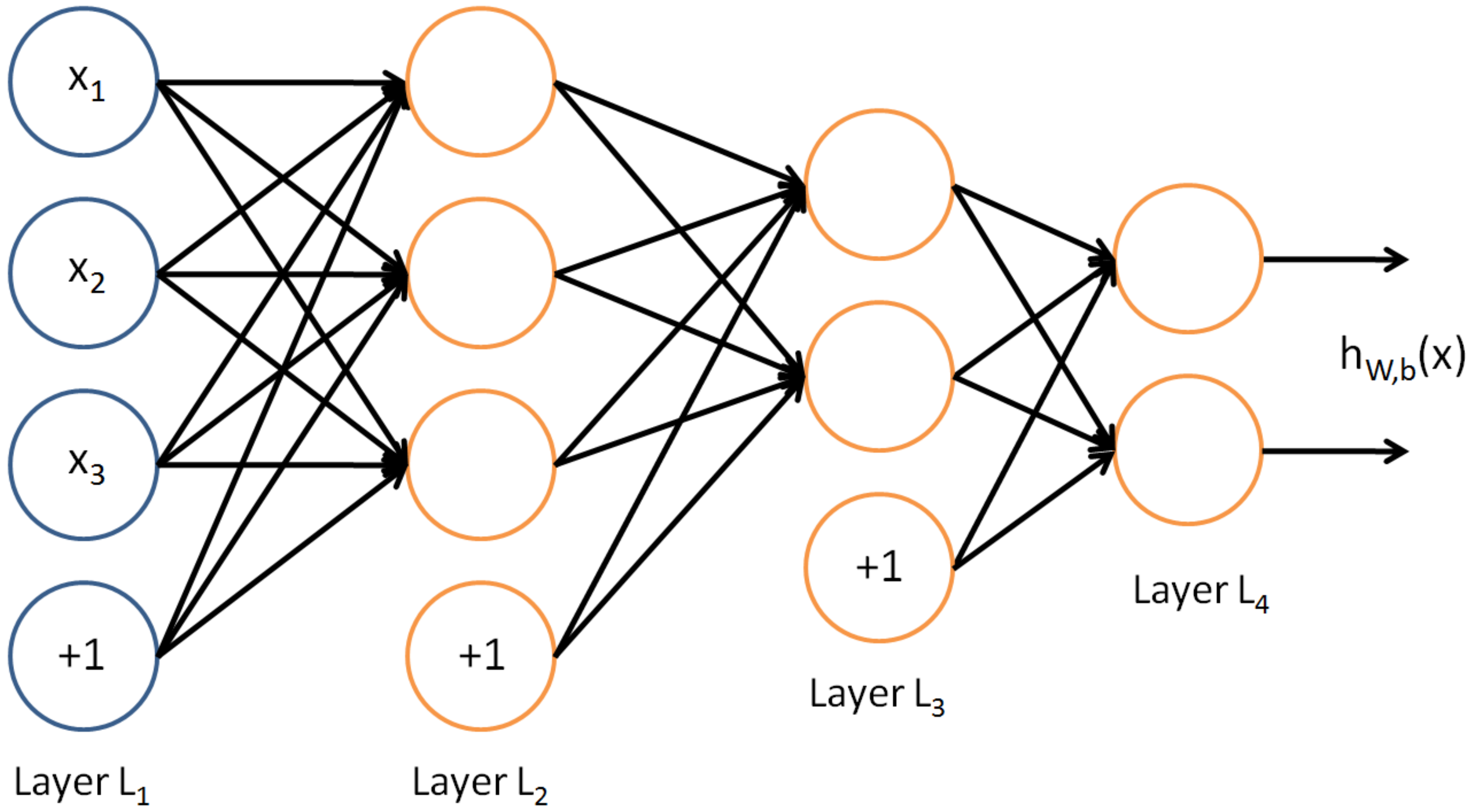
Patricia J Riddle
Computer Science 760

Based on UFLDL Tutorial
by Andrew Ng et. Al.
http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

What is a deep neural network?

- an ANN with multiple hidden layers
  - which allows it to compute much more complex features of the input.
- Each hidden layer computes a non-linear transformation of the previous layer
  - a deep network can have significantly greater representational power (i.e., can learn significantly more complex functions) than a shallow one.
- Note that when training a deep network, it is important to use a non-linear activation function in each hidden layer.
  - Multiple layers of linear functions would itself compute only a linear function of the input (i.e., composing multiple linear functions together results in just another linear function), and thus be no more expressive than using just a single layer of hidden units.

Deep Learning



Common Backpropagation Features

- Weight decay – as before
- Symmetry breaking – initialization to small random numbers

How to get More Data

- Given a sufficiently powerful learning algorithm, one of the most reliable ways to get better performance is to give the algorithm more data.
  - This has led to the that aphorism that in machine learning, "sometimes it's not who has the best algorithm that wins; it's who has the most data."

- One can always try to get more labeled data, but this can be expensive. Researchers have gone to extraordinary lengths using tools such as AMT (Amazon Mechanical Turk) to get large training sets.

- Having large numbers of people hand-label lots of data is a step forward compared to having large numbers of researchers hand-engineer features, but it would be nice to do better.

Promise of self-taught learning

- If we get our algorithms to learn from unlabeled data, then we can easily obtain and learn from massive amounts of it.

- A single unlabeled example is less informative than a single labeled example
- But if we can get tons of the former
  - downloading random unlabeled images/audio clips/text documents off the internet
- And if our algorithms can exploit this unlabeled data effectively,

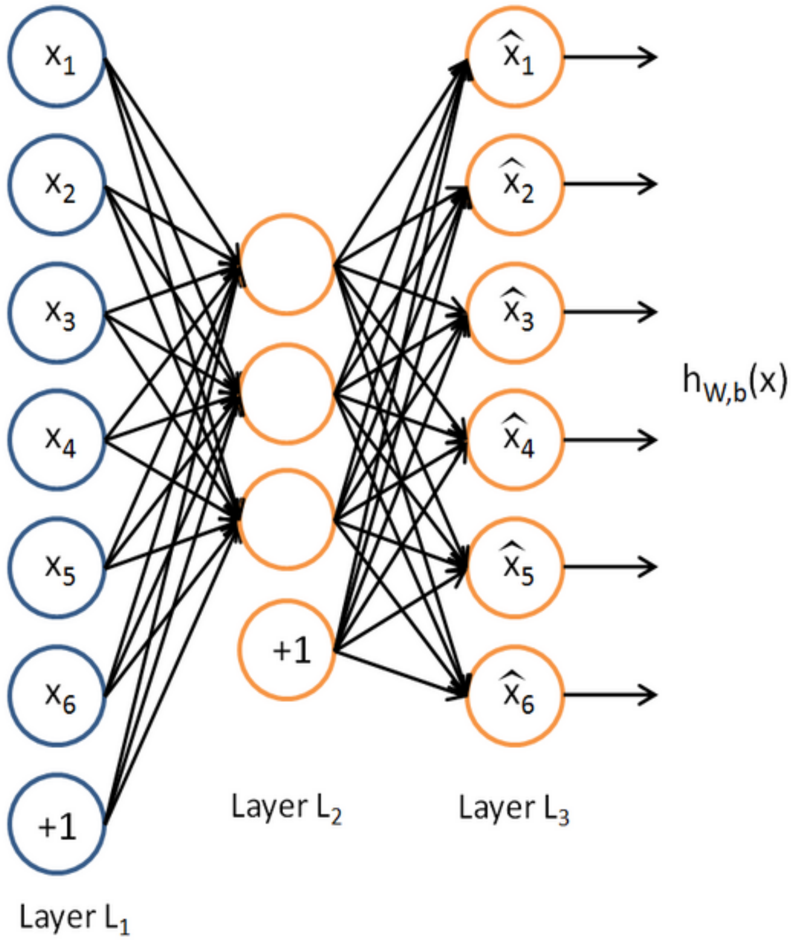
- Then we might be able to achieve better performance than the massive hand-engineering and massive hand-labeling approaches.

Self Taught Unsupervised Feature Learning

- We give our algorithms a large amount of unlabeled data with which to learn a good feature representation of the input.
- If we are trying to solve a specific classification task, we can take this learned feature representation and whatever (perhaps small amount of) labeled data we have for that classification task, and apply supervised learning on that labeled data to solve the classification task.
- These ideas are the most powerful when we have a lot of unlabeled data, and a smaller amount of labeled data.
- They give good results even if we have only labeled data (in which case we usually perform the feature learning step using the labeled data, but ignoring the labels).

Autoencoder

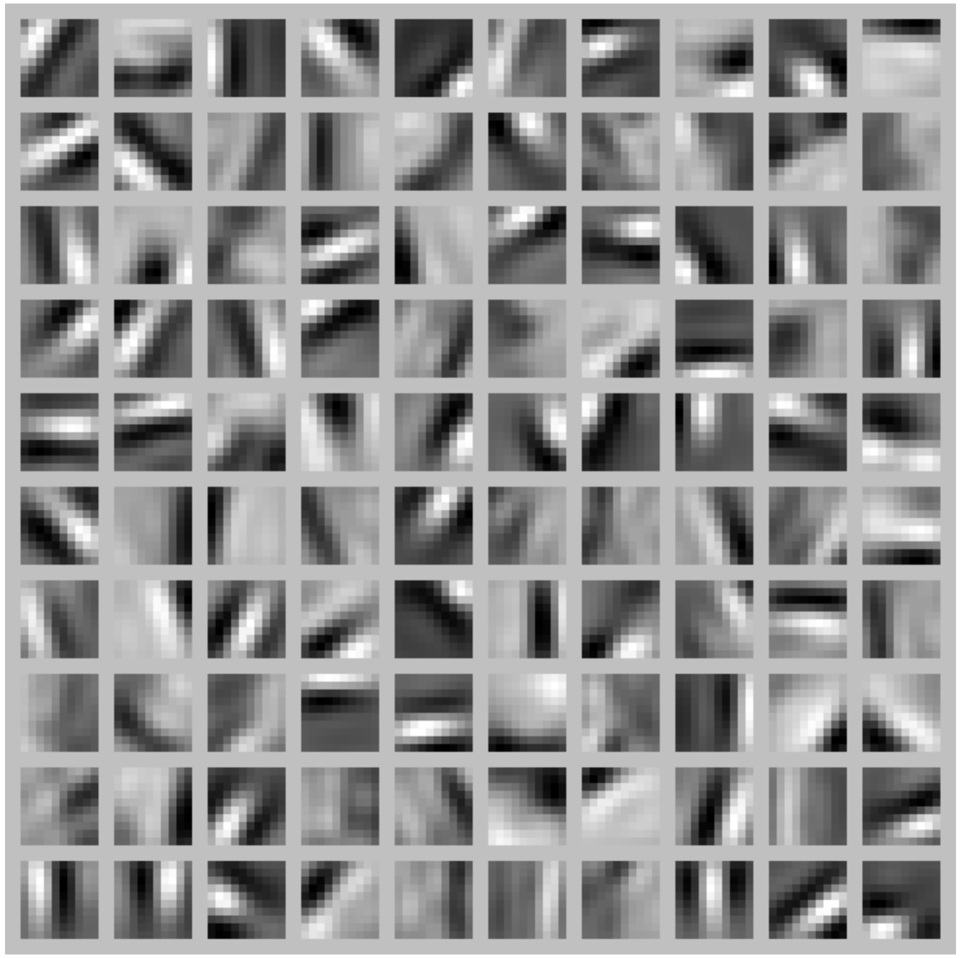
unsupervised learning
compressed representation



Sparsity parameter

- Sparsity Parameter, .05
- Average activation of each hidden neuron
- Most of the hidden units activations must be near 0

Visualization of Hidden Units

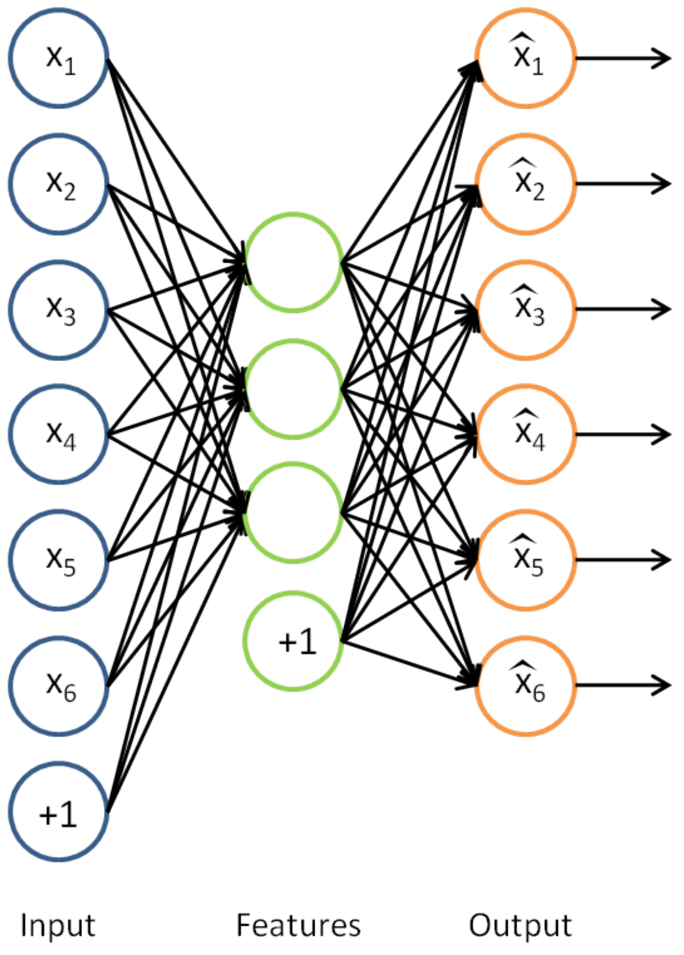


What the Visualization shows

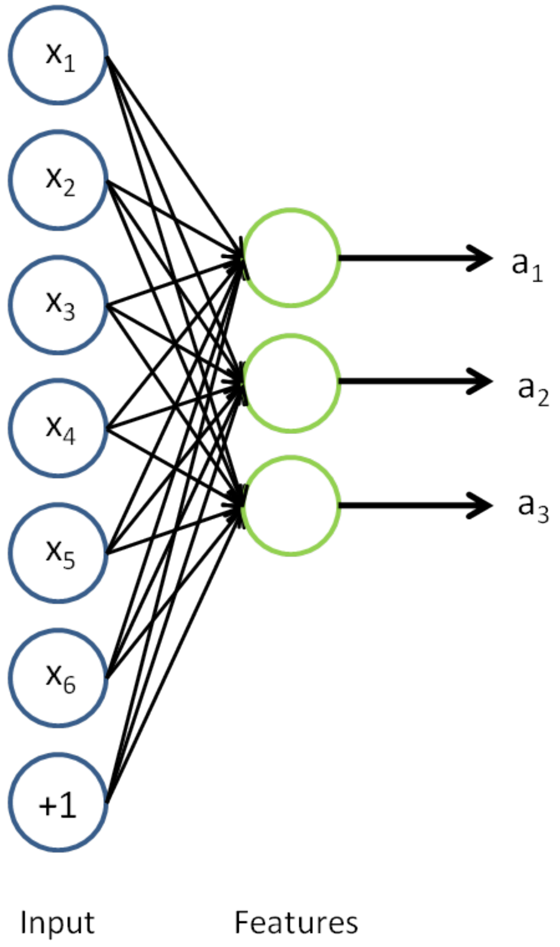
Each square shows the input image x that maximally actives one of 100 hidden units.

Different hidden units have learned to detect edges at different positions and orientations in the image.

Sparse Autocoder



New Features



Self Taught vs Semi-supervised

- Two common unsupervised feature learning settings, depending on what type of unlabeled data you have.
  - The more general and powerful setting is the self-taught learning setting, which does not assume that your unlabeled data has to be drawn from the same distribution as your labeled data.
  - The more restrictive setting where the unlabeled data comes from exactly the same distribution as the labeled data is sometimes called the semi-supervised learning setting

Self Taught

- Your goal is a computer vision task where you'd like to distinguish between images of cars and images of motorcycles;
  - each labeled example in your training set is either an image of a car or an image of a motorcycle.
- Where can we get lots of unlabeled data?
  - Obtain some random collection of images, perhaps downloaded off the internet.
- Train the autoencoder on this large collection of images, and obtain useful features from them.
- Because the unlabeled data is drawn from a different distribution than the labeled data (i.e., perhaps some of our unlabeled images may contain cars/motorcycles, but not every image downloaded is either a car or a motorcycle), we call this self-taught learning.

Semi-supervised

- If we happen to have lots of unlabeled images lying around that are all images of either a car or a motorcycle, but where the data is just missing its label (so you don't know which ones are cars, and which ones are motorcycles), then we could use this form of unlabeled data to learn the features.
- This setting---where each unlabeled example is drawn from the same distribution as your labeled examples---is sometimes called the semi-supervised setting.
- In practice, we often do not have this sort of unlabeled data (where would you get a database of images where every image is either a car or a motorcycle, but just missing its label?), and so in the context of learning features from unlabeled data, the self-taught learning setting is more broadly applicable.

Self taught vs semi-supervised

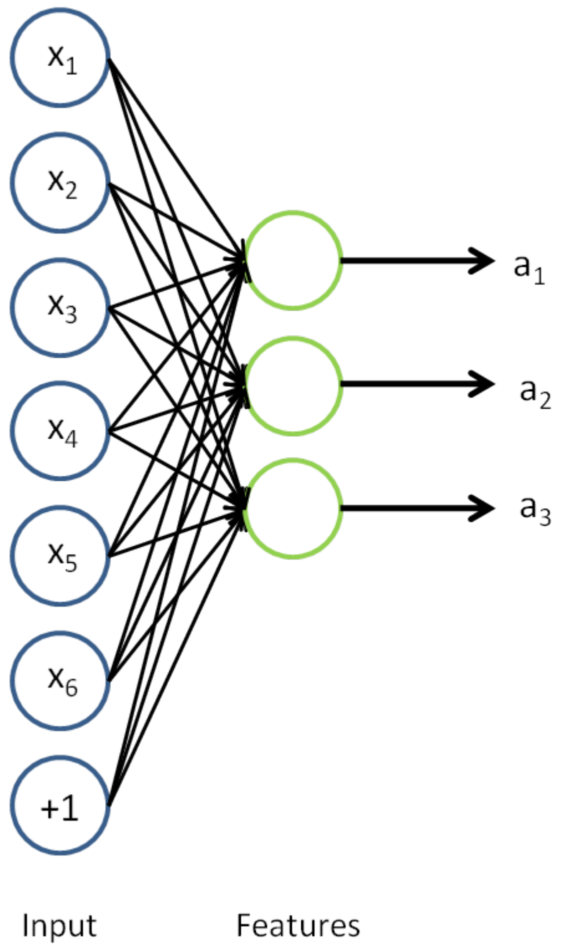
- But self-taught is also more dangerous!!
  - More apt NOT to work at all
  - Must be careful!!!

Deep Networks

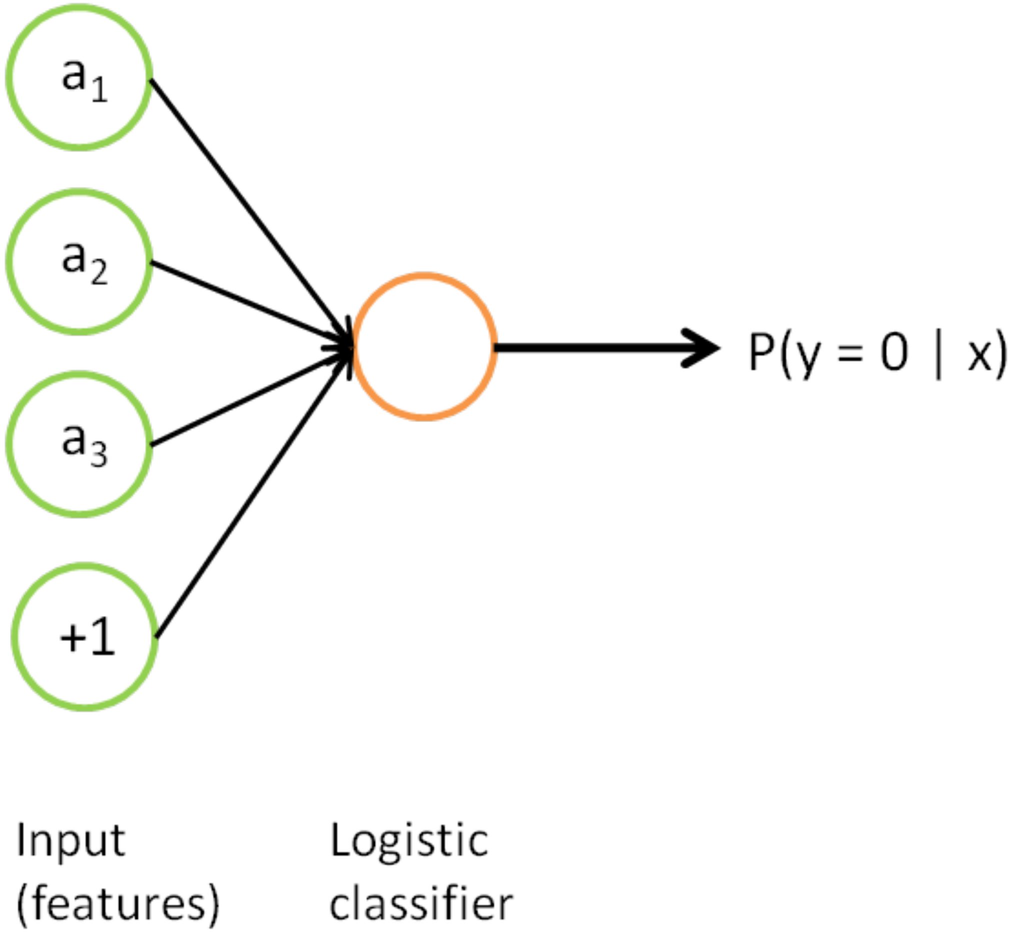
- In self-taught learning, we first trained a sparse autoencoder on the unlabeled data.

- Then, given a new example, we used the hidden layer to extract features.

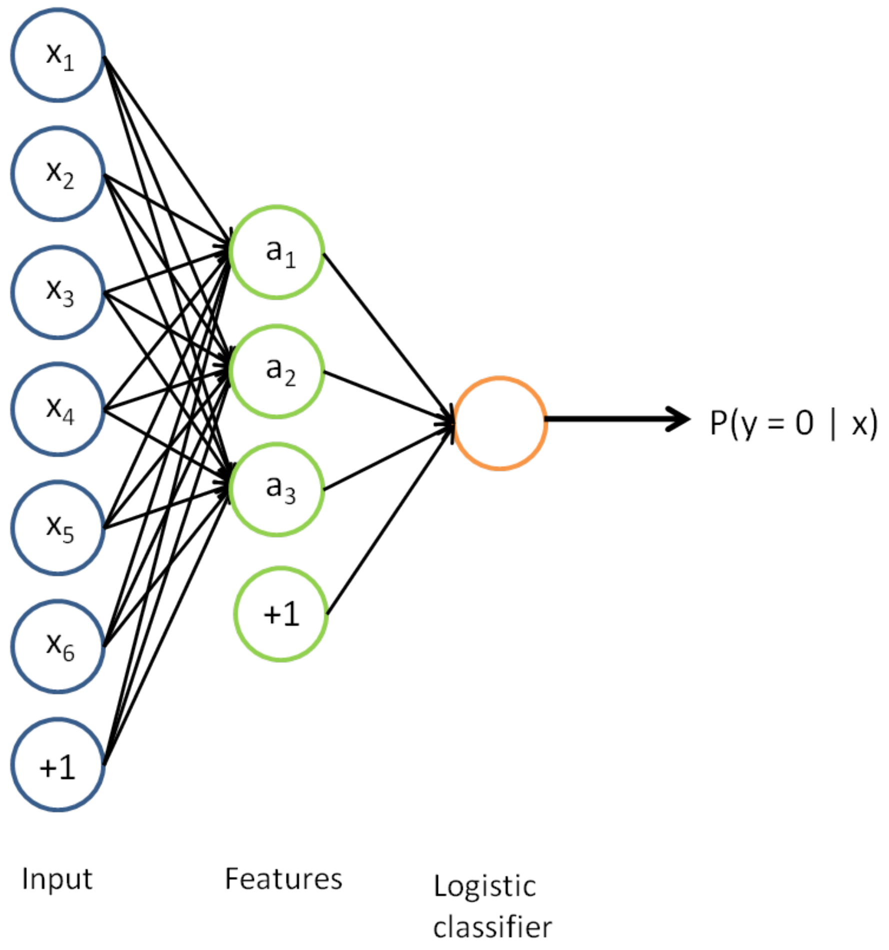
Learn Features



Train logistic classifier



Combine Both



Fine Tuning

- train the first layer using an autoencoder
- train the second layer via logistic/softmax regression)
- further modify all the parameters in our model to try to further reduce the training error.

- perform gradient descent (or use L-BFGS) from the current setting of the parameters to try to reduce the training error on our labeled training set.

- When fine-tuning is used, the original unsupervised feature learning steps are called pre-training.

- labeled data is used to modify the earlier layers weights, so that adjustments can be made to the features extracted by the layer of hidden units.

- When should we use fine-tuning?
  - if you have a large labeled training set,fine-tuning can significantly improve the performance of your classifier.
  - if you have a large unlabeled dataset and only a relatively small labeled training set, then fine-tuning is significantly less likely to help.

Deep Learning Advantages

- It can compactly represent a significantly larger set of functions than shallow networks.
  - Formally, there are functions which a k-layer network can represent compactly (with a number of hidden units that is polynomial in the number of inputs), that a (k − 1)-layer network cannot represent unless it has an exponentially large number of hidden units.

- In the case of images, one can also start to learn part-whole decompositions.
  - the first layer learns to group together pixels in an image in order to detect edges (as seen in the earlier exercises).
  - the second layer groups together edges to detect longer contours, or perhaps detect simple "parts of objects.”
  - An even deeper layer groups together these contours or detect even more complex features.

- Finally, cortical computations (in the brain) also have multiple layers of processing. For example, visual images are processed in multiple stages by the brain, by cortical area "V1", followed by cortical area "V2" (a different part of the brain), and so on.

Deep Learning Difficulties

- Randomly initializing the weights of a deep network, and then training it using a labeled training set using a supervised learning objective, for example by applying gradient descent to try to drive down the training error.
  - However, this usually did not work well.
  - There were several reasons for this.

Availability of Data

- With the method described above, one relies only on labeled data for training.
- However, labeled data is often scarce, and thus for many problems it is difficult to get enough examples to fit the parameters of a complex model.
- For example, given the high degree of expressive power of deep networks, training on insufficient data would also result in overfitting!!!!!!
- So with "enough data" – does it work just fine???

Local Optima

- Training a shallow network (with 1 hidden layer) using supervised learning usually resulted in the parameters converging to reasonable values; but when we are training a deep network, this works much less well.

- In particular, training a neural network using supervised learning involves solving a highly non-convex optimization problem.

- In a deep network, this problem turns out to be rife with bad local optima, and training with gradient descent (or methods like conjugate gradient and L-BFGS) no longer work well.

- Not sure I buy this - ??? – see next slide

Diffusion of gradients

- the gradients become very small, that explains why gradient descent (and related algorithms like L-BFGS) do not work well on a deep networks with randomly initialized weights.
- when using backpropagation to compute the derivatives, the gradients that are propagated backwards (from the output layer to the earlier layers of the network) rapidly diminish in magnitude as the depth of the network increases.
- the derivative of the overall cost with respect to the weights in the earlier layers is very small.
- when using gradient descent, the weights of the earlier layers change slowly, and the earlier layers fail to learn much.

- This problem is often called the "diffusion of gradients.”

- - see next slide!!!

Diffusion of gradients

- If the last few layers in a neural network have a large enough number of neurons, it may be possible for them to model the labeled data alone without the help of the earlier layers.
- training the entire network at once with all the layers randomly initialized ends up giving similar performance to training a shallow network (the last few layers) on corrupted input (the result of the processing done by the earlier layers).
- Is this just telling you that you have too many layers for the problem at hand??? – rip one out and try again???
- Or remove some of the nodes on the later layers (make it narrower)

How can we train a deep network?

- One method that has seen some success is the greedy layer-wise training method.

Greedy layer-wise training

- The main idea is to train the layers of the network one at a time, so that we first train a network with 1 hidden layer, and only after that is done, train a network with 2 hidden layers, and so on.
- At each step, we take the old network with k − 1 hidden layers, and add an additional k-th hidden layer (that takes as input the previous hidden layer k − 1 that we had just trained).
- Training can either be supervised (say, with classification error as the objective function on each step), but more frequently it is unsupervised (as in an autoencoder; details to provided later).
- The weights from training the layers individually are then used to initialize the weights in the final/overall deep network, and only then is the entire architecture "fine-tuned" (i.e., trained together to optimize the labeled training set error).

Success of greedy layer-wise training

Availability of data

- While labeled data can be expensive to obtain, unlabeled data is cheap and plentiful.
- The promise of self-taught learning is that by exploiting the massive amount of unlabeled data, we can learn much better models.
  - By using unlabeled data to learn a good initial value for the weights in all the layers (except for the final classification layer that maps to the outputs/predictions), our algorithm is able to learn and discover patterns from massively more amounts of data than purely supervised approaches.
- This often results in much better classifiers being learned.

Better local optima

- After having trained the network on the unlabeled data, the weights are now starting at a better location in parameter space than if they had been randomly initialized.
- We can then further fine-tune the weights starting from this location.
- Empirically, it turns out that gradient descent from this location is much more likely to lead to a good local minimum, because the unlabeled data has already provided a significant amount of "prior" information about what patterns there are in the input data.

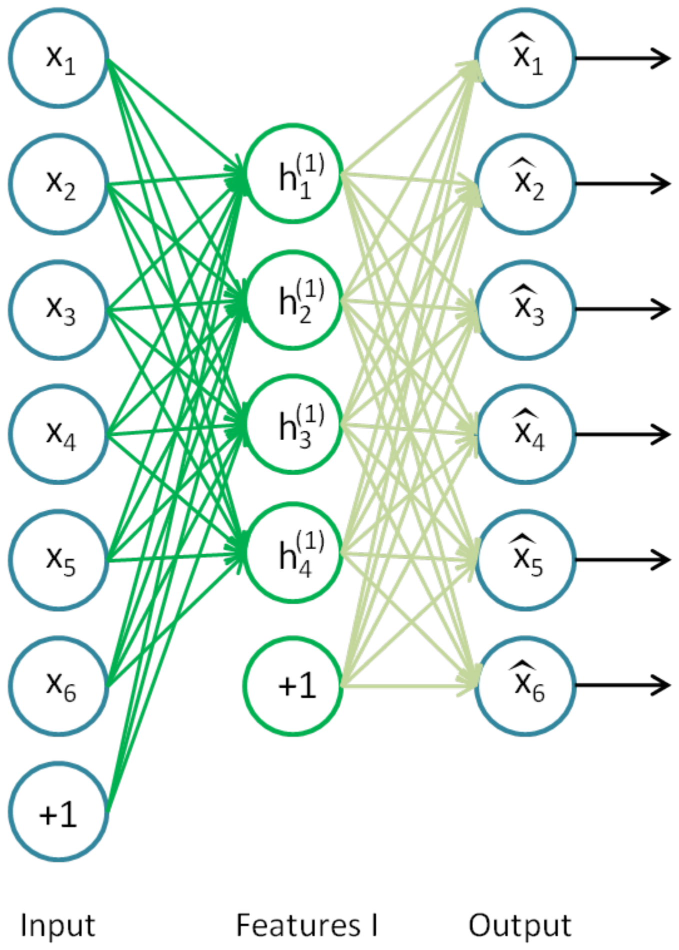
Stacked Autocoders

- The greedy layerwise approach for pretraining a deep network works by training each layer in turn.

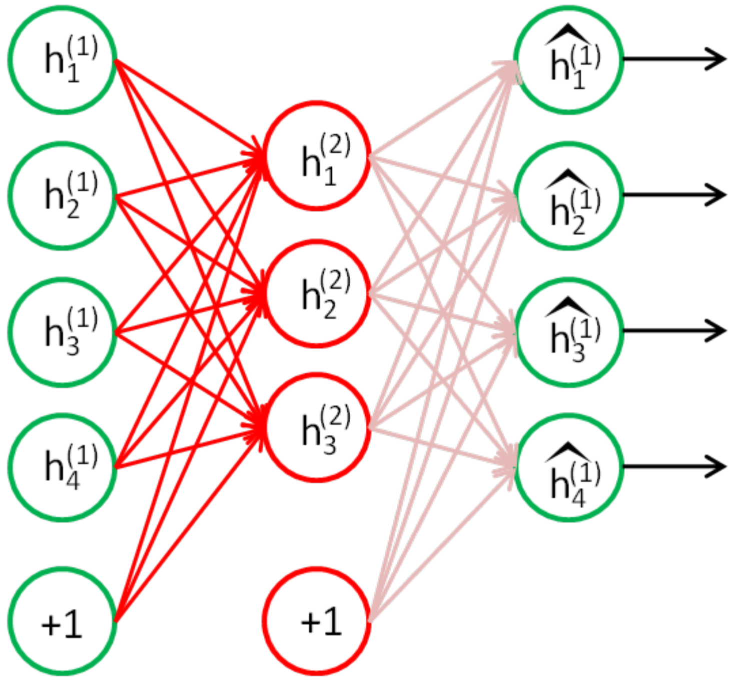
- This method trains the parameters of each layer individually while freezing parameters for the remainder of the model. To produce better results, after this phase of training is complete, fine-tuning using backpropagation can be used to improve the results by tuning the parameters of all layers are changed at the same time.

- If one is only interested in finetuning for the purposes of classification, the common practice is to then discard the "decoding" layers of the stacked autoencoder and link the last hidden layer a(n) to the softmax classifier. The gradients from the (softmax) classification error will then be backpropagated into the encoding layers.

First Layer

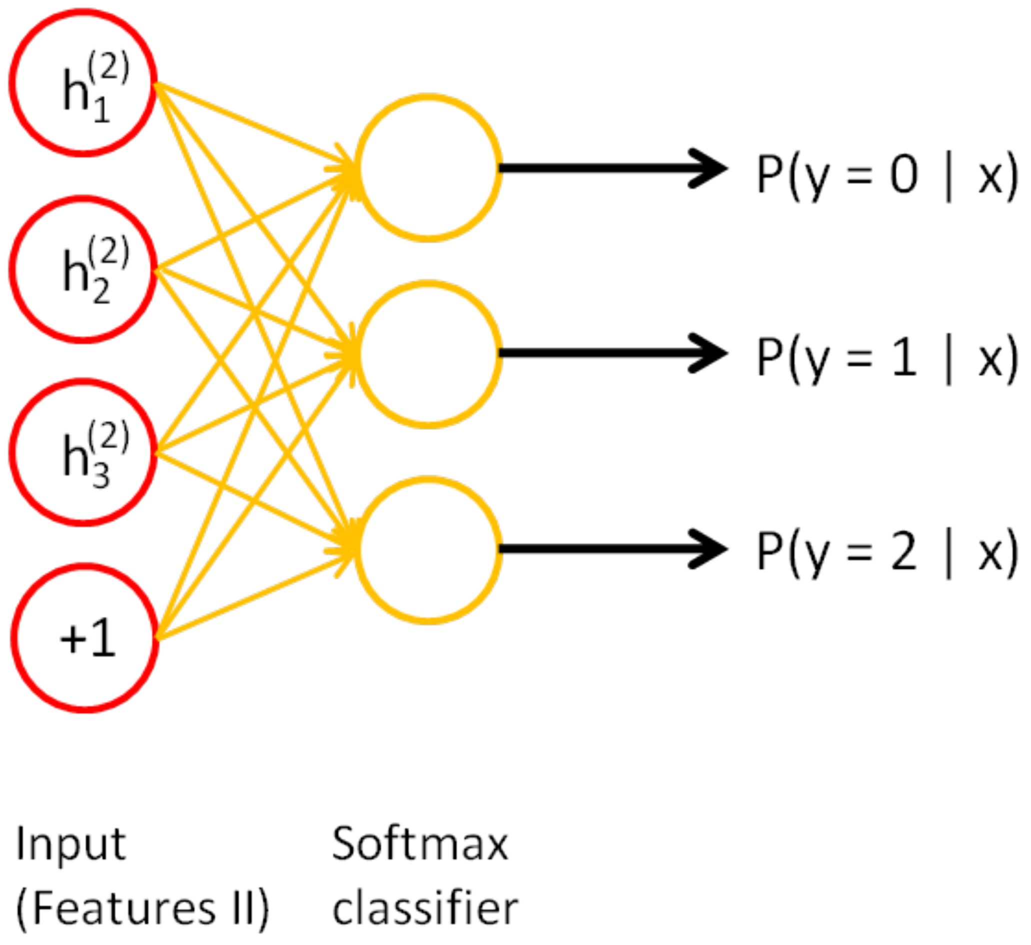

x_1
x_2
x_3
x_4
x_5
x_6
+1
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_4^{(1)}$
+1
$\hat{x}_1$
$\hat{x}_2$
$\hat{x}_3$
$\hat{x}_4$
$\hat{x}_5$
$\hat{x}_6$
Input
Features I
Output

Output Layer

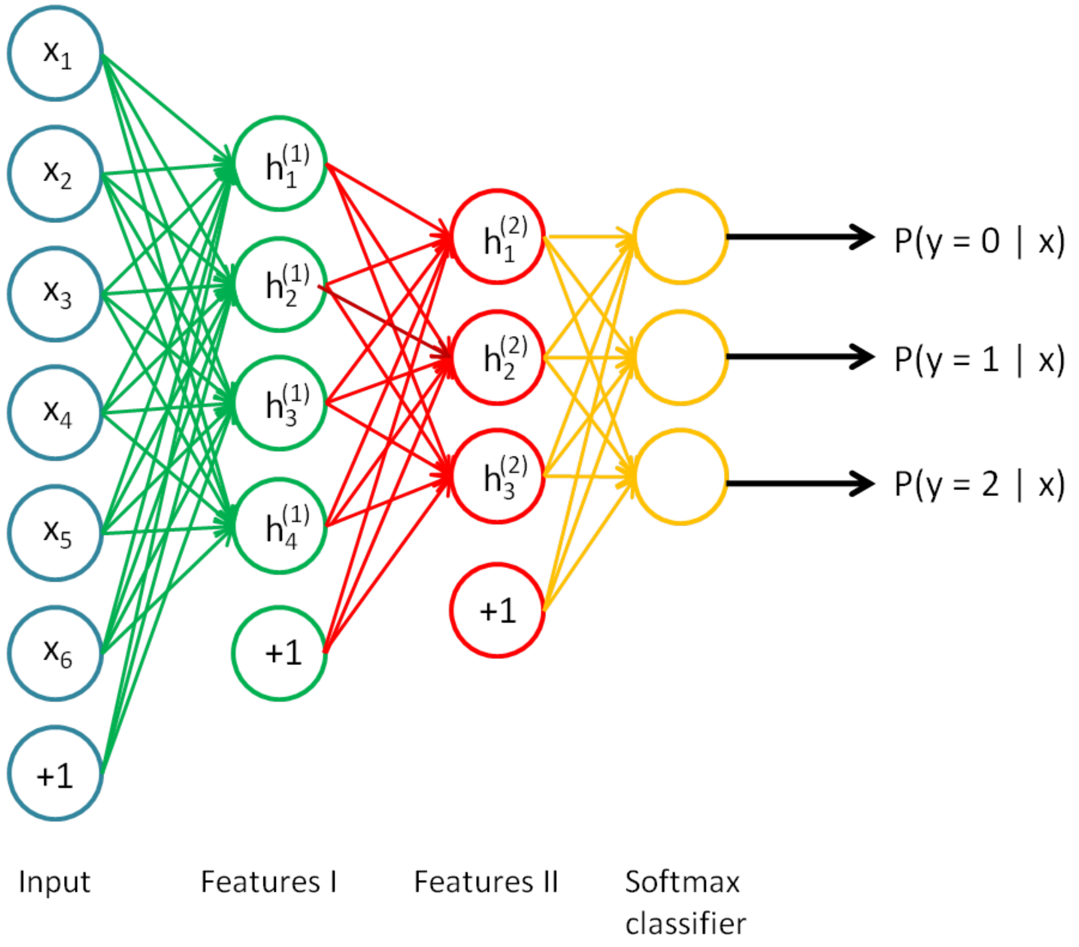


Input (Features I) Features II Output

Softmax Layer



Complete Deep NN



Stacked Autoencoder Benefits

- A stacked autoencoder enjoys all the benefits of any deep network of greater expressive power.

- Further, it often captures a useful "hierarchical grouping" or "part-whole decomposition" of the input.

Good Representation of its input

- recall that an autoencoder tends to learn features that form a good representation of its input.
- The first layer of a stacked autoencoder tends to learn first-order features in the raw input (such as edges in an image).
- The second layer of a stacked autoencoder tends to learn second-order features corresponding to patterns in the appearance of first-order features (e.g., in terms of what edges tend to occur together--for example, to form contour or corner detectors).
- Higher layers of the stacked autoencoder tend to learn even higher-order features.

Fine Tuning Again

- In order to compute the gradients for all the layers of the stacked autoencoder in each iteration, we use the Backpropagation Algorithm.

- As the backpropagation algorithm can be extended to apply for an arbitrary number of layers, we can actually use this algorithm on a stacked autoencoder of arbitrary depth.

Preprocessing

PCA

Principal Components Analysis (PCA) is a dimensionality reduction algorithm that can be used to significantly speed up your unsupervised feature learning algorithm.

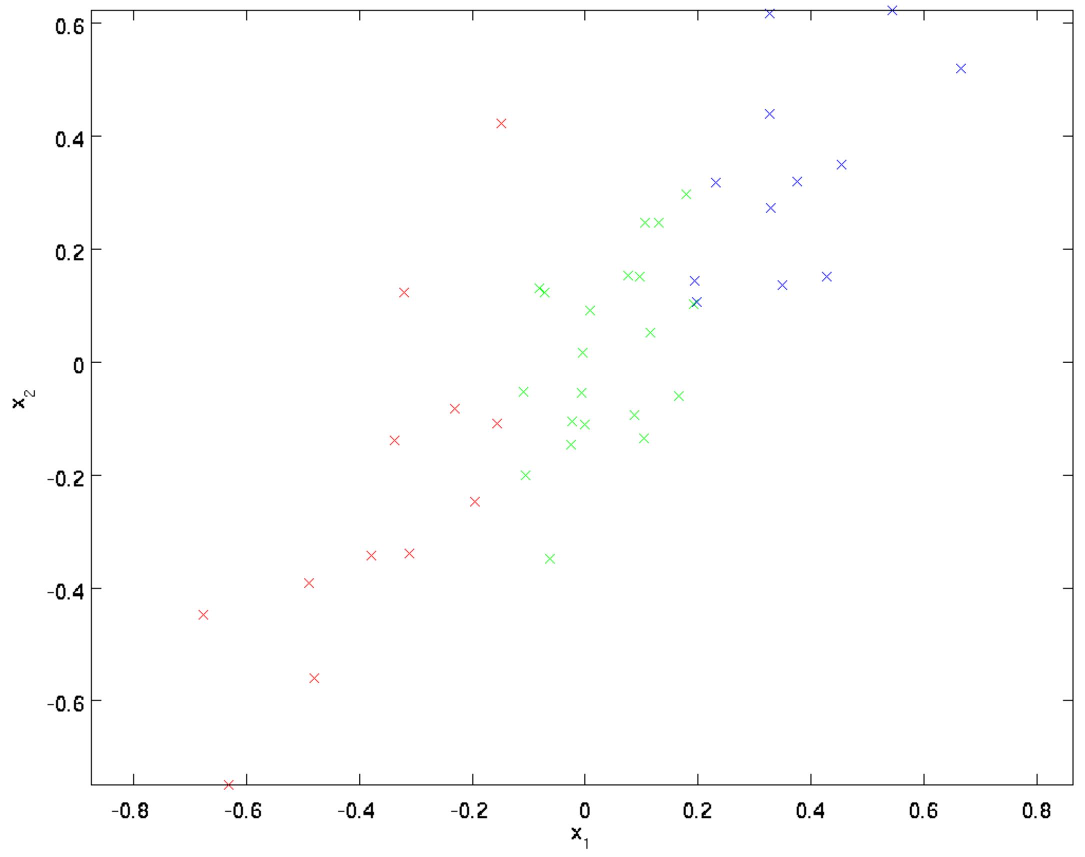
Used commonly for images.

The input is redundant, because the values of adjacent pixels in an image are highly correlated.

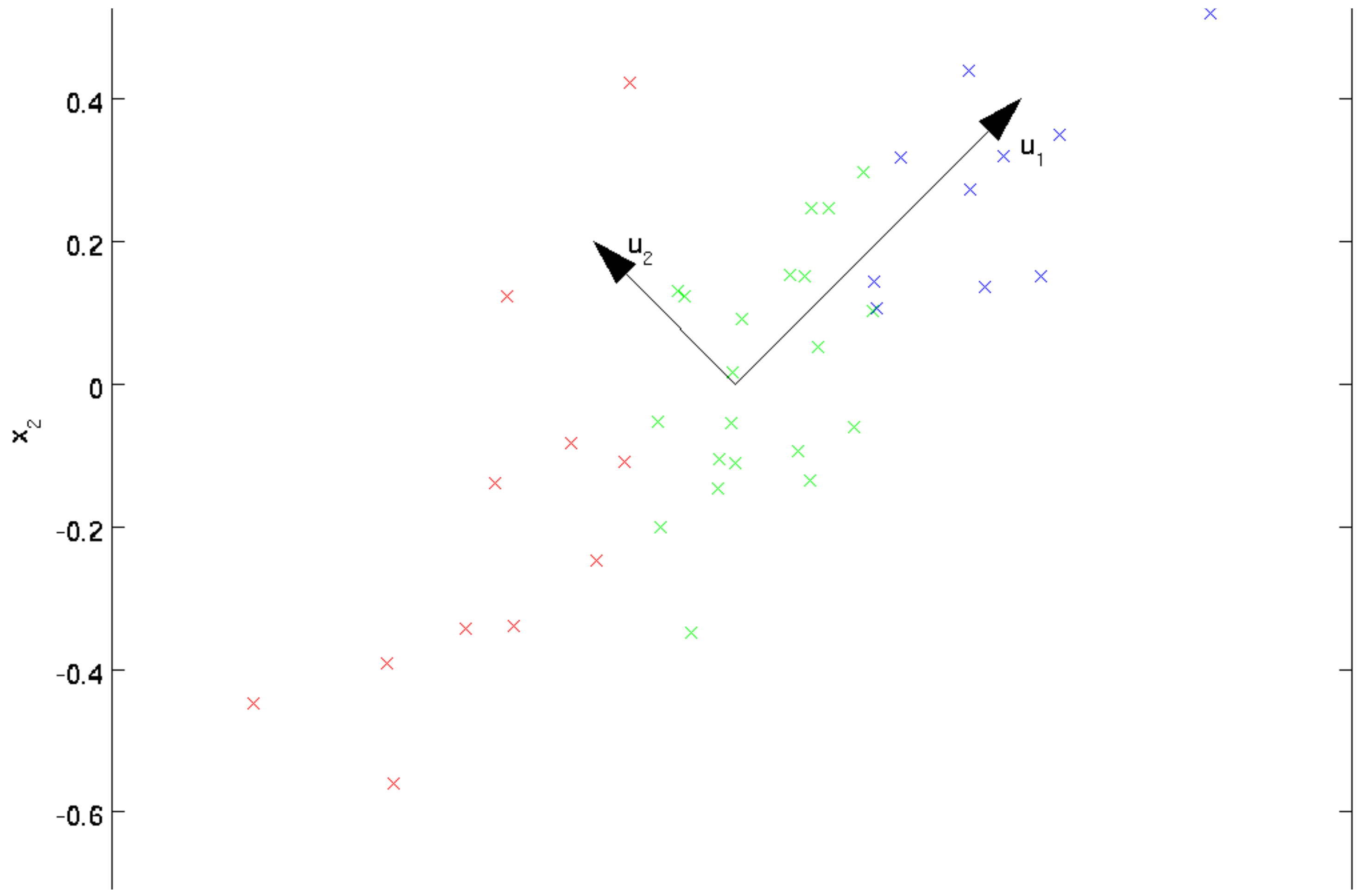
16x16 grayscale images are 256 dimensional vectors, with one feature corresponding to the intensity of each pixel.

Because of the correlation between adjacent pixels, PCA will approximate the input with a much lower dimensional one, while incurring very little error.

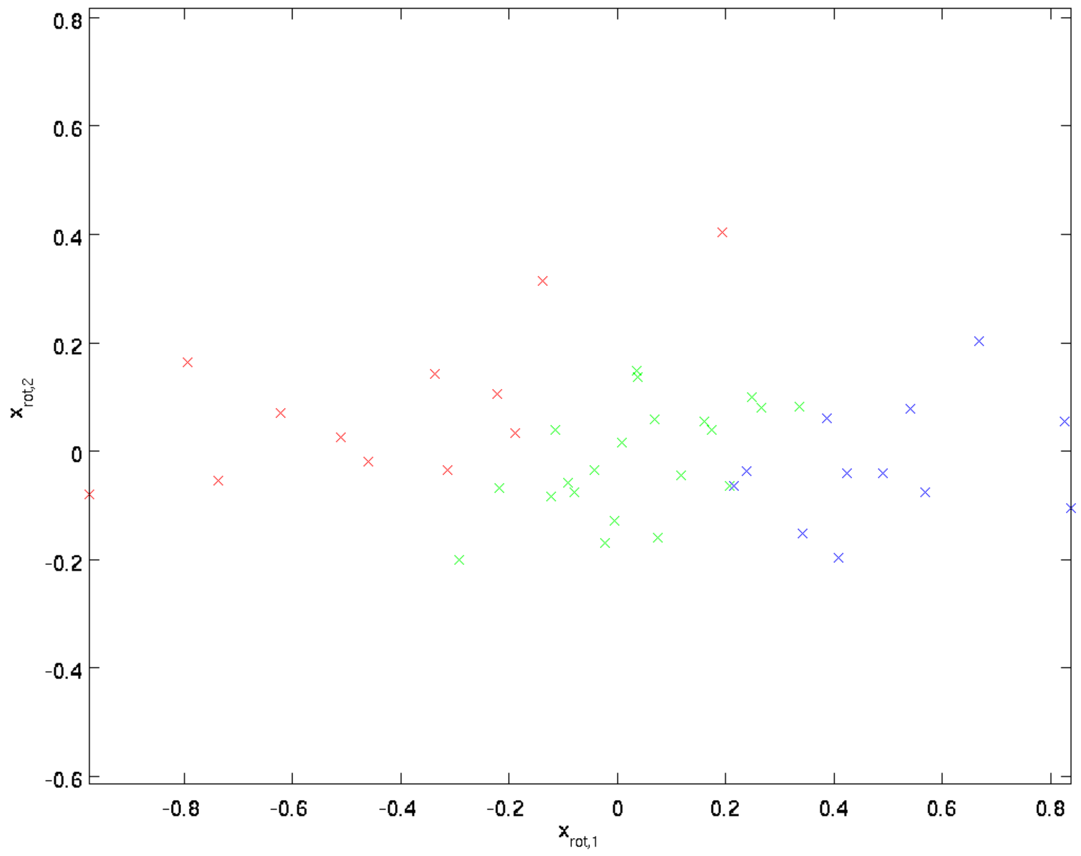
Original Data



First two eigenvalues



Rotated into the $u_1 u_2$ basis



PCA Advantages

The running time of your algorithm will depend on the dimension of the input.

Training on a lower-dimensional input, your algorithm might run significantly faster.

For many datasets, the lower dimensional representation can be an extremely good approximation to the original, and using PCA this way can significantly speed up your algorithm while introducing very little approximation error.

Whitening

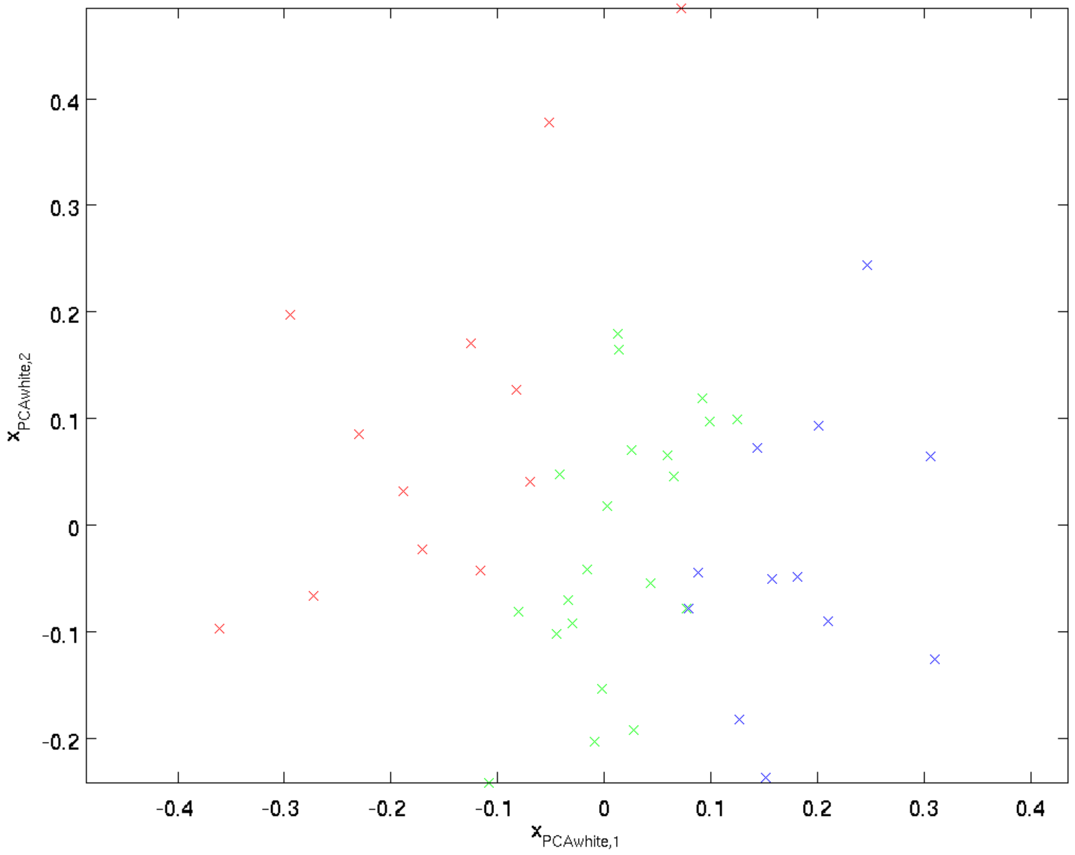
The goal of whitening is to make the input less redundant;

the features are less correlated with each other, and

the features all have the same variance.

If the variables are uncorrelated, the covariance matrix is diagonal. If they are all further standardized to unit variance, the covariance matrix equals the identity matrix

Now with Whitening



Softmax regression model

- generalizes logistic regression to classification problems where the class label y can take on more than two possible values.

- Softmax regression is a supervised learning algorithm

Softmax Regression vs. k Binary Classifiers

- Suppose you have a music classification application, and there are k types of music that you are trying to recognize. Should you use a softmax classifier, or should you build k separate binary classifiers using logistic regression?

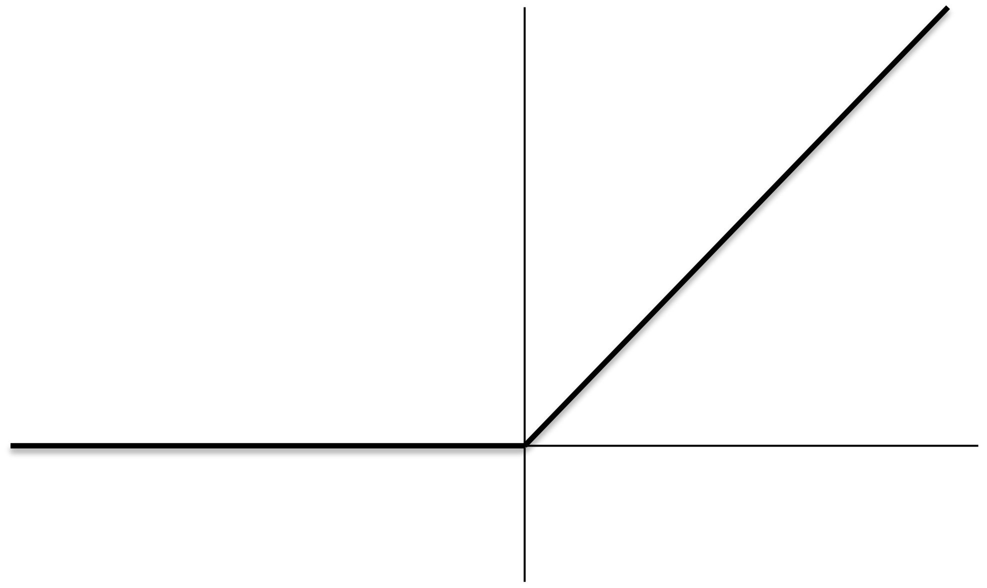
- Are the four classes are mutually exclusive? If your four classes are classical, country, rock, and jazz, then assuming each of your training examples is labeled with exactly one of these four class labels, you should build a softmax classifier with k = 4.

  (Do I agree with this????)

- If however your categories are has_vocals, dance, soundtrack, pop, then the classes are not mutually exclusive; for example, there can be a piece of pop music that comes from a soundtrack and in addition has vocals. In this case, it would be more appropriate to build 4 binary logistic regression classifiers. This way, your algorithm can separately decide whether it falls into each of the four categories.

#4: Deep learning 2.0 (~2010)

- Problem with sigmoid is that the gradients get very small at either end, so gradient descent becomes slow – *vanishing gradients problem*
- Enter *rectified linear*:



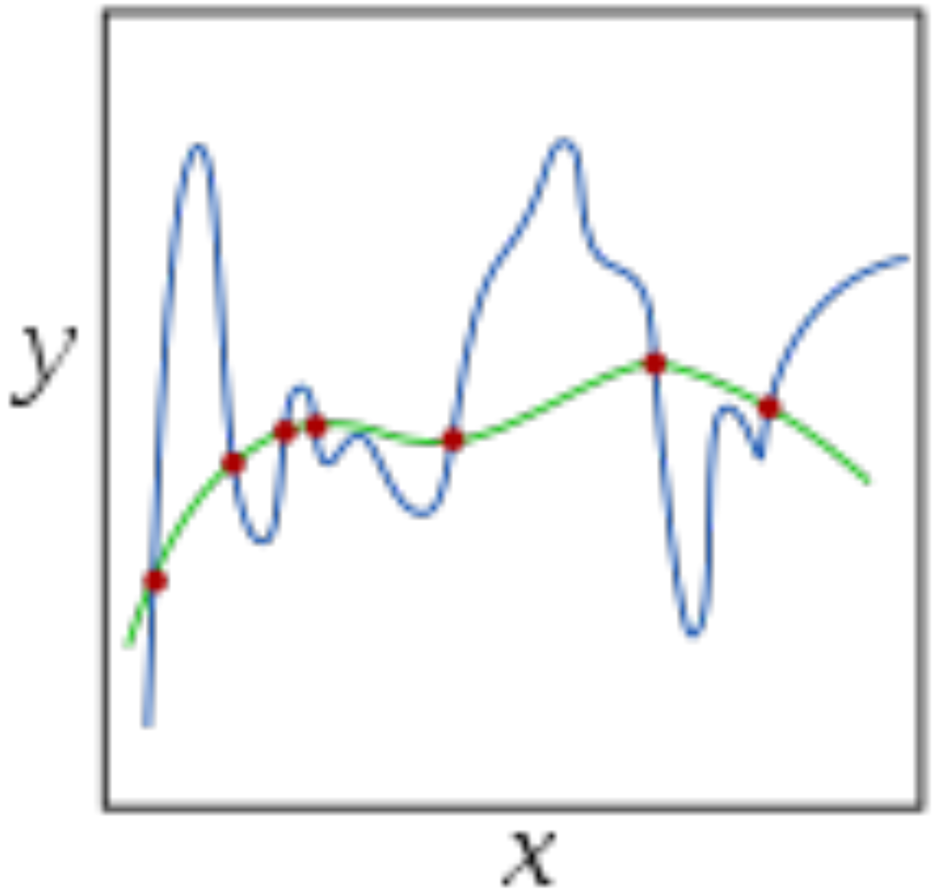
Rectified linear activation function

- No vanishing gradients on right hand side
- Allows deep networks to be trained with gradient descent without using pre-training

Dropout

- Randomly remove units when training
- Acts as a *regulariser* – gives better performance on validation and test
- Why does it work? Maybe something to do with units not relying on other units, so learning more robust features

Regularization example



By Nicoguaro - Own work, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=46259145

Practical aspects of training a deep neural network

Processor

- Within each layer, lots of calculations can be done independently (one for each unit in the next layer)
- Lends itself to parallelisation…
- Use GPUs!
- Coming soon, NPUs! Watch this space.

Validation set to avoid overfitting

- Split data into training, validation and test
- Train network on training set
- Monitor error on validation set. If it starts increasing, then stop training, because we are overfitting to the training set
- However, need to let it run for a bit because validation error can go upwards in the short-term but trend downwards in the long-term

Choosing hyperparameters

- Hyperparameters include initial learning rate, momentum, weight decay…
- Can also use validation set to choose these hyperparameters
- Try different values and look at which gives best performance on validation set after training, while also using the validation set for early stopping

Multiple runs

- Parameters are randomly initialised
- This means that the parameters can end up in a different location in parameter space
- If you are doing comparison between networks, it is good practice to do multiple runs to capture the variance in the final test error
- (However in practice, large networks take so long to train that they only get trained once)

Data augmentation

- More training data = more accurate classifier
- We know we can do certain transformations to examples and retain the same class e.g. a handwritten digit can be skewed slightly and still be the same digit
- So can artificially generate more training examples. This is called *data augmentation.*

The unreasonable effectiveness of deep learning

The power of deep learning

- Broke records by a long way on many image and speech datasets (still hold records)
- Hand-engineered features which took decades to develop have been made redundant
- All this from simply increasing the depth of the network
- Has led to some people labelling deep learning “unreasonably” effective

Re-using representation

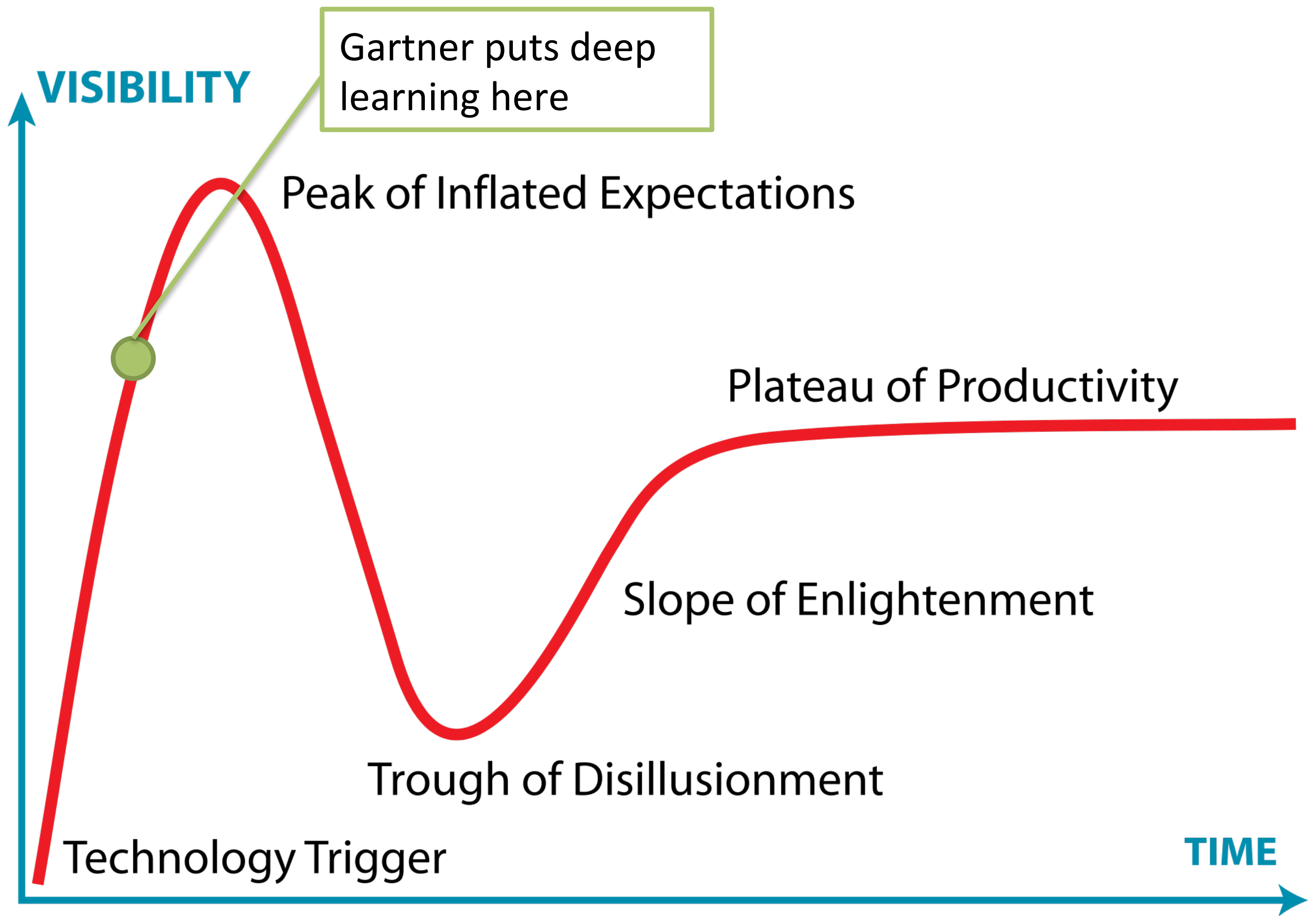
- Why so effective? Because complex features tend to share component features
- So multiple features can use the same feature from the previous layer
- Some theoretical results indicate that some functions that require an exponential number of units in a single-hidden-layer network only require a polynomial number of units in a two-hidden-layer network

Over-hype

Mainstream interest

- Deep learning has received high-profile mainstream press coverage
- Often hailed as a promising step towards strong AI
- Lots of attention from big tech companies - Google, Facebook and Baidu have all hired top experts from academia

Hype cycle


Gartner puts deep learning here
VISIBILITY
Peak of Inflated Expectations
Plateau of Productivity
Slope of Enlightenment
Trough of Disillusionment
Technology Trigger
TIME

Reality

- Over-hype is dangerous – has killed AI research in the past many times
- Is deep learning a silver bullet? No. Not for AI, not even for classification.
- We have a long, long way to go before we can achieve strong AI
- But deep learning has proved itself to be a highly effective method, so it is probably a step in the right direction

Questions you should be able to answer

- What makes deep learning different from old style neural networks?
- What is the difference between self taught versus semi-supervised neural networks?
- What is an autoencoder?
- Why did deep ANNs have trouble learning?
- How was pretraining used in deep learning?
- Why does a rectified linear activation function work better?

References

- Autoencoder - http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
- Rectified Linear Units – http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf
- Dropout - http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf
- DropConnect - http://proceedings.mlr.press/v28/wan13.html