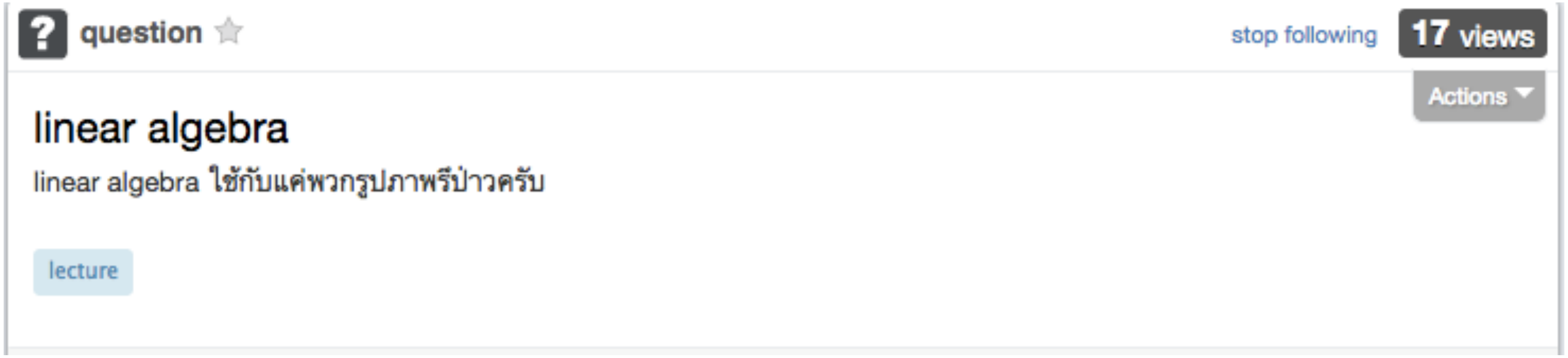
NEURAL NETWORKS

Loose ends from HW3



question

stop following

17 views

Actions

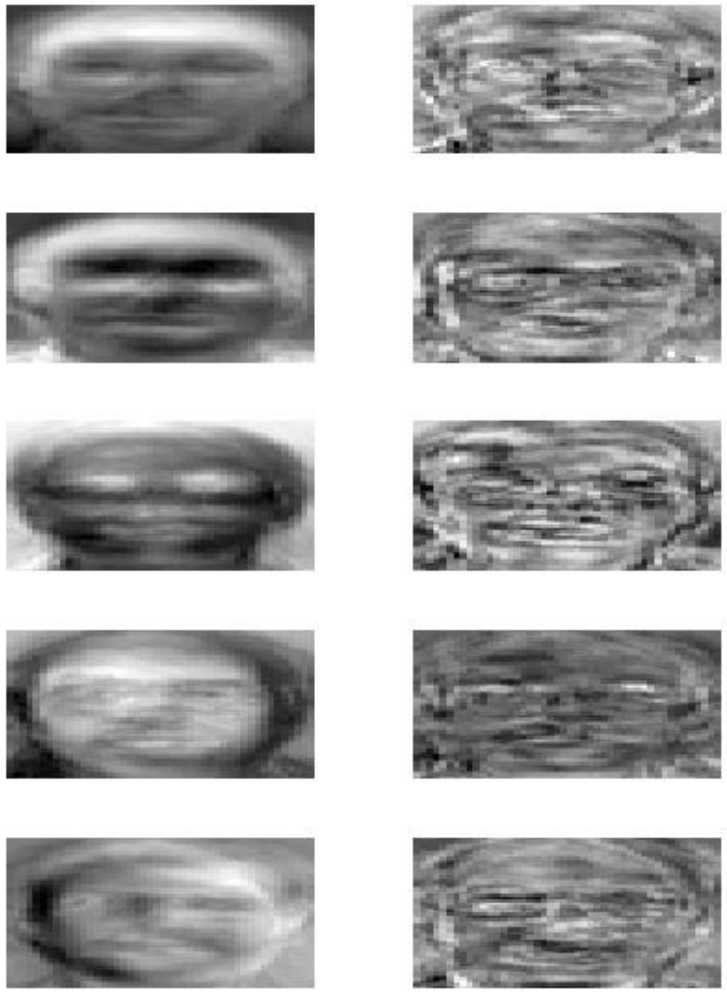
linear algebra

linear algebra ใช้กับแค่พวกรูปภาพรึป่าวครับ

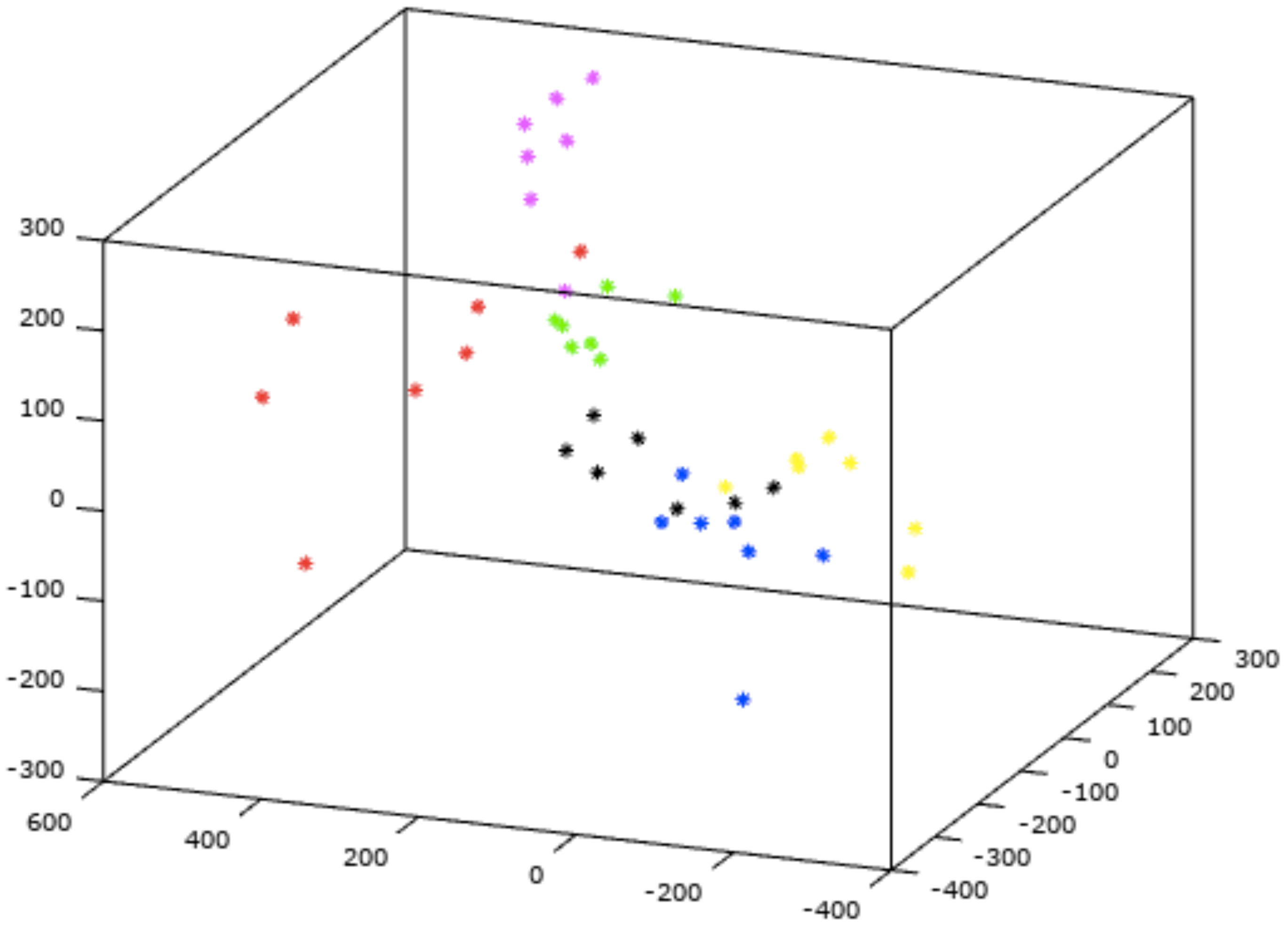
lecture

- PCA as a dimensionality reduction tool (compression)
- PCA/LDA as a feature analysis tool
- PCA/LDA as a visualization tool
- PCA as a feature normalization technique

PCA/LDA as a feature analysis tool



LDA as a visualization tool



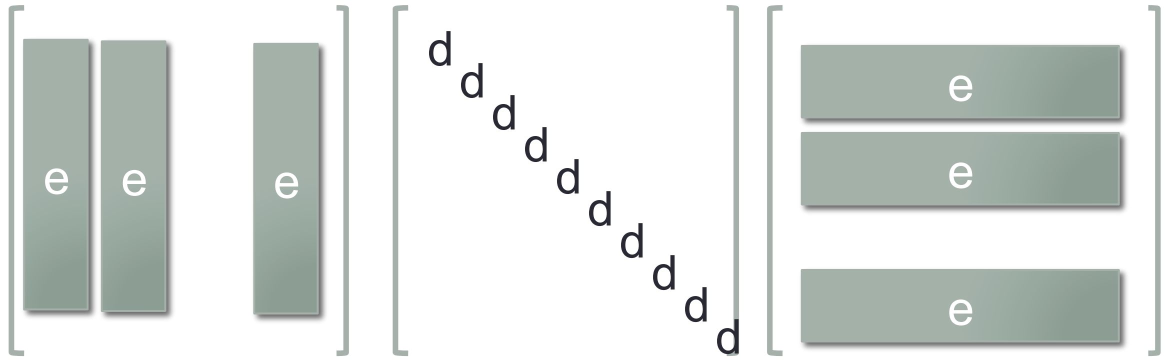
PCA as a feature normalization technique

- We said it's good to normalize features to [0,1],[-1,1], N(0,1).
  - Normalize each dimension independently
- Can we do better?

Whitening (PCA)

- Find the project along the dimensions that has the highest variance in the data
- Let $\sum$ be the covariance matrix. E is the matrix of eigen vectors, and D has eigen values along the diagonal. With eignenvalue decomposition:

$$\Sigma = EDE'$$



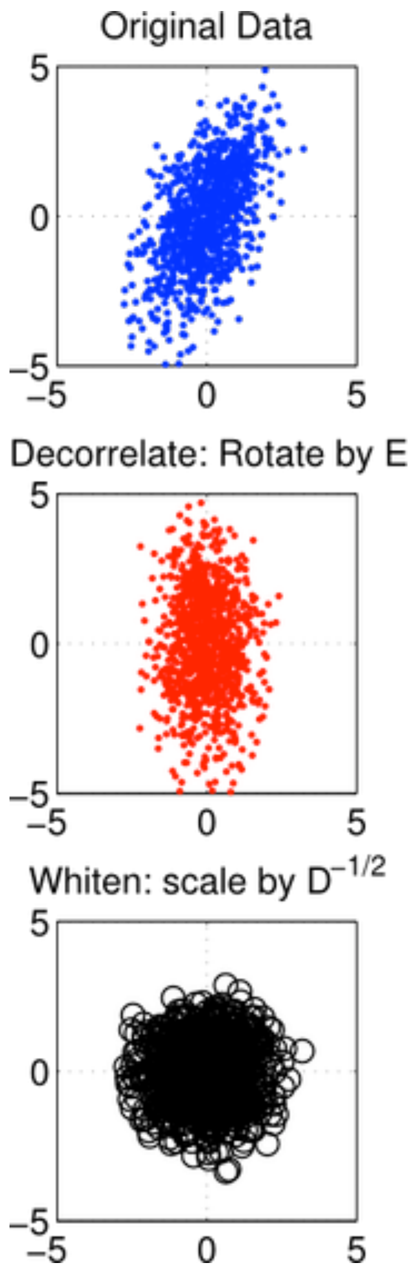
Whitening (PCA)

- Whitening decorrelates and scale

$$Y = D^{-1/2} E' X$$

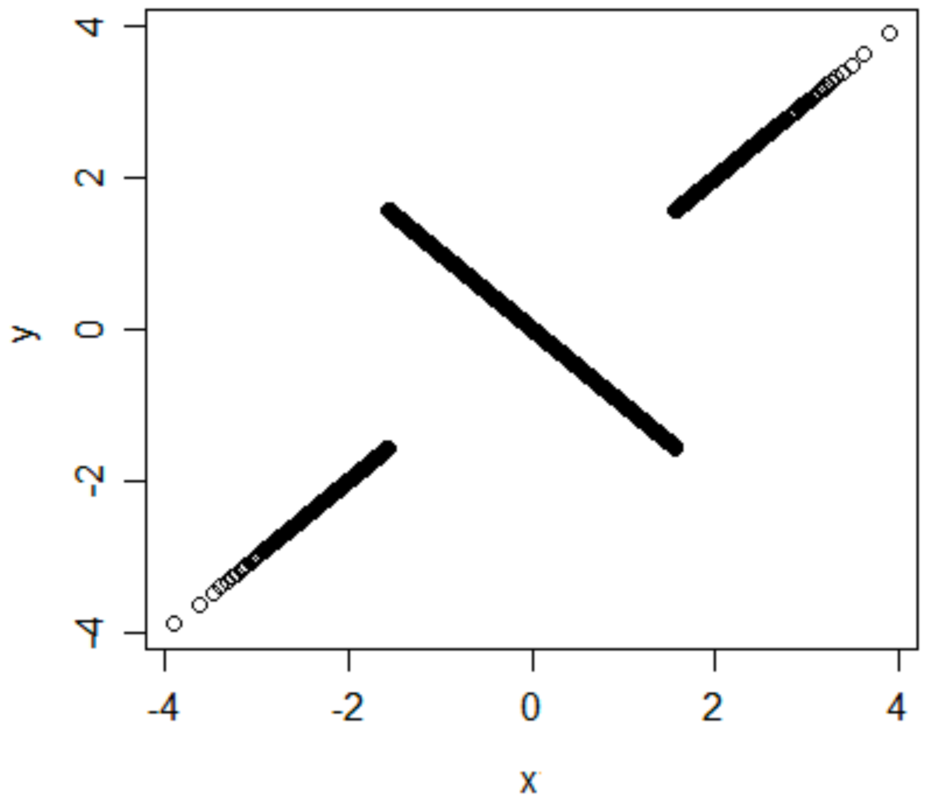
- In homework we only use the decorrelates part (rotation)
- Some models prefer features to be of equal variance (SVMs, Neural networks)
- Scale according to the inverse of the variance.
- This decorrelates the featuers (on the global scale)
  - Correlations can still exist given class
  - Uncorrelated-ness does not imply independence
    - We usually assume so when working with real data though

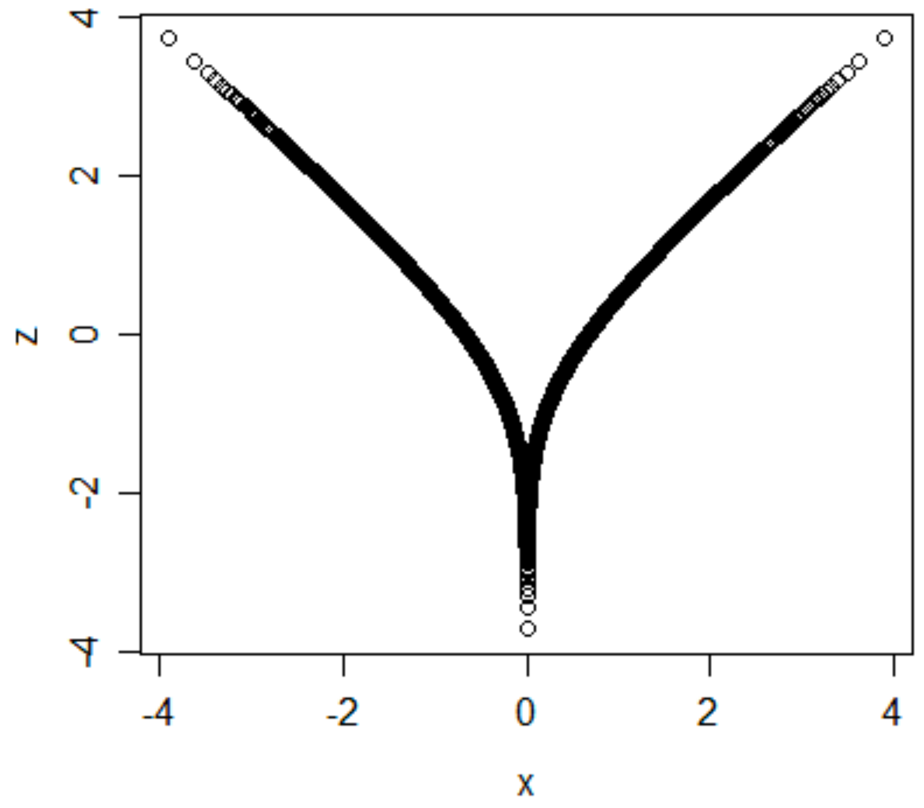
https://theclevermachine.wordpress.com/2013/03/30/the-statistical-whitening-transform/



Uncorrelated but dependence

- Below are example of variables with 0 correlation but definitely not independent





Whitening (PCA)

- What is the covariance matrix of data rotated by PCA?

- What is the covariance matrix of data whiten by PCA?

GMM/Gaussian fitting

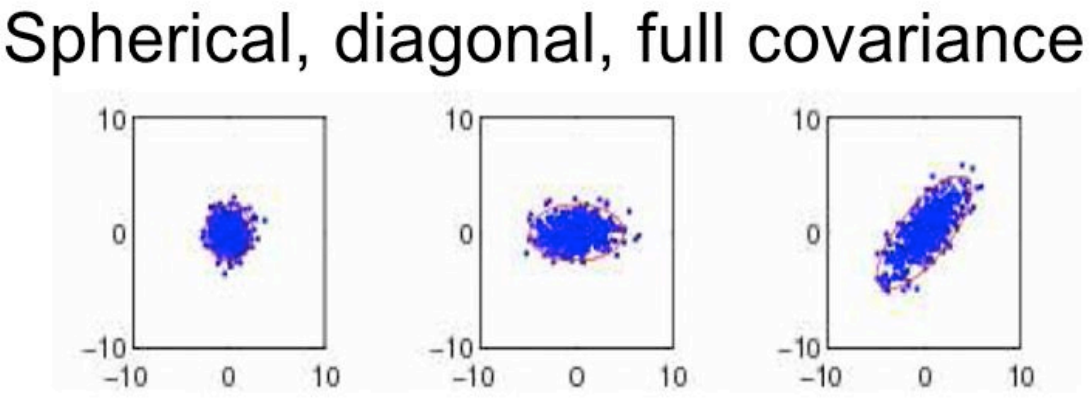
- How many parameters are there in a 2x2 covariance matrix?
- How many data points do you need to estimate a 2v2 covariance matrix (at least)?

$$m_j = \frac{1}{N}\Sigma_n w_{n,j}$$

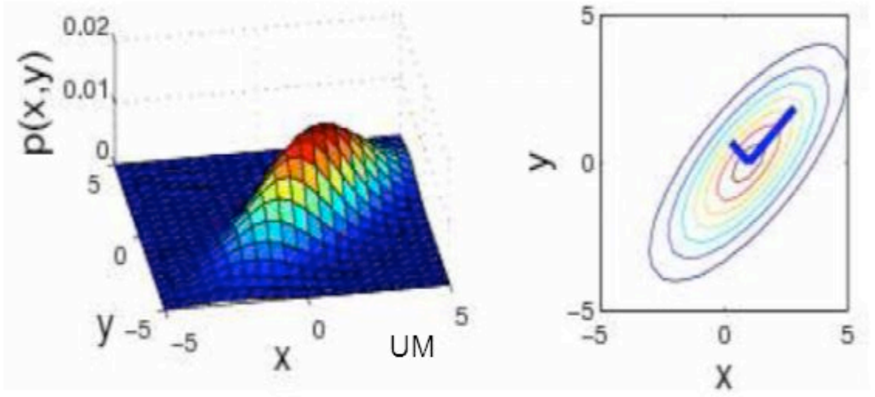
$$\vec{\mu_j} = \frac{\Sigma_n w_{n,j}\vec{x_n}}{\Sigma_n w_{n,j}}$$

$$\mathbf{\Sigma}_j = \frac{\Sigma_n w_{n,j}(\vec{x_n} - \vec{\mu_j})(\vec{x_n} - \vec{\mu_j})^T}{\Sigma_n w_{n,j}}$$

Many forms of covariance matrix



$$\Sigma = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$



http://slideplayer.com/slide/6258331/

Whitening and GMM fitting

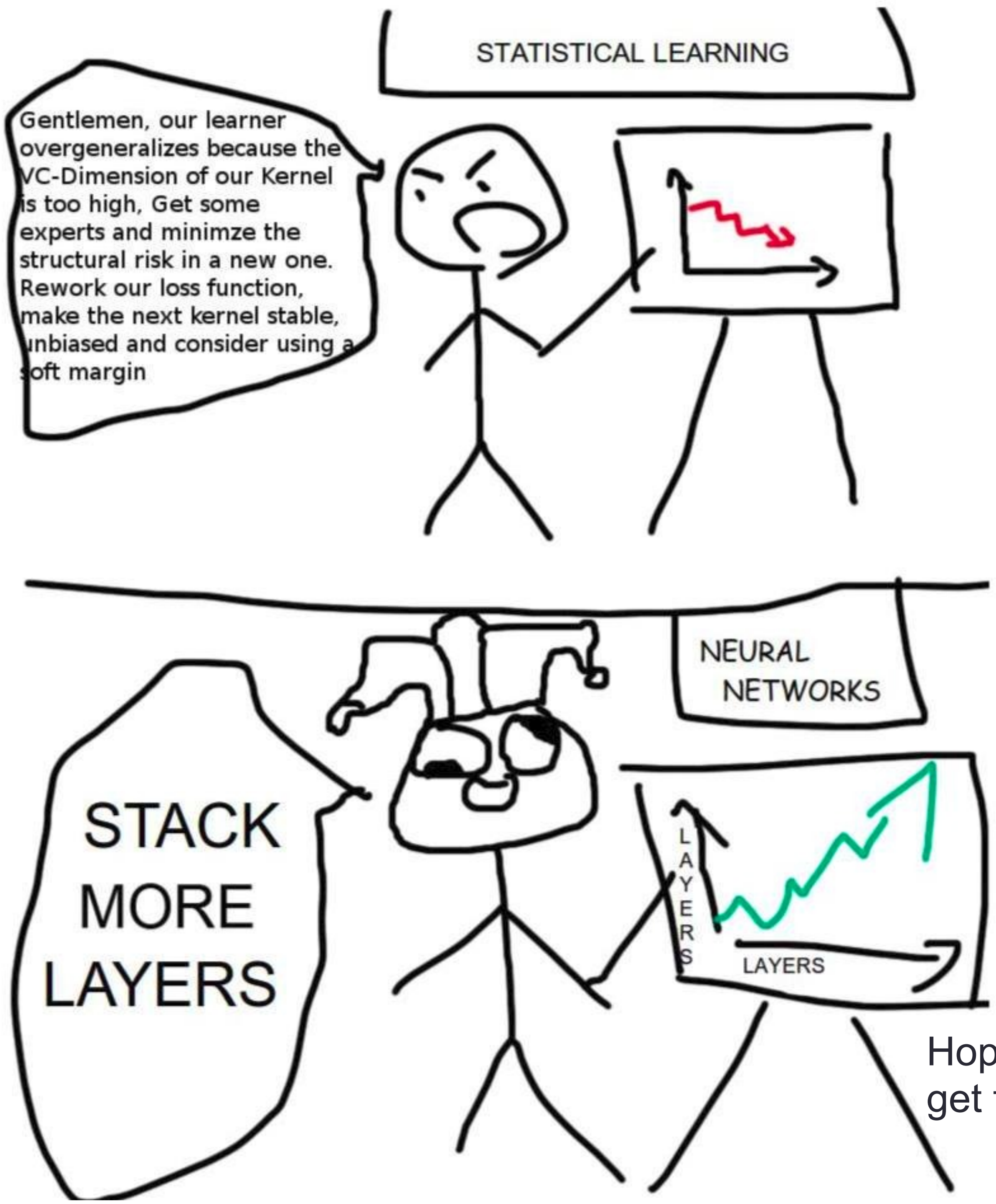
- Spherical/diagonal covariance are less prone to overfitting (less parameters)
- Data are not always distributed like that
- Use whitening to help make them spherical/diagonal distributed
    - Still not quite true, but oh well

NEURAL NETWORKS

Deep learning = Deep neural networks = neural networks

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

https://xkcd.com/1838/


STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS

Hopefully this won’t be all you get from this class

DNNs (Deep Neural Networks)

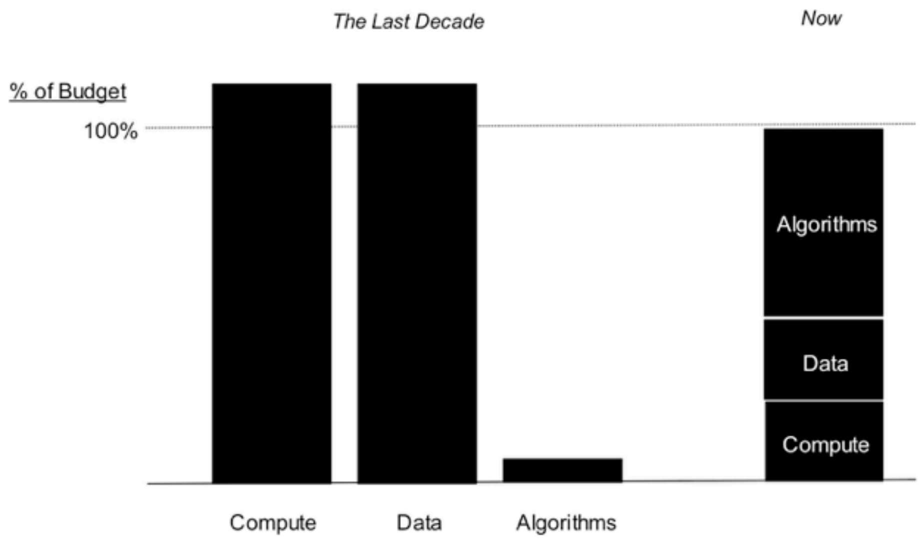
- Why deep learning?
- Greatly improved performance in ASR and other tasks (Computer Vision, Robotics, Machine Translation, NLP, etc.)
- Surpassed human performance in many tasks

Task	Previous state-of-the-art	Deep learning (2012)	Deep learning (2017)
TIMIT	24.4%	20.0%	17.0%
Switchboard	23.6%	16.1%	5.5%
Google voice search	16.0%	12.3%	4.9%

Geoffrey Hinton, Deep Neural Networks for Acoustic Modeling in Speech Recognition, 2012

Why now

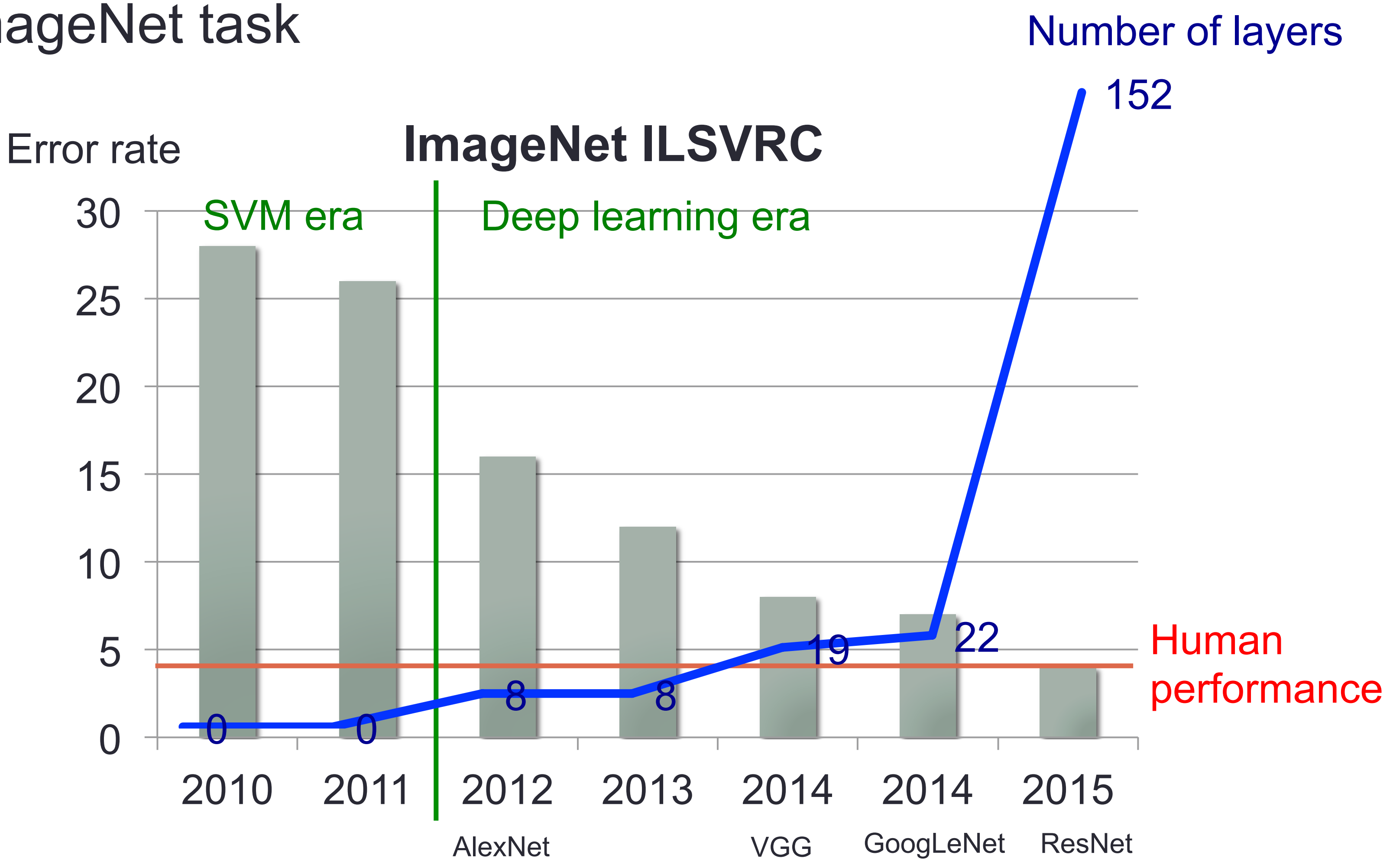
- Neural Networks has been around since 1990s
- Big data – DNN can take advantage of large amounts of data better than other models
- GPU – Enable training bigger models possible
- Deep – Easier to avoid bad local minima when the model is large



http://www.kdnuggets.com/2017/06/practical-guide-machine-learning-understand-differentiate-apply.html

Wider and deeper networks

- ImageNet task

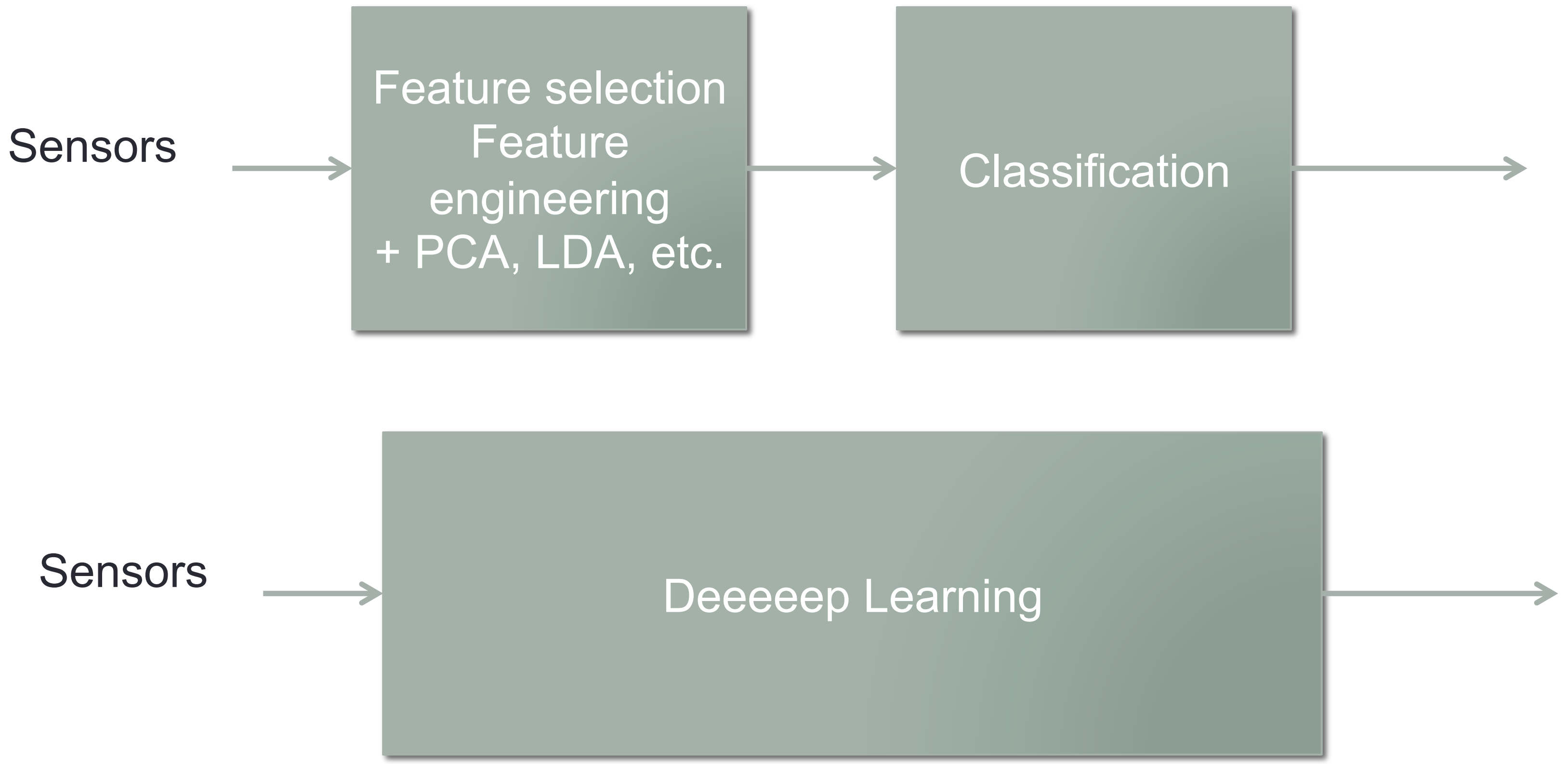


Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Why is deep learning good

- Traditional machine learning approaches need feature engineering
    - Features are based on human understanding of the phenomena
    - Have some simplifying assumptions
- Human knowledge captures known knowns, and Known unknowns NOT unknown unknowns
- Deep learning has enough parameters and model freedom to automatically learn the unknown unknowns
- **Deep learning combines features engineering and modeling into one single optimization problem**

Traditional VS Deep learning

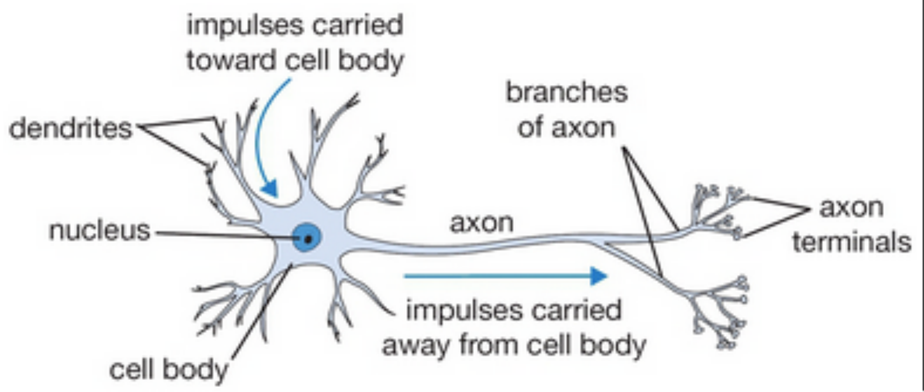


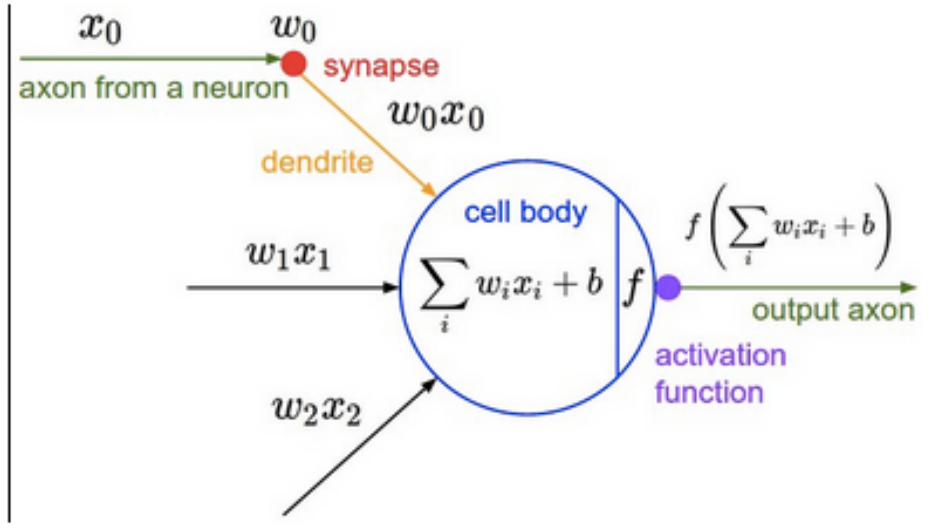
Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- t-SNE
- Demos
  - Tensorflow, Gcloud, Keras
- CNN, RNN, LSTM, GRU <- Next class

Fully connected networks

- Many names: feed forward networks or deep neural networks or multilayer perceptron or artificial neural networks
- Composed of multiple neurons

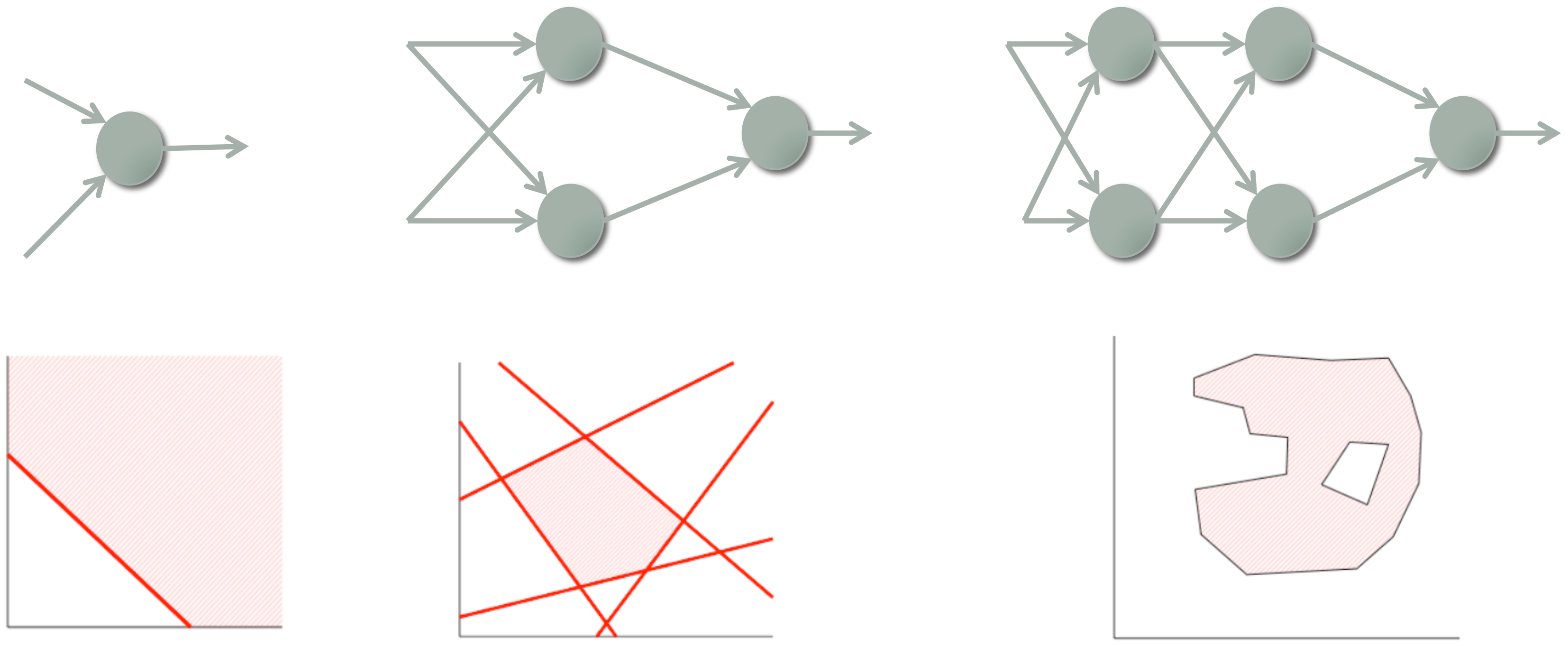

impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body



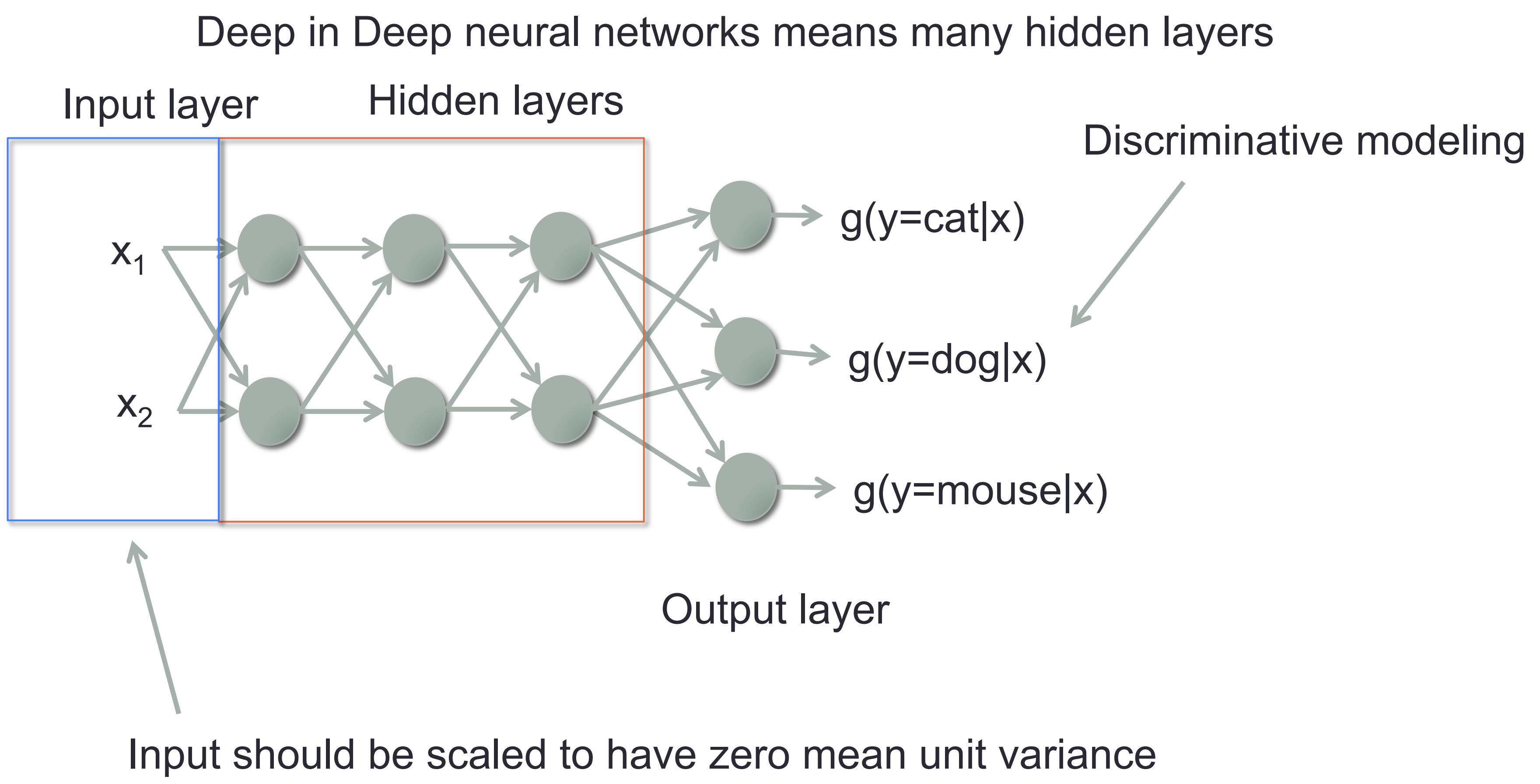
http://cs231n.github.io/neural-networks-1/

Combining neurons

- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting

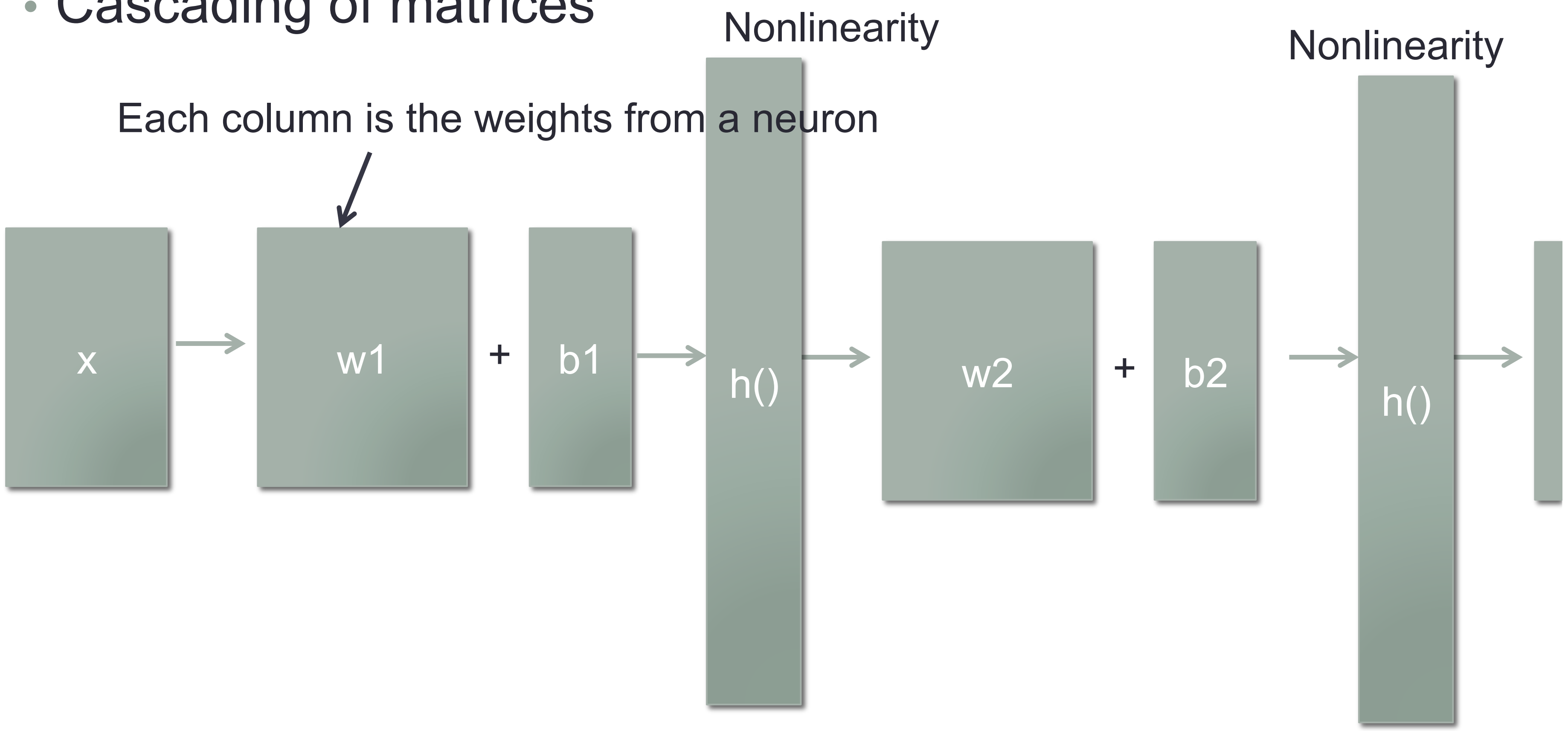


Terminology


Deep in Deep neural networks means many hidden layers
Input layer
Hidden layers
Discriminative modeling
x_1
x_2
g(y=cat|x)
g(y=dog|x)
g(y=mouse|x)
Output layer
Input should be scaled to have zero mean unit variance

More linear algebra

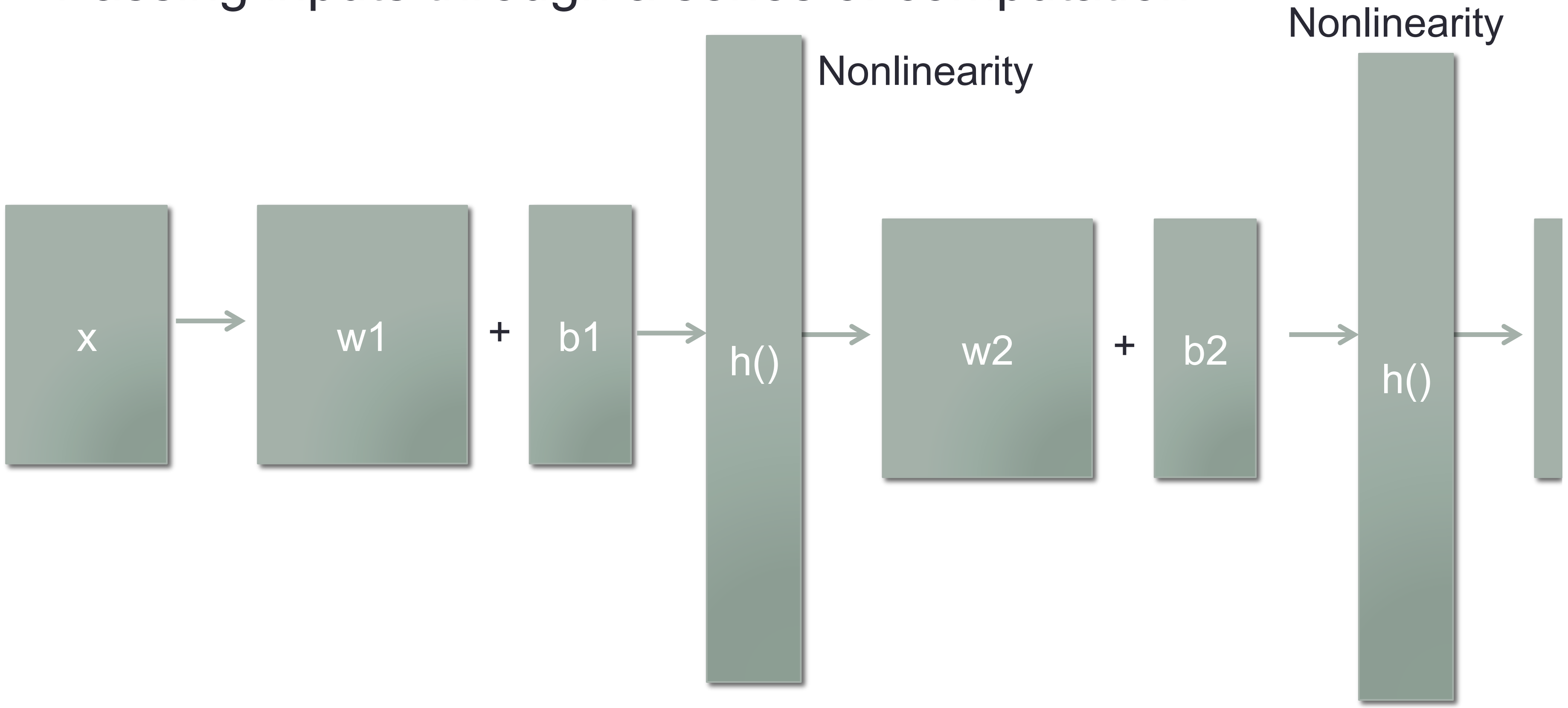
- Cascading of matrices



$h(W_2^T\ h(W_1^T X + \mathbf{b_1}) + \mathbf{b_2}\ )$

Computation graph

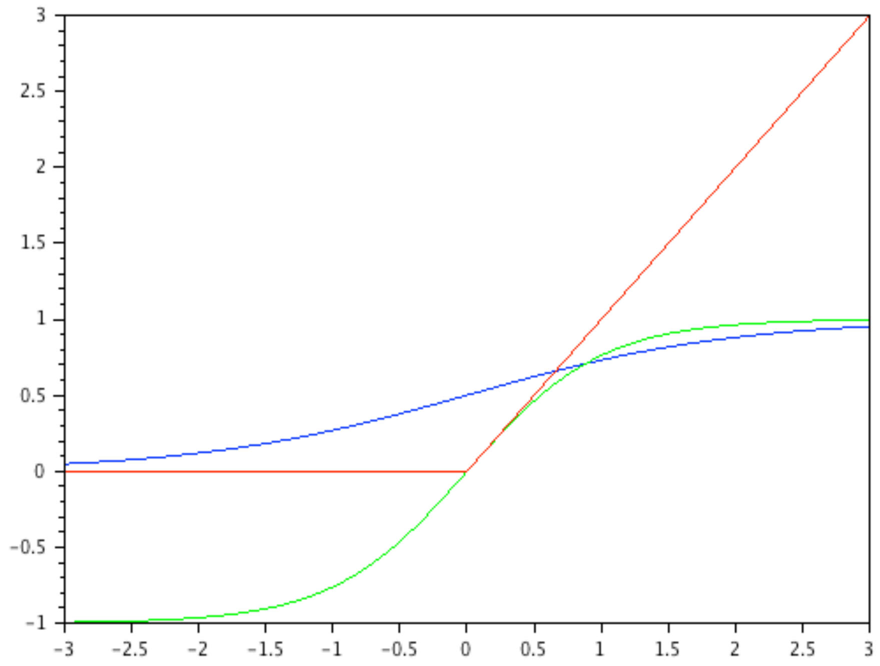
- Passing inputs through a series of computation



$h(W_2^T\, h(W_1^T X + \mathbf{b_1}) + \mathbf{b_2}\,)$

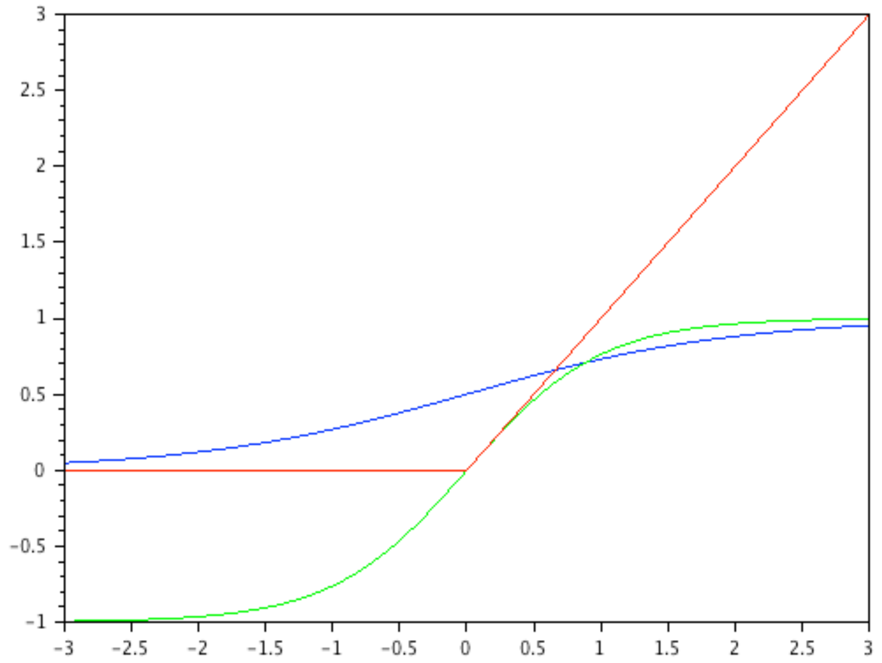
Non-linearity

- The Non-linearity is important in order to stack neurons
  - If F is linear, a multi layered network can be collapsed as a single layer (by just multiplying weights together)
- Sigmoid or logistic function
- tanh
- Rectified Linear Unit (ReLU)
- Most popular is ReLU and its variants (Fast to train, and more stable)



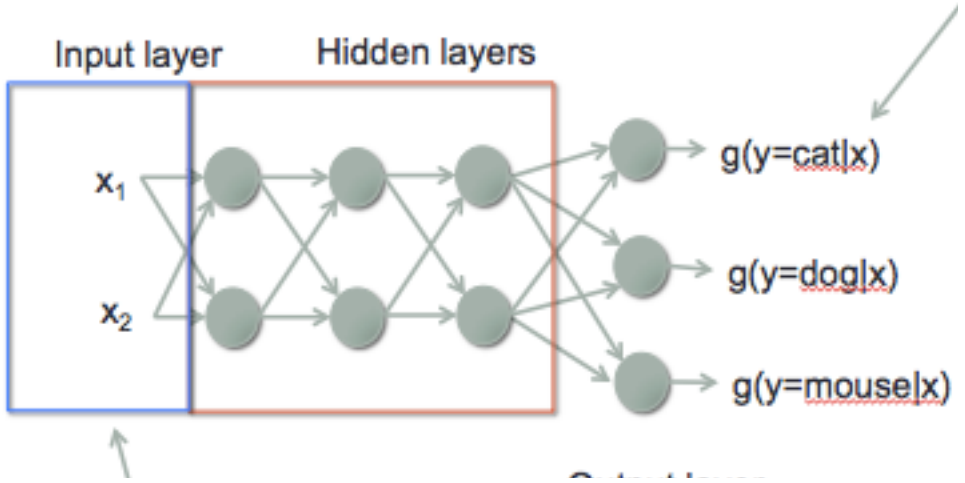
Non-linearity

- Sigmoid $$\frac{1}{1+e^{-x}}$$
- tanh $$tanh(x)$$
- Rectified Linear Unit (ReLU) $$max(0, x)$$



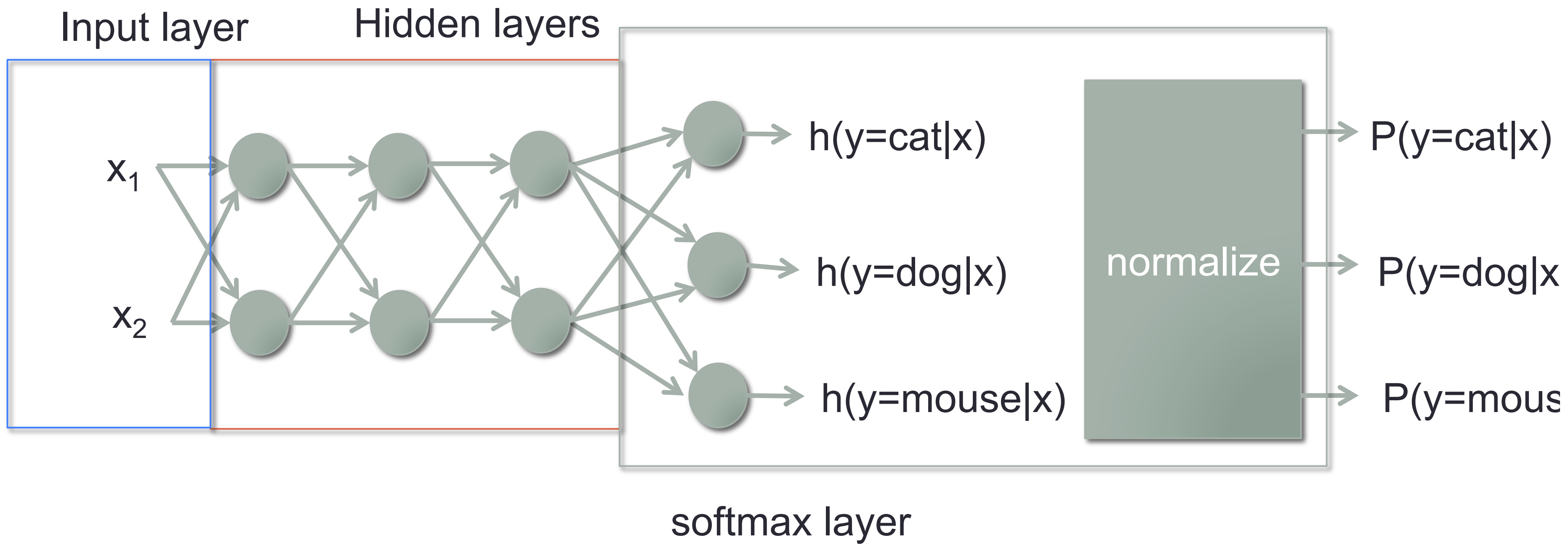
Output layer – Softmax layer

- We usually wants the output to mimic a probability function (0<=P<=1,sums to 1)
- Current setup has no such constraint
- The current output should have highest value for the correct class.
  - Value can be positive or negative number
- Takes the exponent
- Add a normalization



Softmax layer

$$P(y=j|x) = \frac{e^{h(y=j|x)}}{\Sigma_y e^{h(y|x)}}$$

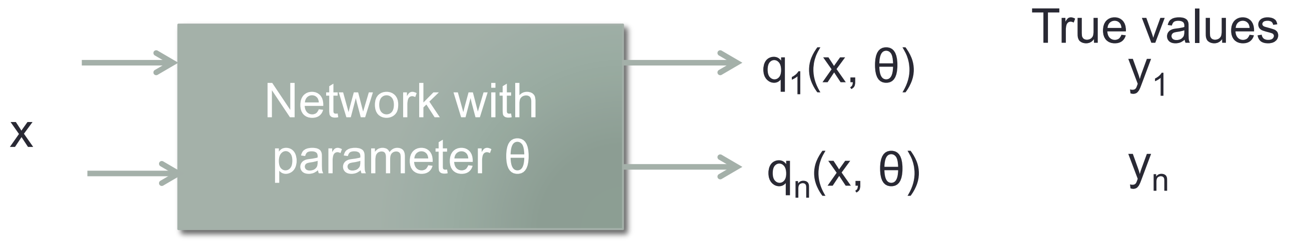


Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate and momentum
  - Overfitting – dropout, batchnorm
- t-SNE
- Demos
  - Tensorflow, Gcloud, Keras
- CNN, RNN, LSTM, GRU <- Next class

Objective function (Loss function)

- Can be any function that summarizes the performance into a single number
- Cross entropy
- Sum of square errors



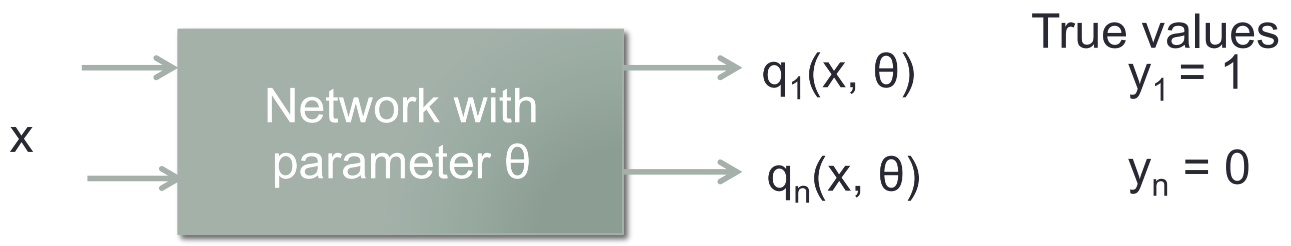
Cross entropy loss

- Used for softmax outputs (probabilities), or classification tasks

$$L = -\Sigma_n y_n log q_n(x, \theta)$$

- Where y_n is 1 if data x comes from class n
  0 otherwise

- L only has the term from the correct class
- L is non negative with highest value when the output matches the true values, a “loss” function

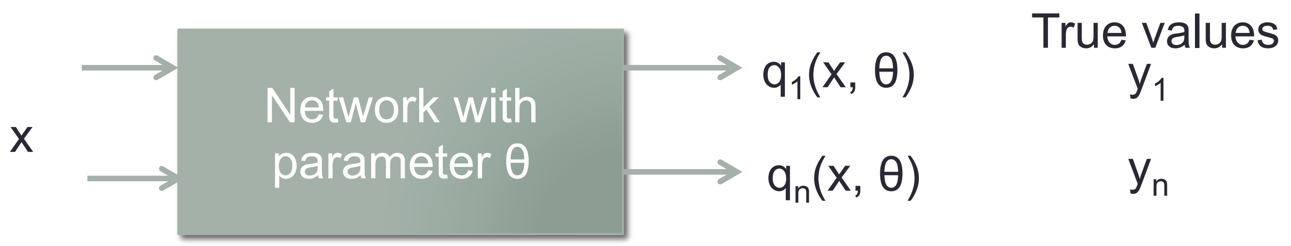


Sum of square errors

- Used for any real valued outputs such as regression

$$L = \frac{1}{2}\Sigma_n (y_n - q_n(x,\theta))^2$$

- Non negative, the better the lower the loss



Regularization terms

Regularization in one slide

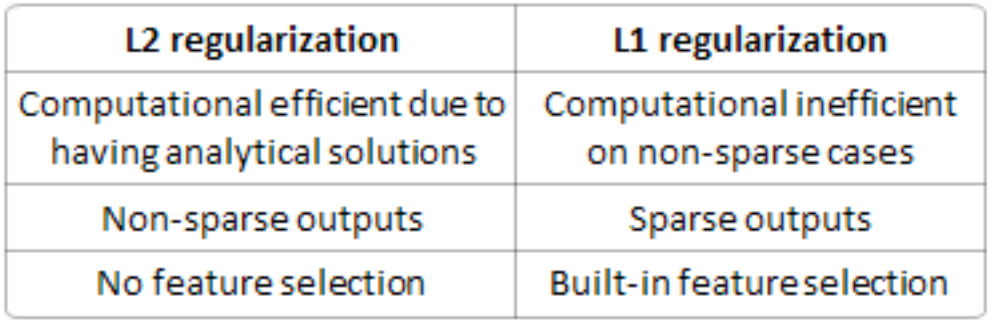
- What?
  - Regularization is a method to lower the model variance (and thereby increasing the model bias)
- Why?
  - Gives more generalizability (lower variance)
  - Better for lower amounts of data (reduce overfiting)
- How?
  - Introducing regularizing terms in the original loss function
    - Can be anything that make sense

$$\mathbf{w}^T\mathbf{w} + C\Sigma\varepsilon_i$$

MAP estimate is MLE with regularization (the prior term)

Famous types of regularization

- L1 regularization: Regularizing term is a sum
  - $\mathbf{w}^T\mathbf{w} + C\Sigma\varepsilon_i$
- L2 regularization: Regularizing term is a sum of squares
  - $\mathbf{w}^T\mathbf{w} + C\Sigma\varepsilon_i^2$



L2 regularization	**L1 regularization**
Computational efficient due to having analytical solutions	Computational inefficient on non-sparse cases
Non-sparse outputs	Sparse outputs
No feature selection	Built-in feature selection

http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/

Regularization in neural networks L2

- We want to improve generalization somehow.
- Observation, models are better when the weights are spread out (no peaky weights).
  - Try to use every part of the model.
- Add a cost if we put some value to the weights

- Regularized loss = Original loss + 0.5 $C\Sigma w^2$

- we sum the square of weights of the whole model
- 0.5 is for prettiness when we take derivative
- C is a hyperparameter weighting the regularization term

Regularization in neural networks

- We want to improve generalization somehow.
- Observation, models behave better when we force the weights to be sparse.
    - Sparse means many weights are zero or close to zero
    - Force the model to focus on only important parts
    - Less prone to noise
- Add a cost if we put some value to the weights
- Regularized loss = Original loss + $0.5\ C\Sigma|w|$

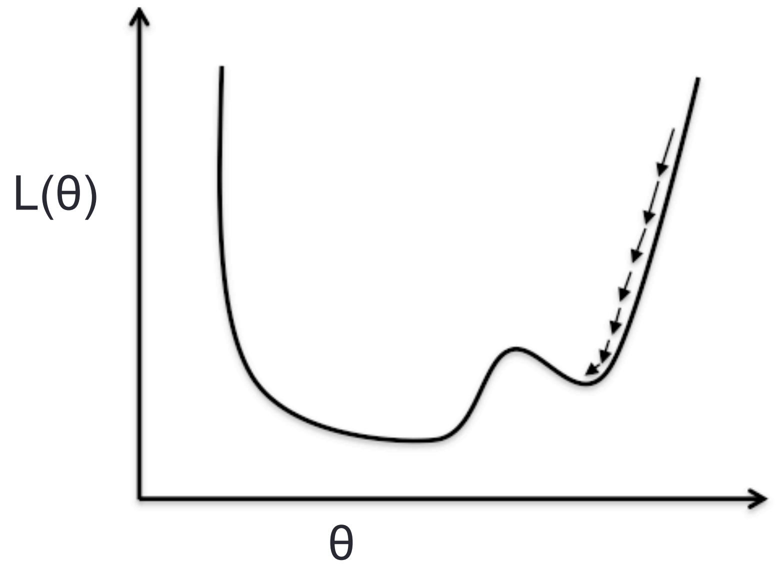
- we sum the absolute weights of the whole model
- 0.5 is for prettiness when we take derivative
- C is a hyperparameter weighting the regularization term

L1 L2 regularization notes

- Can use both at the same time
  - People claim L2 is superior
- I found them useless in practice for deep neural networks
  - Maybe there are some tasks out there that benefit from this? I don’t know
- Other regularization methods exist (we will go over these later)

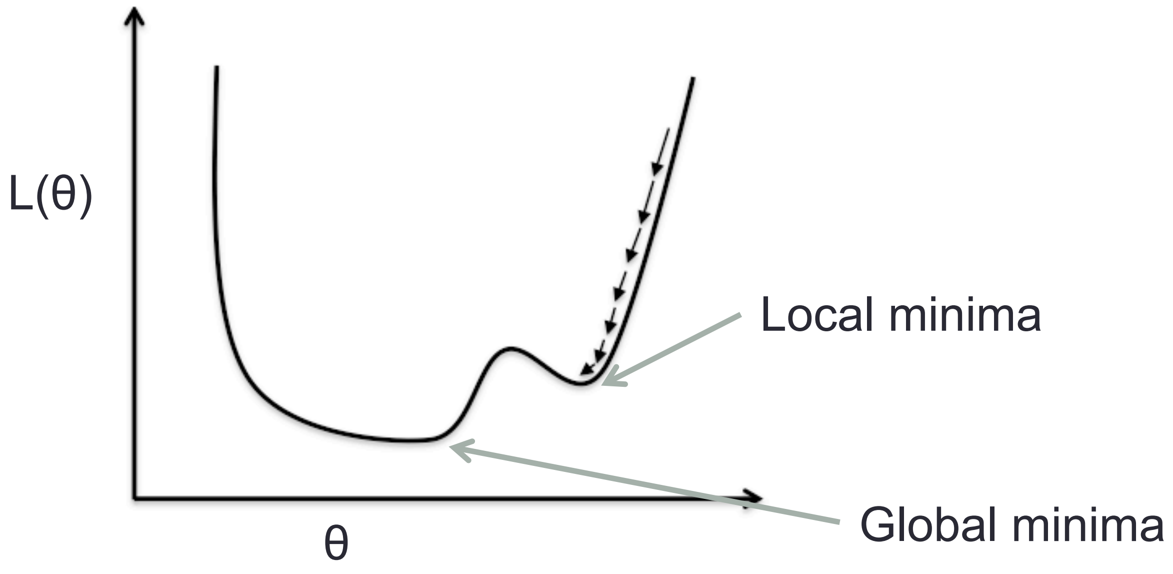
Minimization using gradient descent

- We want to minimize L with respect to θ (weights and biases)
  - Differentiate with respect to θ
  - Gradients passes through the network by Back Propagation



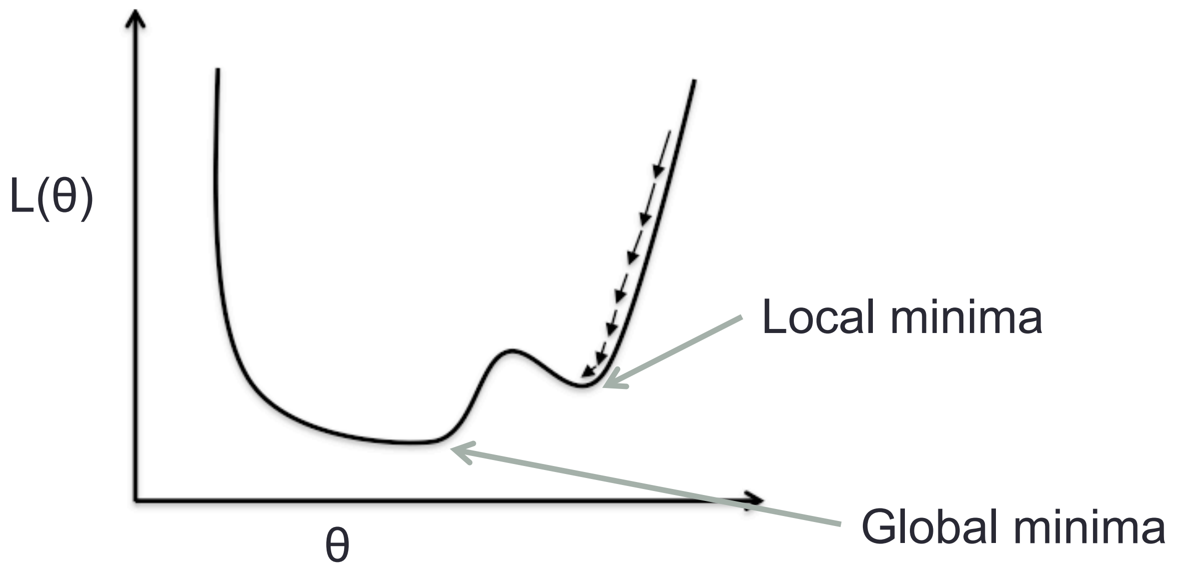
Deep vs Shallow

- The loss function of neural network is non-convex (and non-concave)
  - Local minimas can be avoided with convexity
    - Linear regression, SVM are convex optimization
  - Convexity gives easier training
    - Does not imply anything about the generalization of the model
    - The loss is optimized by the training set



Deep vs Shallow

- If deep, most local minimas are the global minima!
  - Always a way to lower the loss in the network with millions of paramters
  - Enough parameters to remember every training examples
  - Does not imply anything about generalization



Kenji Kawaguchi,Deep Learning without Poor Local Minima, 2016 https://arxiv.org/abs/1605.07110

Differentiating a neural network model

- We want to minimize loss by gradient descent
- A model is very complex and have many layers! How do we differentiate this!!?

Back propagation

- Forward pass
  - Pass the value of the input until the end of the network
- Backward pass
  - Compute the gradient starting from the end and passing down gradients using chain rule

Examples to read

https://alonalj.github.io/2016/12/10/What-is-Backpropagation/

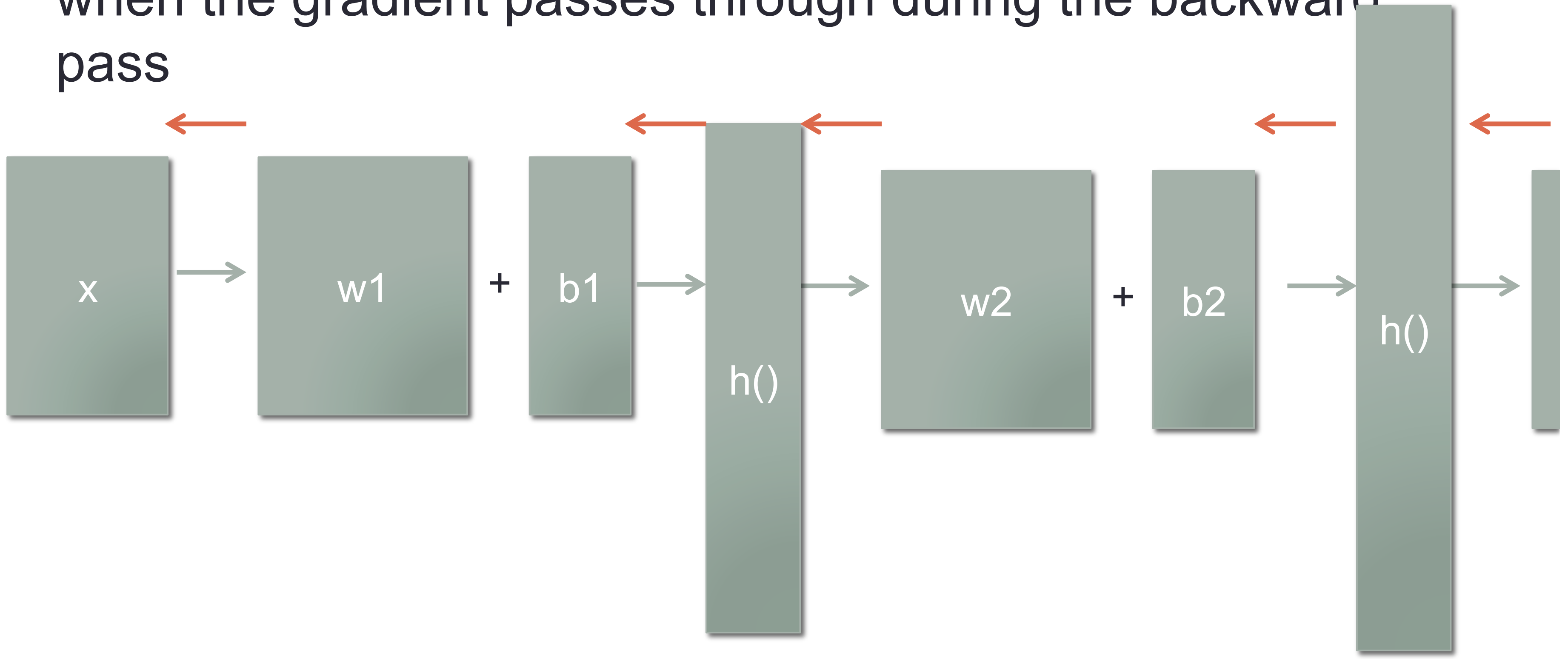
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Back propagation

- Regularization terms only appears at the particular weights when doing the derivative
- What about cross entropy

Backprop and computation graph

- We can also define what happens to a computing graph when the gradient passes through during the backward pass



This lets us to build any neural networks without having to redo all the derivation as long as we define a forward and backward computation for the block.

Initialization

- The starting point of your descent
- Important due to local minimas
- Not as important with large networks AND big data (>100 hours for ASR)
- Now usually initialized randomly
  - One strategy

$$W \sim \text{Uniform}(0, \frac{1}{\sqrt{\text{FanIn} + \text{FanOut}}})$$

  - For ReLUs

```
w = np.random.randn(n) * sqrt(2.0/n)
```

  - Or use a pre-trained network as initialization

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010

Initialization

- Bias
  - All Zeros is fine

Stochastic gradient descent (SGD)

- Consider you have one million training examples
  - Gradient descent computes the objective function of all samples, then decide direction of descent
    - Takes too long
  - SGD computes the objective function on subsets of samples
    - The subset should not be biased and properly randomized to ensure no correlation between samples
- The subset is called a mini-batch
- Size of the mini-batch determines the training speed and accuracy
  - Usually somewhere between 32-1024 samples per mini-batch
- Definition: 1 batch vs 1 epoch

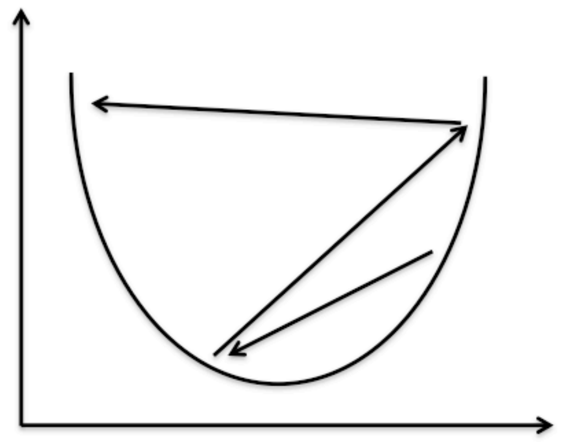
Self regularizaing property of SGD

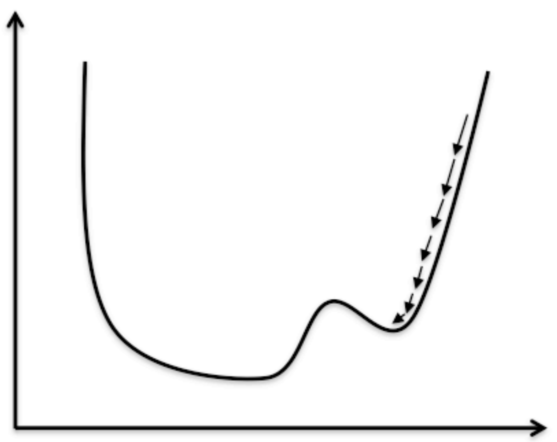
- SGD by its randomized nature does not overfit (as fast)
  - Considered as an implicit regularization (no change in the loss)

https://cbmm.mit.edu/sites/default/files/publications/CBMM-Memo-067-v3.pdf

Learning rate

- How fast to go along the gradient direction is controlled by the learning rate
- Too large models diverge
- Too small the model get stuck in local minimas and takes too long to train

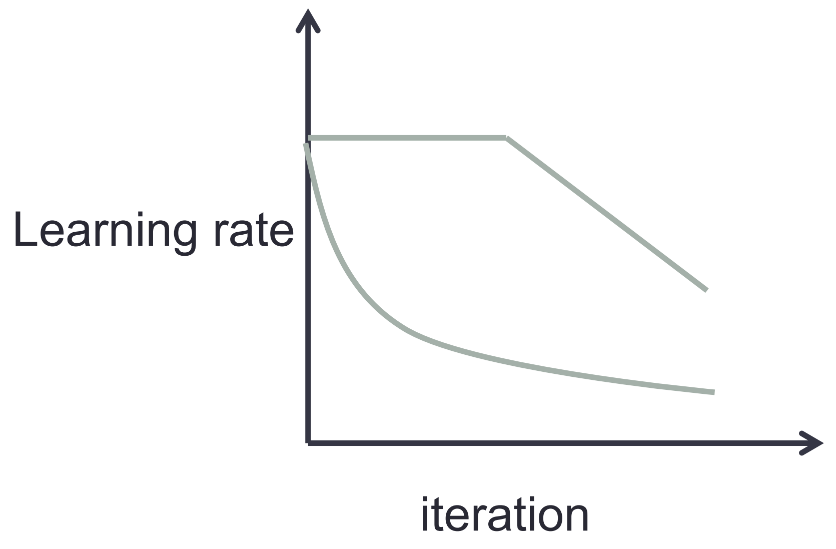




http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

Learning rate scheduling

- Usually starts with a large learning rate then gets smaller later
- Depends on your task
- Automatic ways to adjust the learning rate : Adagrad, Adam, etc. (still need scheduleing still)

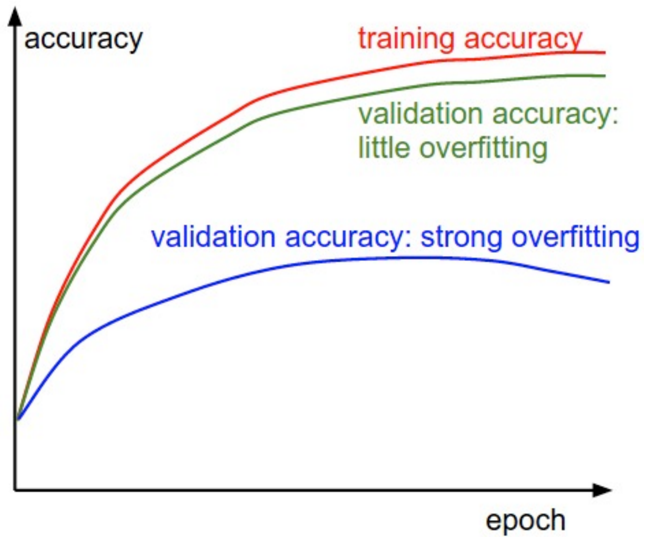


Learning rate strategies (annealing)

- Step decay: reduce learning rate by x after y epochs
- New bob method: half learning rate every time the validation error goes up. Only plausible in larger tasks
- Exponential decay: multiplies the learning rate by exp(-rate*epoch number)

Overfitting

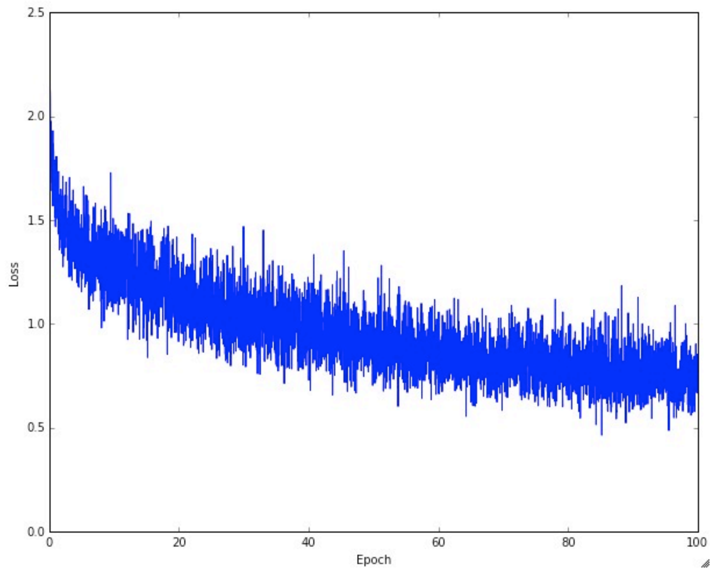
- You can keep doing back propagation forever!
- The training loss will always go down
- But it overfits
- Need to monitor performance on a held out set
- Stop or decrease learning rate when overfit happens



http://cs231n.github.io/neural-networks-3/

Monitoring performance

- Monitor performance on a dev/validation set
  - This is NOT the test set
- Can monitor many criterions
  - Loss function
  - Classification accuracy
- Sometimes these disagree
- Actual performance can be noisy, need to see the trend



http://cs231n.github.io/neural-networks-3/

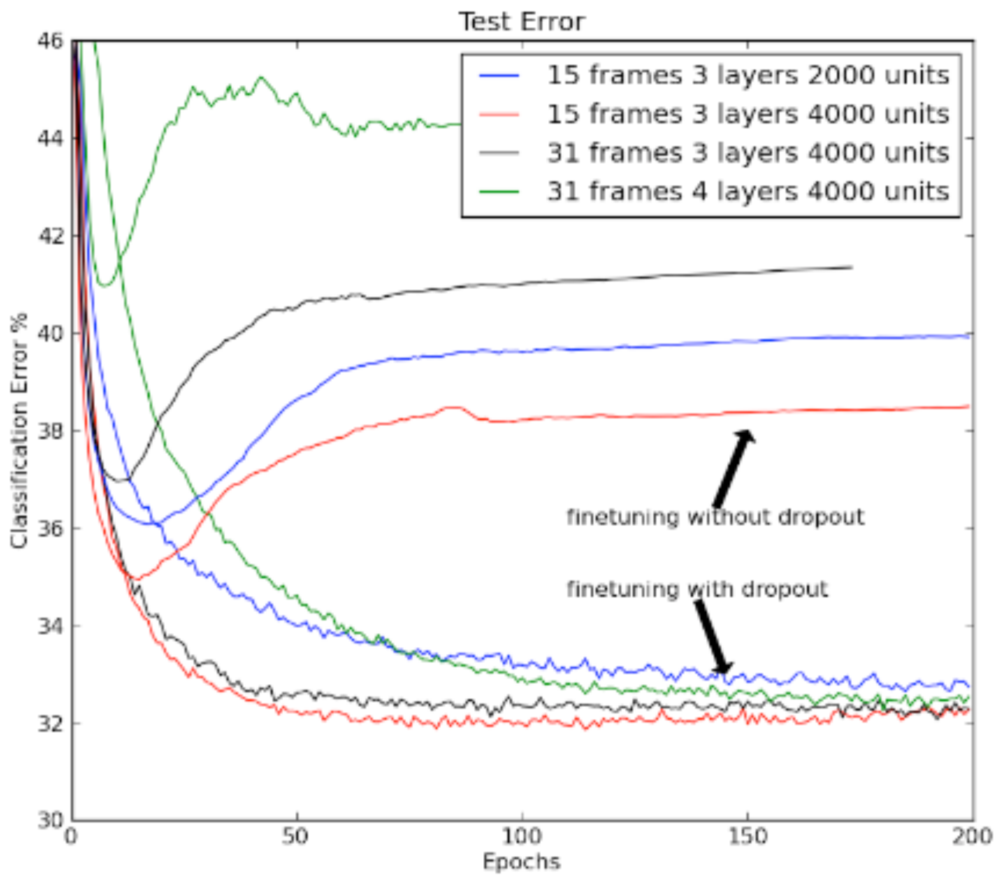
Reducing overfitting - dropout

- A *RECENT* (2012) regularization technique for reducing overfitting
- Randomly turn off different subset of neurons during training
  - Network no longer depend on any particular neuron
  - Force the model to have redundancy – robust to any corruption in input data
  - A form of performing model averaging (assemble of experts)
- Now a standard technique

Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Dropout on TIMIT

- A phoneme recognition task



Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Batch normalization

- Recent technique for (implicit) regularization
- Normalize every mini-batch at various batch norm layers to standard Gaussian (different from global normalization of the inputs)
- Place batch norm layers before non-linearities
- Faster training and better generalizations

https://arxiv.org/abs/1502.03167

Vanishing/Exploding gradient

- Backprop introduces many multiplications down chain
- The gradient value gets smaller and smaller
  - The deeper the network the smaller the gradient in the lower layers
  - Lower layers changes too slowly (or not at all)
  - Hard to train very deep networks (>6 layers)
- The opposite can also be true. The gradient explodes from repeated multiplication
  - Put a maximum value for the gradient (Gradient clipping)

- How to deal with this?
  - Next lectures

Neural networks

- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- t-SNE
- Demos
  - Tensorflow, Gcloud, Keras
- CNN, RNN, LSTM, GRU <- Next class

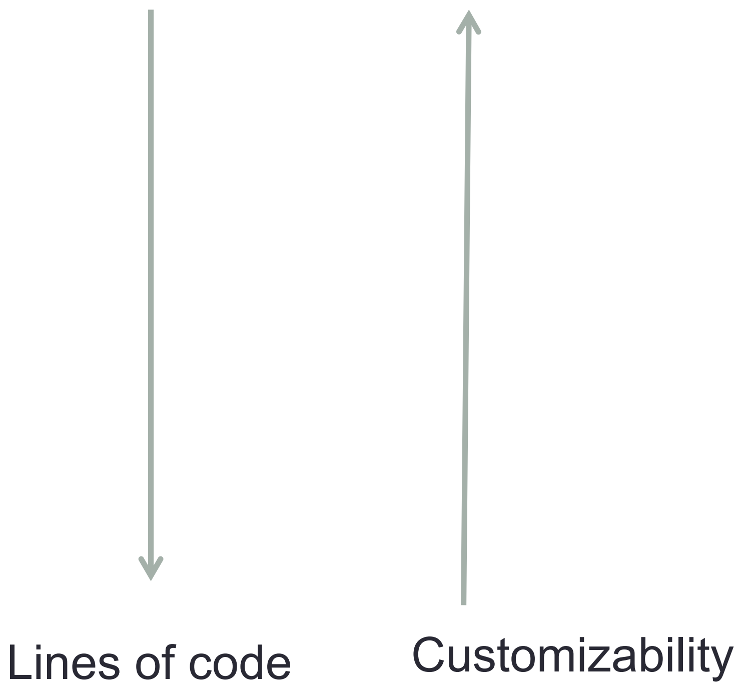
t-distributed Stochastic Neighbor Embedding (t-SNE

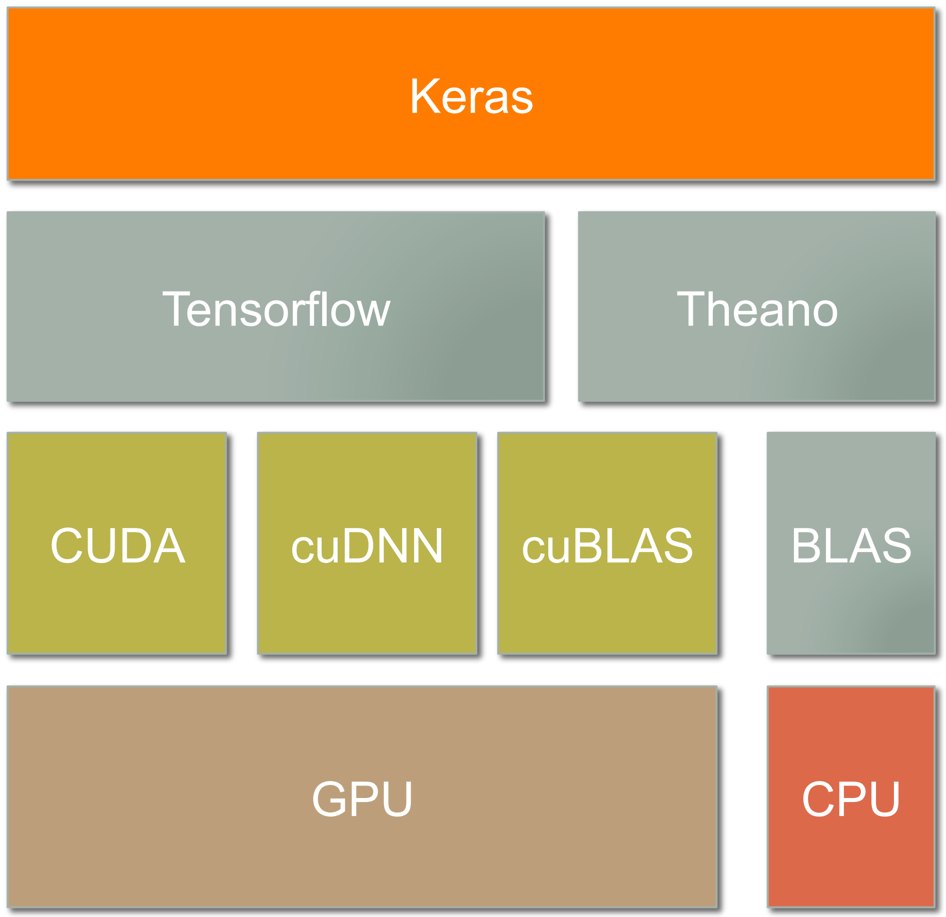
- **Non-linear** dimensionality reduction technique that is good for reducing high dimensional data to 2-3 dimensions
  - Mainly for visualization
- Tries to preserve small pair-wise distances locally
  - Preserves relative order well
- Original features are distorted

http://projector.tensorflow.org/

What toolkit

- Tensorflow – lower level
- Keras – higher level





Demos

- Tensorboard
    - What’s a tensor?
- Gcloud

Homework

- Part I out tonight
- Part II some time this week (this part needs Gcloud)
- Next week office hour is on Wednesday