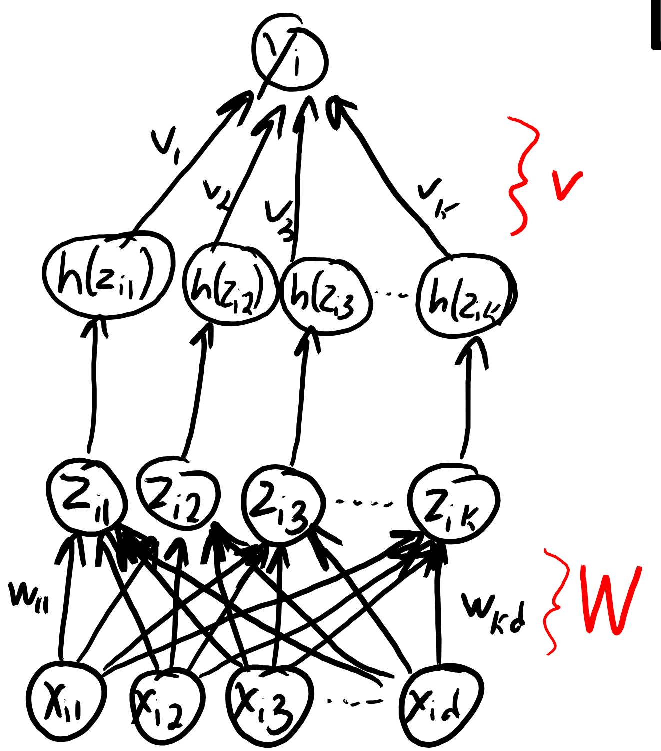
CPSC 340:
Machine Learning and Data Mining

More Deep Learning

Fall 2019

Last Time: Deep Learning

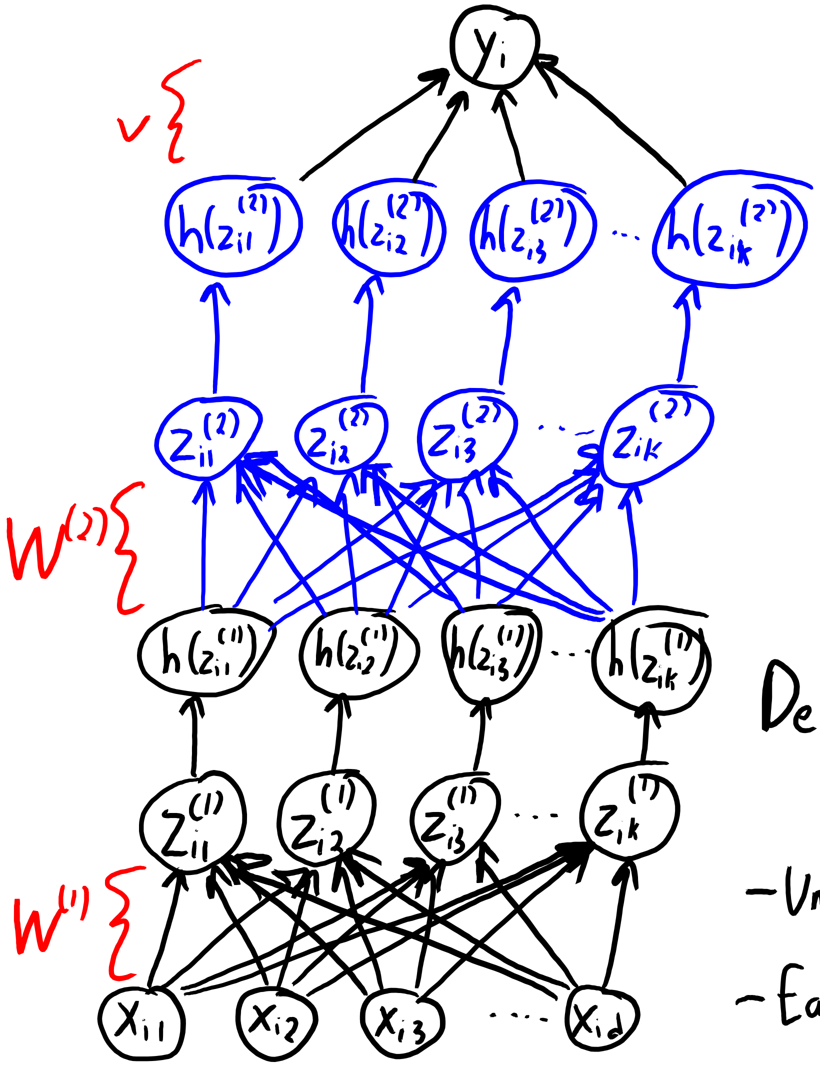
Neural network:

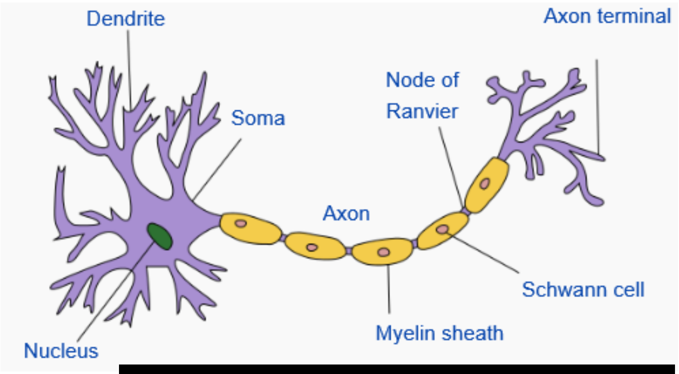


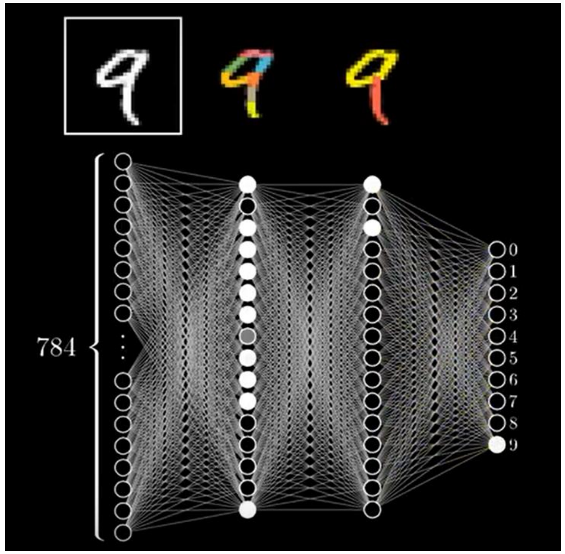
$y_i = v^T h(Wx_i)$

Learn 'W' and 'v' together

- learn features for supervised learning.
- Non-linear 'h' makes it a universal approximator for large 'k'







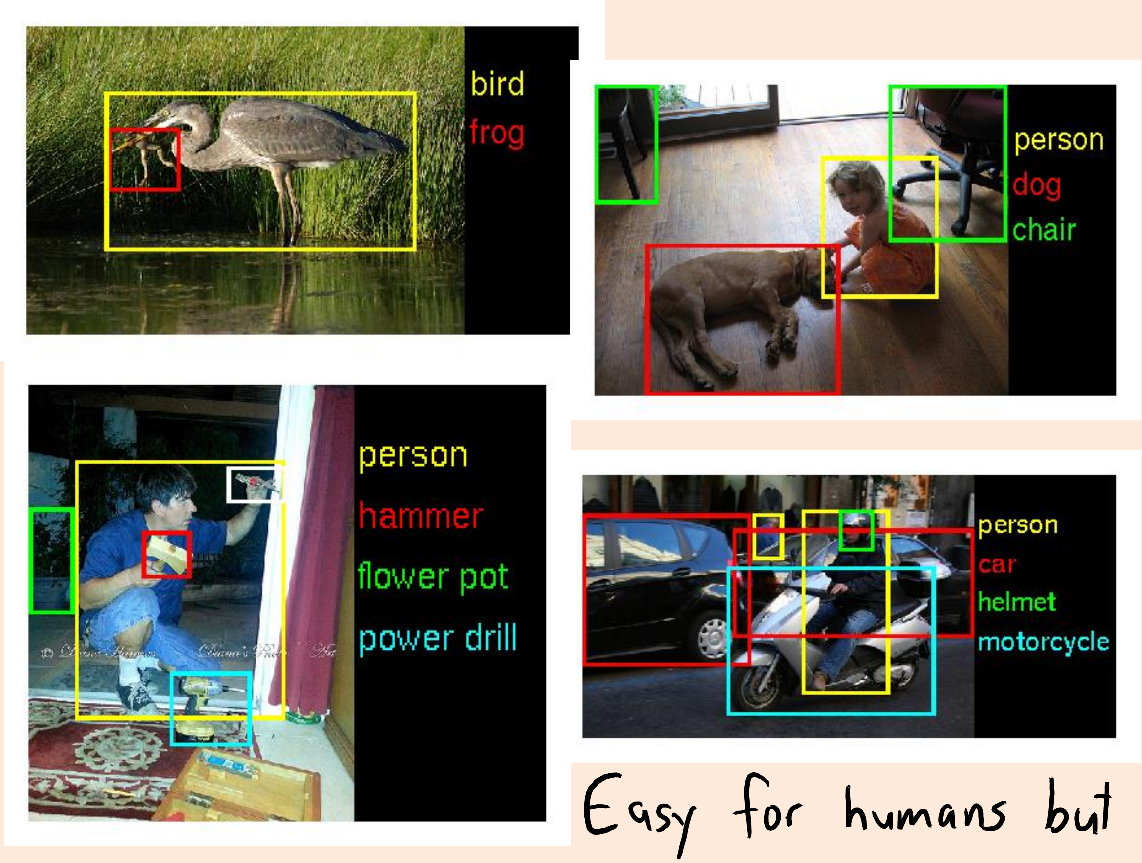
Deep neural networks:

$y_i = v^T h(W^{(2)} h(W^{(1)} x_i))$

- Unprecedented performance on difficult problems.
- Each layer combines "parts" from previous layer.

ImageNet Challenge

- Millions of labeled images, 1000 object classes.



Easy for humans but <u>hard</u> for computers.

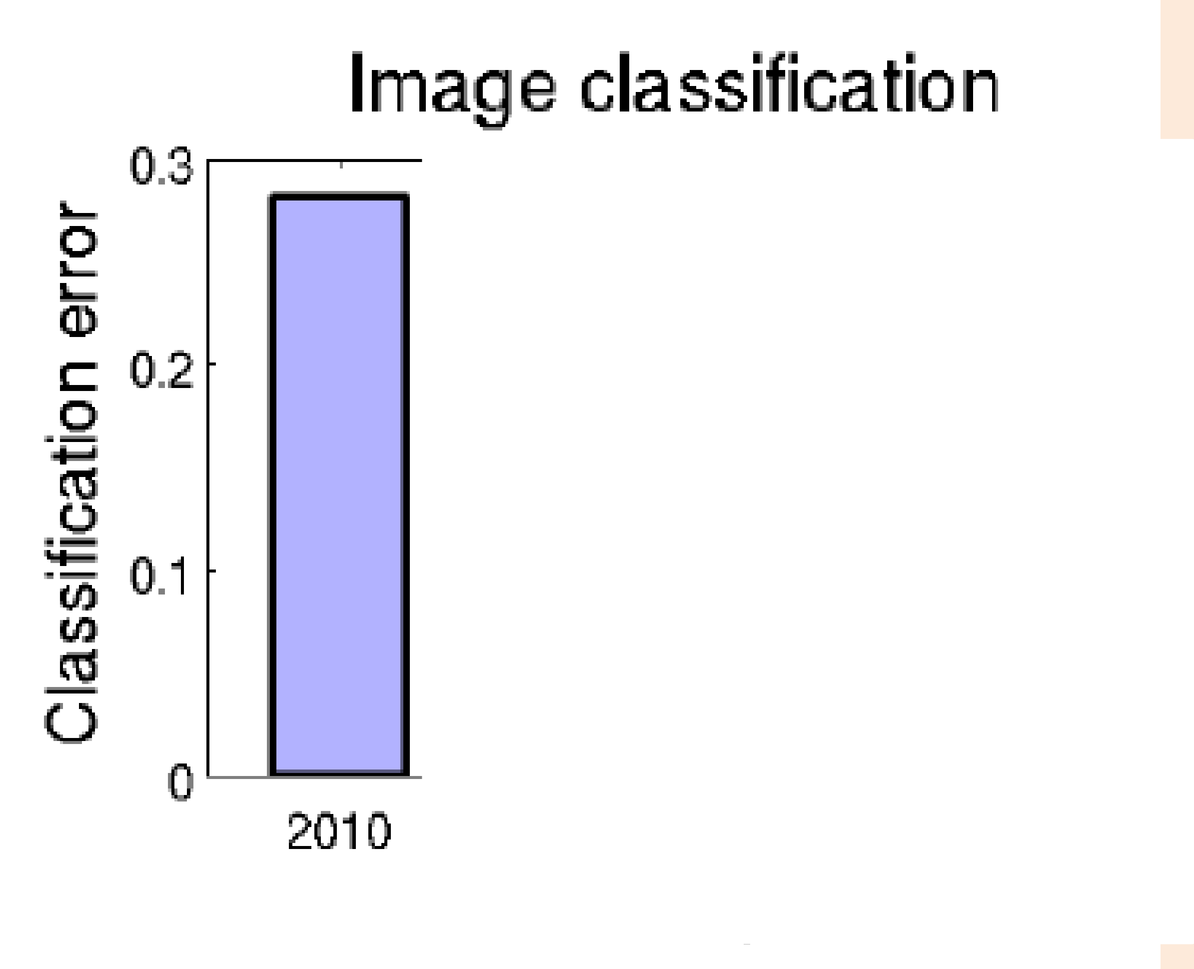
http://www.image-net.org/challenges/LSVRC/2014/

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

Syberian Husky

Canadian Husky

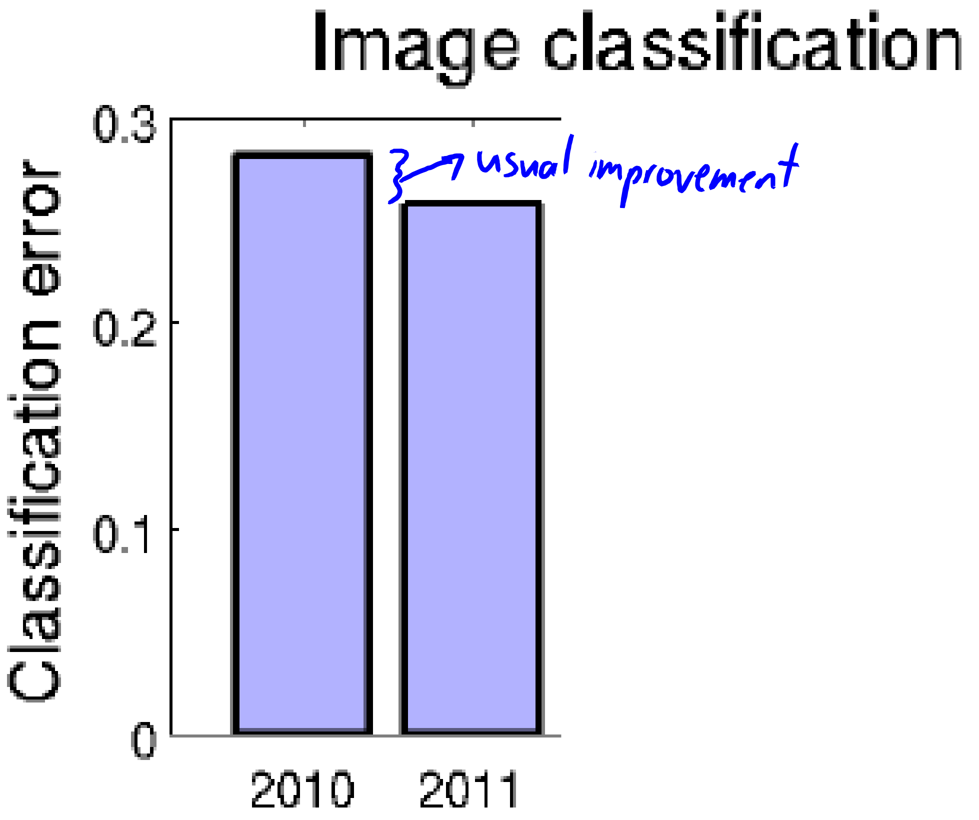


https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

Syberian Husky　　Canadian Husky

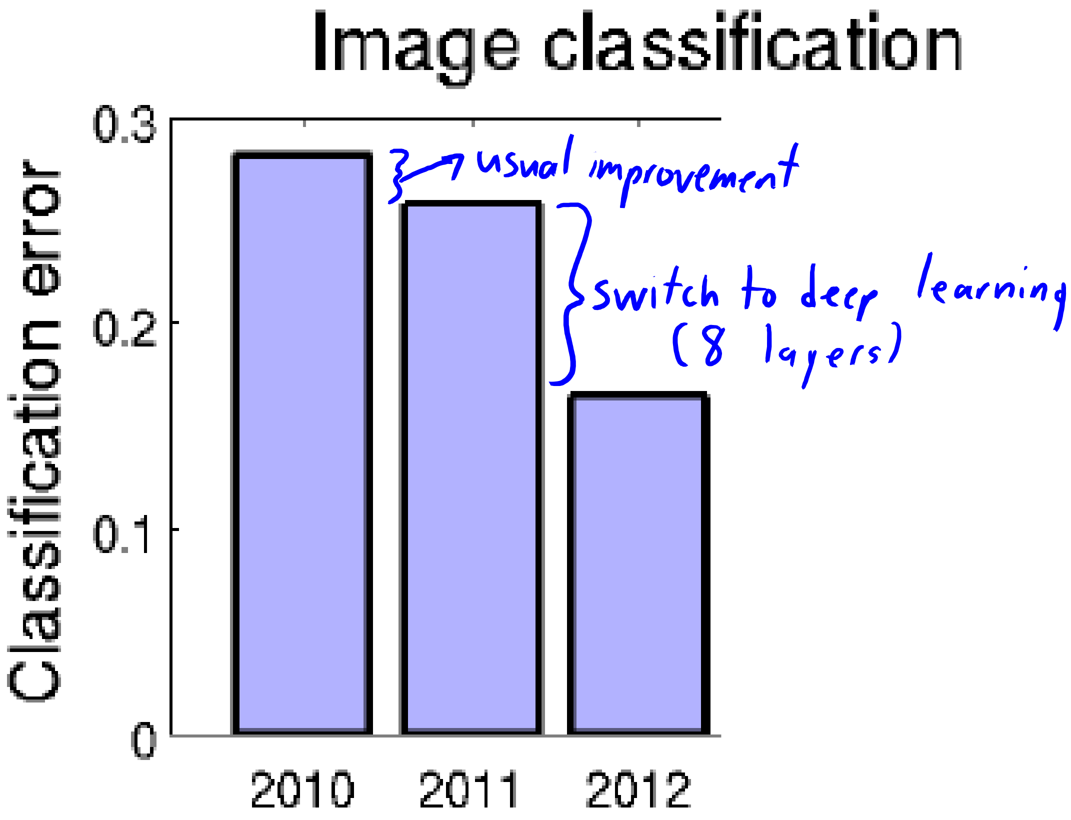


https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

Syberian Husky Canadian Husky

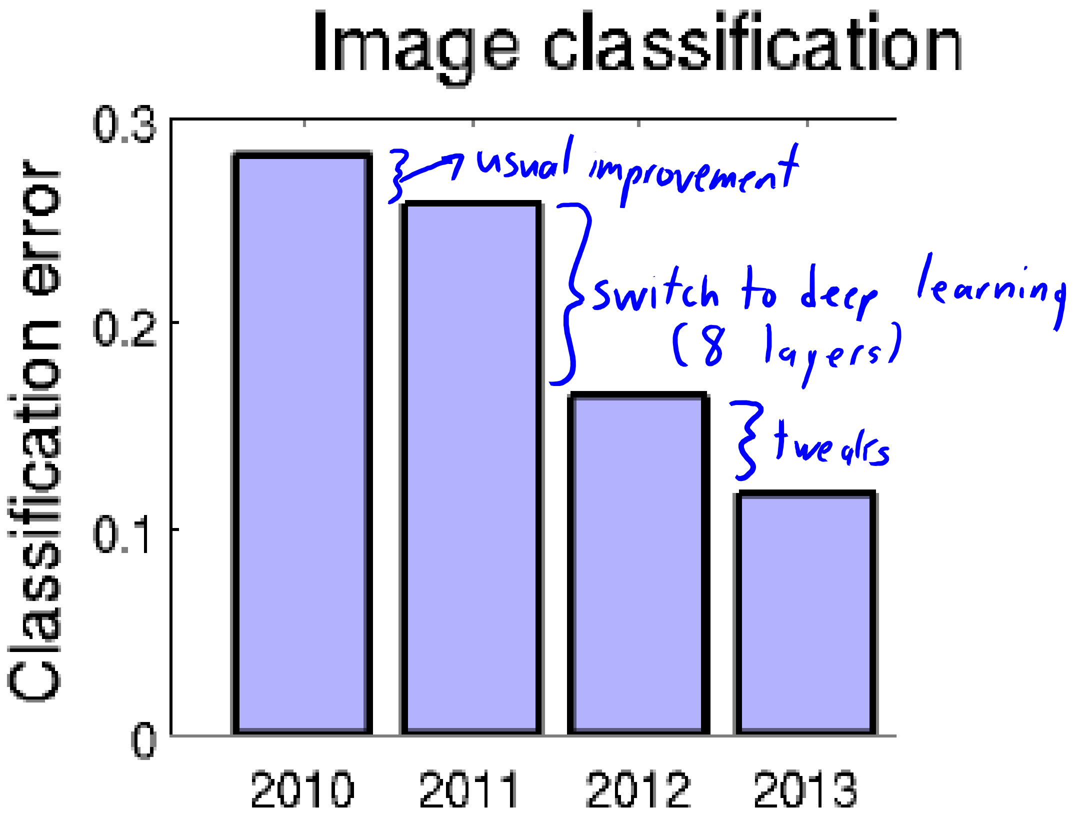


https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

Syberian Husky Canadian Husky



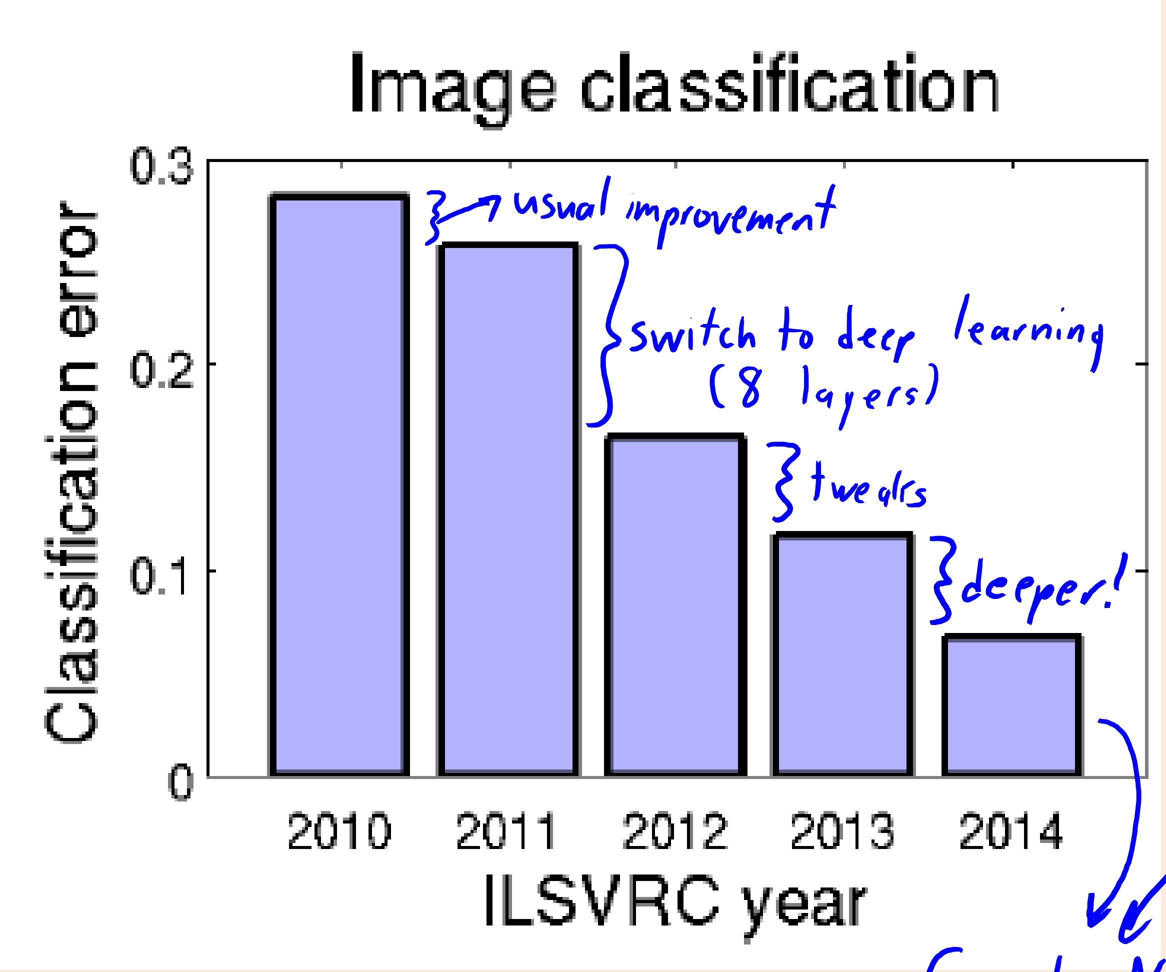
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

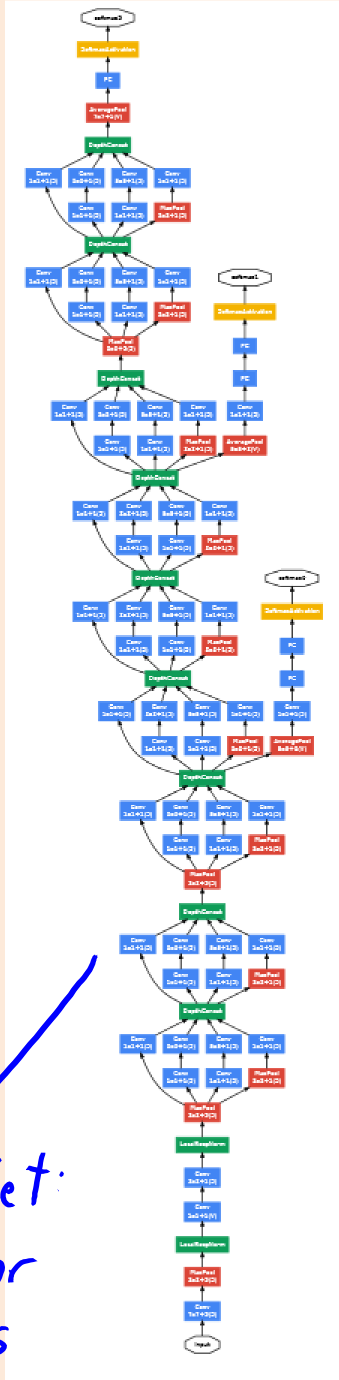
- Object detection task:
  – Single label per image.
  – Humans: ~5% error.

Syberian Husky

Canadian Husky



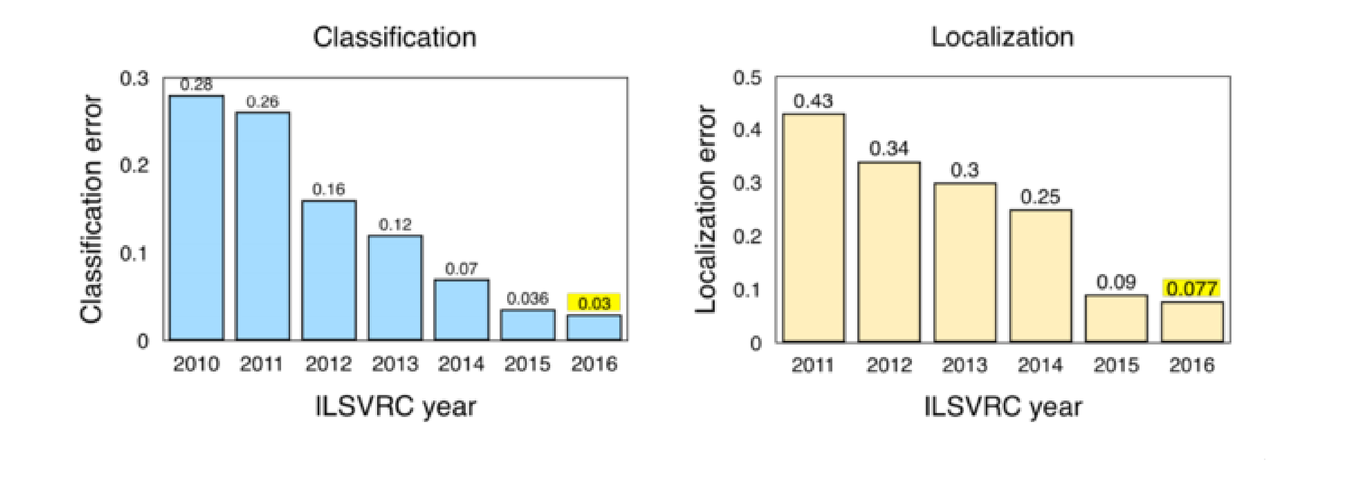
GoogLeNet:
6.7% error
22 layers



https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.
- 2015: Won by Microsoft Asia
  - 3.6% error.
  - 152 layers, introduced “ResNets”.
  - Also won “localization” (finding location of objects in images).
- 2016: Chinese University of Hong Kong:
  - Ensembles of previous winners and other existing methods.
- 2017: fewer entries, organizers decided this would be last year.



http://www.themtank.org/a-year-in-computer-vision

(pause)

Deep Learning Practicalities

- This lecture focus on deep learning practical issues:
  - Backpropagation to compute gradients.
  - Stochastic gradient training.
  - Regularization to avoid overfitting.

- Next lecture:
  - Special ‘W’ restrictions to further avoid overfitting.

But first: Adding Bias Variables

- Recall fitting line regression with a bias:

$$\hat{y}_i = \sum_{j=1}^{d} w_j x_{ij} + \beta$$

  – We avoided this by adding a column of ones to X.

- In neural networks we often want a bias on the output:

$$\hat{y}_i = \sum_{c=1}^{k} v_c h(w_c^T x_i) + \beta$$

- But we also often also include biases on each z_{ic}:

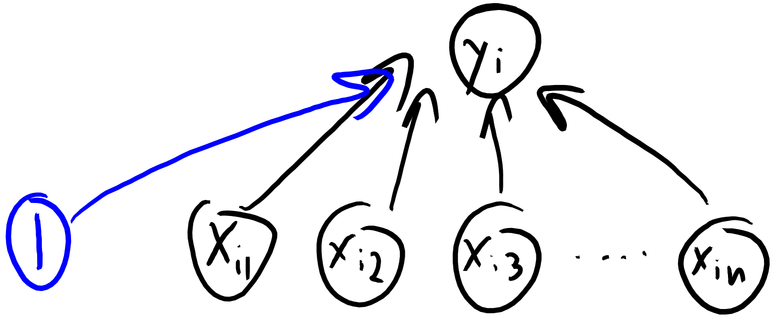
$$\hat{y}_i = \sum_{c=1}^{k} v_c h(w_c^T x_i + \beta_c) + \beta$$

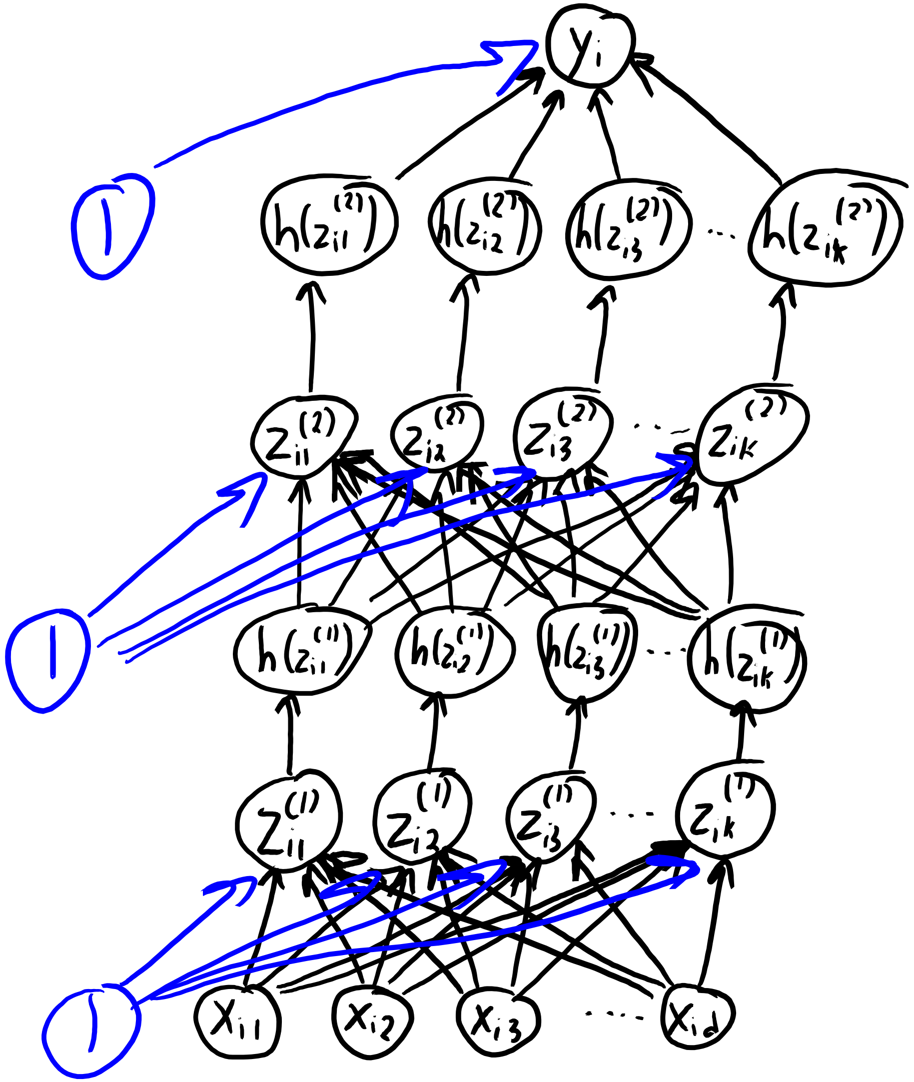
  – A bias towards this $h(z_{ic})$ being either 0 or 1.

- Equivalent to adding to vector $h(z_i)$ an extra value that is always 1.
  – For sigmoids, you could equivalently make one row of w_c be equal to 0.

But first: Adding Bias Variables

Linear model with bias:





Artificial Neural Networks

- With squared loss and 1 hidden layer, our objective function is:

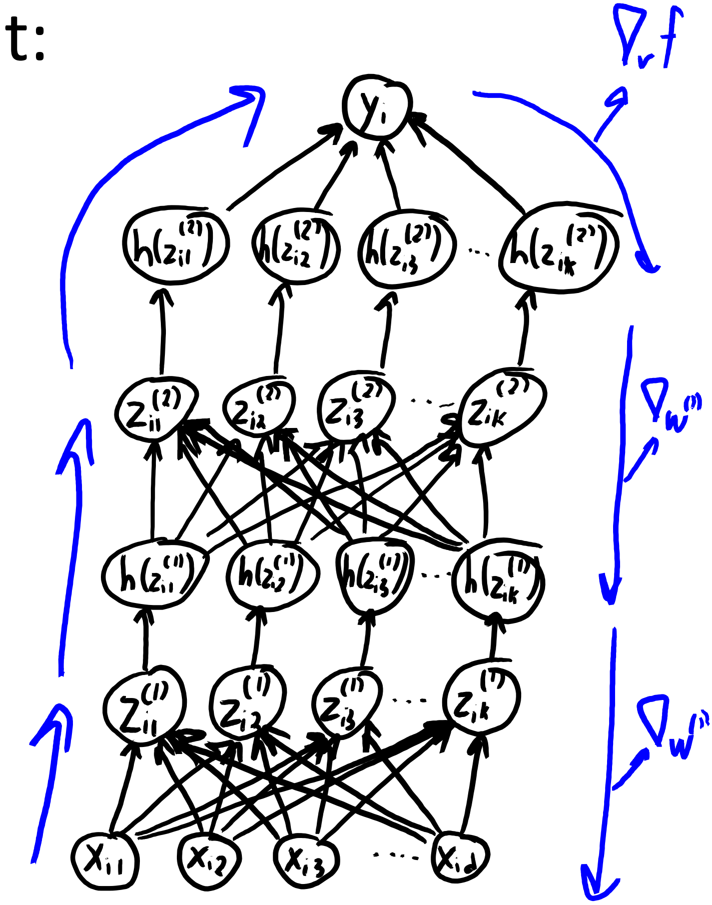
$$f(v, W) = \frac{1}{2}\sum_{i=1}^{n}(v^T h(Wx_i) - y_i)^2$$

- Usual training procedure: stochastic gradient.
  - Compute gradient of random example ‘i’, update both ‘v’ and ‘W’.
  - Highly non-convex and can be difficult to tune.

- Computing the gradient is known as “backpropagation”.
  - Video giving motivation here.

Backpropagation

- Overview of how we compute neural network gradient:
  - Forward propagation:
    - Compute $z_i^{(1)}$ from x_i.
    - Compute $z_i^{(2)}$ from $z_i^{(1)}$.
    - …
    - Compute $\hat{y}_i$ from $z_i^{(m)}$, and use this to compute error.
  - Backpropagation:
    - Compute gradient with respect to regression weights ‘v’.
    - Compute gradient with respect to $z_i^{(m)}$ weights $W^{(m)}$.
    - Compute gradient with respect to $z_i^{(m-1)}$ weights $W^{(m-1)}$.
    - …
    - Compute gradient with respect to $z_i^{(1)}$ weights $W^{(1)}$.



- “Backpropagation” is the chain rule plus some bookkeeping for speed.

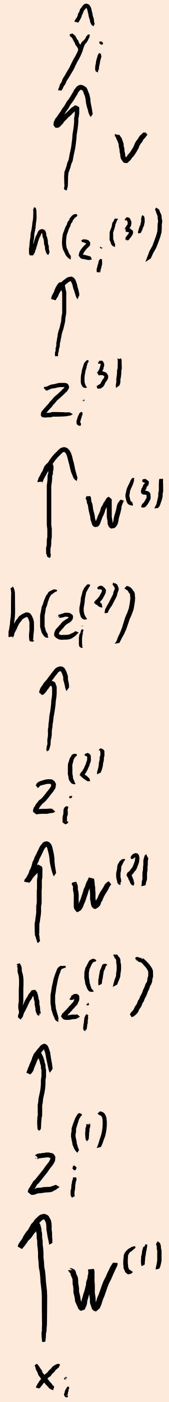
Backpropagation

- Let's illustrate backpropagation in a simple setting:
  - 1 training example, 3 hidden layers, 1 hidden "unit" in layer.

$$f(W^{(1)}, W^{(2)}, W^{(3)}, v) = \frac{1}{2}(\underbrace{\hat{y}_i - y_i}_{r})^2 \quad \text{where} \quad \hat{y}_i = v h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i)))$$

$$\frac{\partial f}{\partial v} = r\, h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) = r\, h(z_i^{(3)})$$

$$\frac{\partial f}{\partial W^{(3)}} = r\, v\, h'(W^{(3)} h(W^{(2)} h(W^{(1)} x_i)))\, h(W^{(2)} h(W^{(1)} x_i)) = r\, v\, h'(z_i^{(3)})\, h(z_i^{(2)})$$



Backpropagation

- Let's illustrate backpropagation in a simple setting:
  - 1 training example, 3 hidden layers, 1 hidden "unit" in layer.

$$f(W^{(1)}, W^{(2)}, W^{(3)}, v) = \frac{1}{2}\underbrace{(\hat{y}_i - y_i)}_{r}^2 \quad \text{where} \quad \hat{y}_i = v h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i)))$$

$$\frac{\partial f}{\partial v} = r\, h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) = r\, h(z_i^{(3)})$$

$$\frac{\partial f}{\partial W^{(3)}} = r v h'(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) h(W^{(2)} h(W^{(1)} x_i)) = r v h'(z_i^{(3)}) h(z_i^{(2)})$$

$$\frac{\partial f}{\partial W^{(2)}} = r v h'(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) W^{(3)} h'(W^{(2)} h(W^{(1)} x_i)) h(W^{(1)} x_i) = r^{(3)} W^{(3)} h'(z_i^{(2)}) h(z_i^{(1)})$$

$$\frac{\partial f}{\partial W^{(1)}} = r v h'(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) W^{(3)} h'(W^{(2)}(W^{(1)} x_i)) W^{(2)} h'(W^{(1)} x_i) x_i = r^{(2)} W^{(2)} h'(z_i^{(1)}) x_i$$

Backpropagation

- Let's illustrate backpropagation in a simple setting:
  - 1 training example, 3 hidden layers, 1 hidden "unit" in layer.

$$\frac{\partial f}{\partial v} = r\, h(z_i^{(3)})$$

$$\frac{\partial f}{\partial W^{(3)}} = r\, v\, h'(z_i^{(3)})\, h(z_i^{(2)})$$

$$\frac{\partial f}{\partial W^{(2)}} = r^{(3)} W^{(3)} h'(z_i^{(2)})\, h(z_i^{(1)})$$

$$\frac{\partial f}{\partial W^{(1)}} = r^{(2)} W^{(2)} h'(z_i^{(1)})\, x_i$$

$$\frac{\partial f}{\partial v_c} = r\, h(z_{ic}^{(3)})$$

$$\frac{\partial f}{\partial W_{c'c}^{(3)}} = \underbrace{r\, v_c\, h'(z_{ic'}^{(3)})}\, h(z_{ic}^{(2)})$$

$$\frac{\partial f}{\partial W_{c'c}^{(2)}} = \underbrace{\left[\sum_{c''=1}^{k} r_{c''}^{(3)} W_{c''c'}^{(3)}\right] h'(z_{ic'}^{(2)})}\, h(z_{ic}^{(1)})$$

$$\frac{\partial f}{\partial W_{cj}^{(1)}} = \left[\sum_{c''=1}^{k} r_{c''}^{(2)} W_{c''c}^{(2)}\right] h'(z_{ic}^{(1)})\, x_j$$

  - Only the first 'r' changes if you use a different loss.
  - With multiple hidden units, you get extra sums.
    - Efficient if you store the sums rather than computing from scratch.

Backpropagation

- I've marked those backprop math slides as bonus.
- Do you need to know how to do this?
  - Exact details are probably not vital (there are many implementations).
  - "Automatic differentiation" is becoming standard and has same cost.
  - But understanding basic idea helps you know what can go wrong.
    - Or give hints about what to do when you run out of memory.
  - See discussion here by a neural network expert.

- You should know cost of backpropagation:
  - Forward pass dominated by matrix multiplications by $W^{(1)}$, $W^{(2)}$, $W^{(3)}$, and 'v'.
    - If have 'm' layers and all z_i have 'k' elements, cost would be $O(dk + mk^2)$.
  - Backward pass has same cost as forward pass.
- For multi-class or multi-label classification, you replace 'v' by a matrix:
  - Softmax loss is often called "cross entropy" in neural network papers.

Deep Learning Vocabulary

- "Deep learning": Models with many hidden layers.
  - Usually neural networks.
- "Neuron": node in the neural network graph.
  - "Visible unit": feature.
  - "Hidden unit": latent factor z_{ic} or $h(z_{ic})$.
- "Activation function": non-linear transform.
- "Activation": $h(z_i)$.
- "Backpropagation": compute gradient of neural network.
  - Sometimes "backpropagation" means "training with SGD".
- "Weight decay": L2-regularization.
- "Cross entropy": softmax loss.
- "Learning rate": SGD step-size.
- "Learning rate decay": using decreasing step-sizes.
- "Vanishing gradient": underflow/overflow during gradient calculation.

(pause)

ImageNet Challenge and Optimization

- ImageNet challenge:
  - Use millions of images to recognize 1000 objects.
- ImageNet organizer visited UBC summer 2015.
- "Besides huge dataset/model/cluster, what is the most important?"
  1. Image transformations (translation, rotation, scaling, lighting, etc.).
  2. Optimization.
- Why would optimization be so important?
  - Neural network objectives are highly non-convex (and worse with depth).
  - Optimization has huge influence on quality of model.

Stochastic Gradient Training

- Standard training method is stochastic gradient (SG):
  - Choose a random example 'i'.
  - Use backpropagation to get gradient with respect to all parameters.
  - Take a small step in the negative gradient direction.
- Challenging to make SG work:
  - Often doesn't work as a "black box" learning algorithm.
  - But people have developed a lot of tricks/modifications to make it work.
- Highly non-convex, so are the problem local mimina?
  - Some empirical/theoretical evidence that local minima are not the problem.
  - If the network is "deep" and "wide" enough, we think all local minima are good.
  - But it can be hard to get SG to close to a local minimum in reasonable time.

Parameter Initialization

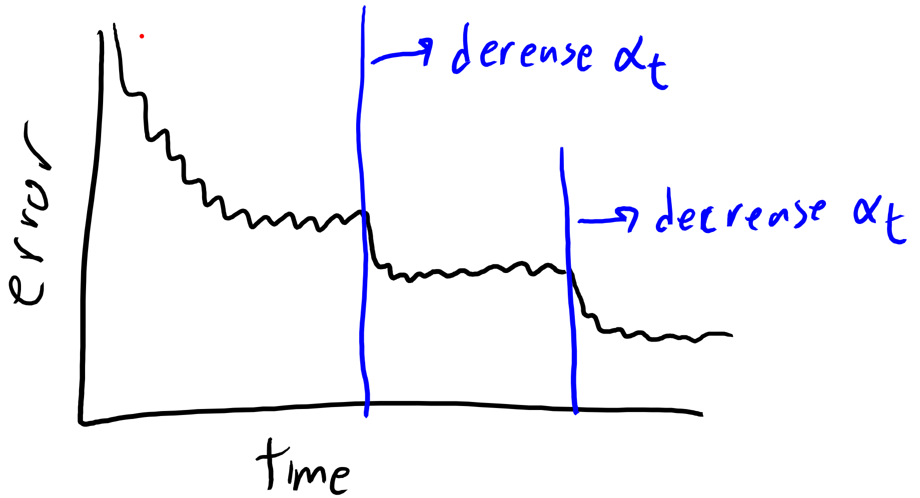
- Parameter initialization is crucial:
  - Can't initialize weights in same layer to same value, or they will stay same.
  - Can't initialize weights too large, it will take too long to learn.
- A traditional random initialization:
  - Initialize bias variables to 0.
  - Sample from standard normal, divided by 10^5 (0.00001*randn).
    - w = .00001*randn(k,1)
  - Performing multiple initializations does not seem to be important.

Parameter Initialization

- Parameter initialization is crucial:
  - Can't initialize weights in same layer to same value, or they will stay same.
  - Can't initialize weights too large, it will take too long to learn.
- Also common to transform data in various ways:
  - Subtract mean, divide by standard deviation, "whitten", standardize y_i.
- More recent initializations try to standardize initial z_i:
  - Use different initialization in each layer.
  - Try to make variance of z_i the same across layers.
    - Popular approach is to sample from standard normal, divide by sqrt(2*nInputs).
  - Use samples from uniform distribution on [-b,b], where $b = \frac{\sqrt{6}}{\sqrt{k^{(m)} + k^{(m-1)}}}$

Setting the Step-Size

- Stochastic gradient is very sensitive to the step size in deep models.
- Common approach: manual "babysitting" of the step-size.
  - Run SG for a while with a fixed step-size.
  - Occasionally measure error and plot progress:



  - If error is not decreasing, decrease step-size.

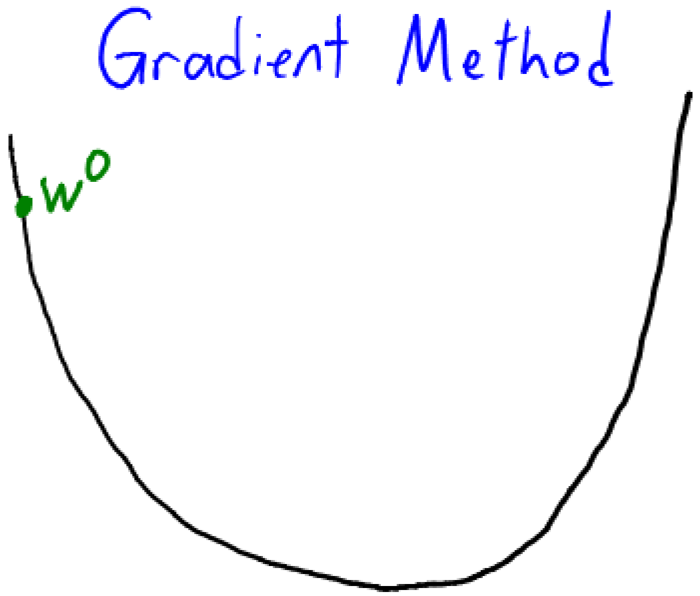
Setting the Step-Size

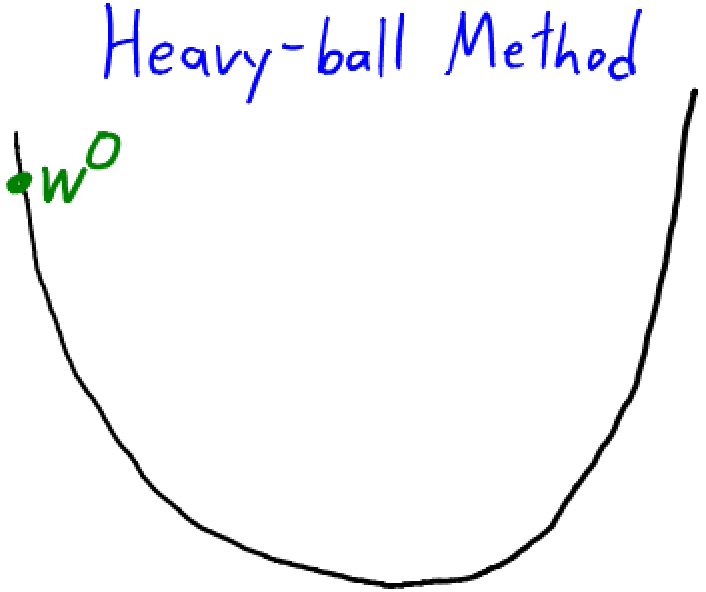
- Stochastic gradient is very sensitive to the step size in deep models.
- Bias step-size multiplier: use bigger step-size for the bias variables.
- Momentum (stochastic version of "heavy-ball" algorithm):
  - Add term that moves in previous direction:

$$w^{t+1} = w^t - \alpha^t \nabla f_i(w^t) + \underbrace{\beta^t(w^t - w^{t-1})}_{\text{keep going in the old direction}}$$

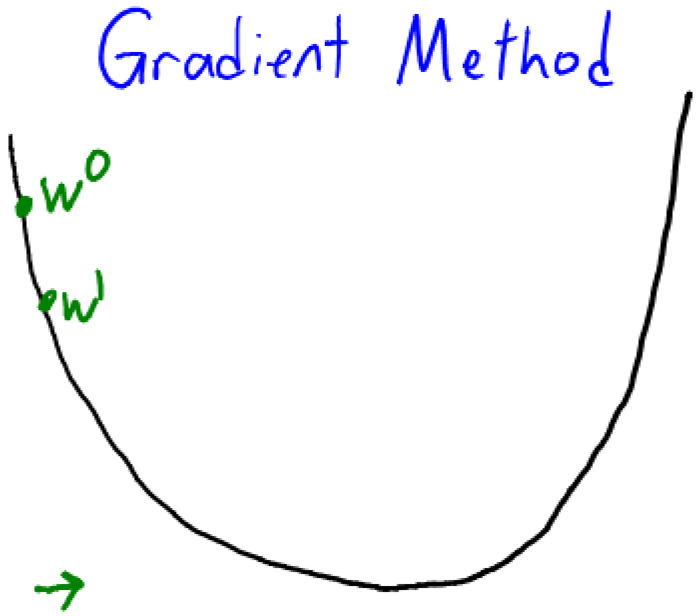
  - Usually $\beta^t = 0.9$.

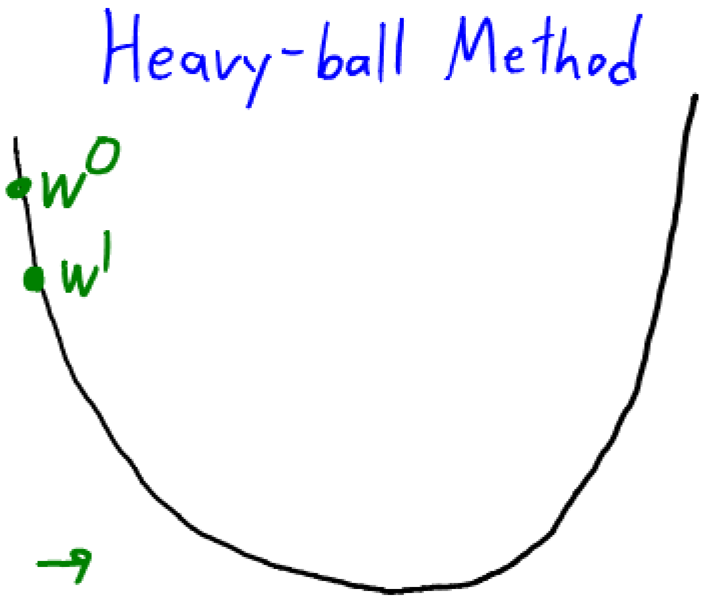
Gradient Descent vs. Heavy-Ball Method



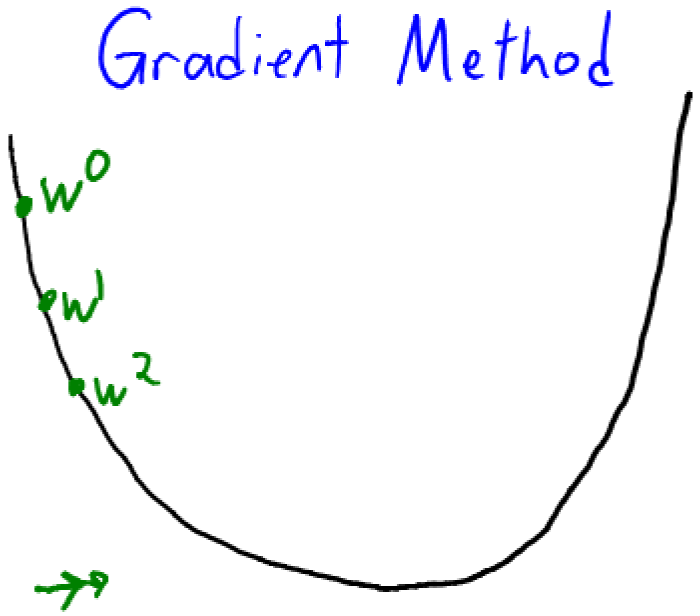


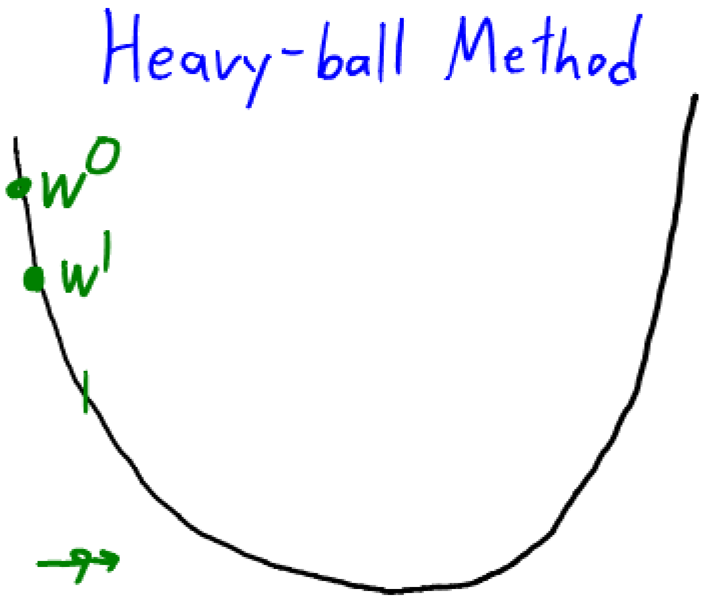
Gradient Descent vs. Heavy-Ball Method



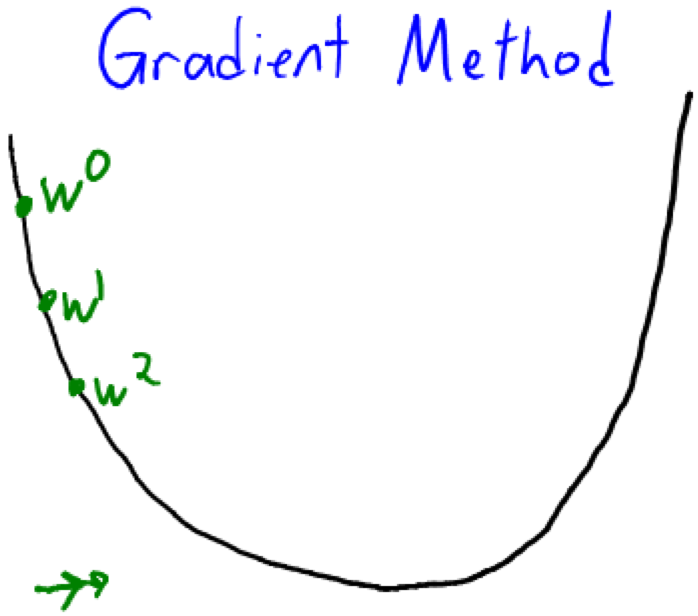


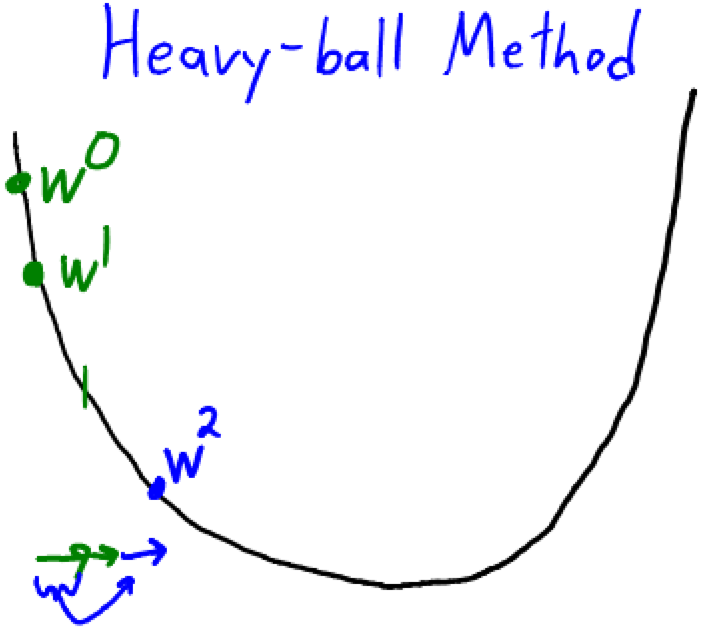
Gradient Descent vs. Heavy-Ball Method



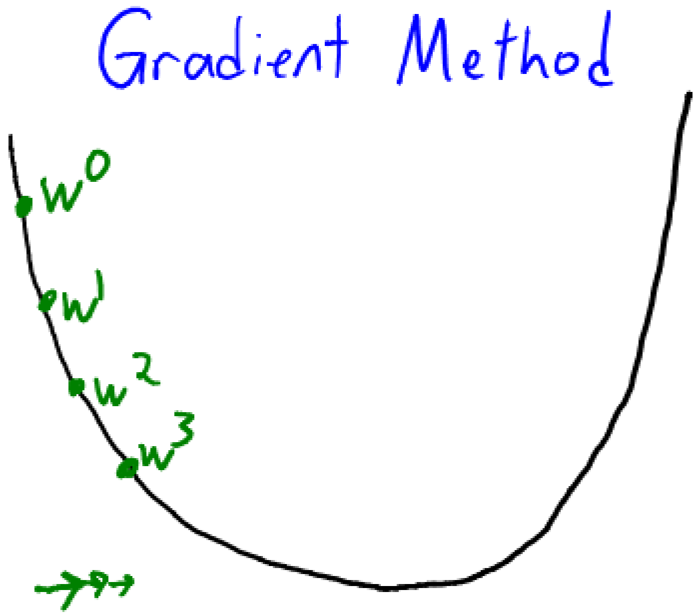


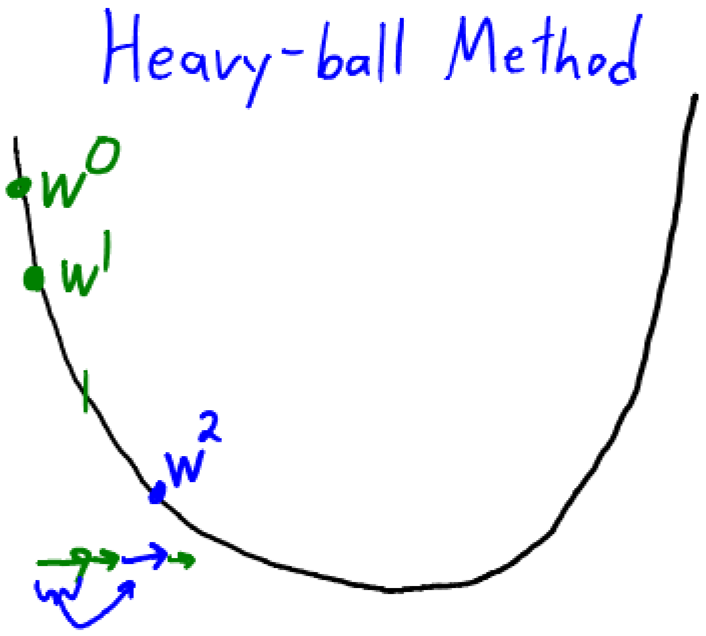
Gradient Descent vs. Heavy-Ball Method



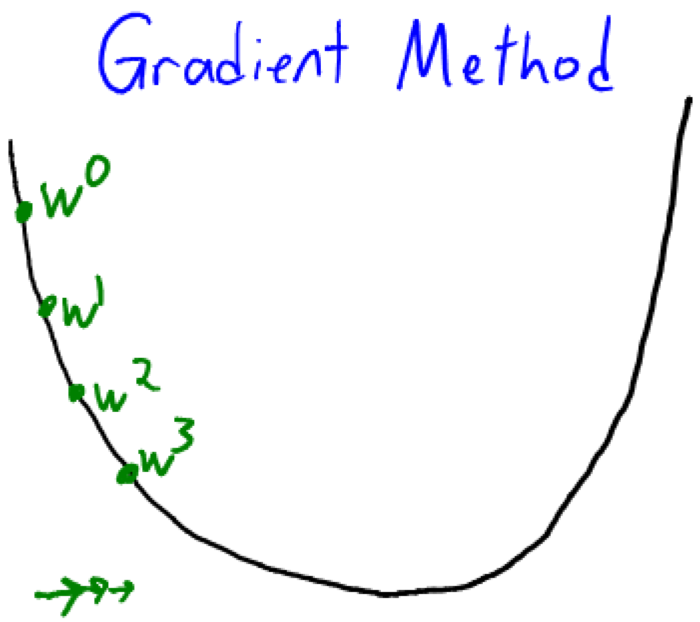


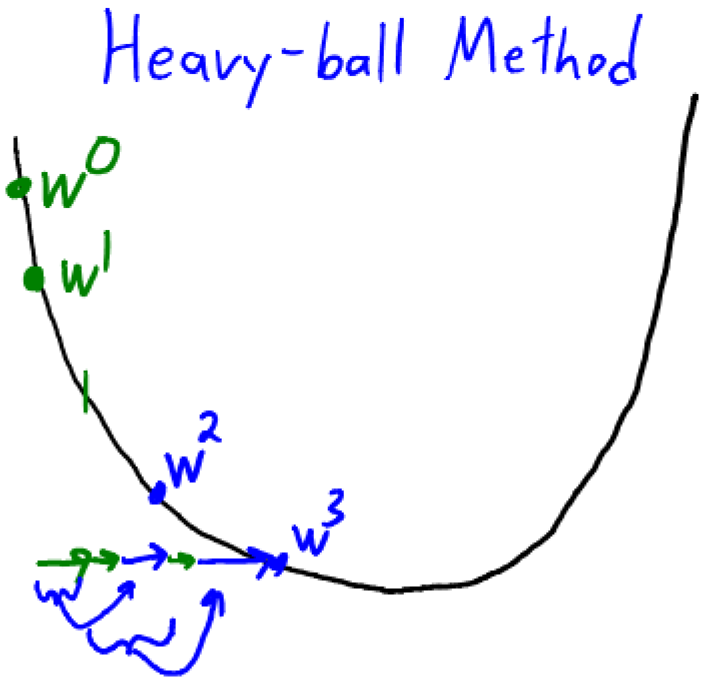
Gradient Descent vs. Heavy-Ball Method



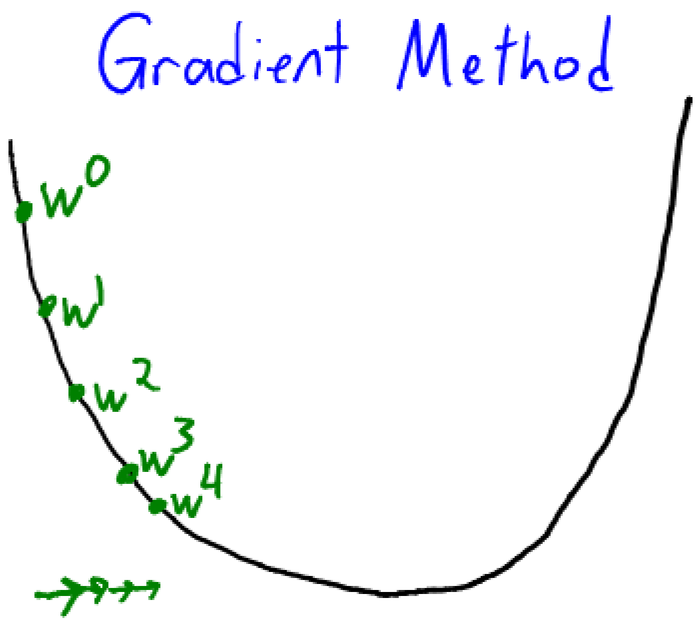


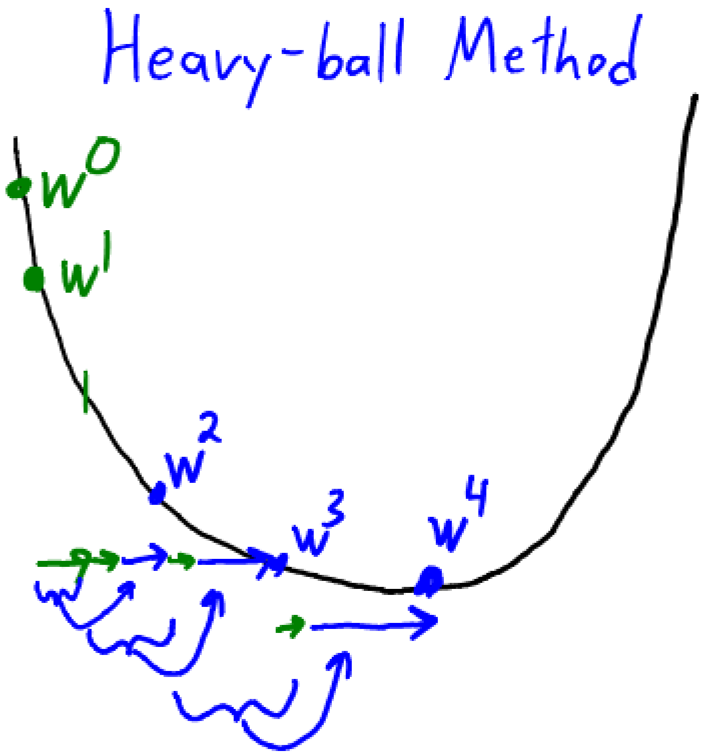
Gradient Descent vs. Heavy-Ball Method



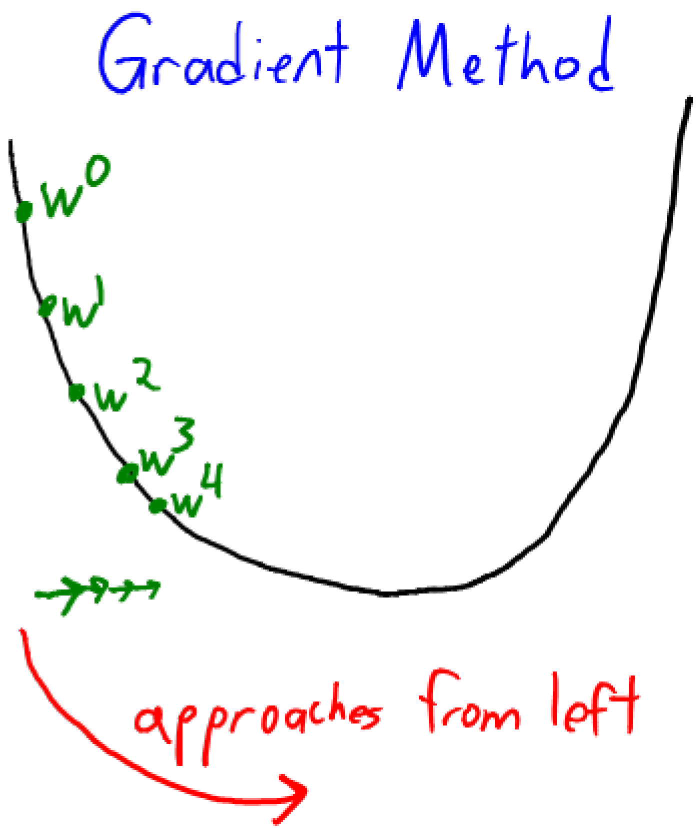


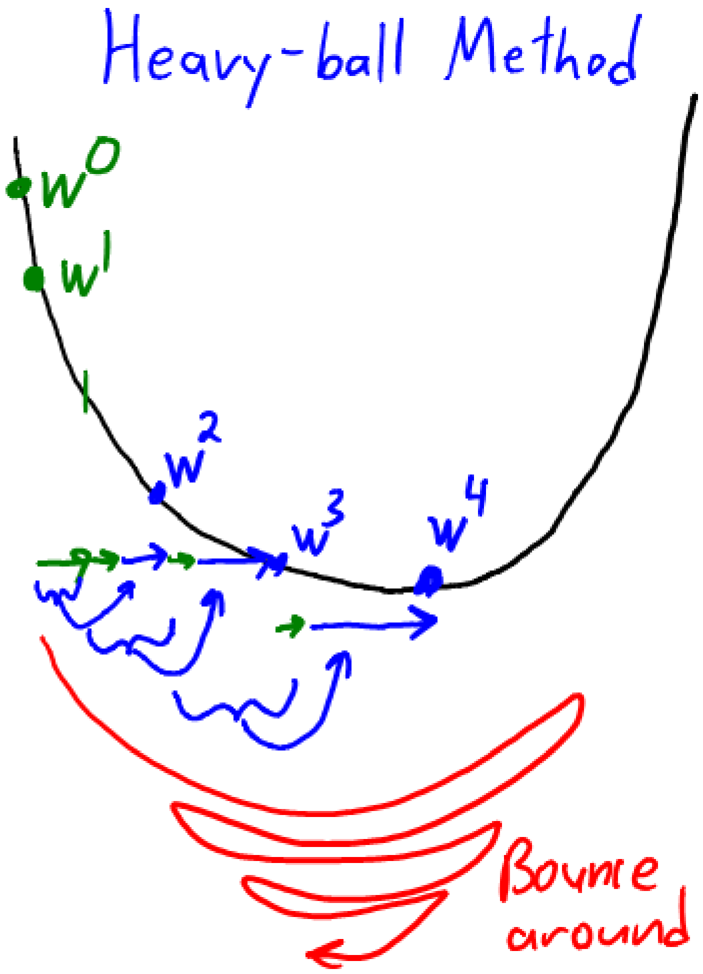
Gradient Descent vs. Heavy-Ball Method





Gradient Descent vs. Heavy-Ball Method





Setting the Step-Size

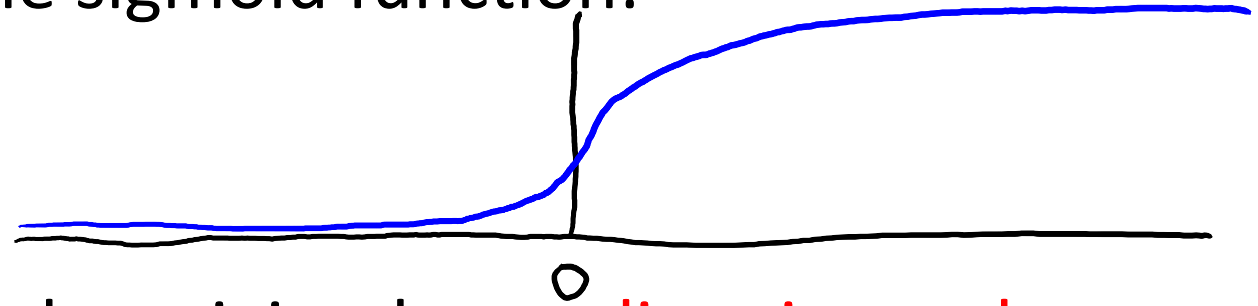
- Automatic method to set step size is Bottou trick:
  1. Grab a small set of training examples (maybe 5% of total).
  2. Do a binary search for a step size that works well on them.
  3. Use this step size for a long time (or slowly decrease it from there).

- Several recent methods using a step size for each variable:
  - AdaGrad, RMSprop, Adam (often work better "out of the box").
  - Seem to be losing popularity to stochastic gradient (often with momentum).
    - SGD often yields lower test error even though it takes longer and requires more tuning of step-size.
- Batch size (number of random examples) also influences results.
  - Bigger batch sizes often give faster convergence but maybe to worse solutions?

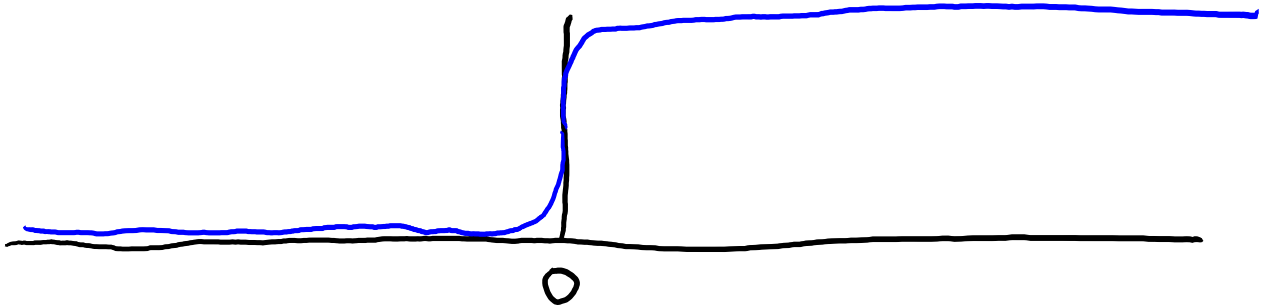
- Another recent trick is batch normalization:
  - Try to "standardize" the hidden units within the random samples as we go.
  - Held as example of deep learning "alchemy" (blog post here about deep learning claims).
    - Sounds science-ey and often works but little theoretical justification/understanding.

Vanishing Gradient Problem

- Consider the sigmoid function:



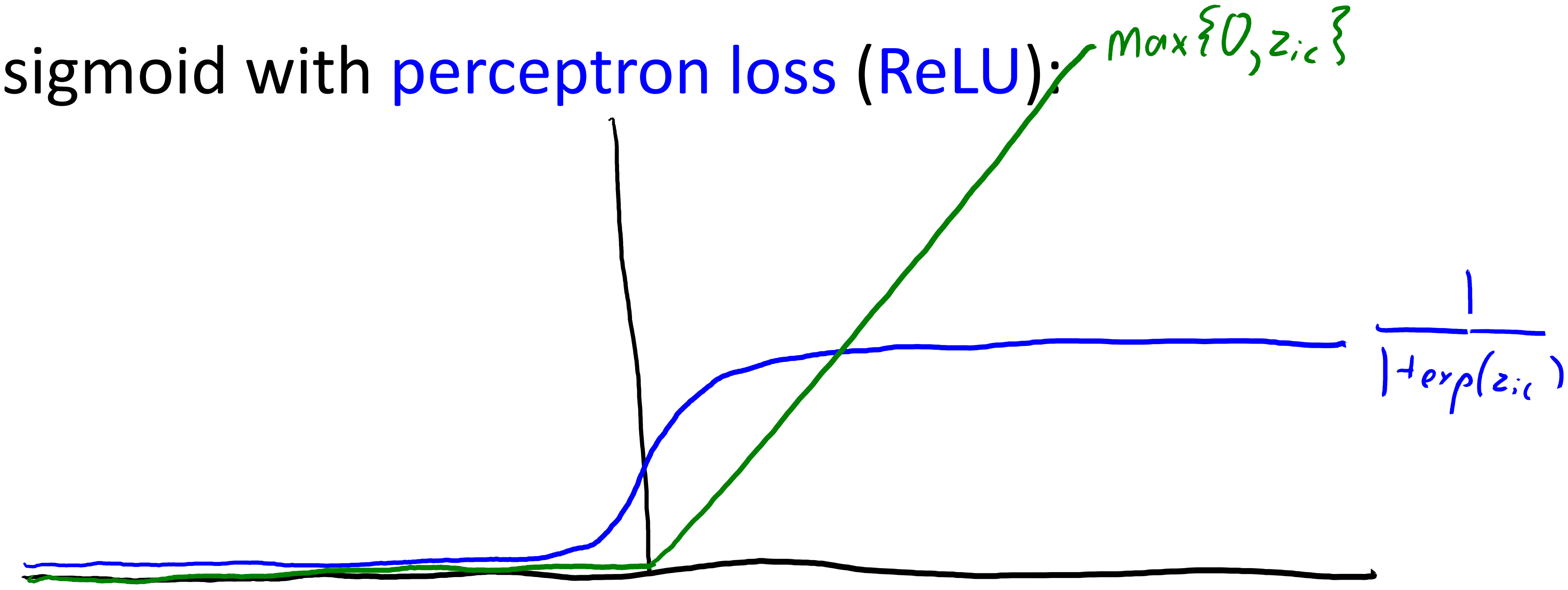
- Away from the origin, the gradient is nearly zero.
- The problem gets worse when you take the sigmoid of a sigmoid:



- In deep networks, many gradients can be nearly zero everywhere.

Rectified Linear Units (ReLU)

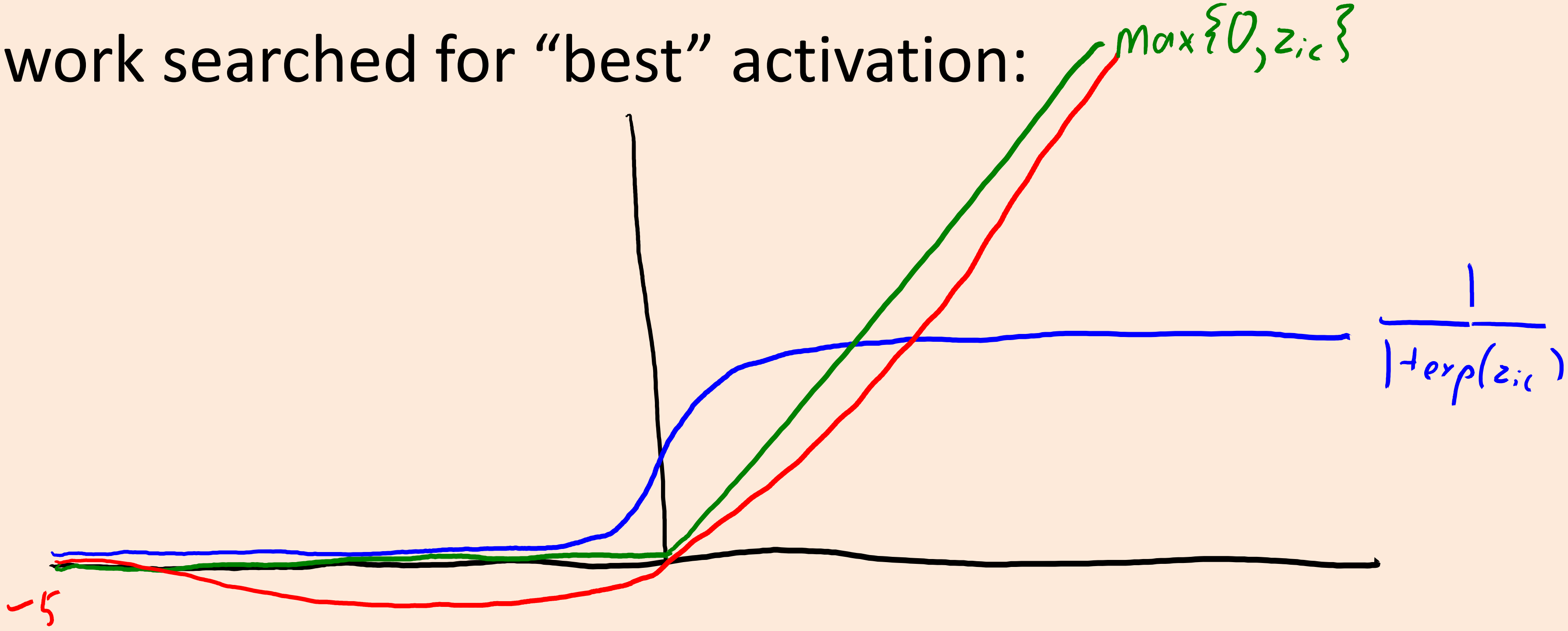
- Replace sigmoid with perceptron loss (ReLU):



- Just sets negative values z_{ic} to zero.
  - Fixes vanishing gradient problem.
  - Gives sparser activations.
  - Not really simulating binary signal, but could be simulating "rate coding".

"Swish" Activation

- Recent work searched for "best" activation:



- Found that $z_{ic}/(1+\exp(-z_{ic}))$ worked best ("swish" function).
  - A bit weird because it allows negative values and is non-monotonic.
  - But basically the same as ReLU when not close to 0.

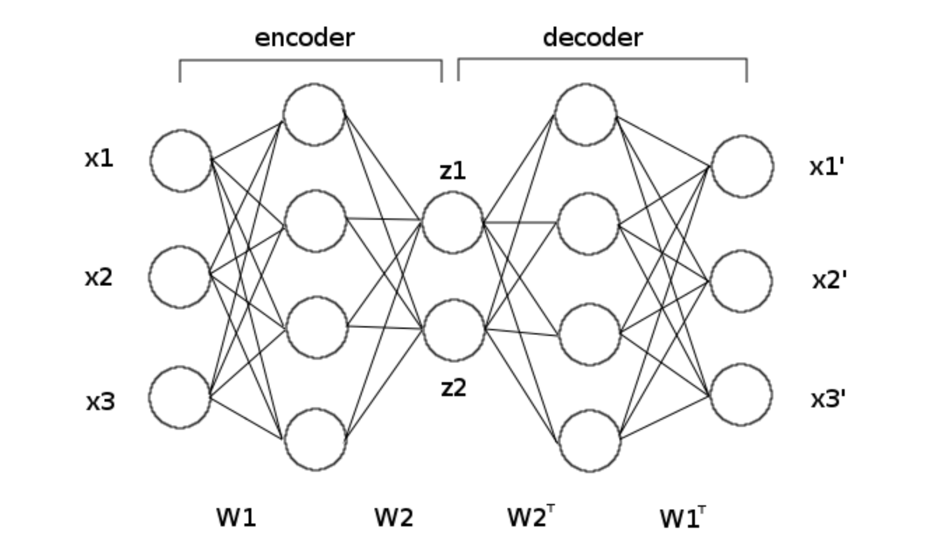
Summary

- Unprecedented performance on difficult pattern recognition tasks.
- Backpropagation computes neural network gradient via chain rule.
- Parameter initialization is crucial to neural net performance.
- Optimization and step size are crucial to neural net performance.
  - “Babysitting”, momentum.
- ReLU avoid “vanishing gradients”.

- Next time:
  - Regularization, and getting these working for vision problems.

Autoencoders

- Autoencoders are an unsupervised deep learning model:
  - Use the inputs as the output of the neural network.

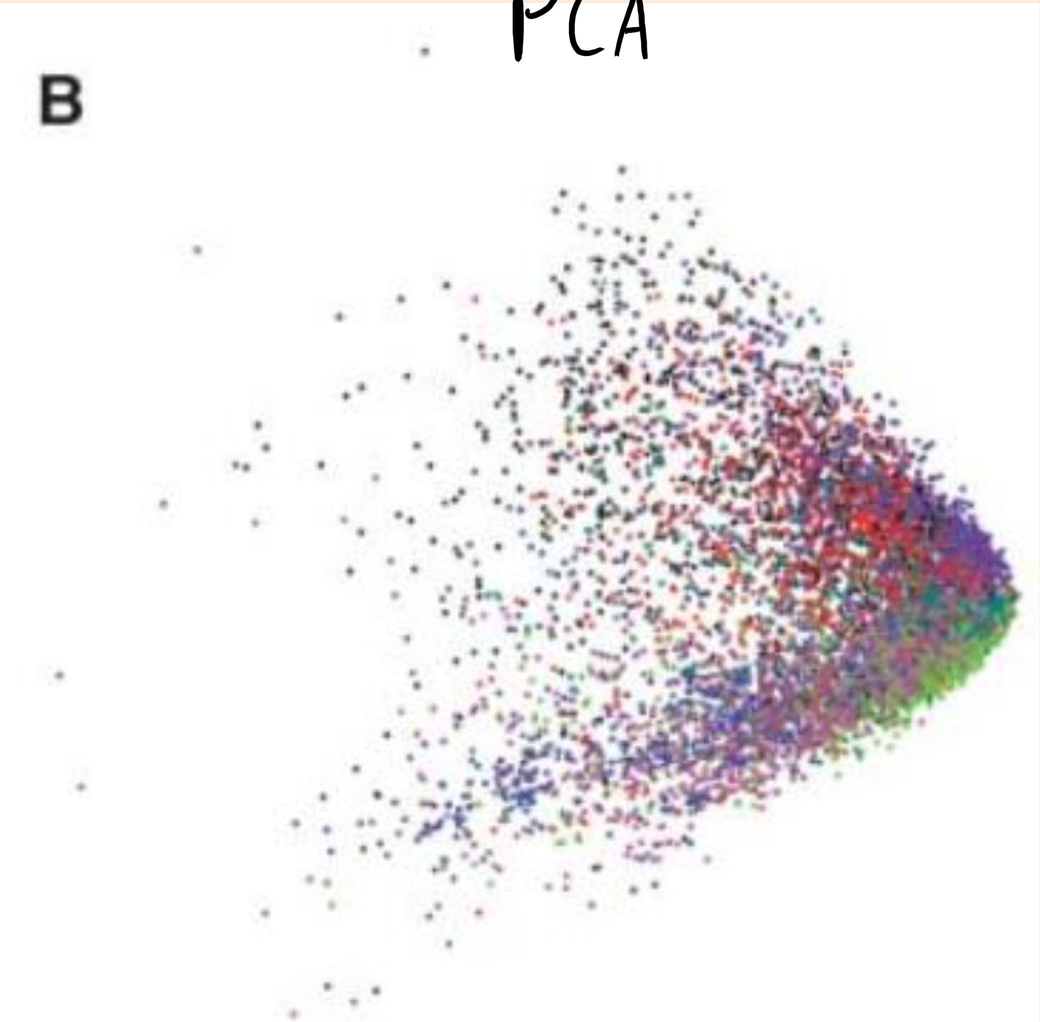


  - Middle layer could be latent features in non-linear latent-factor model.
    - Can do outlier detection, data compression, visualization, etc.
  - A non-linear generalization of PCA.
    - Equivalent to PCA if you don't have non-linearities.

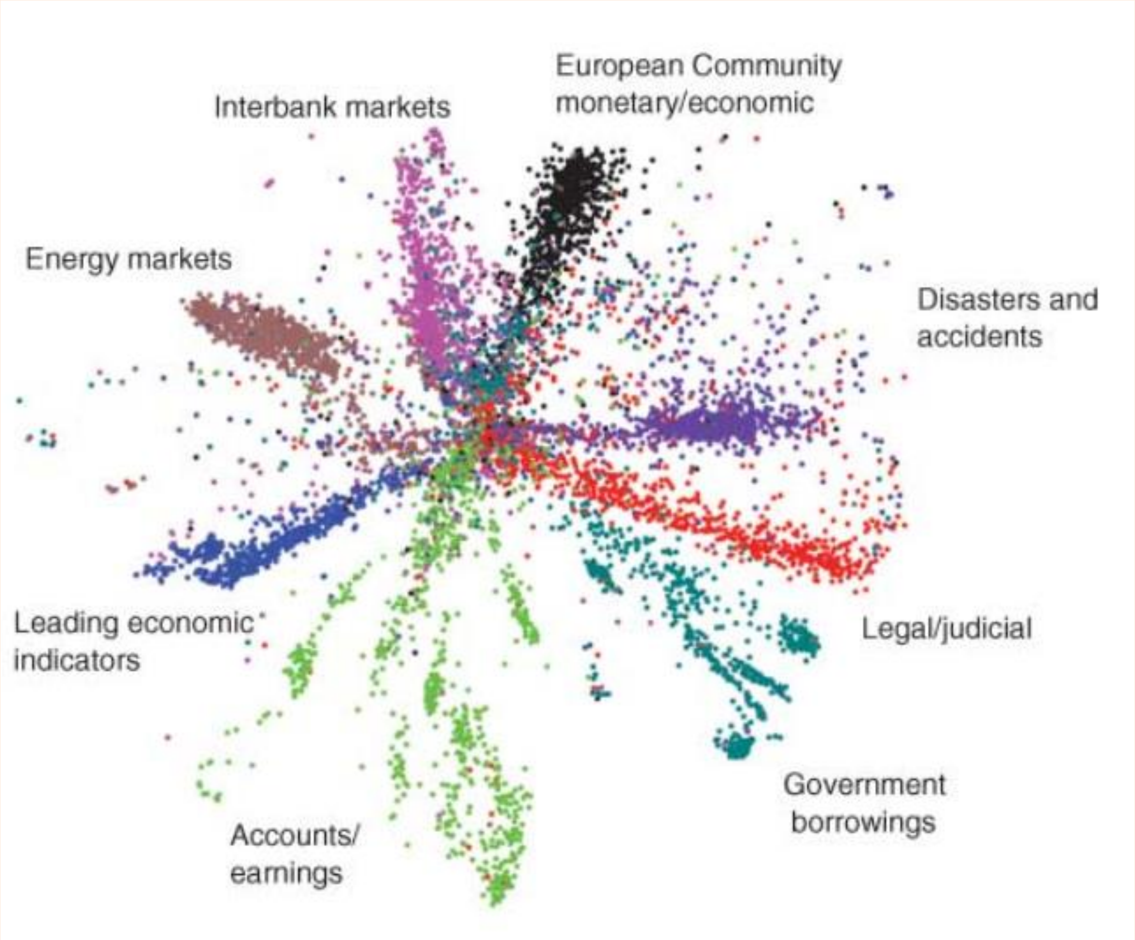
http://inspirehep.net/record/1252540/plots

Autoencoders

PCA



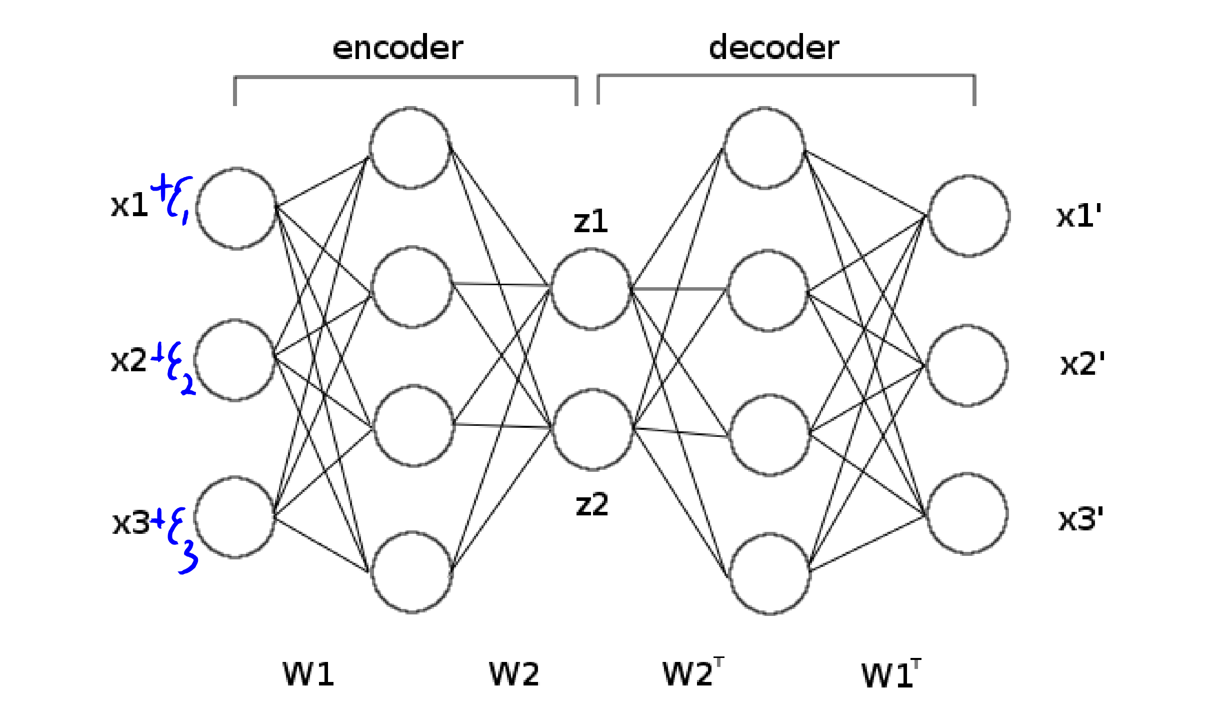
Autoencoder



https://www.cs.toronto.edu/~hinton/science.pdf

Denoising Autoencoder

- Denoising autoencoders add noise to the input:



  – Learns a model that can remove the noise.

http://inspirehep.net/record/1252540/plots