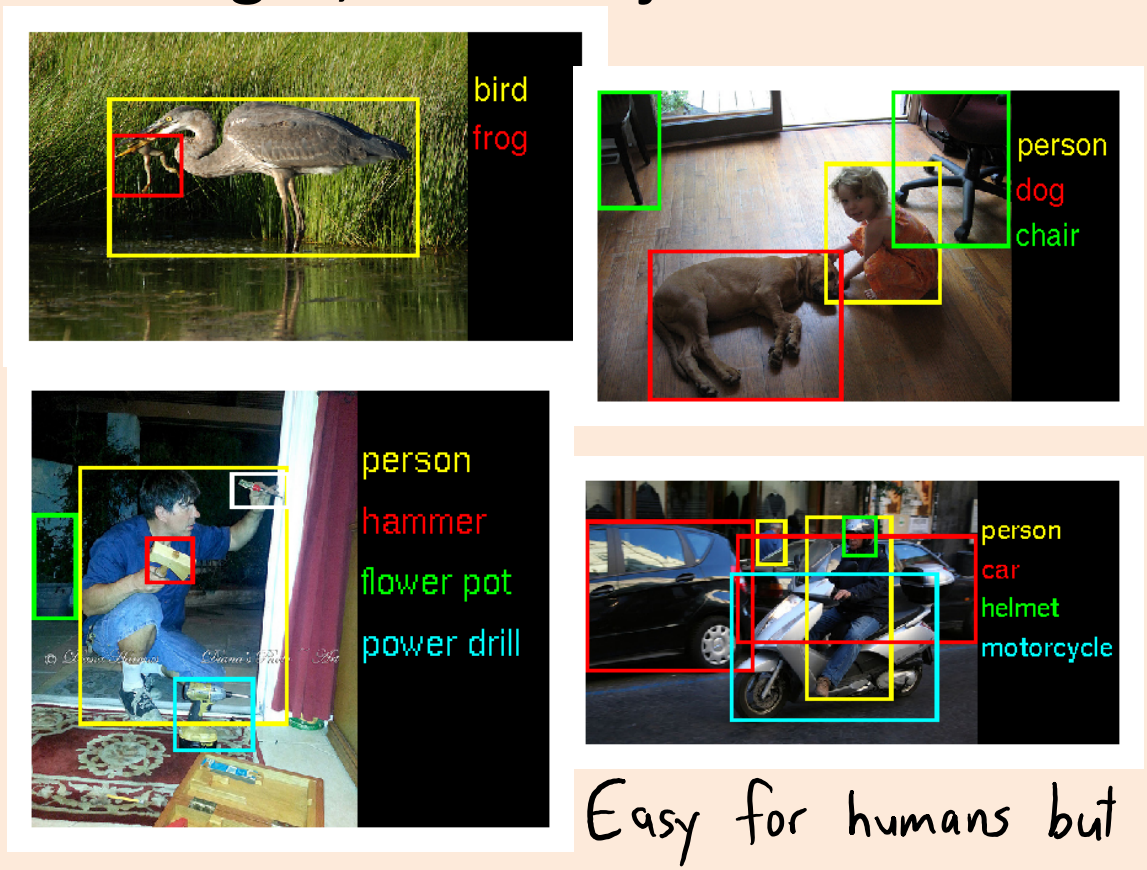
CPSC 340:
Machine Learning and Data Mining

More Deep Learning

Bonus slides

ImageNet Challenge

- Millions of labeled images, 1000 object classes.



Easy for humans but hard for computers.

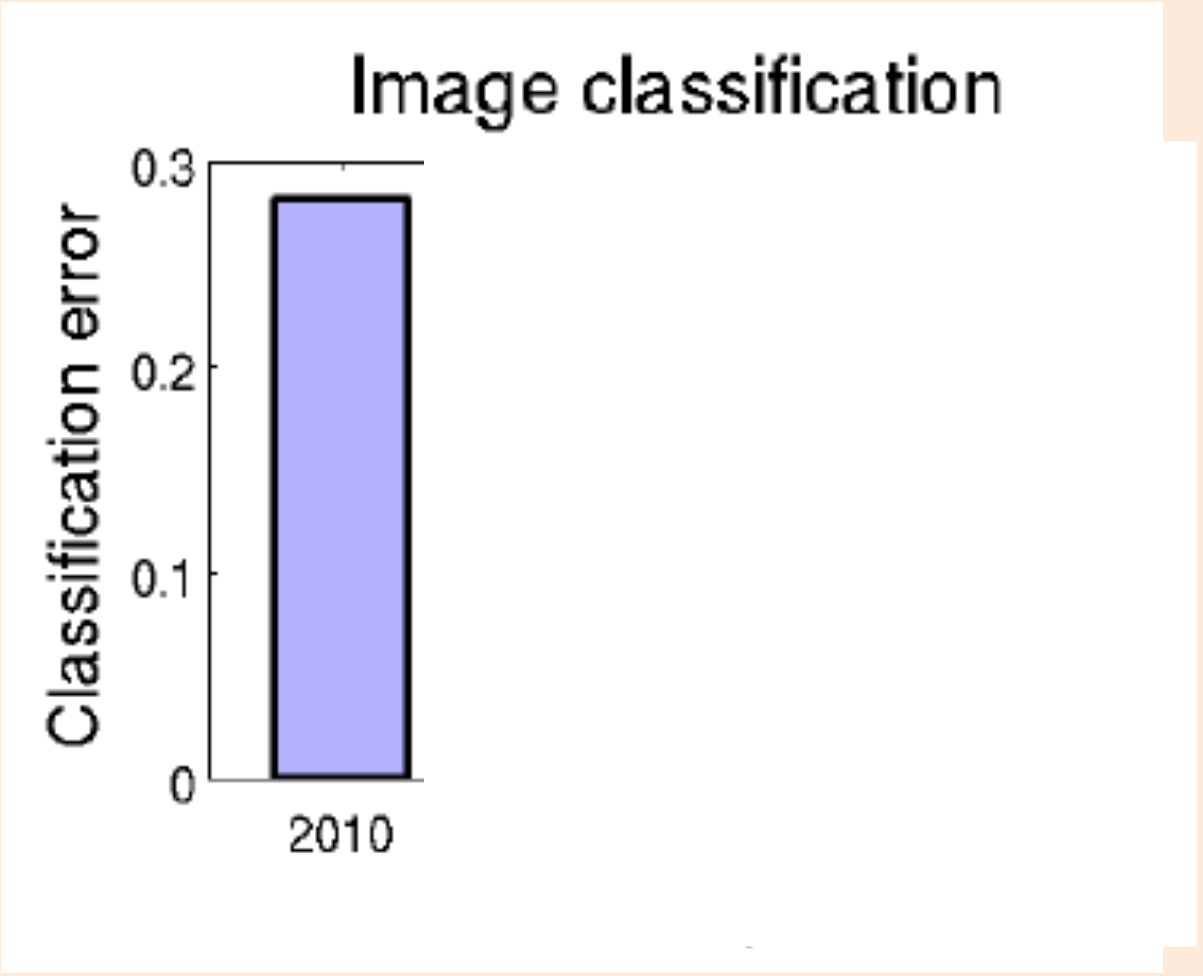
http://www.image-net.org/challenges/LSVRC/2014/

ImageNet Challenge

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

Syberian Husky

Canadian Husky



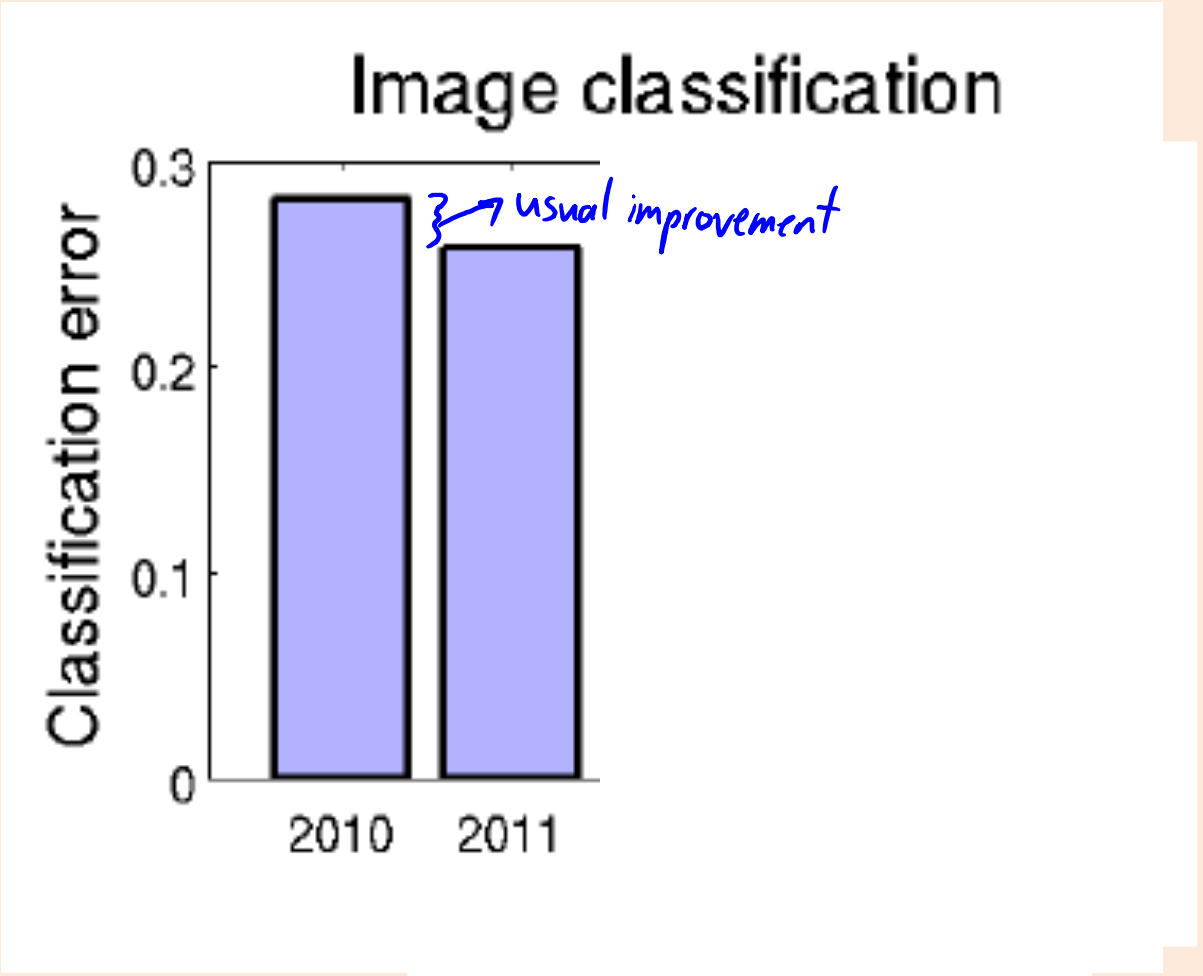
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

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

Syberian Husky

Canadian Husky



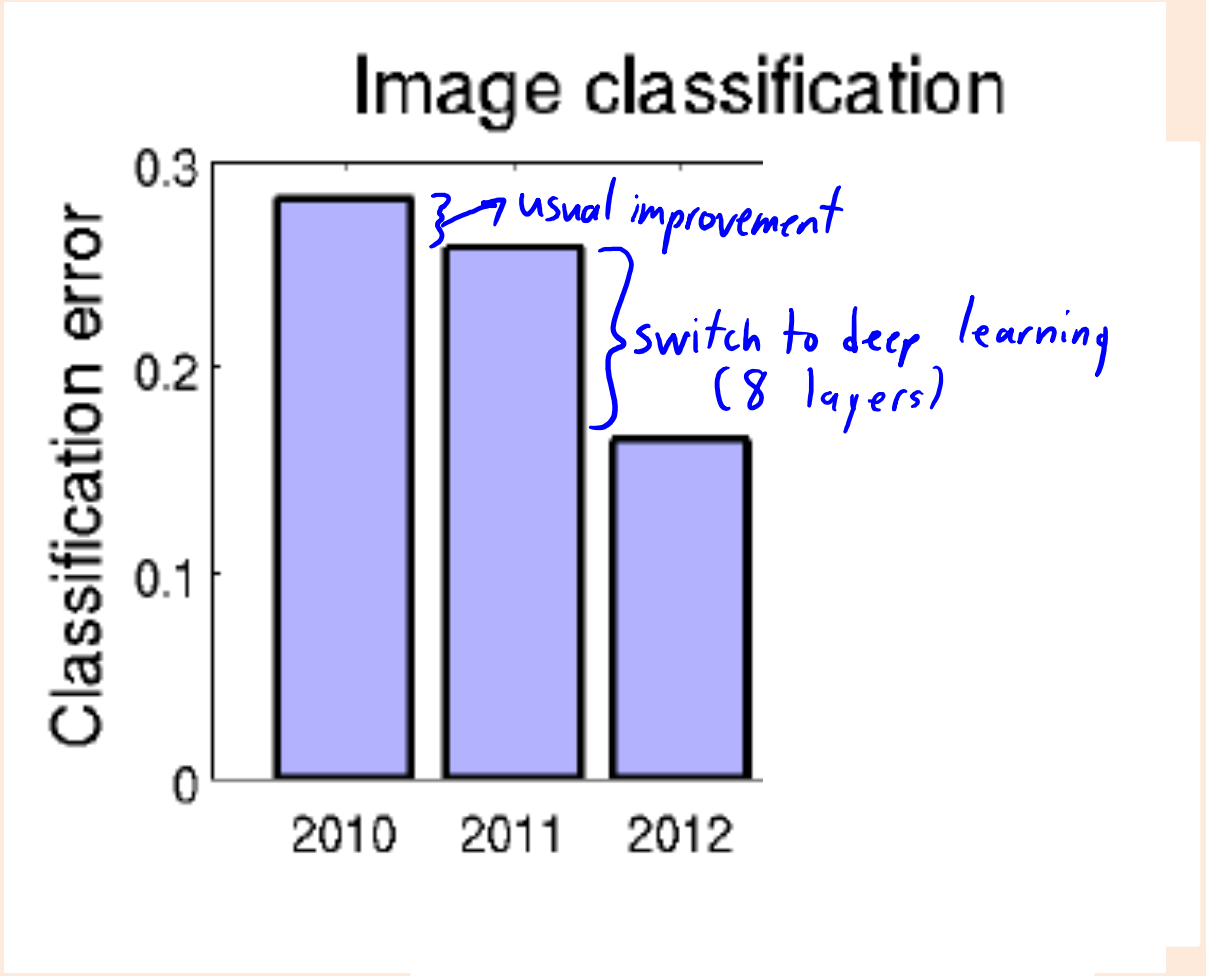
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

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

Syberian Husky

Canadian Husky



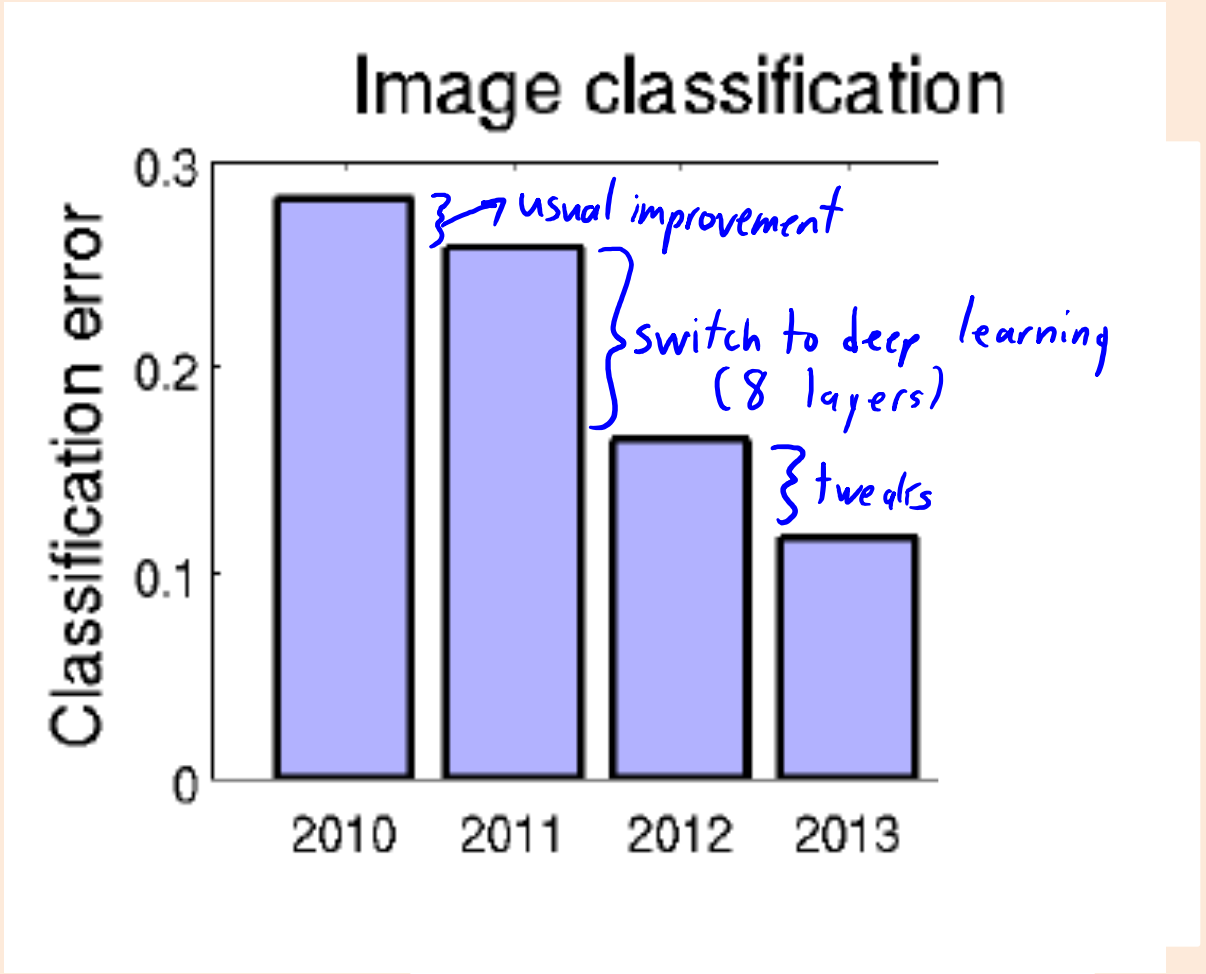
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

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

Syberian Husky

Canadian Husky



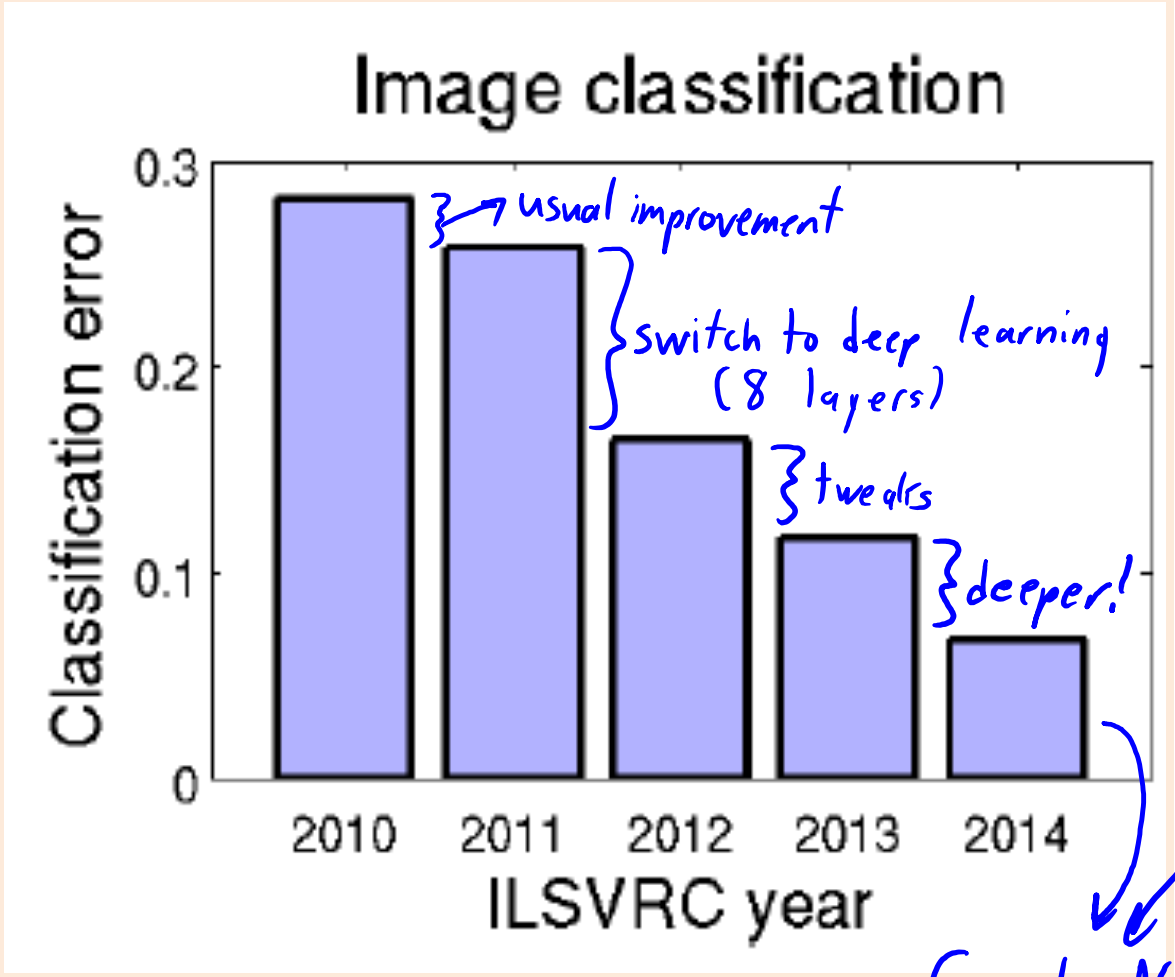
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

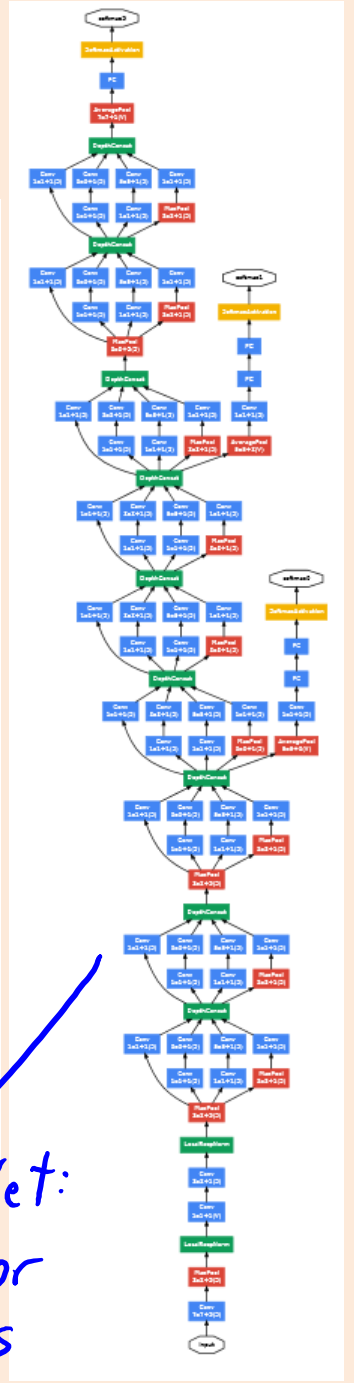
ImageNet Challenge

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

Syberian Husky

Canadian Husky

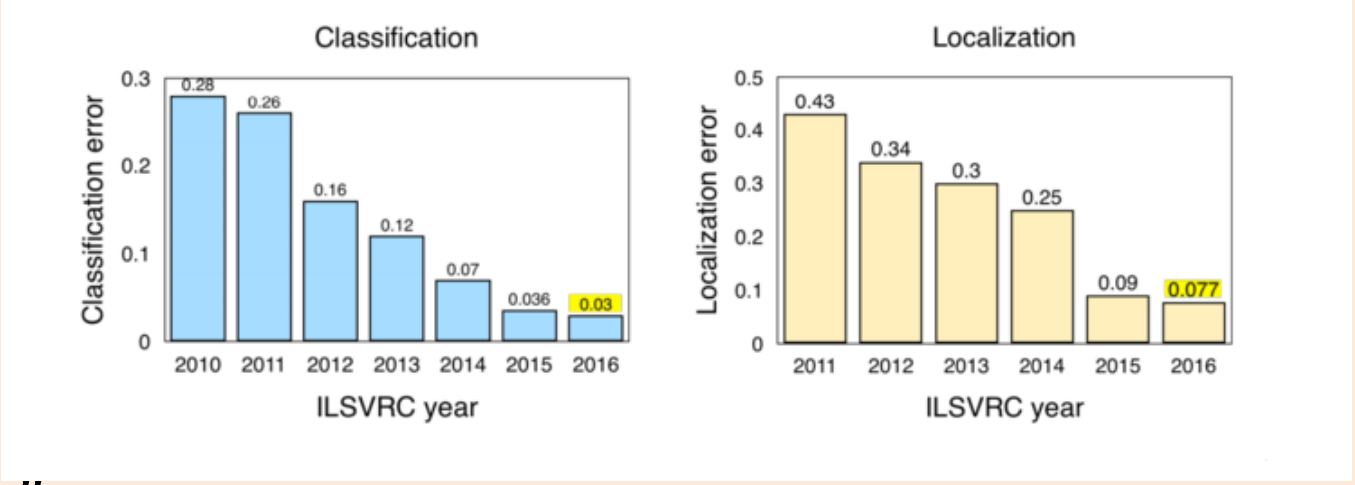




https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

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

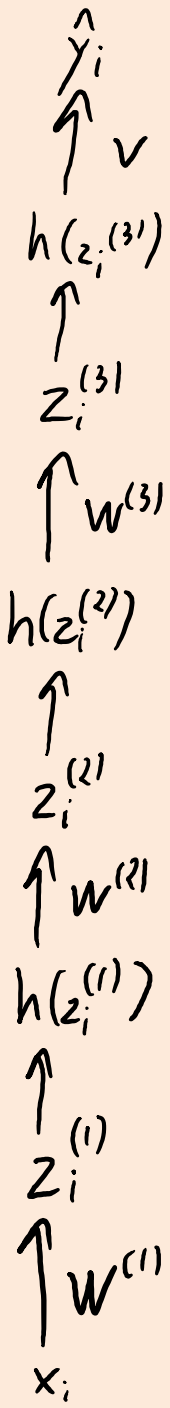
Backpropagation

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

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

$$\frac{\partial f}{\partial v} = (\hat{y}_i - y_i) h(w^{(3)} h(w^{(2)} h(w^{(1)} x_i))) = (\hat{y}_i - y_i) h(z_i^{(3)})$$

$$\frac{\partial f}{\partial w^{(3)}} = (\hat{y}_i - y_i) v h'(w^{(3)} h(w^{(2)} h(w^{(1)} x_i))) h(w^{(2)} h(w^{(1)} x_i)) = (\hat{y}_i - y_i) v h'(z_i^{(3)}) h(z_i^{(2)})$$



Backpropagation

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

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

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

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

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

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

Backpropagation

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

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

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

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

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

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

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

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

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

  - 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.

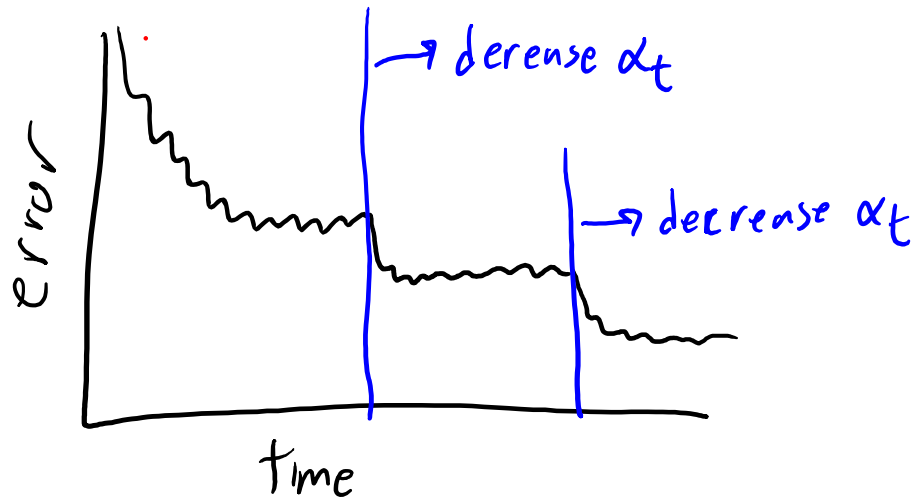
Parameter Initialization

- 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

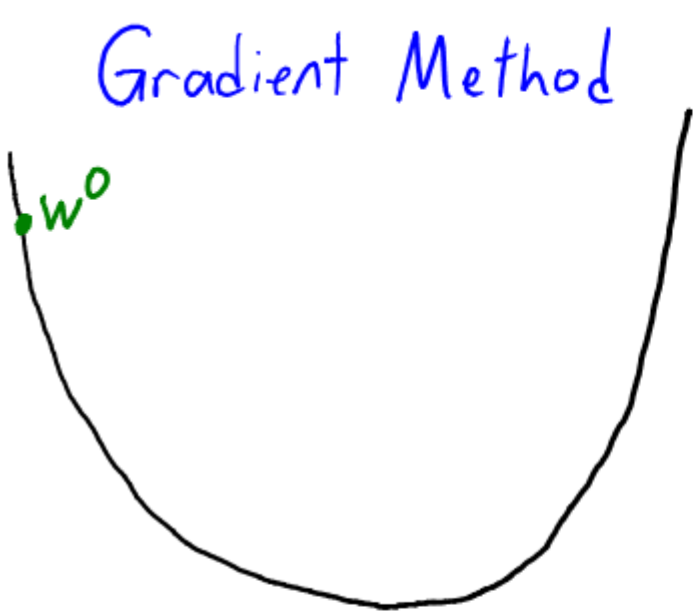
- Stochastic gradient is very sensitive to the step size in deep models.
- Bias step-size multiplier: use bigger step-size for the bias variables.
- Momentum (stochastic version of "heavy-ball" algorithm):
  - Add term that moves in previous direction:

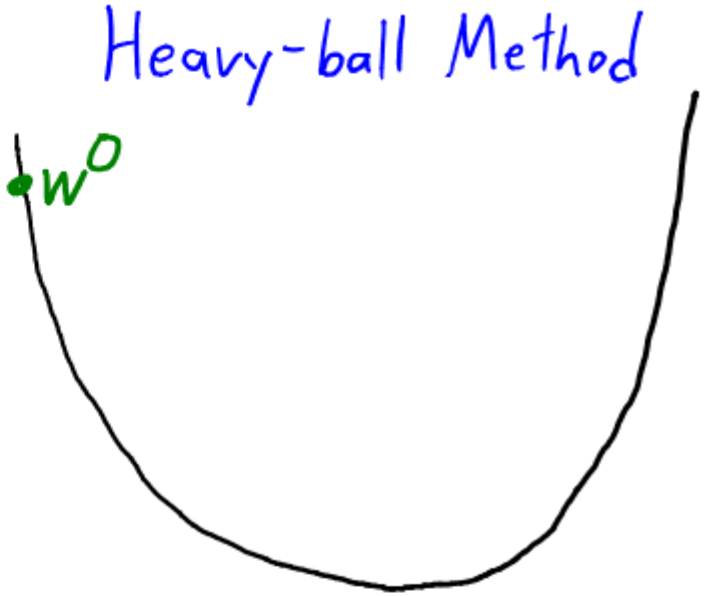
$$w^{t+1} = w^t - \alpha^t \nabla f_i(w^t) + \beta^t (w^t - w^{t-1})$$

keep going in the old direction

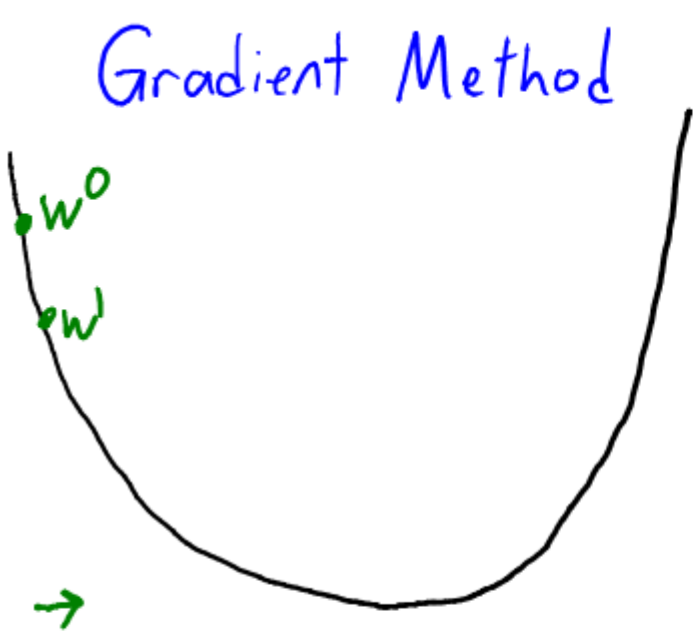
  - Usually $\beta^t = 0.9$.

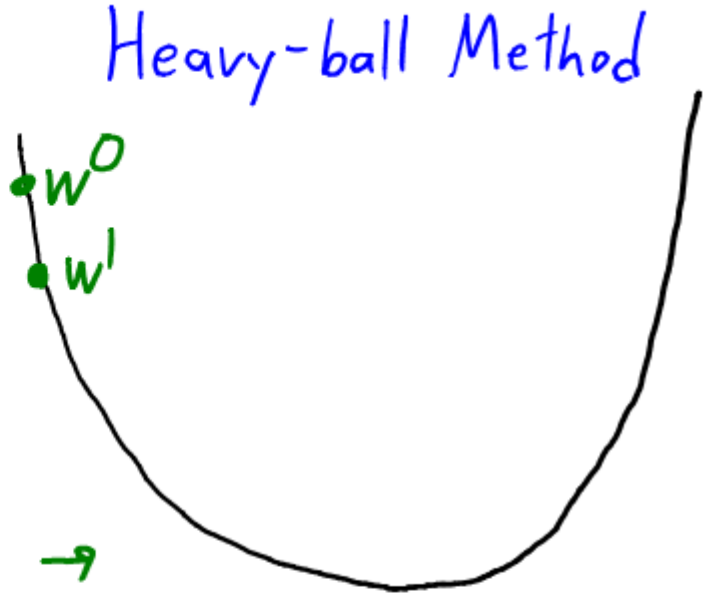
Gradient Descent vs. Heavy-Ball Method



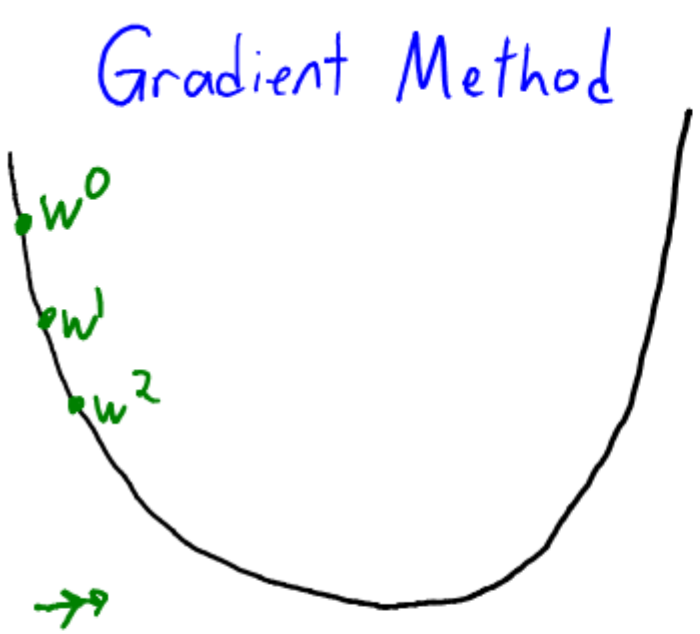


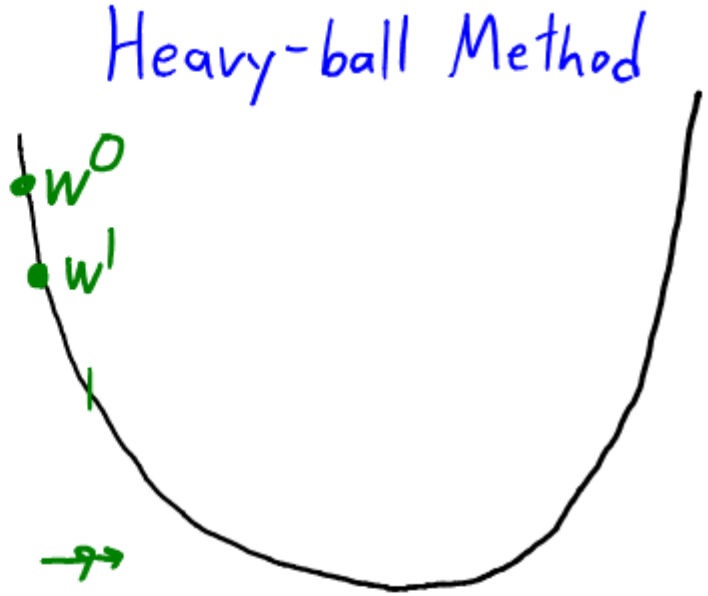
Gradient Descent vs. Heavy-Ball Method



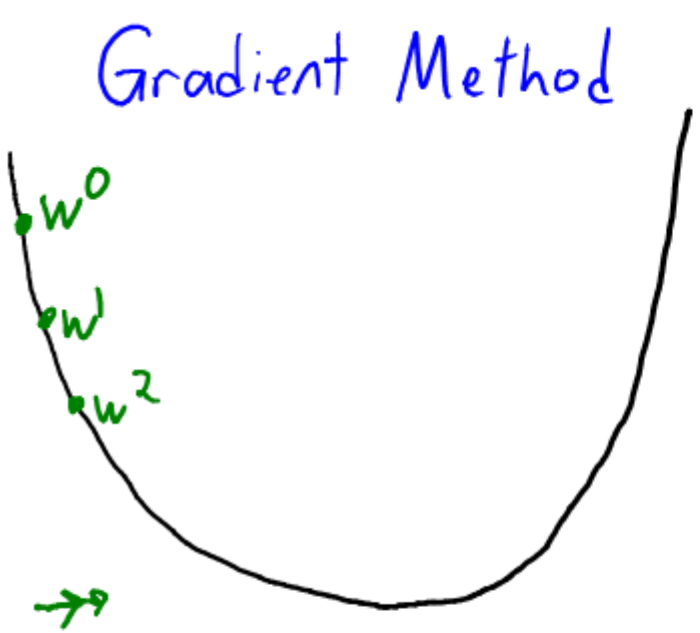


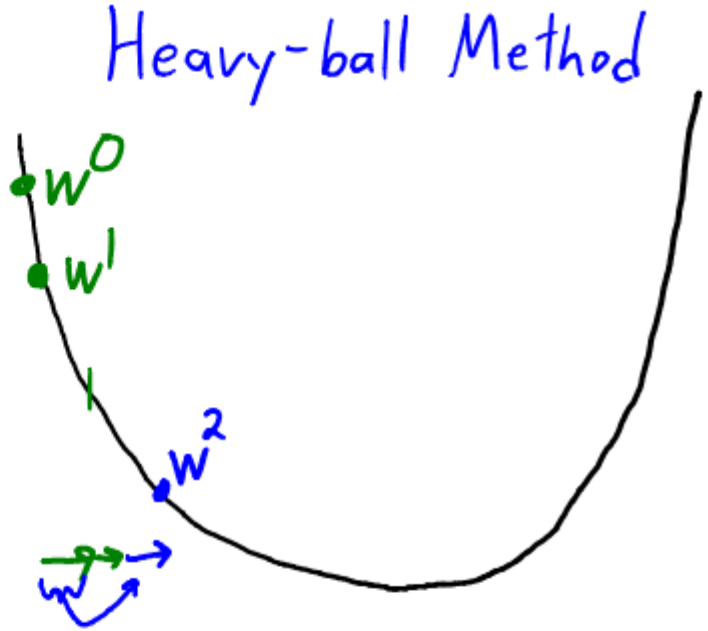
Gradient Descent vs. Heavy-Ball Method



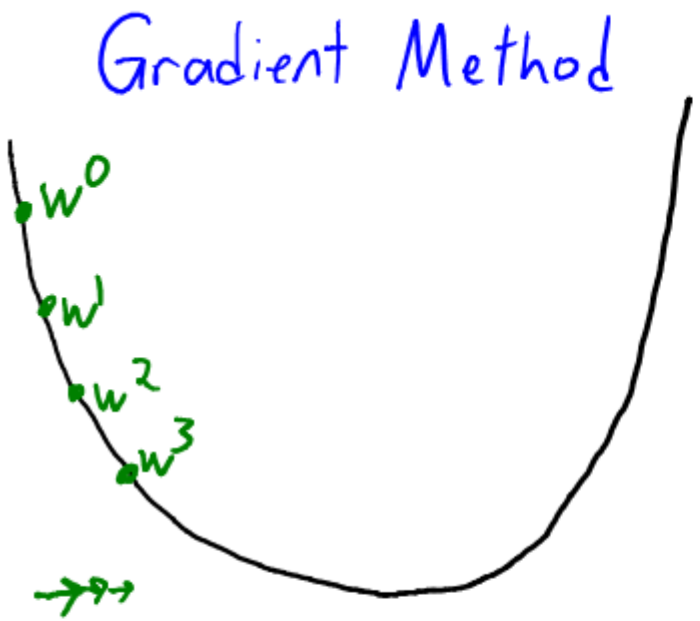


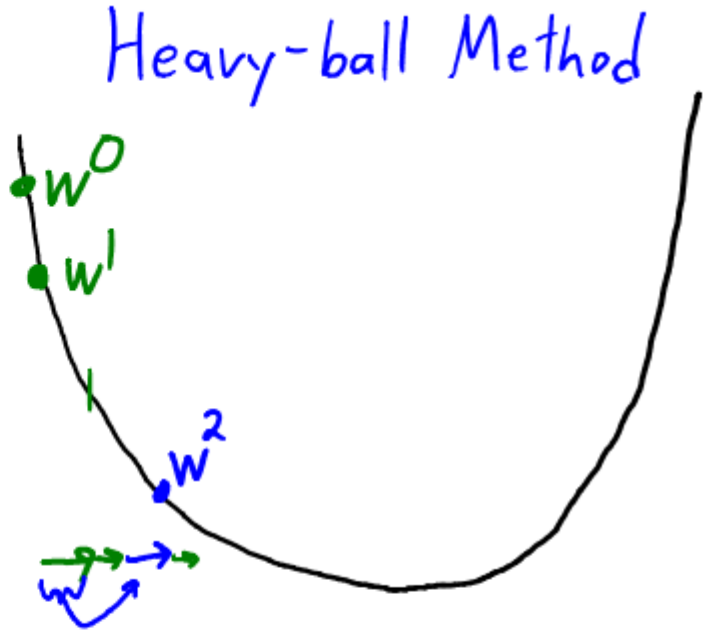
Gradient Descent vs. Heavy-Ball Method



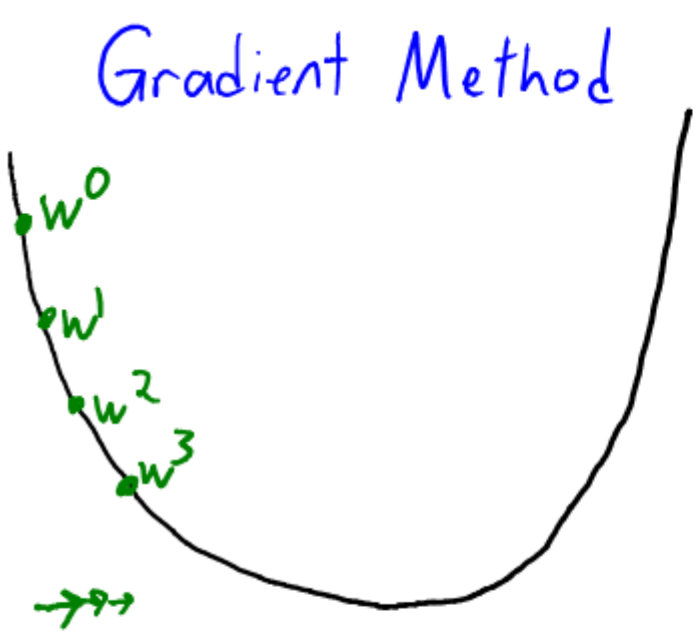


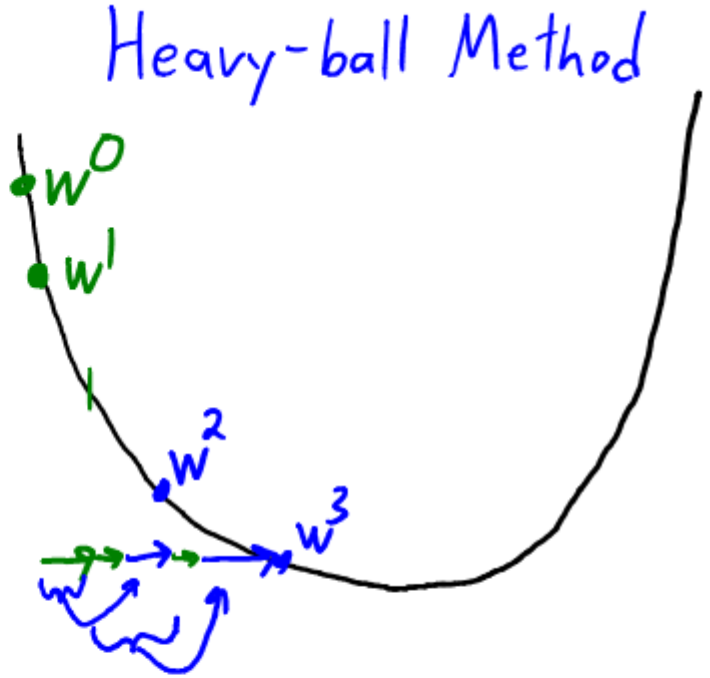
Gradient Descent vs. Heavy-Ball Method



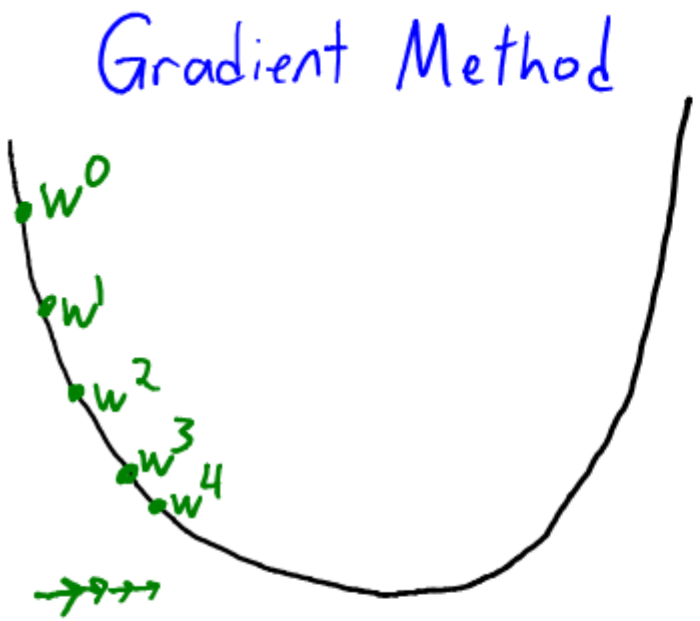


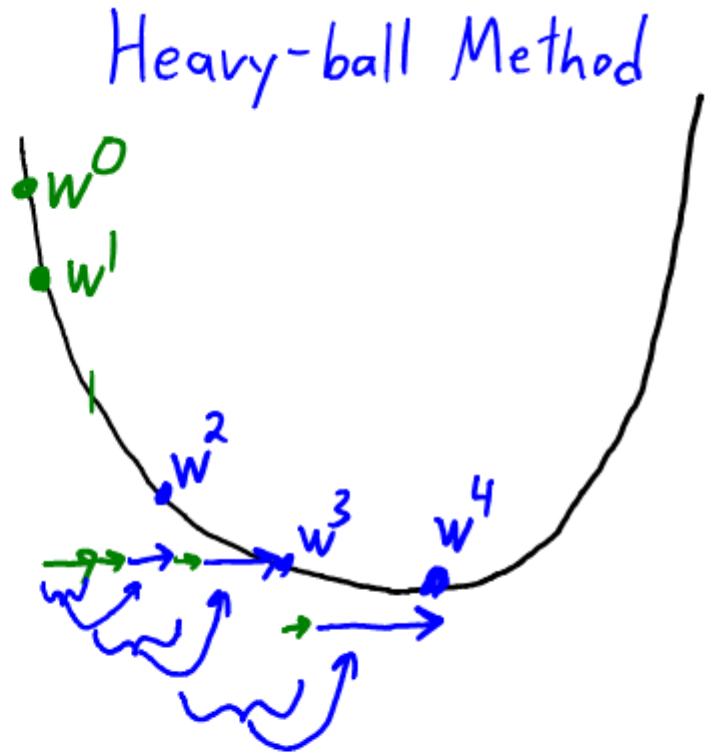
Gradient Descent vs. Heavy-Ball Method



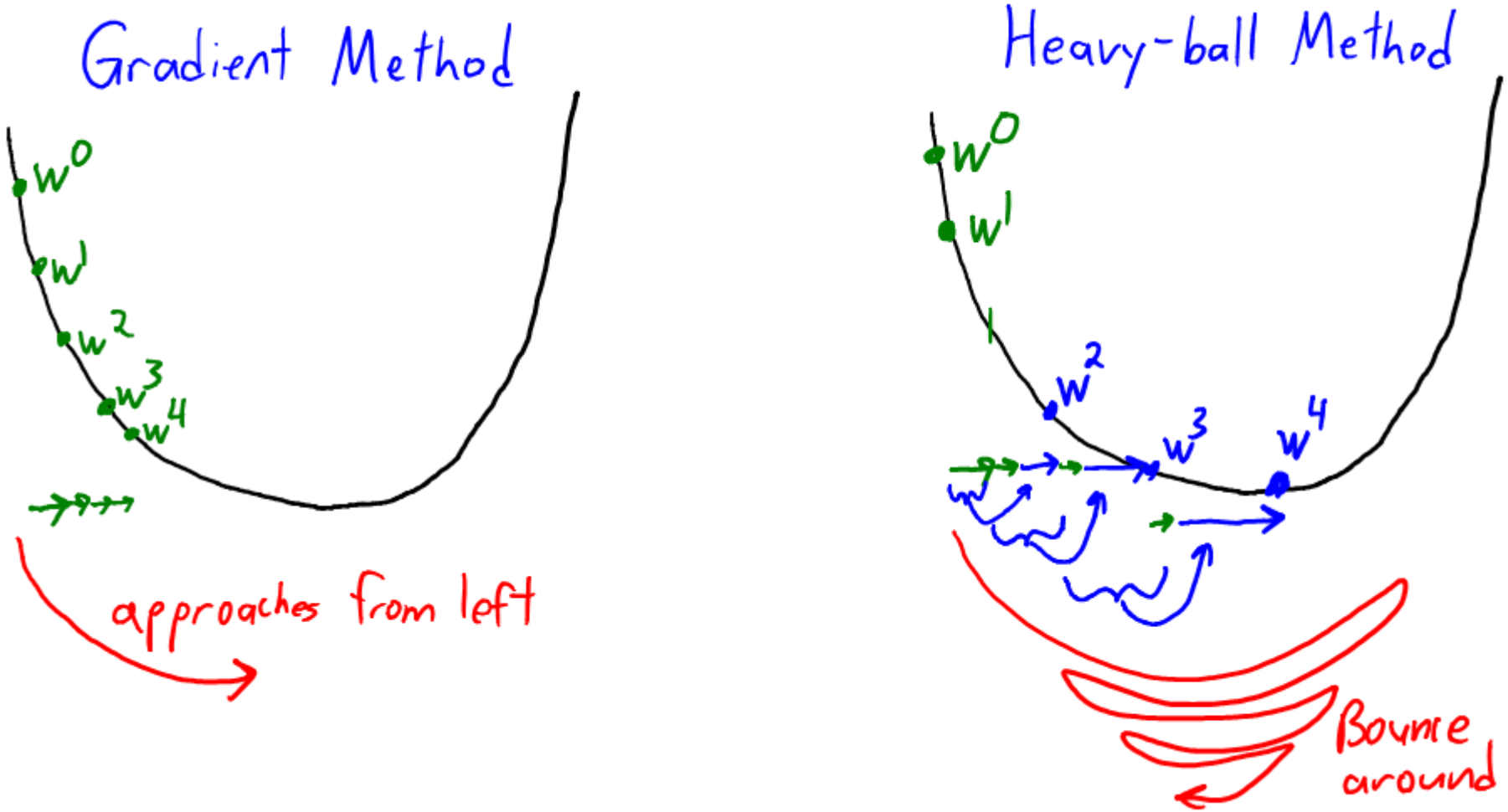


Gradient Descent vs. Heavy-Ball Method





Gradient Descent vs. Heavy-Ball Method



Setting the Step-Size

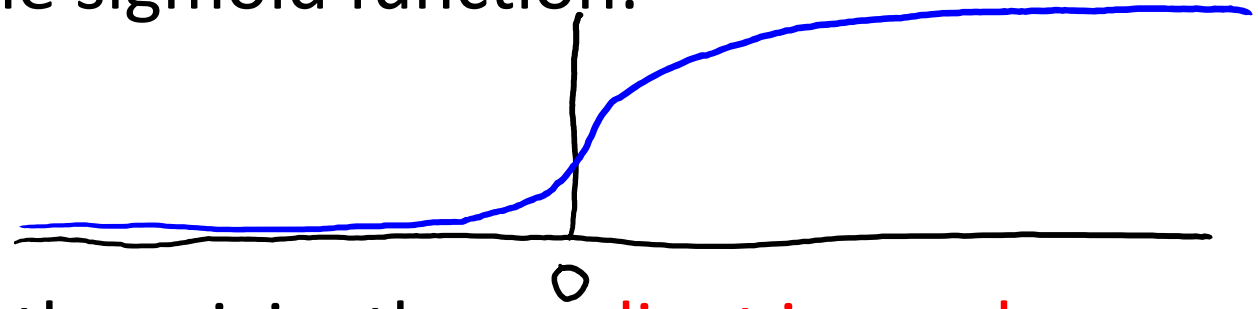
- Automatic method to set step size is Bottou trick:
  1. Grab a small set of training examples (maybe 5% of total).
  2. Do a binary search for a step size that works well on them.
  3. Use this step size for a long time (or slowly decrease it from there).

- 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?

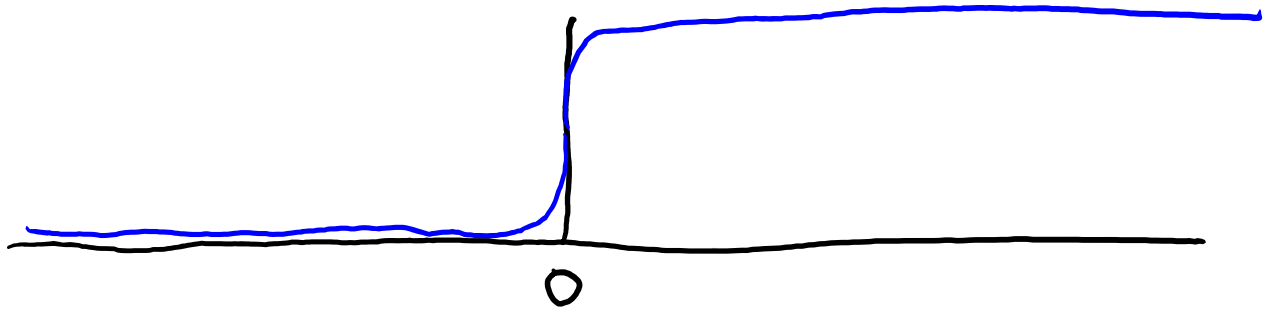
- Another recent trick is batch normalization:
  - Try to "standardize" the hidden units within the random samples as we go.
  - Held as example of deep learning "alchemy" (blog post here about deep learning claims).
    - Sounds science-ey and often works but little theoretical justification/understanding.

Vanishing Gradient Problem

- Consider the sigmoid function:



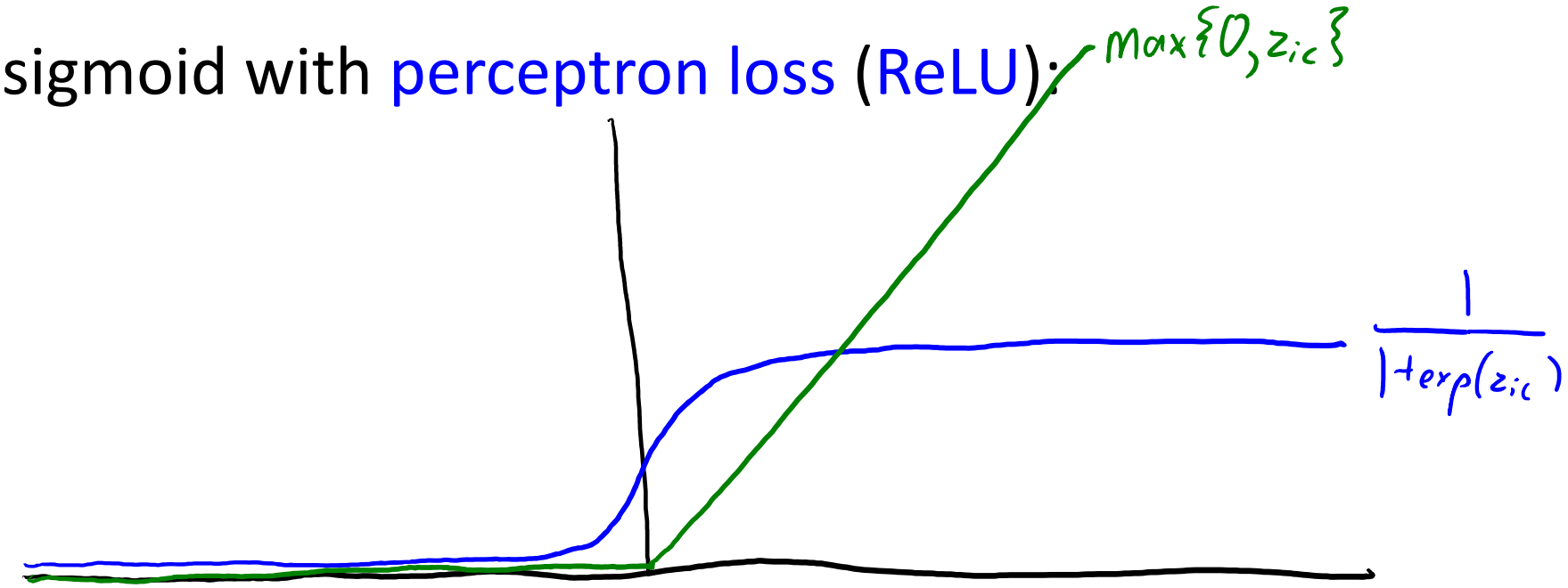
- Away from the origin, the gradient is nearly zero.
- The problem gets worse when you take the sigmoid of a sigmoid:



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

Rectified Linear Units (ReLU)

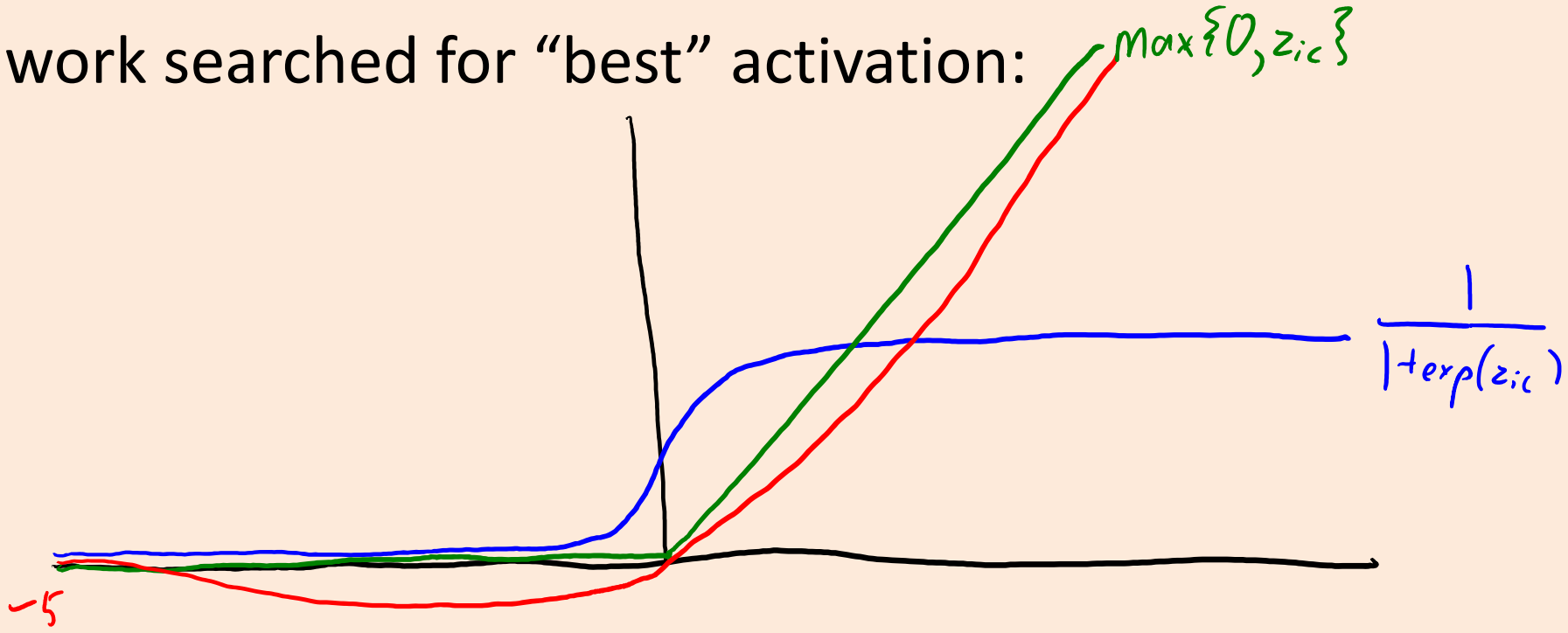
- Replace sigmoid with perceptron loss (ReLU):



- 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” Activiation

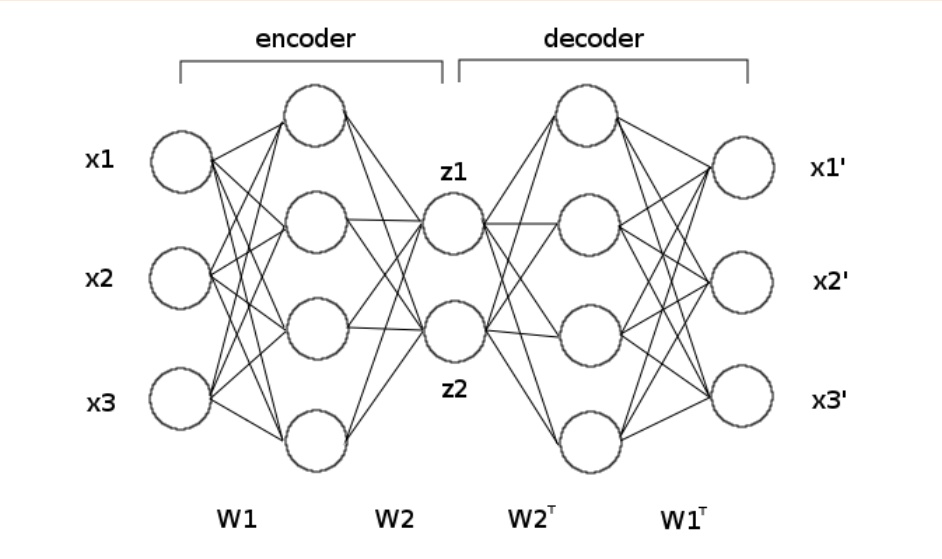
- Recent work searched for “best” activation:



- 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.

Autoencoders

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

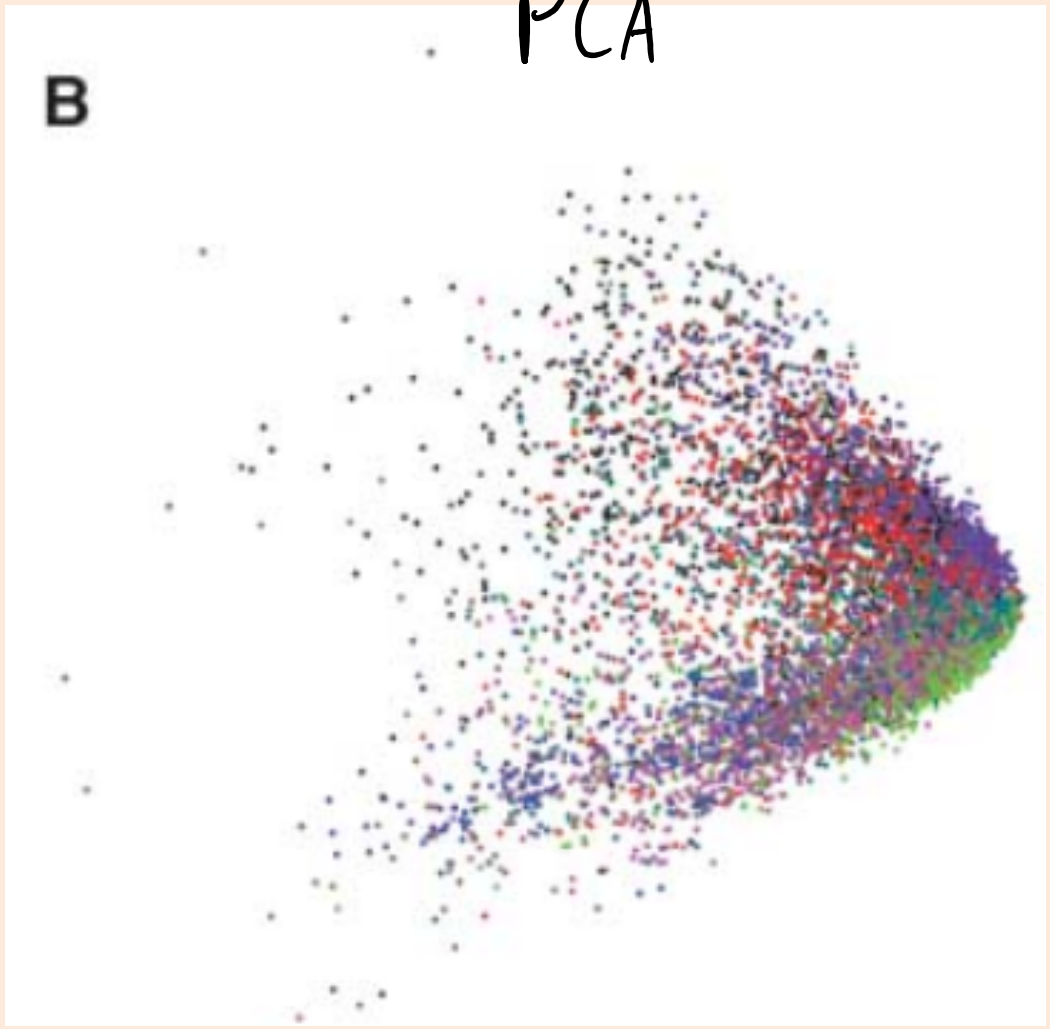


  - Middle layer could be latent features in non-linear latent-factor model.
    - Can do outlier detection, data compression, visualization, etc.
  - A non-linear generalization of PCA.
    - Equivalent to PCA if you don't have non-linearities.

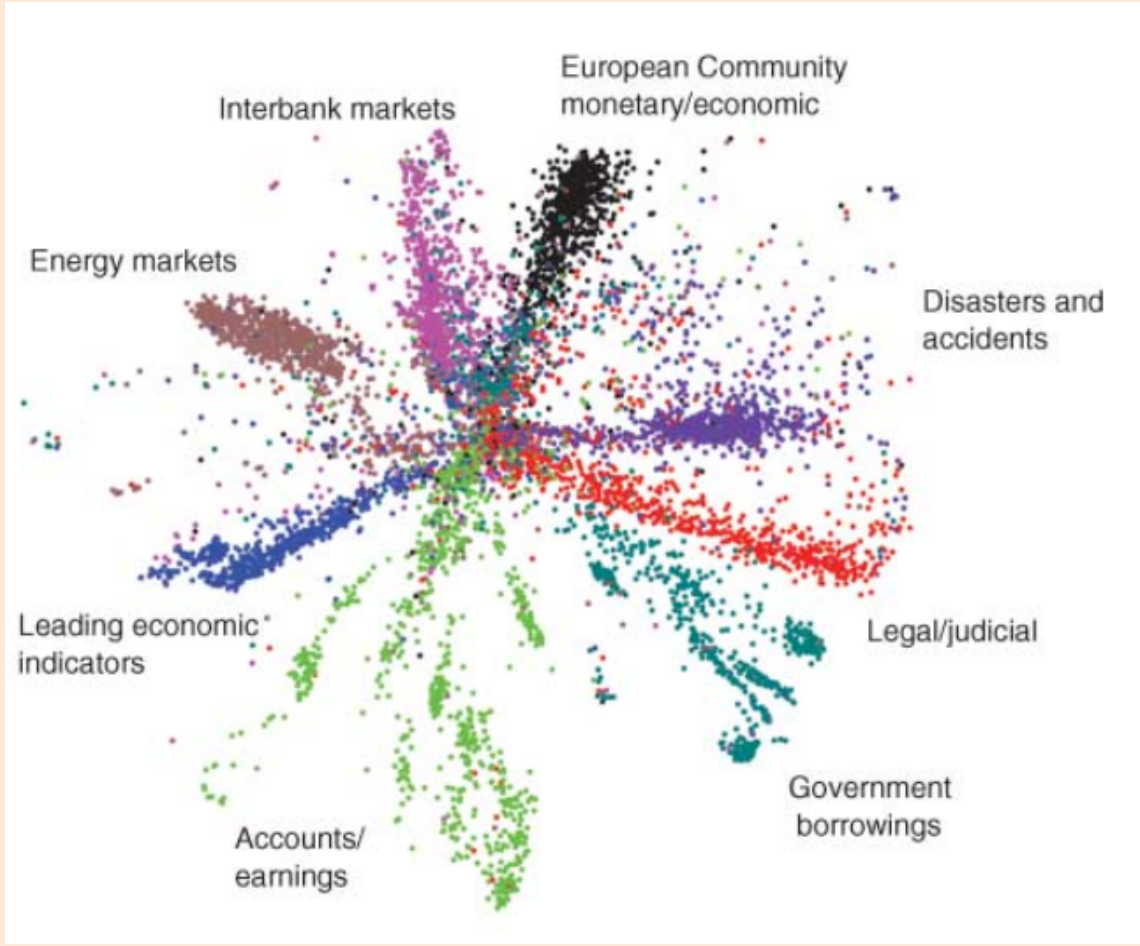
http://inspirehep.net/record/1252540/plots

Autoencoders

PCA



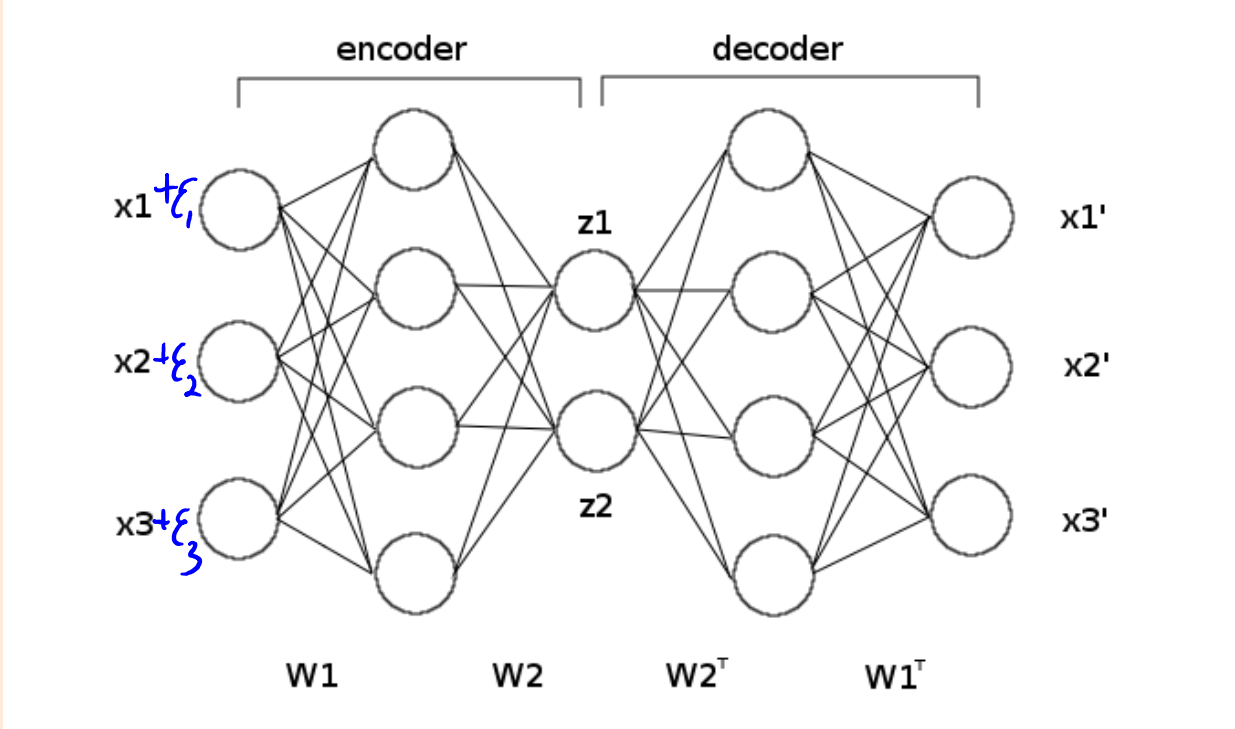
Autoencoder



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

“Residual” Networks (ResNets)

- Impactful recent idea is residual networks (ResNets):

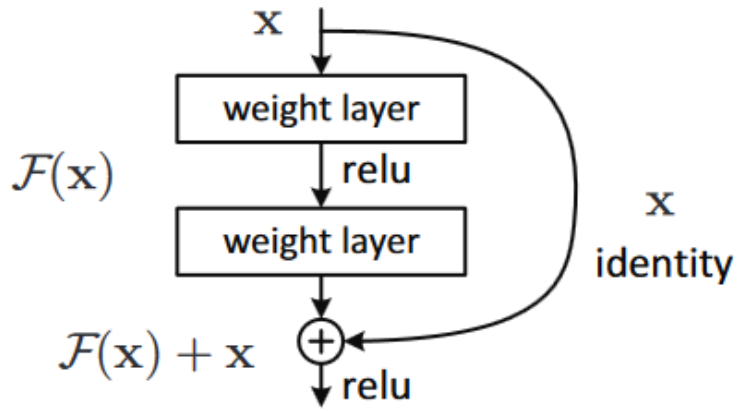


Figure 2. Residual learning: a building block.

  - You can take previous (non-transformed) layer as input to current layer.
    - Also called “skip connections” or “highway networks”.
  - Non-linear part of the network only needs to model residuals.
    - Non-linear parts are just “pushing up or down” a linear model in various places.
  - This was a key idea behind first methods that used 100+ layers.
    - Evidence that biological networks have skip connections like this.

https://en.wikipedia.org/wiki/Residual_neural_network

DenseNet

- More recent variation is “DenseNets”:
  - Each layer can see all the values from many previous layers.
  - Gets rid of vanishing gradients.
  - May get same performance with fewer parameters/layers.

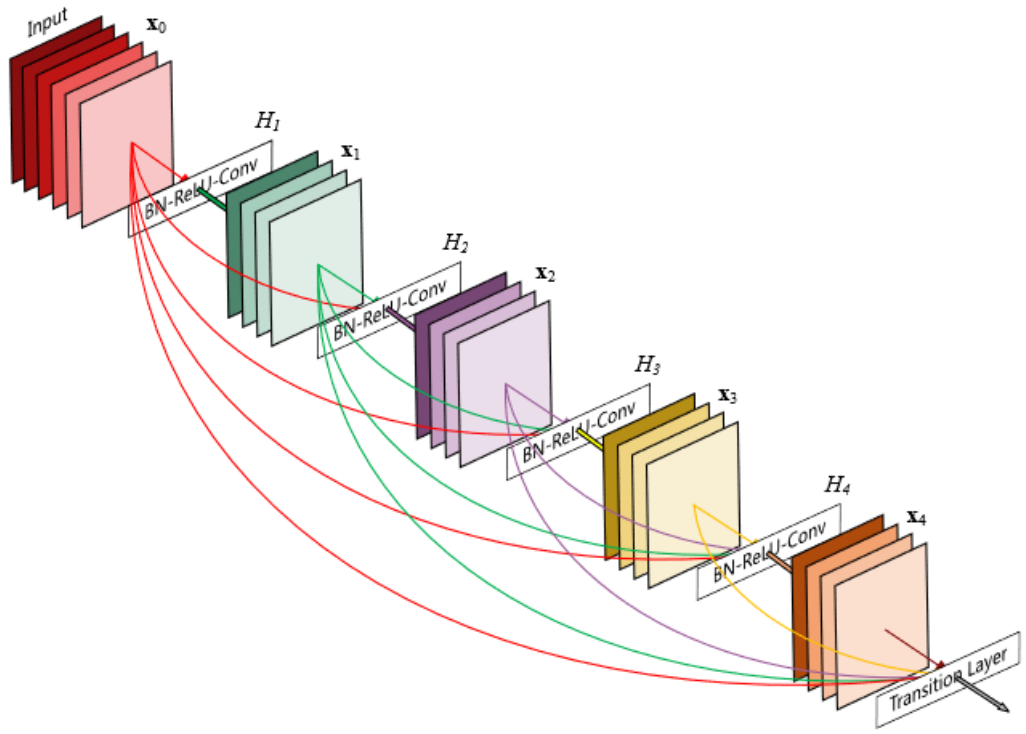


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

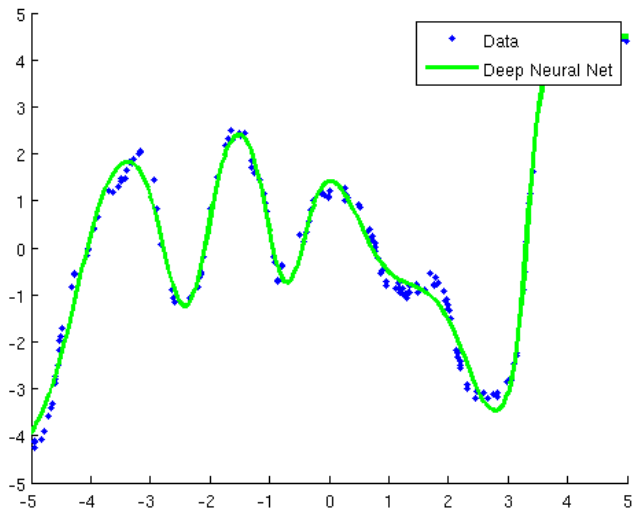
https://arxiv.org/pdf/1512.03385v1.pdf

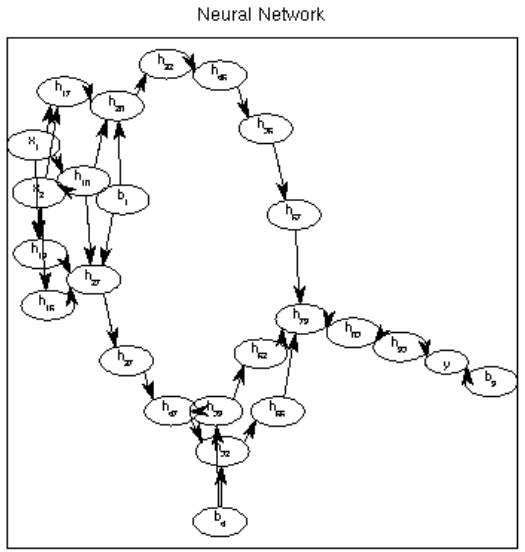
Standard Regularization

- Traditionally, we've added our usual L2-regularizers:

$$f(v, W^{(3)}, W^{(2)}, W^{(1)}) = \frac{1}{2}\sum_{i=1}^{n}(v^T h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) - y_i)^2 + \frac{\lambda_4}{2}\|v\|^2 + \frac{\lambda_3}{2}\|W^{(3)}\|_F^2 + \frac{\lambda_2}{2}\|W^{(2)}\|_F^2 + \frac{\lambda_1}{2}\|W^{(1)}\|_F^2$$

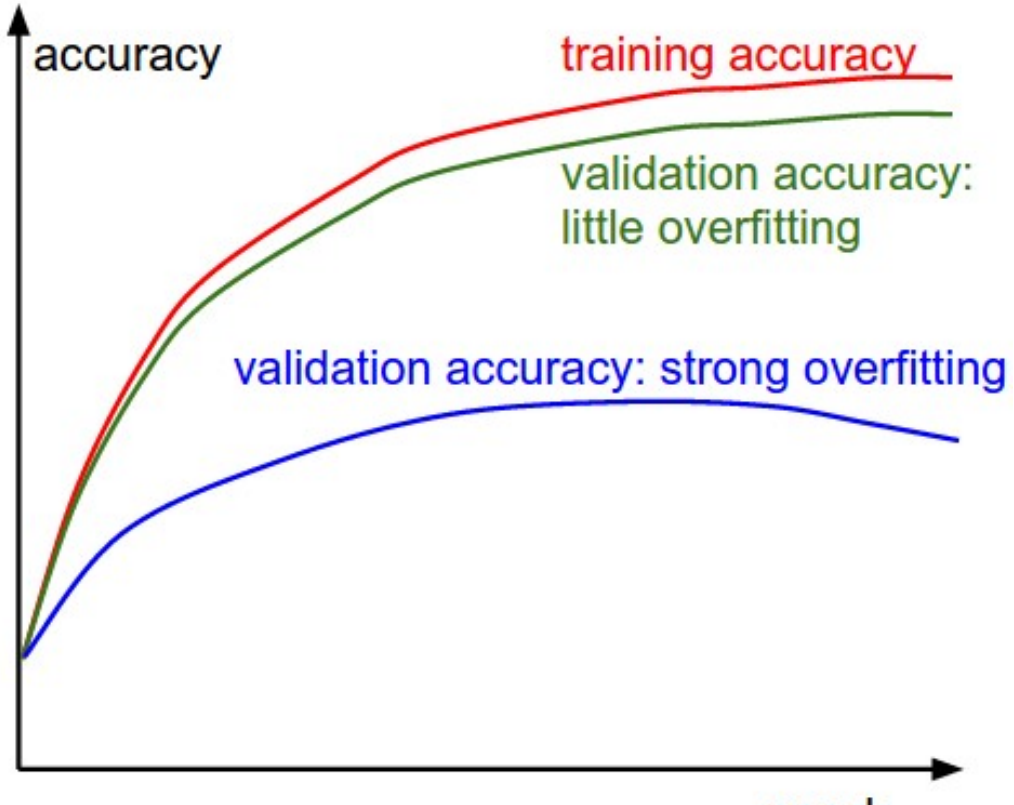
- L2-regularization often called "weight decay" in this context.
  - Could also use L1-regularization: gives sparse network.





Early Stopping

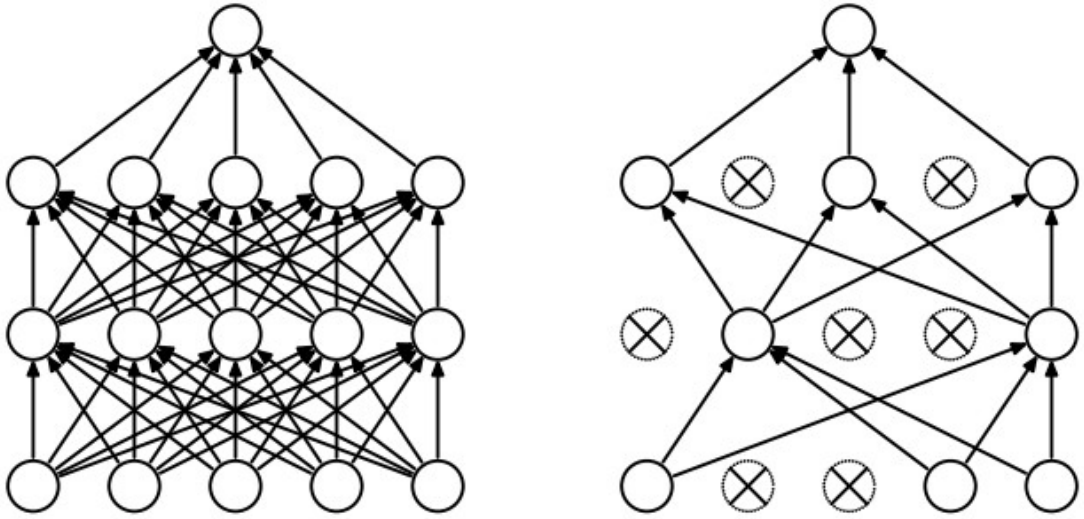
- Another common type of regularization is "early stopping":
  - Monitor the validation error as we run stochastic gradient.
  - Stop the algorithm if validation error starts increasing.



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

Dropout

- Dropout is a more recent form of explicit regularization:
  - On each iteration, randomly set some x_i and z_i to zero (often use 50%).



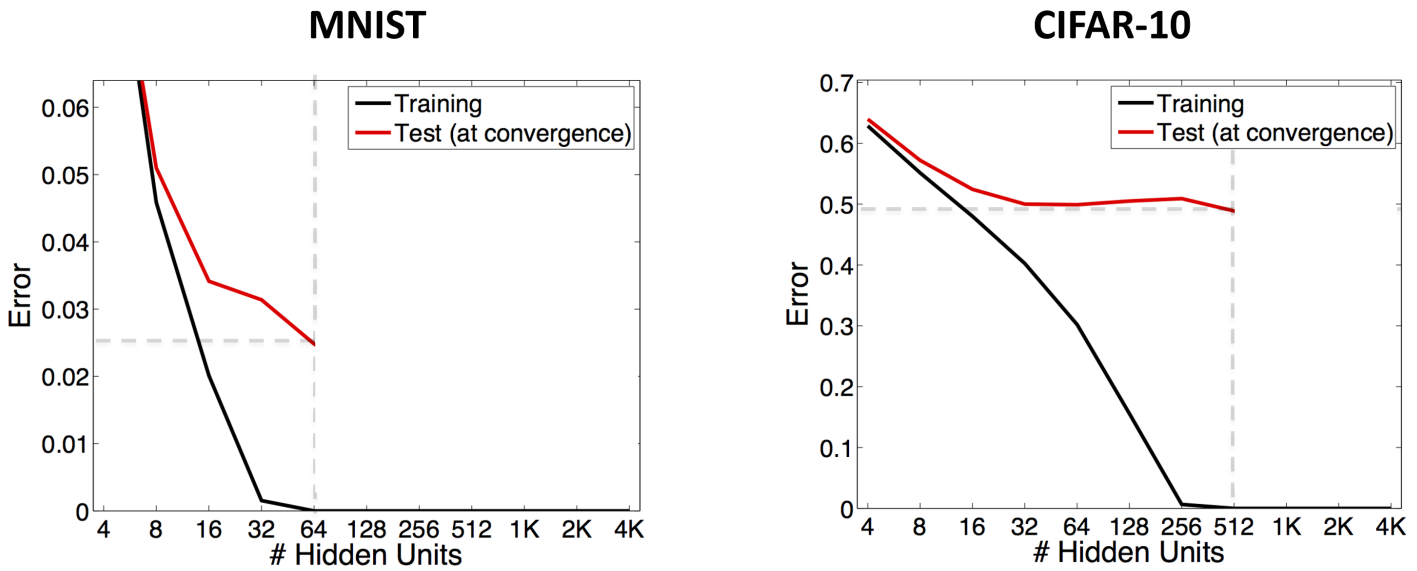
(a) Standard Neural Net (b) After applying dropout.

  - Adds invariance to missing inputs or latent factors
    - Encourages distributed representation rather than relying on specific z_i.
  - Can be interpreted as an ensemble over networks with different parts missing.
  - After a lot of success, dropout may already be going out of fashion.

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

“Hidden” Regularization in Neural Networks

- Fitting single-layer neural network with SGD and no regularization:



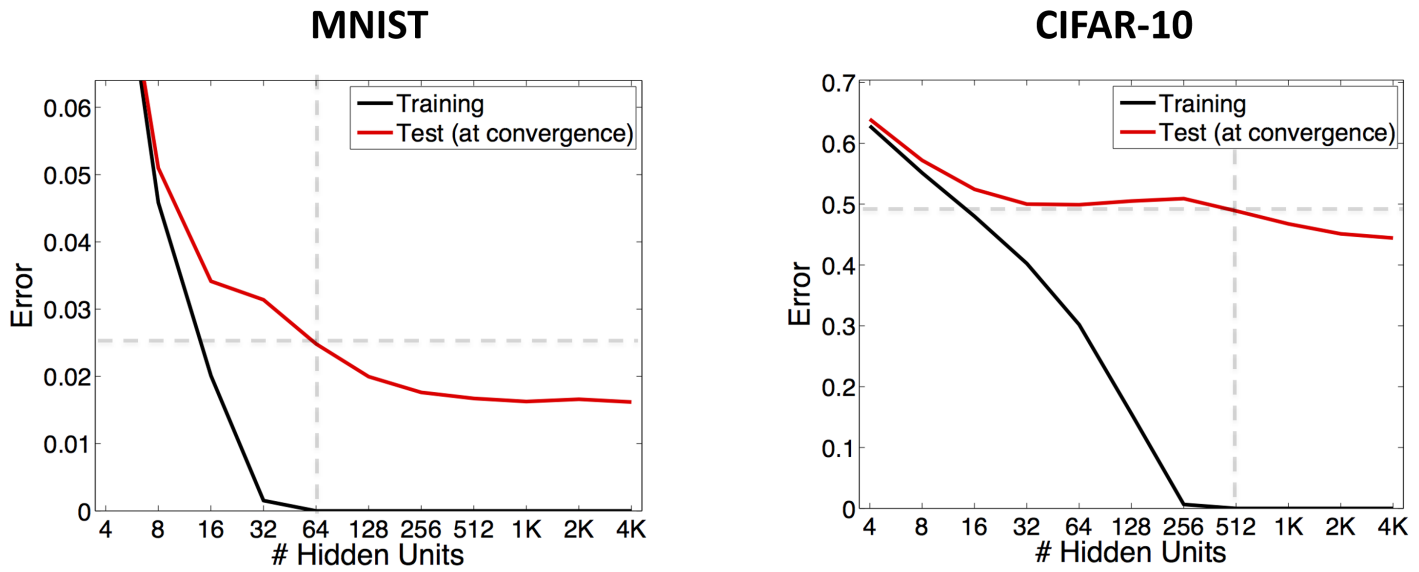
- Training goes to 0 with enough units: we’re finding a global min.

- What should happen to training and test error for larger #hidden?

https://www.neyshabur.net/papers/inductive_bias_poster.pdf

“Hidden” Regularization in Neural Networks

- Fitting single-layer neural network with SGD and no regularization:



- Test error continues to go down!?! Where is fundamental trade-off??
- There exist global mins with large #hidden units have test error = 1.
  - But among the global minima, SGD is somehow converging to “good” ones.

https://www.neyshabur.net/papers/inductive_bias_poster.pdf

Implicit Regularization of SGD

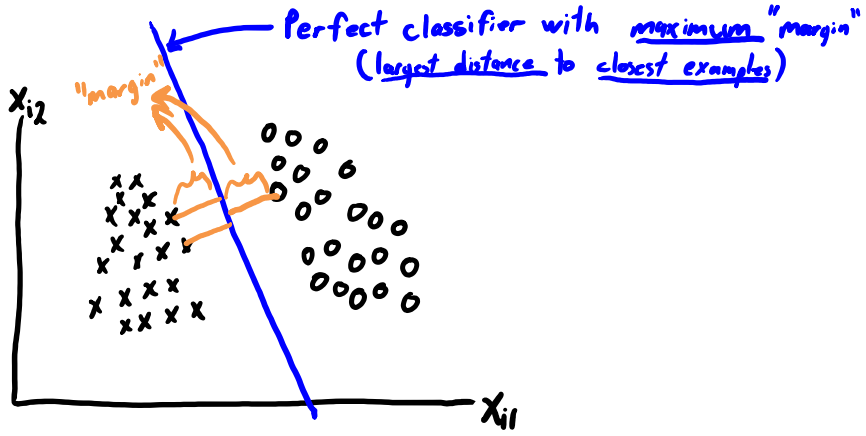
- There is growing evidence that using SGD regularizes parameters.
  - We call this the "implicit regularization" of the optimization algorithm.

- Beyond empirical evidence, we know this happens in simpler cases.

- Example of implicit regularization:
  - Consider a least squares problem where there exists a 'w' where Xw=y.
    - Residuals are all zero, we fit the data exactly.
  - You run [stochastic] gradient descent starting from w=0.
  - Converges to solution Xw=y that has the minimum L2-norm.
    - So using SGD is equivalent to L2-regularization here, but regularization is "implicit".

Implicit Regularization of SGD

- Example of implicit regularization:
  - Consider a logistic regression problem where data is linearly separable.
    - We can fit the data exactly.
  - You run gradient descent from any starting point.
  - Converges to max-margin solution of the problem.
    - So using gradient descent is equivalent to encouraging large margin.



- Similar result known for boosting.

Deep Learning "Tricks of the Trade"

- We've discussed heuristics to make deep learning work:
  - Parameter initialization and data transformations.
  - Setting the step size(s) in stochastic gradient and using momentum.
  - RestNets and alternative non-linear functions like ReLU.
  - Different forms of regularization:
    - L2-regularization, early stopping, dropout, implicit regularization from SGD.

- These are often still not enough to get deep models working.

- Deep computer vision models are all convolutional neural networks:
  - The $W^{(m)}$ are very sparse and have repeated parameters ("tied weights").
  - Drastically reduces number of parameters (speeds training, reduces overfitting).