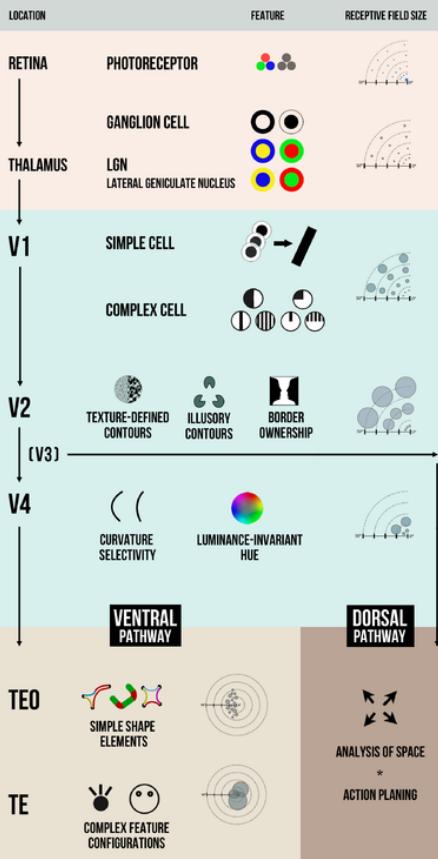


CPSC 340: Machine Learning and Data Mining

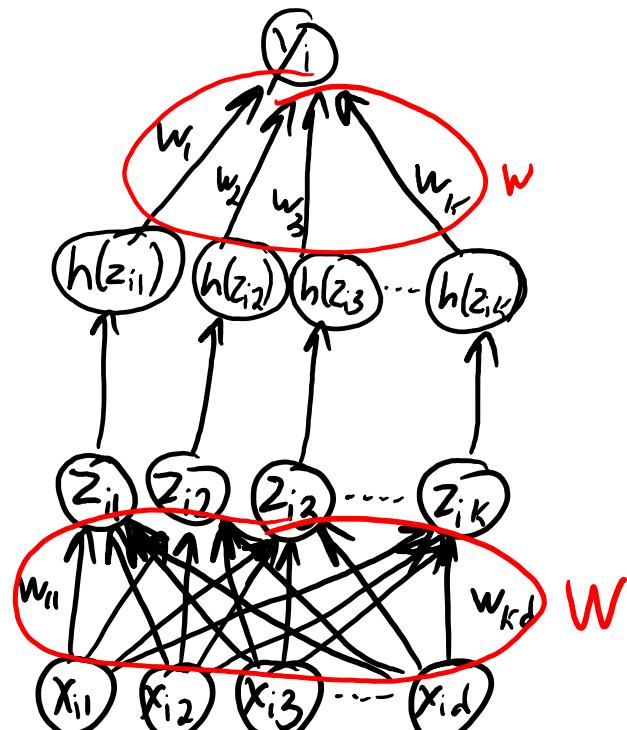
Deep Learning

Admin

- Assignment 5:
 - Due Friday.
- Assignment 6:
 - Due next Friday.



Neural network:

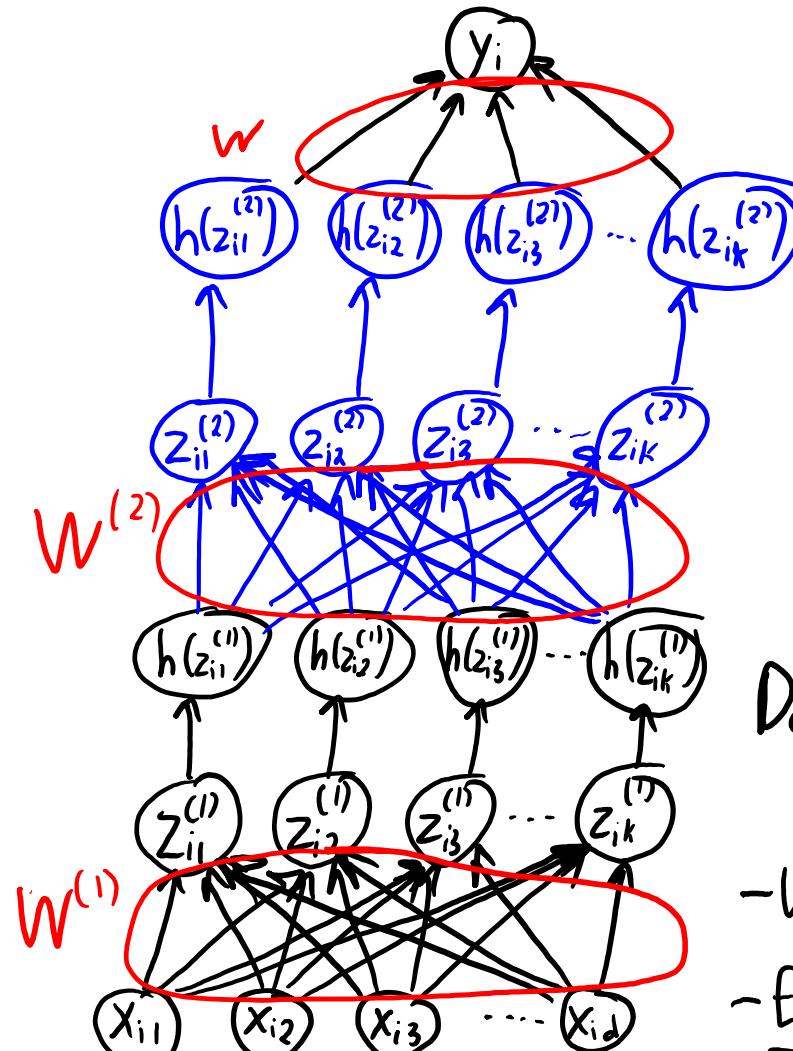


$$y_i = w^T h(Wx_i)$$

Learn ' W ' and ' w ' together.

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

Last Time: Deep Learning



- Deep neural networks:
- $$y_i = w^T h(W^{(2)} h(W^{(1)} x_i))$$
- Unprecedented performance on difficult problems.
 - Each layer combines "parts" from previous layer.
 - Train all layers together.
 - Multiple layers allow more "efficient" representations

Adding a Bias Variable

Remember that in linear models we may non-zero y-intercept:

$$y_i = w^T x_i + \beta$$

For neural networks we could have explicit bias:

$$y_i = w^T h(Wx_i) + \beta$$

We can just use $y_i = w^T x_i$ if we fix $x_{ij} = 1$ for all i for some j .
(constant x_{ij} value)

Or we could set $W_c = 0$ for one row ~~column~~ W_c of 'W'.

$$\frac{1}{1 + \exp(-W_c x_i)} = \frac{1}{1 + \exp(0)} = \frac{1}{2}$$

(constant zic value)

Artificial Neural Networks

- With squared loss, our objective function is:

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

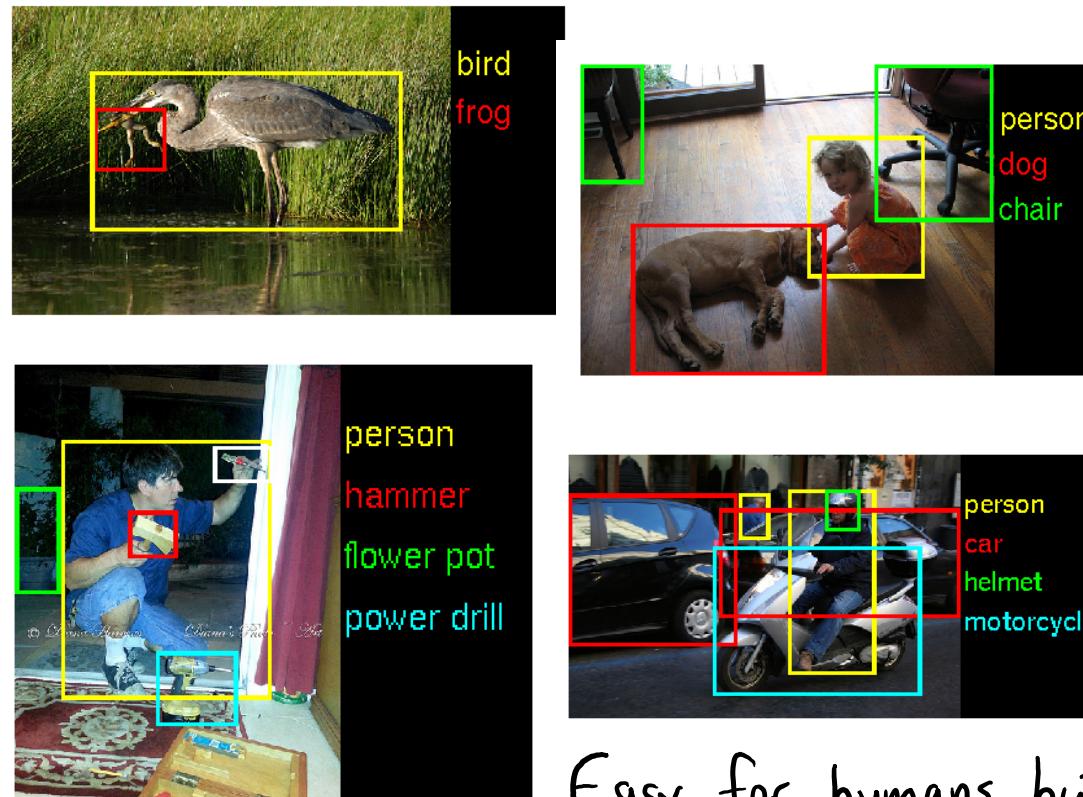
- Usual training procedure: **stochastic gradient**.
 - Compute gradient of random example ‘i’, update both ‘w’ and ‘W’.
 - Highly non-convex and can be difficult to tune.
- Computing the gradient is known as “**backpropagation**”.
 - This is basically the chain rule applied backwards through the layers.
 - We do some careful book-keeping to avoid **re-computing** quantities
 - Sort of like dynamic programming

Backpropagation derivations

- I removed all these slides because I don't believe they can followed/digested at our breakneck lecture pace
- Some people disagree with me on this
 - <https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b>
- I agree it's useful to know, but me teaching ≠ student knowing
 - And we have limited time.
- Hopefully this slide is a good compromise
 - “Knowing what you don't know”
- You can read about it on your own if you're interested.
 - Bonus slides and many free online resources.

ImageNet Challenge

- Millions of labeled images, 1000 object classes.



Easy for humans but
hard for computers.

ImageNet Challenge

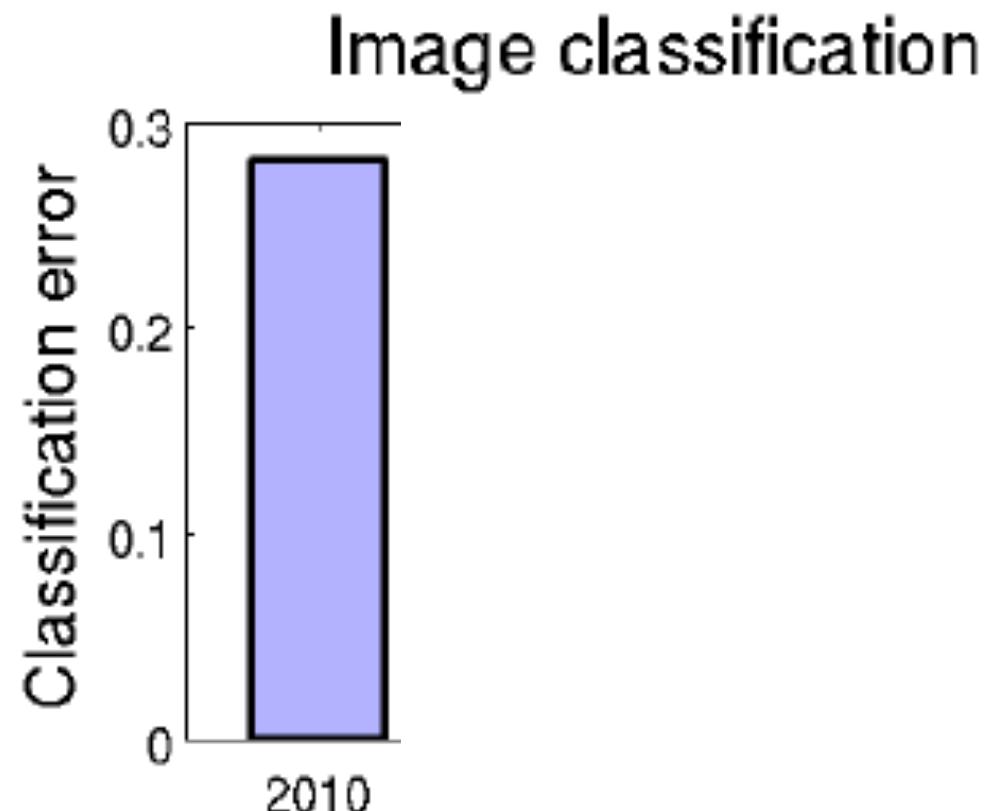
- Object detection task:
 - Single label per image.
 - Humans: ~5% error.



(a) Siberian husky



(b) Eskimo dog



ImageNet Challenge

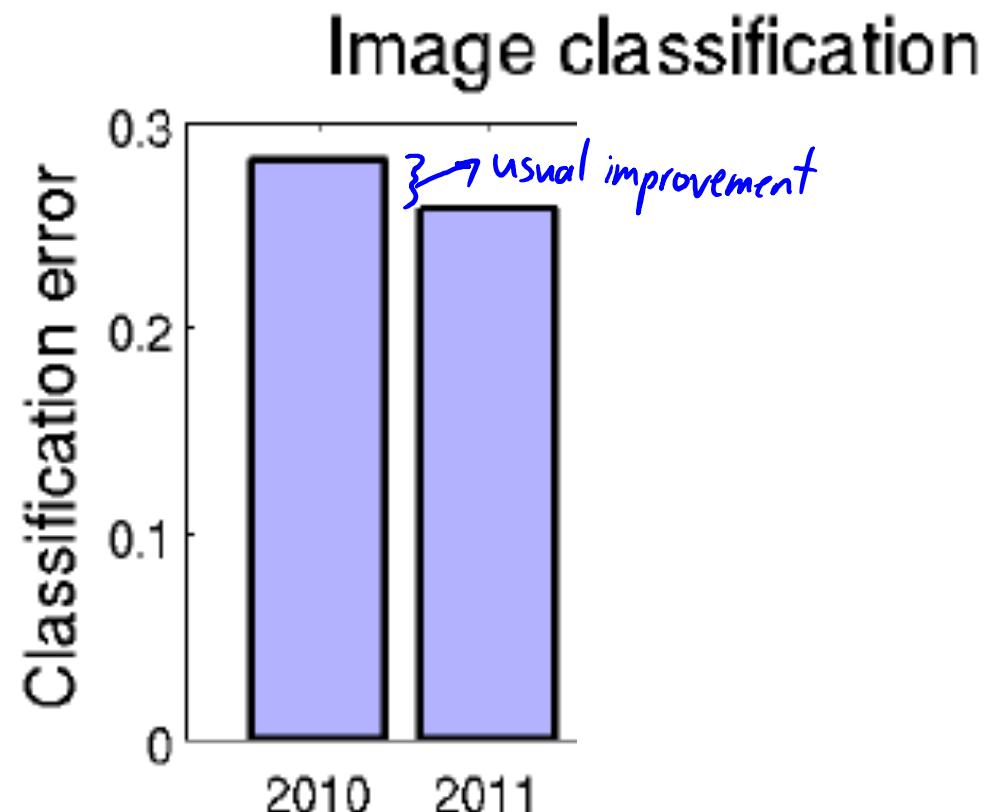
- Object detection task:
 - Single label per image.
 - Humans: ~5% error.



(a) Siberian husky



(b) Eskimo dog



ImageNet Challenge

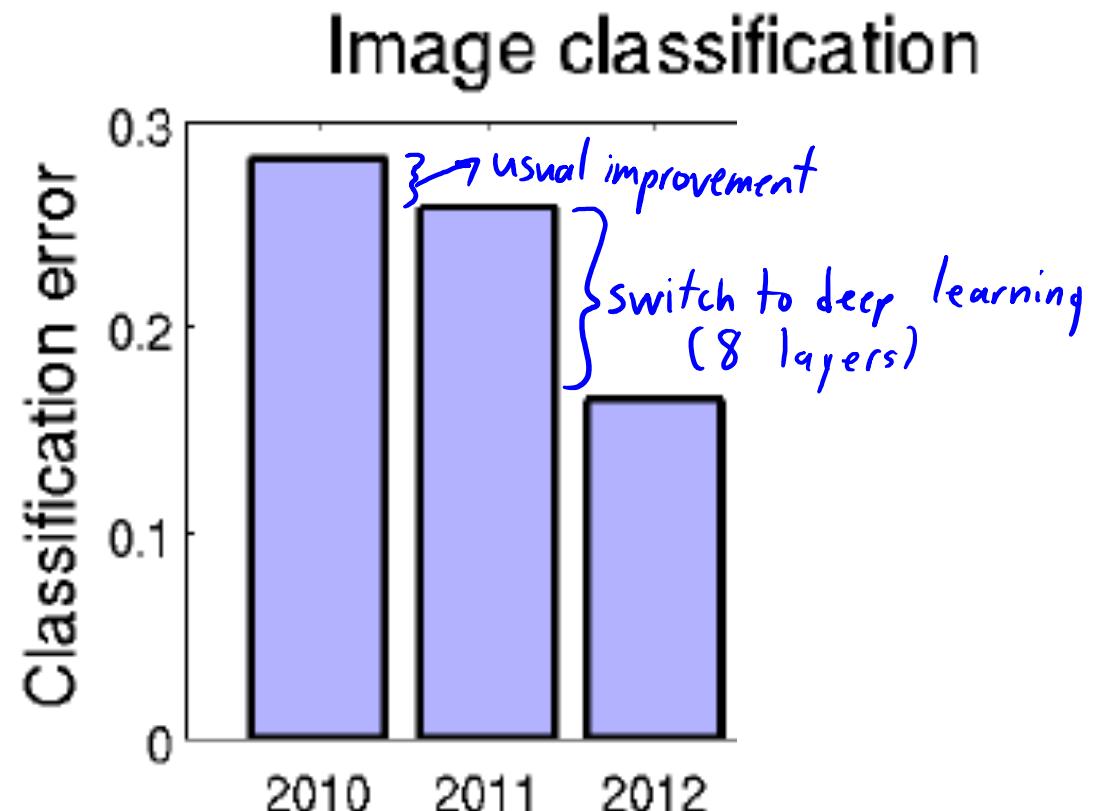
- Object detection task:
 - Single label per image.
 - Humans: ~5% error.



(a) Siberian husky



(b) Eskimo dog



ImageNet Challenge

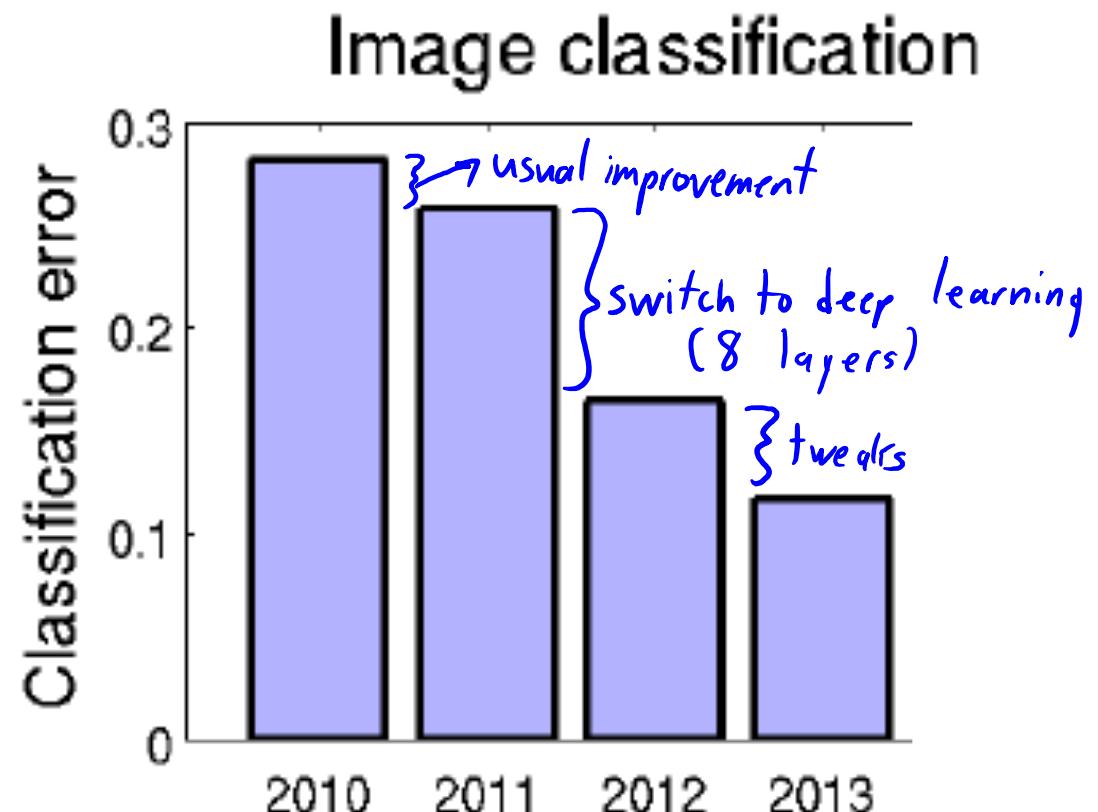
- Object detection task:
 - Single label per image.
 - Humans: ~5% error.



(a) Siberian husky



(b) Eskimo dog



ImageNet Challenge

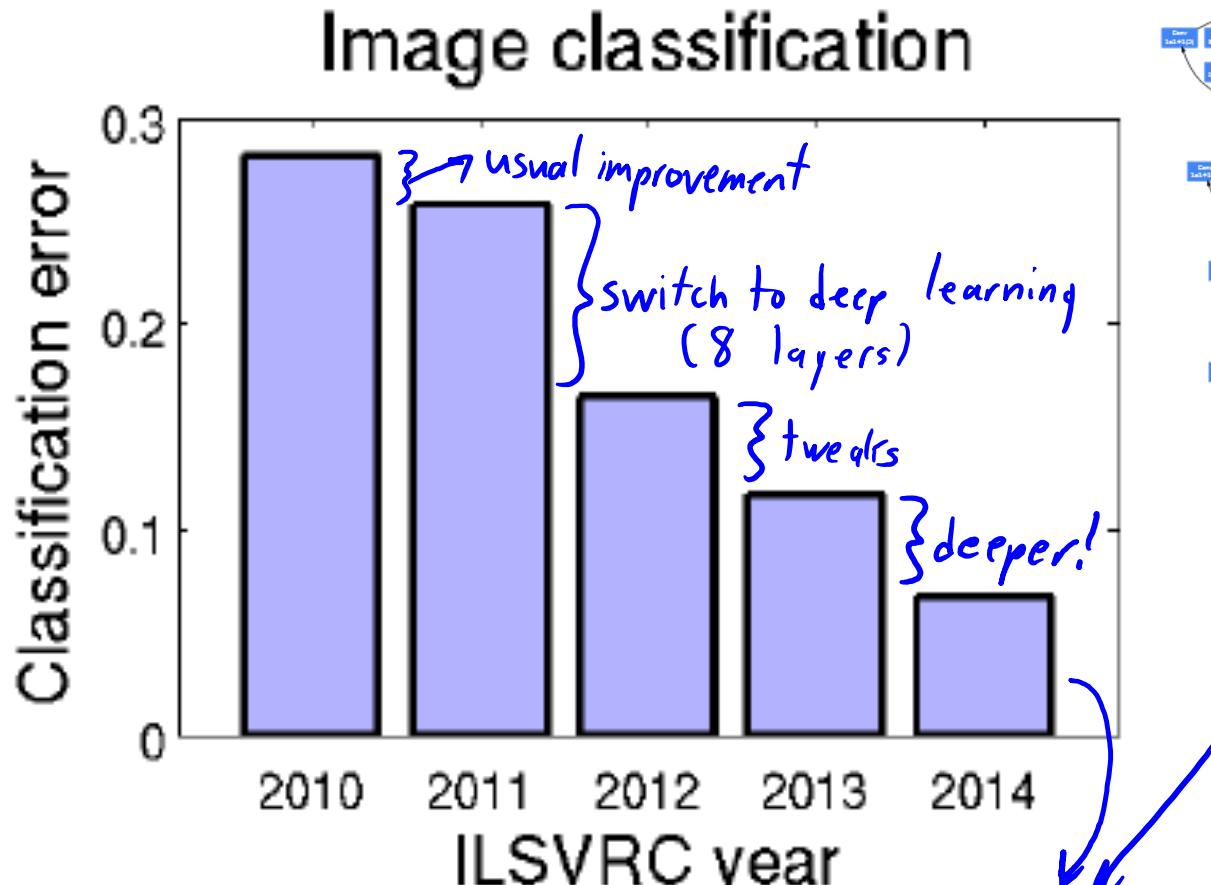
- Object detection task:
 - Single label per image.
 - Humans: ~5% error.



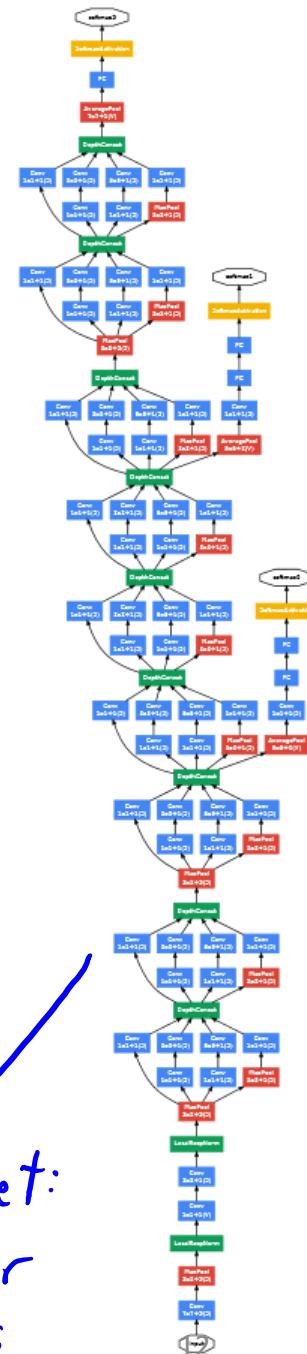
(a) Siberian husky



(b) Eskimo dog



GoogLe Net:
6.7% error
22 layers



ImageNet Challenge

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



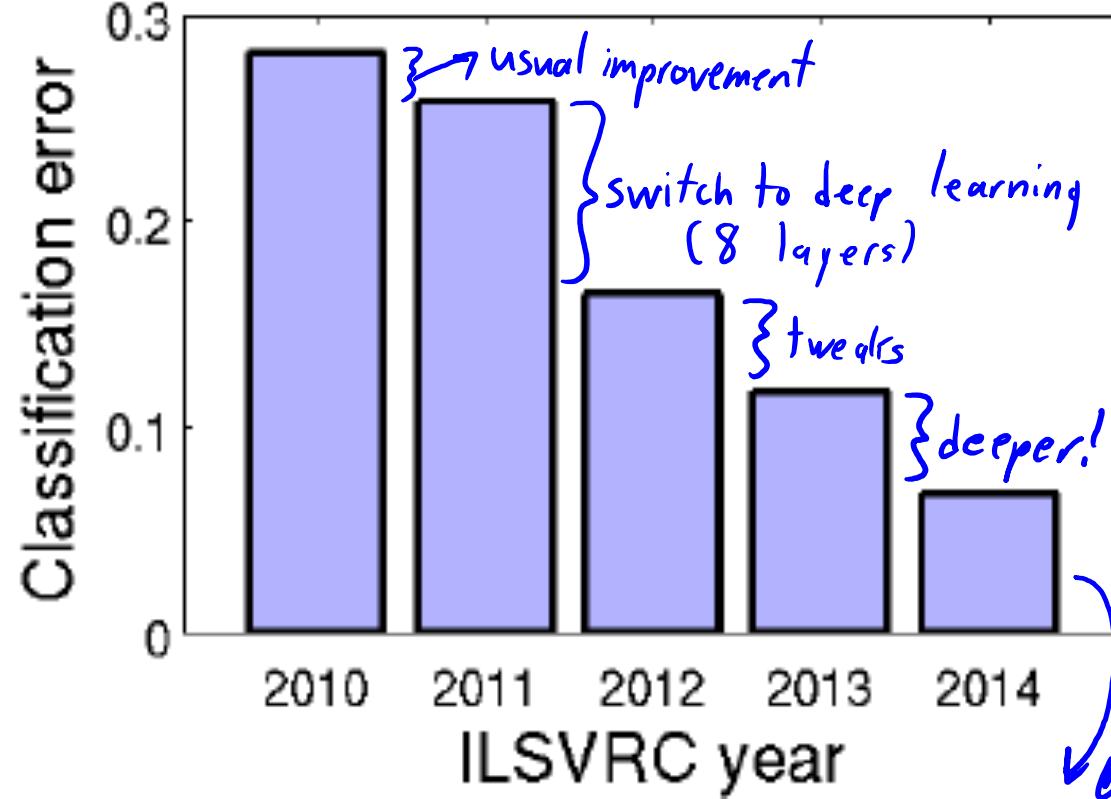
(a) Siberian husky



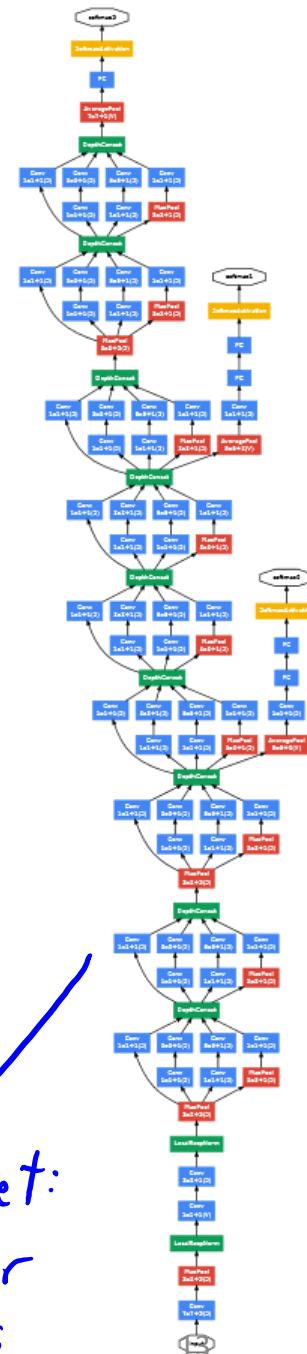
(b) Eskimo dog

- 2015 winner: Microsoft
 - 3.6% error.
 - 152 layers.

Image classification



GoogLe Net:
6.7% error
22 layers



ImageNet Challenge

- 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 mimima?
 - 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 even find a local minimum.

Parameter Initialization

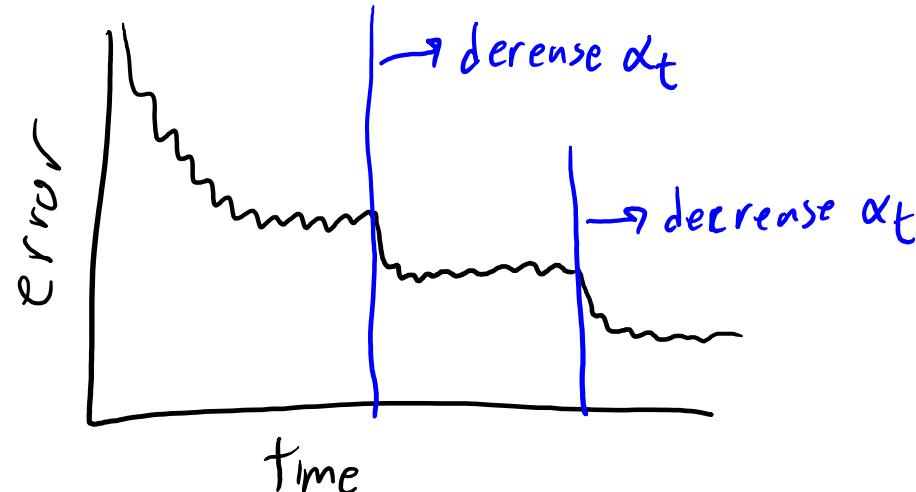
- Parameter initialization is crucial:
 - Can't initialize weights in same layer to same value, or they will stay same.
 - Can't initialize weights too large, it will take too long to learn.
- A traditional random initialization:
 - Initialize bias variables to 0.
 - Sample from standard normal, divided by 10^5 ($0.00001 * \text{randn}$).
 - Performing multiple initializations does not seem to be important.
- Popular approach from 10 years ago:
 - Initialize with deep unsupervised model (like autoencoders).

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 standardize data:
 - Subtract mean, divide by standard deviation, “whiten”, 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.
 - Use samples from standard normal distribution, divide by $\sqrt{2 * n_{\text{Inputs}}}$.
 - 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

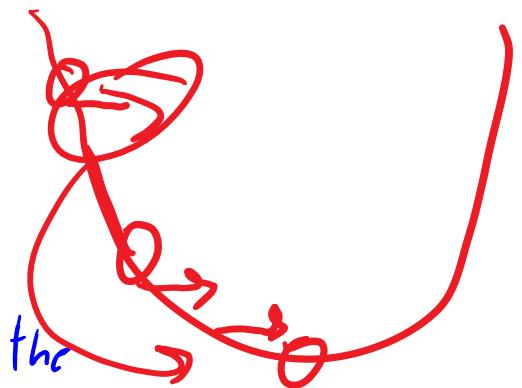
- Stochastic gradient is **very sensitive to the step size** in deep models.
- More automatic method is **Bottou trick**:
 1. Grab a small set of training examples (maybe 5% of total).
 2. Do a **binary search for a step size** that works well on them.
 3. Use this step size for a long time (or slowly decrease it from there).
- Several recent methods using a **step size for each variable**:
 - AdaGrad, RMSprop, Adam.

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:**
 - Add term that moves in previous direction:

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

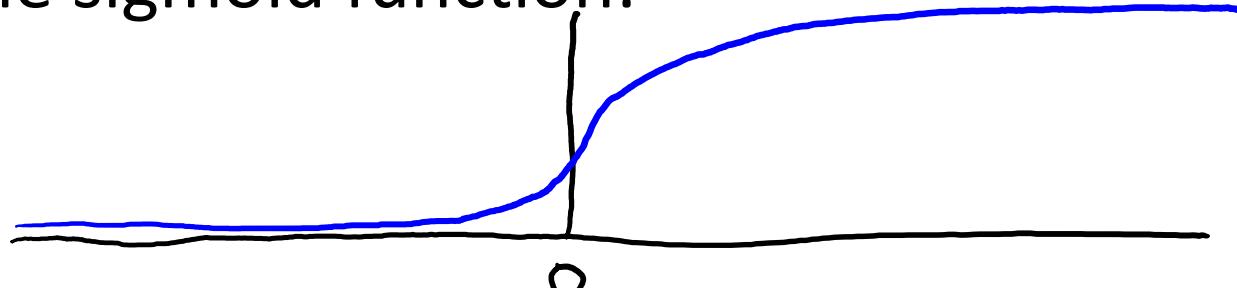
Keep going in the
old direction



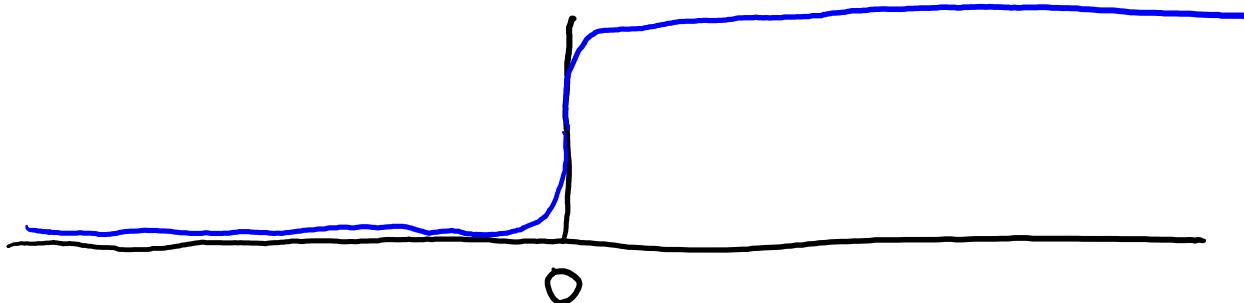
- Batch size (number of random examples) also influences results.
- Another recent trick is **batch normalization**:
 - Try to “standardize” the hidden units within the random samples as we go.

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 hinge-like loss (ReLU):
$$\max\{0, W_c x_i\}$$

The graph shows the ReLU function plotted against its input. The x-axis represents the input, and the y-axis represents the output. A vertical black line at the origin (0) separates the negative and positive regions. For negative inputs, the function is zero (blue line). At the origin, it transitions from zero to a linear increase (green line). For positive inputs, the function follows the linear equation $y = W_c x_i$ (green line), which is labeled as $\frac{1}{t_{exp}(-W_c x_i)}$. The blue line represents the derivative of the ReLU function, which is zero for all negative inputs and increases towards 1 as the input becomes more positive.

- The gradient is zero or x_i , depending on the sign.
 - Fixes vanishing gradient problem.
 - Gives sparser of activations.
 - Not really simulating binary signal, but could be simulating rate coding.

Deep Learning and the Fundamental Trade-Off

- Neural networks are subject to the fundamental trade-off:
 - As we increase the depth, training error decreases.
 - As we increase the depth, training error no longer approximates test error.
- We want deep networks to model highly non-linear data.
 - But increasing the depth leads to **overfitting**.
- How could GoogLeNet use 22 layers?
 - Many forms of **regularization** and keeping model complexity under control.

Standard Regularization

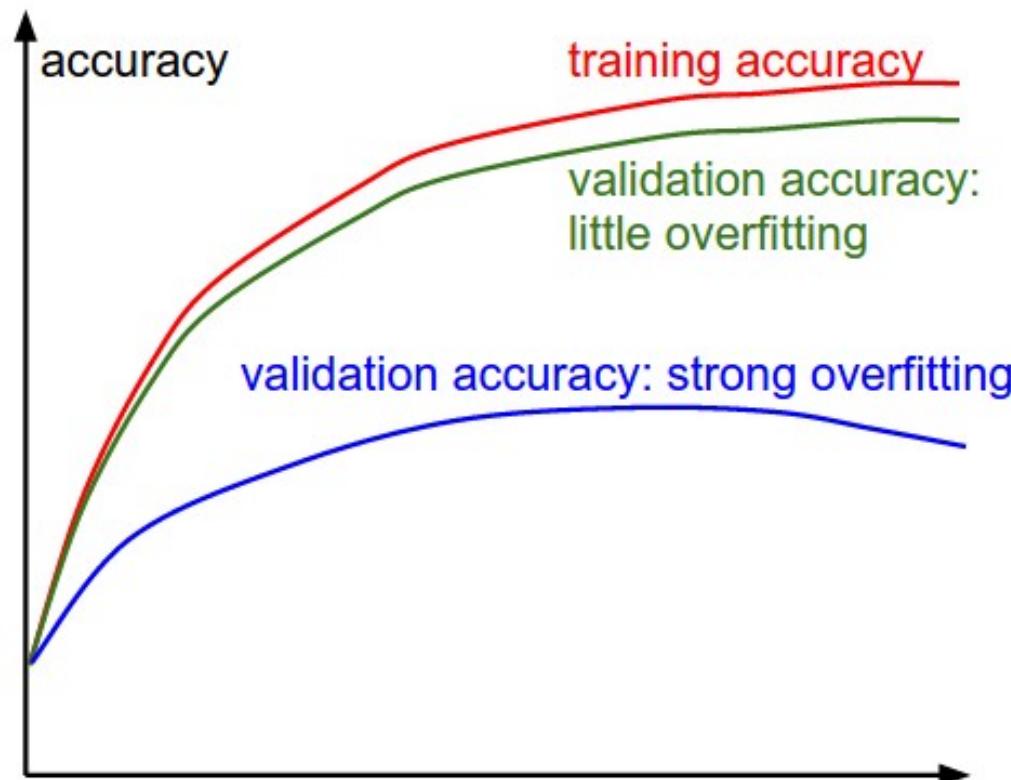
- We typically add our usual L2-regularizers:

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

- L2-regularization is called “weight decay” in neural network papers.
 - Could also use L1-regularization.
- “Hyper-parameter” optimization:
 - Try to optimize validation error in terms of $\lambda_1, \lambda_2, \lambda_3, \lambda_4$.
- Unlike linear models, typically use multiple types of regularization.

Early Stopping

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

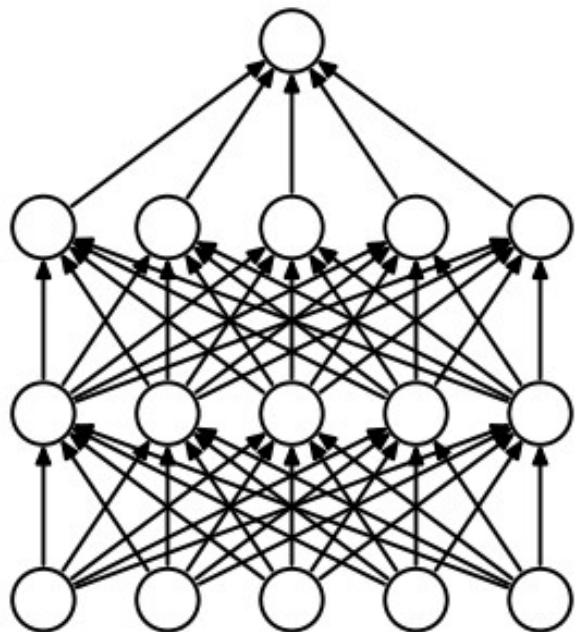


Unfortunately it might look more like

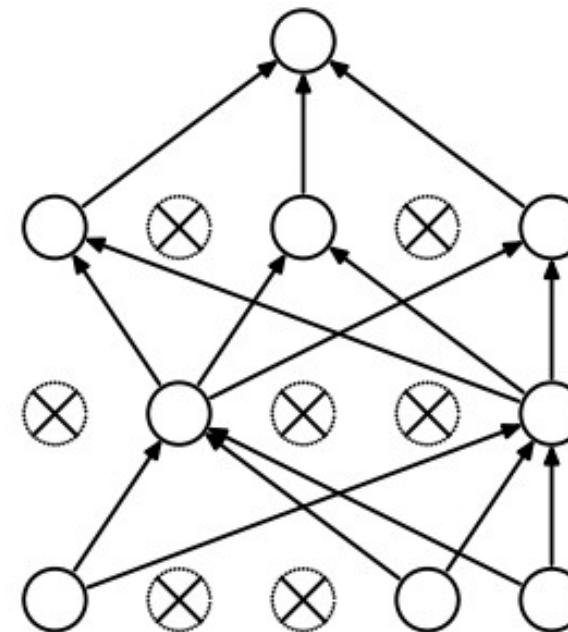
hopefully you don't stop here.

Dropout

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



(a) Standard Neural Net

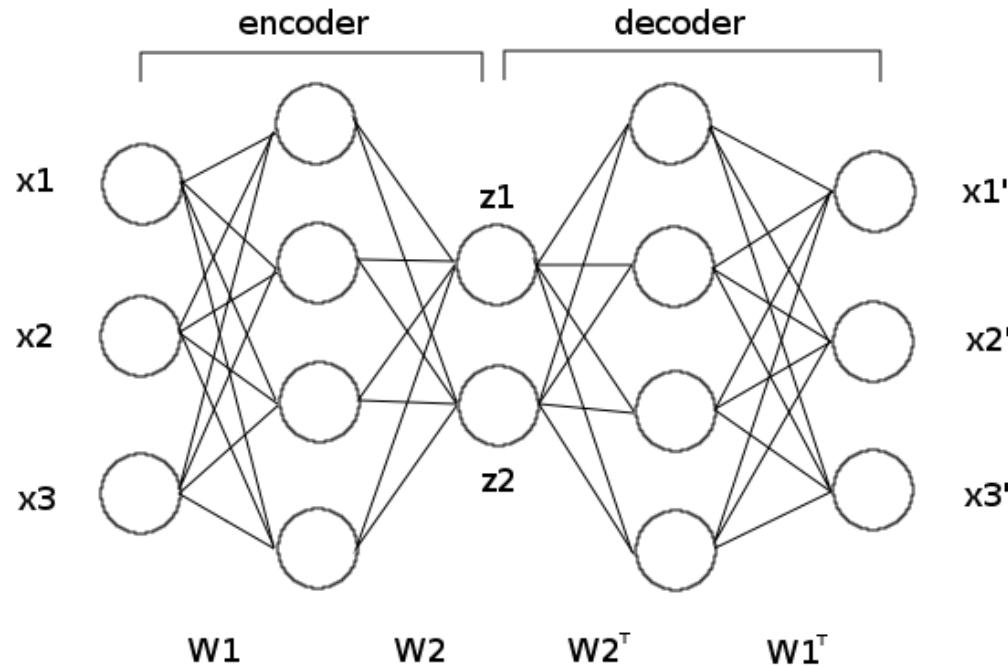


(b) After applying dropout.

- Encourages **distributed representation** rather than using specific z_i .

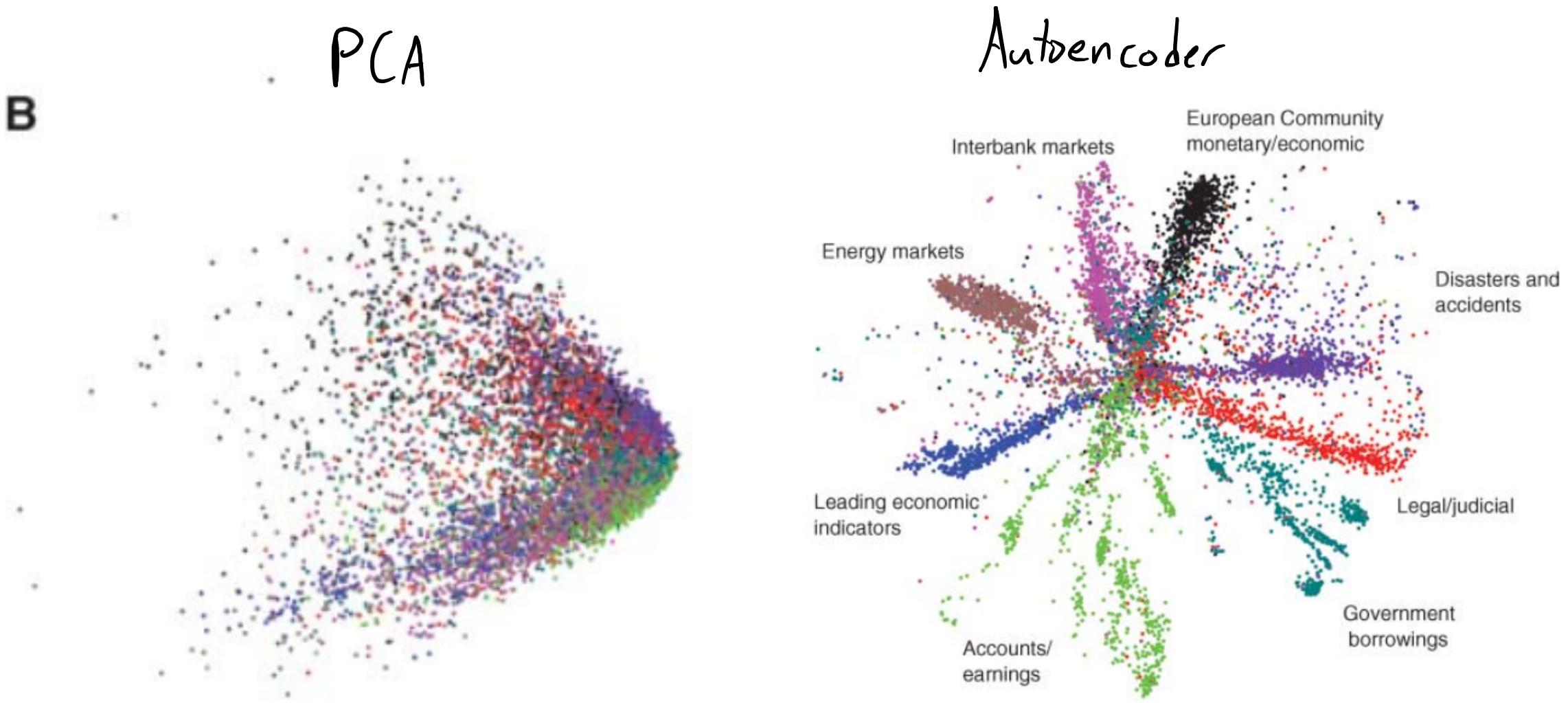
Autoencoders

- 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.

Autoencoders

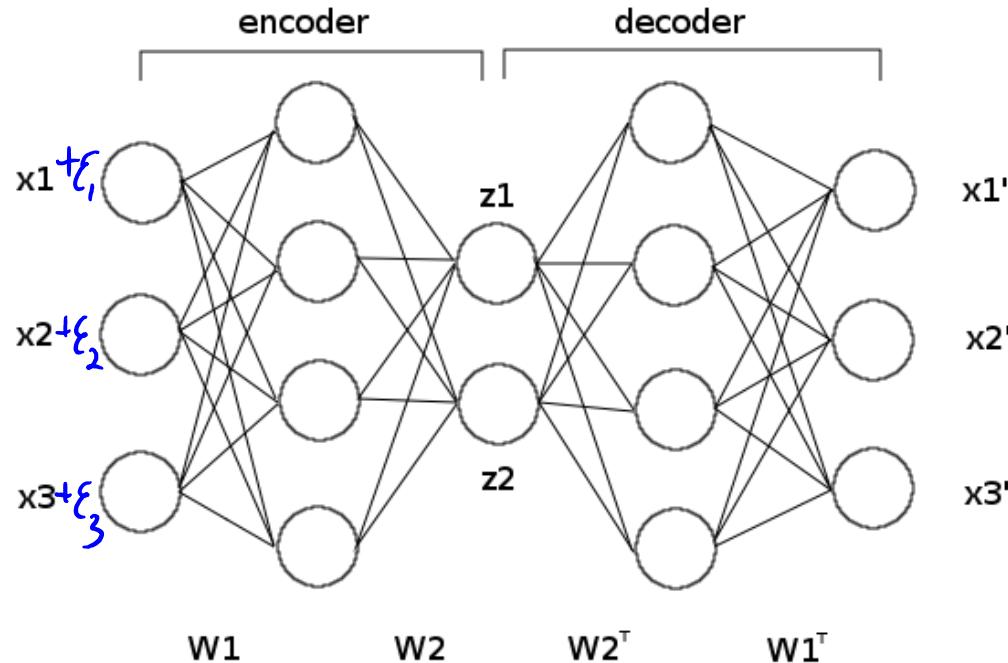


Summary

- Parameter initialization is crucial to neural net performance.
- Optimization and step size are crucial to neural net performance.
- Regularization is crucial to neural net performance:
 - L2-regularizaiton, early stopping, dropout.
- Autoencoders are unsupervised neural net latent-factor models.
- Next time:
 - Convolutions, convolutional neural networks, and rating selfies.

Denoising Autoencoder

- Denosing autoencoders add noise to the input:



- Learns a model that can remove the noise.