CPSC 340:
Machine Learning and Data Mining

Deep Learning

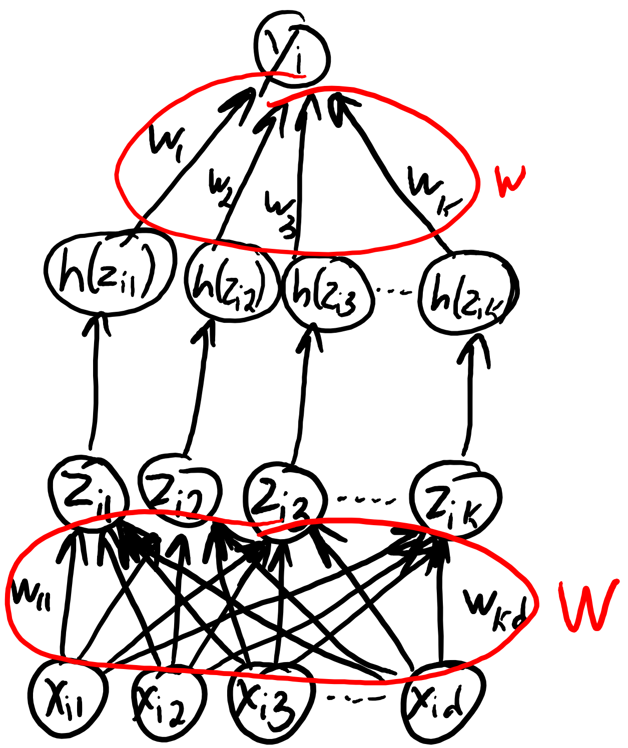
Fall 2016

Admin

- Assignment 5:
  - Due Friday.
- Assignment 6:
  - Due next Friday.
- Final:
  - December 12 (8:30am – HEBB 100)
  - Covers Assignments 1-6.
  - Final from last year and list of topics will be posted.
  - Closed-book, cheat sheet: 4-pages each double-sided.

Last Time: Deep Learning

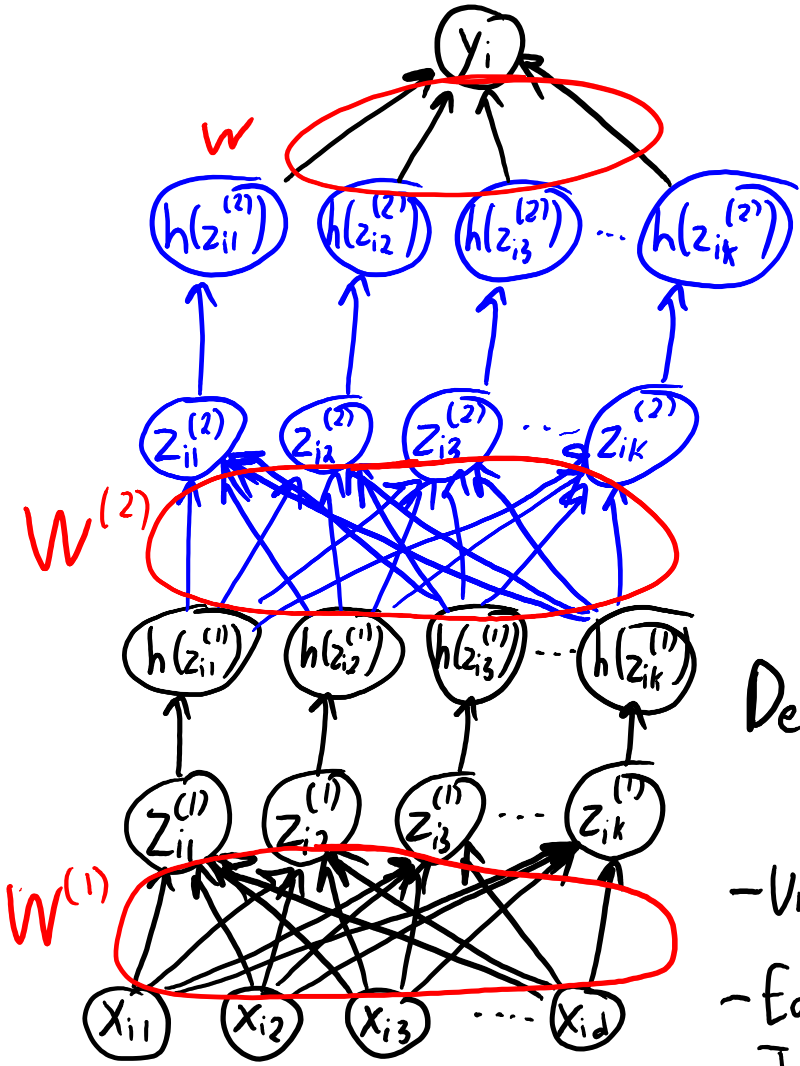
Neural network:

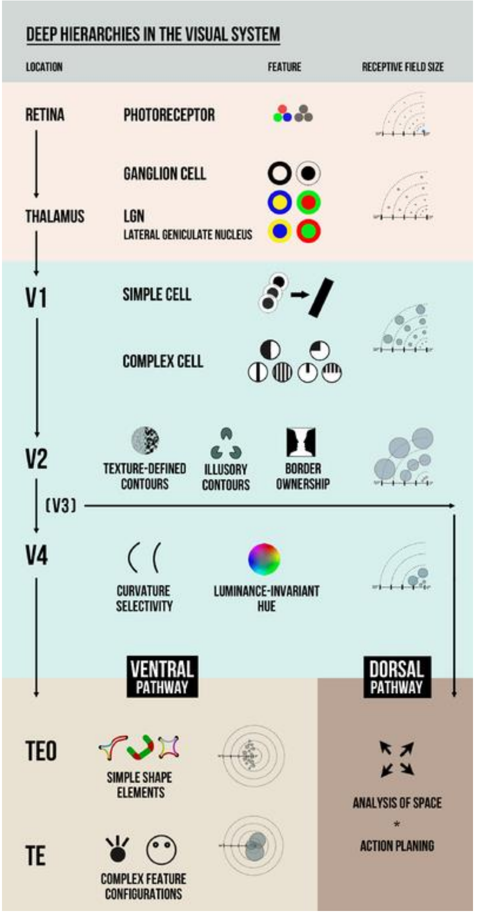


$y_i = w^T h(Wx_i)$

Learn 'W' and 'w' together.

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





Deep neural networks:

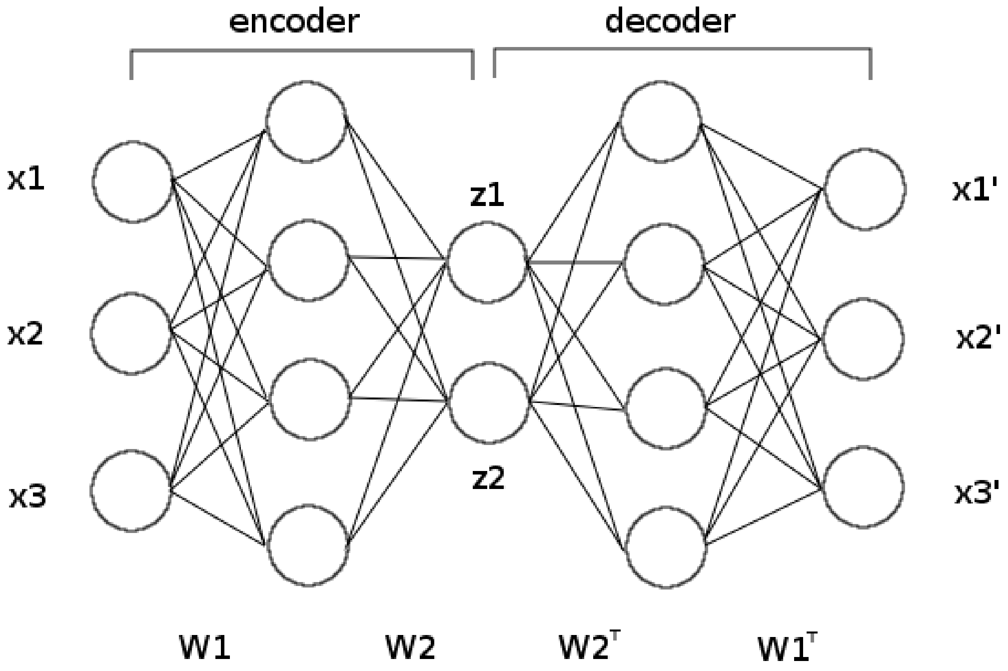
$y_i = w^T h(W^{(2)} h(W^{(1)} x_i))$

- Unprecedented performance on difficult problems.
- Each layer combines "parts" from previous layer.
- Train all layers together.
- Multiple layers allow more "efficient" representations.

https://en.wikibooks.org/wiki/Sensory_Systems/Visual_Signal_Processing

Autoencoders

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

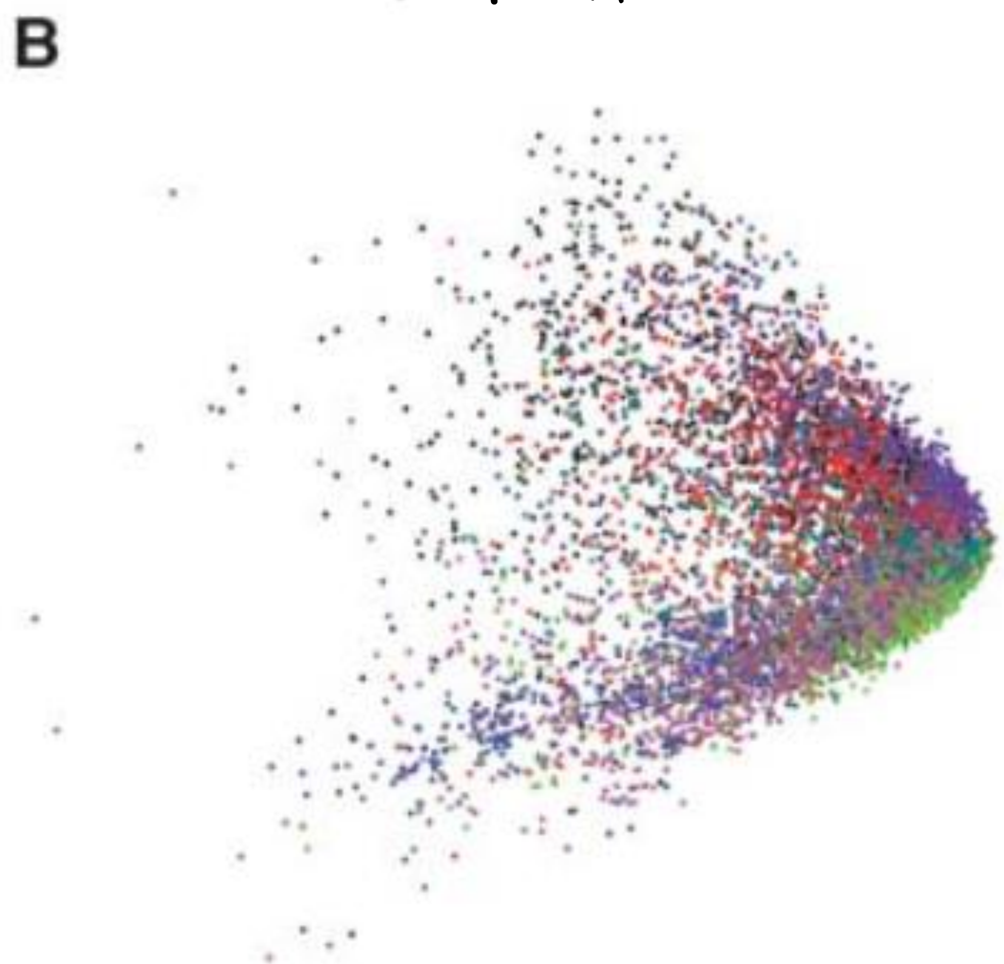


  - Middle layer could be latent features in non-linear latent-factor model.
    - Can do outlier detection, data compression, visualization, etc.

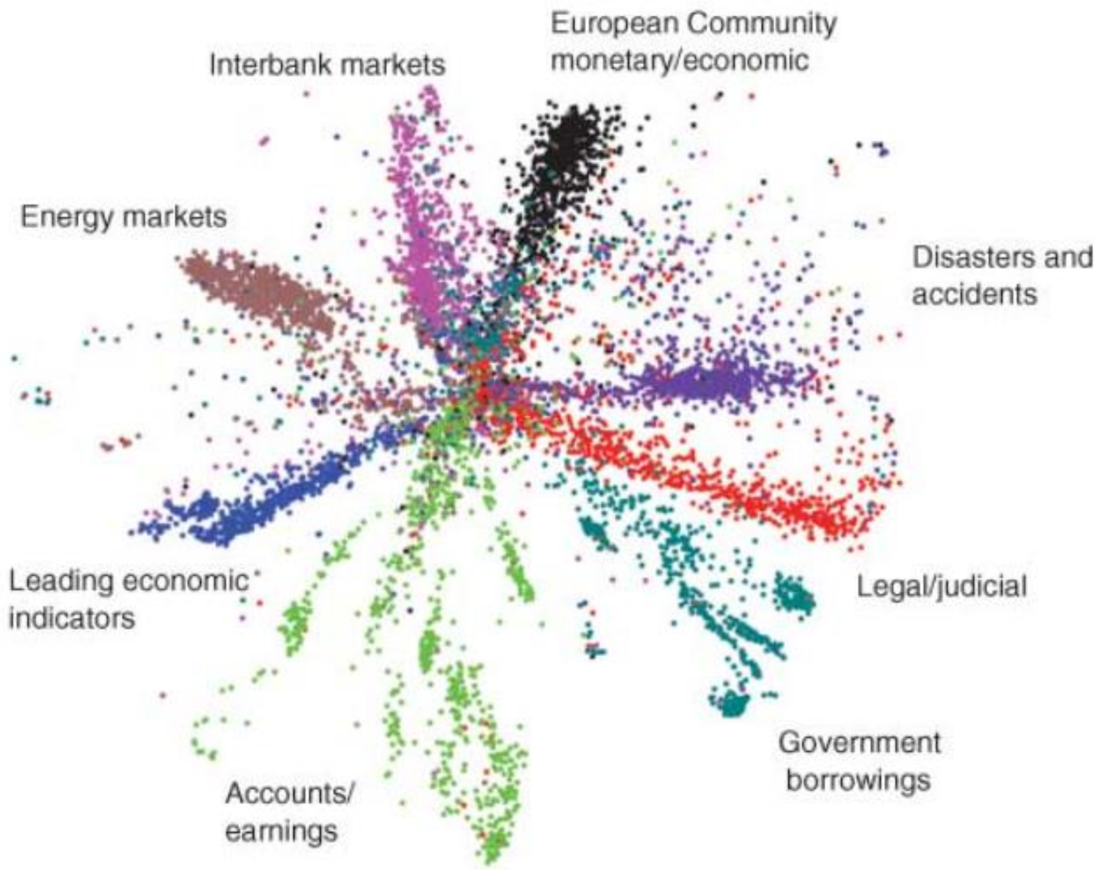
http://inspirehep.net/record/1252540/plots

Autoencoders

PCA



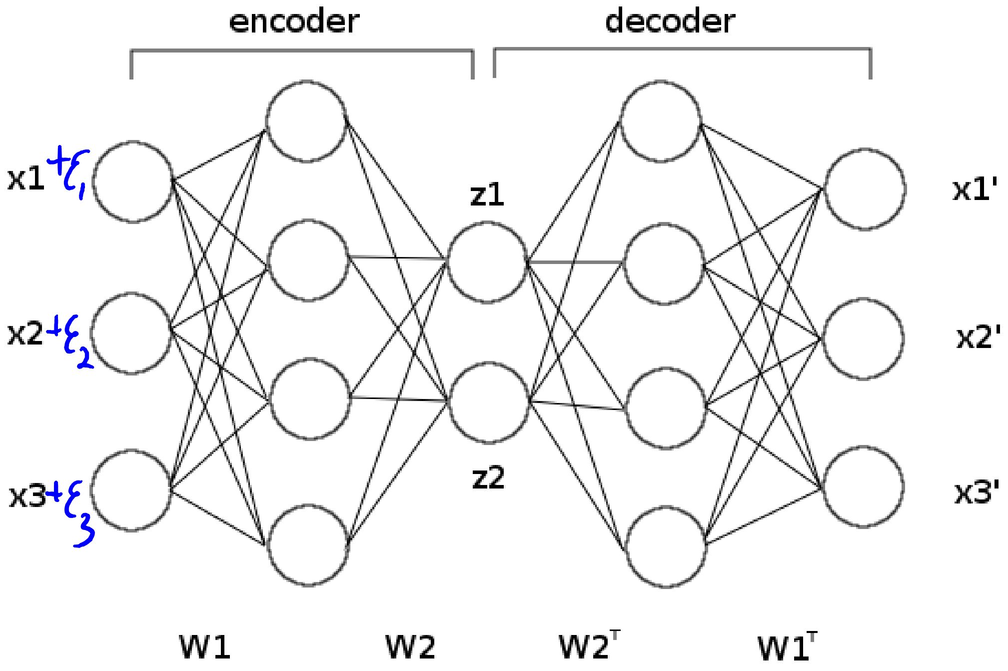
Autoencoder



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

Denoising Autoencoder

- Denosing autoencoders add noise to the input:



  – Learns a model that can remove the noise.

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.

Last Time: Backpropagation with 1 Hidden Layer

- Squared loss our objective function with 1 layer and 1 example:

$$f(w, W) = \frac{1}{2}\left(\sum_{c=1}^{k} w_c h(W_c x_i) - y_i\right)^2$$

- Gradient with respect to element of vector 'w'.

$$\frac{\partial}{\partial w_c}\left[f(w,W)\right] = \underbrace{\left(\sum_{c=1}^{k} w_c h(W_c x_i) - y_i\right)}_{r_i} h(W_c x_i) = r_i h(W_c x_i)$$

- Gradient with respect to element of matrix 'W'.

$$\frac{\partial}{\partial W_{cj}}\left[f(w,W)\right] = \underbrace{\left(\sum_{c=1}^{k} w_c h(W_c x_i) - y_i\right)}_{r_i} \underbrace{w_c h'(W_c x_i)}_{v_c} x_{ij} = r_i v_c x_{ij}$$

- Only r_i changes if you aren't using squared error.

Last Time: Backpropagation with 1 Hidden Layer

- Squared loss our objective function with 1 layer and 1 example:

$$f(w, W) = \frac{1}{2}\left(\sum_{c=1}^{k} w_c h(W_c x_i) - y_i\right)^2$$

- Gradient with respect to elements of vector 'w' and 'W':

$$\frac{\partial}{\partial w_c}\left[f(w,W)\right] = r_i h(W_c x_i) \qquad \frac{\partial}{\partial W_{cj}}\left[f(w,W)\right] = r_i v_c x_{ij}$$

with $r_i = \sum_{c=1}^{k} w_c h(W_c x_i) - y_i$

and $v_c = w_c h'(W_c x_i)$

- Backpropagation algorithm:
  - Forward propagation computes $z_i = h(W_c x_i)$ then $w^T z_i$.
  - Backpropagation step 1: use r_i to get gradient of 'w'.
  - Backpropagation step 2: use r_i and v_c to get gradient of 'W'.

Backpropagation with 2 Hidden Layer

- General objective function with 2 layers and 1 example:

$$f(w, W^{(2)}, W^{(1)}) = f_i(w^T h(W^{(2)} h(W^{(1)} x_i)))$$

- Gradient with respect to element of vector 'w':

$$\frac{\partial}{\partial w_c}[f(w, W^{(2)}, W^{(1)})] = \underbrace{f_i'(w^T h(W^{(2)} h(W^{(1)} x_i)))}_{r} \underbrace{h(W_c^{(2)} h(W^{(1)} x_i))}_{z_c^{(2)}} = r z_c^{(2)}$$

- Gradient with respect to element of matrix 'W$^{(2)}$':

$$\frac{\partial}{\partial W_{cj}^{(2)}}[f(w, W^{(2)}, W^{(1)})] = \underbrace{f_i'(w^T h(W^{(2)} h(W^{(1)} x_i)))}_{r} \underbrace{w_c h'(W_c^{(2)} h(W^{(1)} x_i))}_{v_c} \underbrace{h(W_j^{(1)} x_i)}_{z_j^{(1)}} = r v_c z_j^{(1)}$$

- Gradient with respect to element of matrix 'W$^{(1)}$':

$$\frac{\partial}{\partial W_{cj}^{(1)}}[f(w, W^{(2)}, W^{(1)})] = \underbrace{f_i'(w^T h(W^{(2)} h(W^{(1)} x_i)))}_{r} \sum_{c'=1}^{k} \left[\underbrace{w_{c'} h'(W_{c'}^{(2)} h(W^{(1)} x_i))}_{v_{c'}} \underbrace{W_{c'c}^{(2)} h'(W_c^{(1)} x_i)}_{u_{cc'}} x_{ij}\right] = r(v^T u_c) x_{ij}$$

Last Time: Backpropagation with 3 Hidden Layers

- General objective function with 3 layers and 1 example:

$$f(w, W^{(3)}, W^{(2)}, W^{(1)}) = f_i(w^T h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))))$$

- Gradients have the form:

$$\frac{\partial f}{\partial w_c} = r z_c^{(3)} \qquad \frac{\partial f}{W^{(3)}_{cc'}} = \underbrace{r v_c}_{} z_{c'}^{(2)} = a_c z_{c'} \qquad \frac{\partial f}{\partial W^{(2)}_{cc'}} = r(v^T u_c) z_{c'}^{(1)} = \underbrace{a^T u_c}_{} z_{c'}^{(1)} = b_c z_{c'} \qquad \frac{\partial f}{\partial W^{(1)}_{cj}} = \underbrace{b^T d_c}_{} x_{ij} \approx e_c x_{ij}$$

- Backpropagation algorithm:
  - Forward propagation computes ‘r’ and $z^{(m)}$ for all ‘m’.
  - Backpropagation step 1: use ‘r’ to get gradient of ‘w’.
  - Backpropagation step 2: use a_c to get gradient of $W^{(3)}$.
  - Backpropagation step 3: use b_c to get gradient of $W^{(2)}$.
  - Backpropagation step 4: use d_c to get gradient of $W^{(1)}$.

Last Time: Backpropagation with 3 Hidden Layers

- Backpropagation algorithm:
  - Forward propagation computes 'r' and $z^{(m)}$ for all 'm'.
  - Backpropagation step 1: use r to get gradient of 'w'.
  - Backpropagation step 2: use a_c to get gradient of $W^{(3)}$.
  - Backpropagation step 3: use b_c to get gradient of $W^{(2)}$.
  - Backpropagation step 4: use d_c to get gradient of $W^{(1)}$.
- Cost of backpropagation:
  - Forward pass dominated by multiplications by $W^{(1)}$, $W^{(2)}$, $W^{(3)}$, and 'w'.
    - If have 'm' layers and all z_i have 'k' elements, cost would be $O(dk + mk^2)$.
  - Backward pass has same cost.
- For multi-class or multi-label classification, replace 'w' by matrix.
  - Softmax loss is called "cross entropy" in neural network papers.

Last Time: 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 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 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*randn).
  - Performing multiple initializations does not seem to be important.

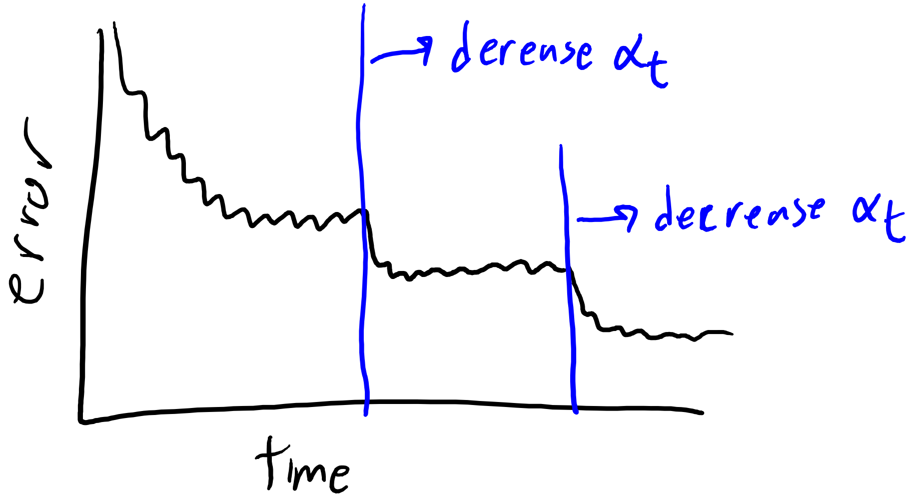
- Popular approach from 10 years ago:
  - Initialize with deep unsupervised model (like autoencoders).

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*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

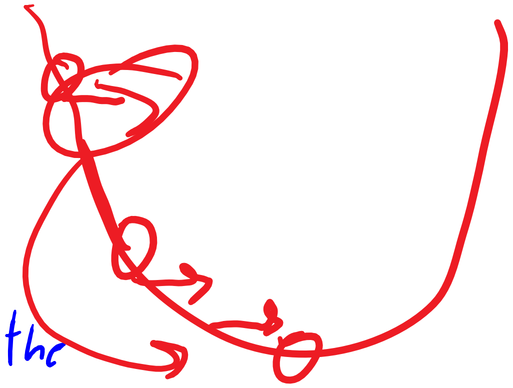
- Stochastic gradient is very sensitive to the step size in deep models.
- More automatic method is Bottou trick:
  1. Grab a small set of training examples (maybe 5% of total).
  2. Do a binary search for a step size that works well on them.
  3. Use this step size for a long time (or slowly decrease 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:

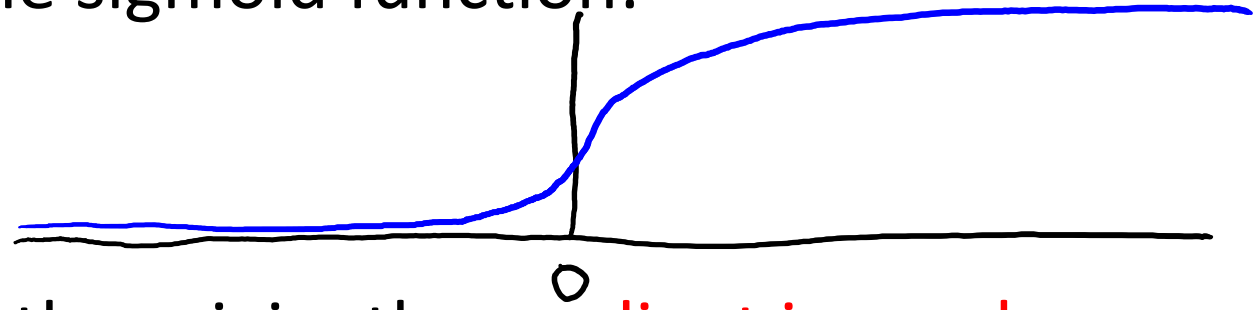
$$w^{t+1} = w^t - \alpha^t \nabla f_i(w^t) + \underbrace{\beta^t (w^t - w^{t-1})}_{\text{keep going in the old direction}}$$



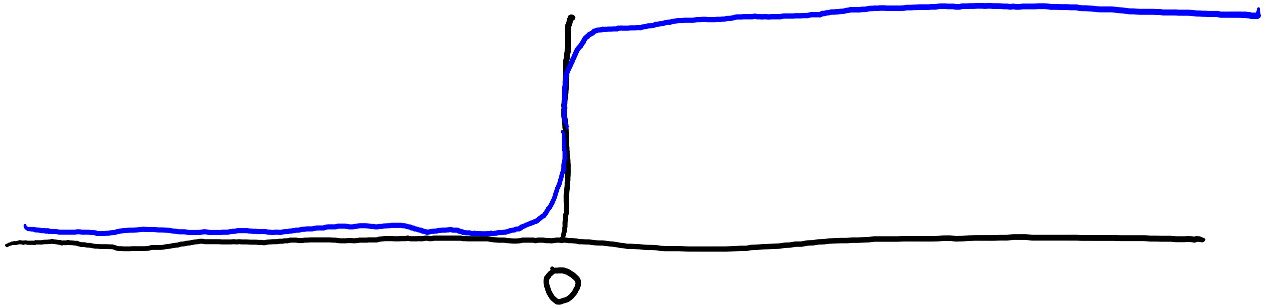
- Batch size (number of random examples) also influences results.
- Another recent trick is batch normalization:
  - Try to "standardize" the hidden units within the random samples as we go.

Vanishing Gradient Problem

- Consider the sigmoid function:



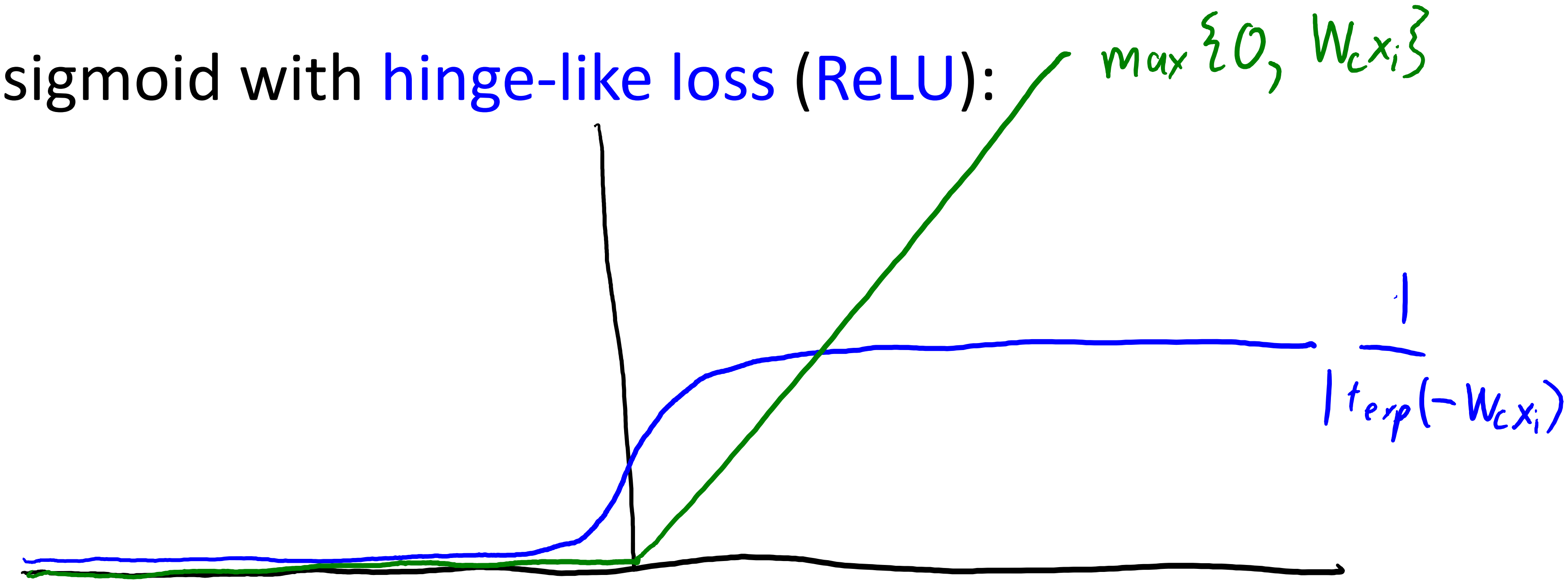
- Away from the origin, the gradient is nearly zero.
- The problem gets worse when you take the sigmoid of a sigmoid:



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

Rectified Linear Units (ReLU)

- Replace sigmoid with hinge-like loss (ReLU):



- 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:

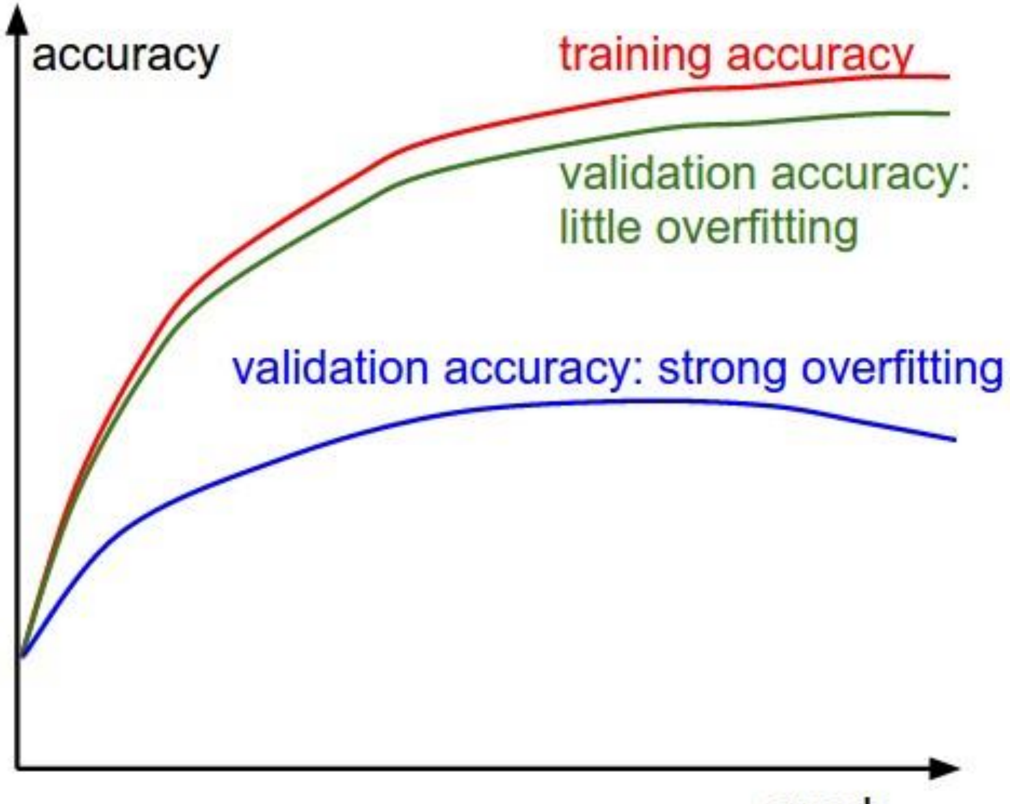
$$f(w, W^{(3)}, W^{(2)}, W^{(1)}) = \frac{1}{2}\sum_{i=1}^{n}\left(w^T h(W^{(3)} h(W^{(2)} h(W^{(1)} x_i))) - y_i\right)^2 + \frac{\lambda_4}{2}\|w\|^2 + \frac{\lambda_3}{2}\|W^{(3)}\|_F^2 + \frac{\lambda_2}{2}\|W^{(2)}\|_F^2 + \frac{\lambda_1}{2}\|W^{(1)}\|_F^2$$

- 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 λ_1, λ_2, λ_3, λ_4.

- Unlike linear models, typically use multiple types of regularization.

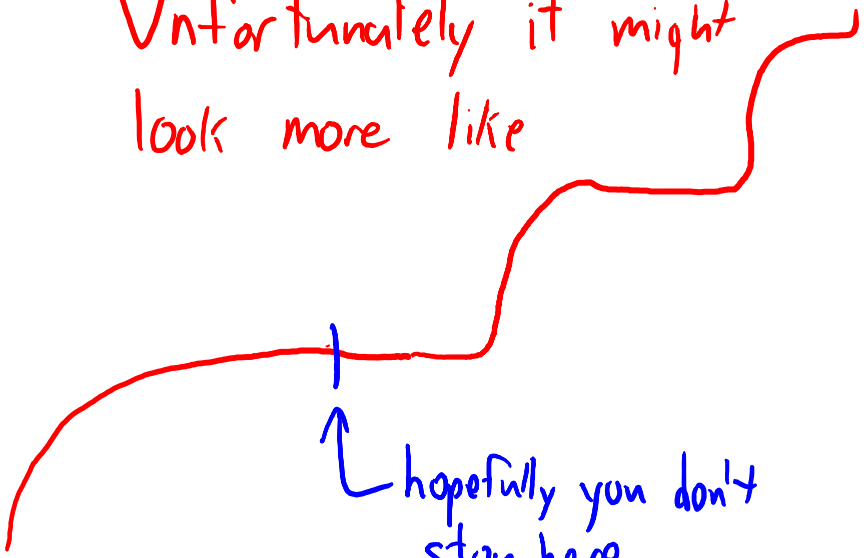
Early Stopping

- 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.



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

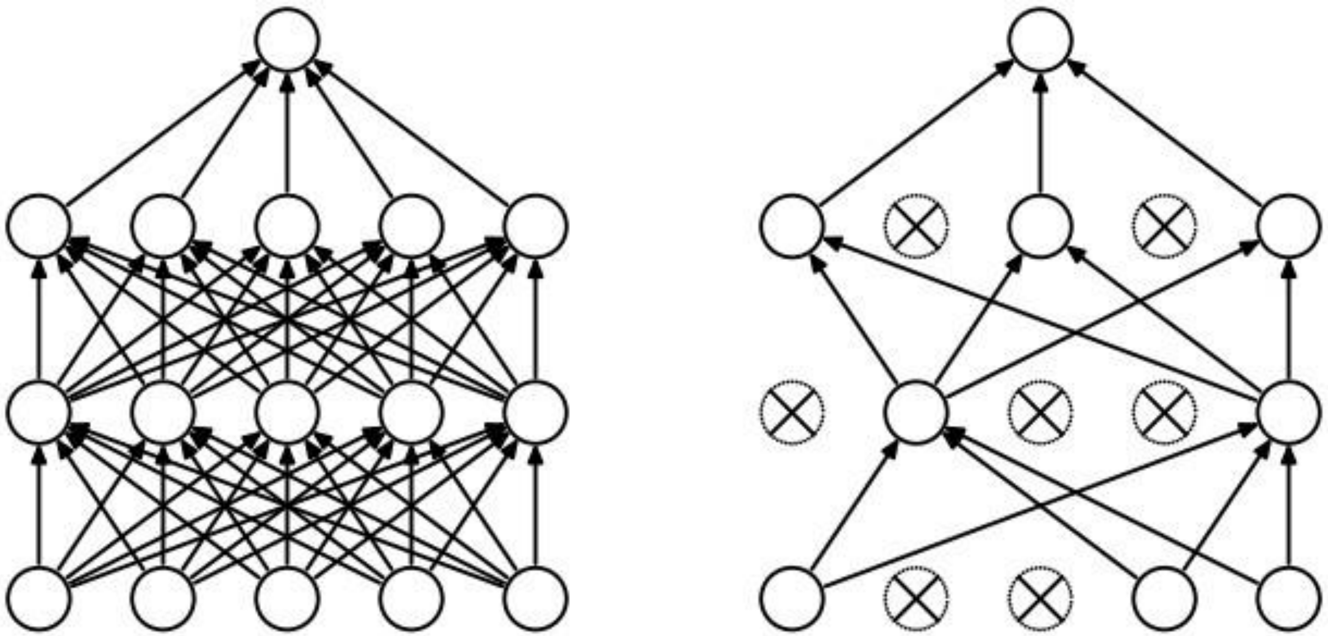
Unfortunately it might look more like



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.

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

Convolutional Neural Networks

- Typically use multiple types of regularization:
  - L2-regularization.
  - Early stopping.
  - Dropout.
- Often, still not enough to get deep models working.

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

Summary

- Autoencoders are unsupervised neural net latent-factor models.
- 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.

- Next time:
  - Convolutions, convolutional neural networks, and rating selfies.