CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 4: Activations, Initializations, Optimization, and Regularization

CSCI 566 Resources

- **Website:** https://csci566-spring2023.github.io/
- **Piazza:** https://piazza.com/usc/spring2023/csci566/info
- **GDrive:** https://drive.google.com/drive/folders/11YcbSZcJUdRSbrNBWiOs_HM2VkxdNKWS?usp=share_link
  - Includes spreadsheet with lecture slide + recording links!
- **Zoom:** https://usc.zoom.us/j/95538924150?pwd=TTRBM283VCt4WDFkN1hLMmtrKzdGUT09

Course Project Team Registration

- Your 4-person teams should be registered now
- If you have not registered your team, fill the form ASAP!
  - Piazza -> Resources -> Course Project -> Project Team Registration
- If you haven't *found* a team, come up to the front during the break to find all the other students who haven't found a team.

Map to the Midterm

January 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

www.a-printable-calendar.com

February 2023

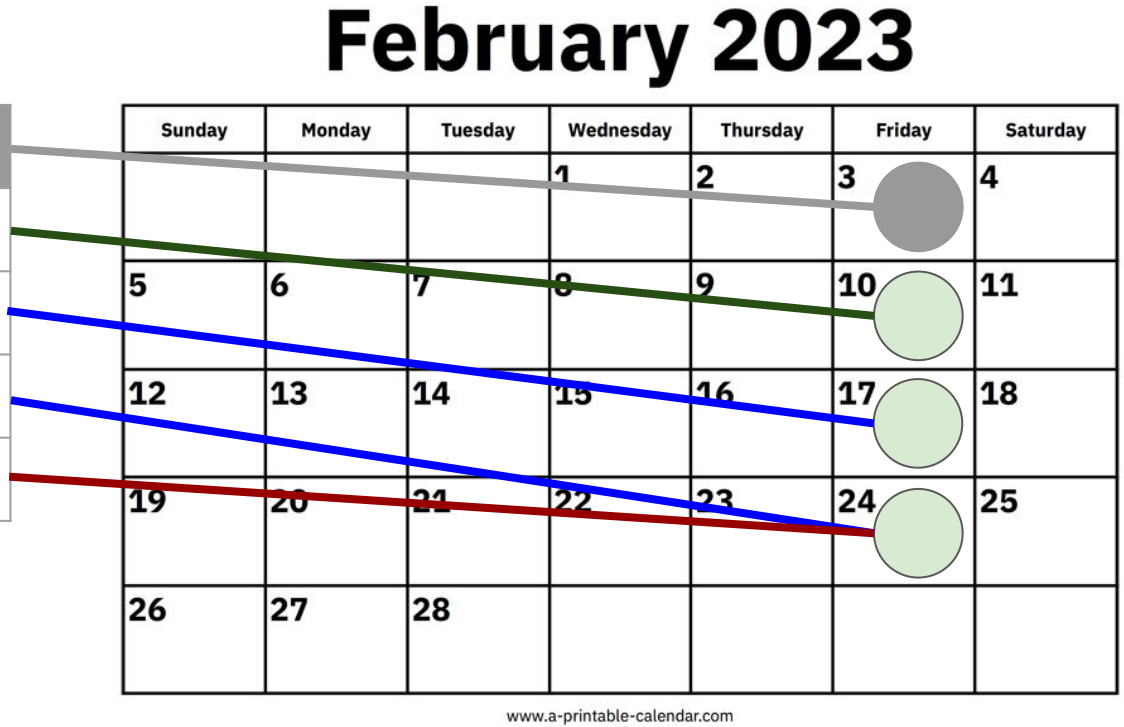
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

www.a-printable-calendar.com

Module 1: Neural Network Basics

[Feb 10] A Look at February Deliverables

Project Teams Formed	Feb 3
Assignment 1 Out	Feb 10
Project Proposal Due	Feb 17
Assignment 1 Due	Feb 24
Midterm Exam	Feb 24

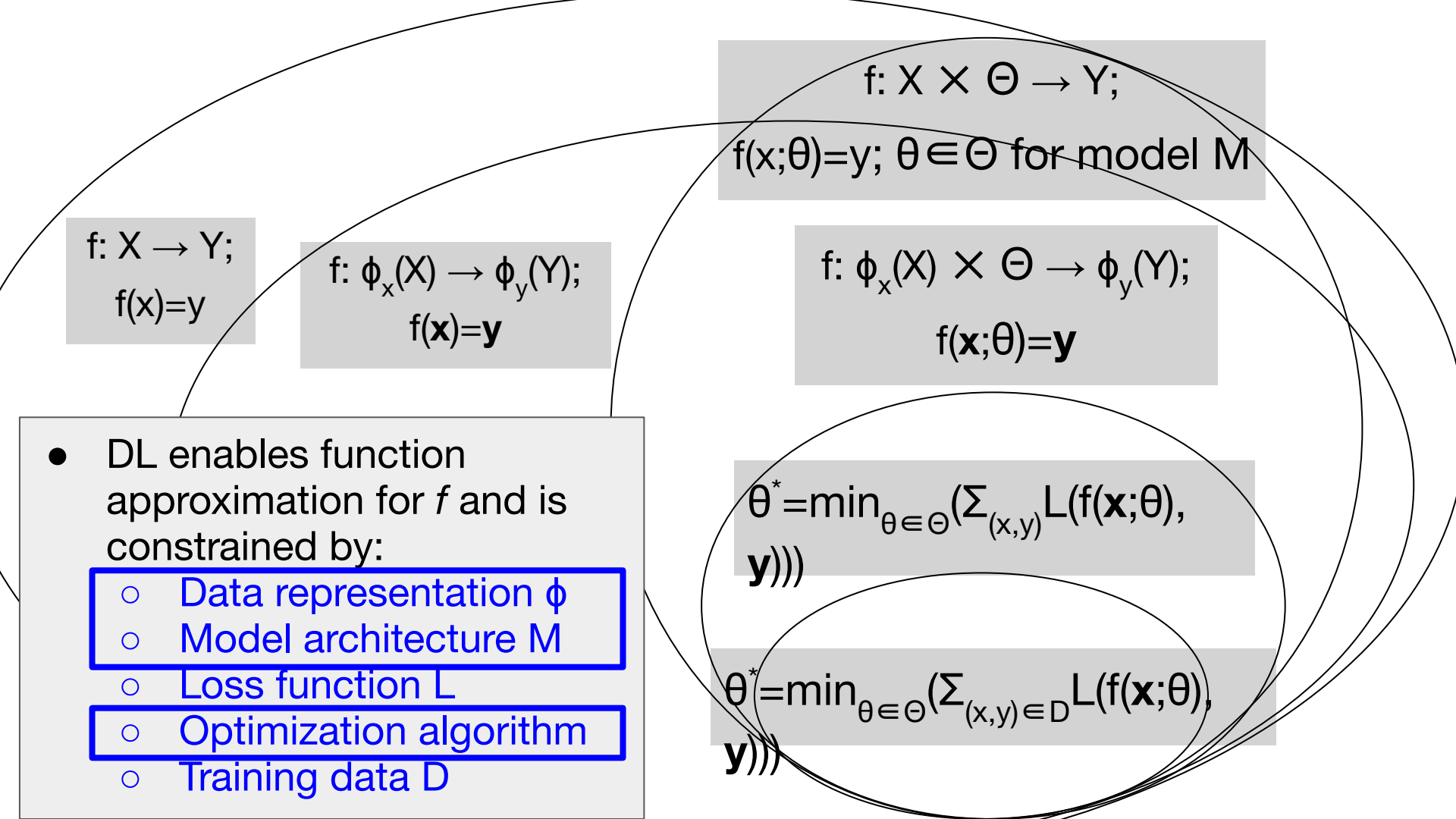


Next Week: Course Project Proposal Due [Feb 17]

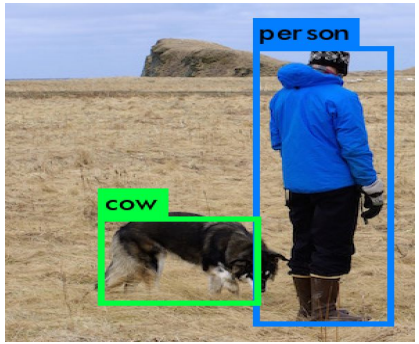
- **Proposal:** 5-6 *slides* on problem (scope & definition) + "today" (status of literature) + challenges + directions to innovate.
  - Look at the project ideas doc populated by the TAs to get a sense of what material your proposal should cover for problem definition and background
  - Include what you hope to be your *key insight or contribution*; what will make your approach unique?
- We will share a link to collect the slides.
- You will condense these longer, more detailed slides into a single lightning pitch slide for the Project Pitch assignment.

Overview of Today's Plan

- ~~Course organization and deliverables~~
  - Any questions before we move on?
- Activation Functions and Preprocessing
- Optimization and Regularization
- Cloud Computing Service Tutorial
- Deep Learning Software Tutorial
- Coding Assignment 1 Release Briefing


f: X → Y;
f(x)=y
f: $\phi_x(X) \rightarrow \phi_y(Y)$;
f(**x**)=**y**
f: X × Θ → Y;
f(x;θ)=y; θ∈Θ for model M
f: $\phi_x(X) \times \Theta \rightarrow \phi_y(Y)$;
f(**x**;θ)=**y**
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y)} L(f(\mathbf{x};\theta), \mathbf{y}))$
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(\mathbf{x};\theta), \mathbf{y}))$
• DL enables function approximation for *f* and is constrained by:
○ Data representation ϕ
○ Model architecture M
○ Loss function L
○ Optimization algorithm
○ Training data D

Representations and Models



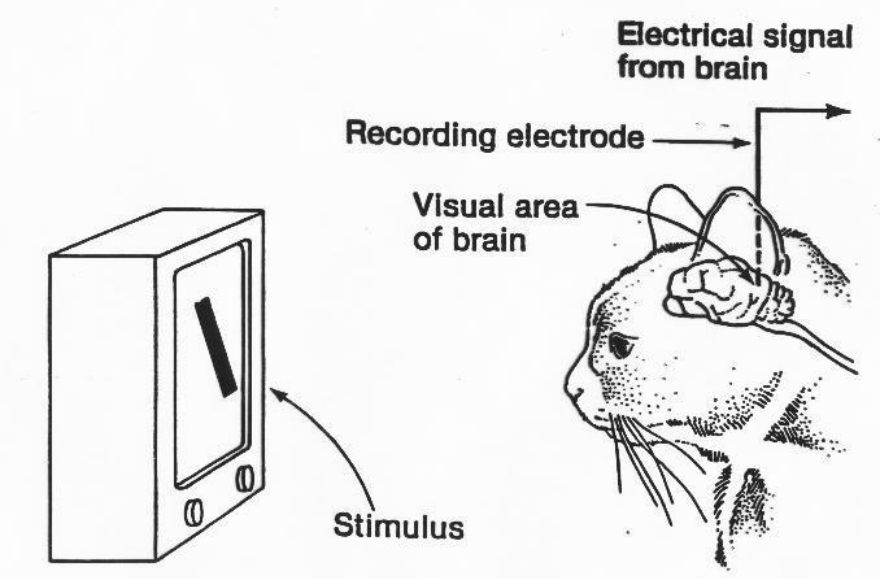
- Input space *X*?
  - Pixels in the image
- Representation?
  - $\phi_x(X)=\boldsymbol{R}^{W,H,3}$ RGB or HSV

- Output space *Y*?
  - Object classes *C* + bboxes
- Representation?
  - $\phi_y(Y) = C \times \boldsymbol{R}^{x,y,w,h} \times N$

- What model architecture might you use?
- What loss function would you want to optimize?

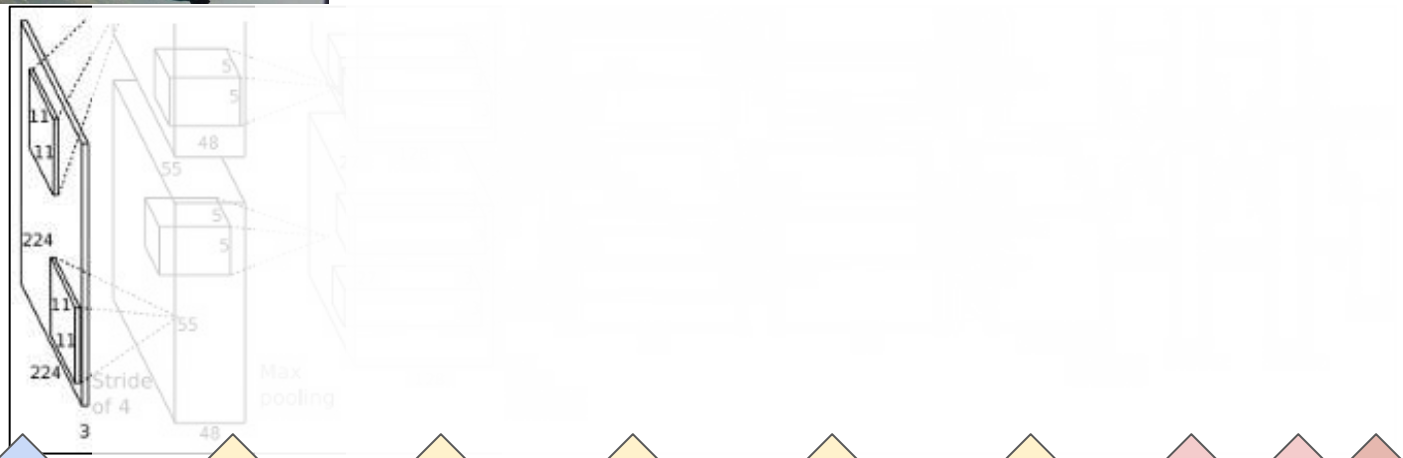
Origins of Convolutional Neural Networks

“They found that particular neurons in the visual cortexes of cats and monkeys—the areas in their brains responsible for processing visual information—didn't respond to simple points of light, but rather to lines, and in particular, lines and contours with specific orientations”



https://hub.jhu.edu/2015/07/14/golden-goose-award-wiesel-hubel/

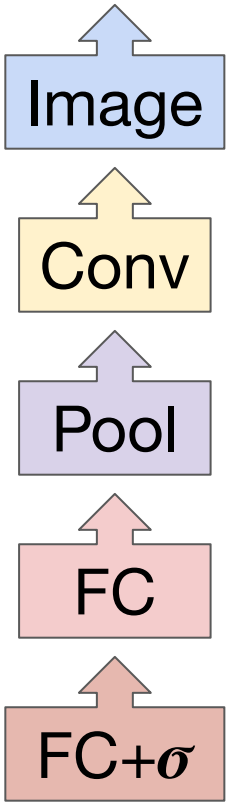
Breaking Down AlexNet



.01	car
.03	dog
.63	person
.11	airplane
.02	flower
…	…

Image
Conv
Conv
FC
FC+σ

Common Layers in a Deep Convolutional Neural Net

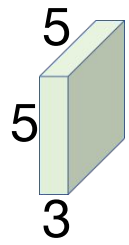


- **x** in our parlance; the vectorized input
- Convolution
- Pooling laye (in AlexNet, MaxPool)
- Fully-connected layer (i.e., *perceptron* layer)
- Softmax layer (fully connected layer + exponentiated normalization)

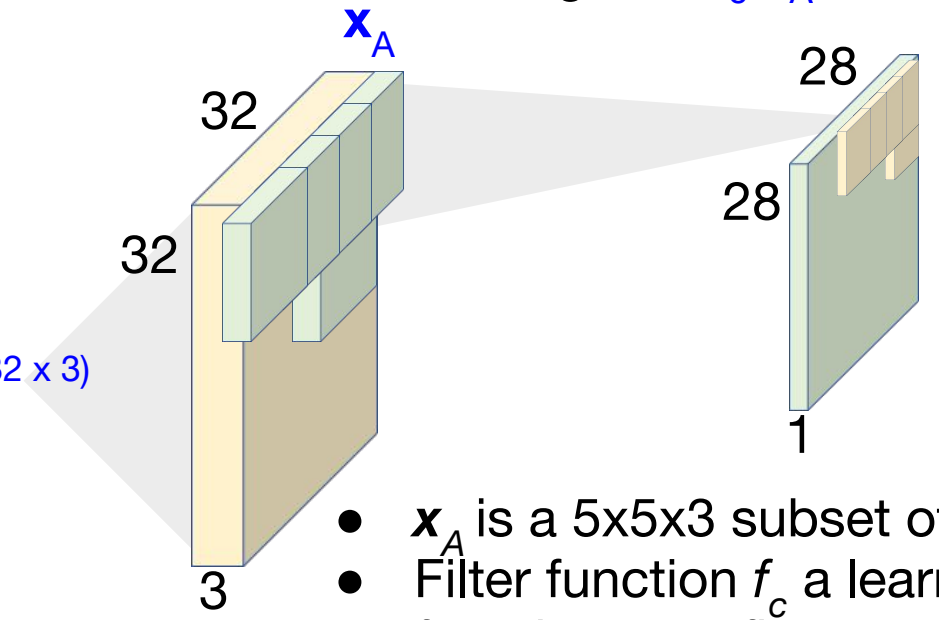
Each layer is a differentiable function whose *input* is the *output* of the previous layer (function)

Convolutional Layer

$\phi_x($  $) = [0, 1]^{(32 \times 32 \times 3)}$



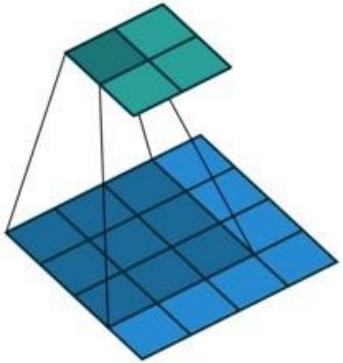
$f_c(\mathbf{x}_A) = W\mathbf{x}_A + \mathbf{b}$



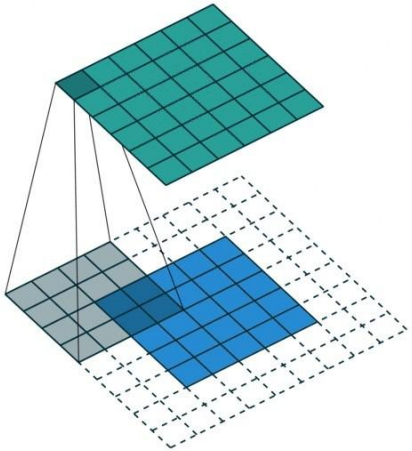
- $\boldsymbol{x}_A$ is a 5x5x3 subset of image tensor $\boldsymbol{x}$
- Filter function f_c a learned, linear function over flattened $\boldsymbol{x}_A$
- W is 1x(5*5*3), $\boldsymbol{b}$ is size 1, so f_c is a single number!

Convolutional Layer Terminology

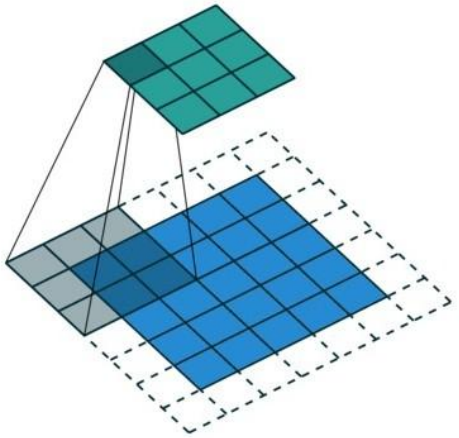
Padding and Stride



Pad=0; Stride=1
Kernel size?
3x3; lose 2 w/h

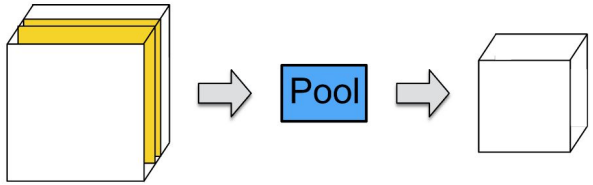


Pad=2; Stride=1
What to pad with?
Zeros common



Pad=1; Stride=2
Effect of stride?
Sparser output

Max Pooling Layer



M x N x P

m x n x P

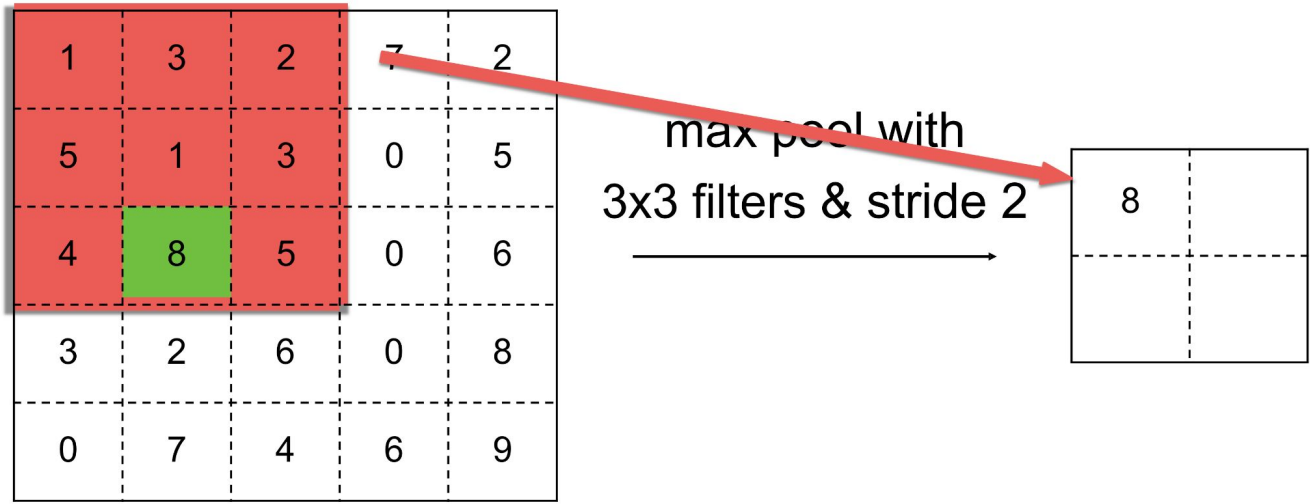
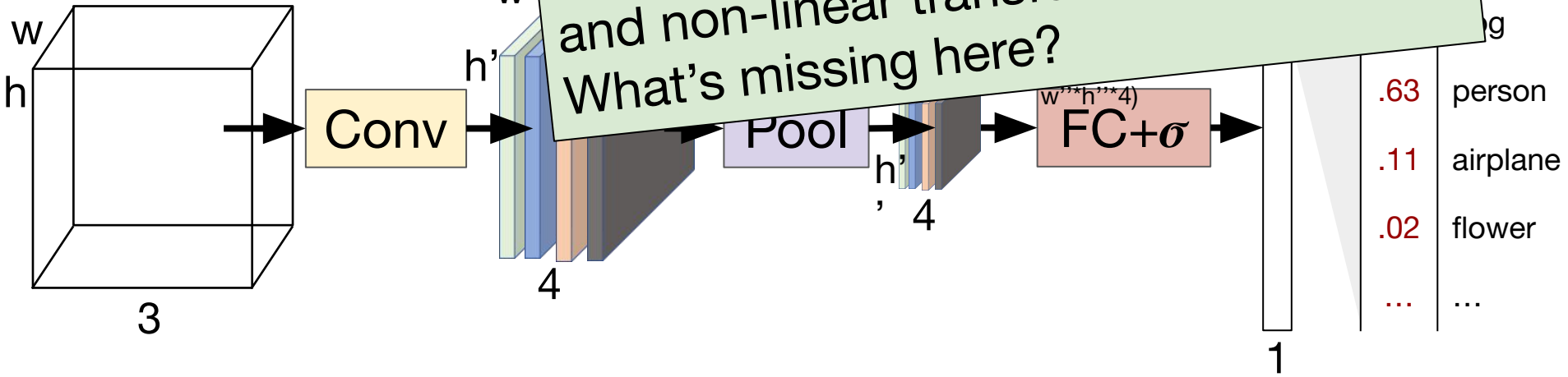


Image Classification with a Conv and Pool Layers

$\phi_x($  $) = [0, 1]^{(w \times h \times 3)}$

A neural network is a composition of a composition of matrix multiplications and non-linear transformations.
What's missing here?

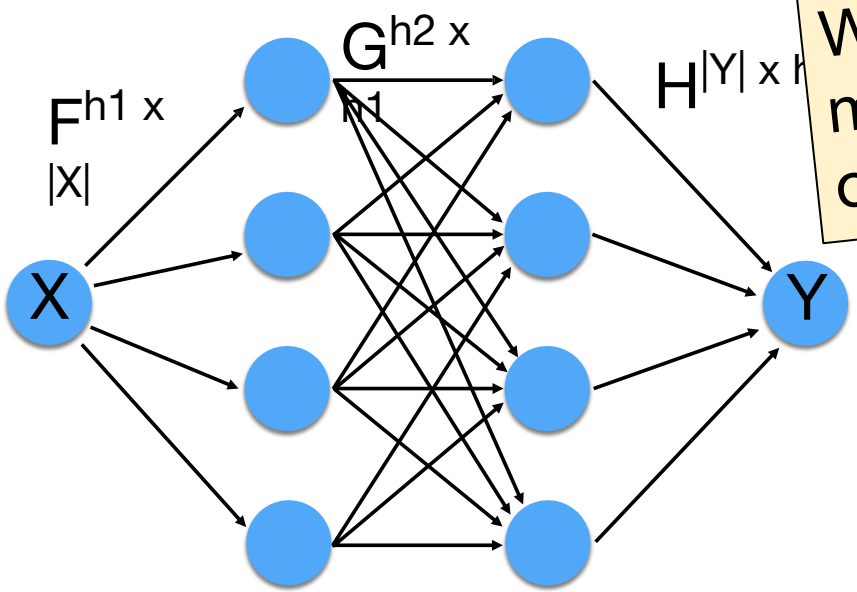


$f_c(\mathbf{x}_A) = W_c\mathbf{x}_A + \mathbf{b}_c$

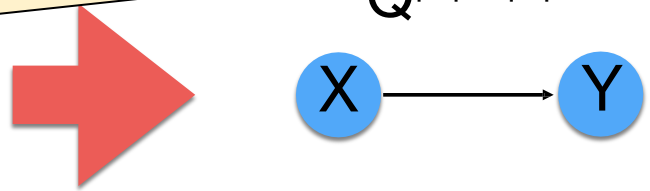
$f_p(f_c(\bullet)) = \max(f_c(\bullet))$

$FC(f_p(f_c(\bullet))) = W_{out}f_p(f_c(\bullet)) + \mathbf{b}_{out}$

Why Utilize Non-Linearities in a Deep Neural Network?



Without nonlinear activations, multiple layers are reduced to one matrix multiplication.

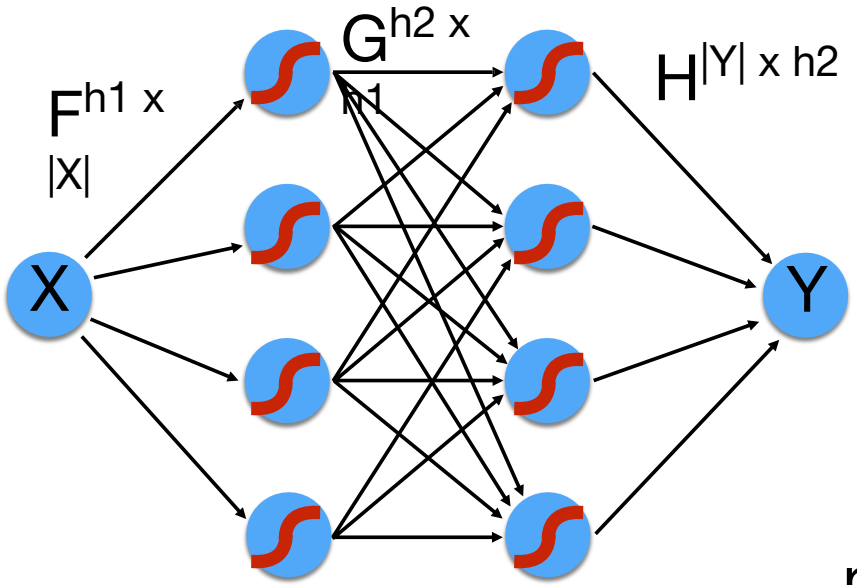


$Y=H(G(F(X)))$

Define $Q^{|Y| \times |X|} = HGF$

Then $Y=QX$

Why Utilize Non-Linearities in a Deep Neural Network?



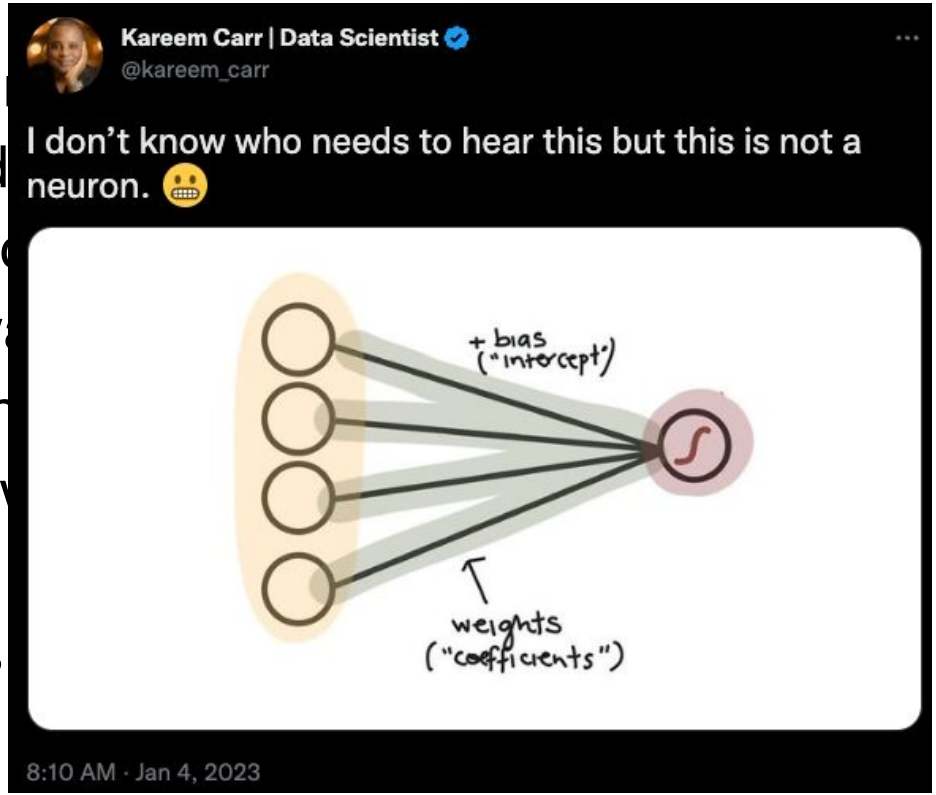
$Y=H(G(F(X)))$

$$F(\mathbf{x}) = a(W_F(\mathbf{x})+\mathbf{b}_F)$$
$$G(\mathbf{x}_{\mathbf{h1}}) = a(W_G(\mathbf{x}_{\mathbf{h1}})+\mathbf{b}_G$$

Activation function *a* is non-linear and not expressible as a matrix multiplication.

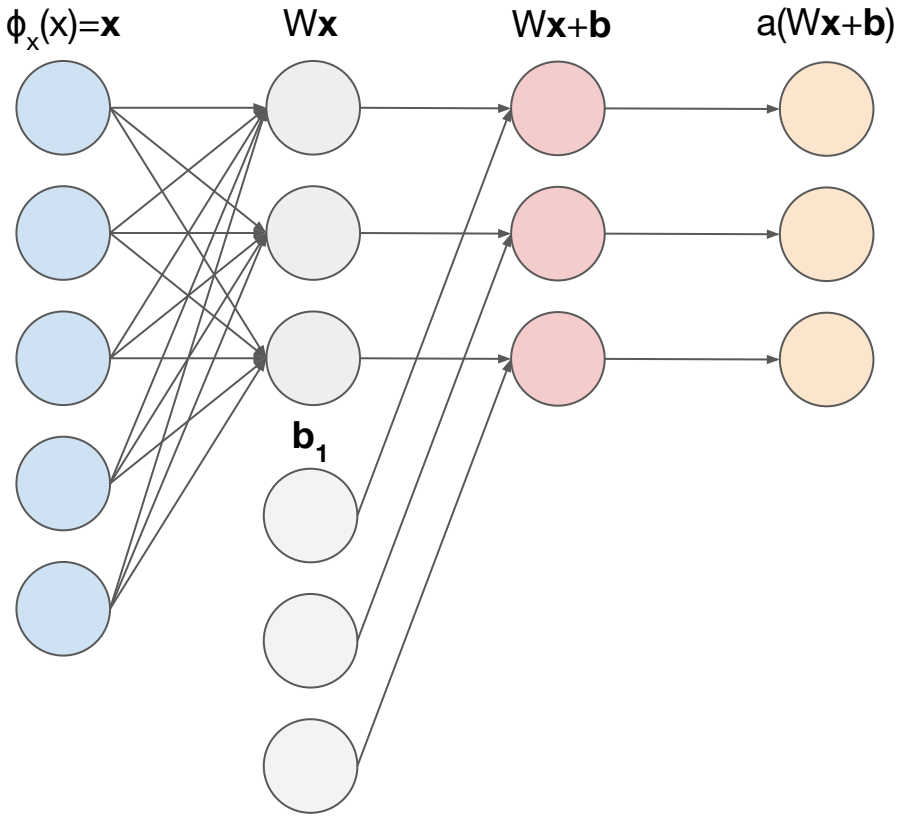
Deep Learning Borrows* Terminology from Neuroscience

- Brains are complex, intercon
- Neurons are connected via d
  through chemical and electri
- “Artificial neural networks” w
  variables “**activated**” by sign
- In a “feed forward neural net
  activated in order from input
- A “deep neural network” has

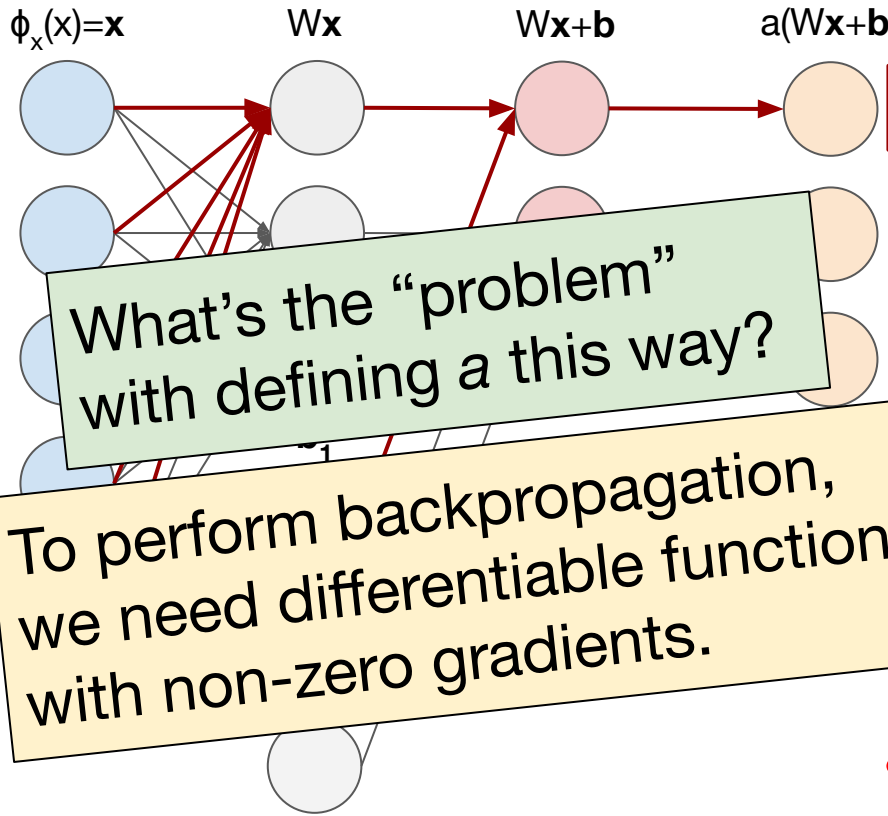


*Abuses, one might say.

Activation Functions



Activation Functions



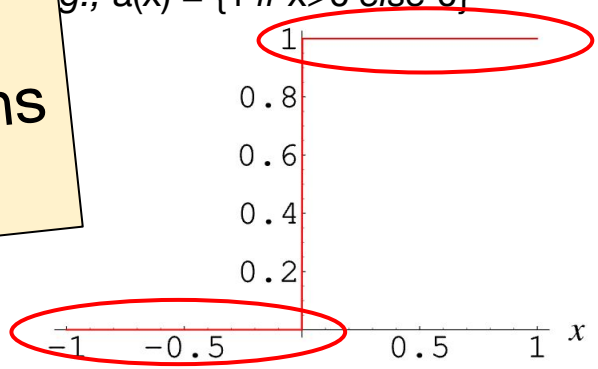
$= a(W_{1,1}\mathbf{x}_1 + W_{1,2}\mathbf{x}_2 + W_{1,3}\mathbf{x}_3 + W_{1,4}\mathbf{x}_4 + W_{1,5}\mathbf{x}_5 + \mathbf{b}_1)$

If we think about a "neuron" as inspiration, the output after *a* should be 1 if $W_{1,1}\mathbf{x}_1 + W_{1,2}\mathbf{x}_2 + W_{1,3}\mathbf{x}_3 + W_{1,4}\mathbf{x}_4 + W_{1,5}\mathbf{x}_5 + \mathbf{b}_1$ exceeds a threshold and 0 else.

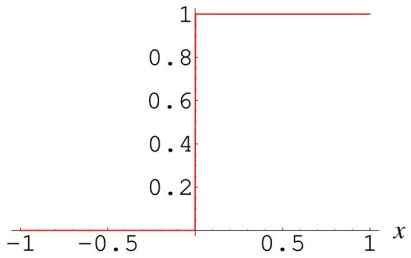
What's the "problem" with defining *a* this way?

To perform backpropagation, we need differentiable functions with non-zero gradients.

g., a(x) = {1 *if* x>0 *else* 0}

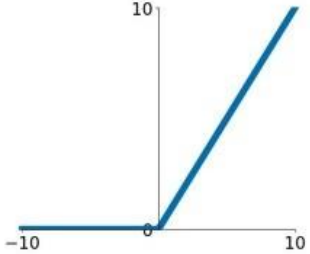


Activation Functions



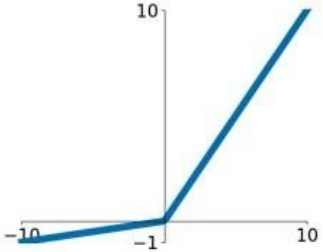
ReLU

$$\max(0, x)$$



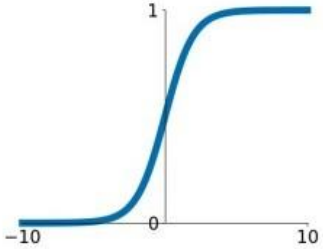
Leaky ReLU

$$\max(0.1x, x)$$



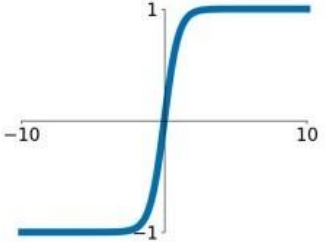
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

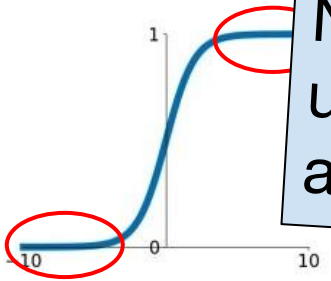
$$\tanh(x)$$



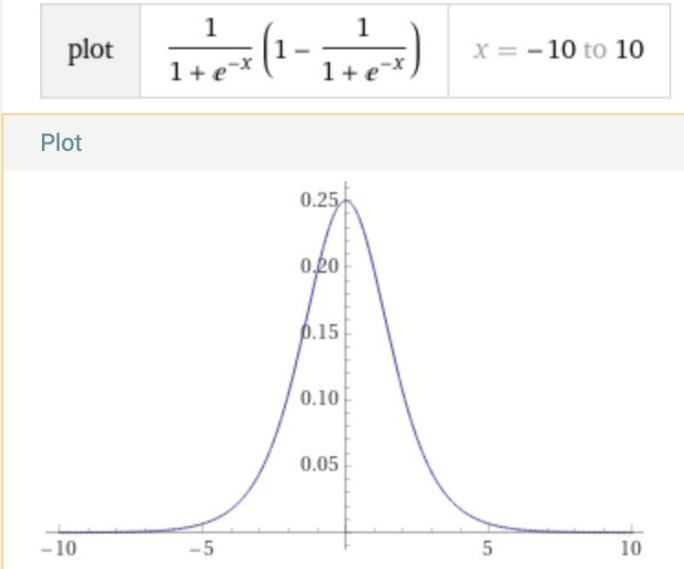
Activation Functions: Sigmoid

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Note: annoyi
uses σ to de
also sigmoid



- Advantages:
  - "Looks" a lot like our step function!
- Disadv
  - ywhere

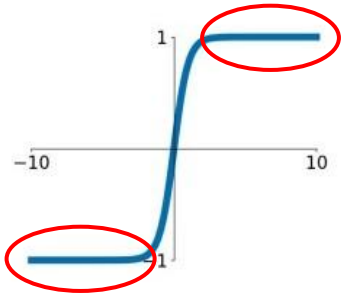
Learning can become prohibitively slow, especially if multiple sigmoids are involved across layers!

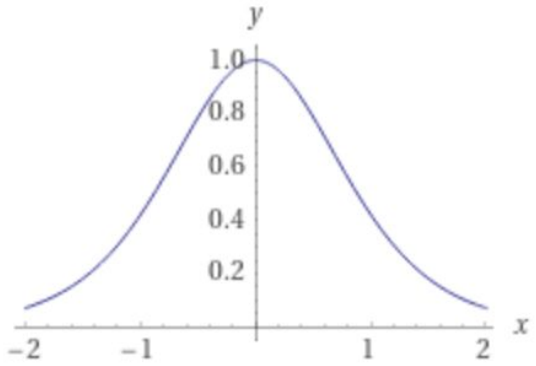
$\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$ [derivation]

Activation Functions: Hyperbolic Tangent

tanh

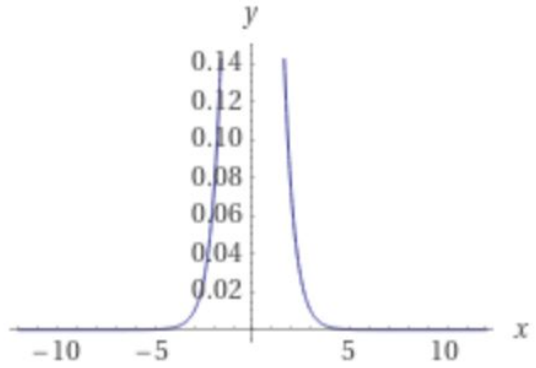
$\tanh(x)$





(x from −2 to 2)

- Advantages:
  - “Looks” a lot like our step fun
- Disadvantages:
  - Derivative is near 0 almost ev
  - What is $^{d}/_{dx}\tanh(x)$?
    - $^{d}/_{dx}\tanh(x) = \text{sech}^2(x)$

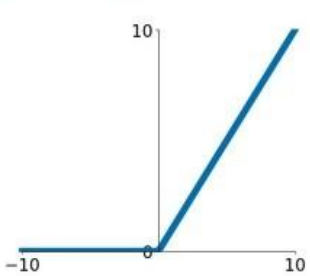


(x from −12 to 12)

Activation Functions: Rectified Linear Unit (ReLU)

ReLU

$\max(0, x)$



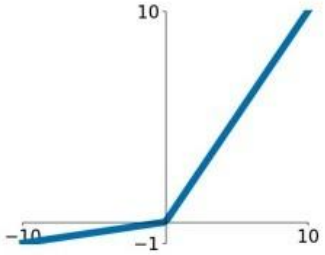
Learning can stall if activations are below zero because gradient stops flowing.

- Advantages:
  - For *x>0*, reliable and cheap to calculate gradient
  - What is $^{d}/_{dx}\max(0, x)$?
    - $^{d}/_{dx}\max(0, x) = \{1 \textit{ if } x>0; \textit{ else } 0\}$
- Disadvantages:
  - Derivative is exactly 0 for *x<0*
  - Output in the positive direction is unbounded

Activation Functions: Rectified Linear Unit (ReLU)

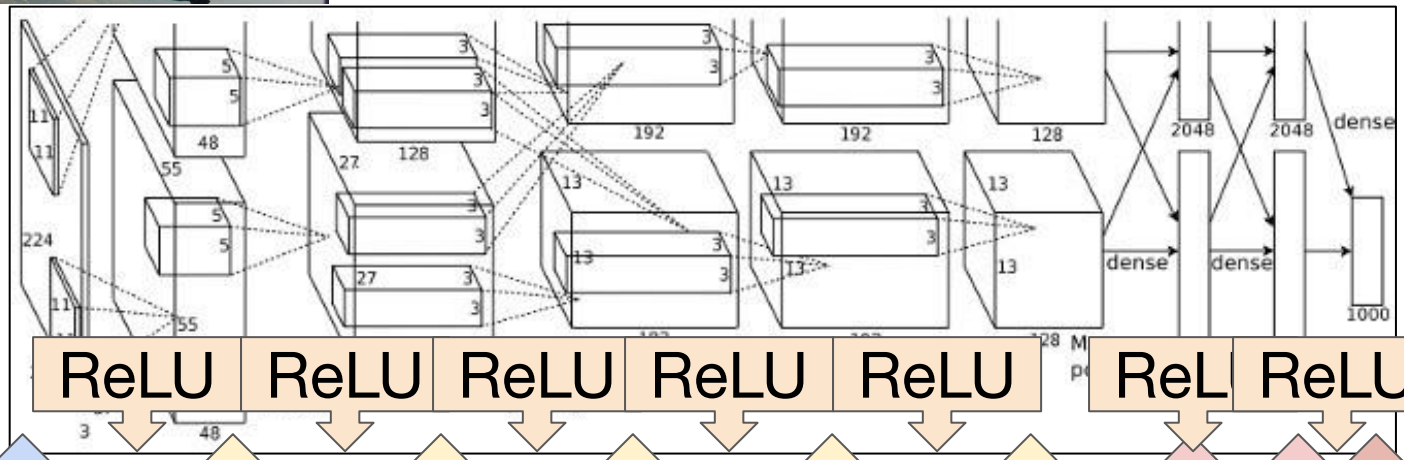
Leaky ReLU

$$\max(0.1x, x)$$



- Advantages:
  - Reliable and cheap to calculate gradient
  - What is $^{d}/_{dx}\max(0.1x, x)$?
    - $^{d}/_{dx}\max(0.1x, x)$ = {1 *if* x>0; *else* 0.1}
- Disadvantages:
  - As with ReLU, outputs are still *unbounded* and could cause problems in later layers

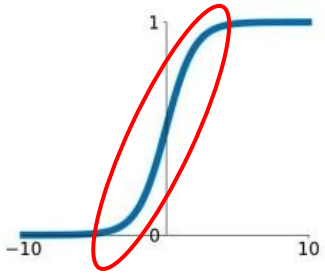
With These Layers, We Can Compose AlexNet!



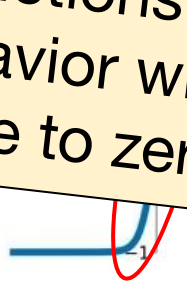
What Assumption Do Our Activation Functions Make?

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



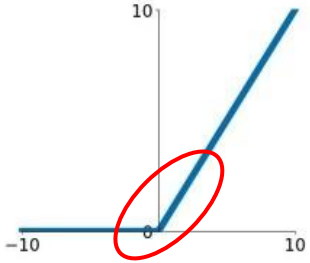
$\tanh(x)$



Our activation functions are all on their best behavior when the input *x* is close to zero.

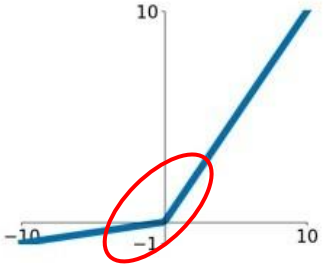
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



Weight Initialization: Zeros

- Since input to activation functions behaves well around zero, how about we start all our weights at zero?
- Initialize all weight matrix *W* and bias vector ***b*** parameters are initialized to zero for all layers (e.g., fully connected, CNN filters, etc.):
  - What is the effect on the final output vector?
    - Guaranteed to be all zeros for any input!
  - Gradients will vanish; model can’t learn

Weight Initialization

- We want well-behaved, random initial parameter values that are near zero but not uniformly zero
- What's the easiest way to accomplish that goal?
  - Draws from a Gaussian distribution!

$$W \sim \mathcal{N}(\mu, \sigma^2)$$

- What's a good value for μ?
  - $\mu=0$
- What's a good value for σ^2?
  - $0<\sigma^2<1$, e.g., $\sigma^2=0.01$

Weight Initialization: Gaussian Distribution

- Draws from a Gaussian distribution

$$W \sim \mathcal{N}(\mu, \sigma^2)$$

- Advantages:
  - Easy to implement
- Disadvantages:
  - Hyperparameter we need to set: σ^2
  - If the weights are too small, the activation signals shrink quickly during training
  - If the weights are too large, the activation signals explode during training

Weight Initialization: Controlling Activation Variance

- Glorot and Bengio argue that when the variance of the layer outputs (and hence the downstream layer inputs) is not ≈1, depending on the activation function, models will converge more slowly, especially when the layer output variance is <1
- “Fixes” that initialize parameters to encourage layer outputs whose variance is ≈1 depend the activation function

Understanding the difficulty of training deep feedforward neural networks

Weight Initialization: Controlling Activation Variance

- Activation functions with centered average (sigmoid; tanh):
  - Set the variance to inverse of the number of input units that contribute to each logit in the layer [*Glorot Initialization*]

$$W \sim \mathcal{N}(\mu, \sigma^2) \longleftarrow Var(W) = \frac{1}{n_{in}}$$

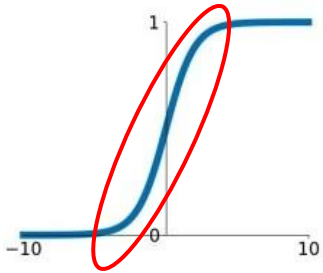
- Activation functions with unbounded average (ReLU):
  - Because average activation is a higher positive number, need to adjust up to $Var(W) = {}^{2}/_{n\_in}$ [*Kaiming initialization*]

Xavier Glorot Initialization in Neural Networks — Math Proof [blog post on Glorot]
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification [Kaiming He paper]
L11.6 Xavier Glorot and Kaiming He Initialization [Brief Lecture on Differences Between These]

What Assumption Do Our Activation Functions Make?

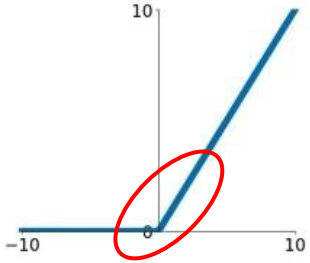
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



ReLU

$$\max(0, x)$$

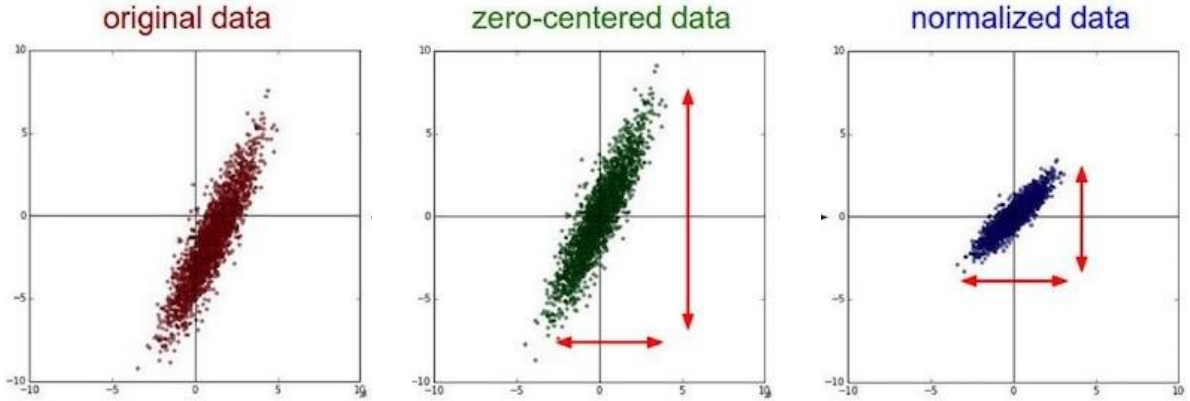


Our activation functions are all on their best behavior when the input *x* is close to zero.

What aspect of the path from input to output won't weight initialization alone help with?

Input Representation and Data Preprocessing

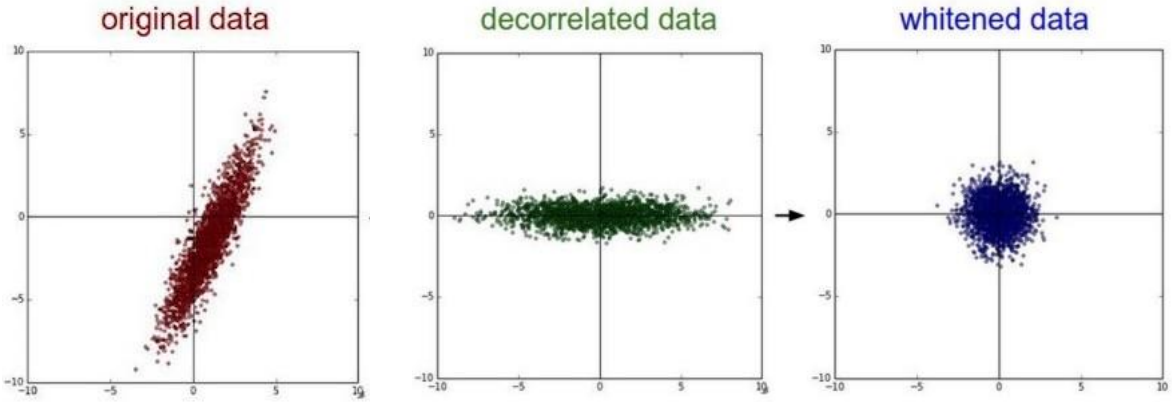
- We can make our input representation ϕ a function that considers the entire training dataset to *standardize* it
- Zero-centering: shift the data to have features with mean 0
- Normalization: adjust features to have variance 1
  - (e.g., *z*-score normalization)



$$z = \frac{x - \mu}{\sigma}$$

Input Representation and Data Preprocessing

- We can make our input representation ϕ a function that considers the entire training dataset to *standardize* it
- Remove correlations between input features via dimensionality reduction (e.g., PCA)



PCA Whitening [blog]

Overview of Today's Plan

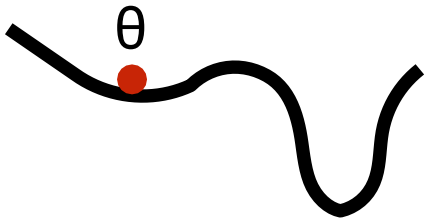
- ~~Course organization and deliverables~~
- ~~Activation Functions and Preprocessing~~
  - Any questions before we move on?
- Optimization and Regularization
- Cloud Computing Service Tutorial
- Deep Learning Software Tutorial
- Coding Assignment 1 Release Briefing

Gradient Descent

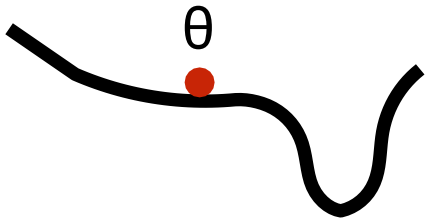
- Vanilla (true) Gradient Descent $\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} f(\theta_t)$
  - Estimate θ over the whole dataset
- Stochastic Gradient Descent (SGD) $\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} f(\theta_t; x^{(i)})$
  - Estimate θ at a random datapoint
- Minibatch Gradient Descent $\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} f(\theta_t; x^{(i:i+n)})$
  - Estimate θ as an average over a random batch of training data
  - Commonly we use “SGD” to refer to this setup

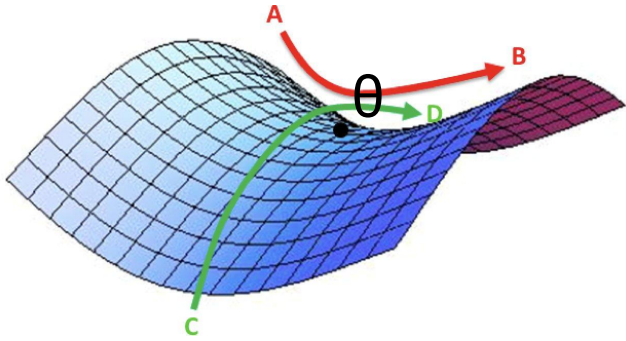
Saddles and Local Minima

- Gradient descent drives the parameters θ to towards the closest low point in the loss space
- Gets “stuck” if the loss is locally minimal (basin) or the gradient goes in orthogonal directions across dimensions (saddle)



local minimum





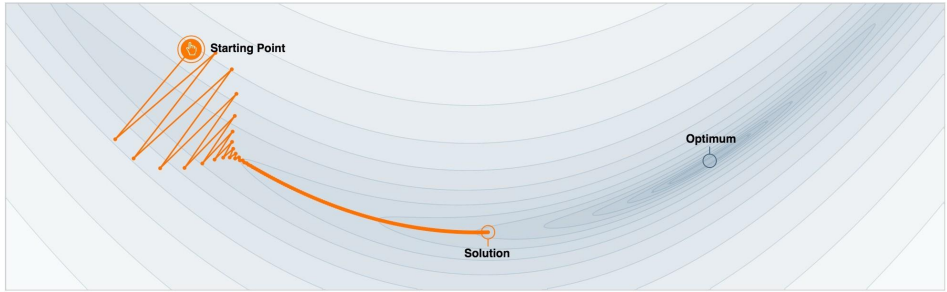
saddle points

Gradient Descent with Momentum

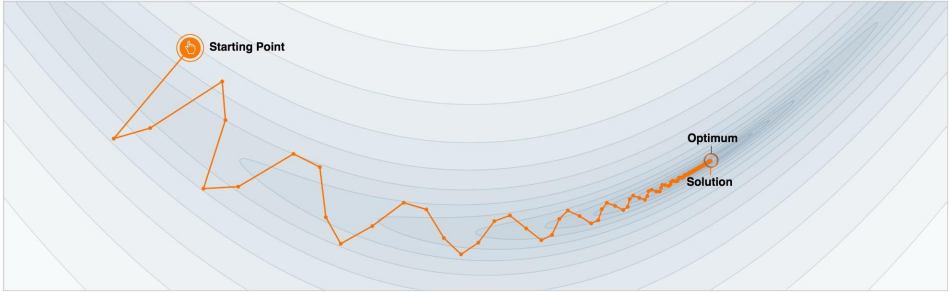
- Consider the history:
  - Add a fraction γ of the update vector of the past time step to the current update vector

$$\nu_t = \gamma\nu_{t-1} + \eta\nabla_{\theta_t} f(\theta_t) \qquad \theta_{t+1} = \theta_t - \nu_t$$

GD *without* momentum



GD *with* momentum

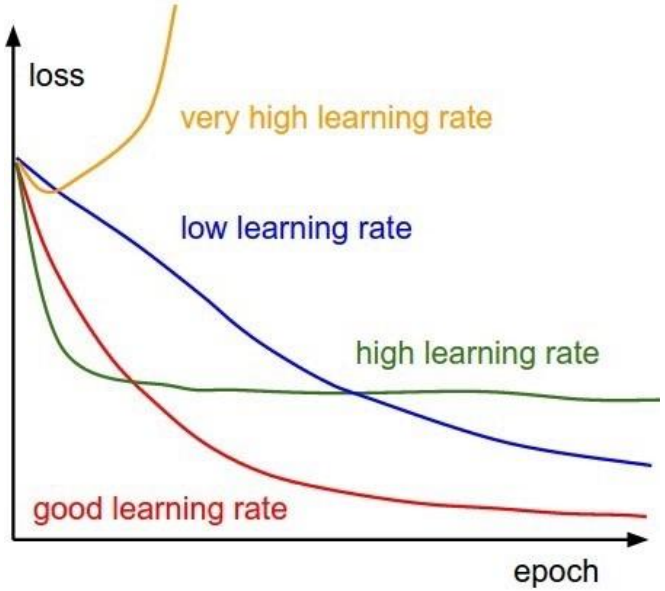


Optimizer Parameters: Learning Rate

- How fast or slow we update the model parameters

$$\theta_{t+1} = \theta_t - \eta_t \nabla f(\theta_t)$$

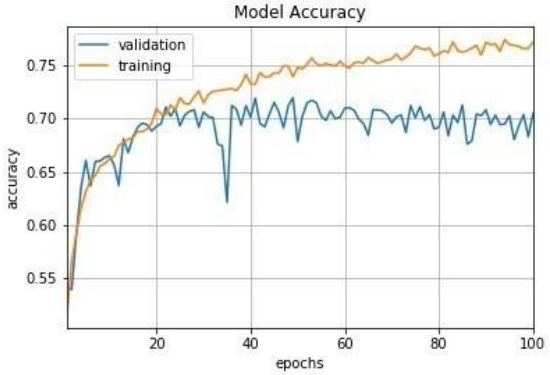
- If the learning rate is too large, the model parameters “explode” (move too fast; gain in magnitude)
- If the learning rate is too small, the model parameters move slowly and may be unable to escape local minima



Optimizer Parameters: Learning Rate

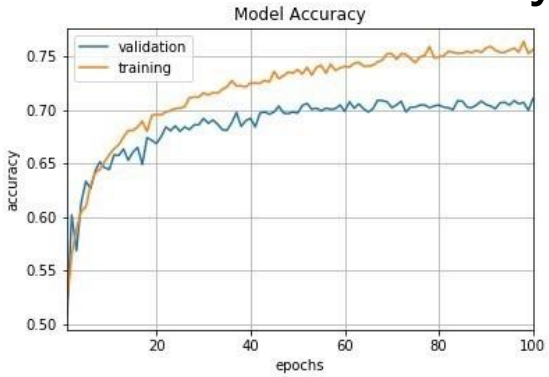
- Learning rate *schedulers* change the LR during training

Constant



$$lr = lr_0$$

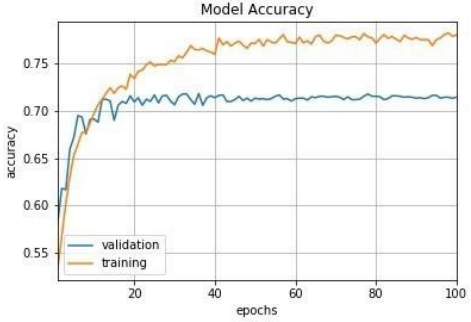
Time-based decay



$$lr = \frac{lr_0}{1 + k * iter}$$

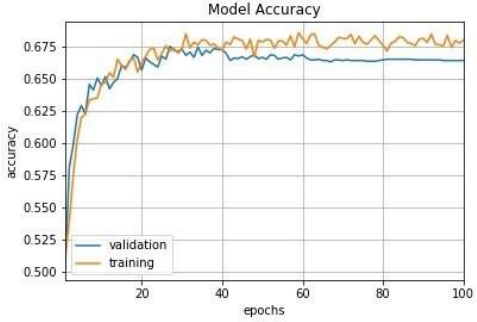
Optimizer Parameters: Learning Rate

Step decay



$$lr = lr_0 * floor(\frac{epoch}{epoch\_drops})$$

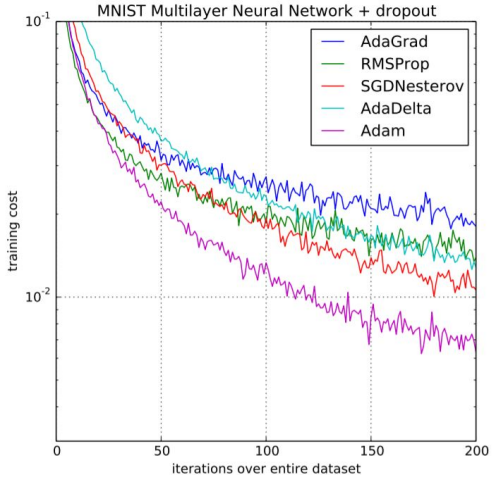
Exponential decay



$$lr = \frac{lr_0}{\exp^{k*iter}}$$

Optimizer Selection In Practice

- Aside from tutorial code, you probably haven't seen "SGD" as the selection for an optimizer
- SGD with momentum is the backbone of most optimizers, with adaptive learning rate schedulers cooked in
- So which optimizer is best?
  - It depends!
- In practice, the optimizer and its parameters are *hyperparameters* you'll jitter around when training a network



Parameter Regularization During Training

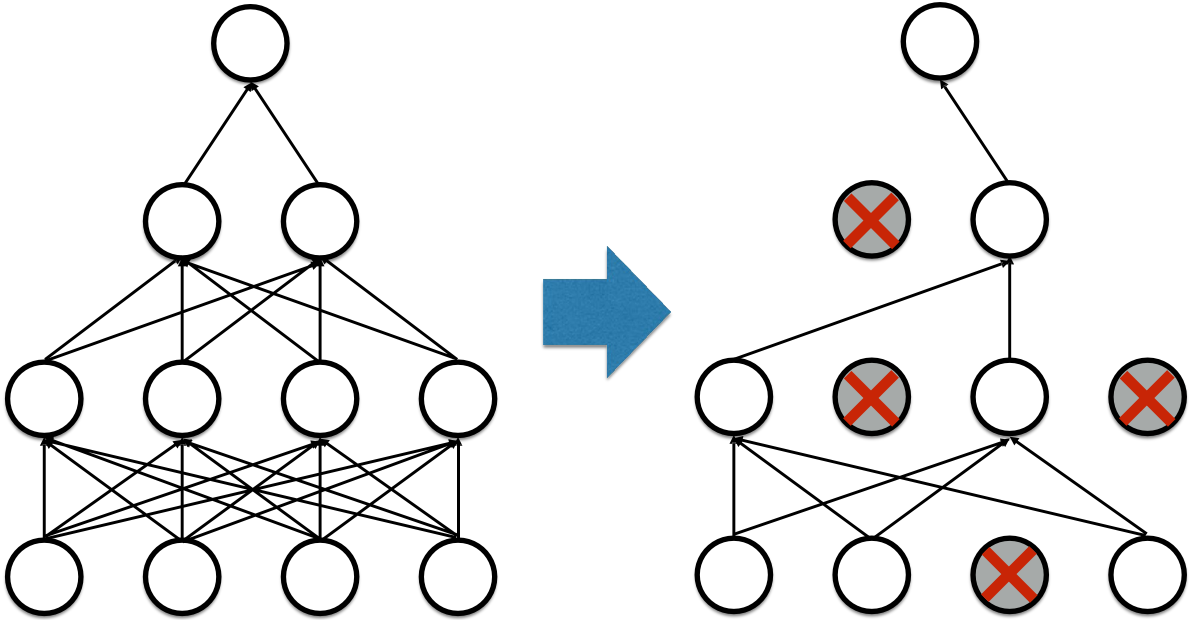
- We can add a regularization “penalty” to our loss *L*

$$L = L_{original} + \lambda R(\theta)$$

- L1 parameter regularization $R(\theta) = \Sigma_i \Sigma_j |\theta_{i,j}|$
- L2 parameter regularization $R(\theta) = \Sigma_i \Sigma_j \theta_{i,j}^2$
- L1+L2 parameter regularization $R(\theta) = \Sigma_i \Sigma_j |\theta_{i,j}| + \alpha \theta_{i,j}^2$

Implicit Ensemble Models: Dropout

- At each iteration, randomly shutdown some neurons with a certain probability (usually 0.5)



Implicit Ensemble Models: Dropout

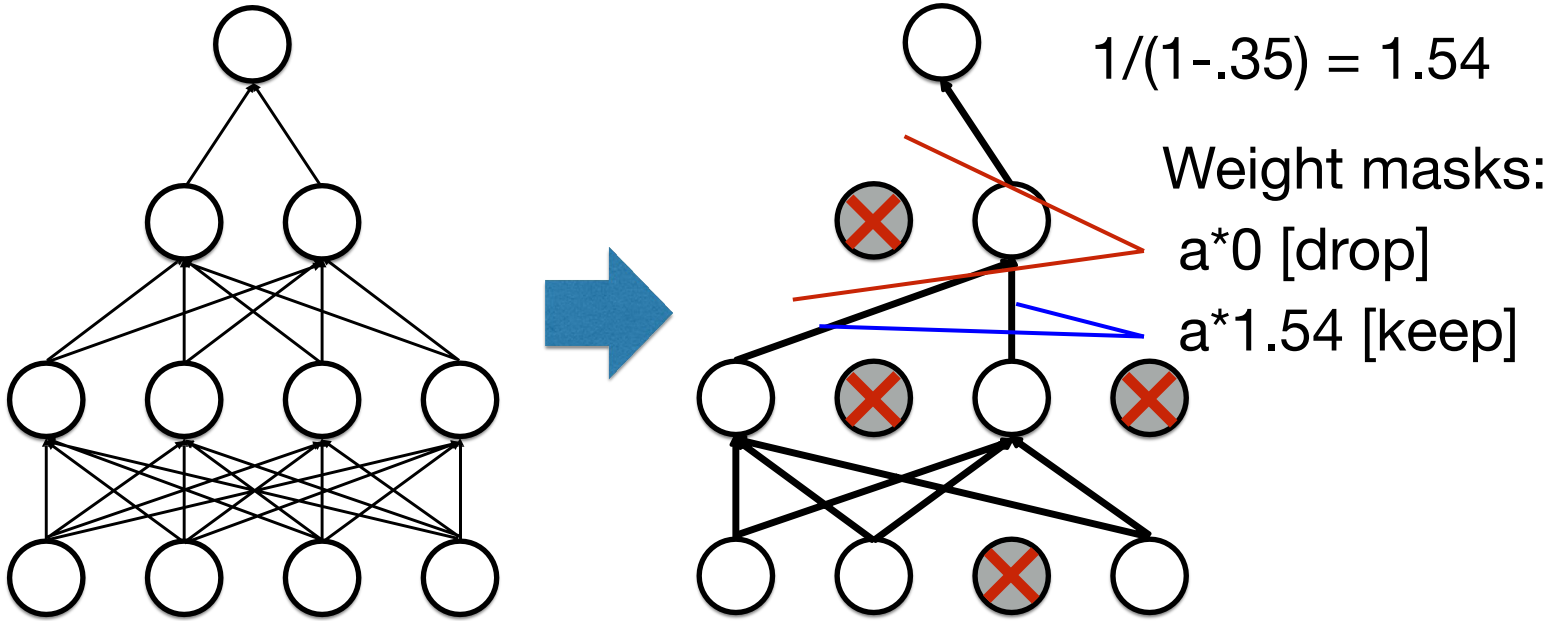
- At each iteration, randomly shutdown some neurons with a certain probability (usually 0.5)
- Intuition:
  - Forces the network to have a redundant representation
    - Same pieces of information are “stored” in multiple different neurons
  - Training a bunch of small models that share parameters
  - Each binary mask is one such “small model”
- At inference time, these models are “ensembled”

Training Versus Inference Considerations for Dropout

- If we randomly drop 0.5 nodes during training time but use all during test time, what might go wrong?
  - Test time inputs at each layer will be, on average, x2 what was seen at training time!
- To compensate, during training time we apply an *inverted dropout* correction
  - At training time, for every node not dropped out, increase its output by a magnifying factor of $1/(1-d)$ for d probability of dropping each unit; e.g., $x10$ for dropout 0.9

Training Versus Inference Considerations for Dropout

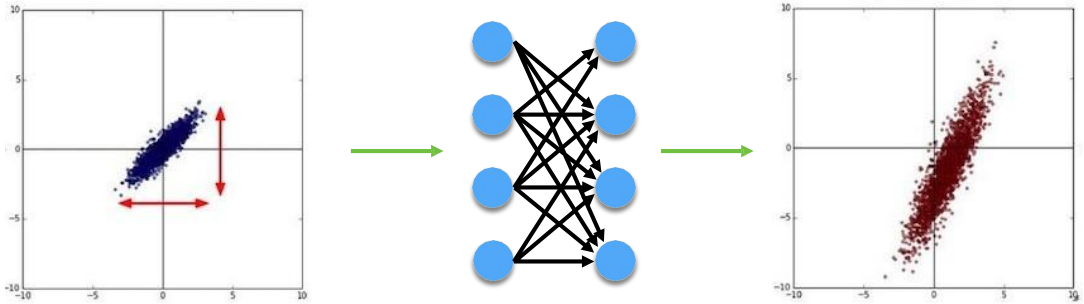
- To compensate, during training time we apply an *inverted dropout* correction; e.g., for dropout value 0.35:

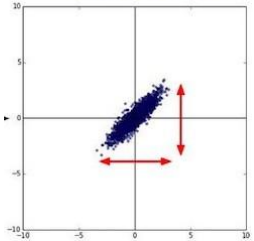


Normalization Between Layers

- After each layer, the distribution of activation signals changes (Internal covariate shift)
- As a network becomes deeper, distribution shifts more
- Just as we preprocess the input data with our representation function to make features well-behaved, we can consider re-normalizing outputs per layer

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$





BatchNorm Layer

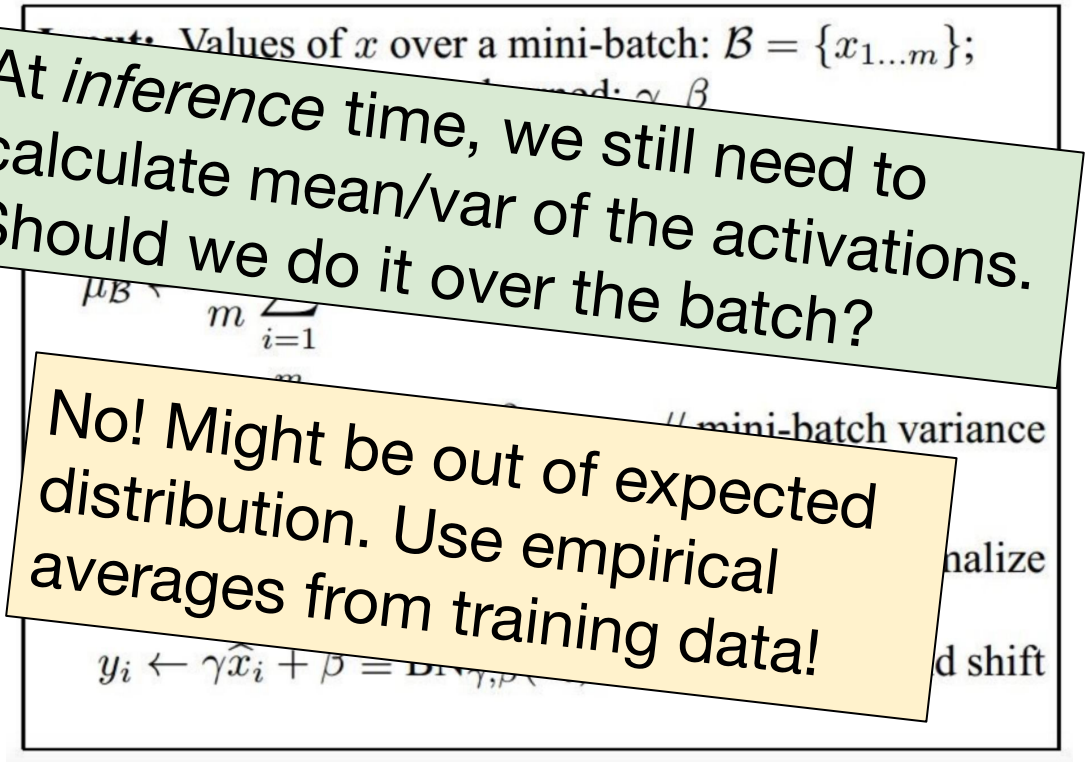
- At training time, compute the estimated mean and variance of the activations of a layer across the current batch

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}} \qquad \mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

- How could we make this transformation a little more flexible?
- Introduce learnable parameters to the BatchNorm Layer that enable *scaling* and *shifting* individual input indices

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [paper]

BatchNorm Layer



At *inference* time, we still need to calculate mean/var of the activations. Should we do it over the batch?

No! Might be out of expected distribution. Use empirical averages from training data!

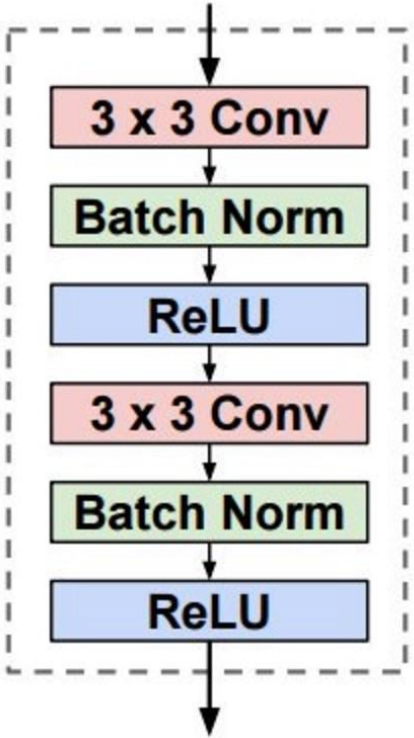
$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)}$$

- Gamma and Beta are learnable vectors
- These params allow the transformed input to have flexible, learned mean & var
- BatchNorm layer is differentiable

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [paper]

BatchNorm Layer

- Usually added after convolutional layers or fully connected layers but before non-linearities
- Why?
  - Point of BatchNorm is to help data get into a space where the activation function behaves well



Perceptual Losses for Real-Time Style Transfer and Super-Resolution [paper]

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Activation Functions and Preprocessing~~
- ~~Optimization and Regularization~~
    - Any questions before we move on?
- Cloud Computing Service Tutorial
- Deep Learning Software Tutorial
- Coding Assignment 1 Release Briefing

Action Items for You

- If you have not registered with a team, do so **ASAP**; the registration is currently a week late
- Start firming up your project proposal ideas and drafting your slides; these proposal slides are due to us in a week
- Coding Assignment 1 will be discussed by TAs and is due two weeks from today
- Keep in mind also that the midterm exam will be in two weeks; the exam will be mostly multiple choice and submitted via a google form; it must be taken during the class period

CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 4: Activations, Initializations, Optimization, and Regularization