Regularization and more topics related to training

Deep Learning

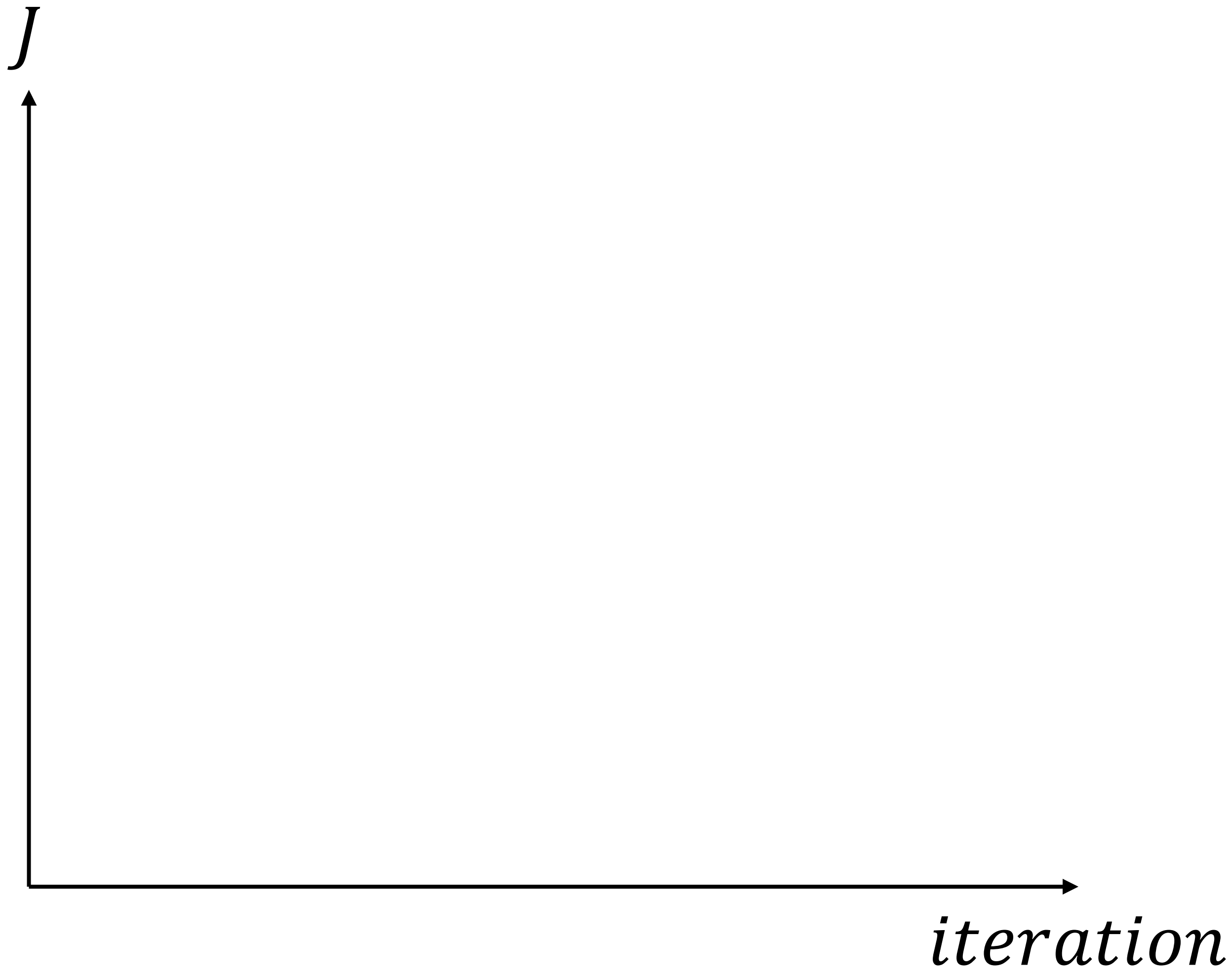
[Brad Quinton](), [Scott Chin]()

Learning Objectives

- Discuss Regularization strategies for combating overfitting
  - L2 Regularization
  - Dropout
  - Data Augmentation
- Discuss strategies for tuning hyperparameters
- Looking at cost curves for hints at where things can go wrong
- Transfer Learning

Overfitting

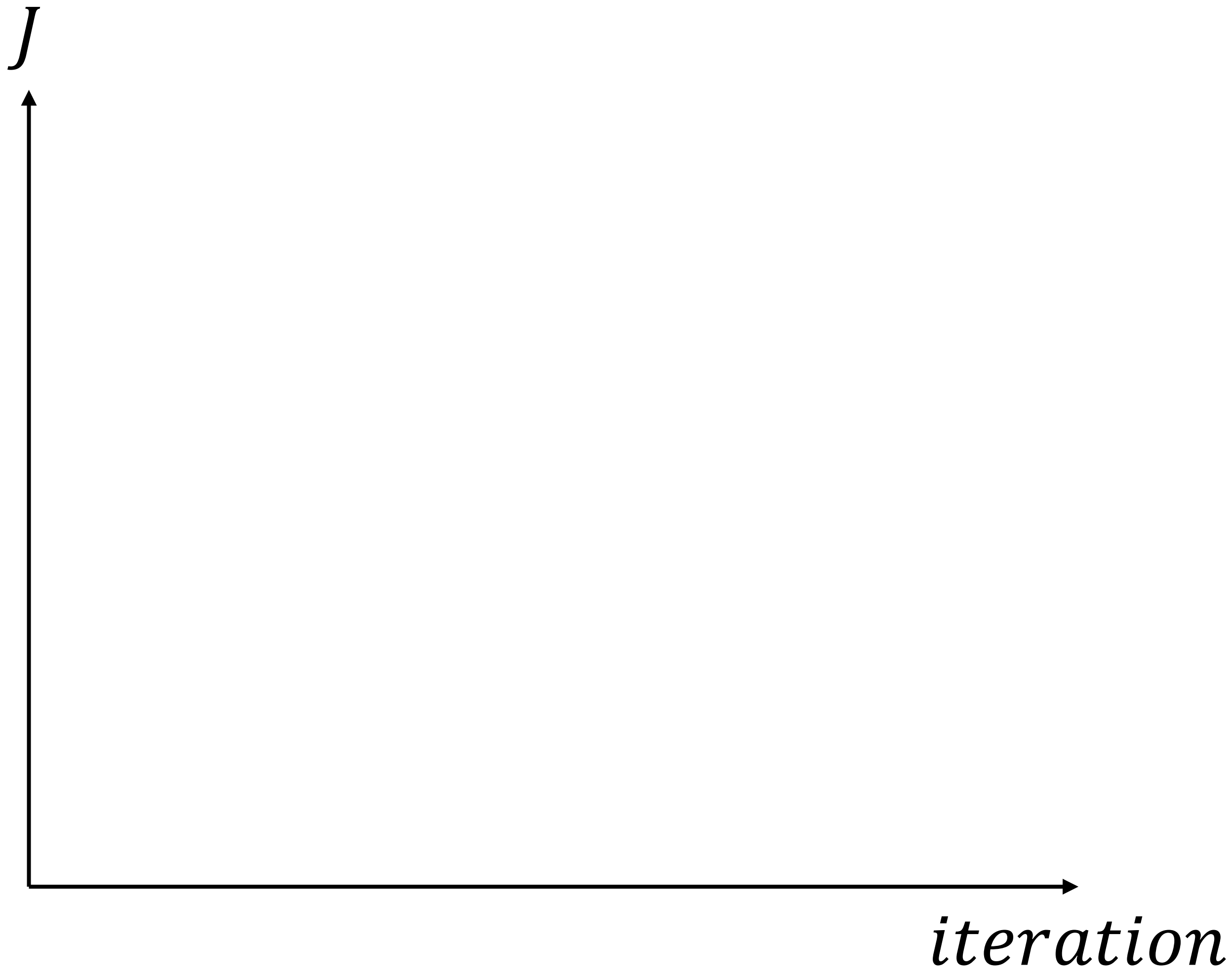
What to do when model overfits



Overfitting

What to do when model overfits

- Get more training data
- Regularization



Brad Quinton, Scott Chin

Regularization via Cost Function

- Recall that the Cost (Objective) function is used to describe the qualities that we want in our solution (i.e. parameter values)
- So far, Cost function only accounts for prediction loss.

$$J = \frac{1}{m} \sum_{j=1}^{m} L(\hat{y}, y)$$

Cost function so far

Brad Quinton, Scott Chin

Regularization via Cost Function

- Recall that the Cost (Objective) function is used to describe the qualities that we want in our solution (i.e. parameter values)
- So far, Cost function only accounts for prediction loss.
- Can add additional terms to encourage regularization in our solutions

$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y)$$

Cost function so far

$$J = \left(\frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y)\right) + R$$

Cost function with Regularization Term

Brad Quinton, Scott Chin

L2 Regularization

$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y) + \sum w^2$$

- Regularization Term: Sum the square of each parameter value
  - Cost will be minimized when each parameter value is small
  - Convex function that is independent of the training set
  - The regularization term has a global minimum when all weights are 0

Brad Quinton, Scott Chin

L2 Regularization

$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y) + \sum w^2$$

- Regularization Term: Sum the square of each parameter value
  - Cost will be minimized when each parameter value is small
  - Convex function that is independent of the training set
  - The regularization term has a global minimum when all weights are 0
- Remember, trying to minimize the Loss AND the Regularization terms
  - Loss term will likely be large if all weights are 0
  - So these are competing terms

L2 Regularization

$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y) + \lambda \sum w^2$$

- Since we have two competing terms, we can specify the importance of each term based on the new λ hyperparameter
- $\lambda = 0$ Means we don't optimize for regularization
- $\lambda = \infty$ Means we don't optimize for Loss
- Need to choose (tune) a value somewhere in between
- Default in Keras is 0.01

Brad Quinton, Scott Chin

L2 Regularization

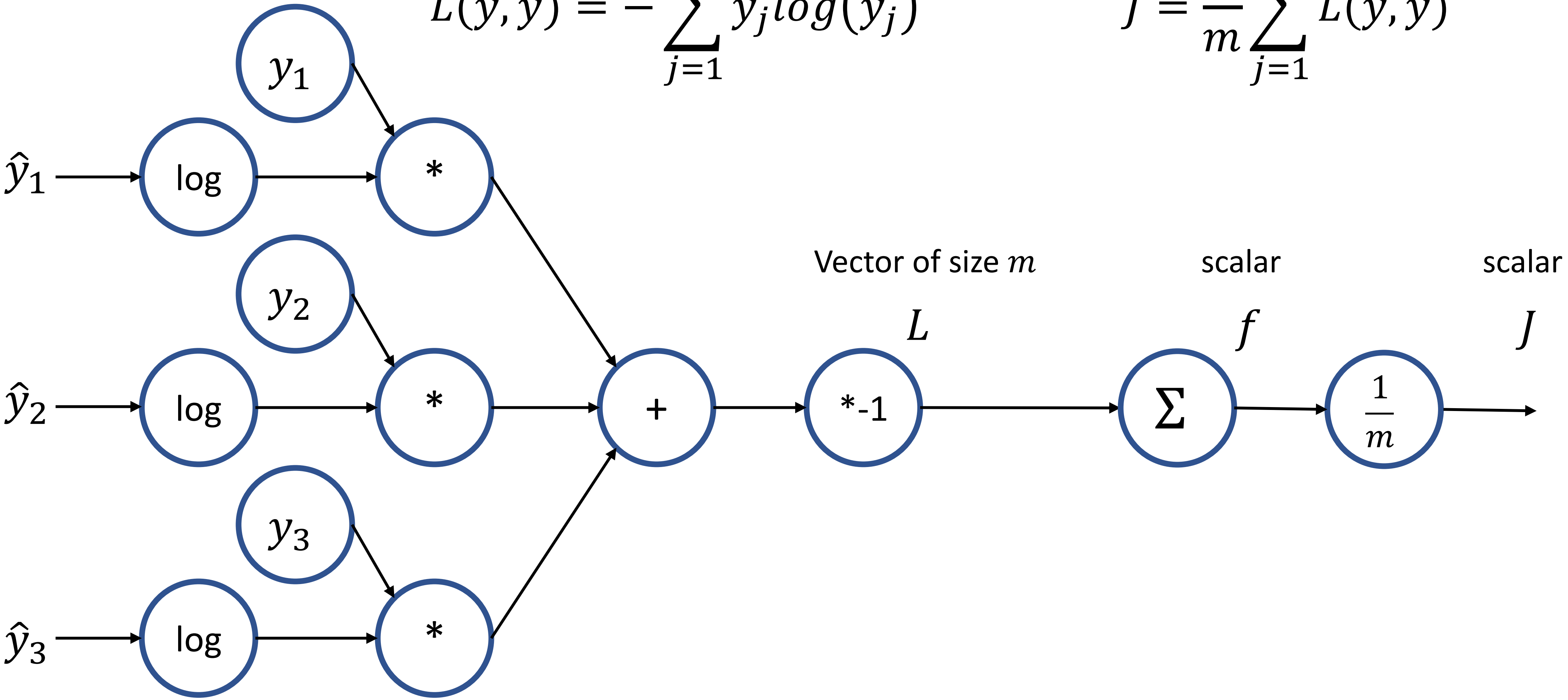
$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y) + \lambda \sum w^2$$

- Not the only way to encourage regularization in cost function
- It's the most popular one.
- Special Name: L2 Regularization
- Sometimes referred to as Weight Decay
- Why does it help reduce overfitting? Intuition next …

Recall: Categorical Loss Cost Function

$$L(\hat{y}, y) = -\sum_{j=1}^{n_c} y_j log(\hat{y}_j)$$

$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y)$$

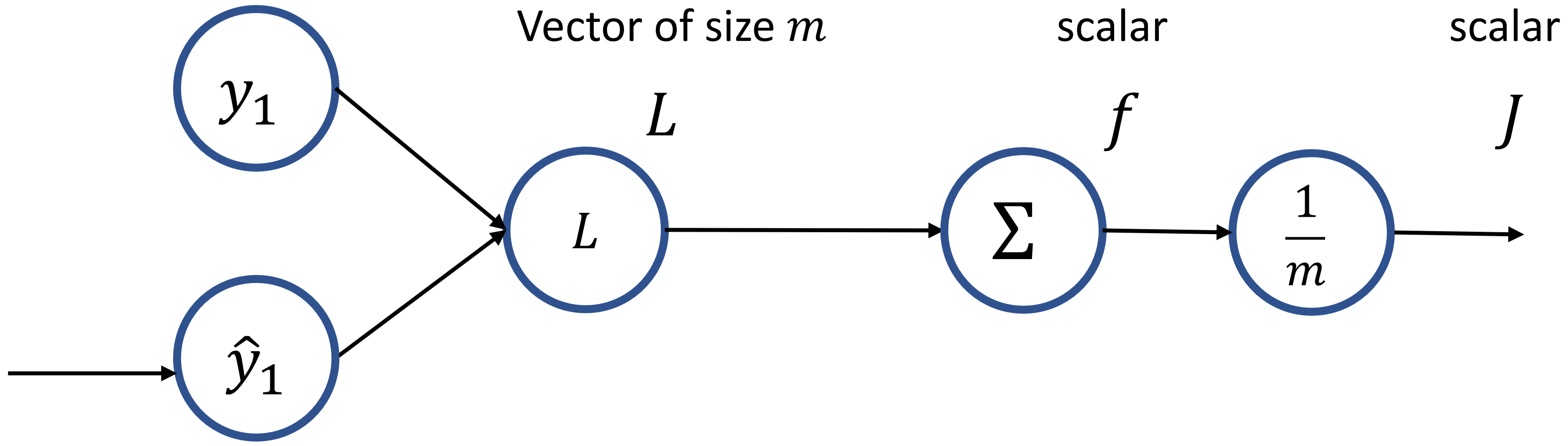


Brad Quinton, Scott Chin

Recall: Categorical Loss Cost Function

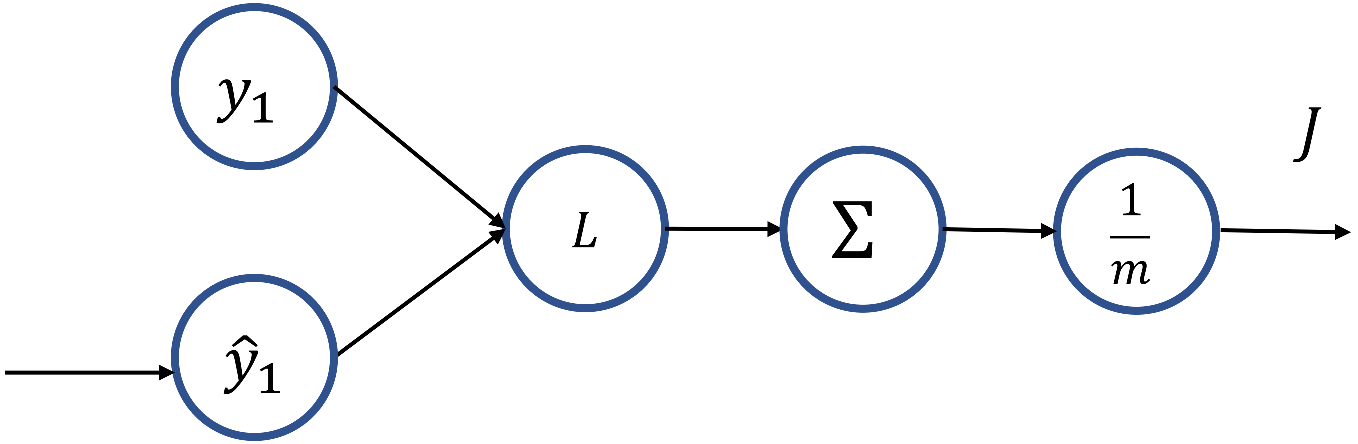
$$L(\hat{y}, y) = -\sum_{j=1}^{n_c} y_j log(\hat{y}_j)$$

$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y)$$



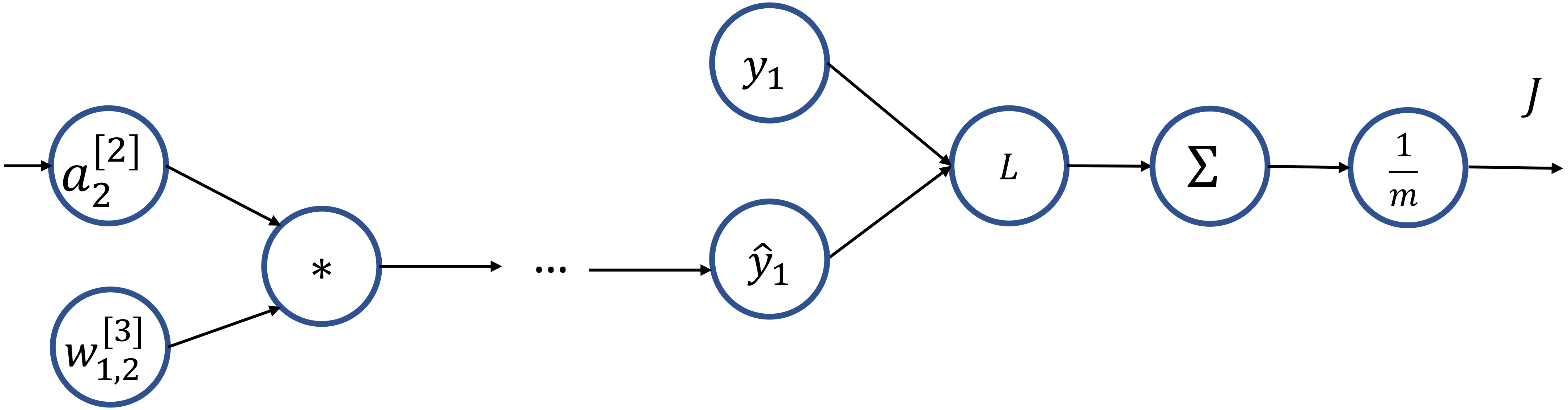
Brad Quinton, Scott Chin

Recall: Categorical Loss Cost Function



$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y)$$

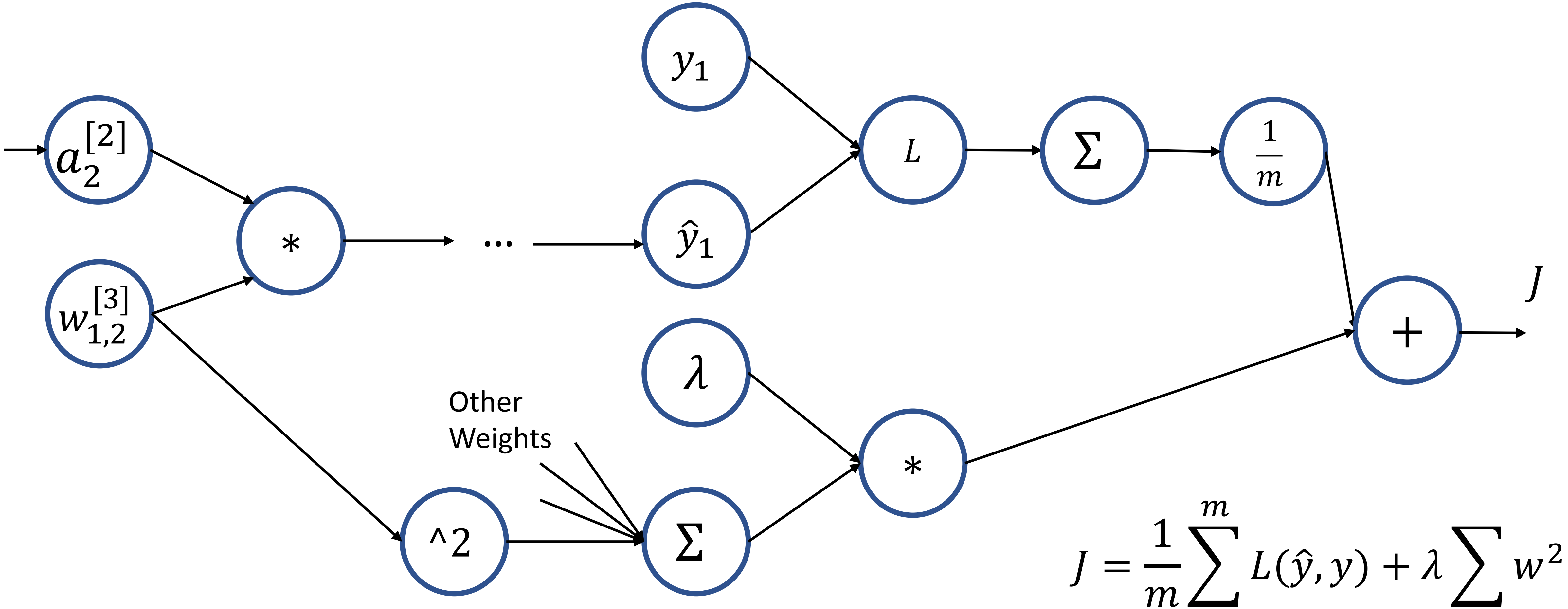
Let's look at Backprop for one weight



$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y)$$

Brad Quinton, Scott Chin

Let's look at Backprop for one weight



$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y) + \lambda \sum w^2$$

Brad Quinton, Scott Chin

Why would this reduce overfitting?

$$J = \frac{1}{m}\sum_{j=1}^{m} L(\hat{y}, y) + \lambda \sum w^2$$

- Discourages subset of weights dominating

Brad Quinton, Scott Chin

Discourages subset of weights dominating

- Consider one fully-connected unit

$$z = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Brad Quinton, Scott Chin

Discourages subset of weights dominating

- Consider one fully-connected unit

$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

$$x = [1, 1, 1, 1, 1, 1]$$

$$w = [1\ 0\ 0\ 0\ 0\ 0]$$

$$w = [0.2, 0.1, 0.1, 0.2, 0.3, 0.1]$$

Other Cost Function Regularizers

- L2 Regularization $R = \lambda \sum w^2$
- L1 Regularization $R = \lambda \sum |w|$
- L2 and L1 (Elastic Net) $R = \lambda_{L1} \sum |w| + \lambda_{L2} \sum w^2$

https://keras.io/regularizers/

Brad Quinton, Scott Chin

Regularizing Bias Parameters

- Not often done
- Doesn’t have a big impact since there are orders of magnitude more weight parameters vs bias parameters

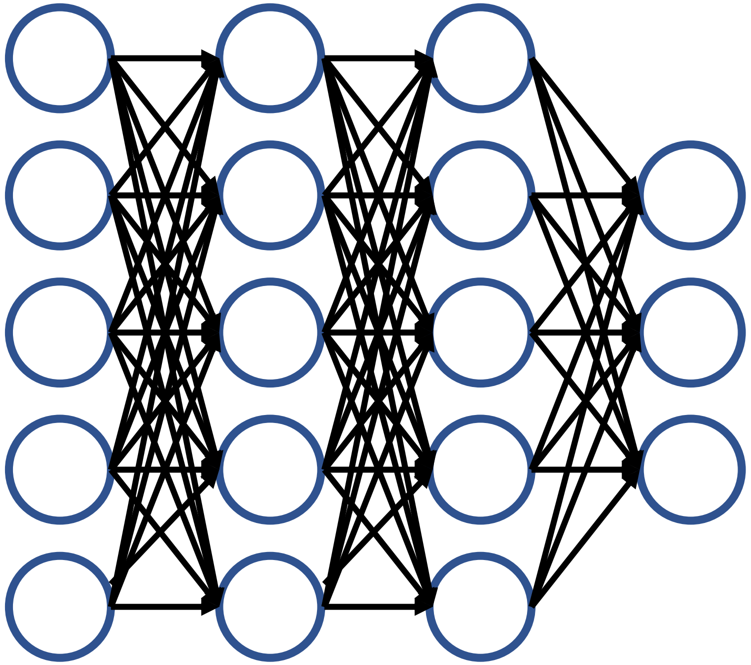
Dropout

” Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, 2014, http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

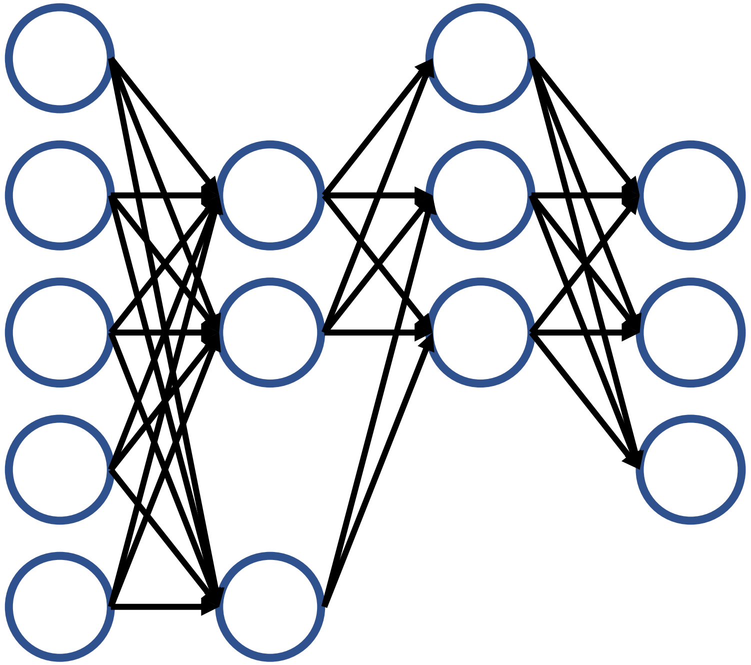
Dropout

- On each param update iteration, randomly remove some hidden units from the network

Actual Network

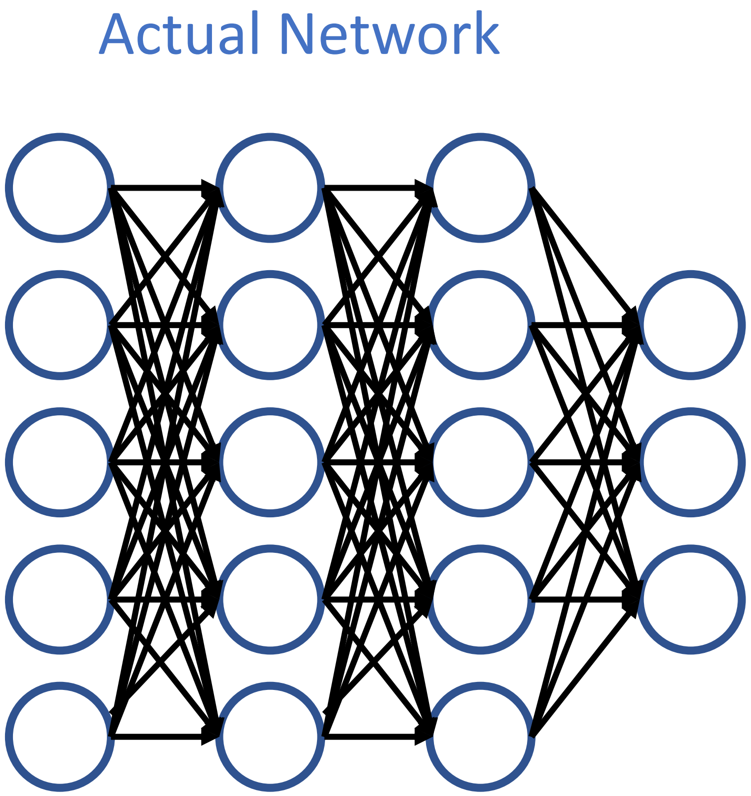


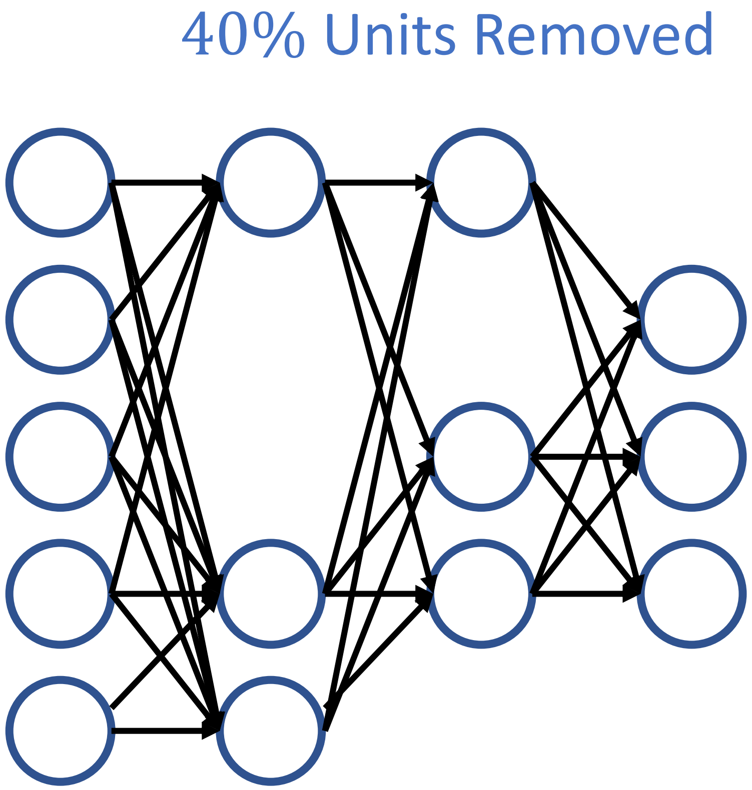
40% Units Removed



Dropout

- On each param update iteration, randomly remove some hidden units from the network

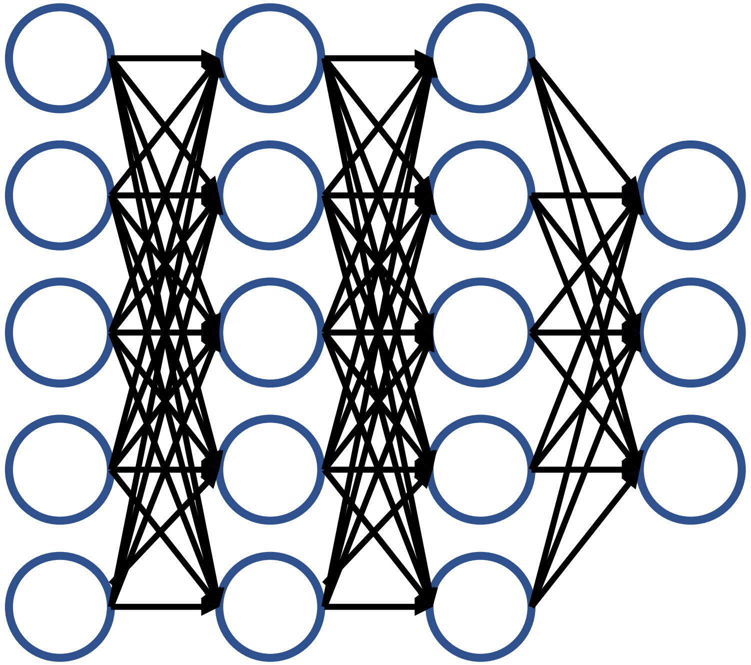




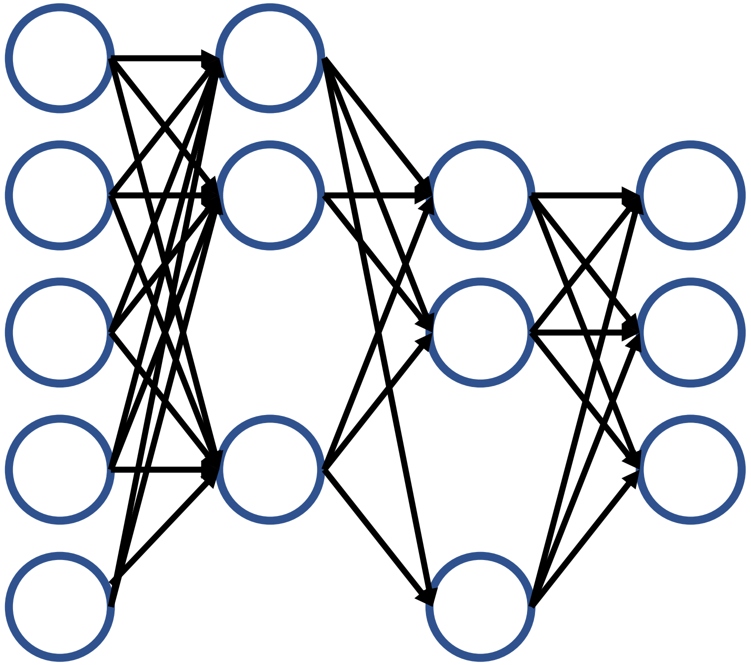
Dropout

- On each param update iteration, randomly remove some hidden units from the network

Actual Network



40% Units Removed



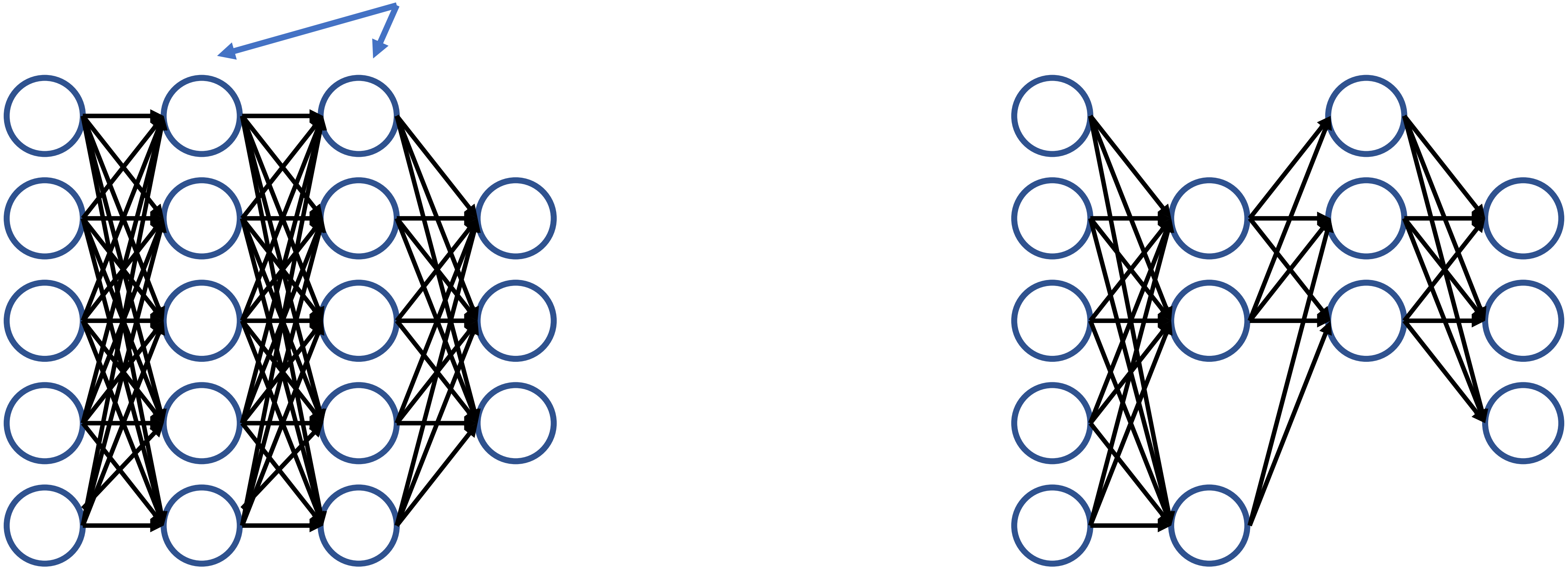
Brad Quinton, Scott Chin

Intuition

- Training a bunch of smaller simpler models and ensembling them together
  - Each model overfits in different ways so averages out
- Don't put too much weight into any particular feature
  - similar effect to L2 regularization
- Force each unit to learn to work well with a random subset of input units
  - Drive it to learn useful features on its own instead of relying on certain input units to correct its shortcomings

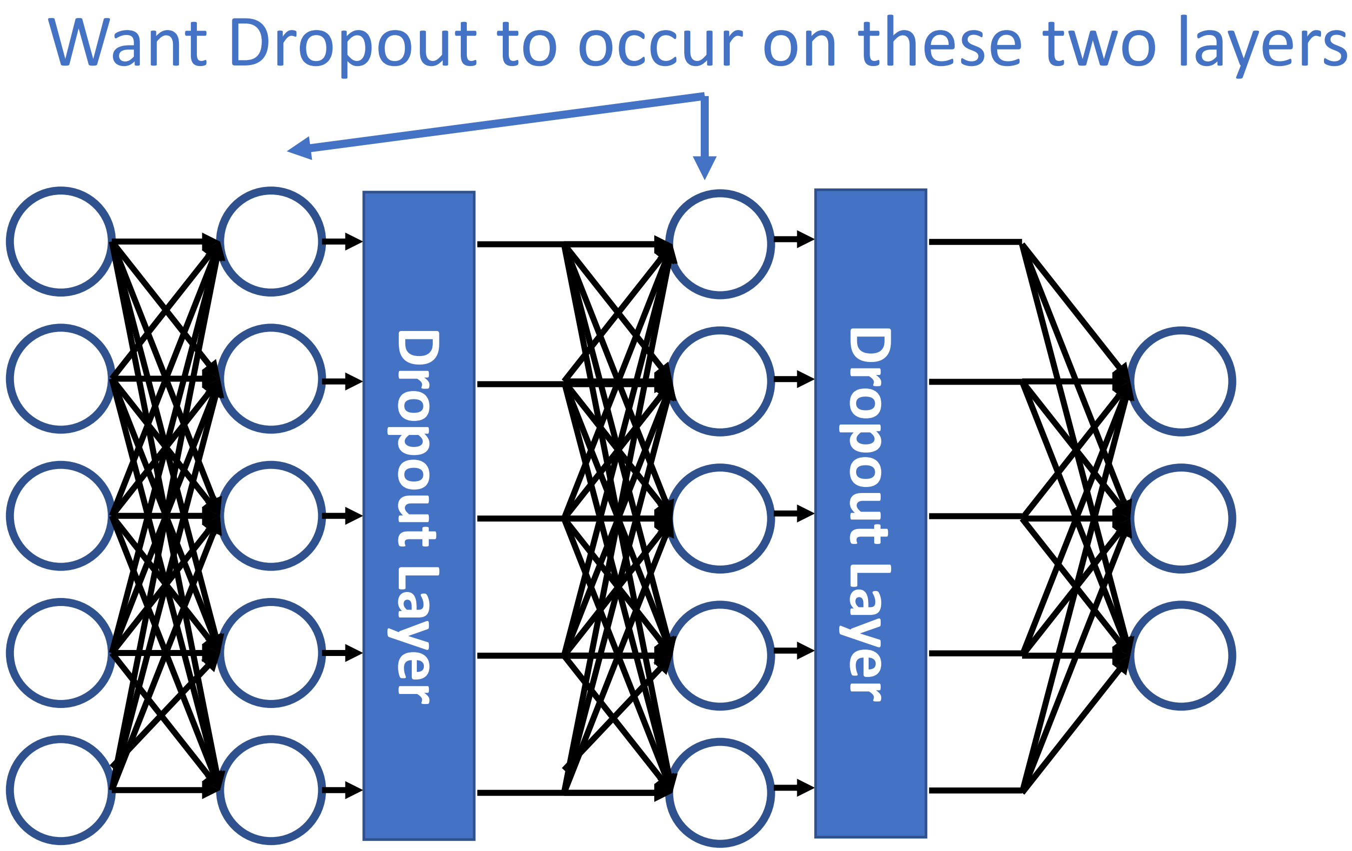
Dropout as a layer

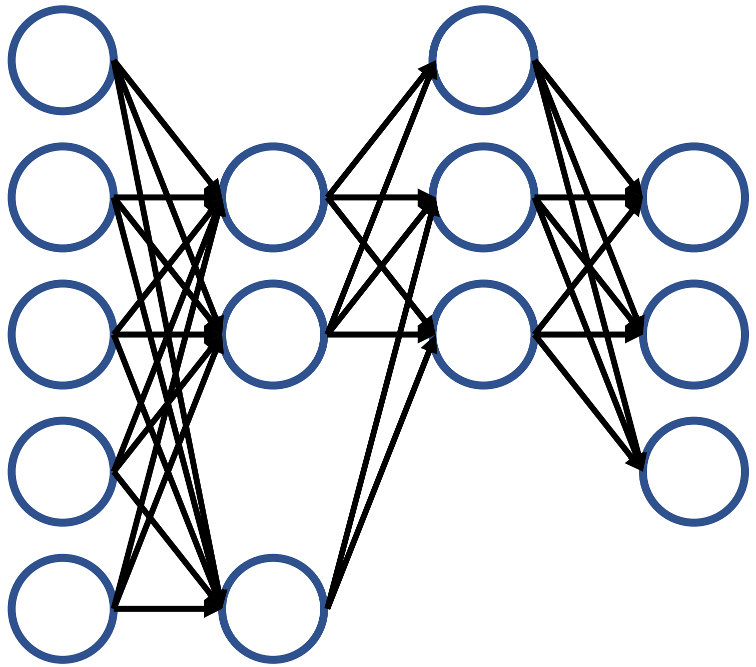
Want Dropout to occur on these two layers



Brad Quinton, Scott Chin

Dropout as a layer

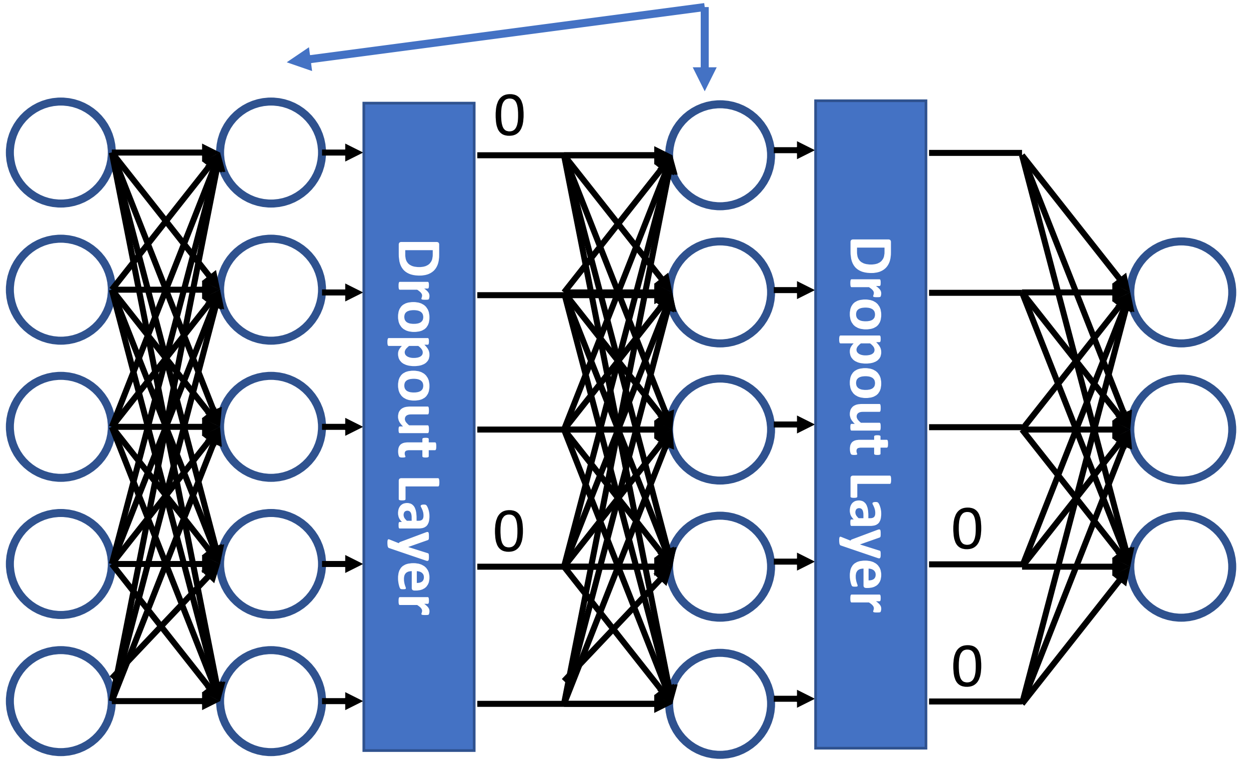


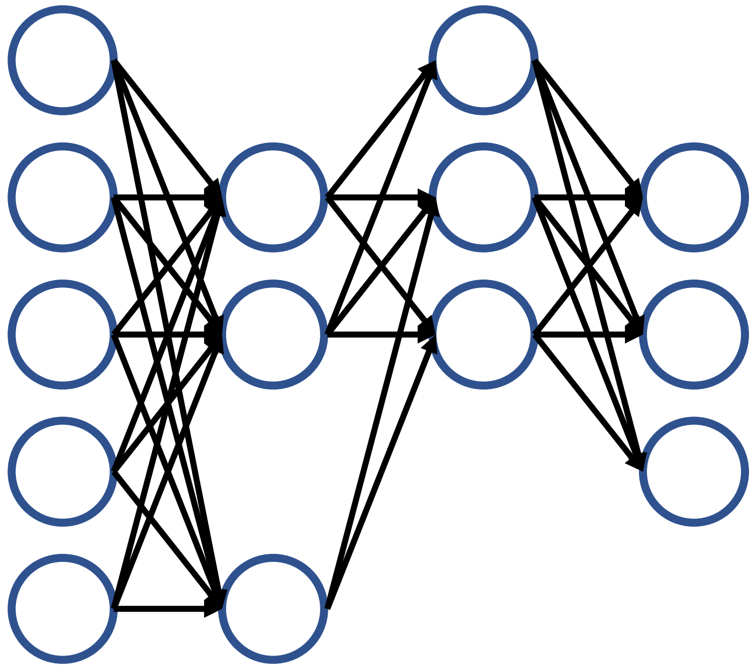


Brad Quinton, Scott Chin

Dropout as a layer

Want Dropout to occur on these two layers

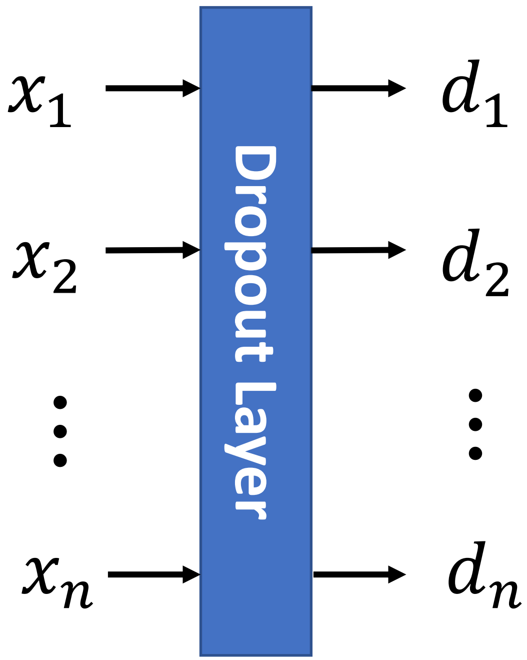




Can implement Dropout by outputting 0
at appropriate locations

Brad Quinton, Scott Chin

Inside a Dropout Layer

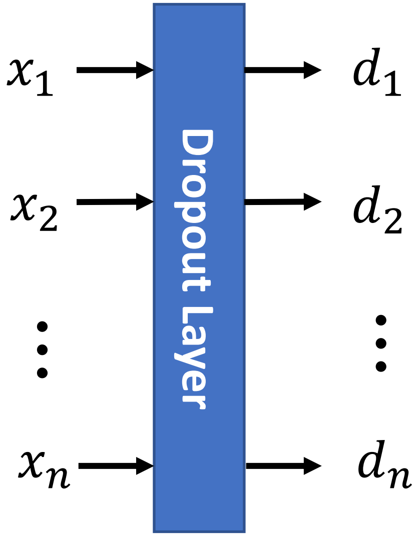



```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

- A new random mask generated on each forward pass
- $keep\_prob$ is probability of NOT dropping a node
- d is output with some nodes changed to output 0

Brad Quinton, Scott Chin

Dropout Example



Note: This is a uniform distribution between [0 1]

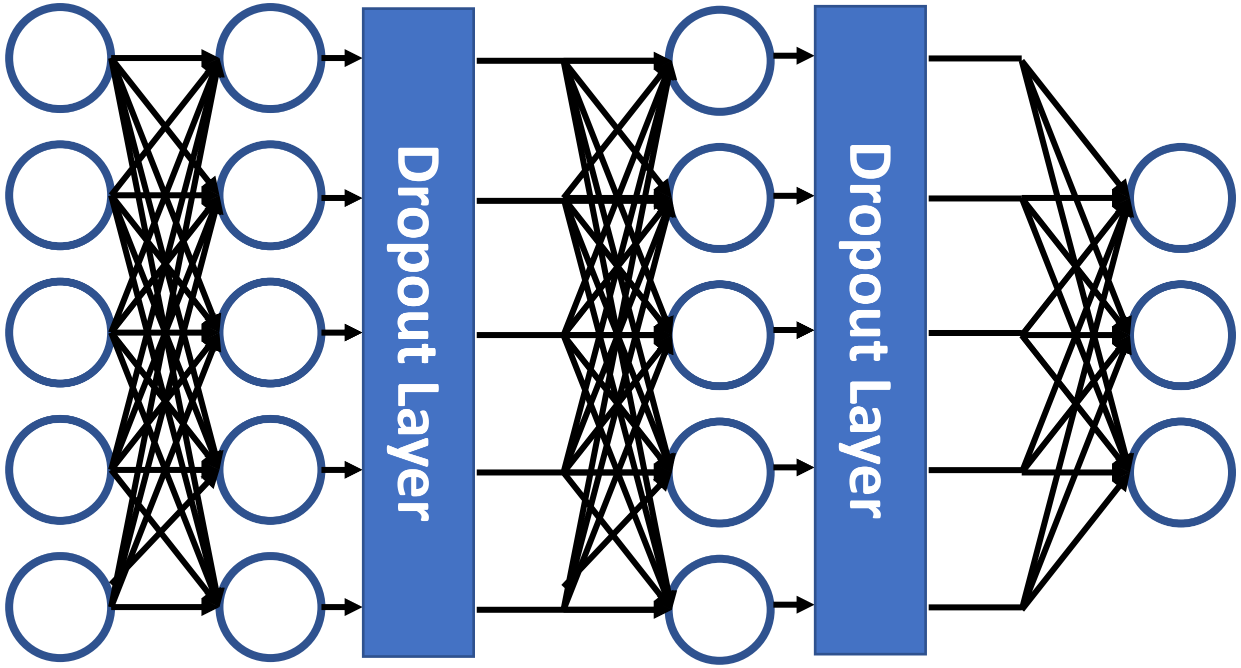
```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

Example with `n=10 keep_prob=0.6`

`np.random.rand(n)`	`mask`
[0.31 0.97 0.24 0.93 0.74 0.85 0.60 0.48 0.46 0.21]	[1 0 1 0 0 0 1 1 1 1]
[0.90 0.25 0.41 0.95 0.35 0.01 0.87 0.67 0.77 0.04]	[0 1 1 0 1 1 0 0 0 1]
[0.08 0.08 0.24 0.81 0.99 0.04 0.83 0.19 0.63 0.39]	[1 1 1 0 0 1 0 1 0 1]
[0.56 0.08 0.14 0.45 0.32 0.10 0.89 0.62 0.35 0.94]	[1 1 1 1 1 1 0 0 1 0]

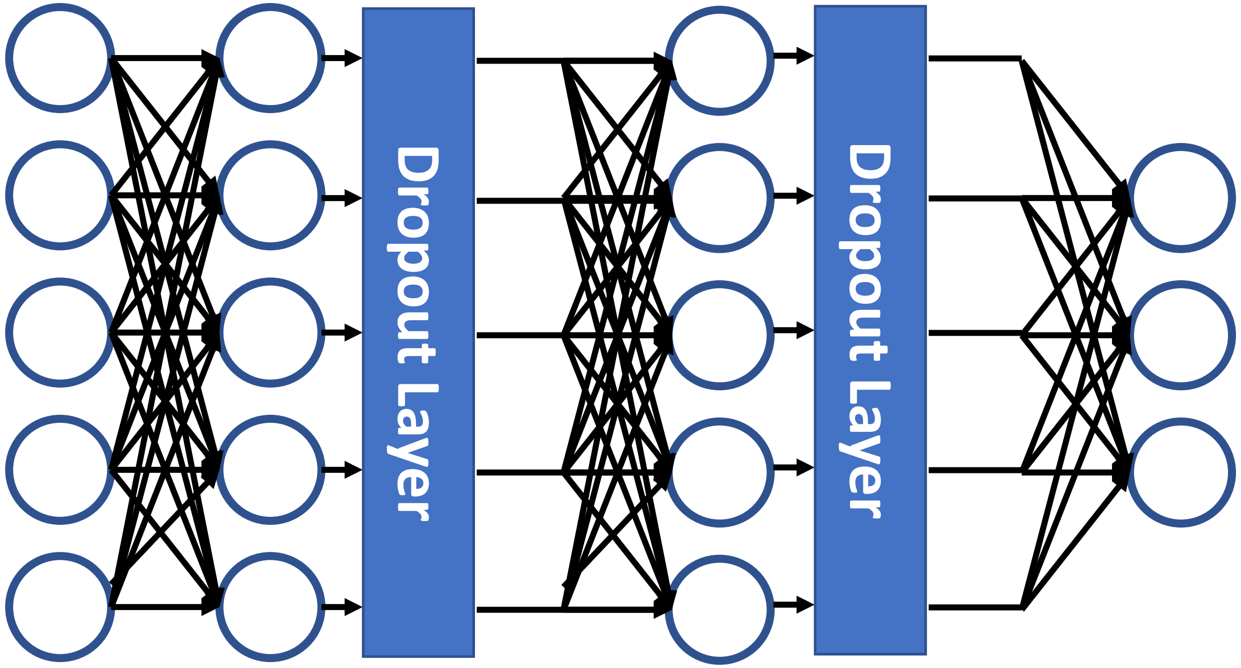
Not always exactly 60%, but on average will be

What Happens At Prediction Time?



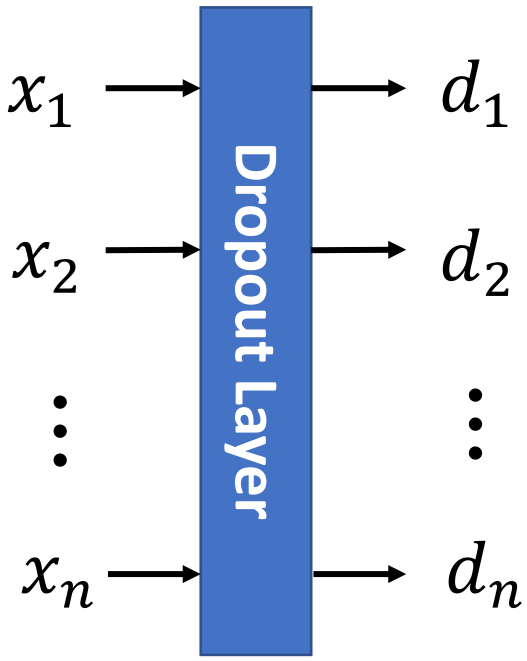
Brad Quinton, Scott Chin

What Happens At Prediction Time?



- Non deterministic predictions!
- Not desirable at all so how to address this?
- Let's look at one layer again

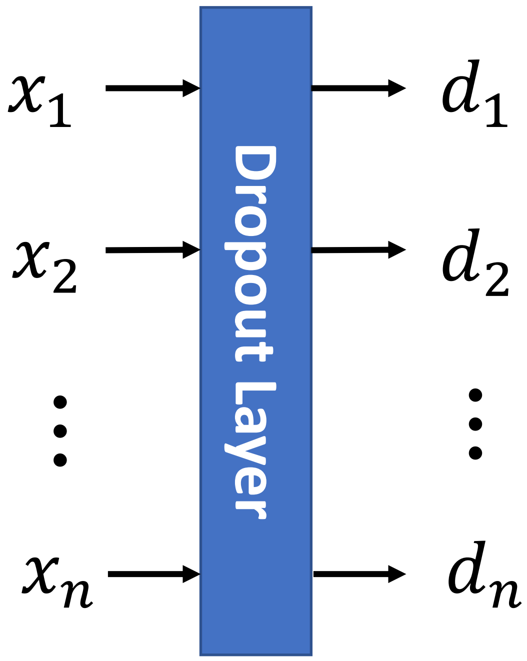
During Prediction Time




```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

- d is a function of x and the mask. i.e. $d(x, mask)$

During Prediction Time

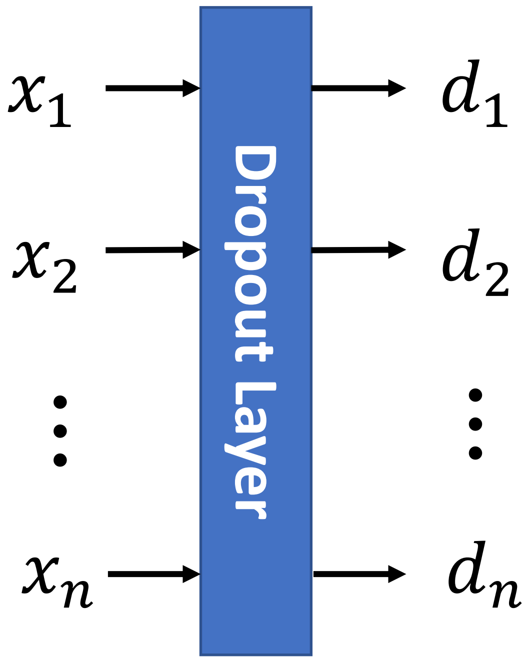



```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

- d is a function of x and the mask. i.e. $d(x, mask)$
- There are 2^n unique masks

Brad Quinton, Scott Chin

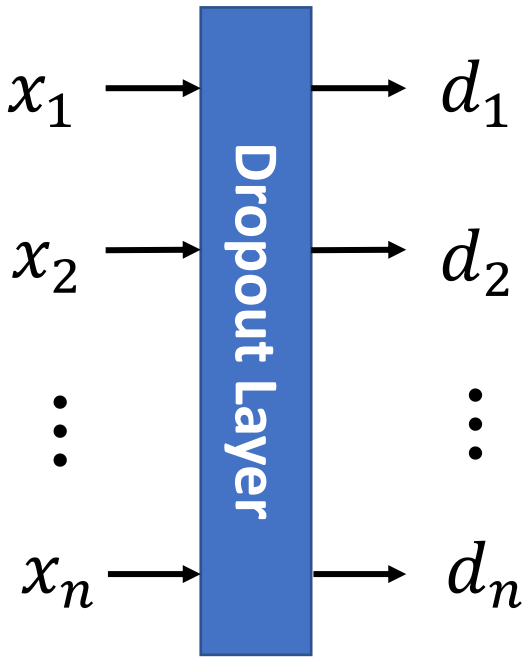
During Prediction Time




```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

- d is a function of x and the mask. i.e. $d(x, mask)$
- There are 2^n unique masks
- Let $d_i(x, mask_i)$ denote output for one mask

During Prediction Time



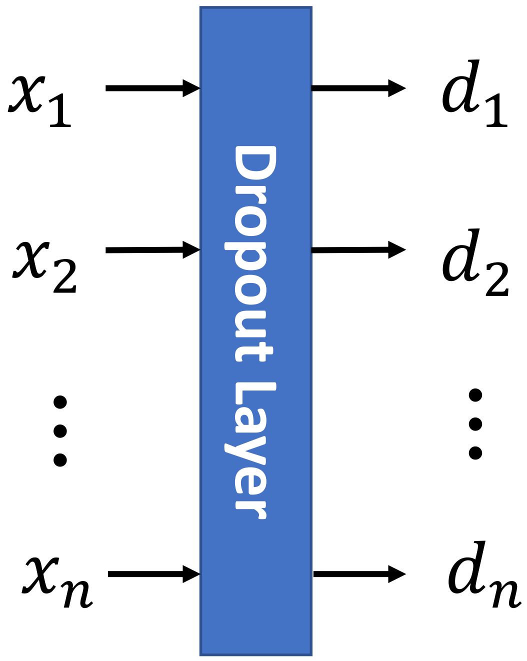

```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

- d is a function of x and the mask. i.e. $d(x, mask)$
- There are 2^n unique masks
- Let $d_i(x, mask_i)$ denote output for one mask
- Each mask can occur with a probability $p(mask_i)$

Brad Quinton, Scott Chin

During Prediction Time

```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

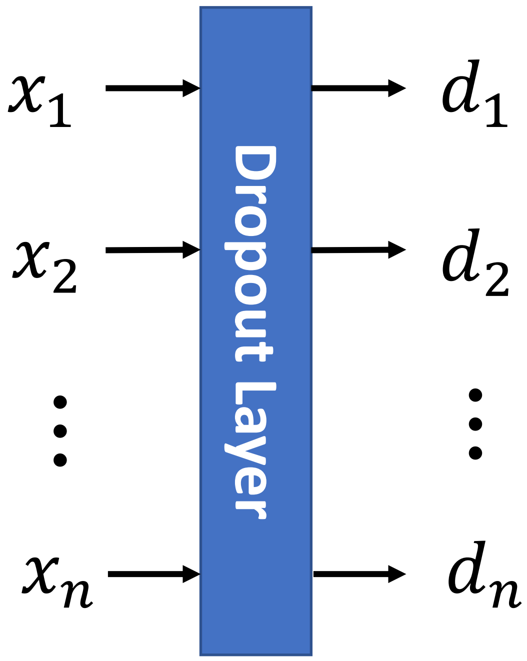


- d is a function of x and the mask. i.e. $d(x, mask)$
- There are 2^n unique masks
- Let $d_i(x, mask_i)$ denote output for one mask
- Each mask can occur with a probability $p(mask_i)$
- Therefore, expected output value at prediction is:

$$E[d] = \sum_{i}^{2^n} p(mask_i) d_i(x, mask_i)$$

Brad Quinton, Scott Chin

During Prediction Time



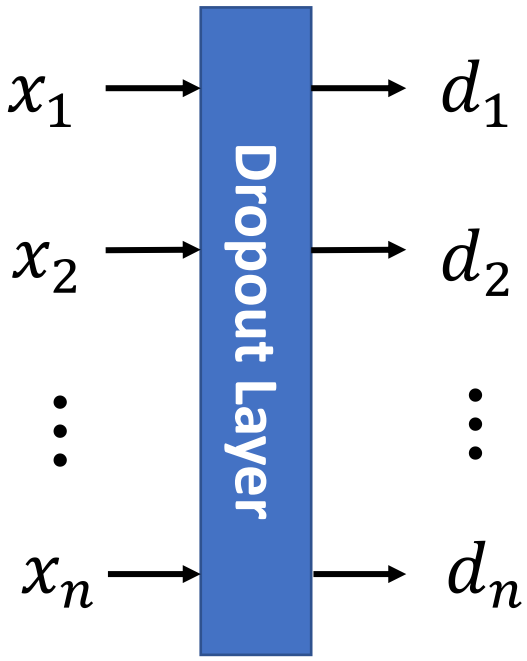

```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

- d is a function of x and the mask. i.e. $d(x, mask)$
- There are 2^n unique masks
- Let $d_i(x, mask_i)$ denote output for one mask
- Each mask can occur with a probability $p(mask_i)$
- Therefore, expected output value at prediction is:

$$E[d] = \sum_{i}^{2^n} p(mask_i) d_i(x, mask_i)$$

Not feasible to compute for any moderate sized layer

During Prediction Time



During Training:

```
mask = np.random.rand(n) < keep_prob
d = x * mask
```

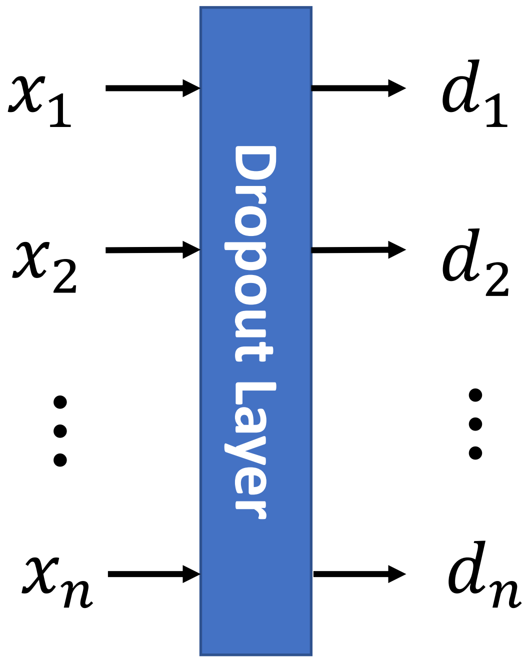
During Prediction:

```
d = x * keep_prob
```

- A good approximation is simply scaling the inputs with $keep\_prob$
- See paper for derivation

Brad Quinton, Scott Chin

Inverted Dropout



During Training:

```
mask = np.random.rand(n) < keep_prob
d = (x * mask)/keep_prob
```

During Prediction:

```
d = x
```

- Instead, can do the scaling during training
- During prediction time, dropout layer simply passes all inputs

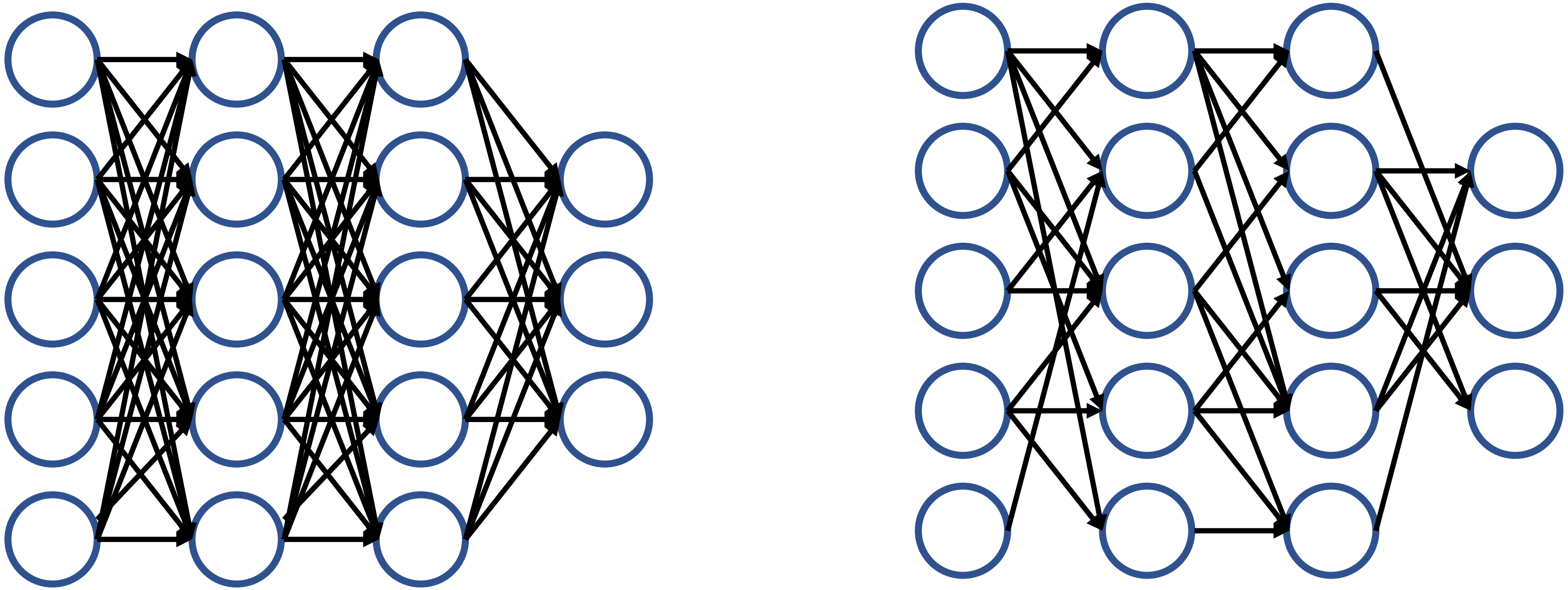
Where to Put Dropout Layers

- Use mainly with fully-connected layers → they are prone to overfitting compared to conv layers
- Don't really see them used with conv layers
- Conv layers aren't as prone to overfitting because each swatch (convolutional location on input volume) is basically a separate piece of training data
- Looking through mainstream CNN architectures, the ones that used fully-connected layers (AlexNet, VGGNet) would use DropOut on the fully-connected layers

Brad Quinton, Scott Chin

DropConnect

- Similar to DropOut except 0 out random weights at training (i.e. connections) instead of nodes



“Regularization of Neural Networks using DropConnect”, Wan et al., 2013, http://proceedings.mlr.press/v28/wan13.pdf

Brad Quinton, Scott Chin

BatchNorm Regularization Effect

- Mean and variance on mini-batch is only an approximation to the actual mean and variance compared to the entire training set activations
- This introduces randomness
- Unintended regularization effect

Data Augmentation

Generating New Training Data from Existing Training Data

Original

Mirror

Rotate

Blur

Saturation

Can View Data Augmentation as Regularization

Original

Mirror

Rotate

Blur

Saturation

Don’t overfit to this

These are also ”views” of the same thing!

Take different crops to generate new images

Brad Quinton, Scott Chin

Various Other Generic Transforms

- Rotate
- Zoom
- Mirror
- Crops
- Shear
- Brightness

- See https://keras.io/preprocessing/image/

What kind of Regularization Should I use?

- Common to use L2
- Consider Dropout for large fully connected layers
- Don’t rely on BatchNorm for regularization, but it is a bonus
- For imaged data use Data Augmentation

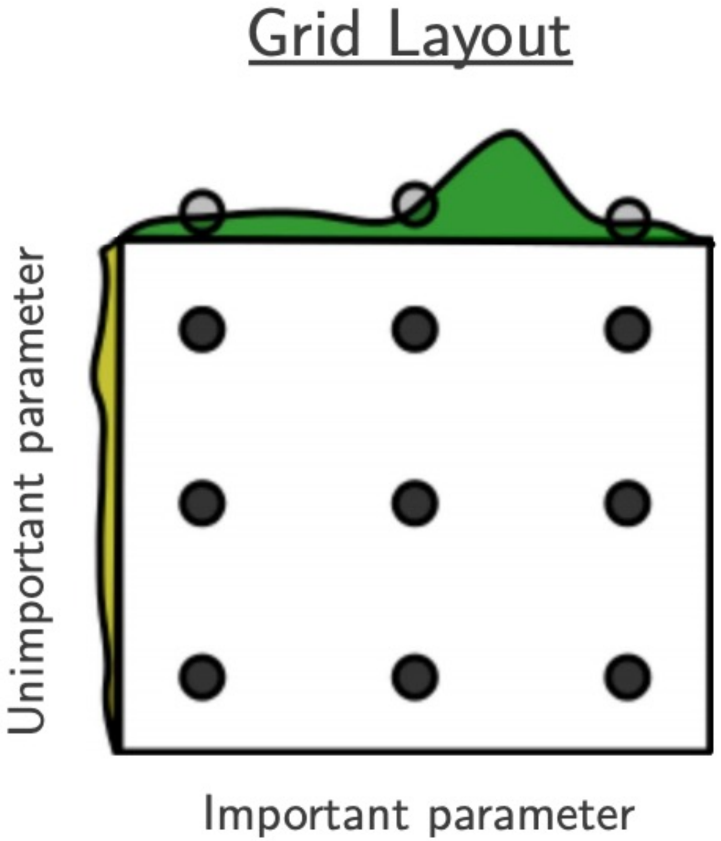
Hyperparameter Tuning Strategies

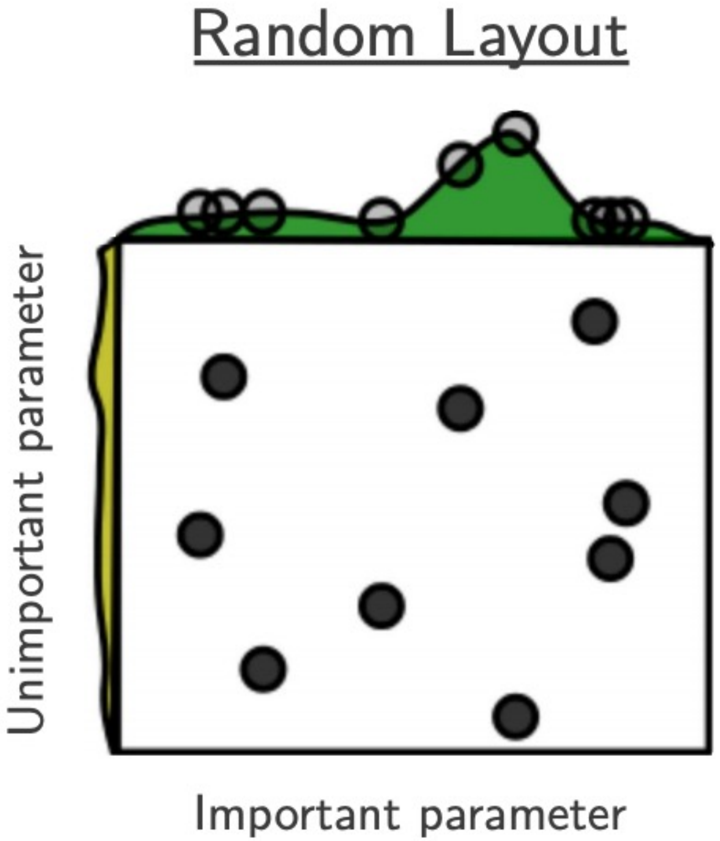
Lots of Hyperparameters You Can Tune

Hyperparameter is any choice that affects your model architecture or optimization process

- Architecture hyperparameters
  - Number of layers
  - Number of units/filters per layer
  - Etc
- Optimization hyperparameters
  - Learning rate
  - Weight initialization
  - Optimizer hyperparameters (e.g. momentum beta)
  - Regularization techniques

Grid Search vs. Random Search



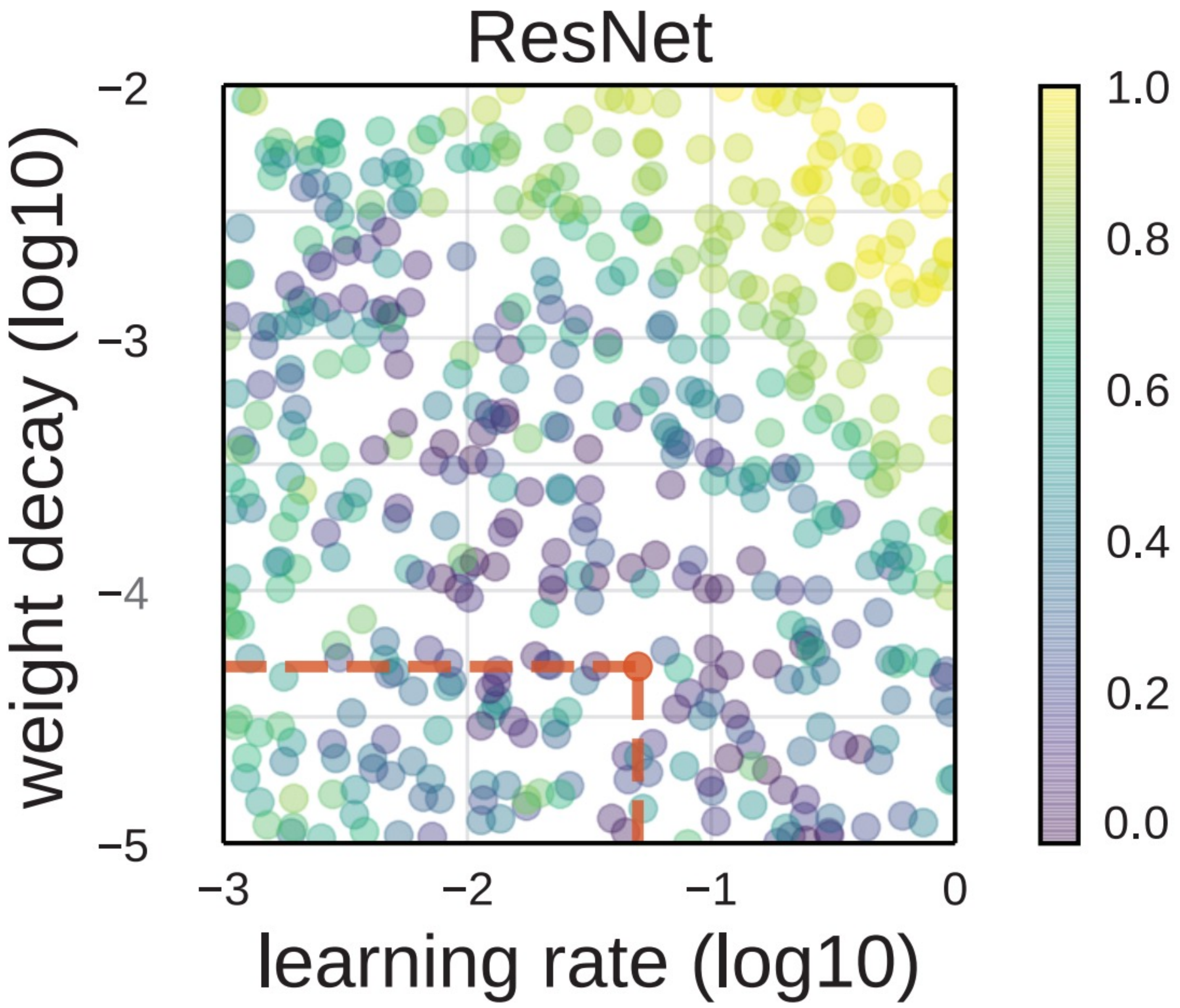


“Random Search for Hyper-Parameter Optimization”, Bergstra and Bengio, 2012, http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Log Scale vs Linear Scale

- For some hyperparameters, you want to search over a log scale
- For example learning rate, want to search in range of 0.0001 to 1

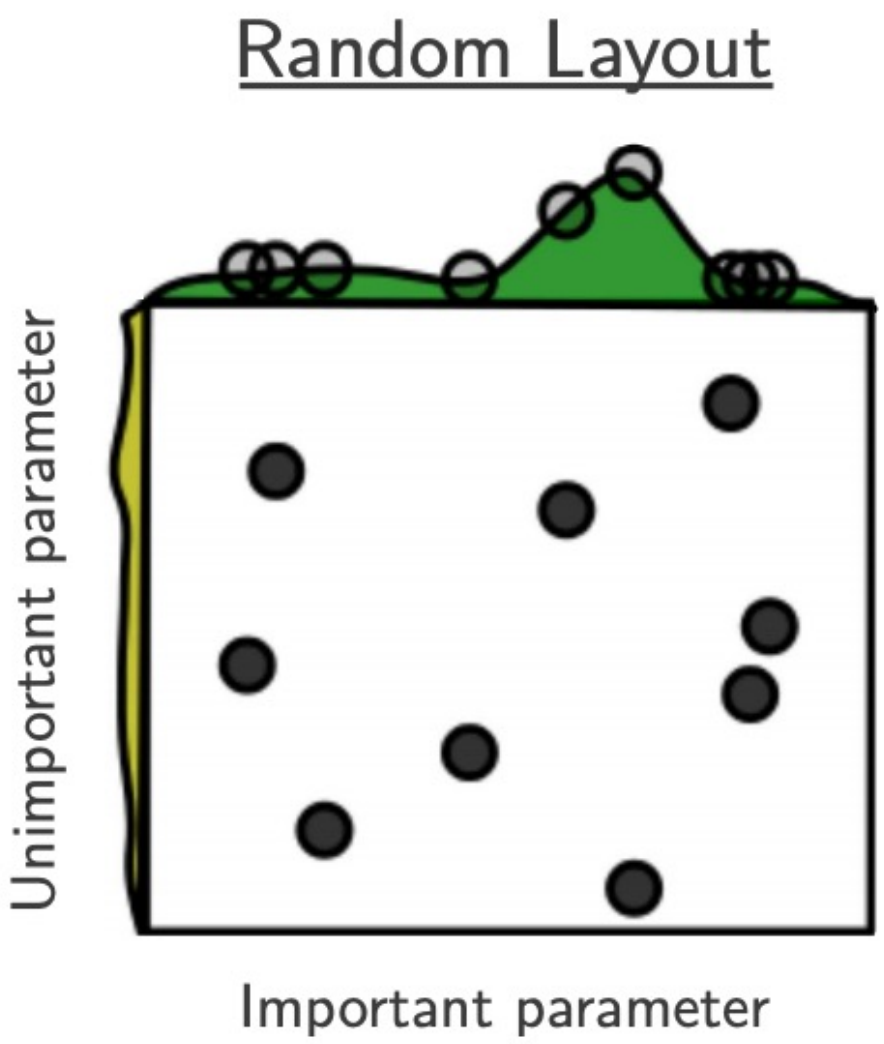
Brad Quinton, Scott Chin



 “On Network Design Spaces for Visual Recognition”, Radosavovic et al, 2019, “https://arxiv.org/pdf/1905.13214.pdf”

Coarse to Fine

- Do hyperparameter search in your initial range of hyperparameter values
- Find the values that minimize your cost the best.
- Zoom into a tighter region of values around this set of hyperparameter values and repeat search



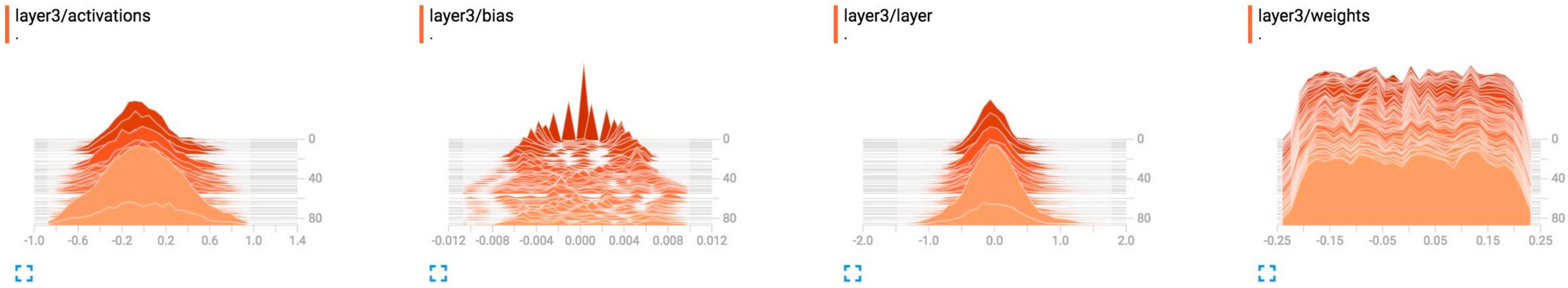
“Random Search for Hyper-Parameter Optimization”, Bergstra and Bengio, 2012, http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

General Advice on Training

1. Start by using only a very small subset of your training set and get your model to 100% accuracy.
   - Turn off regularization in this step
   - Lets you quickly flush out bugs in your optimization flow, and glaring deficiencies in your model architecture
   - If you can't achieve this, definitely can't achieve it on your full training set
2. Using your full training set, quickly find a learning rate that shows good decrease in cost.
   - Turn on regularization here
   - Can see effect of learning rate in small number of training iterations
3. Now do your hyperparameter search as discussed before

General Advice on Training

- Monitor histograms of gradients, parameters, activations during training
  - Can use tools like TensorBoard



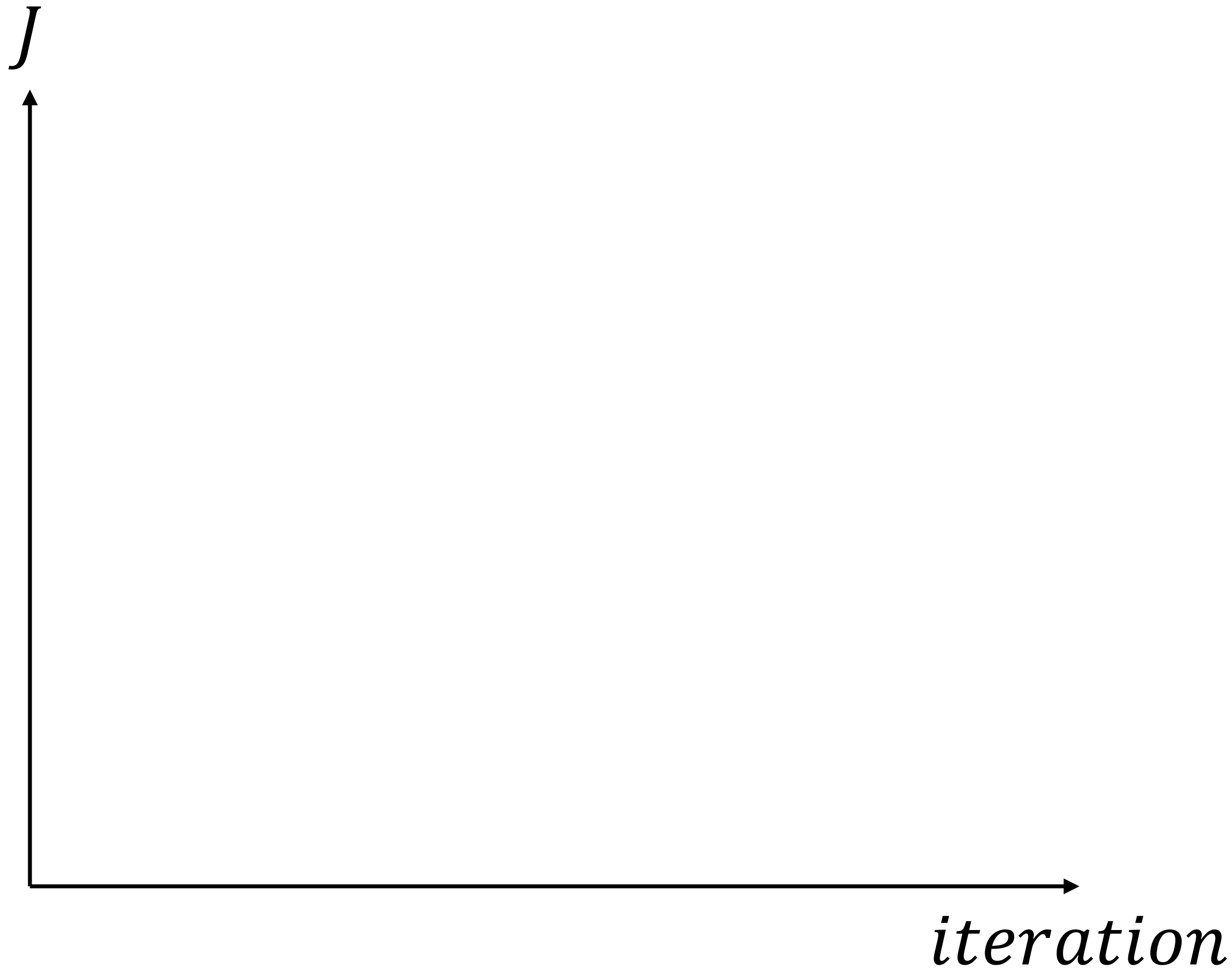
General Advice on Training

- Get your training accuracy high first
  - If you can’t get this to an acceptably high level first, either the network doesn’t have the capacity to learn the underlying mapping function, or your training setup has issues
  - Validation accuracy definitely won’t do better
- Then work on closing the gap and improving your validation accuracy
- Look at the failing cases
  - Think of ways to visualize your data
  - Look for patterns in your failing cases
- Look at your cost curves

Looking at Cost Curves

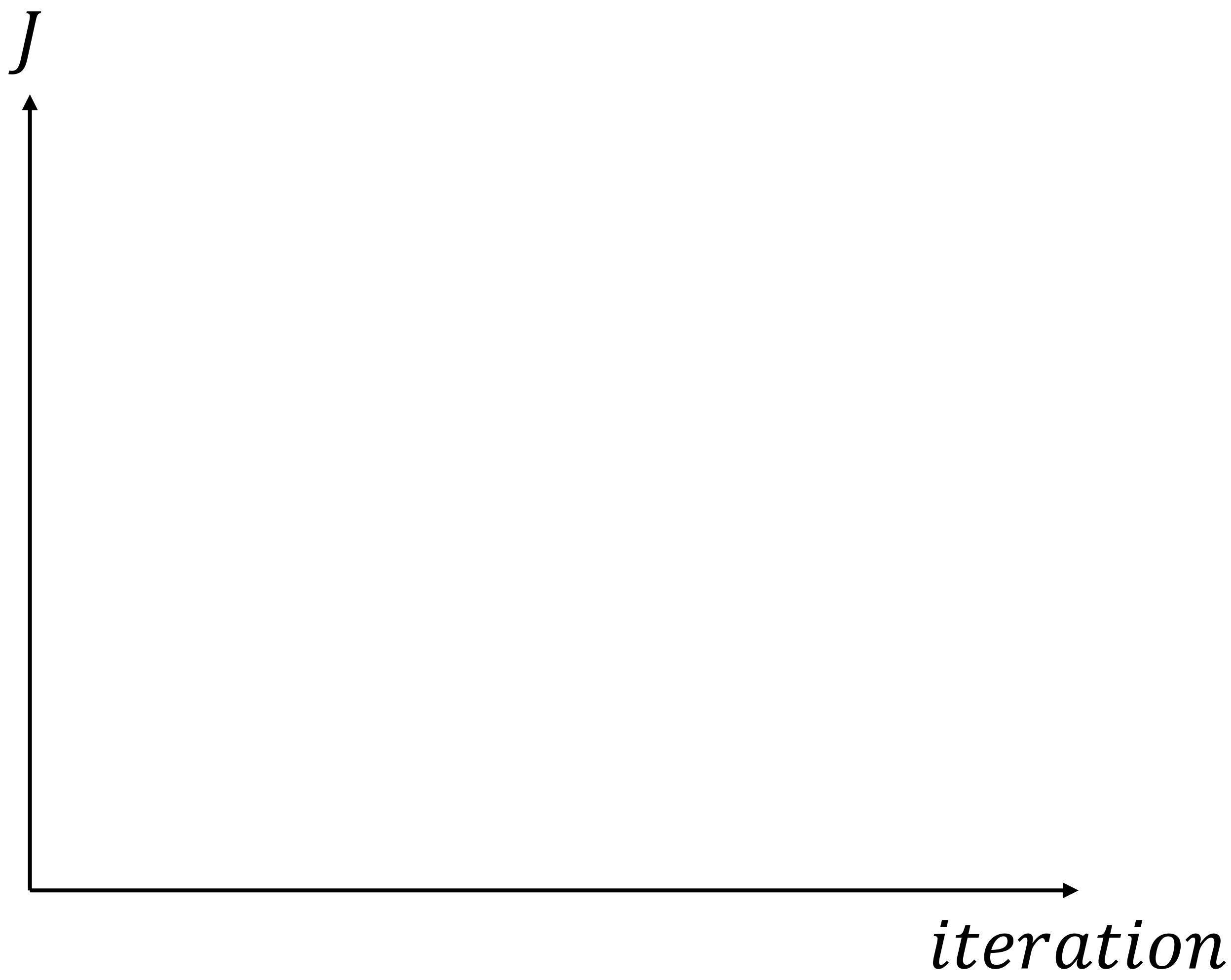
Brad Quinton, Scott Chin

Learning Rate Too Big

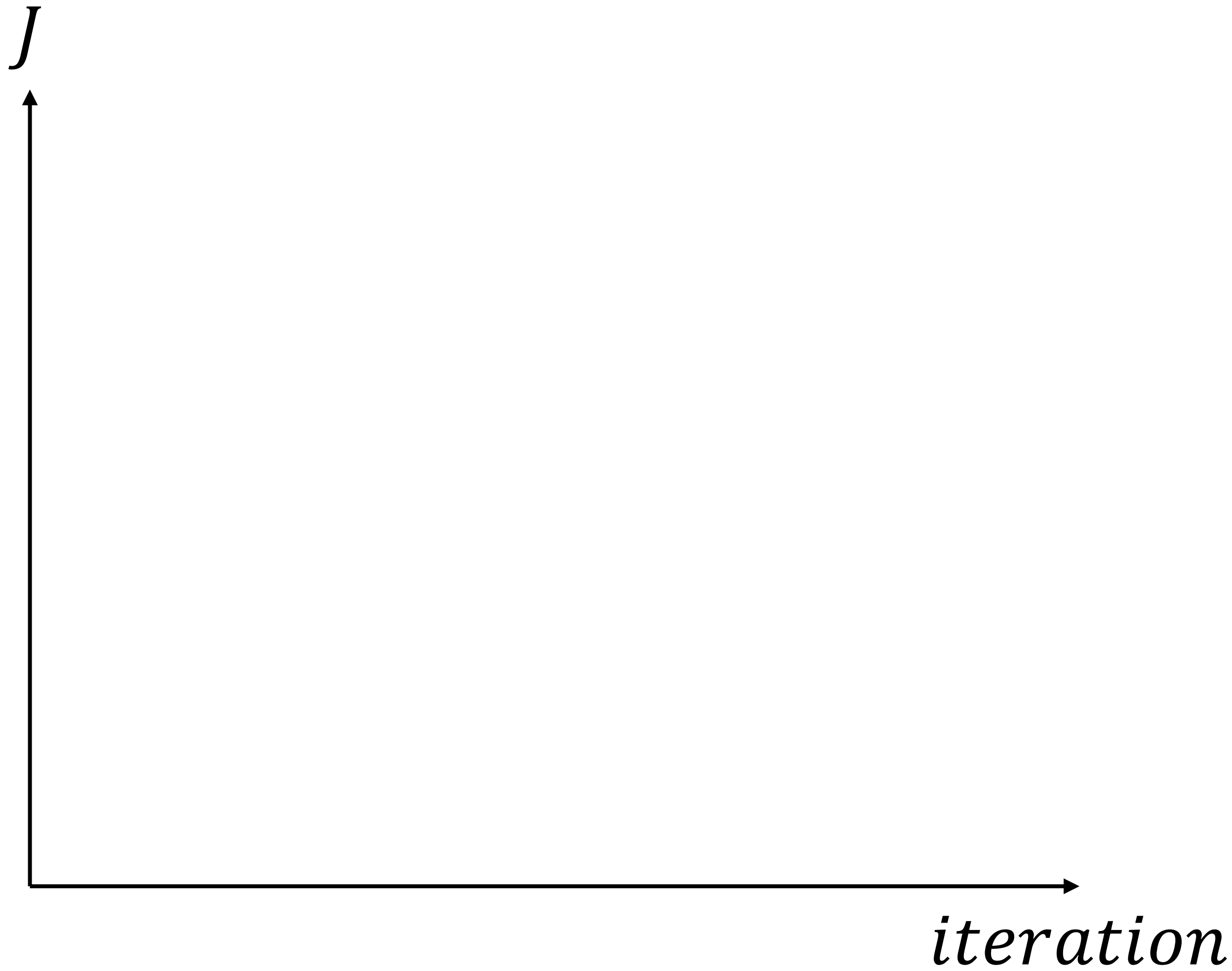


Brad Quinton, Scott Chin

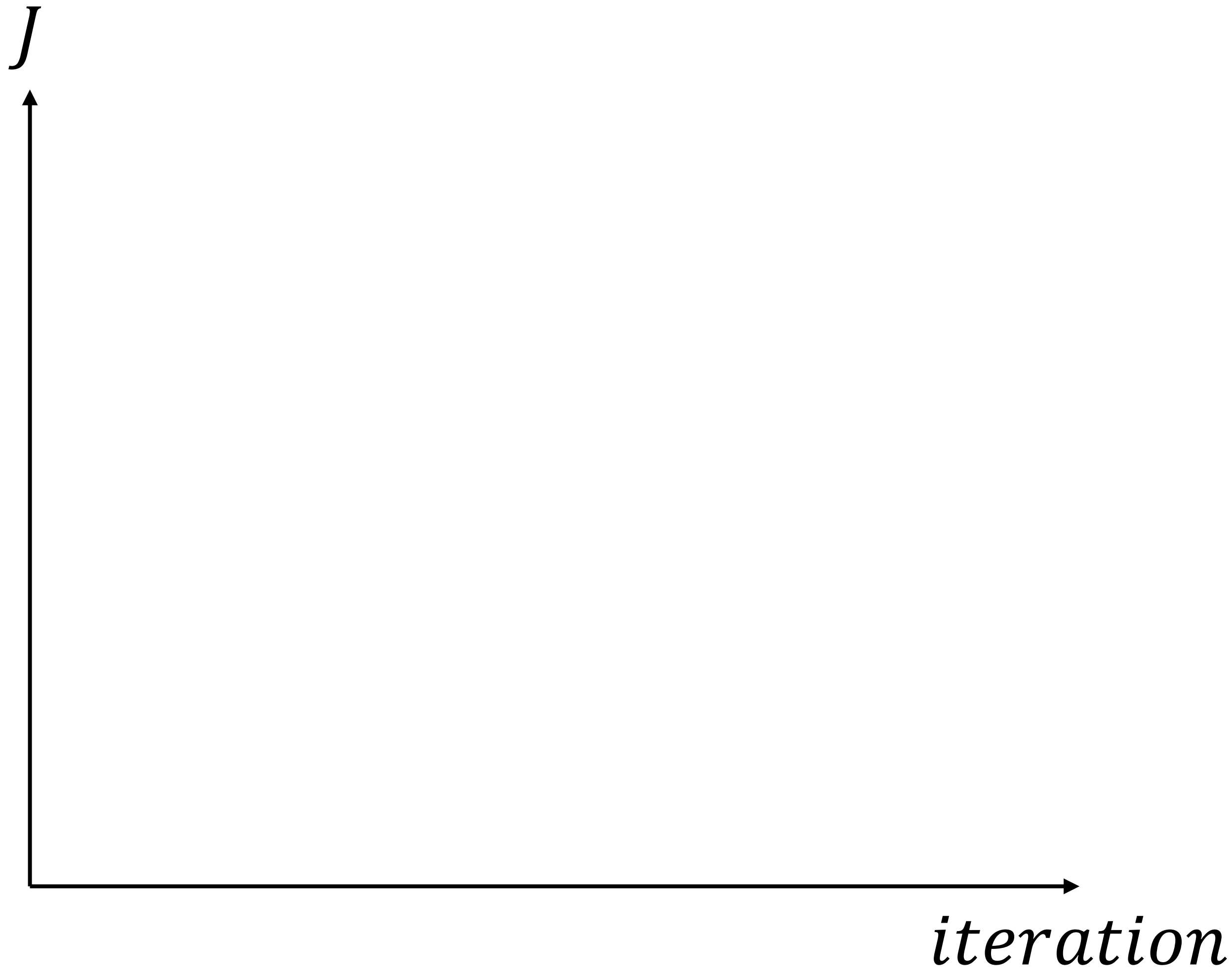
Slow Start - Bad Initialization



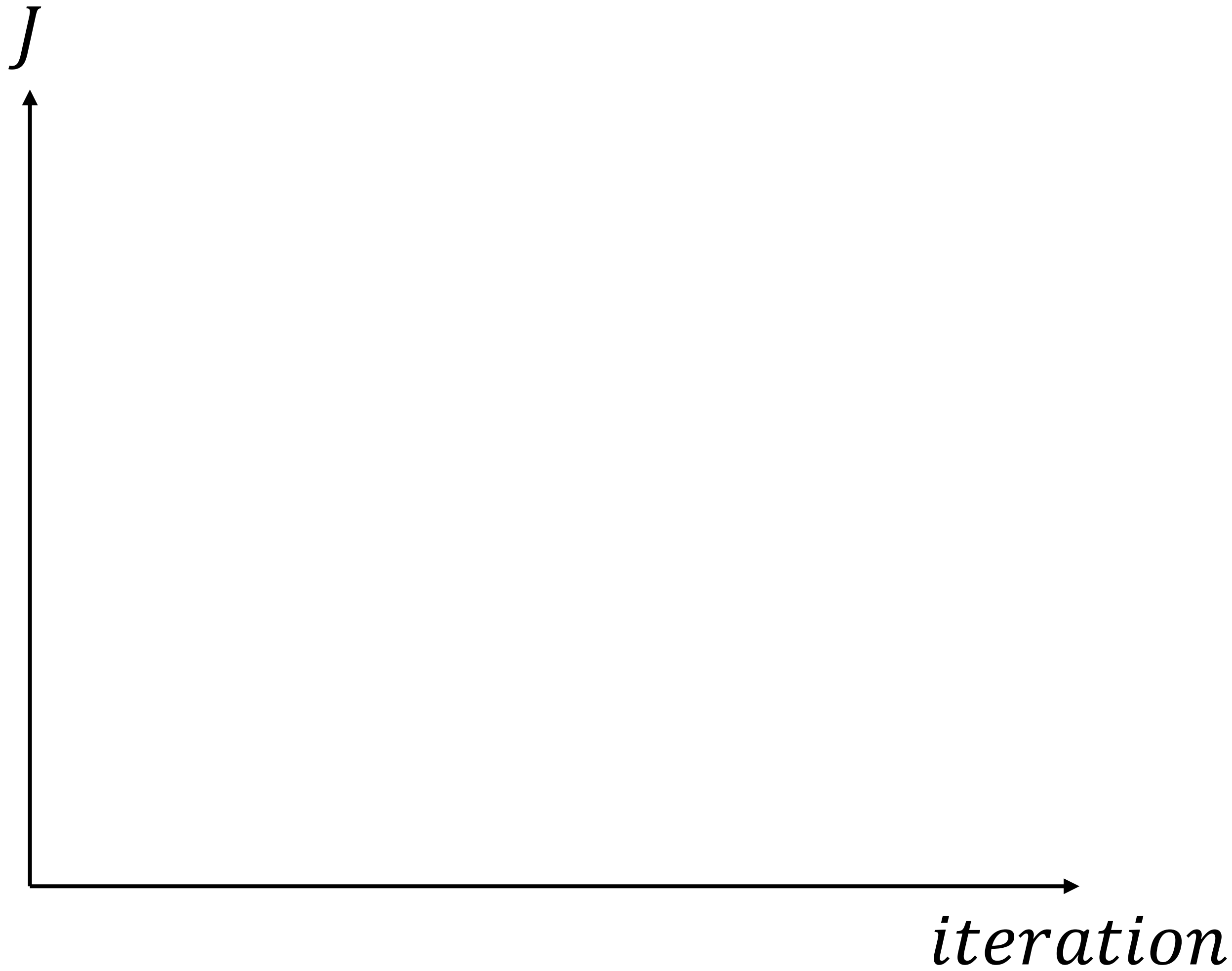
Loss Plateaus



Decayed Learning Rate Too Soon

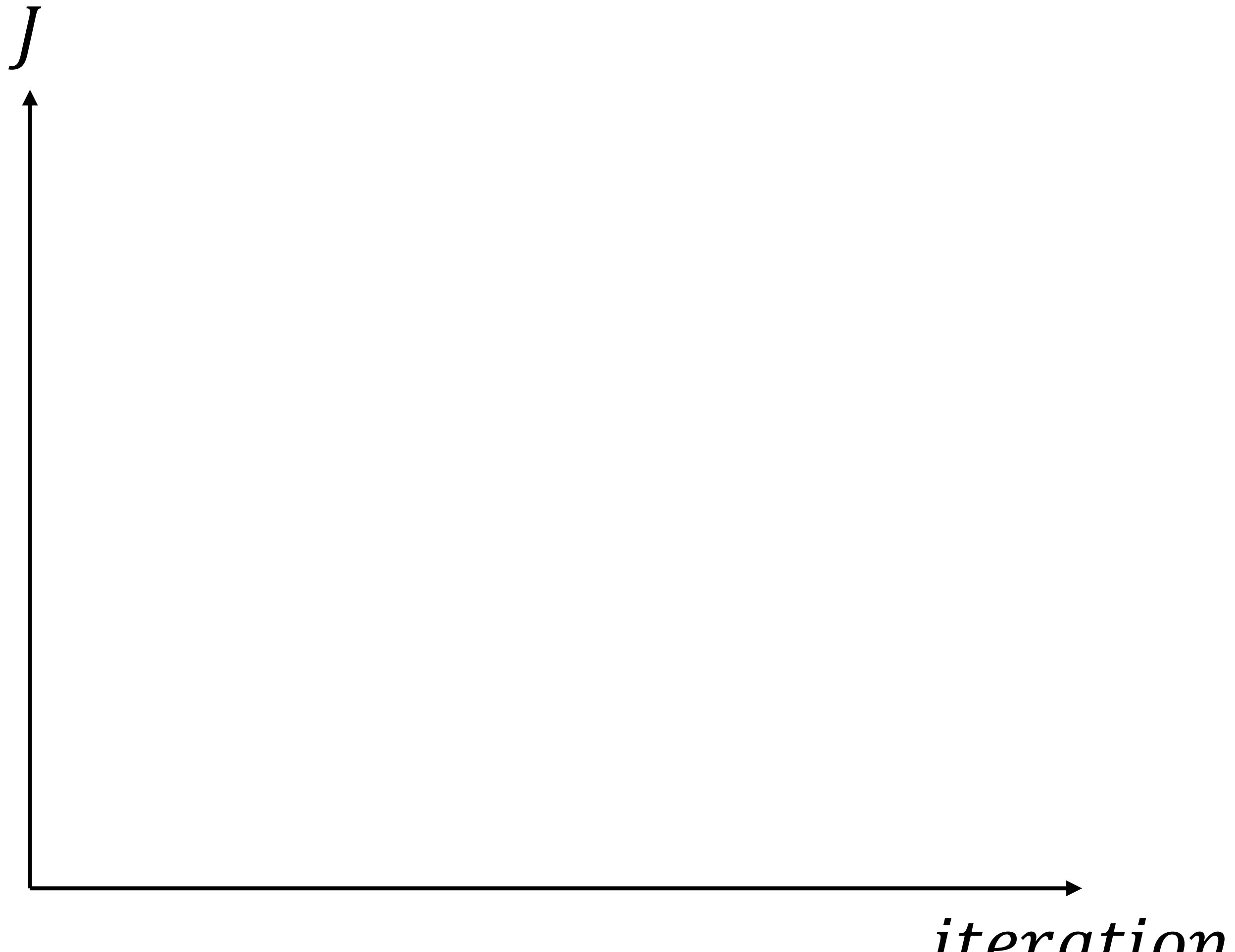


Overfitting



Overfitting

Potential Underfitting



Transfer Learning

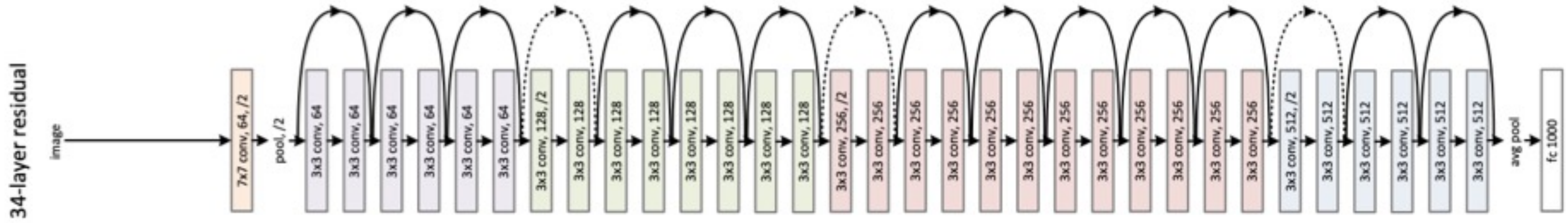
Transfer Learning

- Take a model that was trained for one task, and repurpose it for a second similar task
- When repurposing, keep some of the learnings (i.e. parameter values) from the first task

- Used a lot for image data
- Also used for text and speech

Transfer Learning with Image Data

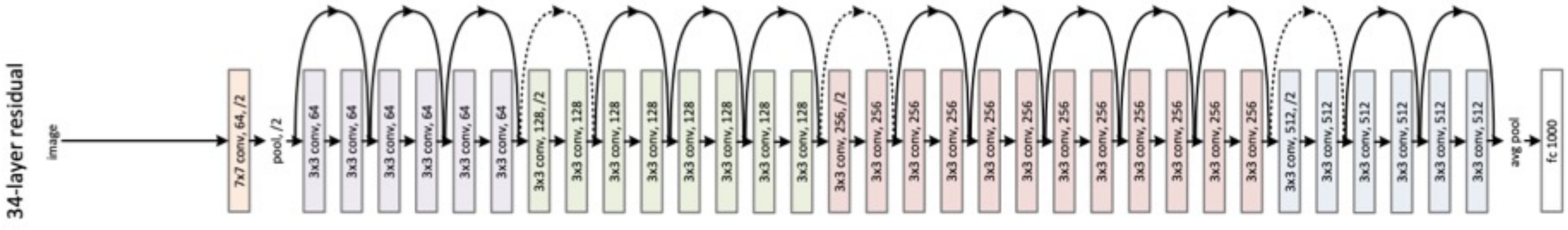
- Consider you want to do image classification on pictures of dogs and predict their breed (say, amongst 10 different breeds)



Brad Quinton, Scott Chin

Transfer Learning with Image Data

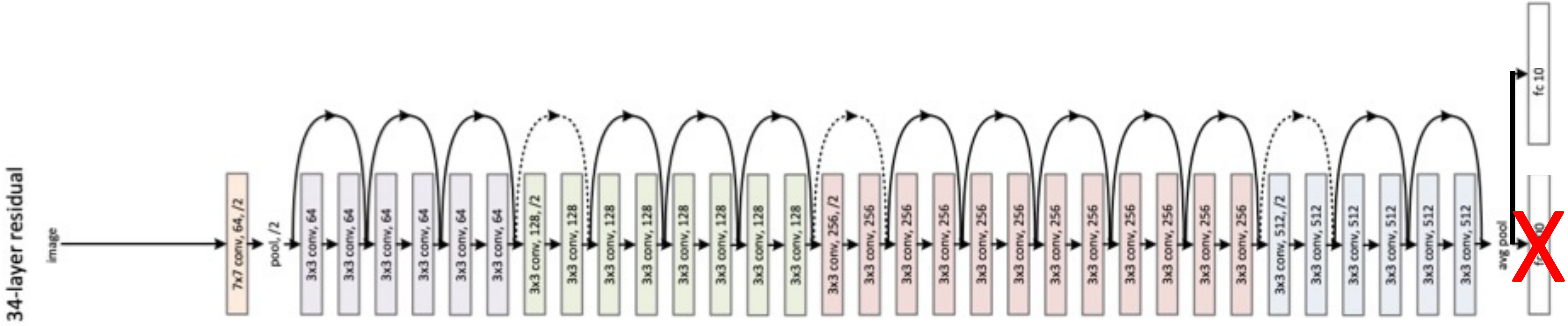
- Start with a CNN trained on a large data set.
- For example, ResNet trained on Imagenet
- Task in this case is image classification on 1000 classes



Transfer Learning with Image Data

One way to repurpose for new task of dog breed classification:

- Replace output layer (1000 outputs) with new output layer (10 outputs)

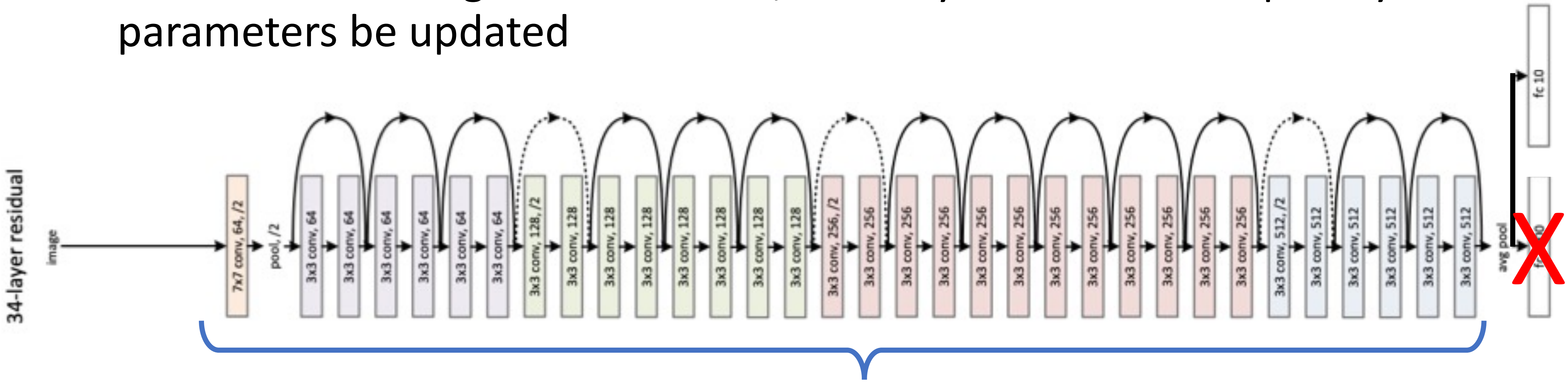


Brad Quinton, Scott Chin

Transfer Learning with Image Data

One way to repurpose for new task of dog breed classification:

- Replace output layer (1000 outputs) with new output layer (10 outputs)
- Train with new dog breed data set, but only let the new output layer's parameters be updated



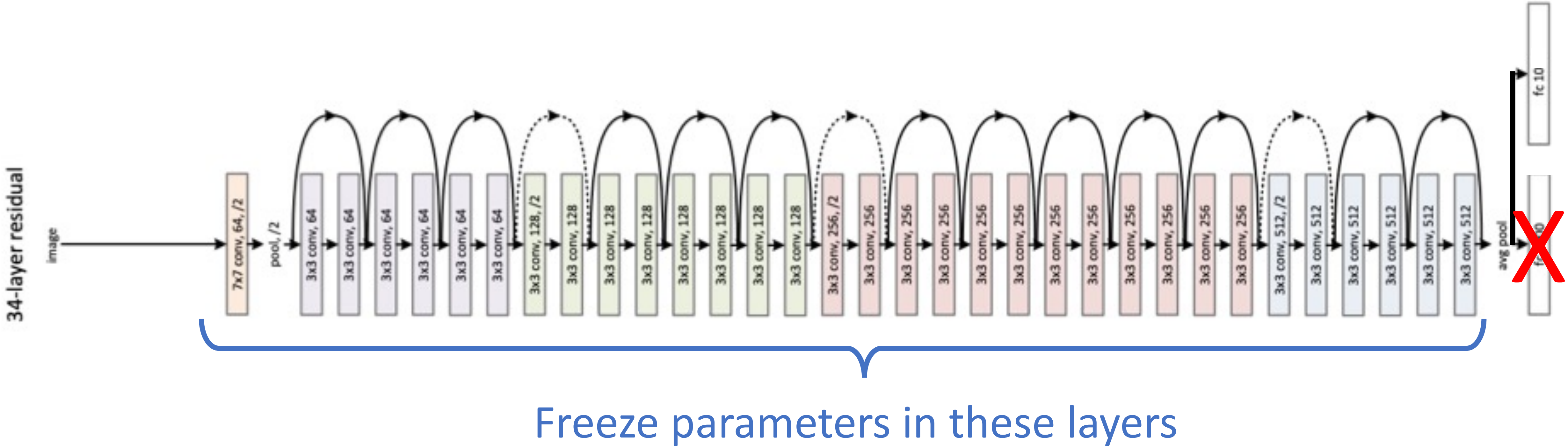
Freeze parameters in these layers

Brad Quinton, Scott Chin

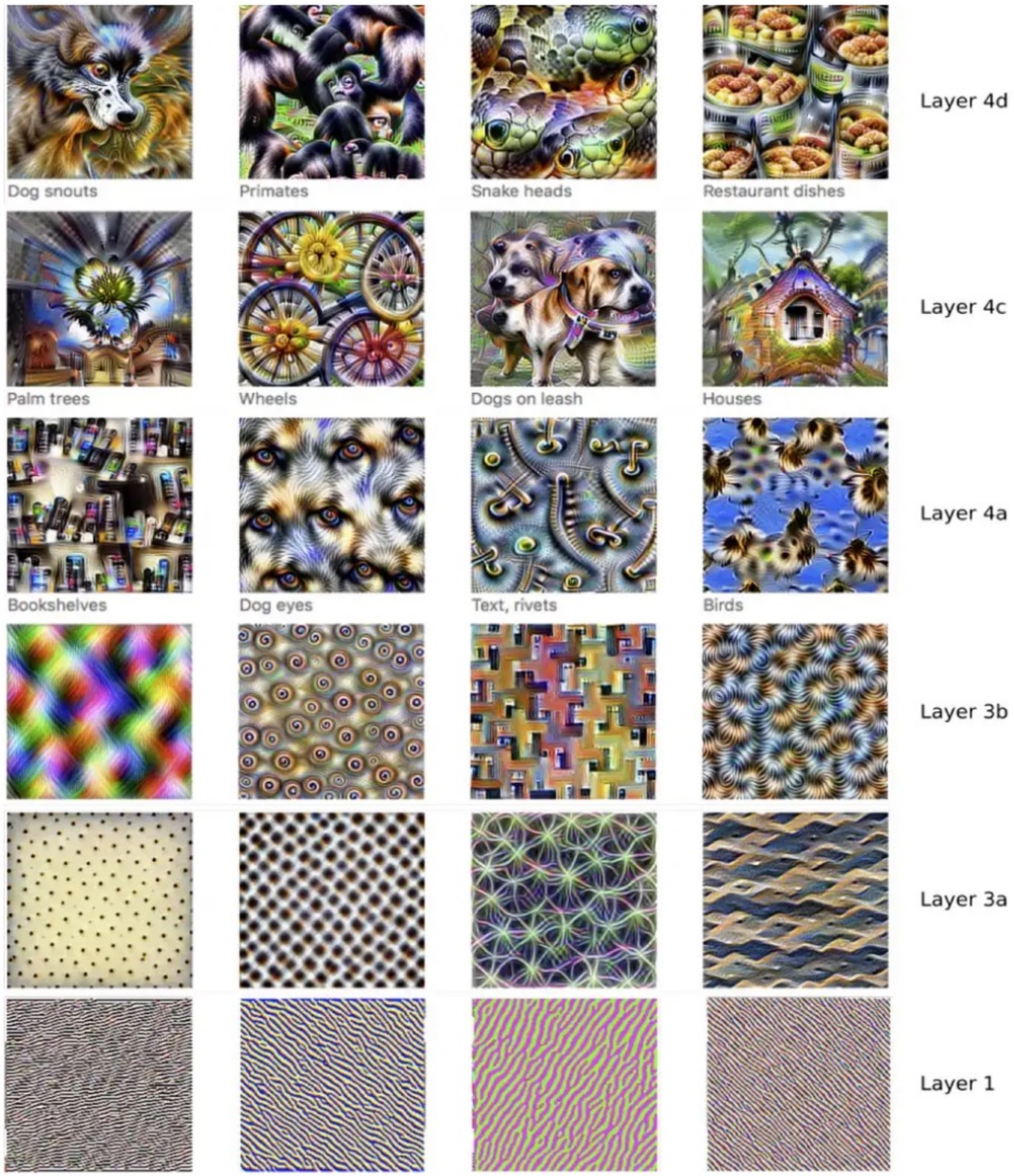
Transfer Learning with Image Data

Intuition for why this might work well

- You expect an ImageNet trained classifier to have learned many important features related to dogs



Brad Quinton, Scott Chin



Intuition

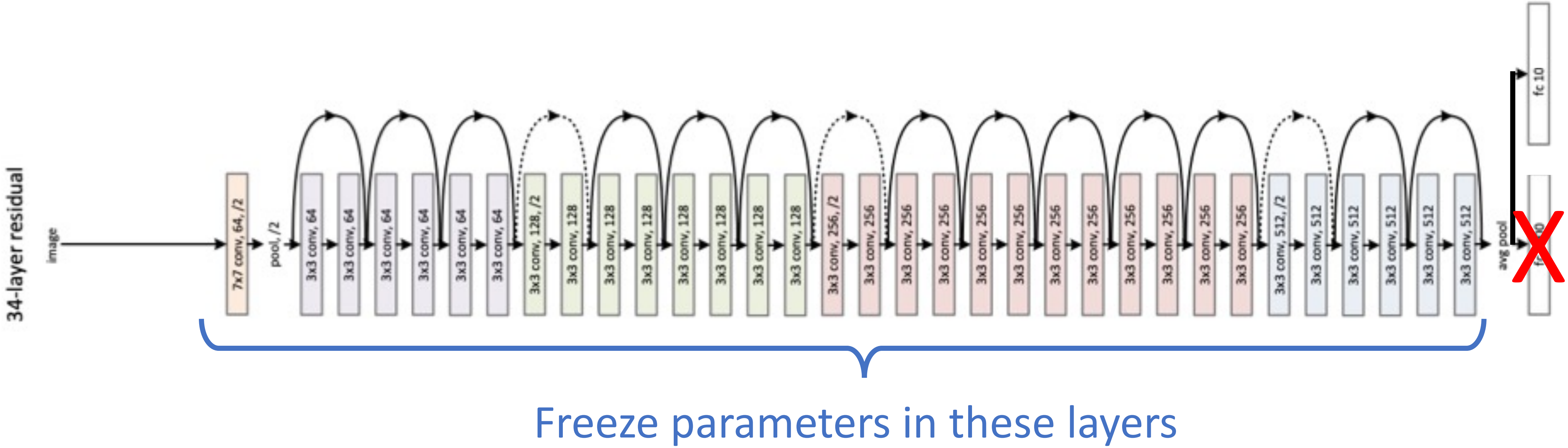
- Early layers of CNNs learn a "vocabulary" of visual constructs (e.g. edges, textures, patterns)
- These are common in any computer vision problem
- Don't need to relearn theses

"Feature Visualization", Olah et al, 2017, https://distill.pub/2017/feature-visualization/

Brad Quinton, Scott Chin

Transfer Learning with Image Data

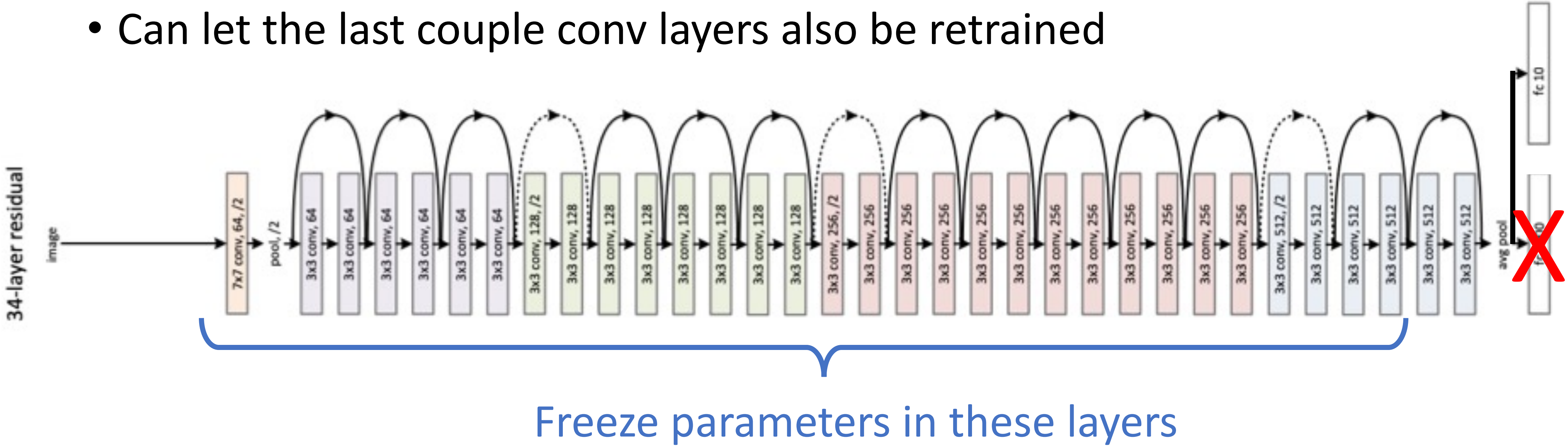
- But maybe your dog breed classifier isn't as accurate as you hoped
- Maybe it needs to learn a little more about features to discriminate between dogs of different breeds



Brad Quinton, Scott Chin

Transfer Learning with Image Data

- But maybe your dog breed classifier isn't as accurate as you hoped
- Maybe it needs to learn a little more about features to discriminate between dogs of different breeds
- Can let the last couple conv layers also be retrained



Brad Quinton, Scott Chin

When Does Transfer Learning Make Sense

- Both tasks have same inputs (e.g. images, audio, language data)
- Significantly less training data available for the new task
  - E.g. pre-trained with Imagenet (1.2 million images) vs your new task of different dog breeds with only about 20,000 images
- Expect low-level features to be similar in both tasks

Brad Quinton, Scott Chin

Benefits of Transfer Learning

- Leverage previous training efforts so don’t need to start from scratch
- Lets you start with very good parameter values
  (i.e. start optimization at a lower loss)
- Don’t need to re-learn common low-level features
  (e.g. lines, textures, geometric shapes, and even higher level objects)
- If you’re new task doesn’t have much data, you can still potentially train a good model because the model was pre-trained on another related task with a much larger data-set

Learning Objectives

- Discuss Regularization strategies for combating overfitting
  - L2 Regularization
  - Dropout
  - Data Augmentation
- Discuss strategies for tuning hyperparameters
- Looking at cost curves for hints at where things can go wrong
- Transfer Learning