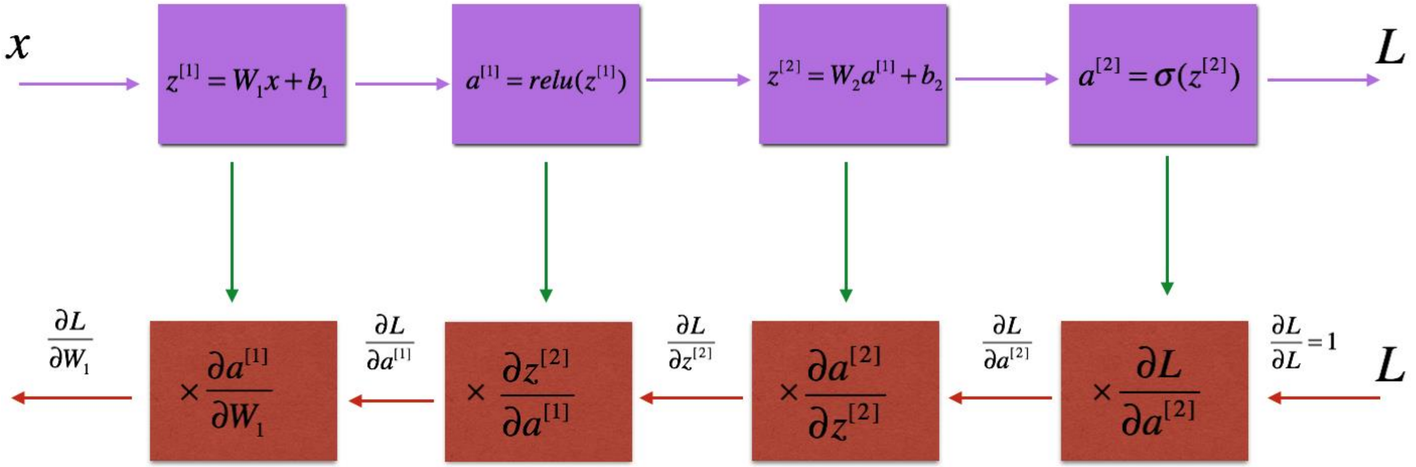
CSCI 1470

Eric Ewing

Monday, 2/24/25

Deep Learning

Day 14: ResNet and Regularization



$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial W_1}$$

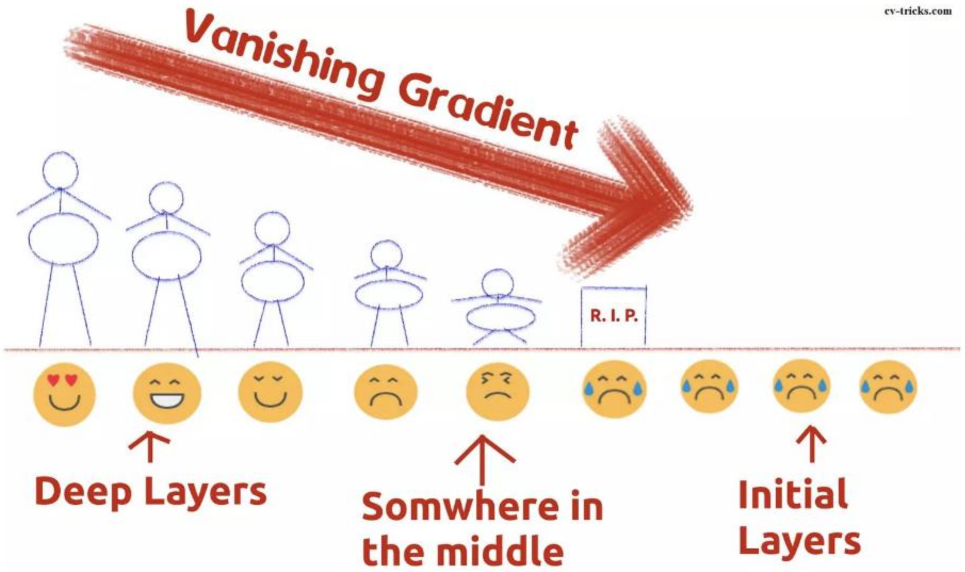
≤1

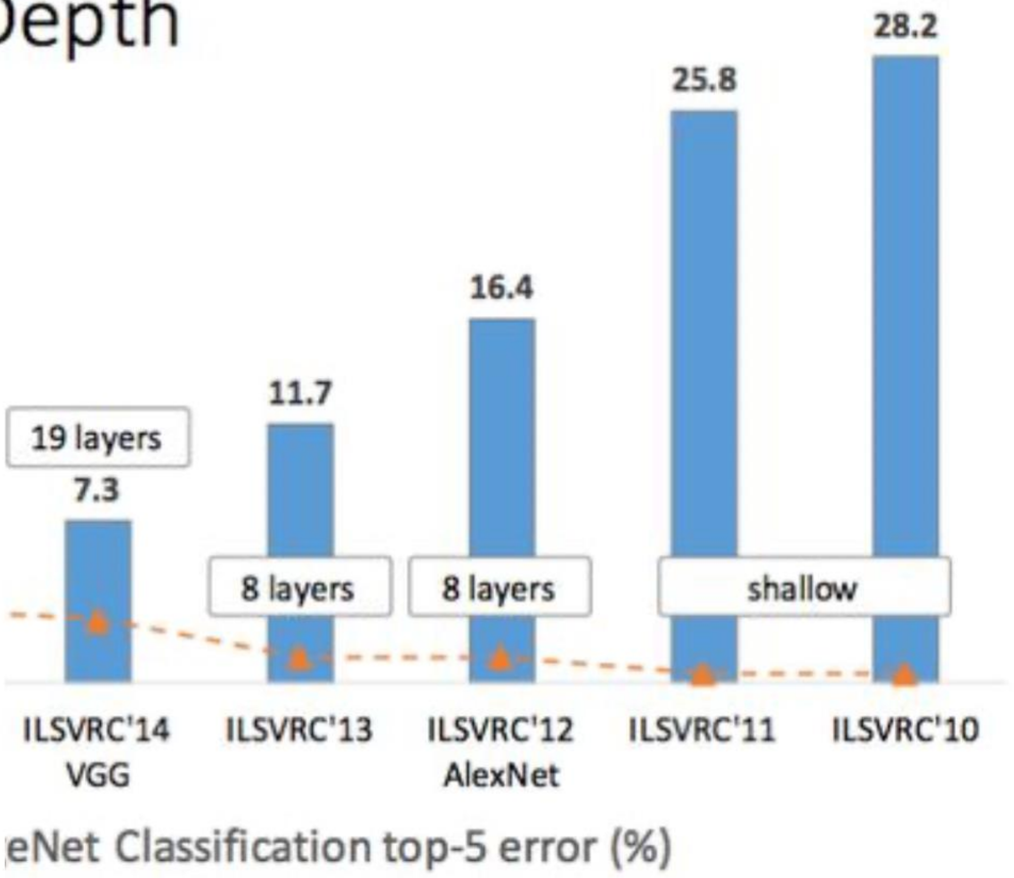
Multiplying by terms ≤1 makes things smaller...
Gradients earlier in the network tend to “Vanish”

Adding more layers adds more terms with gradient ≤1 https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33, https://cv-tricks.com/keras/understand-implement-resnets/

Revolution of Depth

Vanishing gradients!



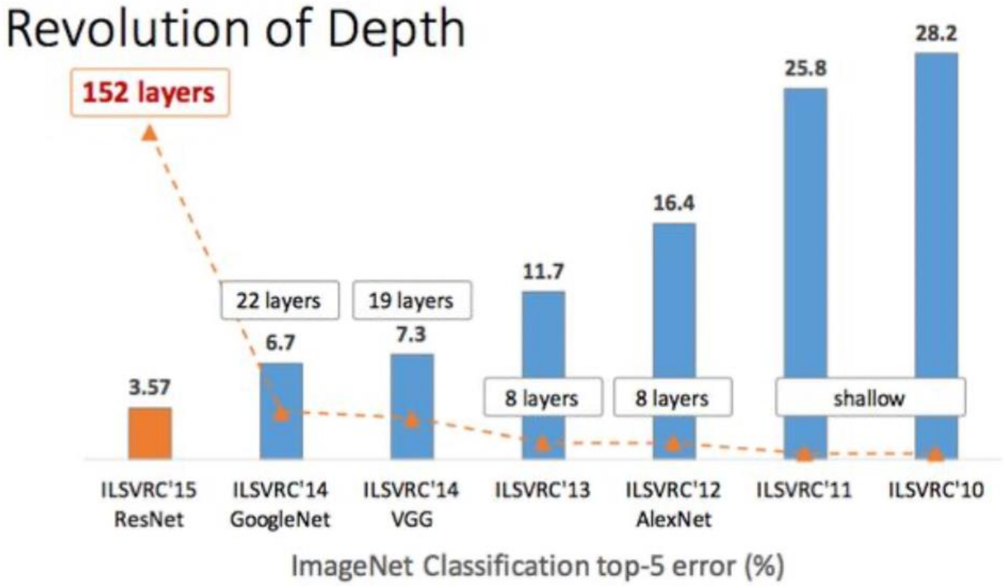


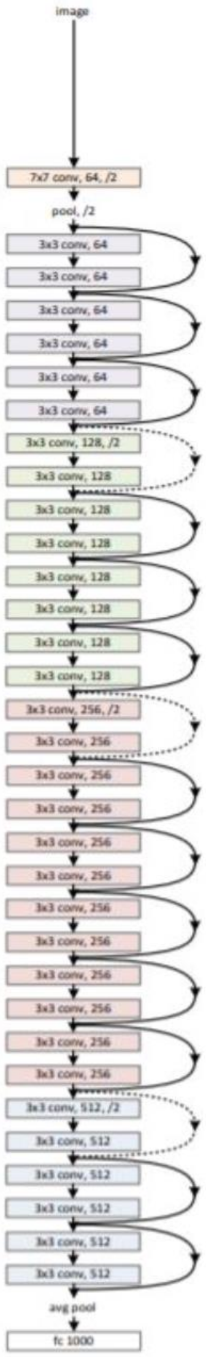
More Complicated Networks

ResNet:

Lots of layers, tons of learnable parameters

Avoids Vanishing Gradient problem
but how?



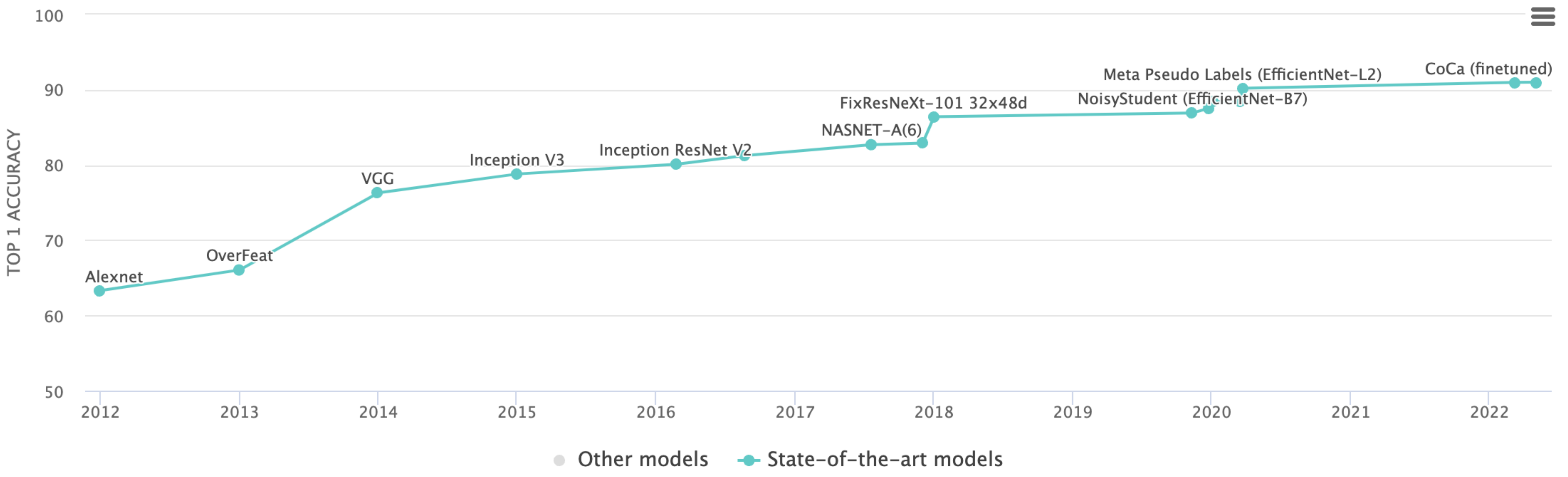


K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

Image Classification on ImageNet

Leaderboard   Community Models   Dataset

View [Top 1 Accuracy] by [Date] for [All models]



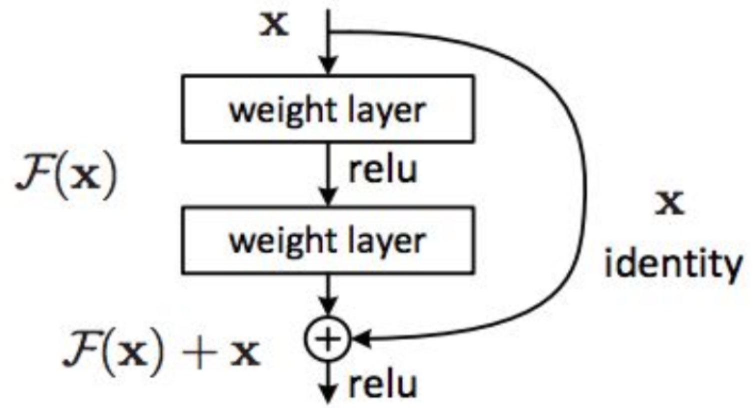
More Complicated Networks

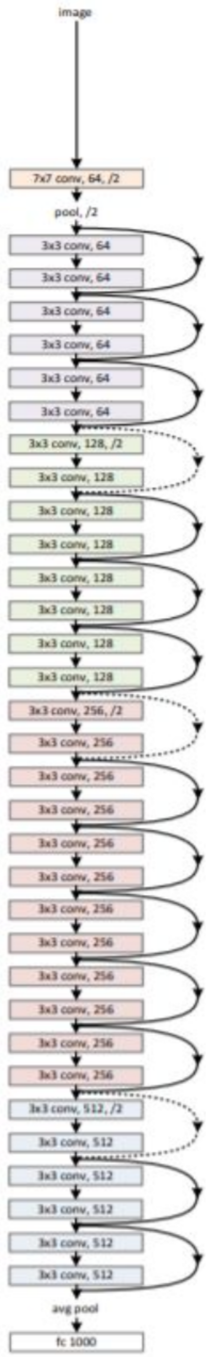
ResNet:

Lots of layers, tons of learnable parameters

Avoids Vanishing Gradient problem

Residual Block

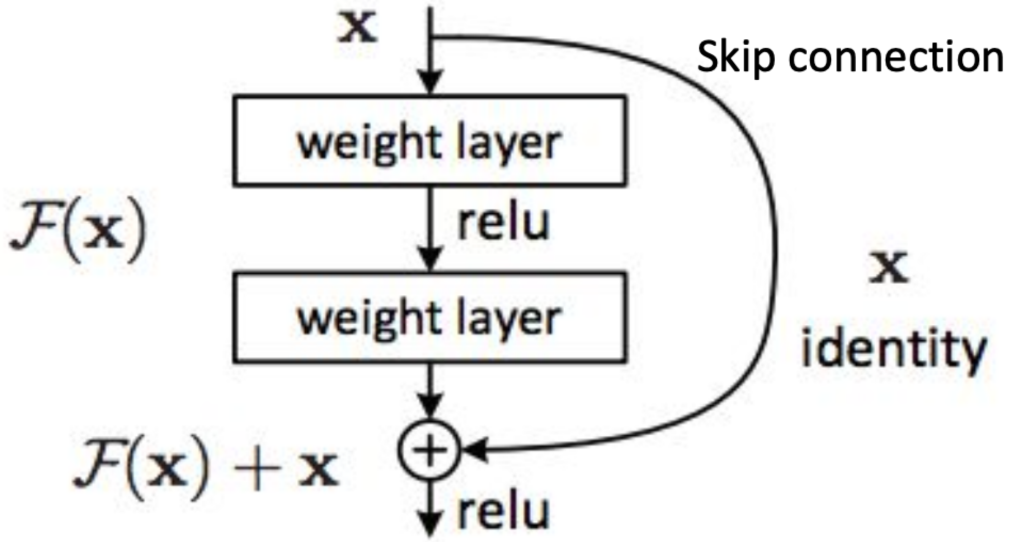




K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

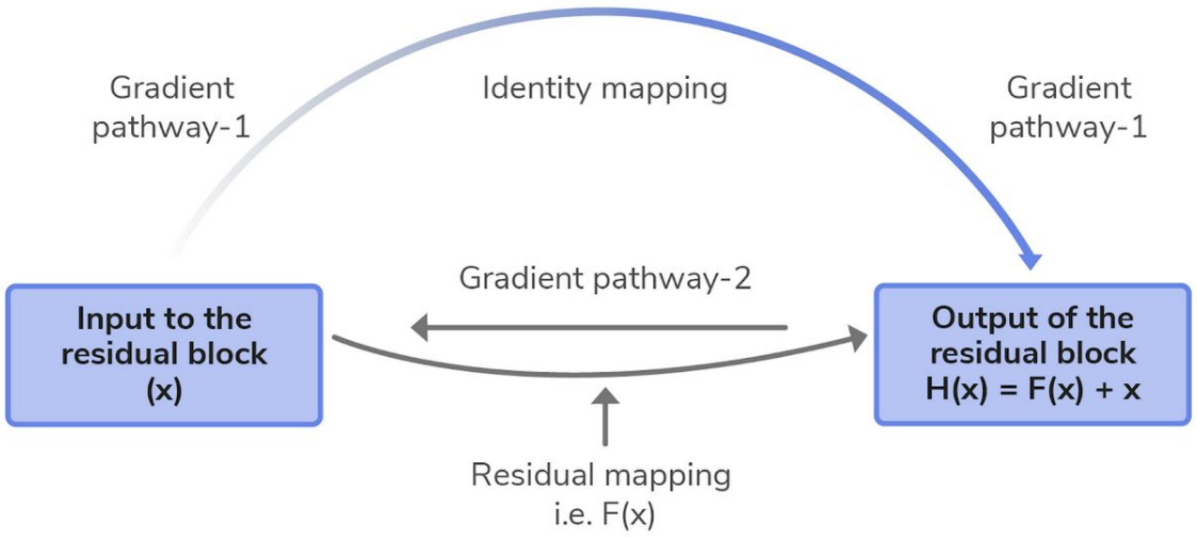
Residual Blocks

- In very deep nets, each layer often needs to learn just a small transformation of the preceding layer (identity + change)
- Idea: explicitly design the network such that the output of each layer is the identity + some deviation from it
  - Deviation is known as a residual



Residual Blocks

- In very deep nets, each layer often needs to learn just a small transformation of the preceding layer (identity + change)
- Idea: explicitly design the network such that the output of each layer is the identi + some deviation from it
  - Deviation is known as a residual
- Allows gradient to flow through two pathways
- **Significantly stabilizes training of very deep networks**



https://blog.perceptilabs.com/using-resnets-to-detect-anomalies-in-industrial-iot-textile-production/

Tensorflow

Option #1: Residual Block

tfm.vision.layers.ResidualBlock(filters, strides)

Option #2:

```python
# Residual Block
def ResBlock(inputs):
    x = layers.Conv2D(64, 3, padding="same", activation="relu")(inputs)
    x = layers.Conv2D(64, 3, padding="same")(x)
    x = layers.Add()([inputs, x])
    return x
```

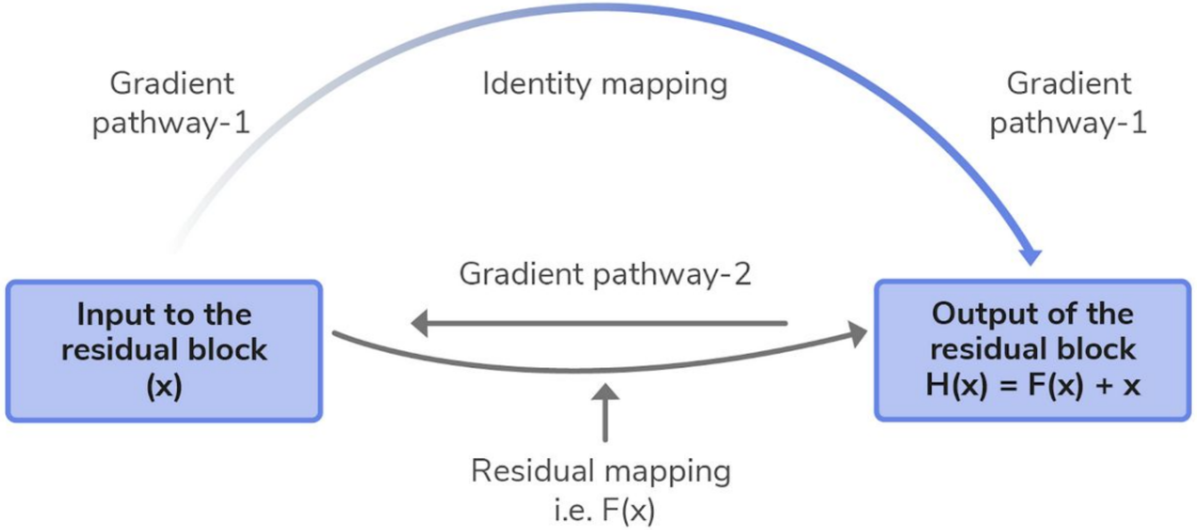
Original Input   Intermediate Output

https://keras.io/examples/vision/edsr/

Any questions?

Residual Blocks

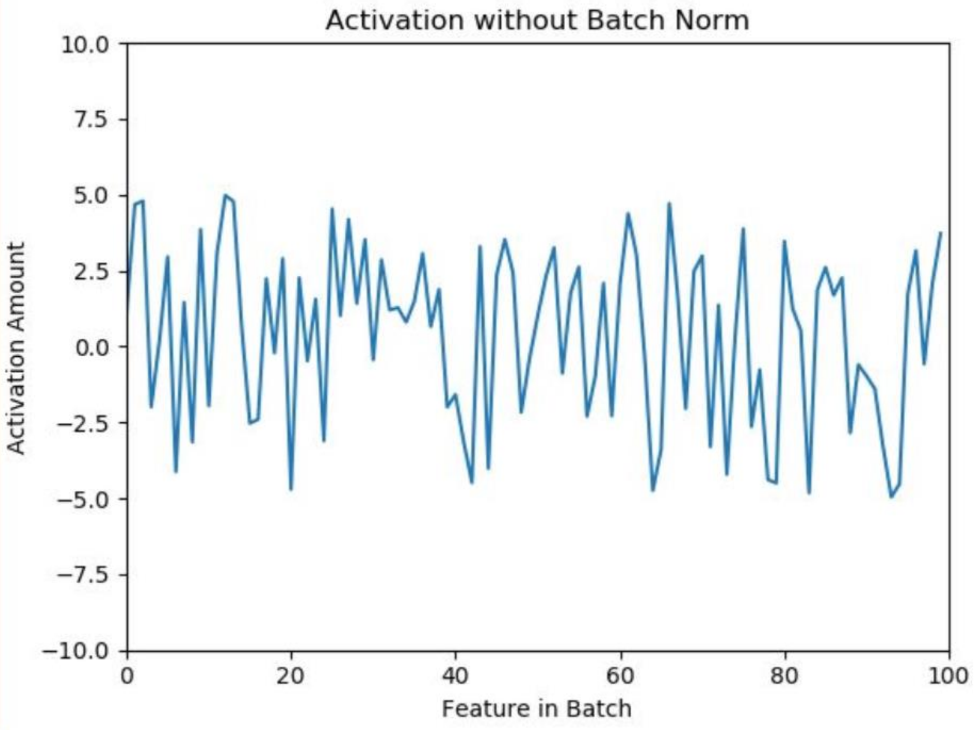
- In very deep nets, each layer often needs to learn just a small transformation of th preceding layer (identity + change)
- Idea: explicitly design the network such that the output of each layer is the identi + some deviation from it
  - Deviation is known as a residual
- Allows gradient to flow through two pathways
- **Significantly stabilizes training of very deep networks**

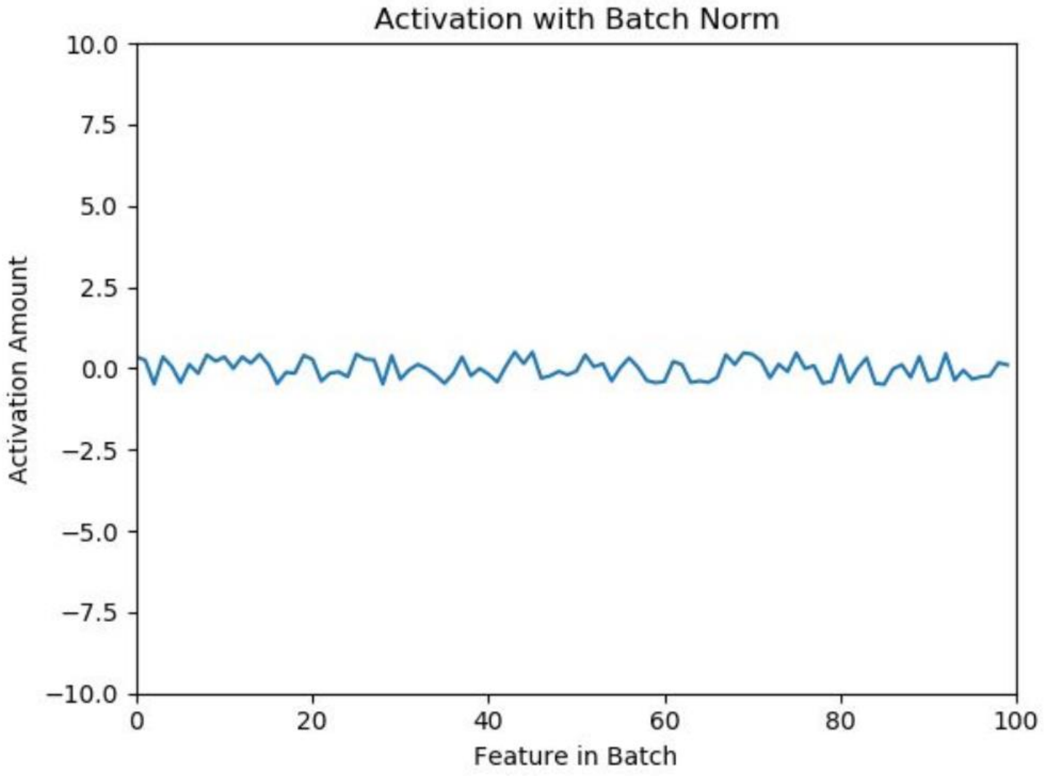


https://blog.perceptilabs.com/using-resnets-to-detect-anomalies-in-industrial-iot-textile-production/

Batch Normalization (stabilizing training)

Idea: normalize the activations for each feature at each layer



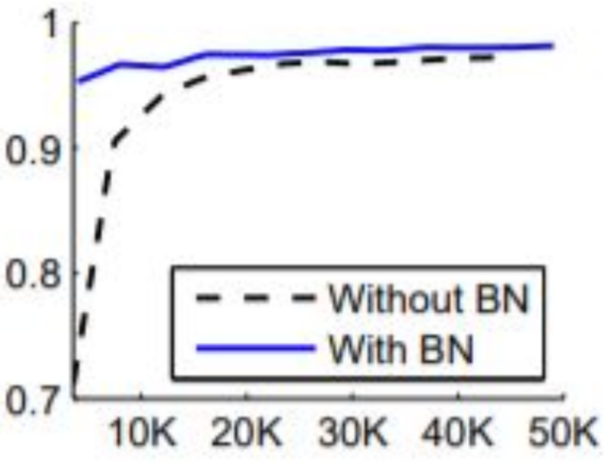


Why might we want to do this?

Batch Normalization: Motivation

More stable inputs = faster training

MNIST test accuracy vs number of training steps



https://arxiv.org/pdf/1502.03167.pdf

Batch Normalization: Implementation

For each feature x, Start by calculating the batch mean and standard deviation for each feature:

$$\mu_{batch} = \frac{\sum_{i=0}^{batch\_size} x_i}{batch\_size}$$

$$\sigma_{batch} = \sqrt{\frac{\sum_{i=0}^{batch\_size} (x_i - \mu_{batch})^2}{batch_{size}}}$$

Batch Normalization: Implementation

Normalize by subtracting feature x's batch mean, then divide by batch standard deviation.

$$x' = \frac{x \ - \mu_{batch}}{\sigma_{batch}}$$

Feature x now has mean 0 and variance 1 along the batch

Batch Normalization in Tensorflow

```
tf.keras.layers.BatchNormalization(input)
```

Documentation: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers/BatchNormalization

Motivation of BatchNorm

- Reduce “internal co-variate shift”
- Neural networks are trained on a certain distribution of data and are expected to be tested on the same distribution
- If we were to scale the colors of an image significantly at test time, we wouldn’t expect a neural network to do well
- The same can be said for our intermediate layers
  - They expect a certain distribution of inputs, if that changes significantly from example to example, it will be hard to learn
- (Most commonly cited reason for using BatchNorm)

The only issue is that controlling internal covariate shift does not matter that much…

How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu

Dimitris Tsipras* MIT tsipras@mit.edu

Andrew Ilyas* MIT ailyas@mit.edu

Aleksander Mądry MIT madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

BatchNorm makes the loss landscape smoother with fewer local minima, saddle points, and other problematic areas for gradient descent

How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu

Dimitris Tsipras* MIT tsipras@mit.edu

Andrew Ilyas* MIT ailyas@mit.edu

Aleksander Mądry MIT madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

Theory, intuition, and experimental results can all tell you different things

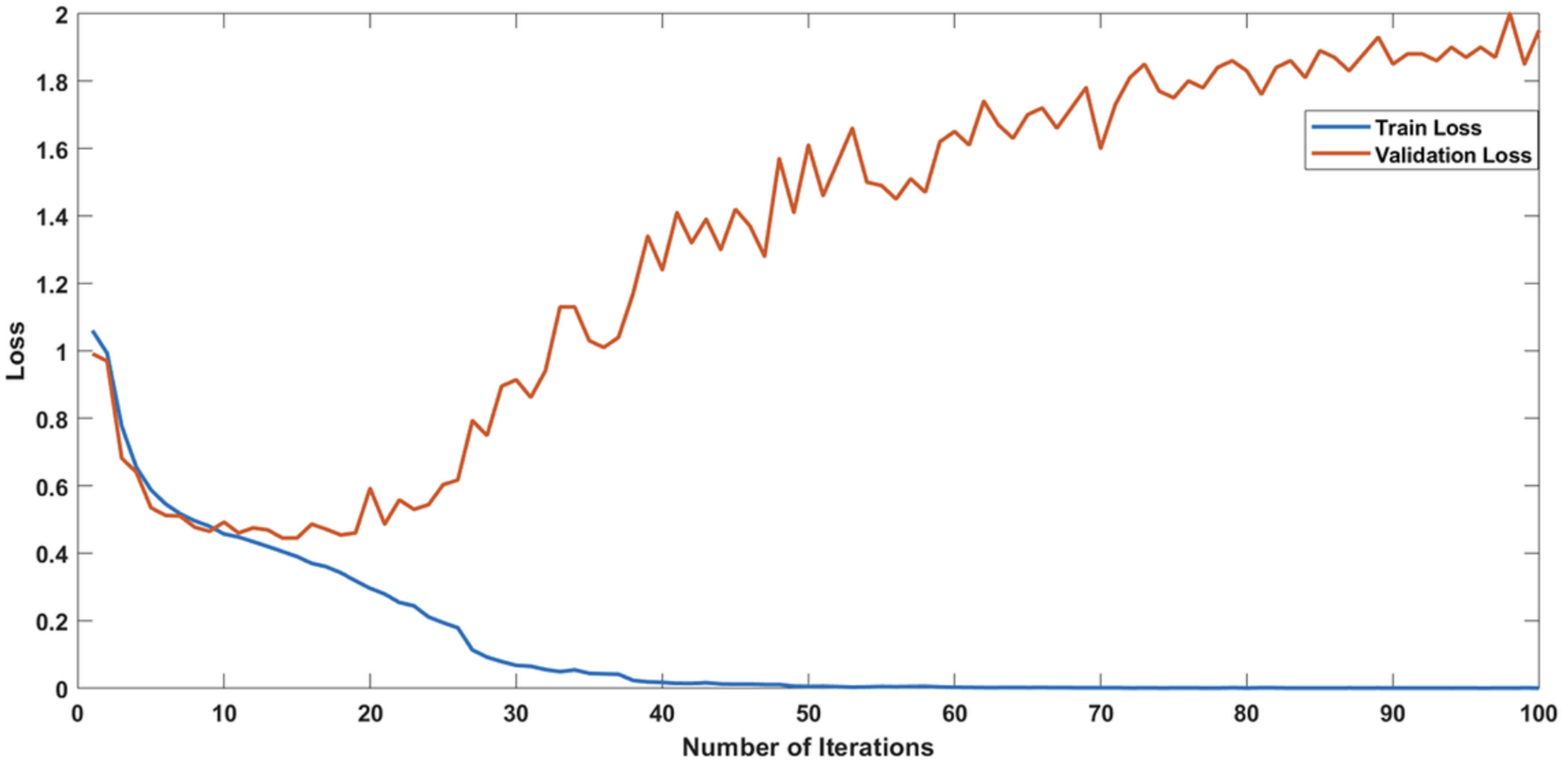
Why does BatchNorm work so well?
Intuition: If normalizing input data works so well for training, why not normalize input features to intermediate layers?

Theory/experiments: Makes gradients of loss function "better"

Why do CNNs work so well?
Intuition: Looking for a way to get "spatial reasoning" or translational invariance

Theory/experiments: Maybe it's just that using fewer weights lets us go deeper and deep networks learn better (and also they have spatial reasoning)

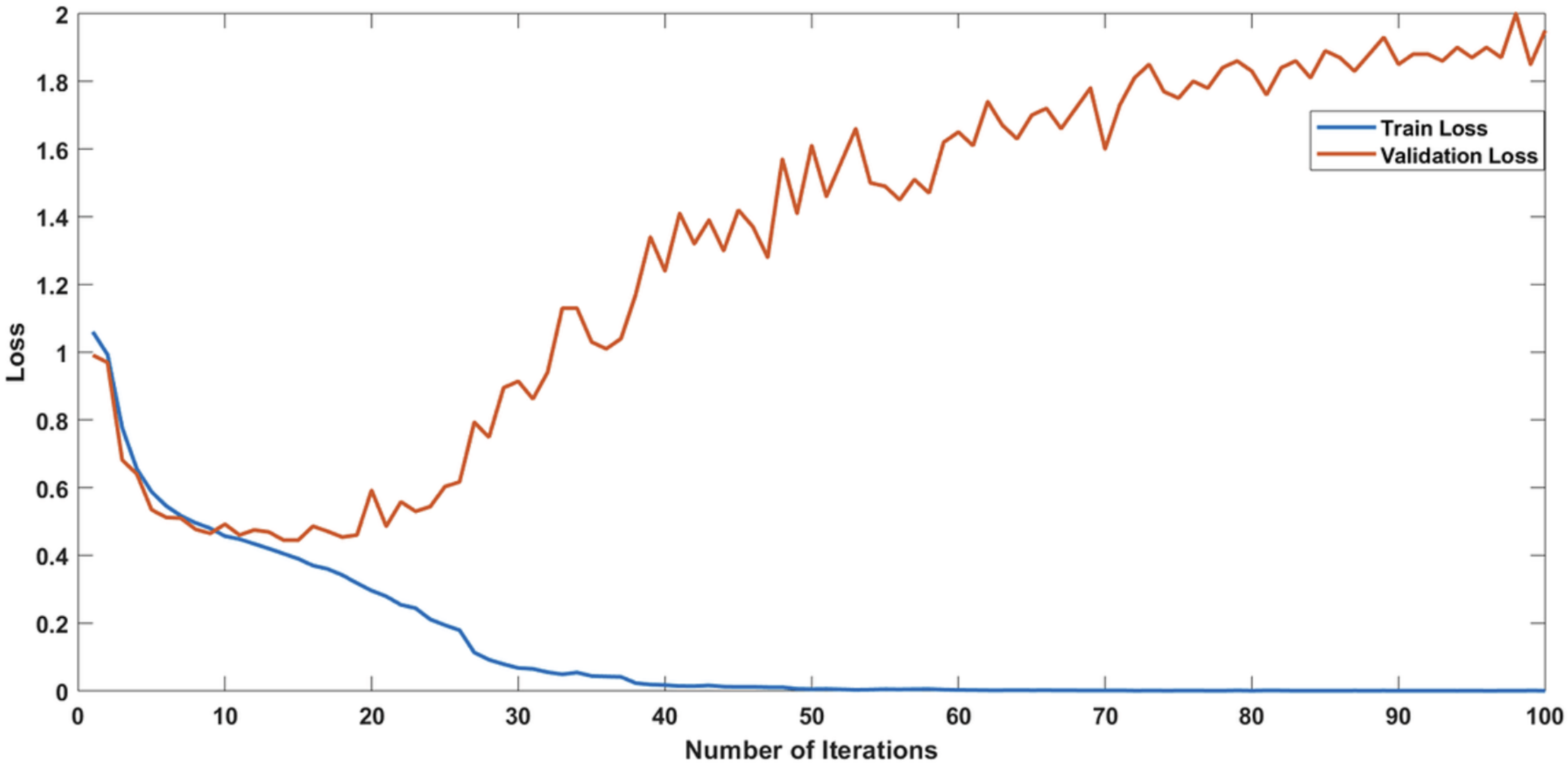
Depth Giveth and Depth Taketh Away



Resnet trained on image classification task

Depth Giveth and Depth Taketh Away

What's the problem?



Resnet trained on image classification task

Dealing with Overfitting (Again)

Option #1: Hyperparameter Tuning

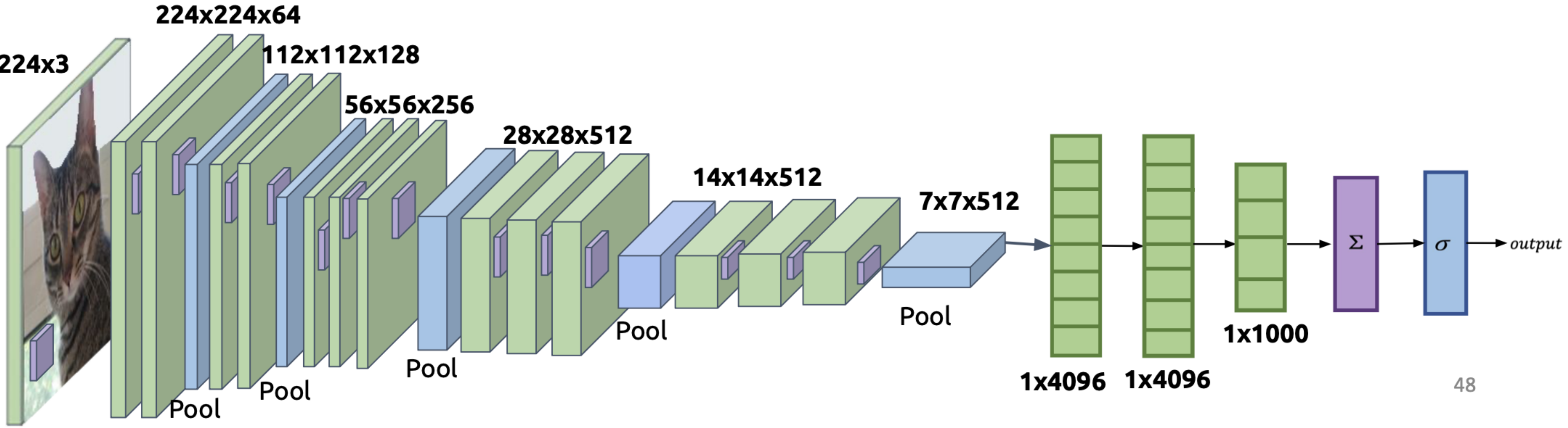
- Try a shallower network

Dealing with Overfitting (Again)

Option #1: Hyperparameter Tuning

- Try a shallower network

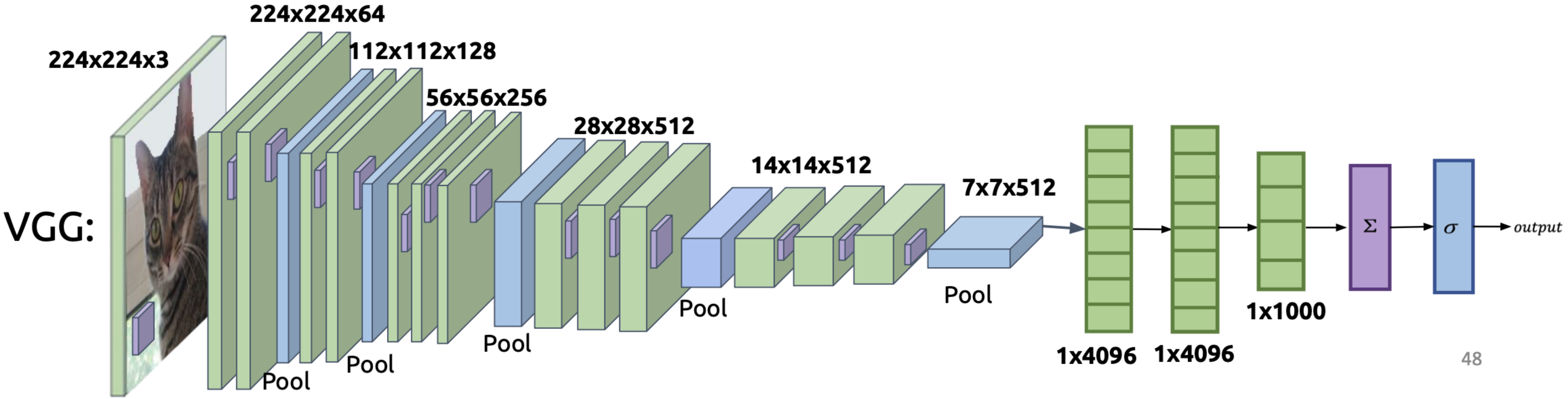
VGG:



Dealing with Overfitting (Again)

Option #1: Hyperparameter Tuning

- Try a shallower network



The size of the linear layer is controlled by number of max-pools
Fewer convolutions could actually increase weights in the network…

Dealing with Overfitting (Again)

Option #1: Hyperparameter Tuning

- Try a shallower network
- Fewer channels in convolutions

Hyperparameter Tuning

- Manually tuning parameters is seen by DL practitioners as a bit “old fashioned”
  - The goal of deep learning is to automatically find good models in a general way
  - Any human-driven heuristic approach makes the process specific

Hyperparameter Tuning

- Manually tuning parameters is seen by DL practitioners as a bit "old fashioned"
  - The goal of deep learning is to automatically find good models in a general way
  - Any human-driven heuristic approach makes the process specific

Can we write a method to ___ and then run deep learning on that output?

(center the image, recognize letters on signs, label parts of a sentence)

The Bitter Lesson of AI

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin.

Richard Sutton

The Bitter Lesson of AI

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin.

Richard Sutton

1) AI researchers have often tried to build knowledge into their agents
2) This always helps in the short term, and is personally satisfying to the researcher, but
3) In the long run it plateaus and even inhibits further progress
4) Breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning.

Hyperparameter Tuning

- Manually tuning parameters is seen by DL practitioners as a bit “old fashioned”
  - The goal of deep learning is to automatically find good models in a general way
  - Any human-driven heuristic approach makes the process specific

Can we write a method to ___ and then run deep learning on that output?
(center the image, recognize letters on signs, label parts of a sentence)

Manual hyperparameter tuning is a flaw that needs to be overcome

Dealing with Overfitting (Again)

Option #1: Hyperparameter Tuning

- Try a shallower network
- Fewer channels in convolutions

Option #2: Regularization

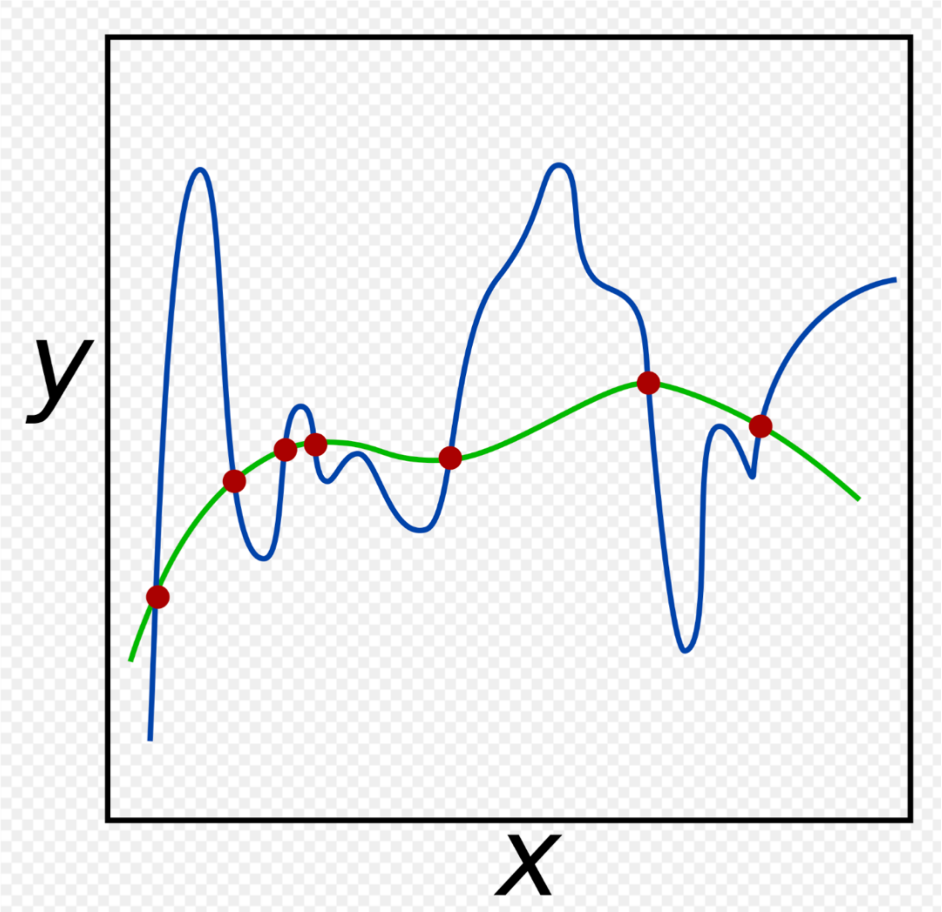
- “Encourage” model to be lower complexity

Regularization: L2 Norm Penalty

Intuition: high degree polynomials typically don't work for regression tasks because they overfit.

When they overfit, the parameters of some terms get very large.

Let's penalize the model for having large parameters.



Regularization: L2 Norm Penalty

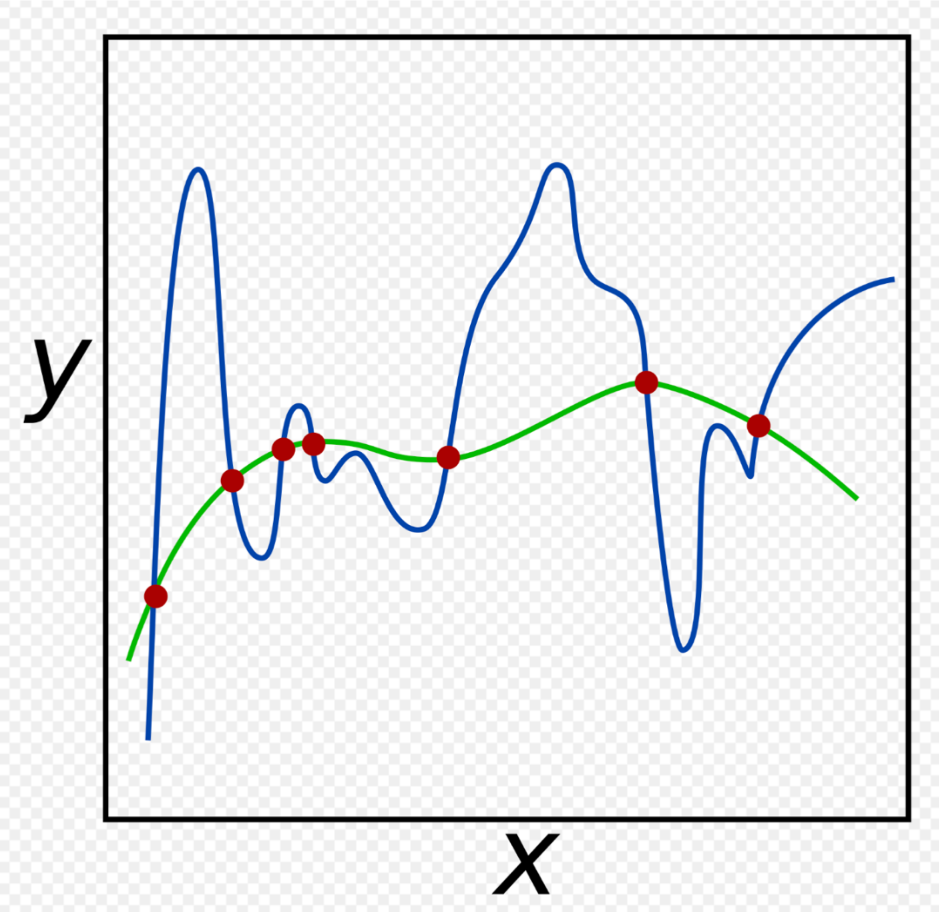
Intuition: high degree polynomials typically don't work for regression tasks because they overfit.

When they overfit, the parameters of some terms get very large.

Let's penalize the model for having large parameters.

Original Loss=$MSE(y, \hat{y})$

L2 Regularization Loss =$MSE(y, \hat{y})$ + $\lambda(w_0^2 + w_1^2 + w_2^2 \ldots)^{\frac{1}{2}}$



Regularization: L2 Norm Penalty

Intuition: high degree polynomials typically don't work for regression tasks because they overfit.

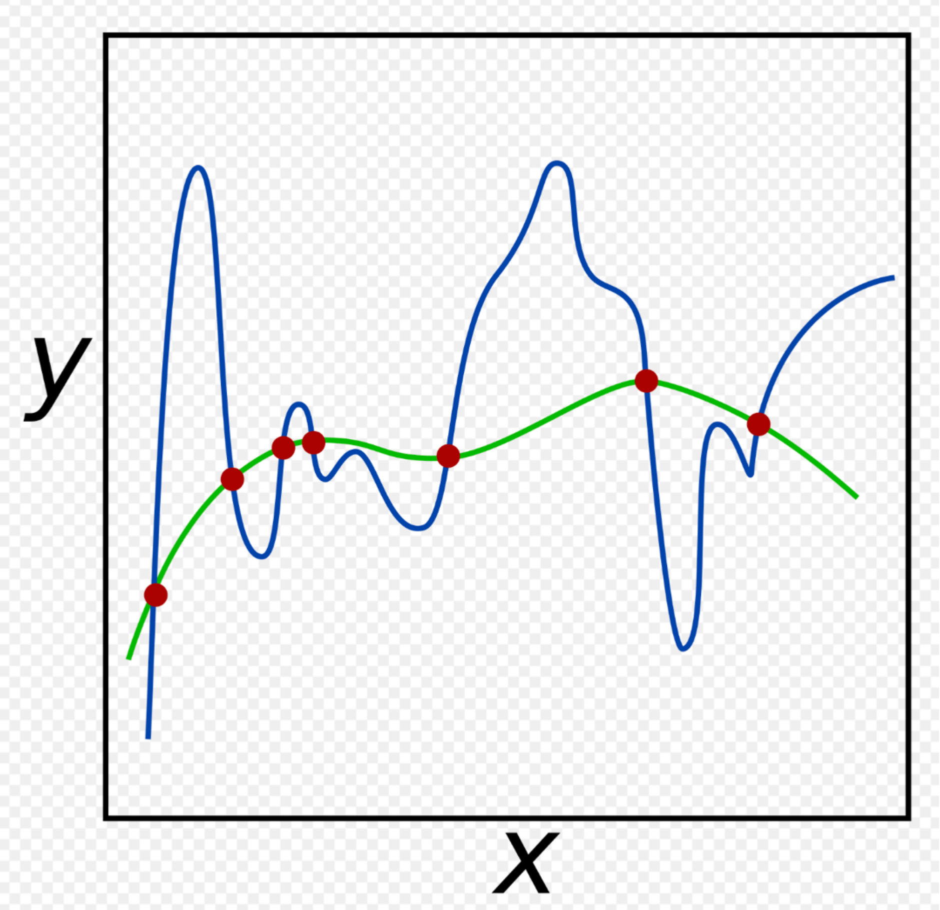
When they overfit, the parameters of some terms get very large.

Let's penalize the model for having large parameters.

Original Loss=$MSE(y, \hat{y})$

L2 Regularization Loss =$MSE(y, \hat{y}) + \lambda(w_0^2 + w_1^2 + w_2^2 \ldots)^{\frac{1}{2}}$

L2 Norm (2 refers to power)



Regularization L2 Norm Penalty

- Why do neural networks overfit? Perhaps their weights get large as well.
- Can add a penalty to all weights or individual layers
- Smaller weights → simpler function learned

```
from keras import regularizers

model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01)))
```

Dropout – general intuition

- Preventing the network from learning under perfect conditions; that is, make it **harder** for the network to learn

A climbing analogy:

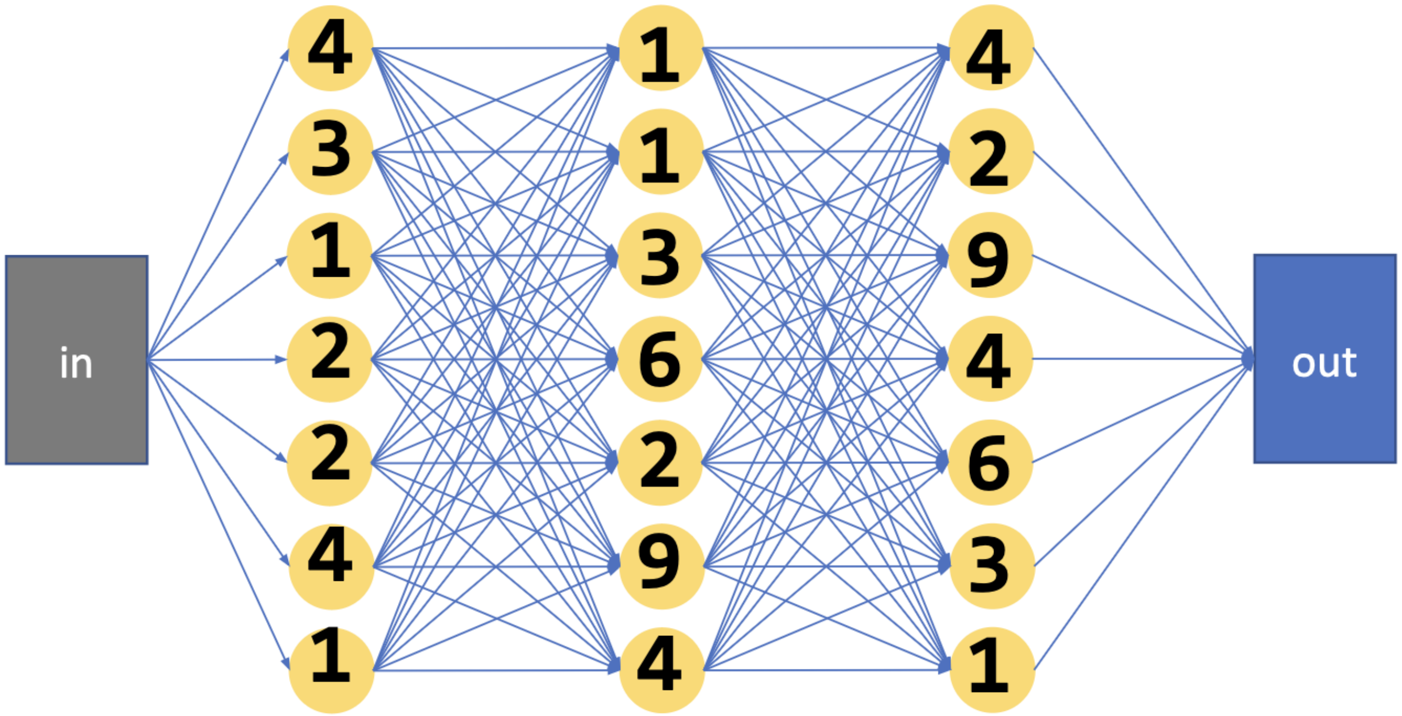
A person is climbing a wall using holds

- What if, I make a rule that she can climb
- ... only using certain holds (say just green ones!)
- If she can learn to do this using fewer holds...
- ...she'll definitely be able to do it with ALL the holds
- (learn better climbing techniques in the process)

Dropout ~= using only a certain holds instead of ALL the holds

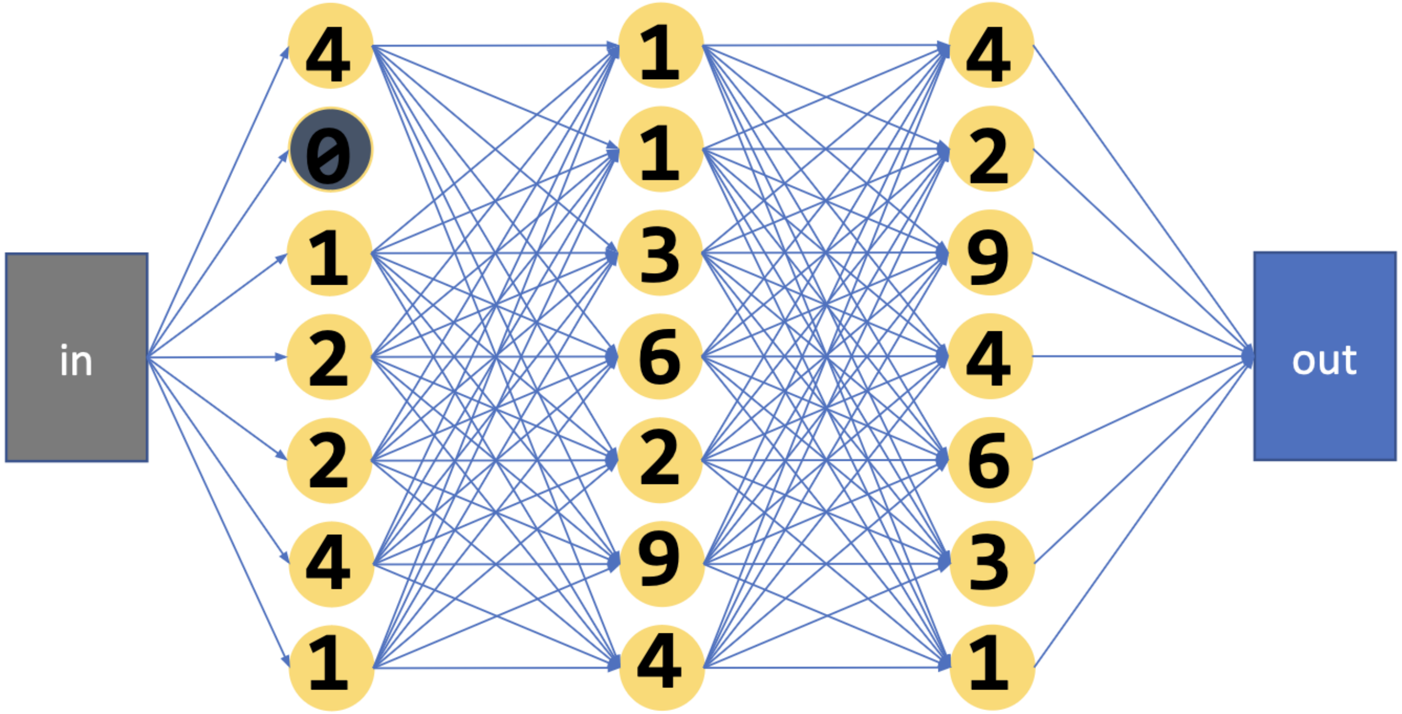
Image source https://www.istockphoto.com/illustrations/indoor-climbing

Dropout - what?



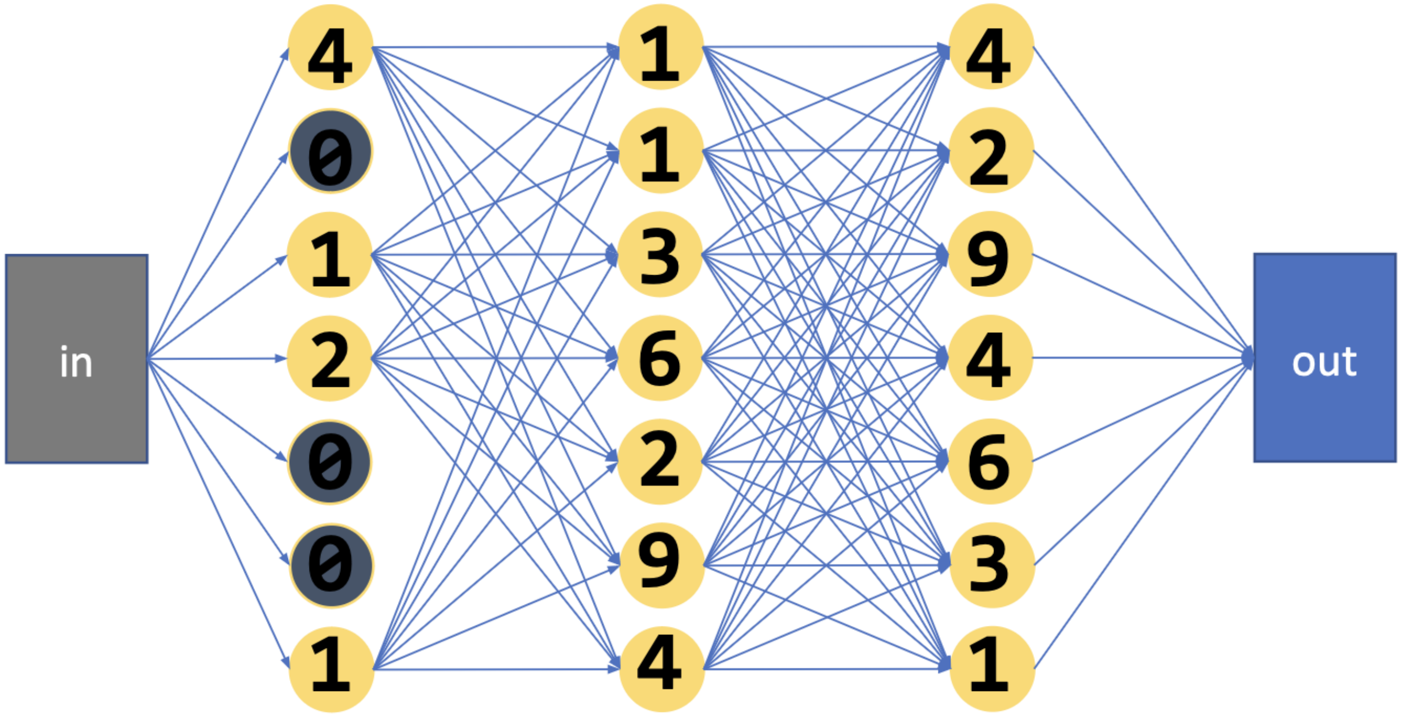
Typical NN: the output of every node in every layer is used in the next layer of the network

Dropout - what?



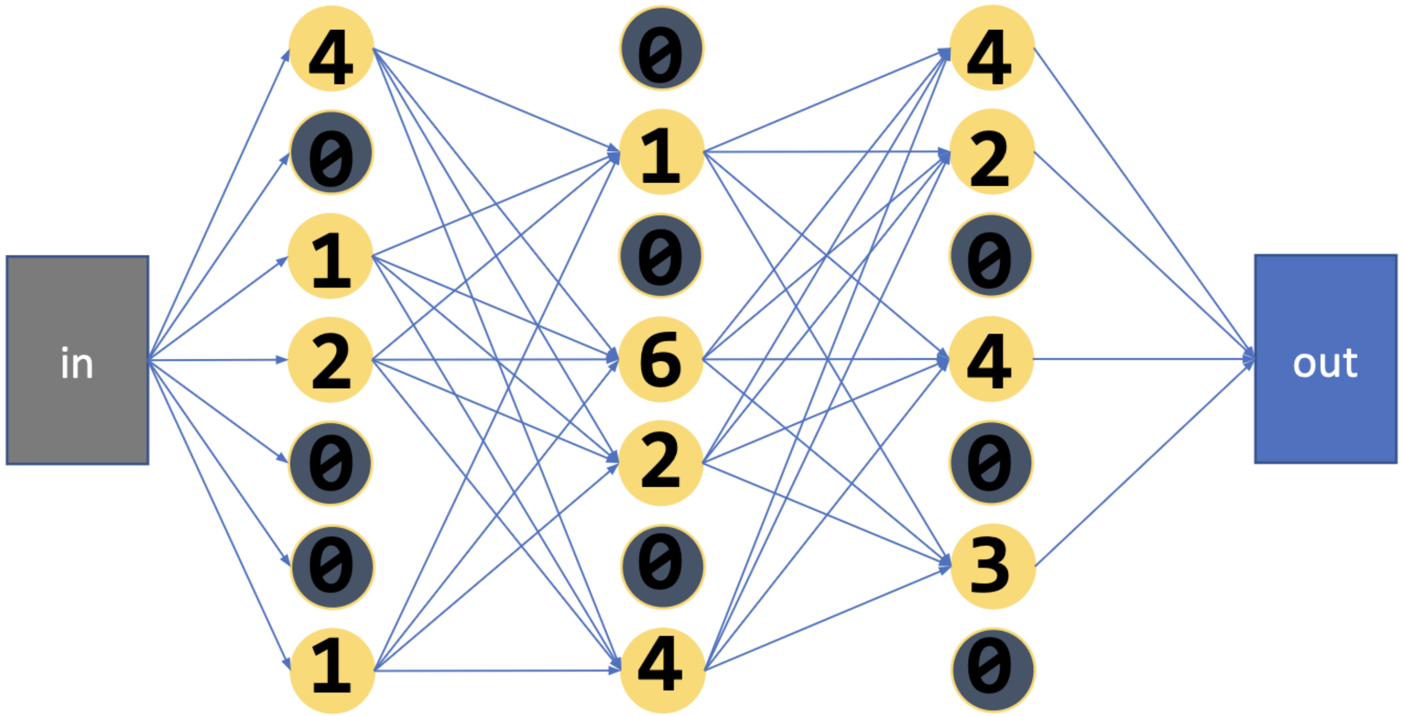
Dropout: *in a single training pass*, the output of randomly selected nodes from each layer will “drop out”, i.e. be set to 0

Dropout - what?



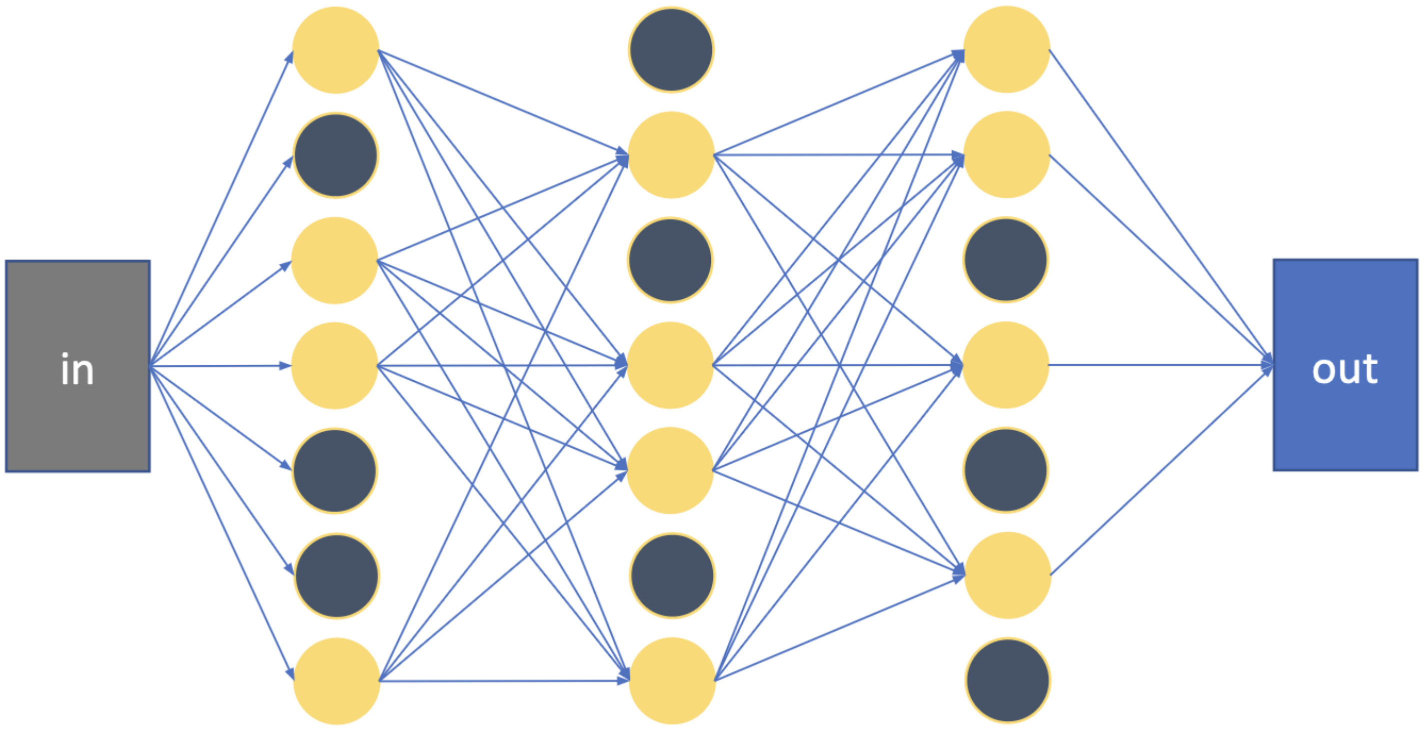
Dropout: *in a single training pass*, the output of randomly selected nodes from each layer will "drop out", i.e. be set to 0

Dropout - what?



Not just limited to the input layer: can do this to *any* layer of the network

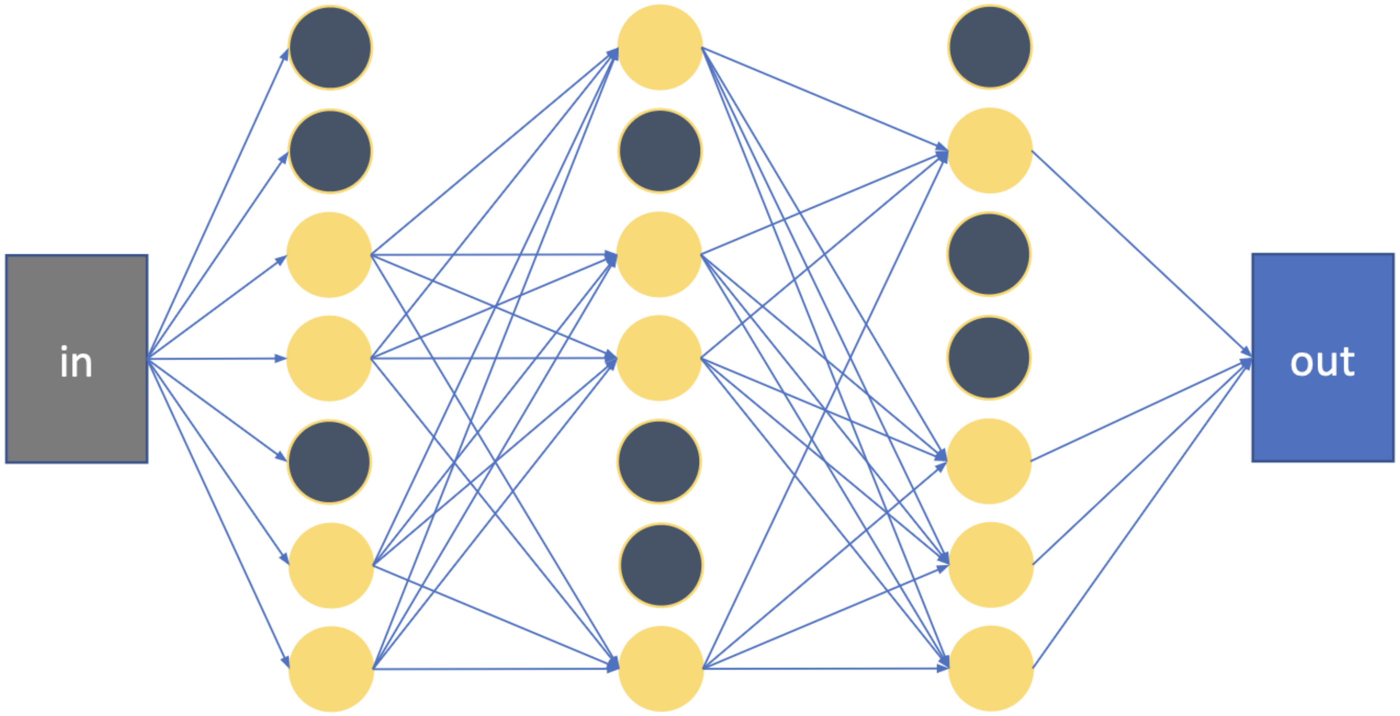
Dropout - what?



The nodes that drop out will be different each pass

(re-randomly selected)

Dropout - what?



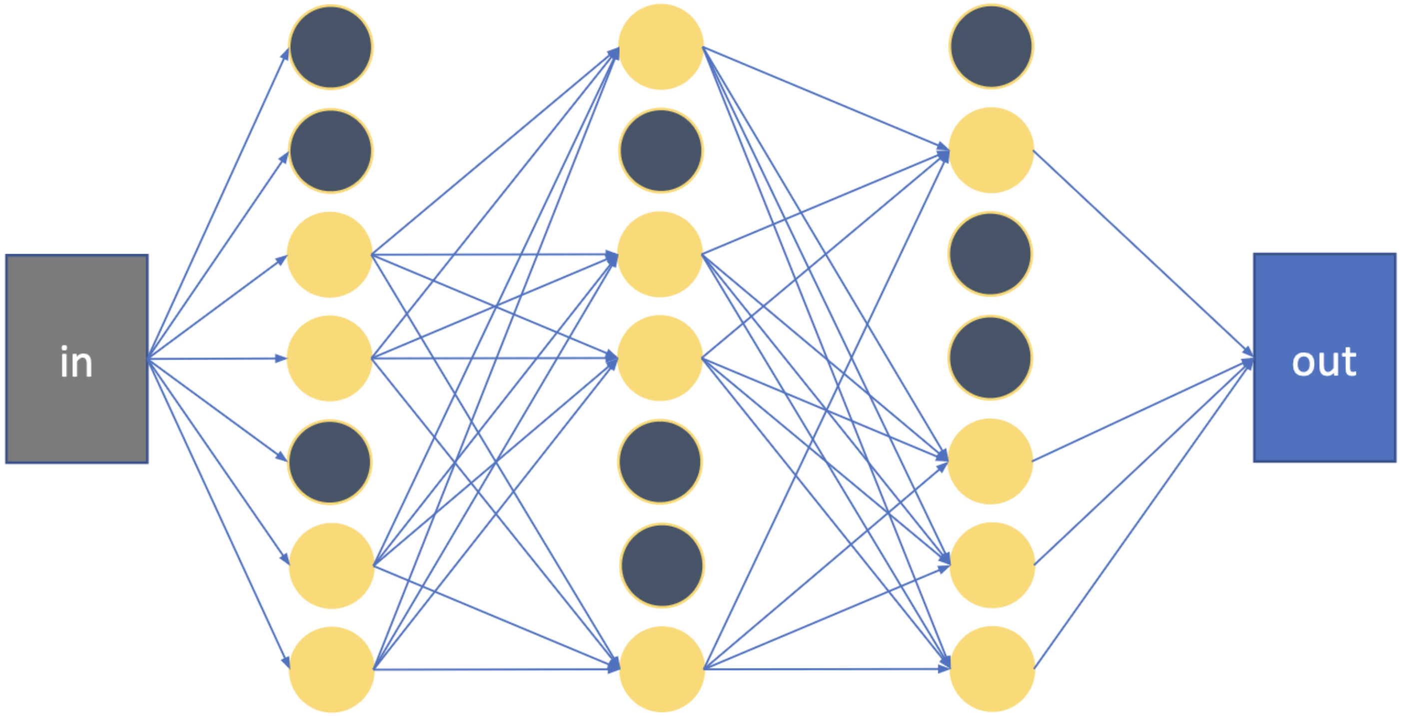
The nodes that drop out will be different each pass

(re-randomly selected)

Dropout - why?

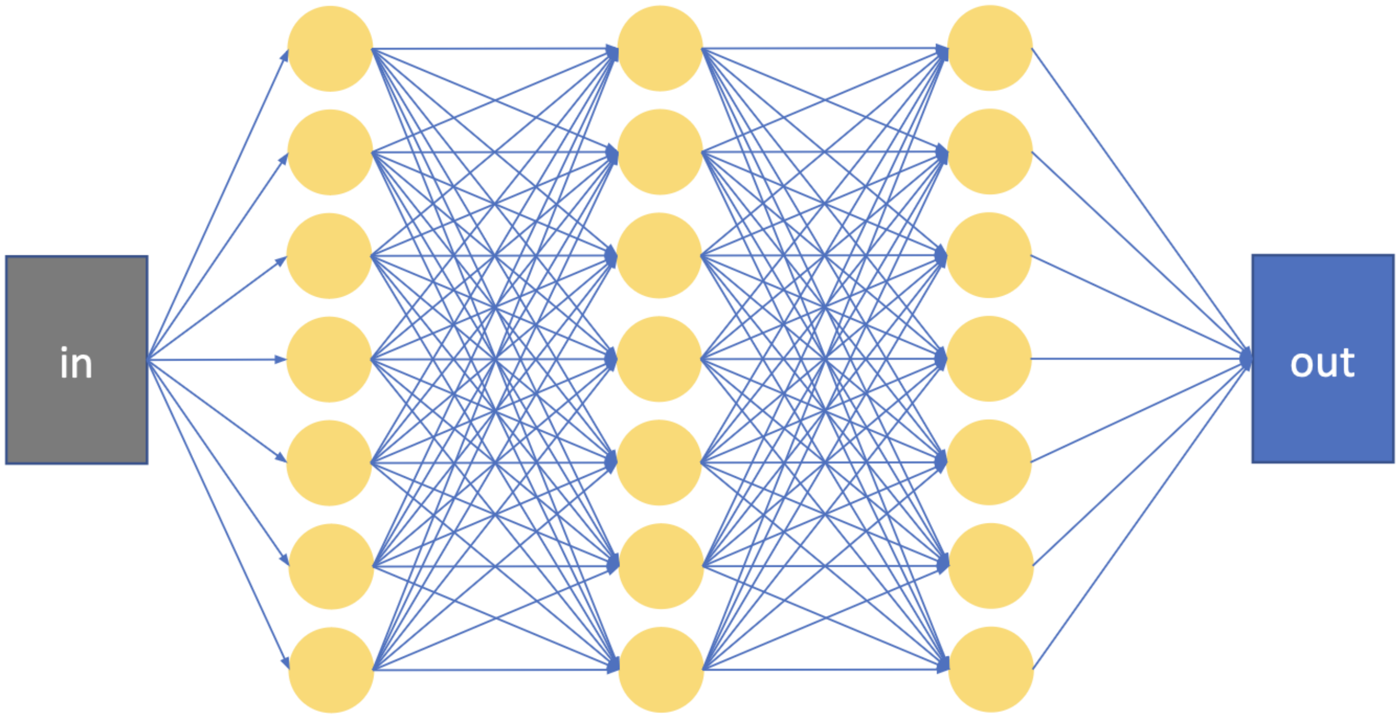
- Sort of looks like data augmentation, if you squint hard enough
  - Augmenting the data by randomly dropping out parts of it
- Over multiple passes through the net (i.e. during training over many epochs):
  - Randomly dropping neurons “forces” each neuron to learn a non-trivial weight
  - The network can’t learn to rely on spurious correlations (i.e. meaningless patterns), because they randomly might not be present

Dropout: Implications for test time



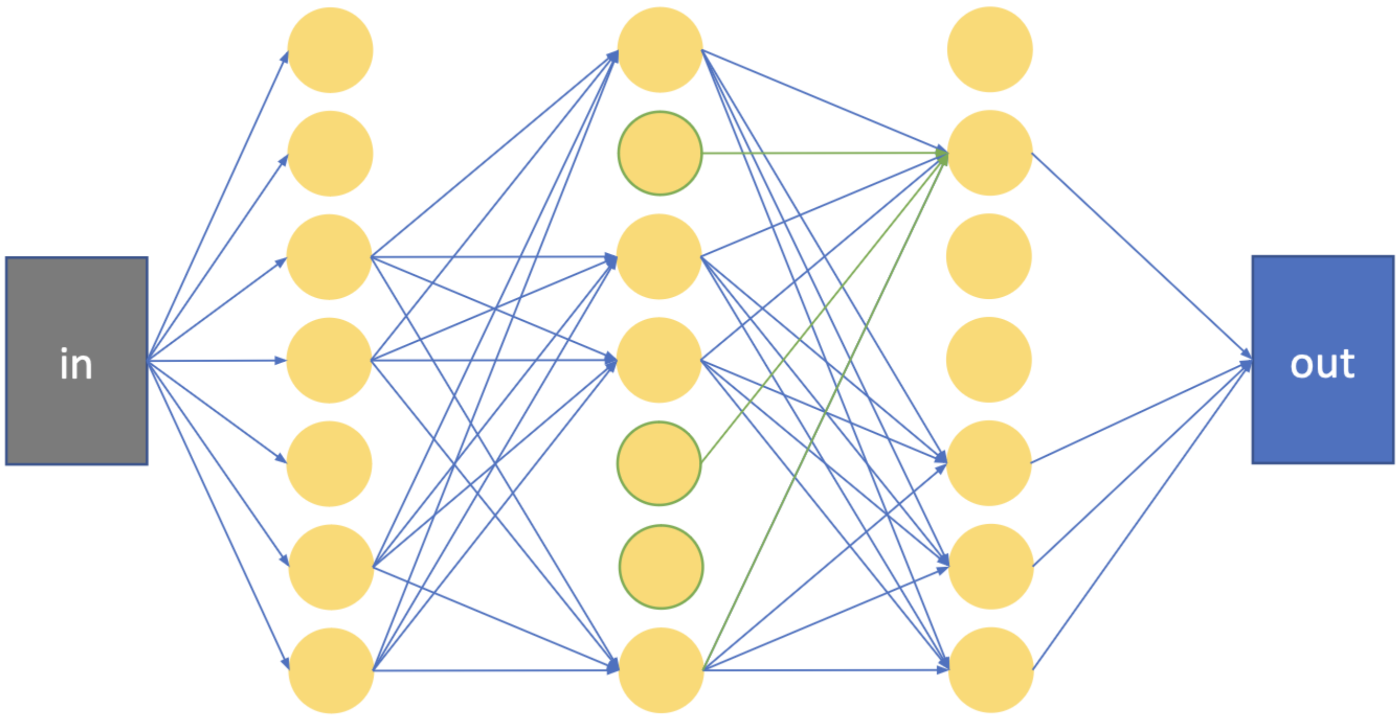
- During testing, we stop dropping out and use all of the neurons again

Dropout: Implications for test time



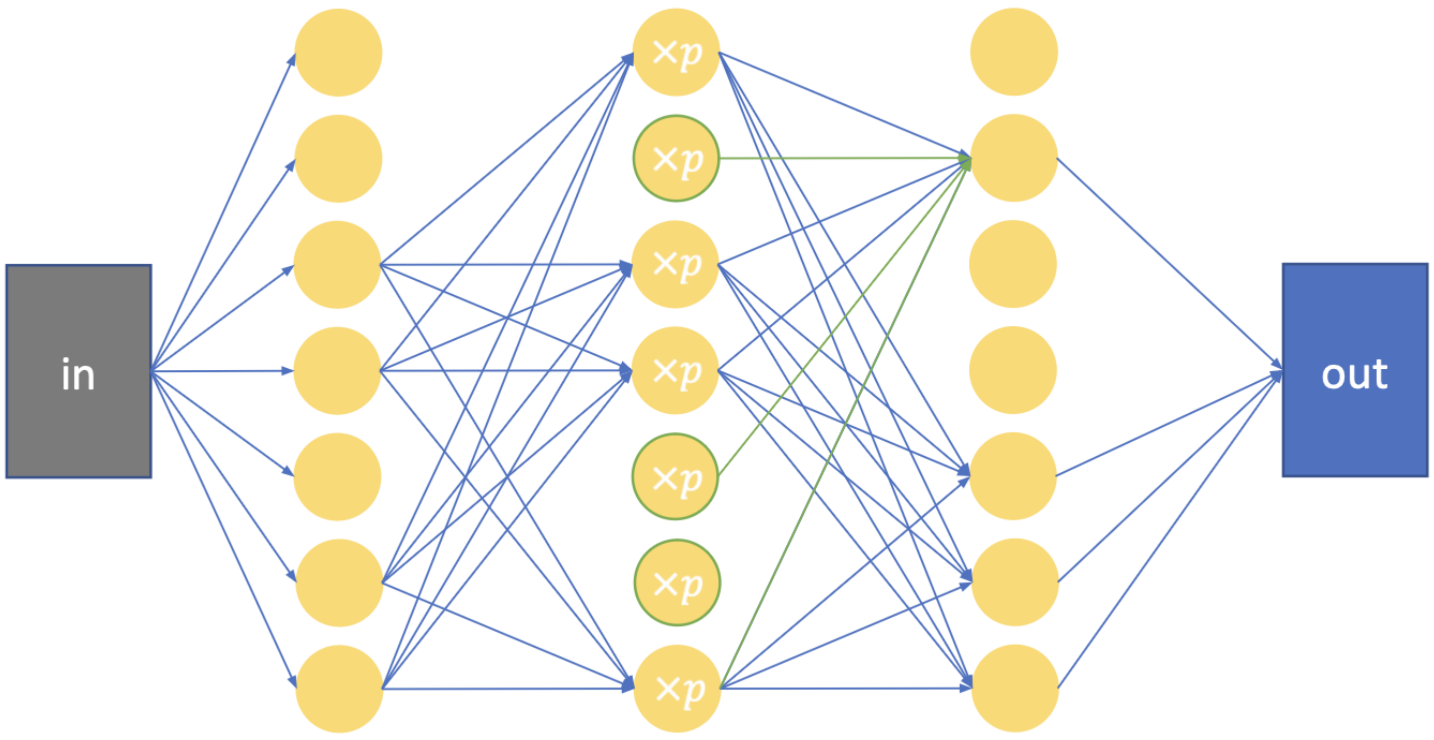
- During testing, we stop dropping out and use all of the neurons again

Dropout: Implications for test time



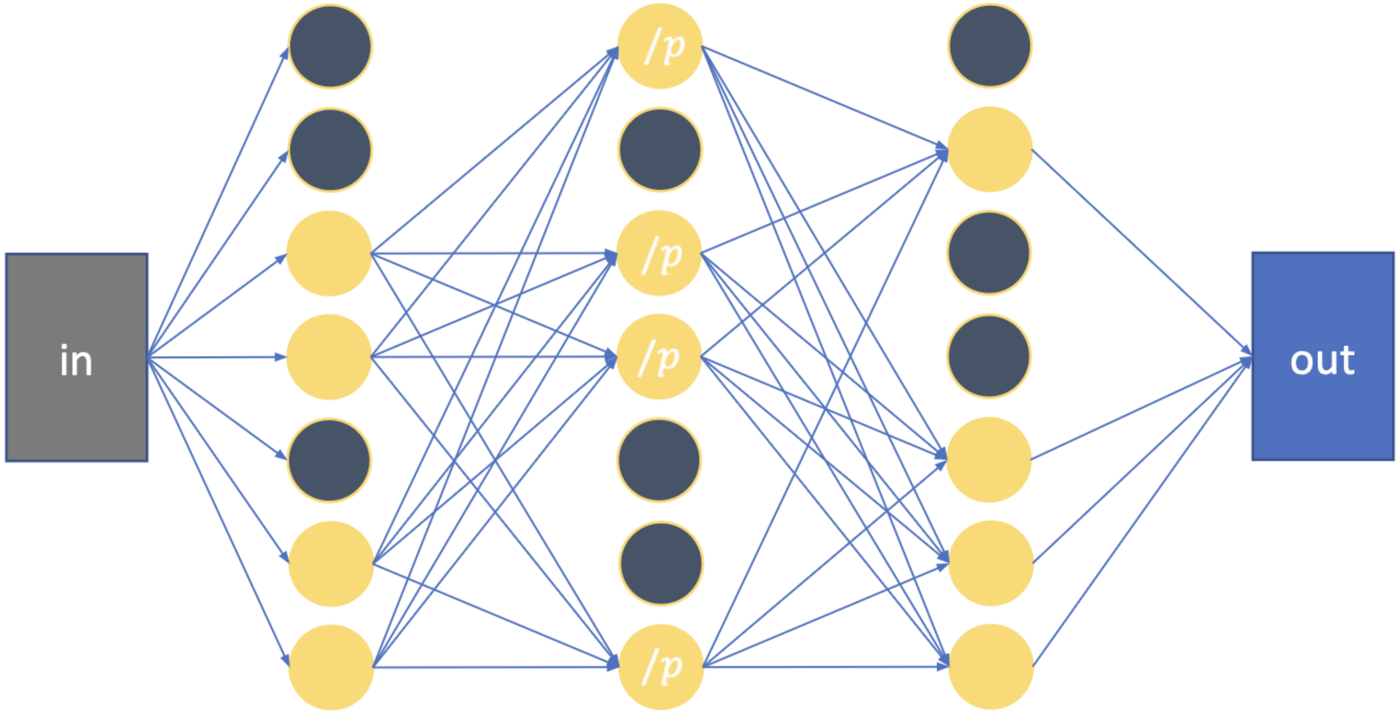
- During testing, we stop dropping out and use all of the neurons again
- If a layer keeps a fraction p of its neurons during training, then when we use all the neurons at test time, the next layer will get a bigger input than expected…
- ***What do we do!?***

Dropout: Implications for test time



- **Solution 1:**
  Multiply the values of all neurons by p, so that the expected magnitude of the sum of neurons is the same

Dropout: Implications for test time



- **Solution 1:**
  Multiply the values of all neurons by p, so that the expected magnitude of the sum of neurons is the same
- **Solution 2:**
  At training time, divide the values of the kept neurons by p

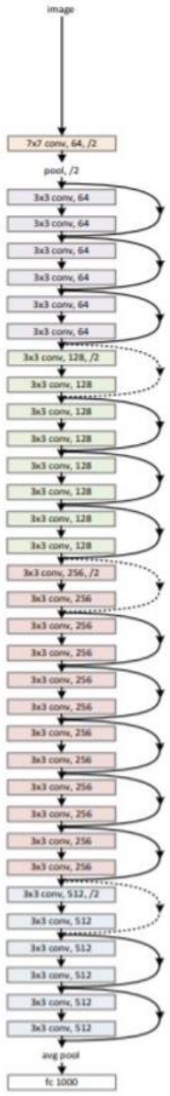
Dropout - implementation

Any questions?

- Handy keras layer!

- **`tf.keras.layers.Dropout(rate)`**
    - Hyperparameter `rate` between [0, 1]: the rate at which the outputs of the previous layer are dropped
    - **`Rate = 0.5`**: drop half, keep half
    - **`Rate = 0.25`**: drop ¼, keep ¾

Recap



Residual blocks prevent vanishing gradients

BatchNorm helps to stabilize training as networks get deep

Regularization is a somewhat automated way of preventing overfitting