- Goal of Pruning: Remove neurons that aren't as involved in producing results
    - Simple solution: Remove the connections of neurons that aren't being activated. This results in sparse matrix multiplications.
- Miscellaneous Links
    - Song Han Guest Lecture at Stanford: https://www.youtube.com/watch?v=eZdOkDtYMoo
    - https://songhan.mit.edu/
    - Tensorflow Pruning: https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras
    - L1 Ridge Regression
        - https://www.youtube.com/watch?v=Q81RR3yKn30
        - Reduces variance of the model (sensitivity to input values)
        - Does so by adding a term to the loss function
        - The term is lambda*(sum of square of each weight)
        - The effect is that weights tend to 0, meaning that there is a reduced variance of prediction with respect to input values
    - L2 Lasso Regression
        - https://www.youtube.com/watch?v=NGf0voTMlcs
        - Similar applications as ridge regression
        - Does so by adding a term to the loss function, just like ridge regression
        - The term is lambda*(sum of absolute value of weights)
        - Difference is that L2 reduces weights closer to 0 by an equal magnitude, while L1 reduces weights asymptotically close to 0
        - Useful in high-dimensional datasets where only a fraction of the parameters are useful
- Ideas
    - RL for predicting which connections should be pruned, which should be kept
    - Dynamically removing and adding nodes to the computational graph
    - How does parameter initialization influence convergence and training?
        - Can we test out different initialization approaches and see which works?
    - Different methods of pruning (e.g. probabilistic pruning)
        - Paper *Learning both Weights and Connections for Efficient Neural Networks*, for example, tested different pruning types
            - "We also experimented with probabilistically pruning parameters based on their absolute value, but this gave worse results."
    - Random thought- can a conv layer be equivalentlyrepresented by series of two dense layers?
    - Having pruning work at intervals that decay over time
        - Ex: Prune first every 1 batch, then every 2 batches then every 4 batches
    - Playing around with pruning schedule
    - Regularizer function that encourages parameters to go to integer values rather than just 0
    - Dataset-dependent model initialization
        - Dude you have no idea turns out it's an actual thing

        - https://arxiv.org/pdf/1511.06856.pdf
        - I wrote this idea down before hearing about it
    - Smart data augmentation
- Mark Kurtz, Pruning Deep Learning Models for Accuracy
    - Most neurons in neural networks are useless
        - Sparsity of model can be reduced 70-90%
    - Effect: Decreasing compute time, improving performance, decreasing memory
    - Structured vs unstructured
        - Structured - removing groups of weights & neurons
        - Unstructured - removing connections at random
    - Pruning has several hyperparameters
        - How often to prune
        - Which neurons to prune
    - Speaker presents automated solution for pruning that his team at NeuralMagic as developed
        - User interface to manipulate sparsity
        - API for Pytorch training that prunes every epoch
- Han, Learning both Weights and Connections for Efficient Neural Networks
    - https://arxiv.org/pdf/1506.02626.pdf
    - Takeaway: Baseline here is that this paper proposed the process of iterative training and pruning. Consequently, the model necessitates less compression, achieves smaller size, with little sacrifice in accuracy
    - How did they do it?
        - Essence of the paper: 3-step flow
            - Training, Pruning, Adjusting weights
            - Training provides insights into which weights to remove
- Han, Deep Compression
    - https://arxiv.org/pdf/1510.00149.pdf
    - https://www.youtube.com/watch?v=CrDRr2fxbsg&t=2s
    - Takeaway: Seems to be the pioneering paper for pruning of neural networks from back in 2015. Emphasis on model size reduction, as well as speed and energy consumption.
    - How did they do it?
        - 3 step procedure
            - Pruning weights with low connection magnitude
                - Make cluster of floating point weights with similar values
                - Approximate cluster of floating point weights with integer centroids
                - Fine tune centroids
            - Quantization of weights → weight sharing
                - Weight sharing implies a discrete set of weights → make lookup table
                - k clusters of weights (that share the same value) → $\log_2 k$ bits to encode index of each weight

        - Given n connections, all connections can be encoded in $\log_2 k$ * n connections
        - Given b bits to represent each connection, lookup table is kb bits
        - Without lookup table, the neural network requires n*b bits
    - Huffman encoding
        - Efficient form of encoding, where key (encoded symbol) length depends on frequency of value in message
        - I guess this is an optional addition to step 2, which further increases the compression rate of the network
- Learning the number of neurons in neural networks
    - https://arxiv.org/pdf/1611.06321.pdf
    - https://www.cs.ubc.ca/~schmidtm/MLRG/structured_sparsity.pdf
    - https://towardsdatascience.com/sparse-group-lasso-in-python-255e379ab892
    - https://www.youtube.com/watch?v=vgeDzIiXhWc
        - This pretty much explains it all
    - Takeaway: This paper proposes a method to cut down the number of neurons during training by means of sparsity regularization without the need for a preprocessing step
    - Questions
        - The authors mention that the groups are non-overlapping. Is this necessarily true, or is this just true in this case?
    - How did they do it?
        - They focus on learning the number of weights in a neural network
        - These guys just added a term to the loss function and are writing a paper
        - $L(\Theta) = L_0(\Theta) + r(\Theta)$
        - $r(\Theta)$ is a regularization term expressed as
            - ($\Sigma\lambda$ sqrt(P_l) for every layer l * ($\Sigma|\theta^n_l|^2$ for every input/output sample N))
            - P_l is a vector of parameters for every layer grouped together
        - Basically it just seems that they found a new regularization term that builds off the ideas of L1 and L2 regularization (notes at top of document)
        - L1 Ridge Regularization
            - r = lambda * (sum of squares of weights)
            - Benefit: Leads to small parameter values
        - L2 Lasso Regularization
            - r = lambda * (sum of absolute value of weights)
            - Benefit: Leads to sparse solutions (zeroed out weights)
        - Group Lasso
            - r = (sum of (lambda * sqrt(p_i) * sum of (every parameter in the group)^2) for every non-overlapping group group$_i$)
        - Sparse Group Lasso
            - Basically group lasso regularization term + L2 regularization term
            - Adding just one more term to group lasso

- Regularization in neural networks: A Taxonomy
  - https://arxiv.org/pdf/1710.10686.pdf
  - Reviewing and summarizing various types of regularization in neural networks
  - Regularization: A method that attempts to increase test dataset performance of a model
  - Types
    - Data
      - Adding noise to the parameters (e.g. Dropout)
      - Data augmentation
      - Changing representation of the input
    - Error function
- Learning to Prune Filters in Convolutional Neural Networks
  - https://arxiv.org/pdf/1801.07365.pdf
  - https://www.youtube.com/watch?v=3yOZxmlBG3Y
  - Takeaway: These guys found a way to prune the majority of filters with 0 loss in accuracy
    - Pruned 92% (!!) of filters with just 3% loss in accuracy
  - How did they do it?
    - They created a separate neural network (NN_{prune}) that predicted which filters to remove.
    - NN_{prune} did not remove individual connections, as they said this did not boost performance by much.
    - NN_{prune} was trained using a reinforcement learning approach where the weights were modified via a policy gradient method.
      - NN_{prune} could not be trained using gradient descent because the reward function was non-differentiable
  - Other notes
    - The paper provides nice references to related work on model pruning techniques
- He, AMC: AutoML for Model Compression
  - https://arxiv.org/pdf/1802.03494.pdf
- J. Frankle, The Lottery Ticket Hypothesis
  - https://arxiv.org/pdf/1803.03635.pdf
  - https://www.youtube.com/watch?v=s7DqRZVvRiQ
  - Takeaway: The researchers investigate whether they can train sparse pruned neural networks from scratch.
    - The typical end-product of the cycle of pruning (removing weights with new magnitude, retraining) is a sparse neural network
    - Training this sparse neural network from scratch results in a neural network with lower accuracy
  - How did they do it?
    - These authors decided to train a dense neural network partially, then prune it

        - They realize that retraining the sparse neural network with **the original initializations** results in a network that can train well
        - Retraining the pruned neural network from scratch (with same initializations) enables faster training, higher test accuracy
            - They call this sub-network the “lottery ticket”
        - But it’s also important to realize that initialization isn’t everything. Rearranging which connections were pruned does not result in final accuracy close to the original
- Han, Once for All
    - https://arxiv.org/pdf/1908.09791.pdf
    - https://www.youtube.com/watch?v=_jKlmMePY-w