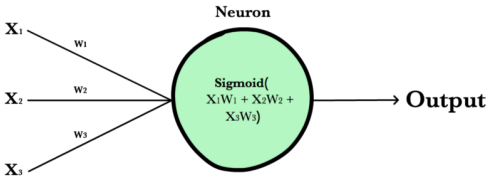
Cumulative, review what may have been on the midterm. Particularly trouble zones (according to Bethard, these are):

- [T/F] Empirically, deeper networks perform better than wider networks. (**T**)
- [T/F] With early stopping it is not possible to train on the combined training and development data.(**F**)
- [T/F] Mini-batch SGD with a stream of training data minimizes generalization error. (**T**)
- [T/F] Momentum can accelerate learning in the face of high curvature.(**T**)
- The questions where you had to decide the correct output unit (activation function)
  - Types of activation: linear (numeric), sigmoid (categorical, binary classification & multilabel), softmax - categorical, multiclass classification
- Why we don't worry about a sigmoid unit saturating when it's the output layer
- What a maxout unit looks like
  - max($W1^TX+b$, $W2^TX+b$,$W3^TX+b$, Wn^TX+b) is convex with =< 5 segments; outputs scalar value; piecewise linear.
- How many random samples when applying dropout

Mathematical form of a neuron. Be able to describe it



- The neuron is a set of inputs, weights and activation function.
- f(WX + b)
  - f(...) = activation function
  - W = weights
  - X = inputs
  - b = bias

Give a description of a problem, what's the most appropriate architecture/output layer for that problem

- Activation functions for output layer
  - linear - numeric data (*maybe* ReLU, if there's never a case (based on the problem) where the output can be, or should be, negative?)
  - sigmoid - categorical, binary classification & multilabel
    - Used in cases where we might predict whether or not, for example, multiple tags might apply to some text, and there *can be more than one* (sequence of binary classifications - tag or not tag?)
  - softmax - categorical, multiclass classification
    - Using the same example from above, say we have a sequence of tags but there *can only be one* chosen to apply to the text (i.e., text ultimately belongs to a single class) (also, in the email example from the midterm - 'spam or ham', either it's spam or not spam - can't be both)
- Architecture

How might you deal with saturation

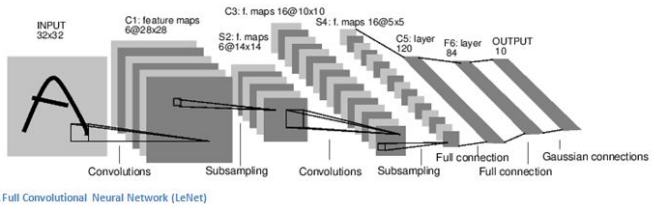
- Change weights/activation. Reference group activity answer below as well for backprop(?) specific example.

- relu units (or leaky relu) because they eliminate the gradient from becoming 0, batch normalization would also work.

Differences between optimizers. RMS Prop/SGD.. What do they change (descriptively)

- (perhaps answered in below notes)
- Optimizer takes the loss function and tries to minimize it by changing the weight values.
  GD (Gradient Descent) - calculate gradient for the whole dataset and updates values in direction opposite to the gradients until we find a local minima.
  SGD (Stochastic Gradient Descent) - performs a parameter update for each training example unlike normal Gradient Descent which performs only one update
  Momentum - Momentum accumulates exponentially decaying moving average of past gradients and continues to move in their direction
  Adagrad - is more preferable for a sparse data set as it makes big updates for infrequent parameters and small updates for frequent parameters. It uses a different learning Rate for every parameter θ at a time step based on the past gradients which were computed for that parameter.
  RMSProp - RMSProp modifies AdaGrad by changing the gradient accumulation into an exponentially weighted moving average, i.e. it discards history from the distant past
  Adam - can be seen as a variant on the combination of RMSProp and momentum, the update looks like RMSProp except that a smooth version of the gradient is used instead of the raw stochastic gradient, the full Adam update also includes a bias correction mechanism

How convolutions look/work, what do they look like when applied to a grid, etc.

- Creates activation map of params/weights by convolving across an input
- 

  A Full Convolutional Neural Network (LeNet)

Different choices for sequence to sequence model. How to know when to stop.

- RNN (LSTM, GRU). CNN (Conv1d) can also be used to help with accuracy, but I think a RNN has to be used during some point of the process.
- Each input stops when it reaches the end of input (EOL for text for instance).
- Training stops as it always does when you reach a comfortable level of accuracy.

How to take a simple rnn and make it more robust for long term dependencies (leaky, GRUs, etc)

- leaky relu rather than relu can help reduce saturation because leaky relu have a small gradient rather than 0.
  Both LSTM and GRU should both be more accurate than a simpleRNN. GRUs are even more simple than LSTMs, they run faster because they only have 2 gates (no forget gate) and they are easier to work with because it fully exposes the memory which a LSTM does not.

Below information begins with unit 8.4

Topics:

- **Parameter Initialization Strategies**
  - In neural network optimization, initial point determines: whether learning converges at all, how quickly learning converges, whether it converges to a point with high or low cost, whether the point that it converges to has high or low generalization error
  - Initial parameters must break symmetry
    - Most common solution: random parameter initialization.

            - biases typically not randomly initialized, weights drawn from Gaussian or uniform distribution
    - Larger initial weights can result in: better symmetry-breaking, less signal lost during forward/back-prop. Downsides: exploding values during forward/back prop, extreme sensitivity to small perturbations of the input, extreme values where the activation function saturates, parameters further from origin
    - Weights can be initialized via training a simpler model. Initialize a supervised model via training an unsupervised model on the same inputs or another supervised model on a related task
- **Algorithms with adaptive learning rates**
    - **AdaGrad**: +not very sensitive to initial learning rate, +large partial derivs -> large learning rate decrease, +small partial derivs -> small learning rate decrease, -aggressive, monotonically decreasing learning rate
    - **RMSProp**: +discards history from the extreme past, +less aggressive than adagrad, -has 1 extra hyperparameter p
    - **ADAM**: +incorporates momentum, +less biased than RMSProp, -has 2 extra hyperparams p1 and p2
    - Selecting a learning algorithm:
        - Tom Schaul, Ioannis Antonoglou, and David Silver. Unit tests for stochastic optimization. ICLR 2014.
            - Tries algorithms on simple cost function shapes
            - Adaptive learning rates robust, but no clear winner
        - Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. NIPS 2017
            - Tries algorithms on image and language tasks
            - AdaGrad, RMSProp, and Adam all have worse generalization error than SGD
- **Batch normalization**
    - If y=1 and B=0, then the activations will be small (mean 0, std 1)
    - Back prop prevents gradient from proposing changes to H that increase std or mean
    - Before, mean and std were determined by complicated interaction between layers. Now y determines std and B mean.
    - Requires a batch to calculate. Run averages to allow predictions for a single item. Apply batch norm after the non-linearity
    - Layers dont need a bias if followed by batch norm.
    - When batch norm is used, adding dropout as well typically does not help and may hurt
- **The Convolution Operation**
    - Convolutional layer properties
        - Sparse interactions, parameter sharing (tied weights), equivariance to translation (shifting an object in the input will also shift its output representation)
    - The receptive field of a unit (which other units affect it) increases with more convolutional layers. (Think the grid projection.. 1x1->3x3->5x5)
- **Pooling**
    - Pooling with a stride n only takes every n^th region
- **Convolution and pooling as an infinitely strong prior**
    - Convolution: weights of unit i=weights of unit i+1 but shifted, weights=0 except for small contiguous region, only local interactions, equivariant to transition
    - Pooling: invariant to small translations.
    - Strong priors -> potential to underfit (if inappropriate)
- **Computational graphs**

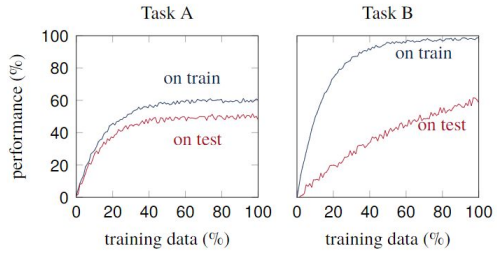
- Feedforward network vs variable length
  - Cant model word order, each unit has to separately learn the same semantics
- Convolutional network vs long distance
  - Can work if deep or wide enough, but its not a natural fit for the word problem
- Recurrent nn properties
  - Handles variable length sequences, shares the same params at each step, incorporates all preceding inputs and hidden states into each decision

- **Recurrent neural networks**
  - 1 output per input
    - Can compute any function that is computable by a Turing machine, training time and memory are O(T) and not parallelizable since all states are needed for backprop, testing time is O(T) and not parallelizable, testing memory is O(1); past states can be discarded, powerful but expensive
  - Output-to-hidden
    - Cannot simulate a universal Turing machine, training is parallelizable with **teacher forcing** ie using the ground truth output, since then each time step can be calculated independently, testing time is O(T) and not parallelizable, testing memory is O(1); past states can be discarded, may predict poorly when ground truth objects are no longer available (solution: mix training on ground truth outputs with training on predicted outputs)
  - 1 output at end
    - Training time and memory are O(T) and not parallelizable since all states are needed for backprop, testing time is O(T) and not parallelizable, testing memory is O(1); past states can be discarded, can be hard to train on long sequences since output gradient must propagate back through the whole sequence
  - Back prop through time
    - Unroll RNN to produce computational graph including all time steps, perform regular back-prop on the graph
  - Graphical models: edge=dependence != operation
  - RNNs efficiently parameterize the joint probability distribution
  - RNNs as graphical model challenges
    - Through parameterization of joint prob dist is efficient, optimizing those params may be difficult
    - Assumes relationship between adjacent time steps, does not depend on *t*
    - Generating samples from an RNN requires extra machinery
      - Can add special stop output symbol, can add binary stop/dont stop output to each step, can add integer (T) output to each step
- **Bidirectional RNNs**
  - Problem: recurrent neural networks assume that a state depends only on previous states. But:
    - pronunciation is influenced by both previous and following sounds (e.g.,/n/intenvs.tenth).
    - a sentence may have to be re-interpreted once laterwords arrive (e.g.,The horse raced past the barn fell.)
    - Solution: run one RNN forward, one RNN backward, and combine the two
  - Merging options: Dont merge, sum, concatenate. Concat is most common (and default in Keras)
  - Bidirectional recurrent nn:
    - Accurately capture local info both before and after current state (similar to convolutional nn)
    - Can capture some long-range info, both before and after current state

        - Are expensive to train (two RNNs)
- **Encoder-Decoder Sequence-to-Sequence Architectures**
    - May help reverse the input, some RNN layers may need to be bidirectional, stacked RNNs may need residual connections
- **Recursive neural networks**
    - Tree structure instead of chain structure
    - When comparing syntactic trees, to find similar sentences you apply recursive NN and take cosines of S vectors
- **Challenge of long-term dependencies**
    - Composing many nonlinear functs: more nonlinear-> harder to optimize with gradient descent
    - If W is different at each time step no problem. If activation at t is too large/small, adjust only W^t
    - Long term dependencies are the problem
    - Gradient of long-term interaction exponentially smaller than gradient of short-term interaction
    - Learning long-term interactions will be slow
    - Steps for long-term interactions will be masked by small fluctuations in steps for short-term interactions
- **Leaky units and other strategies for multiple time scales**
    - A Near 0 -> past is rapidly discarded. A Near 1 -> long term history dominates
    - As compared to skip connections
        - Amount of history is real-valued not int valued number of skips
        - A can be set by hand (as in skip connections) or learned during training
- **long/short term memory and other gated RNNs**
    - GRUs generally perform as well or better than LSTMs with fewer params
- **Performance metrics**
    - accuracy=(items predicted correctly/items), coverage=(items where model made any prediction/items), precision=(items of interest predicted correctly/items of interest predicted), recall=(items of interest predicted correctly, items of interest), F1=(2*precision*recall)/(precision+recall)
- **Default baseline models**
    - Model category
        - Fixed size input vectors ->
            - simple linear interactions -> logistic regression
            - Nonlinear interactions -> feedforward network
        - Multi-dimensional input -> convolutional network
        - Sequential input -> gated recurrent network
    - Model optimizer
        - SGD + momentum + learning rate decay or ADAM
    - Model regularization
        - Early stopping
        - |data|<10,000,000-> dropout
- **Determining whether to gather more data**
    - Performance is low?
    - Check training set performance, if poor:
        - Try to improve model by adding more layers or widen current layers, tune hyperparameters (eg learning rate)
        - Check training data. Are features rich enough/is there a lot of noise?
    - If training performance good; test performance poor
        - Gather more data. How much do you need/can you afford?

        - Try to simplify model. Reduce layers/layer width, add regularization (eg dropout)
    - If both are good, you're done
- **Selecting hyperparameters**
    - Goal: lowest generalization error, subject to runtime/memory budget
    - Could be constrained by model arch, success of optimizer, degree of regularization
    - Increase (+) capacity by: increasing num of hidden units, + convolutional kernel width, - L1 or L2 regularization coeff, -dropout rate, optimally tuning learning rate
    - Automatic hyperparam optim:
        - Grid search: small finite set for each hp, train and evaluate model for each combo, **cost grows exponentially with # of params**
        - Random search: sampling dist for each hp, train and eval model for sampled combos, **always trains a fixed # of models**
        - Bayesian optimization: build mod to predict validation error from hp, explore hp where model is uncertain, **can be unreliable; worse or better than manual tuning**
- **Debugging strategies**
    - Many tests, unit tests for code correctness, plots of optimization progress, analysis of model predictions
    - Visualize model in action: manually examine alongside pred output, display images (input) with detected objs (output), display reviews (input) with predicted ratings (output). Can reveal bugs in eval code, patterns in model errors
    - Examine preds that are furthest from true values, eg where sigmoid output is .99 but true label is 0.. Where true label is smallest val in softmax output. Can reveal problems in preprocessing, errors in annotation, missing features
    - Low training error, high test error: training procedure works, model is overfitting OR bug in testing
    - High training error, high test error: training procedure works + model is underfitting OR bug in training procedure
    - Fit/train on a small fraction of your data eg 1-100 instances. Can reveal bugs in training code (0% training error on 1 example mods), insufficient model capacity
    - Monitor activation/gradient histograms. Plot histograms of activations and gradients over each epoch of training. Can reveal: units that are unused/always off, gradients that are too small to make progress
- **Multi digit number recognition**
    - **Street View transcription**
        1. Data collection
            1. Cars take photographs
            2. Separate machine learning model detects/crops to signs
            3. Humans annotate signs with their numbers
        2. Performance metrics
            - 98% accuracy (required), 95% coverage (desired)
        3. Baseline model
            - Convolutional network, ReLU activation, *n* softmax outputs
        4. Model enhancements
            1. Need to optimize coverage ⇒ new output layer and cost
            2. Training error ≈ test error ⇒ underfitting or bug
            3. Visualize worst errors ⇒ input cropped too tightly
            4. Simple heuristic to expand crop ⇒ +10% coverage
            5. Increase model size, keeping training error ≈ test error ⇒ +last few % coverage
    -

List of answered group activities:

- P(a & b) = P(a)P(b)
  - =f(a)g(b)
- P(c) = sum of f(a)g(b)
  - a + b = c, so…
  - =sum(over a) of f(a)g(c-a)
  - =(f * g)(t) (my notes say t, but I'm not sure where that comes from?)
- Where is the greatest risk of saturation during back-propagation? What could you do to fix it? Would there be any side-effects of your fix?
  - Greatest risk was in tanh
    - @ output layer softmax saturation not as concerning, but it is taken care of by cross-entropy.
  - Replacing tanh with ReLU works if you initialize your recurrent weights to the identity matrix
  - Side effects - as you back propagate, may get closer and closer to 0.
- Both LSTMs and GRUs try to encode the intuition that we need a way to decide, at each time step, whether to keep memory around or discard it for new input. But they encode this intuition in different architectural choices. How you could treat these choices as hyper-parameters? That is,what architectural choices do we need to make, and how would you search through this space?
  - Empirical Exploration of Recurrent Network Architectures
    - LSTM<GRU; discovered architectures=GRU
    - LSTM<LSTM with forget bias initialized to 1=GRU
- You are designing a neural network to help cancer registrars, who read hospital notes and find all patients that have been diagnosed with cancer. Your goal is not to replace the registrars, but to speed up their work. Which evaluation metric(s) would be most important in this task?What level(s) of performance would you aim for?
  - One possible answer: Focus on precision and coverage, setting a threshold for when we want to make a prediction (only when model has some reasonably high level of confidence/certainty). Go for, say, 95% coverage, maybe? Precision might depend on how precise the registrars are (want to do as well as the registrars or better)
- Recursive neural networks are typically applied to natural language problems. What would it mean to apply a recursive NN to a task like stock price prediction or image classification? Are there tasks outside of language processing where a recursive NN would be a good fit for the problem?
  - Not really a good fit for tasks outside of language processing?
  - What would it mean to apply it to those tasks
- 

  - The plots above show learning curves for two different tasks. For each of the tasks, decide if anything is wrong with the model, and if so, discuss what steps you would take.
    - Task A: underfitting - means the model isn't complex enough, so increase the size of the model (width, depth, etc.)
    - Task B: Doubling data would help us to see if we can continue to increase test performance, given training data performance is good.

- Should think about how costly it would be to get more data. If you can't get more data, try to optimize model other ways (ex. regularization)
- Could try increasing learning rate

Daily class quiz questions:

1. With complete certainty, we know that initial parameters in a neural network must:
    a. **Break symmetry between different units**
2. Which of the following optimization algorithms takes velocity into account when taking a gradient descent step?
    a. **Adam**
3. Batch normalization makes optimization easier by:
    a. **Reducing the dependence of parameters between layers**
4. The receptive field of an output unit in a convolutional neural network
    a. **Covers more input if the convolution window is wider**
5. A pooling layer **cannot**:
    a. **Have its weights updated during training**
6. A recurrent neural network's solution to problem of variable-length sequences is to:
    a. **Define the current hidden state as a function of the previous hidden state and the current input**
7. Which of the following is **required** to allow a recurrent neural network (RNN) to compute any function that is computable by a Turing machine?
    a. **There must be hidden-to-hidden recurrent connections**
8. Which of the following tasks would be handled well by the typical RNN formulation, where there is exactly one output per input, and hidden-to-hidden recurrent connections?
    a. **Detecting seizures in EEG signals**
        → Is there a seizure or is there not a seizure happening *now*?
9. Which of the following would be the most appropriate place to use a bidirectional RNN?
    a. **Classifying Youtube comments as spam or not**
10. Which of the following problems would be the best fit for a sequence-to-sequence model?
    a. **Summarizing text**
11. The biggest problem for learning long-term dependencies in RNNs is:
    a. **Smaller weights are given to long-term interactions**
12. Gated RNNs are better than leaky units because:
    a. **They can choose when to forget information**
13. The first neural network model you train for a supervised learning task should always include:
    a. **Early stopping**
        → Should almost universally be used, need to figure out how long your model should train.
14. If you have time to tune only one hyperparameter, tune the:
    a. **Learning rate**
15. The code for your performance metric was written by someone else. Which debugging strategy is most likely to reveal the bug in this code if there is one?
    a. **Visualize the model in action**