CSE 151B Cheatsheet

Brandon Szeto

PID: A17002478

Perceptrons

Learning Rule:
$w_{ij} = w_{ij} + \alpha\delta_j z_i$ where $z_i = g(a_i)$
In our examples (SSE, Cross Entropy), we have
$\delta_j = (t_j - y_j)$ if j is an output unit and
$\delta_j = g'(a_j)\sum_k \delta_k w_{jk}$ if j is a hidden unit.
Backpropagation is a linear function with respect to δ not the weights.

Generalization

Best way is to get more data.
Regularization: Minimizing $J = E + \lambda C$ where C is a measure of complexity.
L_1 minimizes $|W|$
L_2 minimizes $\|W\|_2^2$
Dropout: Randomly turning off hidden units during learning.
Early Stopping: After waiting for the error to rise after some number of epochs, stop training.

Techniques

Batch Gradient Descent: Running average of gradients across the whole dataset is used to update the weights. Weights updated when all examples in a batch are seen.
Stochastic Gradient Descent: Picks a random gradient at each iteration. The examples are shuffled so the actual gradients you follow are in random order. Helps reduce computations. A mini-batch is commonly used as a compromise between BGD and SGD.
Shuffling: Helps minibatch contain most of the categories, infrequent examples, and patterns different classes, but be careful of errors in data as more has to be learned about easy examples before hard ones can be tackled.
Principle Component Analysis: Decorrelates (makes orthogonal) inputs and shifts the mean of the input variables to be zero (not all positive or negative). Each component captures the maximum available variance in the data. Common to also normalize by dividing by the standard deviation.
Z Scoring: Shifts mean of the input variables to zero, but does not decorrelate the inputs. Cannot throw away dimensions (no eigenvectors computed). Divides by standard deviation by default.
Weight Initialization: Should all be different or else all δ will be the same. Idea: put gradients in range of activation function to maximize gradient and coordinate with input normalization.
Batch Normalization: Normalizes all of the inputs in the network including to each hidden layer on a per unit basis over each minibatch. For each minibatch, z-scores each variable and gives the network the chance to undo batch normalization by giving it adaptive parameters (undoing is learnable).
Adaptive Learning Rates: Dynamically adjust learning rate based on observed gradients. Individual learning rates for each weight. Rprop increases or decreases the learning rate by 1.2 or 0.5 times depending on the sign of the gradient. Rmsprop uses a moving average of the squared gradient for each weight.
Momentum: Speed up learning rate by keeping a running average of weight changes. Change these learning rates for each parameter. Weight change is calculated as a function of its current gradient and some decay γ and its previous weight.

Convnets

Main Idea: Multiple filters are run over an image to learn different features (e.g. weights are shared). Example:
$224 \times 224 \times 3 * (5 \times 5 \times 3) \rightarrow 217 \times 217 \times 1$

$$W_{\text{out}} = \frac{W_{\text{in}} - F - 2P}{S} + 1$$

Properties
Locality: nearby pixels correlate the most with nearby pixels, not pixels far away
Stationary Statistics: the statistics of pixels are relatively uniform across the image
Translation Invariance: identity of an object doesn't depend on its location in the image. This is a result of conv + pool layers. A conv layer detects an object and pooling layers preserves the most prominent features in different regions and discards the rest, making it less sensitive to shifts.
Compositionality: objects are made of parts
Receptive fields get larger deeper into the network.
Pool Layer Reduces spatial size and number of parameters to prevent overfitting. Increases receptive field.

Objective Functions

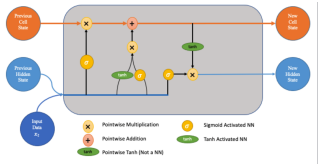
Goal: Find the most likely weights given the data. Using Baye's Theorem, figure out how to maximize the likelihood of the data given the weights. Must figure out the pdf of the data and maximize that.
When doing regerssion, if we assume the targets are gaussian distributed and we maximize the likelihood of the data by minimizing the negative log likelihood of the data, we end up with SSE.
When doing logistic and multinomial regression, if we assume the targets assume a binomial/multinomial distribution and maximize the likelihood of the data by minimizing the negative log likelihood of the data, we end up with cross entropy error.
Siamese Neural Networks: Two sets of the same weights are placed together. Given two inputs each with a corresponding target. Trained to move outputs closer or farther apart. If inputs are the same, closer, if inptus are different, farther.

Recurrent Networks

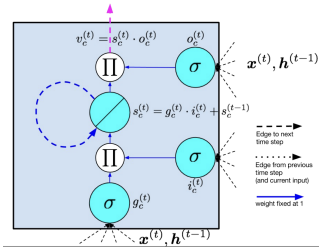
Map time into state.
Elman's network:

- Next word prediction (29 word bank)
- backprop only one step back in time

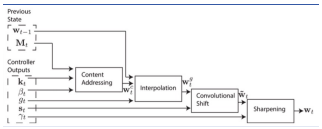
RNNs have the vanishing/exploding gradient problem. In backprop, grad is multiplied by weights at each layer, so if the weights are < 1, the gradient may vanish and long term dependencies can't be learned. Otherwise, it may explode, and wont converge.



Forget, input, and output gates (left to right).



Neural Turing Machines



Can explicitly teach a neural net to program. However, the network must learn how to remember things in its internal state space.
NTMs add structured memory to a neural controller to which it can read/write. **Controller:** can be feed-forward or recurrent (LSTM units). Recurrent ones typically work better. **Read** and **write** heads are content and location addressable. Below r_t is the read vector.

$$r_t \leftarrow \sum_i w_t(i) M_t(i)$$

$$M = \begin{bmatrix} 3 & 1 & 4 & 1 \\ 5 & 9 & 9 & 7 \\ 2 & 7 & 2 & 8 \end{bmatrix}$$

$$w = (0, 1, 0)^T, r_t = (5, 9, 9, 7)$$

$$w = (1/2, 1/2, 0)^T, r_t = (4, 5, 6.5, 4)$$

Write operations happen in two steps: erase then add. Here e_t is the erase vector and i is the location.

$$M_t(i) \leftarrow M_{t-1}(i)[1 - w_t(i)e_t]$$

$$w = (0, 1, 0)^T, e_t = (1, 1, 1, 1)$$

$$M = \begin{bmatrix} 3 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 7 & 2 & 8 \end{bmatrix}$$

Must be done in wo steps because there is no "overwrite" in arithmetic.
Content-based Addressing: Takes cosine similarity of key vector and memory element.
Location-based Addressing: Takes location-based addresses and offsets can be used to manipulate specific locations in memory.
NTMs learn to use memory because it is end-to-end differentiable.
NTM with LSTM controller performs best.

GAN

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p(data)} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output For real data}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output For fake data}}) \right]$$

This has bad training properties so we use this instead

$$\max \log D(G(z))$$

- The Discriminator wants to maximize this objective so that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- The Generator wants to minimize this objective such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

CycleGan
Translates from one domain to another without pairing data. Uses "cycle consistency loss": if X translates to Y, then Y should translate to X
$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]$

Rezero

Trains networks faster with a special residual connection

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i F[\mathcal{W}_i](\boldsymbol{x}_i)$$

If $\alpha = 1$, network sensitive to small preturbations of input. If $\alpha = 0$, network returns input.
"self-attention and layernorm does not satisfy dynamical isometry", so they made rezero transformers. Showed that log-eigenvalues were closer to 1 (desirable) than regular transformers

Rein. Learning

TD learning makes nearby (in time) states have nearby values. The TD λ term controls how far back in time error feeds back from 0 to 1.

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

Alpha go:
Had a policy and value network. Policy helps select paths in MCTS, value determines $P(win)$ given current configuration.
Alpha go ZERO was not trained with any human data.
Policy: mapping from states to action $\pi_t(s, a)$ = probability that $a_t = a$ when $s_t = s$

RAMNet

Recurrent Attention Models iteratively attends to different parts of the input sequence over multiple time steps, adjusting its focus based on the task at hand.

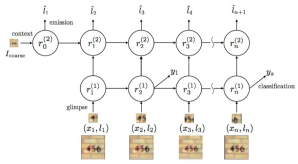


Figure 1: The deep recurrent attention model.

We compare y to target and backprop. Stop the gradient after the first mislabeled target. Choose random locations at first, and reward it for picking locations that work well.

Transformers

Attention based methods (transformers) compute an attention vector dynamically, which is then used to gate the previous hidden states. Then during decoding, the decoder can choose to pay attention to any of the states of the encoder. Feed-forward. Replicate the network for every input. Fast to train. Can be used for images (encoder only). Takes in patches of an image without convolutions.
Reccurence-based methods have to wait since hidden states depend on all previous hidden states (i.e. can't take advantage of parallelism).

Reinforcement Learning

An agent that acts in some environment. Recieves rewards, good or bad, to tell it if something happened. Feedback is typically delayed. The agent wants to maximize its long-term expected reward.
Agents learn via a policy, $\pi(s) = P(a_1 \ldots a_n | s)$, which is a function that takes the state of the environment and output action probabilities. This function can be implemented as a deep network that maps states (i.e. images of a game) to a softmax over actions. Atari used a deep network to learn games.
Policy Gradient: 1) Sample from softmax distribution. 2) Treat the sample as a teacher for that state/action pair. 3) Compute the weight changes, add them to a running average. 4) Multiply the weight changes by the sign of the reward. 5) Change the weights after a game.
Maximizing long-term reward:
$r_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ where $0 \leq \gamma \leq 1$ is the discount rate.
Markov Property: The future behavior of a process depends only on its present state, not on the sequence of events that preceded it.
Model-based RL: The agent either has avaialable to it, or it learns, a model of its environment.
Value function: The expected long-term return of being in a state, following π. If you have values of states, and a model of the world, you can decide which actions to take by $\pi(s) \max_a \sum_{s'} P(s, a, s')V(s')$. Epsilon-greedy introduces a probability, ϵ, that we pick a random action instead of the policy.
Q-Learning: 1) Use epsilon-greedy to choose an action: with probability ϵ choose at random, otherwise use policy. 2) Take action a, observe the reward r and the new state s'. 3) Update the Q-value for this state-action pair: $Q(s, a) = Q(s, a) + \alpha(r + \gamma * \max_{a'} Q(s', a') - Q(s, a))$.
4) If s' is a final state, get an actual for the state. Idea is to minimize the difference between where you are now and where you can get to from this state.
Minimax: Produces perfect play for deterministic, perfect-information games. Choose action with highest minimax value (i.e. search the tree choosing the max for agent 1 and choosing the min for agent 2). This branching is $O(b^m)$ where b is the number of possible branches at a given state and m is the number of games.
TD-Gammon: Example of learning a value function using a neural network approximation to a real game. The network takes the board state as input and learns teh value of the board position by playing games with itself. Weights are update using a temporal difference rule on every move where the current estimate of the board value is updated to be closer to the next board's value. $w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$
Monte Carlo Tree Search

Clickers

True:

A perceptron with no binary threshold unit is simply a linear regression model.

If we reverse the delta rule ($\delta = y - t$ instead of $\delta = t - y$), the perceptron will **never** converge to a minima.

The goal of linear regression is to find a set of weights $\mathbf{W}$ that map the design matrix $\mathbf{X}$ to targets $\mathbf{t}$.

For softmax regression, the weight matrix $\mathbf{W}$ has number of entries equal to the number of edges in the network graph.

Backpropagation allows a network to learn latent (abstract) representations of features and uses the chain rule to propagate gradients.

Convergence with a multi-layer, non-linear network indicates that the network has found a local minima along the error surface.

The locality property of the visual world is represented in convolutional neural networks by receptive fields that are only connected to a small patch of the image.

The stationary statistics property of the visual world is represented in convolutional neural networks by learned features are applied all over the image (convolution).

The translation invariance property of the visual world is represented in convolutional neural networks by learned features are applied all over the image (convolution) and max pooling.

Convolution followed by nonlinearity then sub-sampling is equivalent to convolution followed by sub-sampling then nonlinearity.

Processing each pixel of an image with a square convolutional kernel containing all 1s would result in a feature map brighter than the image and resembling a blurred version of the image.

Maximizing the likelihood also maximizes the log likelihood.

The posterior in Maximum Likelihood Estimation tells us the probability of the weights given the data.

Siamese networks can perform clustering, supervised learning, and dimensionality reduction.

A RNN maps time into state.

In RNNs order of inputs matter.

A NTM is essentially a neural network with a working memory.

False:

Gradient descent uses linear regression (however, linear regression can use gradient descent when psuedoinverse is computationally expensive).

A single layer perceptron is equivalent to logistic regression (a perceptron can only provide a binary outcome, whereas logistic regression provides a Bernoulli pdf)

Cross entropy is a better loss function than MSE for linear regression (it's better for logistic regression)

A network's activation function must be differentiable everywhere (ReLU)

Backpropagation is a nonlinear operation.

In stochastic gradient descent (SGD), we should shuffle our minibatch before computing its gradients (Shuffling now has no impact on the weight changes, you need to shuffle the data before partitioning it into mini-batches).

The more parameters in a CNN, the better it will perform at classifying images.

In RNNs, the output of a timestep t is the input to the same network at timestep $t + 1$.

To write to memory in a (NTM), you can overwrite the memory in 1 operation (erase then add).