HW #1 , Solving Output of Neural Networks

Part #1

For each neuron,

Part #2

Start w/ known formulas

For earlier neurons,

Part #3

Use softmax to convert to probabilities :

Note ground truth class, Class 1 is [1,0], Class 2 is [0, 1]

Compute gradients using Cross Entropy Loss which we have :

Top-5 Error = (# of times the true label is not in top-5 predictions) / total samples

Classification – Predicts discrete categories (binary)

Regression – Predicts continuous outputs (any value)

P ( y | x ) is discriminative because it only focused on predicting the label(y) given some input (x), don’t care how data was generated just how to draw a decision boundary between classes.

Discriminative = classification focused, Generative = modeling how data was generated

Data collection helps extract features that actually matter within the problem space turning random values into set features for model.

L2 (Ridge) (Mean Squared Error), sensitive to outliers, represented by ( f(x) – y)^2

Lasso regression encourages sparsity by using L1 norm of weights, with regularization it helps balance the trade-off between bias and variance, as a large penalty reduces model complexity by shrinking weights more

Step functions ued to convert linear model output into discrete class labels (serves as activation function to make binary decision)

Sigmoid = , Symmetric as show by partial , Gradient is shown by , The exponential in sigmoid helps the smoothness consistently while differentiating for different purposes, crucial for gradient operations.

In linaer classification model, the prediction is determind byt sign of

Online learning data points are processed one by one

W/ Perceptron, if missclassifcation, a positive example (answer was +1) guessed wrong, add x to weight, for negative example (answer was -1), subtract x, goal is to nudge weights slightly based on incorrect classifications, if correct nothing happens.

SGD updates on random point instead of full dataset

We only need to use loss function over misclassified data because they are the only ones contributing to updates they’re the only thing we’re trying to fix, it would be unnecessary to solve for loss as correctly classified add 0 to loss.

Support Vector Machines (SVM) the goal is to maximize the margin between classes, so the distance between decision boundary and the points, it works well in high-dimensional spaces, it doesn’t always find a unique solution, and its not just limited to binary classification. SVM has Soft and Hard, soft uses slack variables to tolerate some misclassification to improve overall model generalization.

Kernel methods like Radial Basis Functions main purpose it to transform non-linearly separable data into a higher-dimensional space where it then becomes linearly separarble, these kernels apply a non-linear transformation, so the data becomes linearly separable.

The role of the weight vector in SVM decision boundary is that it is perpidicular to the decision boundary which helps determine its orientation, it points in direction of increasing prediction score, and controls the til of the hyper plane, the bias determines the offset of the hyperplane from the origin

Cross-validation with folds is just how to split it, save one fold for validation, validation is used to approximate the test error by quickly testing a validation set during training to see if its moving in a good direction, more folds in cross-validation means a lower bias, higher variances and then an increased computational cost, there is less bias because there is more training data per fold, higher variances because there is more variation across folders, and because there is more training it has increased computation, if you don’t use cross-validation you most likely cannot find the best degree for a polynomial curve, training data alone is not a good indicator of how well the model generalizes, validation is needed for evaluating generalization, we don’t prioritize minimizing training error.

Increasing the polynomial degree on training functions doesn’t always reduce test error as it can begin to overfit the training data and the test error begins to decrease

Bias is simply described as error from being too simple, Variance is error from being too sensitive, Bias = Underfitting, Variance = Overfitting. Bias means not learning enough, Variance is not generalizing well.

Neural networks require non-lineartiy in activation functions to model complex patterns, if you only have a bunch of linear layers it will just be one big linear transformation, without non-linearity, the network can’t learn things like curves, boundaries or interactions between features.

Deep learning learns features automatically layer by layer, normal machine learning requires engineers to create good features from their outputs, hierarchical just refers to the beginning being simple and gradually gaining complexity in analyzation as it progresses.

Activation functions serve to introduce non-linearity.

Backward propagation / pass occurs during training only, not inference, as it serves to compute the gradients and thus update weights. We use partial derivatives in back propagation to compute how each weight affects the loss and then update them accordingly, basically like “If I tweak this weight, how much will the loss change?”

The main purpose of regularization in machine learning is to prevent overfitting, it helps the model generalize better to unseen data by penalizing complexity keeping the model from memorizing training data,

L2 regularization helps reduce variance by penalizing large model weights, L2 follows the gaussian (normal) distribution as it uses a prior (a belief of what the weights will be like) that most of the weights are probably close to 0 but some might be bigger or smaller,

With L1, MSE is used for Regression, Cross Entrop is used for classification

Drop out works by applying a binary mask to inputs and randomly deciding to drop specific neurons in each update, the typical probabilites are 0.2 for input layers and 0.5 for hidden layers, dropout randomly sets a subset of neurons to zero to prevent overfitting.

Data augmentation is just basically artificially expanding the dataset, with like cropping, different lighting configurations, rotations.

Early stopping uses validation loss by checking it consistently to determine when to stop training, if training continues after validation loss flattens or begins to increase, then overfitting may occur, early stopping prevents overfitting by stopping training when the validation loss stops improving,

Pooling layer doesn’t extract features from input data, it truly serves just to reduce the dimensionality of the data. Convolutional layers serve to learn patterns in the data from the same process, instead they output features maps showing where those patterns may appear. Max pooling is just a type of downsampling with a pooling layer where it just takes the max of that window, average pooling takes average instead, with pooling it adds translation invariance which is basically just that when data slightly shifts like an image of a cat is slightly different, the output will be way more similar because it ends up the same in the pooling output, translation invariance just refers to stability.

Equation for Output with filter(kernel) :

M – m + 1

If stride is not 1 or padding is not just 0, formula is

Flattening is another form of reduction with reducing from a higher dimension to a lower dimension, like from 2 or 3 to 1. It always results in 1D, its just a way to unroll the data, data is not loss, just organized.

With PCA graphs, black line refers to principal component, which is the direction in which the data varies the most, PCA finds this line to capture the essence of the data in fewer dimensions, the projection of the points is done in the direction of the line to keep the aprt of the data that aligns with the most important direction, PCA reduces but tries to capture as much of the original variance as possible/

When data is centered, PCA, low-rank approximation, and eigen-decomposition/SVD are equivalent, as they all just do some sort of project onto the principal subspace.

Logistic regression really just equals a linear model using a sigmoid, so it is still regression as it is predicting probabilities but it is being used for binary classification.

L1 (lasso) really just adds a penalty term to the loss function, it encourages sparsity as some weights become exactly 0, it really is just like “hey model, keep your weights small unless absolutely necessary”, cross entropy is compared the predicted probabilities from the activation functions to the true labels.

Neural networks contain non-linear functions making them impossible to just solve directly for the weights using algebra like in linear regression, instead we have to use GD to find a good set of weights, gradient based learning is necessary for training neural networks, neurla networks are not always linaer models, loss functions in neural networks are not generally convex as there is usually multiple non-linear layers.

Logistic regression =     , use this when you need a probability output and a binary decision, sigmoid can cause vanishing gradients, activation functions are used to introduce non-linearity which lets neural networks model complex patterns.

Used for binary.

Both Sigmoid and Tanh suffer from vanishing gradients.

, Avoids vanishing gradients but can cause dead neurons if values are always negative.

Used for multiclass.

Gradient Based Optimizations

– Simple but small gradients.

– Centered at 0

- Fast but not smooth

– Simplifies the gradient efficiently.

Likelihood is key in probabilistic models, it measures how likely the model’s parameters are to generate the observed data, if you’re multiplying a lot of small probabilites the result becomes tiny which causes instability. Log likelihood is taking the log of the likelihood value which turns the product into a sum which is much easier to optimize.

Some optimization algorithms such as Gradient Descent, specifically Full Batch uses all the training data to compute the gradient, which is very accurate but slow for large datasets, SGD is a random version of gradient descent which uses one random sample or small batch at a time, which is much fast enad helps introduce noise which can help prevent overfitting, Adam Optimizer uses Momentum and RMSProp, momentum is how you adjust the weights, and RMSProp is used to specify learning rates for specific weights instead of taking the usual one size fits all approach, it keeps a moving average of the squared gradients and divides the current gradient by this average, so large gradients get scaled down and small ones stay effective./

Batch Sizes have a big impact on the networks, smaller like from (1-32) have fast updates and good generalization, but a much slower convergence, Medium is like (32 – 128) which is the best of Small and Large, large is like (256+) which has smooth gradients but risk overfitting.

Gradient tells us how much a function changes with respect to each input variable, helps us know how to update weights, gradients are the vector of the partial derivatives of a function, they give us the slope of the function in each direction.

Max Pooling is a process of down sampling to reduce spatial size of feature maps while keeping the most important features, it reduces dimensionality, prevents overfitting, adds stability with translation invariance and keeps the strongest signals.

Linearly separable refers to the ability of a datset to be split by a straight line or a hyperplane that separates the classes perfectly, if you can draw a line that separates the data perfectly then the data is linearly separable, it is important as things like Perceptron and SVMs can only separate linearly separable data, loss functions measure how wrong the models predictions are, you optimize model by minimizing loss.

MSE , this is squaring the difference between the true value and the predicted value, it is sensitive to outliers however, used for regression.

Cross Entropy aka Log Loss , this penalizes confident but wrong predictions heavily, works well with probabilities, used for classification.

FNN is a basic type of neural network where information only flows in one direction, consisting of input layers, hidden layers, and output layers, input is taking in raw data, then passes through to hidden layers where each neuron does its thing with maybe linear or non-linear transformations, you can have multiple hidden layers, the output layer then produce the final prediction, with regression is usually a raw number, and for binary classification use sigmoid, and for multi-class classification use softmax. Layers are connected to the neuron as each connection has a weight w, and each neuron hjas a bis b, so they compute the output using whatever the activation function is.

True Positive = Predicted Positive and Actually Positive, True Negative = Predicted Negative and Actually Negative, False Positive = Predicted Postiive but Actually Negative, False Negative is Predicted Negative but actually positive,

Can be misleading if classes are imbalanced.

, good for when false positives are costly.

, good for when false negatives are costly

, balanced score when you care about both false postiives and false negatives

plots the true positive rate vs false positive rate at different thresholds, AUC = 1 means perfect model, AUC = 0.5 is random guessing, this Is good for evaluating how well your model separates the classes overall.

SGD helps optimization by quicky computing gradients of a random one or small batch of data points at a time, this is better because it means faster batches, helping escape local minima or saddle points, and better generalization. Momentum works like momentum, keeps some of speed direction as before a movement, it creates a smoother update, a faster converge and helps escape shallow areas where normal SGD might slow down.

PCA is a technique for dimensionality reduction, it helps reduce the # of features in your dataset while keeping the most important information, we do this to create faster training, reduce noise, visualize data better, and avoid overfitting, PCA finds directions of the data called principal components that capture the most variance, PCA looks at “super” features that combines the data in a smart way. The heart of PCA uses eigenvectors and eigenvalues, first you center the data by subtracting the mean, then you compute the covariance matrix, then we obtain the eigenvectors which are the directions of the new space, where each eigenvector = a principal component, they are orthogonal to each other, eigenvalues tells us the importance of each eigenvector, meaning a large eigenvalue = more variance captured. With PCA is may affect interpretability as it is just combining the features, it isn’t always helpful either, as if the important info was in low variance PCA might drop it.