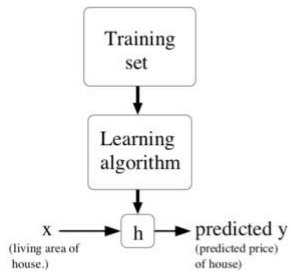
Cost Function

- Aka Cost function ,error function
- Def: how close the hypothesis functions are to the corresponding y.

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

- A machine learning hypothesis is a model that approximates the target function and maps the set of inputs to the set of outputs.



Common Loss functions

- Mean Absolute Error

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}^{(i)} - y^{(i)}|$$

  -
- Mean Squared Error

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

  -
- Binary cross-entropy
  → A binary classification

Gradient descent

- an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient
- Y: Cost function(J); X,Z: parameters from hypothesis
- Stochastic gradient descent
  - update the weights after each training sample

Activation Function

- a function that you use to get the output of node

$$z = w^T x + b$$

$$a = \sigma(z)$$

Common Types

- Sigmoid: used when output is between 0 and 1
- ReLu: default
- Tanh: between -1 and 1

Backward propagation

- output values are compared with the correct answer to compute the value of some predefined error-function

Bias & variance

- high bias: training set errors are high
    - Solution
        - Bigger network
        - Advance algo
        - NN architecture search
- high variance: (Dev set errors – training set errors) very large
    - more data
    - regularization
    - NN architecture

Cross-Validation

- a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data.

Regularization

- Reduce the test error at the expense of increased training error
- Most regularization strategies are based on regularizing estimators
    - Works by trading increased bias for reduced variance

Addressing overfitting

- Manually select which features to keep

Parameter Norm penalties

- Add a parameter norm penalty to the cost function

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

- α correspond to more regularization
- Works well when there are lots of features
- When our training algorithm minimizes the regularized cost function J, it will decrease both the original cost J on the training data and some measure of the size of the parameters θ (or some subset of the parameters)
- typically choose to use a parameter norm penalty Ω that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized

L2 Regularization (more often)

- aka Weight decay
- Drives the weights closer to the origin by adding a regularization term
- Multiplicatively shrink the weight vector by a constant factor on each step

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|^2$$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2 \qquad w: (n^{[l]}, n^{[l-1]})$$

    - Forbenius norm

$$W^{[l]} := W^{[l]} - \alpha\left[(\text{from backprop}) + \frac{\lambda}{m} W^{[l]}\right]$$
$$= W^{[l]} - \frac{\alpha\lambda}{m} W^{[l]} - \alpha(\text{from backprop})$$

- λ : regularization parameters
  - A hyperparameter
  - Set this using Dev set
  - Using hold out cross-validation to find the best values

L1 Regularization

- The sum of absolute values of the individual parameters

$$\frac{\lambda}{2m} \sum_{i=1}^{n_x} |w| = \frac{\lambda}{2m} ||w||_1 \qquad ||\boldsymbol{w}||_1 = \sum_i |w_i|,$$

Dataset Augmentation

- Create fake data and add it the training set

Noise Robustness

- the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights
- For RNN
  - noise has been used in the service of regularizing models by adding it to the weights
  - a stochastic implementation of Bayesian inference over the weights
  - Adding noise to the weights is a practical, stochastic way to reflect this uncertainty
- Injecting noise at the output targets
- assume that for some small constant a, the training set label y is correct with probability 1 − a, and otherwise any of the other possible labels might be correct.
  - label smoothing

Dropout Regularization

- go through each of the layers of the network and set some probability of eliminating a node in NN
- Inverted dropout
  - Randomly zero out different hidden units

Multi-Task Learning

- improve generalization by pooling the examples (which can be seen as soft constraints imposed on the parameters) arising out of several tasks.
- when part of a model is shared across tasks, that part of the model is more constrained towards good values (assuming the sharing is justified), often yielding better generalization.

Early Stopping

- We can obtain a model with better validation set error (and thus, hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error.
- Every time the error on the validation set improves, we store a copy of the model parameters.

- When the training algorithm terminates, we return these parameters, rather than the latest parameters.

Optimization

Normalization

- Normalize training sets
  - Subtract mean

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$x := x - \mu$$

  - Normalize variance

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} ** 2 \quad x \mathrel{/}= \sigma$$

- Use same μ σ^2 to normalize test set

Weight initialization for DN

- Partially solve Vanishing/exploding gradients problem
  - When training a very deep NN, the slopes can get either very big or very small, which makes training difficult
- Random initialization (Hyperparameter)
  - Set the variance of Wi to be equal to 1/n

$$W^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$$

  - ReLu: use 2/n
  - Tanh: use 1/n

Gradient Checking

- kind of debugging your back prop algorithm
- Take W and b → concatenate and reshape into a big vector θ

concatenate

$$J(W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}) = J(\theta)$$

Gradient Clipping

- Threshold the values of the gradients before performing ta gradient descent step

Minibatch

- Split examples to batches and each batch uses more than one but less than all of the training examples
- Mini-batch gradient descent
  - Process single batch of training samples at a time
- Take many gradient descent passes/epoch

Bias correction in exponentially weighted averages

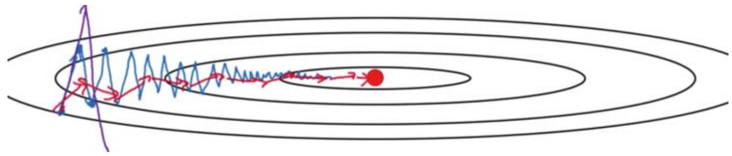
- make more accurate estimate during initial phase of the estimate

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t$$

Gradient descent with momentum

- compute an exponentially weighted average of your gradients and then use that gradient to update your weight
- take steps that are much smaller oscillations in the vertical direction and moving quickly to horizonal direction

$$V_{dW} = \beta V_{dW} + (1-\beta)\, dW$$
$$V_{db} = \beta V_{db} + (1-\beta)\, db$$
$$W := W - \alpha V_{dW}, \quad b := b - \alpha V_{db}$$

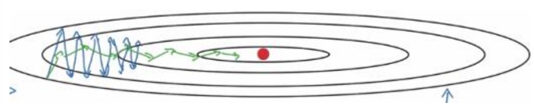


- Hyperparameters
  - Learning rate α
  - Control EWA: β (most common = 0.9)

RMSprop

- Speed up gradient descent
- Slow down the learning in the vertical direction and speed up learning in the horizontal direction

$$S_{dW} = \beta S_{dW} + (1-\beta)\, dW^2$$
$$S_{db} = \beta S_{db} + (1-\beta)\, db^2$$
$$W := W - \alpha \frac{dW}{\sqrt{S_{dW}}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$



Adam

- Idea: take Momentum and RMSprop, and put them together

$$V_{dW} = \beta_1 V_{dW} + (1-\beta_1)\, dW, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db \leftarrow \text{"moment"}\ \beta_1$$
$$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)\, dW^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db \leftarrow \text{"RMSprop"}\ \beta_2$$
$$V_{dW}^{corrected} = V_{dW} / (1-\beta_1^t), \quad V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$$
$$S_{dW}^{corrected} = S_{dW} / (1-\beta_2^t), \quad S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$$
$$W := W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

- Hyperparameters
  - Learning Rate α
  - β_1 : 0.9
  - β_2 : 0.999
  - ε : 10^{-8}

Learning Rate decay

- Slowly reduce learning rate overtime
- Learning rate
  - = $\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$
  - Decay-rate (hyperparameter)

- $= \alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ — exponentially decay.

Hyperparameter Tuning

- choosing a set of optimal hyperparameters for a learning algorithm

Order	Hyperparameters
Most important	Learning rate
Less important	Momentum term (β = 0.9); Mini-batch Size: # hidden units
Least Important	# of layers; Learning rate decay
Usually don't change	β_1 (0.9); β_2 (0.999); ε (10**-8)

How to select a set of values to explore

- Coarse to fine finding scheme
    - Zoom in a smaller region of the hyperparameters which work better and then sample more density within this space
- Appropriate scale for hyperparameters
    - Search in a log scale
    - Special case (Hyperparameters for EWA)

Training many models in parallel

- Use different models with different set of the hyperparameters at the same time

Batch normalization

- normalize the input layer by re-centering and re-scaling
- reduces the problem of coordinating updates across many layers
- Normalize any hidden layer (a) so as to train (W,b) faster
- Normalize (Z)

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \qquad \tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

learnable parameters of model.

$$X \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{W^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[\text{BN}]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]} \rightarrow$$

- Parameters: W, b, β, γ
- Working mini-batch

$$X^{\{1\}} \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow g^{[1]}(\tilde{z}^{[1]}) = a^{[1]} \xrightarrow{W^{[2]}, b^{[2]}} z^{[2]} \rightarrow \dots$$

$$\boxed{X^{\{2\}}} \longrightarrow z^{[1]} \xrightarrow[\text{BN}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow \dots$$

- At test time

$$\mu = \frac{1}{m}\sum_i z^{(i)} \qquad z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \leftarrow$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2 \qquad \tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$$

μ, σ^2: estimate using exponentially weighted average (across mini-batches)

Multi-class classification

Softmax

- Classify more than 2 classes
- assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0

Steps to build Deep learning model

1. set up dev/test metrics
2. Build initial ML System quickly
3. Use Bias, variance to priorize next steps

RNN

Gated Recurrent Units

- A modification to the RNN hidden layer that makes it much better capturing long range connection

LSTM

Peephole:

- Peephole connections allow the gates to access the constant error carousel (CEC), whose activation is the cell state.

CTC score function

- Used to find an RNN weight matrix that maximizes the probability of the label sequences in a training set, given the corresponding input sequences.