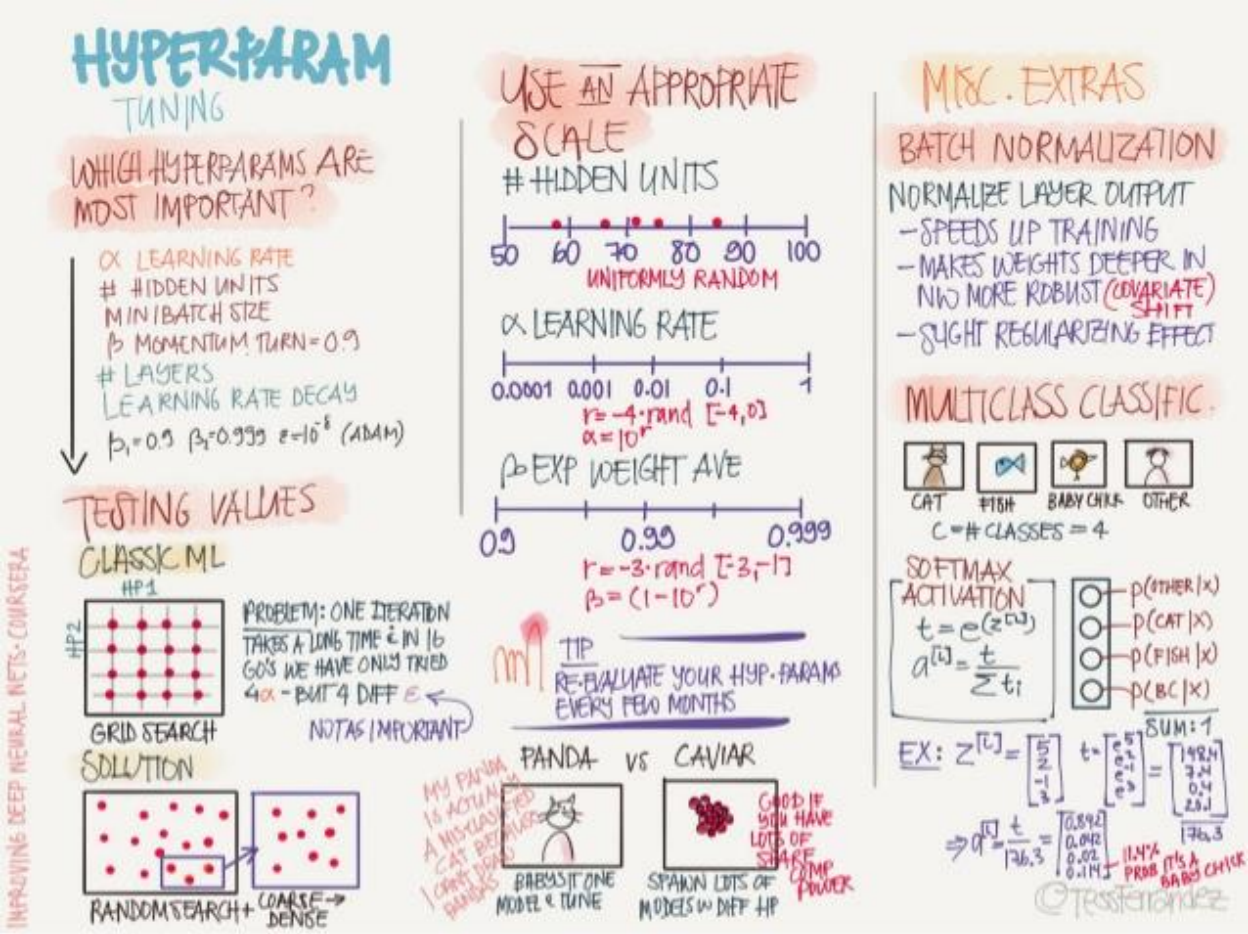
Hyperparameter tuning, Batch Normalization and Programming Frameworks

Hyperparameter tuning, Batch Normalization and Programming Frameworks

- Tuning process
- Using an appropriate scale to pick hyperparameters
- Hyperparameters tuning in practice: Pandas vs. Caviar
- Normalizing activations in a network
- Fitting Batch Normalization into a neural network
- Why does Batch normalization work?
- Batch normalization at test time
- Softmax Regression
- Training a Softmax classifier
- Deep learning frameworks
- TensorFlow

Extra Notes



Tuning process

- We need to tune our hyperparameters to get the best out of them.
- Hyperparameters importance are (as for Andrew Ng):
    1. Learning rate.
    2. Momentum beta.
    3. Mini-batch size.
    4. No. of hidden units.
    5. No. of layers.
    6. Learning rate decay.
    7. Regularization lambda.
    8. Activation functions.
    9. Adam `beta1` & `beta2`.
- Its hard to decide which hyperparameter is the most important in a problem. It depends a lot on your problem.
- One of the ways to tune is to sample a grid with `N` hyperparameter settings and then try all settings combinations on your problem.

- Try random values: don't use a grid.
- You can use `Coarse to fine sampling scheme`:
    - When you find some hyperparameters values that give you a better performance - zoom into a smaller region around these values and sample more densely within this space.
- These methods can be automated.

Using an appropriate scale to pick hyperparameters

- Let's say you have a specific range for a hyperparameter from "a" to "b". It's better to search for the right ones using the logarithmic scale rather then in linear scale:
    - Calculate: `a_log = log(a) # e.g. a = 0.0001 then a_log = -4`
    - Calculate: `b_log = log(b) # e.g. b = 1 then b_log = 0`
    - Then:
    - `r = (a_log - b_log) * np.random.rand() + b_log`
    - `# In the example the range would be from [-4, 0] because rand range [0,1)`
    - `result = 10^r`

      It uniformly samples values in log scale from [a,b].

- If we want to use the last method on exploring on the "momentum beta":
    - Beta best range is from 0.9 to 0.999.
    - You should search for `1 - beta in range 0.001 to 0.1 (1 - 0.9 and 1 - 0.999)` and the use `a = 0.001` and `b = 0.1`. Then:
    - `a_log = -3`
    - `b_log = -1`
    - `r = (a_log - b_log) * np.random.rand() + b_log`
    - `beta = 1 - 10^r   # because 1 - beta = 10^r`

Hyperparameters tuning in practice: Pandas vs. Caviar

- Intuitions about hyperparameter settings from one application area may or may not transfer to a different one.
- If you don't have much computational resources you can use the "babysitting model":
    - Day 0 you might initialize your parameter as random and then start training.
    - Then you watch your learning curve gradually decrease over the day.
    - And each day you nudge your parameters a little during training.
    - Called panda approach.
- If you have enough computational resources, you can run some models in parallel and at the end of the day(s) you check the results.
    - Called Caviar approach.

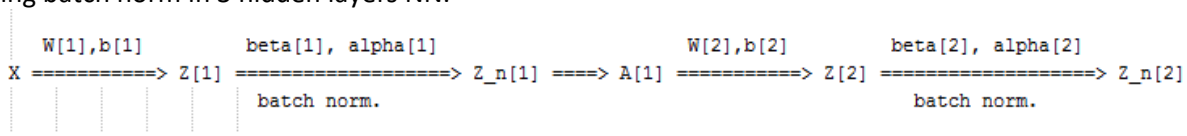
Normalizing activations in a network

- In the rise of deep learning, one of the most important ideas has been an algorithm called **batch normalization**, created by two researchers, Sergey Ioffe and Christian Szegedy.
- Batch Normalization speeds up learning.
- Before we normalized input by subtracting the mean and dividing by variance. This helped a lot for the shape of the cost function and for reaching the minimum point faster.

- The question is: *for any hidden layer can we normalize* `A[l]` *to train* `W[l]`, `b[l]` *faster?* This is what batch normalization is about.
- There are some debates in the deep learning literature about whether you should normalize values before the activation function `Z[l]` or after applying the activation function `A[l]`. In practice, normalizing `Z[l]` is done much more often and that is what Andrew Ng presents.
- Algorithm:
    - Given `Z[l] = [z(1), ..., z(m)]`, i = 1 to m (for each input)
    - Compute `mean = 1/m * sum(z[i])`
    - Compute `variance = 1/m * sum((z[i] - mean)^2)`
    - Then `Z_norm[i] = (z(i) - mean) / np.sqrt(variance + epsilon)` (add `epsilon` for numerical stability if variance = 0)
        - Forcing the inputs to a distribution with zero mean and variance of 1.
    - Then `Z_tilde[i] = gamma * Z_norm[i] + beta`
        - To make inputs belong to other distribution (with other mean and variance).
        - gamma and beta are learnable parameters of the model.
        - Making the NN learn the distribution of the outputs.
        - *Note:* if `gamma = sqrt(variance + epsilon)` and `beta = mean` then `Z_tilde[i] = Z_norm[i]`

Fitting Batch Normalization into a neural network

- Using batch norm in 3 hidden layers NN:



- Our NN parameters will be:
    - `W[1]`, `b[1]`, ..., `W[L]`, `b[L]`, `beta[1]`, `gamma[1]`, ..., `beta[L]`, `gamma[L]`
    - `beta[1]`, `gamma[1]`, ..., `beta[L]`, `gamma[L]` are updated using any optimization algorithms (like GD, RMSprop, Adam)
- If you are using a deep learning framework, you won't have to implement batch norm yourself:
    - Ex. in Tensorflow you can add this line: `tf.nn.batch-normalization()`
- Batch normalization is usually applied with mini-batches.
- If we are using batch normalization parameters `b[1]`, ..., `b[L]` doesn't count because they will be eliminated after mean subtraction step, so:
- `Z[l] = W[l]A[l-1] + b[l] => Z[l] = W[l]A[l-1]`
- `Z_norm[l] = ...`
- `Z_tilde[l] = gamma[l] * Z_norm[l] + beta[l]`
    - Taking the mean of a constant `b[l]` will eliminate the `b[l]`
- So if you are using batch normalization, you can remove b[l] or make it always zero.
- So the parameters will be `W[l]`, `beta[l]`, and `alpha[l]`.
- Shapes:
    - `Z[l] - (n[l], m)`
    - `beta[l] - (n[l], m)`
    - `gamma[l] - (n[l], m)`

Why does Batch normalization work?

- The first reason is the same reason as why we normalize X.
- The second reason is that batch normalization reduces the problem of input values changing (shifting).
- Batch normalization does some regularization:
  - Each mini batch is scaled by the mean/variance computed of that mini-batch.
  - This adds some noise to the values `Z[l]` within that mini batch. So similar to dropout it adds some noise to each hidden layer's activations.
  - This has a slight regularization effect.
  - Using bigger size of the mini-batch you are reducing noise and therefore regularization effect.
  - Don't rely on batch normalization as a regularization. It's intended for normalization of hidden units, activations and therefore speeding up learning. For regularization use other regularization techniques (L2 or dropout).

Batch normalization at test time

- When we train a NN with Batch normalization, we compute the mean and the variance of the mini-batch.
- In testing we might need to process examples one at a time. The mean and the variance of one example won't make sense.
- We have to compute an estimated value of mean and variance to use it in testing time.
- We can use the weighted average across the mini-batches.
- We will use the estimated values of the mean and variance to test.
- This method is also sometimes called "Running average".
- In practice most often you will use a deep learning framework and it will contain some default implementation of doing such a thing.

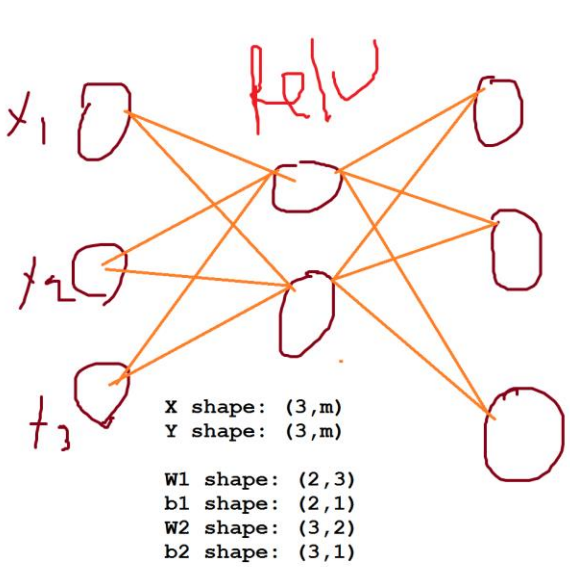
Softmax Regression

- In every example we have used so far we were talking about binary classification.
- There are a generalization of logistic regression called Softmax regression that is used for multiclass classification/regression.
- For example if we are classifying by classes `dog`, `cat`, `baby chick` and `none of that`
  - Dog `class = 1`
  - Cat `class = 2`
  - Baby chick `class = 3`
  - None `class = 0`
  - To represent a dog vector `y = [0 1 0 0]`
  - To represent a cat vector `y = [0 0 1 0]`
  - To represent a baby chick vector `y = [0 0 0 1]`
  - To represent a none vector `y = [1 0 0 0]`
- Notations:
  - `C = no. of classes`
  - Range of classes is `(0, ..., C-1)`
  - In output layer `Ny = C`

- Each of C values in the output layer will contain a probability of the example to belong to each of the classes.
- In the last layer we will have to activate the Softmax activation function instead of the sigmoid activation.
- Softmax activation equations:
- `t = e^(Z[L])` `# shape(C, m)`
- `A[L] = e^(Z[L]) / sum(t)` `# shape(C, m), sum(t) - sum of t's for each example (shape (1, m))`

Training a Softmax classifier

- There's an activation which is called hard max, which gets 1 for the maximum value and zeros for the others.
    - If you are using NumPy, its `np.max` over the vertical axis.
- The Softmax name came from softening the values and not harding them like hard max.
- Softmax is a generalization of logistic activation function to `C` classes. If `C = 2` softmax reduces to logistic regression.
- The loss function used with softmax:
- `L(y, y_hat) = - sum(y[j] * log(y_hat[j])) # j = 0 to C-1`
- The cost function used with softmax:
- `J(w[1], b[1], ...) = - 1 / m * (sum(L(y[i], y_hat[i]))) # i = 0 to m`
- Back propagation with softmax:
- `dZ[L] = Y_hat - Y`
- The derivative of softmax is:
- `Y_hat * (1 - Y_hat)`
- Example:




```
Z1 = W1* X + b1      #shape (2,m)
A1 = RELU(Z1)        #shape (2,m)

Z2 = W2 * A1 + b2    #shape (3,m)

Then we need to get A of softmax
We compute:
t = e^Z2    #shape (3,m)
sumAll = sum(t,C)      #shape (1,m)

Finally
A2 = t / sumAll
```

Deep learning frameworks

- It's not practical to implement everything from scratch. Our numpy implementations were to know how NN works.

- There are many good deep learning frameworks.
- Deep learning is now in the phase of doing something with the frameworks and not from scratch to keep on going.
- Here are some of the leading deep learning frameworks:
    - Caffe/ Caffe2
    - CNTK
    - DL4j
    - Keras
    - Lasagne
    - mxnet
    - PaddlePaddle
    - TensorFlow
    - Theano
    - Torch/Pytorch
- These frameworks are getting better month by month. Comparison between them can be found here.
- How to choose deep learning framework:
    - Ease of programming (development and deployment)
    - Running speed
    - Truly open (open source with good governance)
- Programming frameworks can not only shorten your coding time but sometimes also perform optimizations that speed up your code.

TensorFlow

- In this section we will learn the basic structure of TensorFlow programs.
- Lets see how to implement a minimization function:
    - Example function: `J(w) = w^2 - 10w + 25`
    - The result should be `w = 5` as the function is `(w-5)^2 = 0`
    - Code v.1:

```
import numpy as np
import tensorflow as tf


w = tf.Variable(0, dtype=tf.float32)                    # creating a variable w
cost = tf.add(tf.add(w**2, tf.multiply(-10.0, w)), 25.0)        # can be written as this - cost = w**2 - 10*w + 25
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
session.run(w)    # Runs the definition of w, if you print this it will print zero
session.run(train)

print("W after one iteration:", session.run(w))

for i in range(1000):
    session.run(train)
```

```
print("W after 1000 iterations:", session.run(w))
```

- Code v.2 (we feed the inputs to the algorithm through coefficients):

```
import numpy as np
import tensorflow as tf


coefficients = np.array([[1.], [-10.], [25.]])

x = tf.placeholder(tf.float32, [3, 1])
w = tf.Variable(0, dtype=tf.float32)                    # Creating a
variable w
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]

train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
session.run(w)    # Runs the definition of w, if you print this
it will print zero
session.run(train, feed_dict={x: coefficients})

print("W after one iteration:", session.run(w))

for i in range(1000):
    session.run(train, feed_dict={x: coefficients})

print("W after 1000 iterations:", session.run(w))
```

- In TensorFlow you implement only the forward propagation and TensorFlow will do the backpropagation by itself.
- In TensorFlow a placeholder is a variable you can assign a value to later.
- If you are using a mini-batch training you should change the `feed_dict={x: coefficients}` to the current mini-batch data.
- Almost all TensorFlow programs use this:
- `with tf.Session() as session:       # better for cleaning up in case of error/exception`
- ```
  session.run(init)
  session.run(w)
  ```

- In deep learning frameworks there are a lot of things that you can do with one line of code like changing the optimizer. ***Side notes:***
- Writing and running programs in TensorFlow has the following steps:
    1. Create Tensors (variables) that are not yet executed/evaluated.
    2. Write operations between those Tensors.
    3. Initialize your Tensors.
    4. Create a Session.
    5. Run the Session. This will run the operations you'd written above.
  ```

- Instead of needing to write code to compute the cost function we know, we can use this line in TensorFlow : `tf.nn.sigmoid_cross_entropy_with_logits(logits = ..., labels = ...)`
- To initialize weights in NN using TensorFlow use:
- `W1 = tf.get_variable("W1", [25,12288], initializer = tf.contrib.layers.xavier_initializer(seed = 1))`
-
- `b1 = tf.get_variable("b1", [25,1], initializer = tf.zeros_initializer())`
- For 3-layer NN, it is important to note that the forward propagation stops at `Z3`. The reason is that in TensorFlow the last linear layer output is given as input to the function computing the loss. Therefore, you don't need `A3`!
- To reset the graph use `tf.reset_default_graph()`

Extra Notes

- If you want a good papers in deep learning look at the ICLR proceedings (Or NIPS proceedings) and that will give you a really good view of the field.
- Who is Yuanqing Lin?
    - Head of Baidu research.
    - First one to win ImageNet
    - Works in PaddlePaddle deep learning platform.