


Neural Networks (part 2)



Most of the materials are taken from [here](#)

Nastaran Okati

Outline

- ▷ Data Preprocessing
- ▷ Weight Initialization
- ▷ Regularization
- ▷ Loss Functions

1.

Data Preprocessing

Mean Subtraction

- ▷ The most common form of preprocessing
- ▷ Subtracting the mean across every individual feature in the data
- ▷ It has the geometric interpretation of centering the data around the origin along every dimension.

Normalization

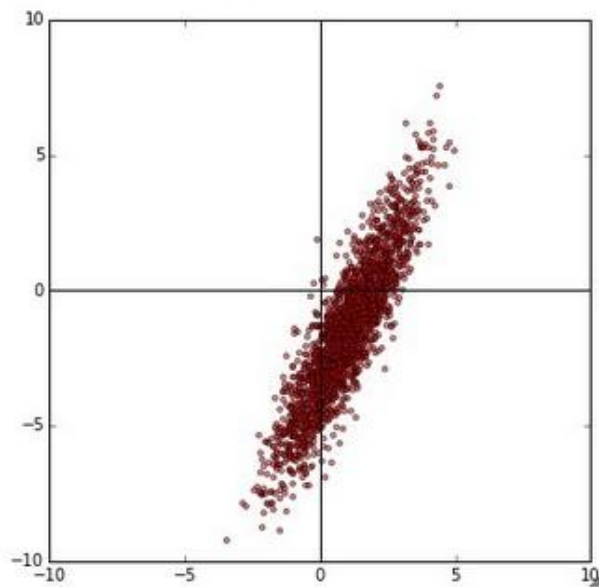
- ▷ Normalizing the data dimensions so that they are of approximately the same scale
- ▷ One way is to divide each dimension by its standard deviation, once it has been zero-centered
- ▷ Another form of this preprocessing normalizes each dimension so that the min and max along the dimension is -1 and 1 respectively

Normalization

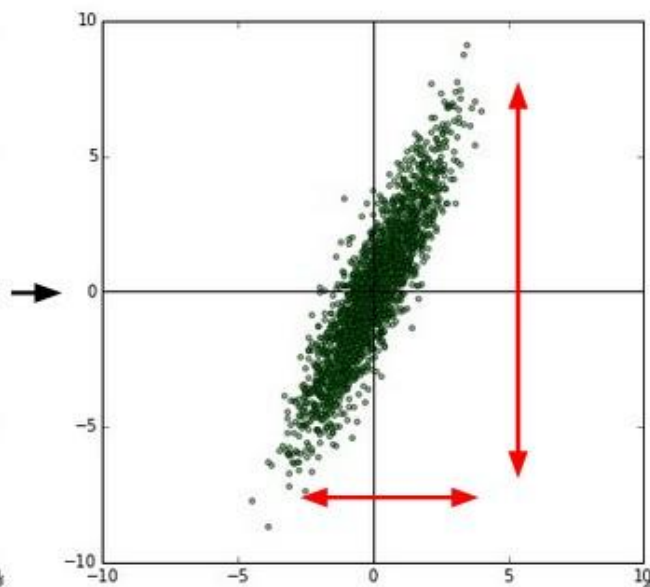
- ▷ Apply this preprocessing if you believe that different input features have different scales but they should be of approximately equal importance to the learning algorithm
- ▷ Do you need this preprocessing for images?

Effect of Preprocessing

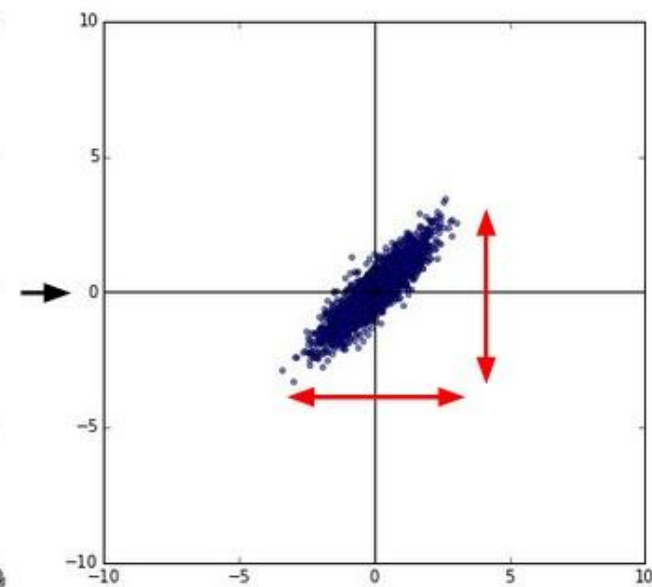
original data



zero-centered data



normalized data



Be careful!

- ▷ Any preprocessing statistics (e.g. the data mean) must only be computed on the training data, and then applied to the validation / test data.
- ▷ The mean must be computed only over the training data and then subtracted equally from all splits (train/val/test).

2.

Weight Initialization

Pitfall: all zero initialization

- ▷ We do not know the final value of every weight in the trained network, but it is reasonable to assume that approximately half of the weights will be positive and half of them will be negative.
- ▷ A reasonable-sounding idea then might be to set all the initial weights to zero.
- ▷ This is a mistake, because if every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates.

Small random numbers

- ▷ We still want the weights to be very close to zero, but not identically zero.
- ▷ As a solution, it is common to initialize the weights of the neurons to small numbers and refer to doing so as symmetry breaking.
- ▷ The neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the full network

Warning!

Are smaller numbers strictly better for weight initialization?

Calibrating the variances with $1/\sqrt{n}$

- ▷ The distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs.
- ▷ We can normalize the variance of each neuron's output to 1 by scaling its weight vector by the square root of its number of inputs)
- ▷ That is, the recommended heuristic is to initialize each neuron's weight vector as: $w = \text{np.random.randn}(n) / \sqrt{n}$,

$$\text{Var}(s) = \text{Var}\left(\sum_i^n w_i x_i\right)$$

$$= \sum_i^n \text{Var}(w_i x_i)$$

$$= \sum_i^n [E(w_i)]^2 \text{Var}(x_i) + E[(x_i)]^2 \text{Var}(w_i) + \text{Var}(x_i) \text{Var}(w_i)$$

$$= \sum_i^n \text{Var}(x_i) \text{Var}(w_i)$$

$$= (n \text{Var}(w)) \text{Var}(x)$$

if we want s to have the same variance as all of its inputs x , then during initialization we should make sure that the variance of every weight w is $1/n$.

Initializing the biases

- ▷ It is possible and common to initialize the biases to be zero, since the asymmetry breaking is provided by the small random numbers in the weights.
- ▷ For ReLU non-linearities, some people like to use small constant value such as 0.01 for all biases because this ensures that all ReLU units fire in the beginning and therefore obtain and propagate some gradient

Batch Normalization

- ▷ Force the activations throughout a network to take on a unit gaussian distribution at the beginning of the training.
- ▷ This is possible because normalization is a simple differentiable operation
- ▷ Insert the BatchNorm layer immediately after fully connected layers (or convolutional layers), and before non-linearities.

Batch Normalization

- ▷ BN has become a very common practice to use Batch Normalization in neural networks.
- ▷ Networks that use Batch Normalization are significantly more robust to bad initialization
- ▷ Batch normalization can be interpreted as doing preprocessing at every layer of the network, but integrated into the network itself in a differentiable manner.

3. Regularization

Regularization

- ▷ Techniques which are used to reduce the error on the test set at an expense of increased training error
- ▷ Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error
- ▷ Controlling the capacity of Neural Networks to prevent **overfitting**

L2 regularization

- ▷ The most common form of regularization
- ▷ Penalizes the squared magnitude of all parameters directly in the objective.
- ▷ For every weight w in the network, we add the term $\frac{1}{2}\lambda w^2$ to the objective, where λ is the regularization strength
- ▷ It has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring diffuse weight vectors

L2 regularization

- ▷ Encouraging the network to use all of its inputs a little rather than some of its inputs a lot.
- ▷ Notice that during gradient descent parameter update, using the L2 regularization ultimately means that every weight is decayed linearly: $W \leftarrow W - \lambda W$ towards zero.

L1 regularization

- ▷ For each weight w we add the term $\lambda |w|$ to the objective.
- ▷ It is possible to combine the L1 regularization with the L2 regularization: $\lambda_1 |w| + \lambda_2 w^2$
- ▷ The L1 regularization leads the weight vectors to become sparse during optimization
- ▷ Neurons with L1 regularization use only a sparse subset of their most important inputs and become nearly invariant to the “noisy” inputs.

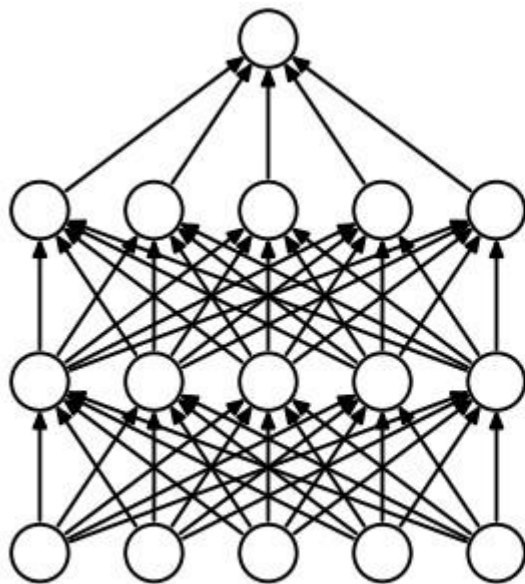
Max norm constraints

- ▷ Enforce an absolute upper bound on the magnitude of the weight vector for every neuron and use projected gradient descent to enforce the constraint.
- ▷ In practice, this corresponds to performing the parameter update as normal, and then enforcing the constraint by clamping the weight vector \vec{w} of every neuron to satisfy $\|\vec{w}\|_2 < c$. Typical values of c are on orders of 3 or 4.

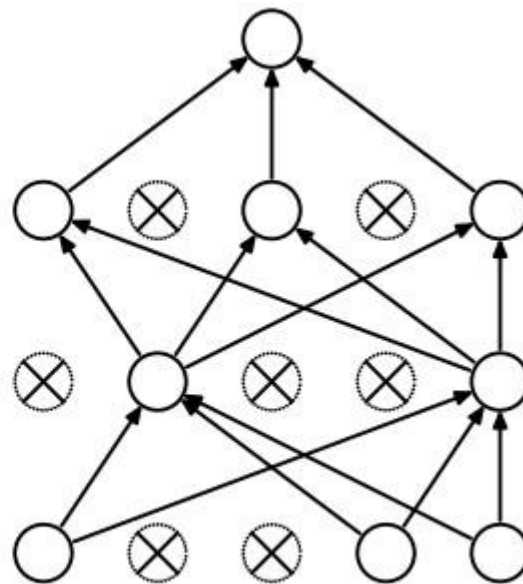
Dropout

- ▷ Effective, simple and recently introduced regularization technique that complements the other methods
- ▷ Dropout is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise.
- ▷ During training, Dropout can be interpreted as sampling a Neural Network within the full Neural Network, and only updating the parameters of the sampled network based on the input data.

During testing there is no dropout applied, with the interpretation of evaluating an averaged prediction across the exponentially-sized ensemble of all sub-networks



(a) Standard Neural Net



(b) After applying dropout.

Noise in forward pass

- ▷ Dropout falls into a more general category of methods that introduce stochastic behavior in the forward pass of the network.
- ▷ Convolutional Neural Networks also take advantage of this theme with methods such as stochastic pooling, fractional pooling, and data augmentation.

Bias regularization

- ▷ Do we need to regularize bias parameters?
- ▷ They do not interact with the data through multiplicative interactions, and therefore do not have the interpretation of controlling the influence of a data dimension on the final objective.

Final word on regularization

- ▷ It is most common to use a single, global L2 regularization strength that is cross-validated.
- ▷ It is also common to combine this with dropout applied after all layers.
- ▷ The value of $p=0.5$ is a reasonable default, but this can be tuned on validation data.

4. Loss Functions

Loss functions

- ▷ We have discussed the regularization loss part of the objective
- ▷ The second part of an objective is the data loss which in a supervised learning problem measures the compatibility between a prediction and the ground truth label.
- ▷ The data loss takes the form of an average over the data losses for every individual example

SVM Loss

- ▷ Lets abbreviate $f=f(x_i;W)$ to be the activations of the output layer in a Neural Network.
- ▷ One of two most commonly seen cost functions in this setting is the SVM

$$L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)$$

Softmax Loss

- ▷ The second common choice is the Softmax classifier that uses the cross-entropy loss:

$$L_i = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right)$$

Hierarchical Softmax

- ▷ What if you have a large number of classes?
- ▷ The hierarchical softmax decomposes labels into a tree.
- ▷ Each label is then represented as a path along the tree, and a Softmax classifier is trained at every node of the tree to disambiguate between the left and right branch.

Attribute Classification

- ▷ What if every example may or may not have a certain attribute, and the attributes are not exclusive?
- ▷ What are some examples of this domain?
- ▷ A sensible approach in this case is to build a binary classifier for every single attribute independently.

Attribute Classification

$$L_i = \sum_j \max(0, 1 - y_{ij} f_j)$$

- ▷ The sum is over all categories j , and Y_{ij} is either +1 or -1 and the score vector f_j will be positive when the class is predicted to be present and negative otherwise.
- ▷ In which cases is the loss accumulated?

Regression

- ▷ The task of predicting real-valued quantities
- ▷ It is common to compute the loss between the predicted quantity and the true answer and then measure the L2 squared norm, or L1 norm of the difference.

$$L_i = \|f - y_i\|_2^2$$

$$L_i = \|f - y_i\|_1 = \sum_j |f_j - (y_i)_j|$$

Be careful!

- ▷ Note that the L2 loss is much harder to optimize than a more stable loss such as Softmax.
- ▷ In Softmax, the precise value of each score is less important: It only matters that their magnitudes are appropriate.
- ▷ When faced with a regression task, first consider if it is necessary. Instead, have a strong preference to discretizing your outputs to bins and perform classification over them

What is the learning process?

Wait until next session! 

Thanks!

Any questions?