DropOut:

* Randomly remove hidden units from the neural network. (regularizes your network)
* The way this is done is by setting hidden unit to 0.  That means their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.
* These hidden units remain dropped for one single forward and backward propagation.
* At each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.
* The probability of a hidden unit being removed is called dropout probability. In general, the probability of a hidden unit being dropped out is 0.5
* Dropout prevents overfitting of data. The reason is, at fully connected layer each unit is weighted sum of all the units in the previous layers.
* As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations.
* As a result there is co-dependency among the hidden units. More often hidden units co-adopt to generate complicated patterns, by introducing dropout a hidden unit will generate a feature which is general and just part of the feature which only when combined with the feature extract of other hidden unit makes sense.
* Because of the above it is one of the most common regularization method.
* In machine learning, regularization is way to prevent over-fitting. Regularization reduces over-fitting by adding a penalty to the loss function
* Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.
* How is dropout done: forward propagation
  + Use random binary mask mk
  + Layer pre-activation – ak(x) = bk + Wk\*h(k-1)\*x
  + Hidden layer activation hk(x) = g(ak(x))\*mk
  + o/p layer activation -h(L+1)(x) = o(a(L+1)(x)) = f(x)
* Back propagation
  + We need to replace h(x) with h(x)\*mk
* Training Phase: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).
* Test time
  + Use all activations, but reduce them by a factor *p* (to account for the missing activations during training).
  + If you have used a probability of 0.5, you need to replace each of these masks with 0.5
  + For single hidden layer In practice this can be shown as averaging of the hidden layer output
* Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.
* With H hidden units, each of which can be dropped, we have
* 2^H possible models. In testing phase, the entire network is considered and each activation is reduced by a factor p.

References: <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>

<http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf>

**Batch Normalization**

* In neural networks we follow normalization/standardization to bring all the data in one common scale
* Normalization – bring data to scale 0-1
* Standardization – (x – m)/s (also called normalization in practice)
* Done to prevent skewed data
* Also features might be on different scale (eg : age, salary of a person)
* The larger data points in the non – normalized data can cause imbalanced gradients which therefore cause exploding gradient problem
* Also non normalized data decreases the training speed
* By normalizing the data we can increase training speed and get rid of exploding gradient problem
* But in neural network in spite of having normalized data we still have another problem. This is because when the network learns we update the weights with each epoch based on SGD, if the learned weights of one filter is very very much larger than others.
* This large weight will cause the output from corresponding neuron to be very large, this will be cascaded throughout the network causing instability
* To prevent this, we apply batch normalization to the o/p of hidden activation layers.
* By applying batch normalization, we perform x` = (x-m)/s on each of the hidden unit
* Multiply this normalized o/p with arbitrary parameter g, i.e. z = x`\*g
* Add another parameter b to this product => (z + b)
* This sets new standard deviation and mean for the data
* m,s,g,b all are trainable
* This makes sure weights in the hidden layers do not go to arbitrary high value
* Batch normalization also takes care of covariance shift, that is if we have trained our model let’s say h(X) = y , if the distribution of X changes , our model will not be able to predict h(X`) = y
* The distribution of hidden layer activation changes with each iteration of neural network as a result shifting the mean and variance, by doing batch norm we are telling no matter how the weights and biases are changing, the mean and variation of a particular hidden layer activation remains the same

**Dense Net**

* Each layer receives i/p from all its preceding layers, because of this property we can reduce the number of hidden units of each layer and compactly pack the neurons. So, the network will be faster and computationally more efficient.
* If a normal neural network of depth L has L connnection, while a denseNet will have L(L+1)/2 connections.
* For current layer, the feature-maps of all previous layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.
* DenseNets exploit the potential of the network through feature reuse, yielding condensed models that are easy to train and highly parameterefficient. Concatenating feature-maps learned by different layers increases variation in the input of subsequent layers and improves efficiency
* DenseNet layers are very narrow (e.g., 12 filters per layer), adding only a small set of feature-maps to the “collective knowledge” of the network and keep the remaining feature maps unchanged—and the final classifier makes a decision based on all feature-maps in the network.

I/p is combined by channelized concatenation (?)

* Feature generation in dense Net (Forward Propagation)
* In dense net we have multiple dense blocks, with in each dense block the size of all the feature maps are same since we perform concatenation.
* Advantages of using a densenet:
  + Error signal can be propagated to pervious layers directly
  + No. of parameters in dense net is lower than convolutional neural network
  + More diversified features
  + Classifier uses features from all the complexity levels, this tend to give more smoother decision boundary’

References:

<https://arxiv.org/abs/1608.06993>

http://openaccess.thecvf.com/content\_cvpr\_2017/papers/Huang\_Densely\_Connected\_Convolutional\_CVPR\_2017\_paper.pdf