1. **Can we use dropout with batch normalization together? Why or why not?**

   We can. Dropout focuses on dropping neurons randomly during training to prevent model overfitting. Whereas batch normalization focuses on lessening the covariate shift of the model, applying normalization at each layer. Both have regularizing effects during training and can be used together.

   Furthermore, in the paper "*Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks (Chen., et al 2019)*", the authors demonstrated using batch normalization and dropout together with a proposed novel "Independent-Component (IC) layer", thus demonstrating that it is indeed possible to use both dropout and batch normalization together.

2. **Why ADAM optimizer might not always reach better convergence results compared to SGD?**

   Because from some experiments carried out and demonstrated in some papers, like "*The Marginal Value of Adaptive Gradient Methods in Machine Learning (Wilson., et al 2018)*", found that SGD outperform adaptive methods with the same amount of hyperparameter tuning, and that adaptive methods often display faster initial progress on the training set, but their performance quickly plateaus on the testing set.

   This may be due to the reason that adaptive methods tend to give undue influence to spurious features that have no effect on out-of-sample generalization, causing it to not always reach better convergence results compared to SGD.

3. **What are the differences between a traditional Autoencoder (AE), Variational Autoencoder (VAE), and Adversarial Autoencoder (AAE)?**

   Autoencoder (AE)-
   An autoencoder compresses its input to vector with fewer inputs than the original data and transforms it back into a tensor with the same shape as the input over several neural net layers. Kind of like learning a compression algorithm for the specific dataset.

   Adversarial Autoencoder (AAE)-
   An adversarial autoencoder differs from the autoencoder in that it can turn an autoencoder into a generative model. It does this by training it with an adversarial criterion: the encoder learns to convert the data distribution to the prior distribution, while the decoder learns a deep generative model that maps the imposed prior to the data distribution.

Variational Autoencoder (VAE)-
The main difference between a variational autoencoder and an adversarial autoencoder is in the loss computed on the latent representation. VAE relies on KL divergence between the distribution obtained from the encoder and the prior, in order to enforce that the probabilistic encoder yields the prior distribution. In contrast, AAE uses the adversarial loss similar to GANs.

4. **What's the objective function in training the discriminator in the original GAN? Why Least Square Gan (LSGAN) might improve on the original GAN?**

   The objective function in training the discriminator in the original GAN is as follows:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

   The discriminator seeks to maximize the average of the log probability for real images and the log of the inverted probabilities of fake images. Such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake).

   According to the original paper "*Least Square Generative Adversarial Networks (Mao., et al 2017)*", the authors found that the loss function of regular GANs may lead to the vanishing gradient problem during the learning process. Whereas by the proposed LSGAN, minimizing the objective function of LSGAN yields minimizing the Pearson x^2 divergence. This allows LSGAN to generate higher quality images than regular GANs and that it also more stable during the learning process.

5. **How WGAN improves on the original GAN?**

   WGAN stands for Wasserstein GAN. Wasserstein Distance is a measure of the distance between two probability distributions. The original authors proposed a smart transformation of the formula based on the Kantorovich-Rubinstein duality to:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \le K} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]$$

   So compared to the original GAN, WGAN improves by undertaking the following changes:

   - After every gradient update on the critic function, it clamps the weights to a small fixed range, [-c, c]
   - It uses a new loss function derived from the Wasserstein distance, where there is no logarithm anymore. Rather than playing a direct critic, the discriminator is a helper for estimating the Wasserstein metric between real and generated data distribution.

6. **Training of GANs is notoriously hard to converge. What are some effective tips for training GAN?**

   One of the approaches to training a stable GAN model was proposed in the paper "*Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (Radford., et al 2016)*", in which the authors describe a Deep Convolutional GAN (DCGAN) which performs very well for image synthesis tasks.
   Some tips included in the paper are:

   - Replace any pooling layers convolutions with strided convolutions. This allows the network to learn its own spatial downsampling.
   - Use batch normalization in both the generator and the discriminator. This stabilizes the training process.
   - Remove fully connected hidden layers for deeper architectures. In the discriminator, after the convolutional layers, the results are flattened and passed directly to the output.
   - Use ReLU activation in generator for all layers except for the output, which uses Tanh. And use Leaky ReLU activation in the discriminator for all layers. Activation functions such as ReLU are used to address the vanishing gradient problem in deep convolutional neural networks and promote sparse activations.

   Another paper "*Improved Techniques for Training GANs (Salimans., et al 2016)*" listed five techniques to consider that are claimed to improve convergence when training GANs:

   - Feature matching- develop a GAN semi-supervised learning.
   - Minibatch discrimination- develop features across multiple samples in a minibatch
   - Historical averaging- update the loss function to incorporate history
   - One-sided label smoothing- scaling target values for the discriminator
   - Virtual batch normalization- calculation of batch norm statistics using a reference batch of real images