1. **Learning with Large Datasets**:
   * True, large datasets can help improve performance, especially when a model has high variance. But, if a model has high bias, more data might not necessarily improve its performance. This emphasizes the importance of understanding the bias-variance tradeoff.
2. **Stochastic Gradient Descent (SGD)**:
   * Instead of using all the training samples to compute the gradient, SGD uses just one example at a time. This can be faster, especially for large datasets.
   * Since it's stochastic in nature, it might not settle down at the exact minimum but will oscillate around it. However, in complex landscapes (like neural networks), this oscillation can help SGD escape local minimums (which is a good thing).
3. **Mini-Batch Gradient Descent**:
   * It strikes a balance between batch gradient descent and SGD. It updates the weights using a subset of the training data (a mini-batch) instead of a single data point or the entire dataset.
   * Vectorized operations on mini-batches can make this approach computationally efficient.
4. **Stochastic Gradient Descent Convergence**:
   * Monitoring the cost function is crucial to ensure SGD is working correctly.
   * Adjusting the learning rate during training (known as learning rate annealing or scheduling) can help achieve better convergence.
5. **Online Learning**:
   * This approach is suitable for continuously incoming data. For instance, if you're running a website and you're constantly getting new user data, online learning allows you to update your model in real-time based on this data.
   * It's great for adapting to new data distributions or patterns as the data comes in.
6. **Map Reduce and Data Parallelism**:
   * This is a way to scale up machine learning when the dataset is too big to fit on a single machine. By splitting data and computations across machines, you can speed up training significantly.
   * It's not suitable for all algorithms, but as you mentioned, algorithms that can be expressed as computing sums over the dataset (like linear regression) can benefit from it.
   * In the era of deep learning, frameworks like TensorFlow and PyTorch offer distributed training which abstracts away many of the complexities of data parallelism.

In conclusion, the choice of optimization technique, or a combination of them, will depend on the specific problem, the nature of the data, and the computational resources available. Understanding the underlying principles and trade-offs, as you've outlined, is key to making the right decisions.