Week 9

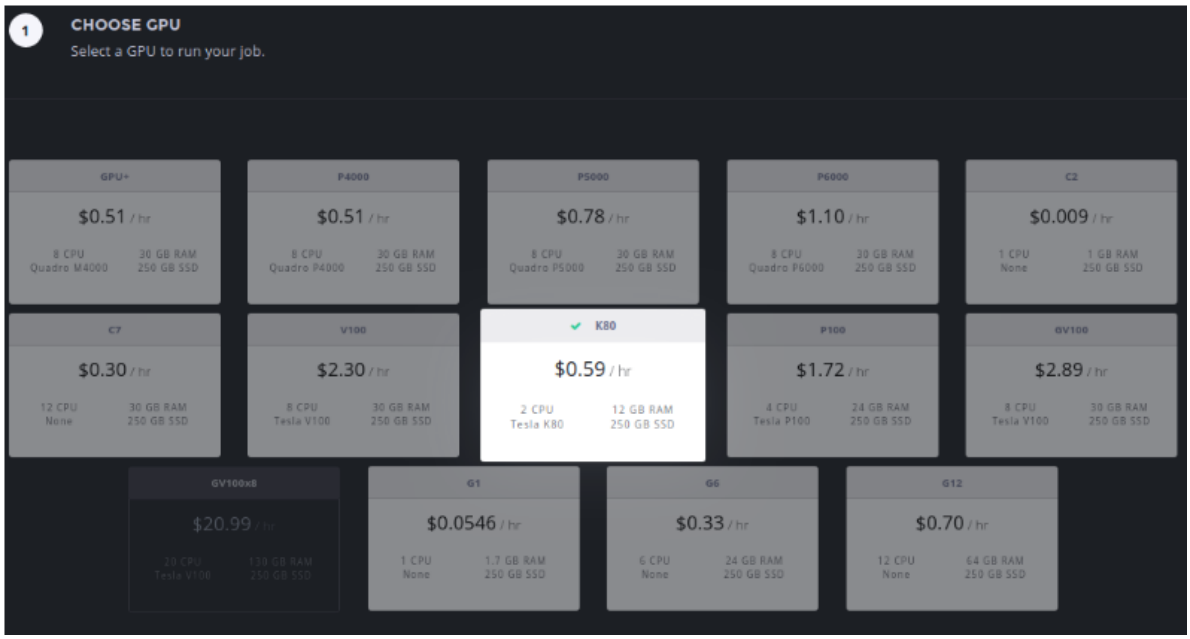
What has been done this week

Worked a bit on getting Paperspace gradient up and running and cleaning up some stuff. Also report writing. Lost some momentum in the project.

- [x] Work on Report
- [x] Revised Project Plan

Week	Activity	Risk
9	Report Writing	2
10	Report Writing + Embedding Network	4
11	Text Localization	4
12	Text localization	4
13	Combination of pipeline	3
14	Combination of pipeline	3
15	Embedded performance	2
16	Writing report	2
Extension? + 3 weeks		

- [x] More fiddeling with paperspace:
  - [x] Got the gradient option to work, now I can spin up any machine that I desire with my dataset available on all of them



- [x] Better folder structure on project, now each experiment is confined to its own folder:

```
+Models --From tensorflow/models
    +research
        +object_detection
+Pretrained_Models
    +rcnn
    +ssd
+Experiments
    +rcnn_alldata_reduced
        +pibeline.config
```

- [x] Write a guide on getting data onto Paperspace Gradient Paperspace didn't have any documentation on how to upload data to their new gradient service and after a lot of time, I finally figured out how. I wrote a guide on it:

  https://medium.com/@jacobjonhansen/getting-your-data-onto-paperspace-gradient-176c97fb176e

  Paperspace liked the guide, and gave me 50$ in credits for writing it

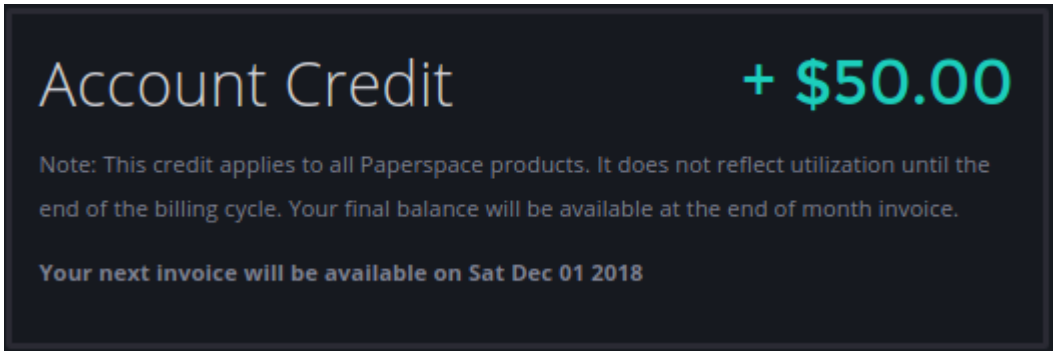
  Stan (Paperspace)
  Nov 12, 2:16 PM EST

  Hi Jacob,

  I'm glad you got this figured out. We depend on users like you who may have a better way to explain it to other users. I've gone ahead and put $50 in your account as I'm sure this will help other users as well.

  Best,

  -Stan



Literature

In Defense of the triplet loss for person re-identification: https://arxiv.org/abs/1703.07737

Essential paper on the triplet loss.

> A major caveat of the triplet loss, though, is that as the dataset gets larger, the possible number of triplets grows cubically, rendering a long enough training impractical. To make matters worse, f_θ relatively quickly learns to correctly map most trivial triplets, rendering a large fraction of all triplets uninformative. Thus mining *hard* triplets becomes crucial for learning. Intuitively, being told over and over again that people with differently colored clothes are different persons does not teach one anything, whereas seeing similarly-looking but different people (*hard negatives*), or pictures of the same person in wildly different poses (*hard positives*) dramatically helps understanding the concept of "same person". On the other hand, being shown *only* the hardest triplets would select outliers in the data unproportionally often and make f_θ unable to learn "normal" associations, as will be shown in Table 1. Examples of typi-

Introduces online triplet mining using the *Batch Hard* approach:

- Randomly sample P classes (person identities)
- From each sampled class, sample K images of each class
- Giving PK images in a batch
- Now for each sample a in the batch, select the hardest positive and hardest negative samples within the batch to calculate the loss.

$$\mathcal{L}_{\text{BH}}(\theta; X) = \overbrace{\sum_{i=1}^{P}\sum_{a=1}^{K}}^{\text{all anchors}} \Big[m + \overbrace{\max_{p=1\ldots K} D\left(f_\theta(x_a^i), f_\theta(x_p^i)\right)}^{\text{hardest positive}} - \underbrace{\min_{\substack{j=1\ldots P \\ n=1\ldots K \\ j\neq i}} D\left(f_\theta(x_a^i), f_\theta(x_n^j)\right)}_{\text{hardest negative}} \Big]_+, \quad (5)$$

- "Additionally, the selected triplets can be considered moderate triplets, since they are the hardest within a small subset of the data, which is exactly what is best for learning with the triplet loss"

They also have a Batch All approach, where they use all possible combination of triplets. This is shown to perform worse than the batch hard loss.

Distance measure is set to normal euclidean distance, as the L2 norm seemed to make training unstable at times. And in general provide no benefit.

They tried to train a network from a pretrained feature extractor (Here called TriNet, based on Resnet50, 22M params) and from a homemade one from scratch (here called LuNet, 5.5m Params)

	Type	Market-1501 SQ			Market-1501 MQ			MARS		
		mAP	rank-1	rank-5	mAP	rank-1	rank-5	mAP	rank-1	rank-5
TriNet	E	**69.14**	**84.92**	**94.21**	**76.42**	**90.53**	**96.29**	**67.70**	**79.80**	**91.36**
LuNet	E	60.71	81.38	92.34	69.07	87.11	95.16	60.48	75.56	89.70
IDE (R) + ML ours	I	58.06	78.50	91.18	67.48	85.45	94.12	57.42	72.42	86.21
LOMO + Null Space [45]	E	29.87	55.43	-	46.03	71.56	-	-	-	-
Gated siamese CNN [39]	V	39.55	65.88	-	48.45	76.04	-	-	-	-
CAN [25]	E	35.9	60.3	-	47.9	72.1	-	-	-	-
JLML [22]	I	65.5	85.1	-	74.5	89.7	-	-	-	-
ResNet 50 (I+V)† [52]	I+V	59.87	79.51	90.91	70.33	85.84	94.54	-	-	-
DTL† [10]	E	41.5	63.3	-	49.7	72.4	-	-	-	-
DTL† [10]	I+V	65.5	83.7	-	73.8	89.6	-	-	-	-
APR (R, 751)† [24]	I	64.67	84.29	93.20	-	-	-	-	-	-
Latent Parts (Fusion) [20]	I	57.53	80.31	-	66.70	86.79	-	56.05	71.77	86.57
IDE (R) + ML [55]	I	49.05	73.60	-	-	-	-	55.12	70.51	-
spatial temporal RNN [57]	E	-	-	-	-	-	-	50.7	70.6	90.0
CNN + Video† [46]	I	-	-	-	-	-	-	-	55.5	70.2
TriNet (Re-ranked)	E	**81.07**	**86.67**	**93.38**	**87.18**	**91.75**	**95.78**	**77.43**	**81.21**	**90.76**
LuNet (Re-ranked)	E	75.62	84.59	91.89	82.61	89.31	94.48	73.68	78.48	88.74
IDE (R) + ML ours (Re-ra.)	I	71.38	81.62	89.88	79.78	86.79	92.96	69.50	74.39	85.86
IDE (R) + ML (Re-ra.) [55]	I	63.63	77.11	-	-	-	-	68.45	73.94	-

Table 2: Scores on both the Market-1501 and MARS datasets. The top and middle contain our scores and those of the current state-of-the-art respectively. The bottom contains several methods with re-ranking [55]. The different types represent the optimization criteria, where I stands for identification, V for verification and E for embedding. All our scores include test-time augmentation. The best scores are bold. †: Concurrent work only published on arXiv.

Triplet Loss: https://omoindrot.github.io/triplet-loss

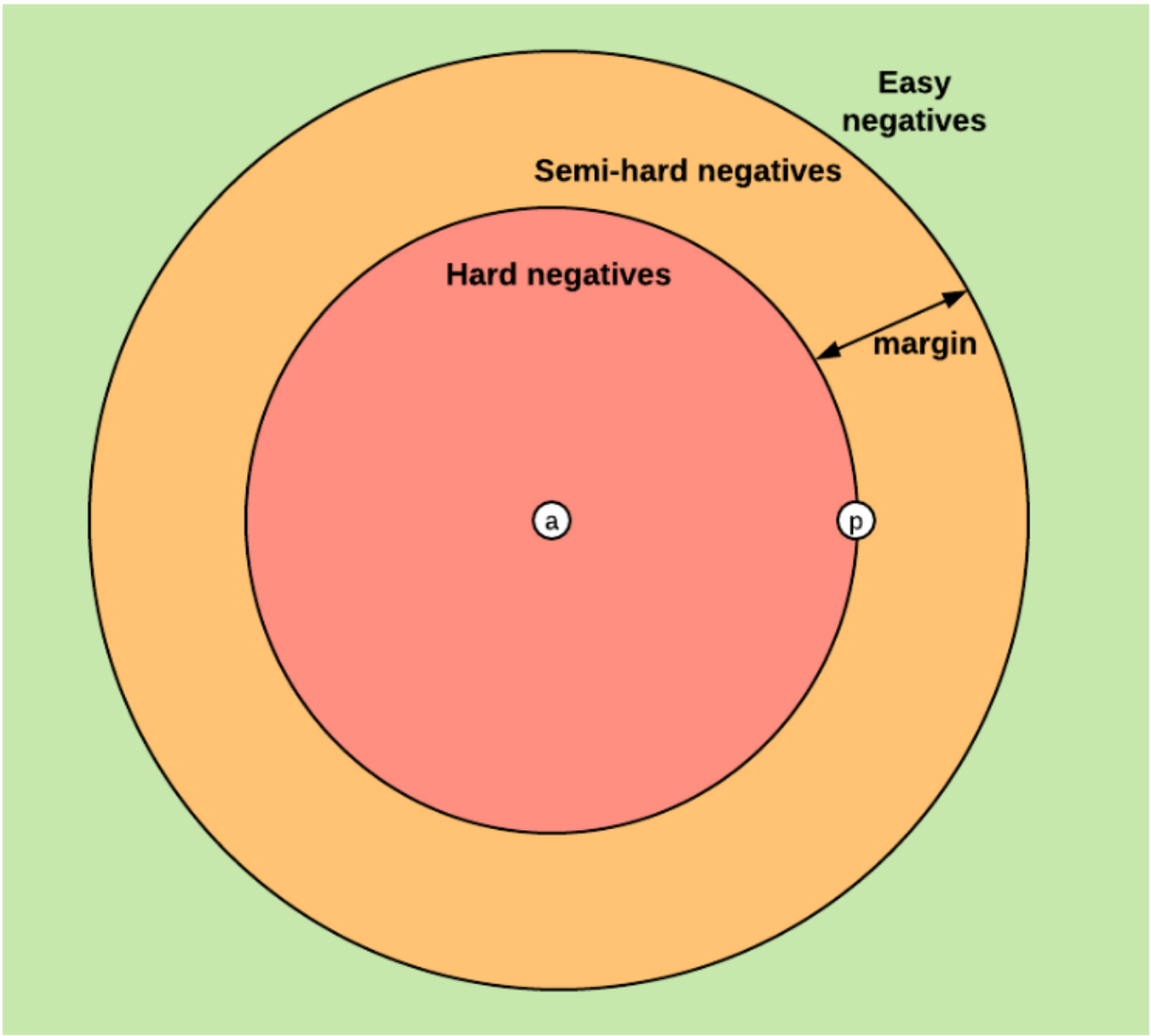
Triplet mining

Based on the definition of the loss, there are three categories of triplets:

- **easy triplets**: triplets which have a loss of 0, because $d(a, p) + margin < d(a, n)$
- **hard triplets**: triplets where the negative is closer to the anchor than the positive, i.e. $d(a, n) < d(a, p)$
- **semi-hard triplets**: triplets where the negative is not closer to the anchor than the positive, but which still have positive loss: $d(a, p) < d(a, n) < d(a, p) + margin$

Each of these definitions depend on where the negative is, relatively to the anchor and positive. We can therefore extend these three categories to the negatives: **hard negatives**, **semi-hard negatives** or **easy negatives**.

The figure below shows the three corresponding regions of the embedding space for the negative.



The three types of negatives, given an anchor and a positive

Hidden Technical Debt in Machine Learning Systems https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

- Not all debt is bad, but all debt needs to be serviced
- ML systems have a special capacity for incurring technical debt, because they have all of the maintenance problems of traditional code plus an additional set of ML-specific issues. This debt may be difficult to detect because it exists at the system level rather than the code level.
- Unfortunately, it is difficult to enforce strict abstraction boundaries for machine learning systems by prescribing specific intended behavior. Indeed, ML is required in exactly those cases when the desired behavior cannot be effectively expressed in software logic without dependency on external data. The real world does not fit into tidy encapsulation.
- **Entanglement.** Machine learning systems mix signals together, entangling them and making isolation of improvements impossible. For instance, consider a system that uses features x_1 , ...x_n in a model. If we change the input distribution of values in x_1 , the importance, weights, or use of the remaining n_1 features may all change. | No inputs are ever really independent. We refer to this here as the CACE principle: Changing Anything Changes Everything.
- **Correction Cascaes** There are often situations in which model m a for problem A exists, but a solution for a slightly different problem A ' is required. In this case, it can be tempting to learn a model m'a that takes m a as input and learns a small correction as a fast way to solve the problem. However, this correction model has created a new system dependency on m'a , making it significantly more expensive to analyze improvements to that model in the future. The cost increases when correction models are cascaded, with a model for problem A '' learned on top of m'a , and so on, for several slightly different test distributions.
- **Undeclared consumers**. If your software is not access protected, some other guy in the company will use your model output for something. So suddenly changing your model will mess up other parts of the company without you even being aware of it.

Underutilized data dependencies can creep into a model in several ways.

- **Legacy Features.** The most common case is that a feature F is included in a model early in its development. Over time, F is made redundant by new features but this goes undetected.
- **Bundled Features.** Sometimes, a group of features is evaluated and found to be beneficial. Because of deadline pressures or similar effects, all the features in the bundle are added to the model together, possibly including features that add little or no value.
- ϵ**-Features.** As machine learning researchers, it is tempting to improve model accuracy even when the accuracy gain is very small or when the complexity overhead might be high.
- **Correlated Features.** Often two features are strongly correlated, but one is more directly causal. Many ML methods have difficulty detecting this and credit the two features equally, or may even pick the non-causal one. This results in brittleness if world behavior later changes the correlations.

Underutilized dependencies can be detected via exhaustive leave-one-feature-out evaluations. These should be run regularly to identify and remove unnecessary features.

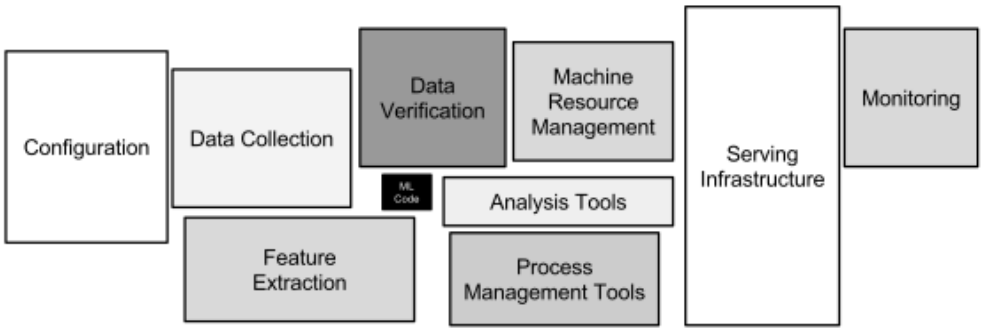
- 

Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Glue Code. ML researchers tend to develop general purpose solutions as self-contained packages. A wide variety of these are available as open-source packages at places like `mloss.org`, or from in-house code, proprietary packages, and cloud-based platforms.

- Using generic packages often results in a *glue code* system design pattern, in which a massive amount of supporting code is written to get data into and out of general-purpose packages. Glue code is costly in the long term because it tends to freeze a system to the peculiarities of a specific package; testing alternatives may become prohibitively expensive. In this way, using a generic package can *inhibit* improvements, because it makes it harder to take advantage of domain-specific properties or to tweak the objective function to achieve a domain-specific goal. Because a mature system might end up being (at most) 5% machine learning code and (at least) 95% glue code, it may be less costly to create a clean native solution rather than re-use a generic package.

An important strategy for combating glue-code is to wrap black-box packages into common API's. This allows supporting infrastructure to be more reusable and reduces the cost of changing packages.

- **Pipeline jungles**: Without care, the resulting system for preparing data in an ML-friendly format may become a jungle of scrapes, joins, and sampling steps, often with intermediate files output. Man- aging these pipelines, detecting errors and recovering from failures are all difficult and costly
- **Dead Experimental Codepaths**. A common consequence of glue code or pipeline jungles is that it becomes increasingly attractive in the short term to perform experiments with alternative methods by implementing experimental codepaths as conditional branches within the main production code. Testing all possible interactions between codepaths becomes difficult or impossible. A famous example of the dangers here was Knight Capital's system losing $465 million in 45 minutes, apparently because of unexpected behavior from obsolete experimental codepaths [15].

What to do next week

- [ ] Train FaceNet-ish network (**tuesday**)
- [ ] Write section on Detection network (**wednesday**)
- [ ] Write section on golf club terminology (**wednesday**)