Multi-task learning: what is it, how does it work and why does it work?

Adam K · Follow
Published in GumGum Tech Blog
16 min read · Nov 11, 2022

Listen Share More

Doing two things at once can make DL models better at both.

One of the most exciting and seemingly ubiquitous recent topics in deep learning (DL) is without a doubt multi-task learning. Multi-task learning (MTL) is a model training technique where you train a single deep neural network on multiple tasks at the same time. Though it may seem a little counter-intuitive at first — shouldn't learning two problems be harder than learning one? — MTL turns out to often improve performance of the final model compared to a single-task variant (depending on the tasks, of course), which is why it's becoming more and more popular in deep learning.

In fact, a variant of MTL, transfer learning, where a model is trained first on a large general dataset and then tuned on data specific for a particular task has become the standard for both natural language processing (NLP) with BERT, GPT-3 and Google's newest model, Multitask Unified Model, and computer vision (CV) with models such as EfficientNet. We'll discuss exactly why that might be in a little bit, but the writing is on the wall. MTL is pretty cool.

It was these aspects of MTL that first inspired me to check it out. On the Verity team at GumGum, we build models to understand the content of webpages **without using cookies or personal data**. This means that we need to deliver accurate classification while being robust to the wide variety of webpages out there on the wild, wild, world-wide web. In order to keep at the top of the game, we are always looking for ways to improve the performance of our brand-safety and contextual understanding models, and MTL seemed like a great technique to add to our repertoire.

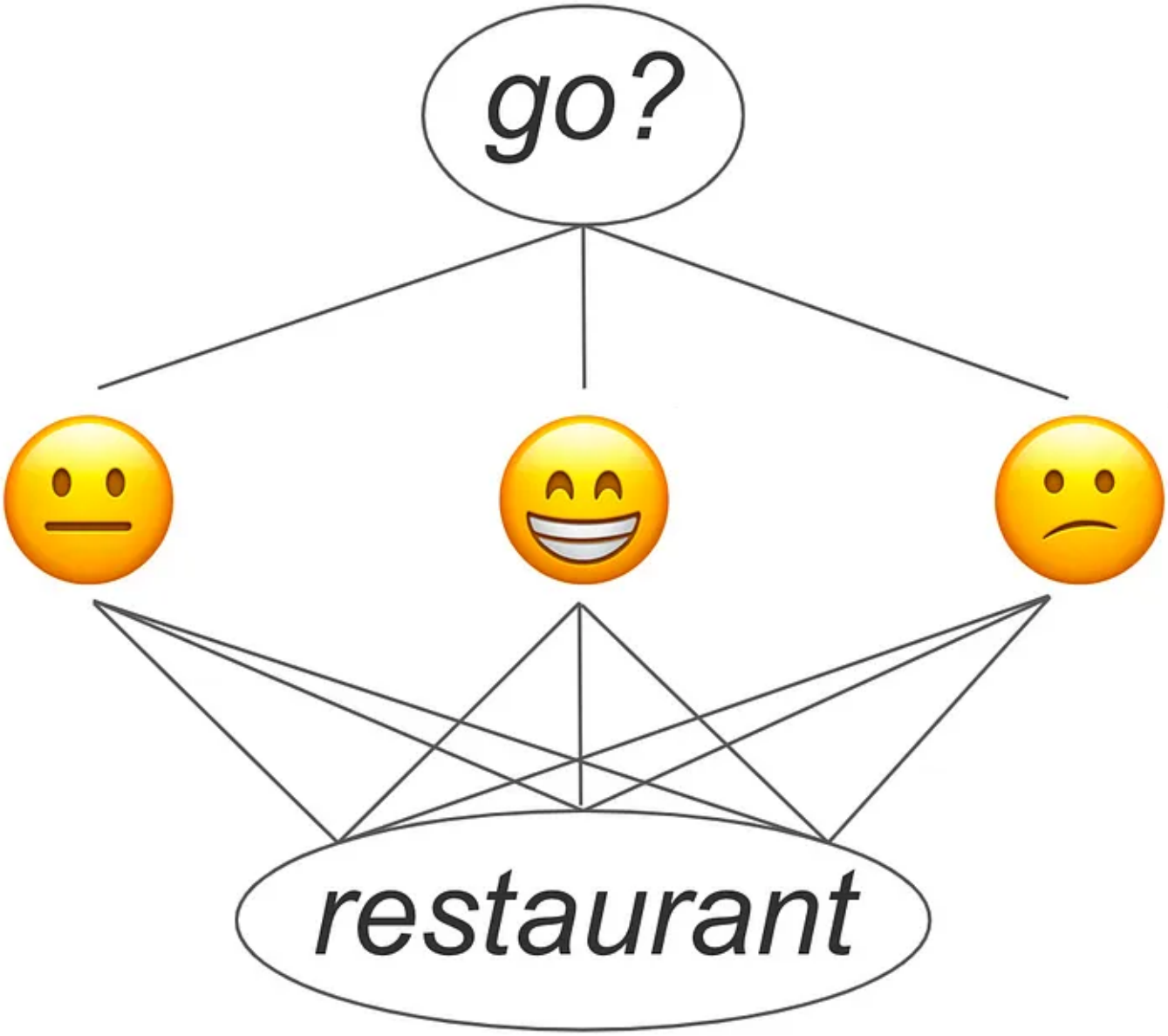
One of our core models is a brand-safety model, an NLP model that detects if there is any threatening content on a web page (including the rough and tumble jungle of Tumblr comments!), such as hate speech, violence or explicit material. We recently updated our labeling scheme to be more in line with GARM standards. Though it's always fun to collect and annotate a whole new set of training data, this left our training data for the previous iteration of the model just sitting there potentially unused. **By leveraging MTL, we were able to take advantage of the data for a previous iteration of the model and get a sizable boost over models trained with only the new data.** At least anecdotally, I can say that MTL works!

In this blog post, I want to talk about how exactly MTL works and then look at an example of it in action to see if we can piece apart exactly how MTL training might affect the model itself.

How does deep learning work in a nutshell?

To start off, let's talk about how deep learning works at a very high level. Imagine there's a new restaurant in town and you're wondering if it's a place that you might like. Now, you've never been there yourself, but you have three friends who have. You ask all their opinions on the place, and one says it's okay, the other says it's not worth it and the third says that it's the best food ever. You go to the restaurant and it's terrible, so

terrible that you get food poisoning. This is definitely going to affect your restaurant decision-making for the future.



Well, next time you're deciding whether to check out a new restaurant, you're likely going to trust the input of the friend who told you to stay away a little more and trust the friend who insisted the food was divine a whole lot less. If you were to repeat this paradigm over and over for many, many restaurants, eventually you'd *learn* exactly how much you should trust each friend. In other words, over several *iterations,* you'd know the *optimal weight* to put on each friend's input such that you could predict with a good degree of certainty whether you'd like a restaurant just from what they've told you. How does this have anything to do with MTL or deep learning in the slightest?

This is actually a very simple, but useful analogy for how deep learning works, without needing all the intimidating language like backpropagation, gradients and neurons. Putting it into a little more algorithm-y but still friendly format:

1. First, we need observations in the world, i.e., *input data*. In this case, the data are visits to the restaurant, but it could be images, texts, tabular data, whatever. Each data point needs a *label* on it, e.g., this restaurant was good, this image has a dog in it, so that the model can eventually learn to associate input data with its proper label.

2. Then, a series of intermediaries examine the data and make decisions on it using their own weights for different features of the input. In this example, these are your restauranteur friends who visit a restaurant and make their own decisions.

3. Next, those intermediary decisions are evaluated by a final decision function and a prediction is made. In this example, you are the final decider, and you make your decision by how much you trust each of your friends' recommendations to try out a new place to eat.

4. Finally, we learn. After making your prediction, you change the network based on how close your prediction was to the true label for the data. In this example, this is trusting your friends less who encouraged you to try a place where you got food poisoning.

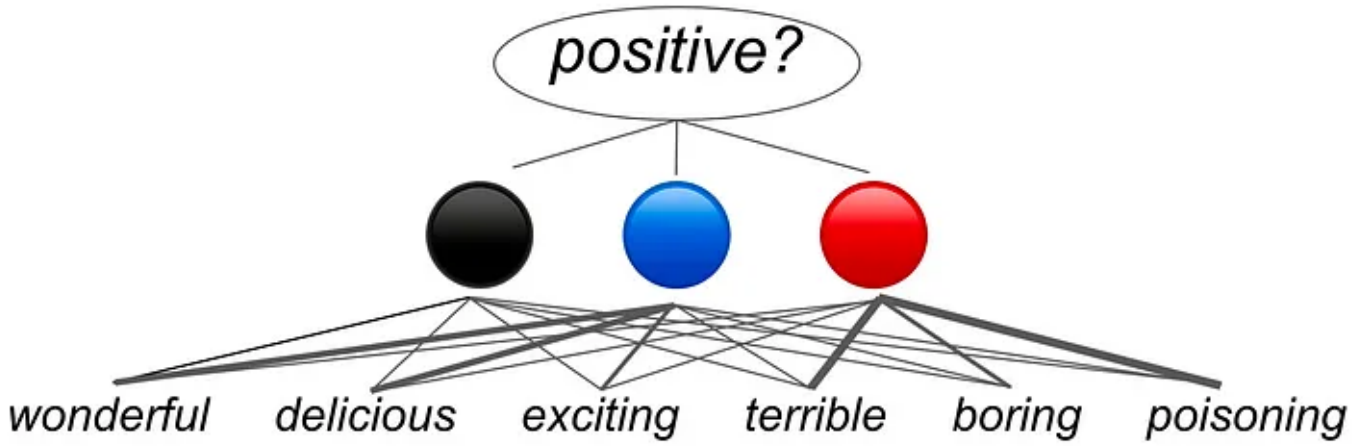
This general framework describes deep learning, albeit very broadly: the hidden layers learn interesting patterns in the input (your friends), while the final layer (or final layers) learn how to use those interesting patterns to solve a particular problem the network has been trained on (your trust of each of your friends). Check out this for a simple but complete walkthrough of a small example network.

How does multi-task learning work in a nutshell?

Now, imagine a similar problem but instead of evaluating restaurants, you're evaluating movies. That is, rather than asking your friends their opinions on whether you should visit a restaurant, you're asking them their opinions on a new show or

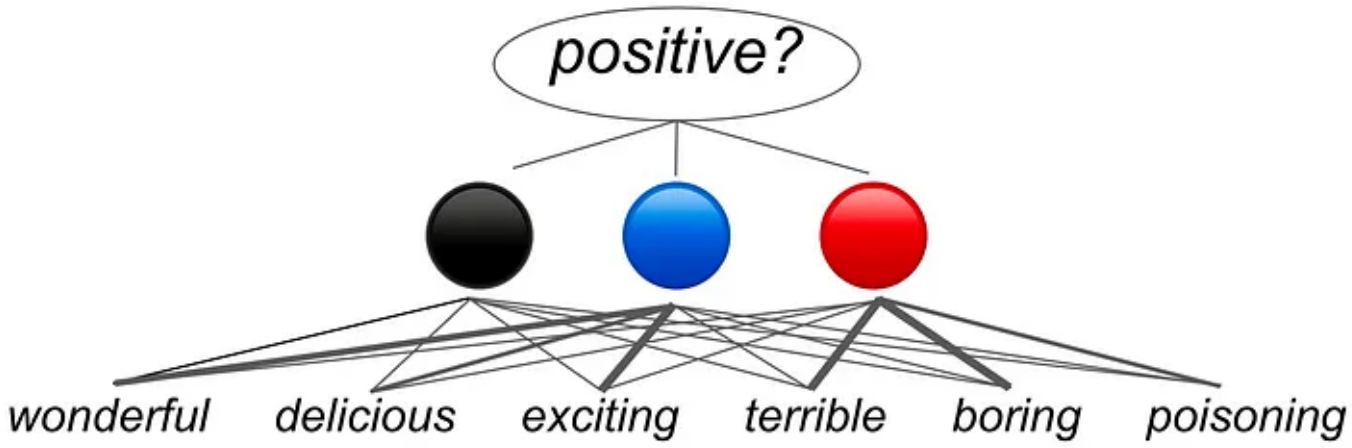
movie to watch. Now, you could start over from the beginning, find three other friends and gradually learn how much to trust each of their input. But what if there was another way? What if you asked the three restaurant-friends their input on movies, as well as restaurants, at the same time? Sure, exactly how you interpret their inputs would vary from task to task — from restaurants to movies—but there should still be a good deal of overlap in what kind of information each of your friends is giving you. Some friends you have similar taste with and some you very much don't.

To move to another little more concrete, although still hypothetical example, imagine if you were training an NLP model to predict whether a Yelp review for a restaurant was positive or negative. Regardless of the actual architecture of the model, one of the first things it's going to do is to learn to associate particular words or phrases with each label; "*wonderful*" and "*delicious*" are positive, "*terrible*" and "*food poisoning*" are negative.



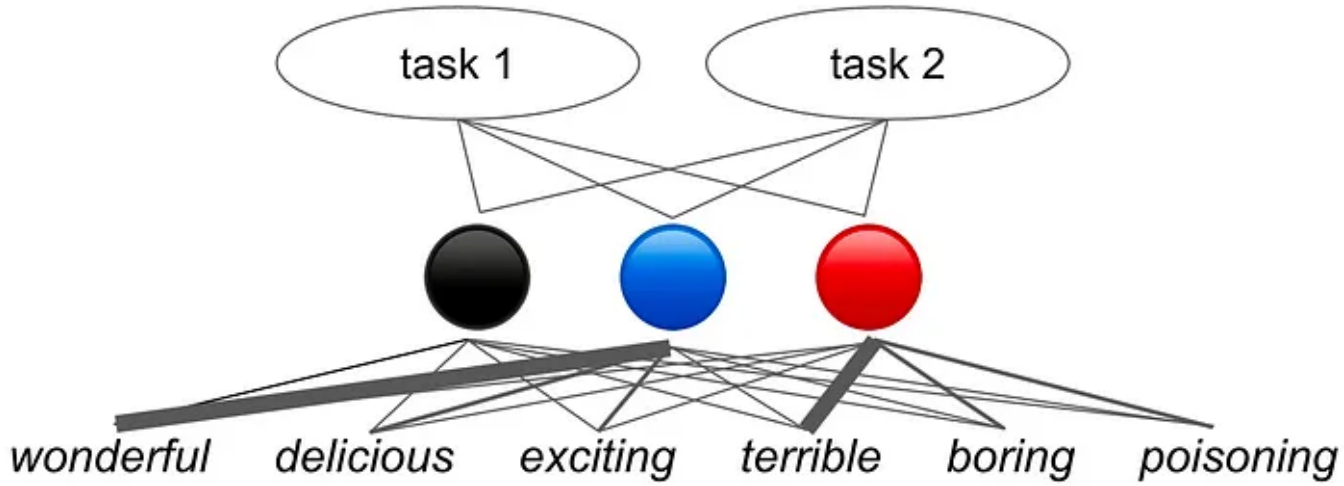
Single-task network trained on restaurant reviews.

If we were to do the same, but with IMDB reviews of movies, we'd expect the something similar but slightly different. That is, the model would still learn to associate words and phrases with positivity and negativity, but those associations would be slightly different than with Yelp; "*wonderful*" and "*exciting*" are positive, "*terrible*" and "*boring*" are negative.



Single-task network trained on movie reviews.

As you might expect, the exact kind of language that correspond with positive restaurant reviews and positive movie reviews won't be exactly the same, but still there should be a non-trivial amount of overlap. Recall that a neural network's training consists of the deep layers learning interesting patterns in the input and the final layer learning to use those interesting patterns for a specific task.



Multi-task network trained on both tasks. Each task has its own final decision function based on the hidden middle layers.

Now, what would happen if we were to train a network to perform both tasks simultaneously? That is, rather than have a single *decision function*, or way of interpreting the hidden layers output, you have two, each trained on a specific task. This would encourage the hidden layers to find patterns that are more generally useful for both tasks. In other words, we would be training a model that focuses on generally positive or negative language, rather than language focused on restaurants or movies or uninteresting idiosyncrasies of either dataset. This would create a more robust model, one better at avoiding the common pitfalls of over-fitting and that better generalizes to new data.

Experiment!

Thus far, I've talked in generalities and hypotheticals. Let's try this out in practice! What we'll do is compare the performance for two classification tasks when trained as individual models and when combined into a single multi-task model. A fair warning, because I'll be talking about the models more in depth, I will be using some more technical terms. I've done my best, however, to provide graphs and to frame everything in a way that you don't need to understand the fine-grained ML details to understand the main takeaways.

Now, what data will we use? You guessed, I'll be predicting positive or negative labels and for Yelp reviews and the same for IMDB reviews. I've conspicuously chosen very similar tasks with very similar data because, as I mentioned at the beginning, the nature of each task has a big influence on the final performance of the model as a whole. **One really big thing to point here is that these are completely different datasets!** That is, these are not the same texts labeled for both movie and restaurant sentiment (what would that even look like?) but completely different data, collected and labeled separately. The fact that MTL can use different datasets like this is exactly how we were **able to leverage data labeled with a different labeling scheme to increase the performance of our newest brand-safety model without needing to re-annotate the older data.**

As for the model architecture that I'll be fitting, I will use a simple bag-of-words representation for each type of review. The bag-of-words will be passed to a single

fully-connected layer which will then connect to a single final output layer for each single-task model and to two final layers for the multi-task model. Just so it's clear, the difference between the architecture for a single-task model and the multi-task model is that the multi-task model shares all penultimate layers and only diverges at the end.

Let's talk a little bit about the input layer. To make everything comparable, I'll limit the possible words for each model to those that appear at least once in both datasets. This will make the models more comparable down the line. Now, let's talk about the hidden layer. I'm going to be using a simple sigmoid activation function, rather than another function like TanH or ReLU. Granted, this architecture is by no means the state-of-the-art or likely to be the most performant model, but I've chosen this kind of model because it will let us dissect exactly what's going on under the hood in a way that an LSTM or Transformer wouldn't.

If you're curious about the details of implementation or to see the code so you can try this yourself, stay tuned! That's coming in another blog post soon.

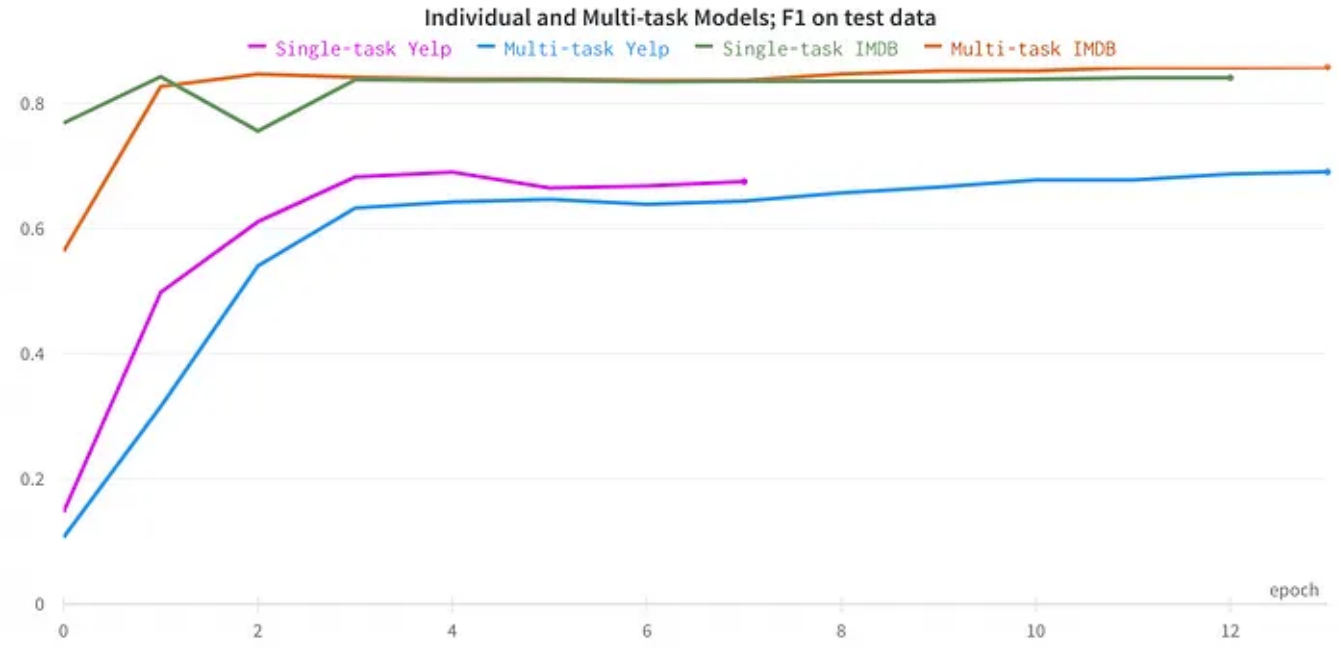
Three views into a MTL model

Now, it's notoriously difficult to see exactly what a DL model has *learned*. However, we can look at certain aspects of the MTL model and see how they differ than the single-task variants. For good measure, I'll look from three different angles to give a good general idea of what's going on. First, we'll look at aggregate metrics across for the data and training as a whole. Then, we'll look at the actual learned weights and see if we can piece apart what the multi-task model has learned. Finally, we'll check how well the multi-task model is able to generalize what it's learned, compared to either of the single-task models.

Performance on held-out data for single-task and multi-task models

The first and most straightforward way to look at how multi-task learning affects training is to look at overall metrics for the datasets. Here are the plots of F1 on a held-out test set for both the Yelp and IMDB data:

Not too bad. Again, remember we're not trying to get the best performance possible on these datasets; we're simple trying to set a baseline for each task where a different model is trained for each. When a multi-task model with the same basic architecture, i.e., a single hidden layer with the same number of neurons as each individual model, is trained with both datasets, we see a small improvement!



Granted, it's nothing to write home about (it's around a .01 increase for both tasks) but it's striking to see that a single model can contain the information needed to classify

two types of data without significant loss. **Regardless, this shows that the single hidden layer in the multi-task model has learned how to take the bags-of-words from the input for both texts and find the patterns that are useful for both tasks.** Now, let's look at exactly what those are.

Input feature importance

Let's take a step to the side and instead of looking at the performance of the model as a whole, let's see if we can go inside the model and see exactly what both the single-task models and the multi-task model have learned. What we're looking for is some signal that the multi-task model has learned to pay attention to features that are important to both tasks and, importantly, the features it is paying attention to make sense, given the task. It's a time for some common sense, if nothing else.

We want to see if the multi-task model has learned to weigh words common to both tasks higher than the single-task models. Whereas the single-task Yelp and IMDB models can focus on food-related and film-related terms respectively, the multi-task model should learn to focus more on general positive or negative words. If this were a standard ML model like logistic regression, we could look directly at the values of the weights assigned to each word in the model's input. However, since this is a *deep* learning model, it's not so easy. Luckily, though, since this model is so simple, we can try and infer something similar.

So, remember how I mentioned that I chose the architecture for the models we're looking at for very particular reasons? Here's where those reasons come to light. The final output of one of these models is a sigmoid applied to the weighted sum of activations from the neurons of the single hidden layer. the activation values will generally be a 0 or 1, since the hidden neurons are all sigmoids themselves whose input come from weights assigned to each input neuron, i.e., each word in our bag-of-words representations per datum.

Granted, it's very difficult to piece apart exactly how much impact a word might have on the final prediction, but it's a fair assumption that if a word in the input leads to more active neurons in the hidden layer, that word is contributing more to the final prediction. In other words, the simplistic design of these models allows us to estimate overall impact of each word from the input, despite this being a deep learning problem.

Here are the five words from the single-task Yelp model that have the greatest sum of weights for their connections to the hidden layer:

1. *awesome*
2. *loved*
3. *excellent*
4. *great*
5. *waste*

Pretty interesting! These words are clearly relevant for a sentiment task but "waste", for example, might be something only relevant for restaurants. No one wants to go somewhere where they waste food, and I can see plenty of negative reviews having a sentence like "don't *waste* your time by going to ...."! Let's now look at the highest weighted words for the single-task IMDB task:

1. *struggling*
2. *astounding*
3. *asleep*
4. *usa*
5. *designed*

This set is more clearly geared towards movies, while still being related to sentiment. I guess you could argue that "astounding" would fit for both, but with all of Hollywood's campy political dramas about American government, "USA" is certainly movie-focused. The next question is, what do the top words for the multi-task model look like?

1. *awesome*
2. *poor*
3. *loved*

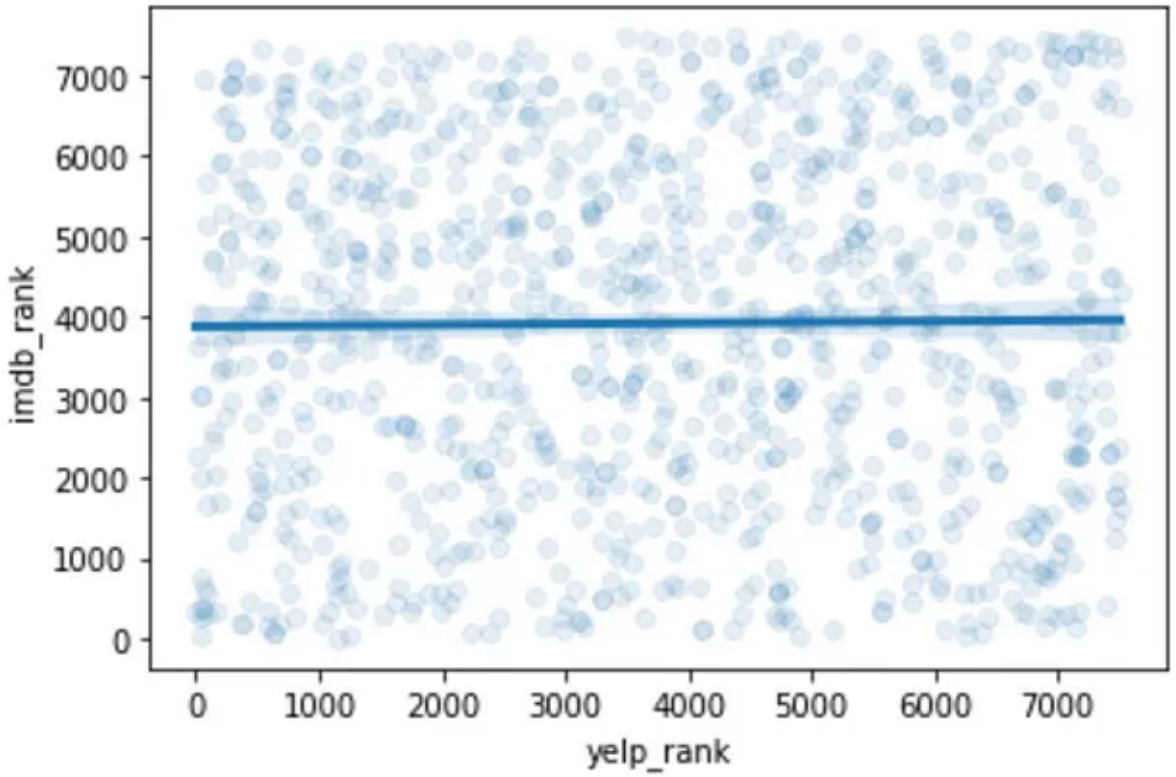
4. *amazing*

5. *delicious*

Now we've got a set that's a little less tuned to either task but still focused on sentiment! Granted, "delicious" is more food-based but "awesome", "poor", "loved" and "amazing" work equally well for both tasks. We're not done though.

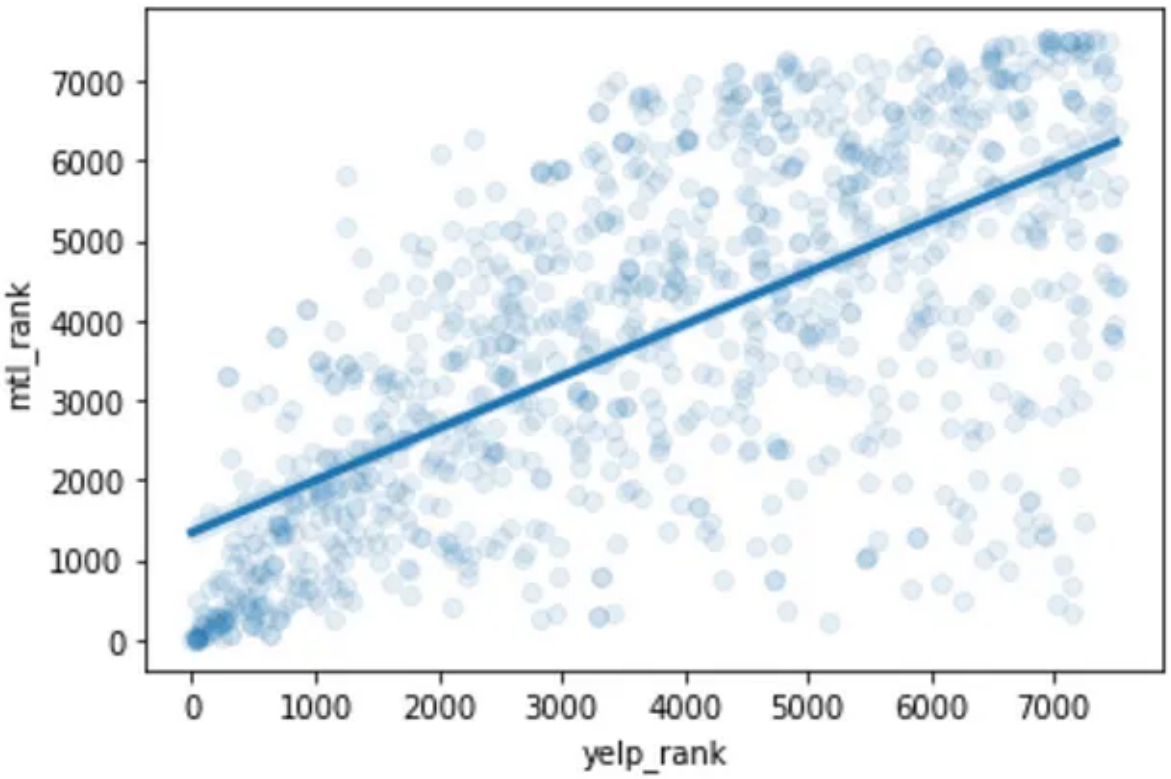
We might have just gotten lucky looking at a few top words and this is a bit subjective example. As a more objective test, we can take the words for both single-task models and see where they fall in the other single-task model's ranks. What we'd expect is that the highly-ranked words for the individual single-task models would be not-so-highly ranked for the other single-task's model. For instance, *struggling* is the most informative word for IMDB but the 5527th word for Yelp, while *awesome* is a top word for Yelp while being the 334th word for IMDB. However, since the multi-task model is learning important features for both tasks, the words for the individual tasks should move up the ranking for the multi-task model. In the multi-task model, *struggling* is now the 108th most important word, a big step up from 5527. **Explicitly, what we expect is there to be a linear relationship between the ranks of words in either single-task model with the multi-task model, but no relationship between the single-task models themselves.**

Below is a scatter plot of the ranking for word importance for both the single-task Yelp and IMDB models. Interestingly, there seems to be no relationship overall between the importance of a word in one task for the other task. Put another way, there's little overlap in the kinds of words the single-task Yelp model and the IMDB model are using to make their decisions.



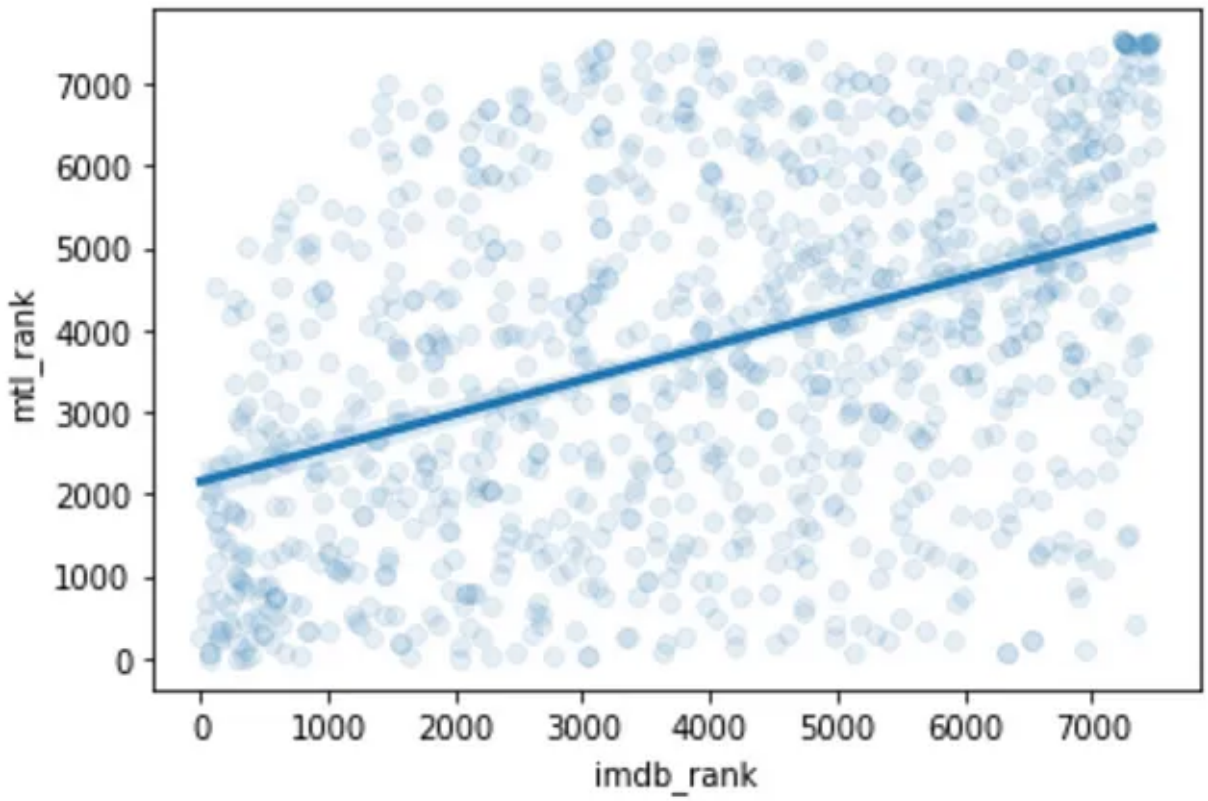
Relationship between feature importance in the single-task Yelp model and single-task IMDB model.

Let’s look at a similar plot but comparing the ranks of words in the Yelp model and the multi-task model.



Relationship between feature importance in the single-task Yelp model and the multi-task model.

What we have here is a very clear linear trend; words that are highly ranked and important for the single-task Yelp model are also important for the multi-task model. The same is true for IMDB. What this shows is a clear overlap in the kinds of words the multi-task model is focusing on and that of the single-task models, a pattern not found when comparing the two single-task models directly. Put another way, the multi-task is, in fact, learning the words most important for both tasks, as we expected and hoped.

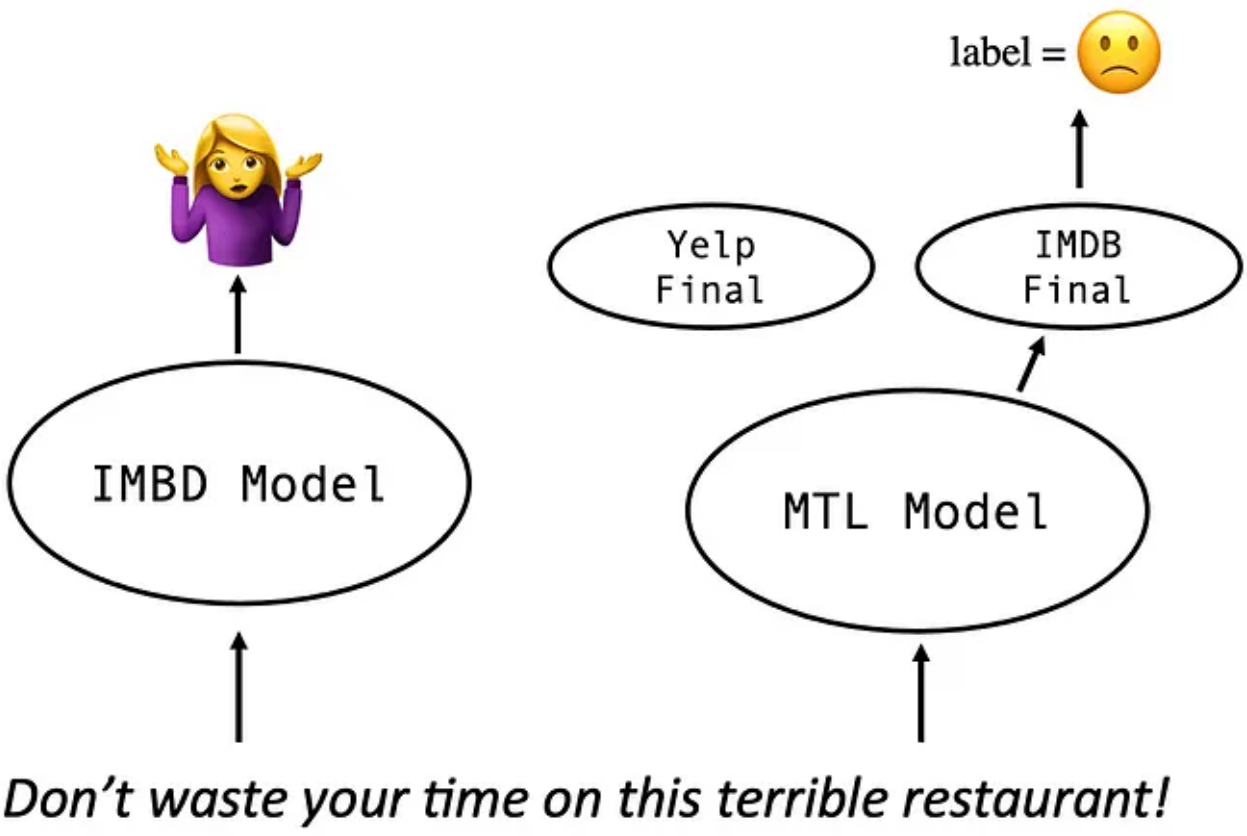


Relationship between feature importance in the single-task IMDB model and the multi-task model.

Cross-task performance

Once again, we return to my clever choices when designing this exposition of MTL (bravo, me). Because the two tasks are very similar, the information that each single-task model looks at in the input should overlap with the other, i.e., what are generally positive and negative words? The multi-task model should, then, capture generally applicable features of the input, because it's tasked with a more general understanding of positive and negative reviews.

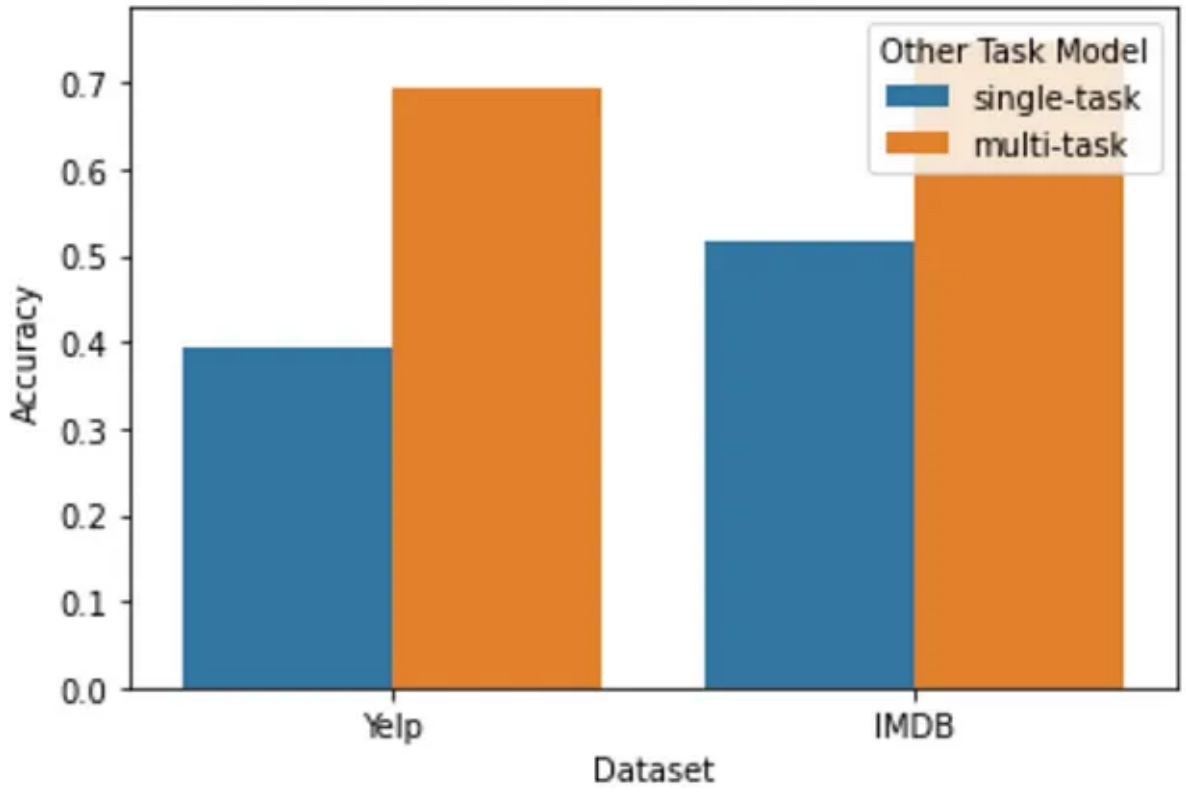
How can we test this? If we were to pass the data from the Yelp task, for instance, through the single-task model trained on IMDB reviews, we would expect okay performance. The model is trained to look at generally positive or negative language, but not tailored to restaurant reviews. On the other hand, if we were to pass the Yelp data through the multi-task model *but take the output of the final layer trained on IMDB data*, the output of the IMDB final layer should be a little better. Granted, it won't be as good as a model trained on Yelp model per se, but because the multi-task model has a more general understanding of sentiment in its hidden layers, its predictions should better match the gold labels for Yelp data. Just so it's clear, **we'll make predictions using data the final layer of the model has NOT been trained on.**



The final layer of the multi-task model for IMDB should perform better on Yelp data since the hidden layers have learned to pull out general sentiment features from the input.

Keep in mind, the final layer of the IMDB task has ***never seen*** any Yelp data; it has never had any gradients or updates to the weights based on those data during training. The multi-task model has, though, tuned its hidden layer for these kinds of data and so should capture more useful information that can be applied to a different task. Pretty easy and straightforward test, huh?

Passing the test set for Yelp through the single-task IMDB model yields a 40% accuracy (it's a simple task, I'm going to be a slovenly data scientist and use accuracy instead of cooler metrics). However, passing Yelp test data through the multi-task label and using the final layer for IMDB boosts accuracy to 70%! Similarly, IMDB as interpreted by the single-task Yelp model has an accuracy of 52%, just above chance. This is boosted to 75% when predicted by the MTL model and using the final layer for the Yelp task. In both cases, it's a clear lift, putting the writing on the wall: **the MTL model has learned how to interpret positive or negative language more generally than its single-task counterparts.**



Accuracy when passing the test data from either task through the other single-task model or other task's final layer in the MTL model.

Putting it together

In this blog post, I've discussed exactly what MTL is and why it should, in theory, give a boost to a machine learning task. I've also used a simple problem and a MTL solution to it as a demonstration of both overall performance gains for the model externally and how it affects what the model learns itself. From this small experiment, it does seem clear that the excitement of MTL in the machine learning zeitgeist is certainly earned and I encourage you, dear reader, to explore a model that combines multiple tasks in your next project. If you have the patience, stay tuned in the next few weeks for another blog post where I'll provide a barebones skeleton of MTL for PyTorch and PyTorch Lightning that you can use for yourself!

We're always looking for new talent! View jobs.

Follow us: Facebook | Twitter | LinkedIn | Instagram

NLP Machine Learning Multi Task Learning Data Science Classification

Follow

Written by Adam K

15 Followers · Writer for GumGum Tech Blog

More from Adam K and GumGum Tech Blog

	toxic	severe_toxic	obscene	threat	insult	identity_
toxic	1.000000	0.313512	0.669702	0.132540	0.640426	0.23
_toxic	0.313512	1.000000	0.429728	0.073953	0.397987	0.16
scene	0.669702	0.429728	1.000000	0.118247	0.760913	0.25
threat	0.132540	0.073953	0.118247	1.000000	0.137142	0.10
insult	0.640426	0.397987	0.760913	0.137142	1.000000	0.28
y_hate	0.237735	0.168355	0.258591	0.100562	0.286363	1.00

Adam K in GumGum Tech Blog

Creating a balanced multi-label dataset for machine learning

Streamlining your Kubernetes adoption with Helmfile / ArgoCD and GitOps

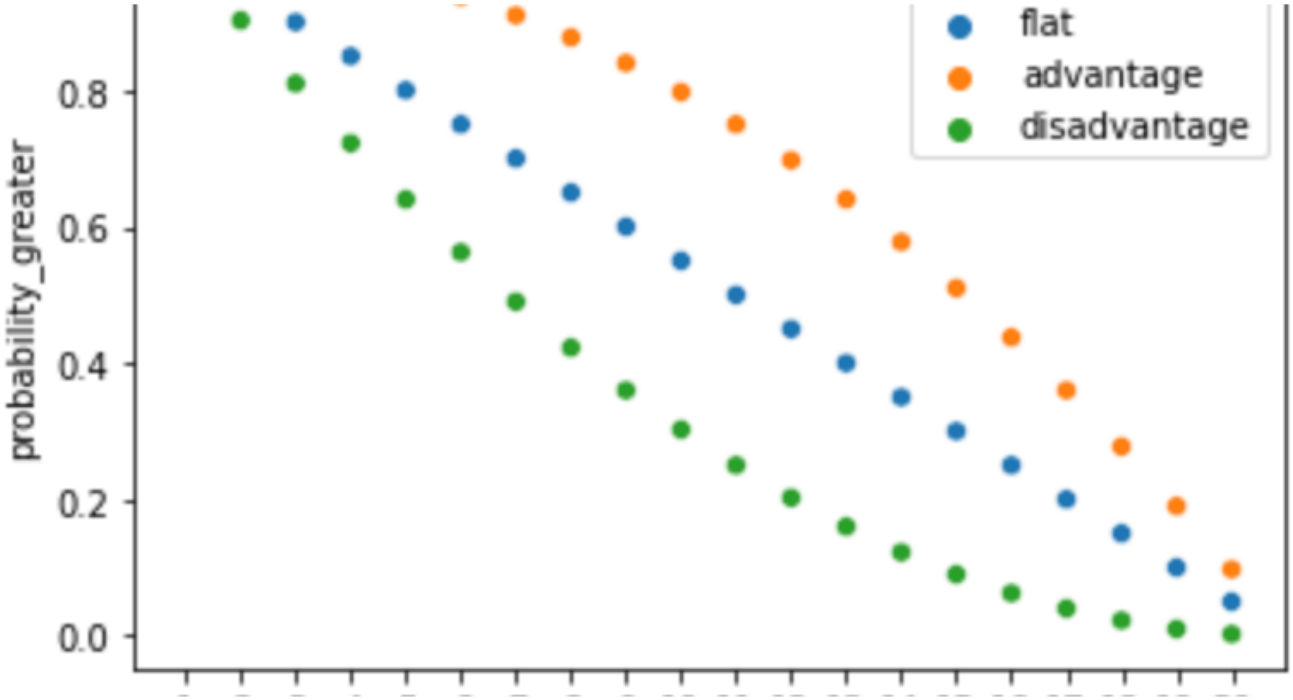
In a previous article titled Stop being selfish !—Open up Terraform to your team with Atlantis we showcased how powerful and convenient...

Ishan Shrivastava in GumGum Tech Blog

Handling Class Imbalance by Introducing Sample Weighting in the Loss Function

"Nobody is Perfect" This quote not just applies to us humans but also the data that surrounds us. Any data science practitioner needs to...

8 min read · Dec 16, 2020



Adam K

How advantageous is Advantage in D&D 5e?

Man, I love D&D 5e. I got my first set back at the end of the TSR days and I've been playing regularly since early in the 4e days and I can...

8 min read · Sep 9, 2022

See all from Adam K

See all from GumGum Tech Blog

Recommended from Medium

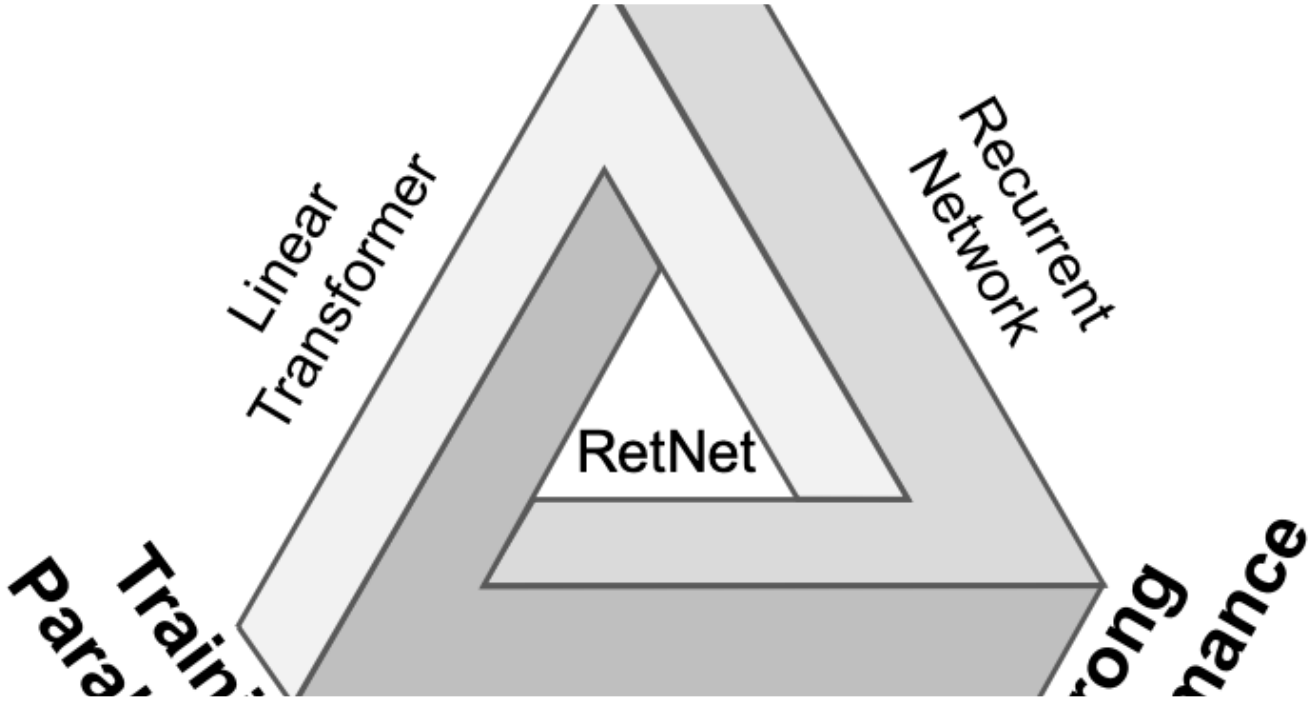
hengtao tantai

Use weighted loss function to solve imbalanced data classification problems

Imbalanced datasets are a common problem in classification tasks, where number of instances in one class is significantly smaller than...

8 min read · Feb 26

24



Sehyun Choi

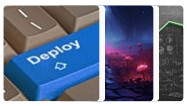
The Rise of RNN? Review of Retentive Network

TLDR; RetNet meets all three criterias: O(1) inference, parallel training, and beats same-sized transformers.

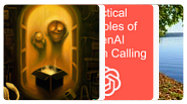
7 min read · Jul 21

107 3

Lists



Predictive Modeling w/ Python
20 stories · 302 saves



Natural Language Processing
543 stories · 159 saves

Practical Guides to Machine Learning
10 stories · 323 saves

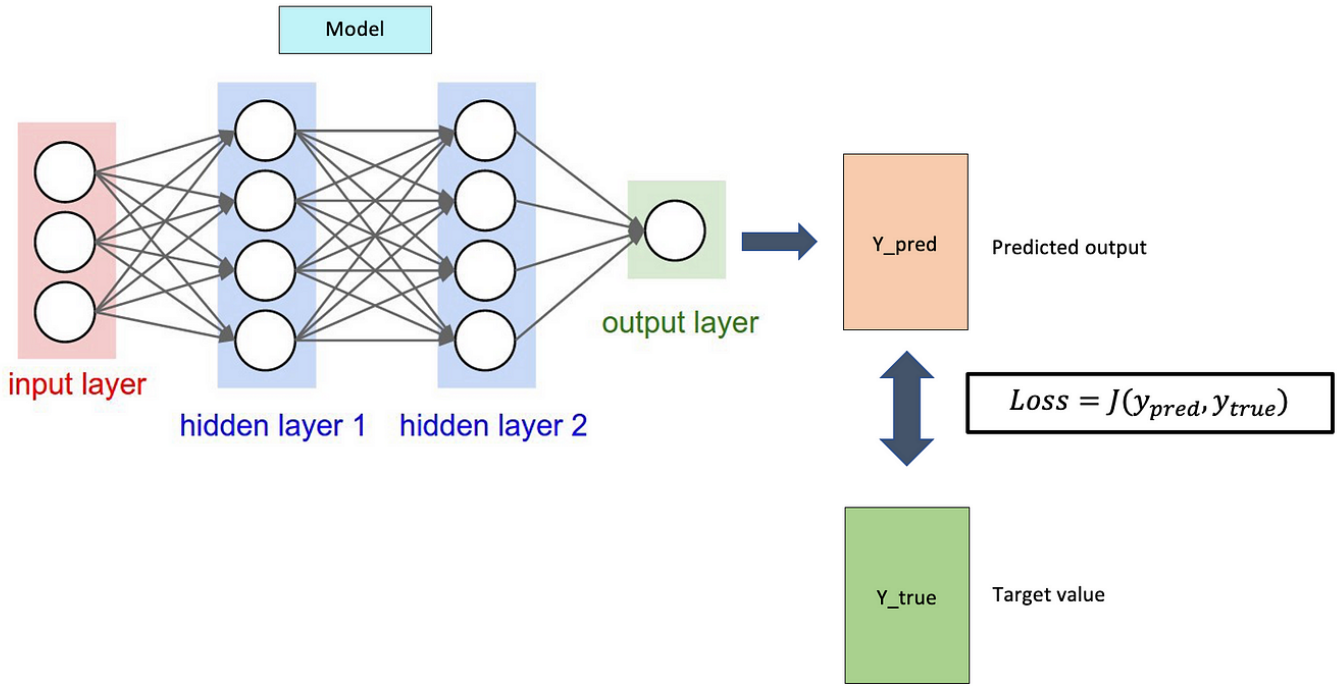
The New Chatbots: ChatGPT, Bard, and Beyond
13 stories · 91 saves

Cameron R. Wolfe, Ph.D. in Towards Data Science

The Best Learning Rate Schedules

Practical and powerful tips for setting the learning rate

· 16 min read · Nov 16, 2022



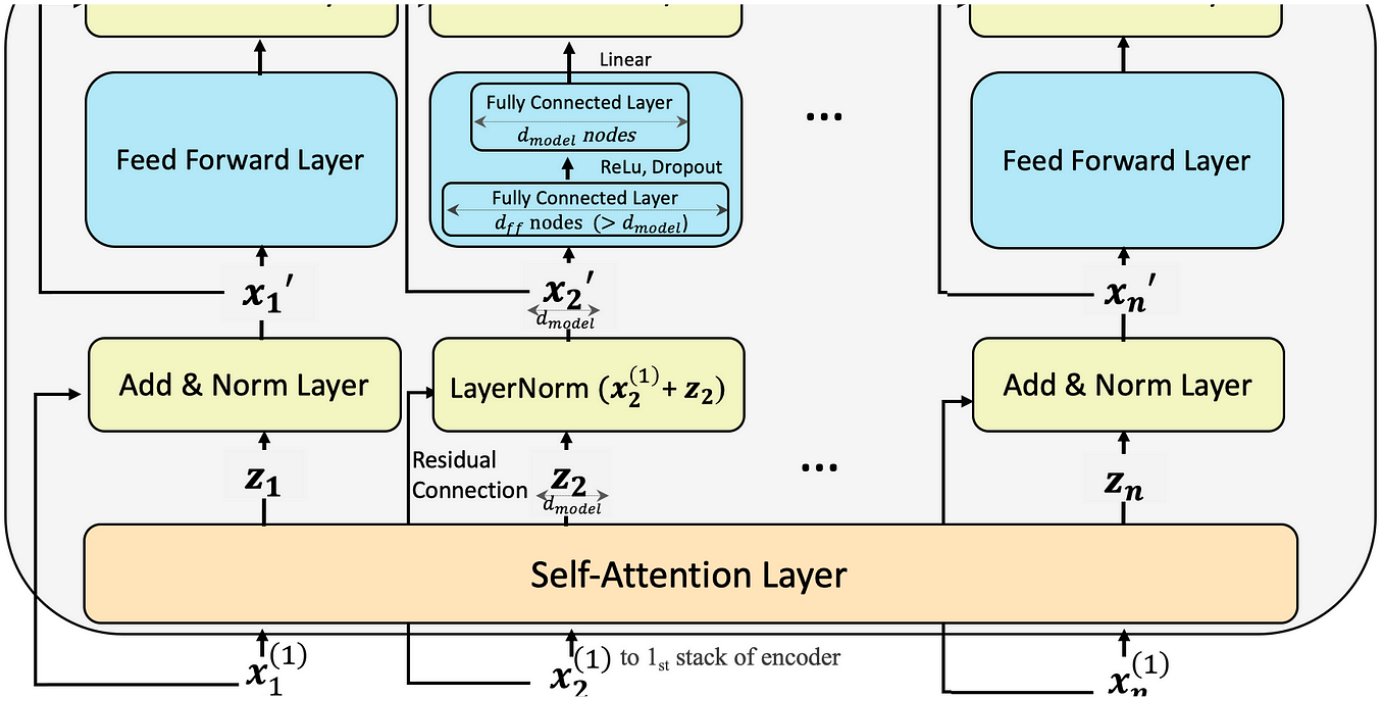
Nghi Huynh in MLearning.ai

Understanding Loss Functions for Classification

Implementation of loss functions for classification task in Python

6 min read · Mar 1

56



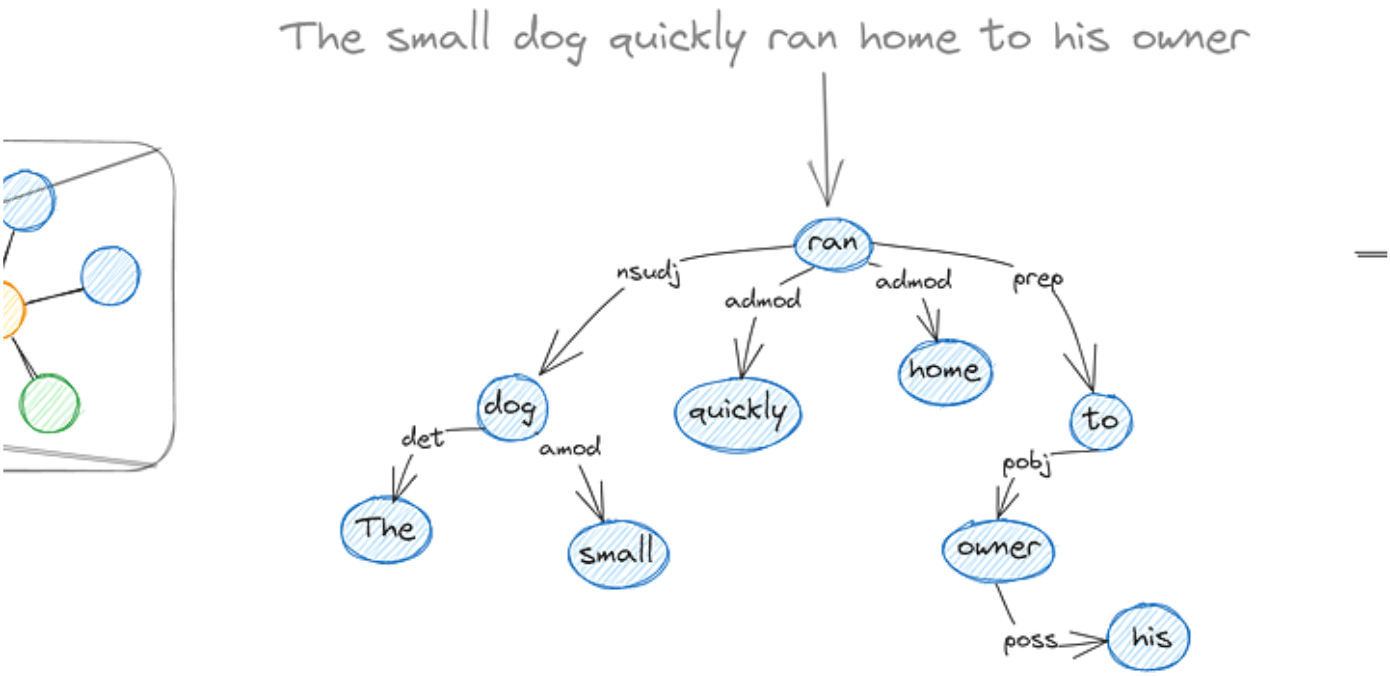
Yule Wang, PhD

Step-by-Step Illustrated Explanations of Transformer

with detailed explained Attention Mechanism

8 min read · Feb 27

242 3



José Luis Castro García

Spectral Graph Convolutions

It is not surprising that Graph Neural Networks have become a major trend in both academic research and practical applications in the...

10 min read · Jul 6

31 1

See more recommendations