1 Ideas

1.1 Line search

Line search in **??** is not working very well.

line search

$$\gamma' = \underset{\gamma\in[0,1]}{\operatorname{argmin}}\ \underbrace{KL\left(q^t + \gamma(s-q^t)\,\|\,p\right)}_{\tilde{D}_{KL}(\gamma)} \quad = \quad \nabla_\gamma f(\tilde{q}_t)$$

grad descent, $\gamma \leftarrow \gamma - \eta \nabla_\gamma \tilde{D}_{KL}(\gamma)$

func. grad,

$$\nabla_{x} f(x) = \log \frac{x}{p}$$

$$-g_x = \langle \nabla f(x), s\rangle - \langle \nabla f(x), x\rangle$$

$$\nabla_\gamma KL\left(q^t + \gamma(s-q^t)\,\|\,p\right)$$

$$\downarrow$$

$$\int \underbrace{\left(q^t + \gamma(s-q^t)\right)}_{\tilde{q}_t} \log\left(\frac{q^t+\gamma(s-q^t)}{p}\right) dx$$

$$= \nabla_\gamma \int \tilde{q}_t \log \frac{\tilde{q}_t}{p}$$

$$= \int \left(\nabla_\gamma \tilde{q}_t\right) \log\frac{\tilde{q}_t}{p} + \cancel{\tilde{q}_t}\, \nabla_\gamma \log \frac{\tilde{q}_t}{p}$$

$$\nabla_\gamma \tilde{q}_t \to s - q^t \qquad \nabla_\gamma \log\frac{\tilde{q}_t}{p} \to \frac{1}{\cancel{\tilde{q}_t}} \nabla_\gamma \hat{q}_t \to (s-q^t)$$

$$= \int (s-q^t) \log \frac{\tilde{q}_t}{p} + (s-q^t)$$

$$= \int (s-q^t)\left(1 + \log\frac{\tilde{q}_t}{p}\right)$$

$$= \int s \log \frac{\tilde{q}_t}{p} - \int q^t \log\frac{\tilde{q}_t}{p}$$

$$= \left\langle s, \log\frac{\tilde{q}_t}{p}\right\rangle - \left\langle q^t, \log\frac{\tilde{q}_t}{p}\right\rangle$$

$$= \mathbb{E}_s\left[\log \tilde{q}_t - \log p\right] - \mathbb{E}_{q^t}\left[\log\tilde{q}_t - \log p\right]$$

$$= \mathbb{E}_s\left[\nabla_{\tilde{q}_t} f(\tilde{q}_t)\right] - \mathbb{E}_{q^t}\left[\nabla_{\tilde{q}_t} f(\tilde{q}_t)\right]$$

```
def line_search_dkl(weights, locs, diags, mu_s, cov_s, x, k):
  """Perform line search for the best step size gamma.

  Uses gradient ascent to find gamma that minimizes
  KL(q_t + gamma (s - q_t) || p)

  Args:
      weights: weights of mixture components of q_t
      locs: means of mixture components of q_t
      diags: deviations of mixture components of q_t
      mu_s: mean for LMO Solution s
      cov_s: cov matrix for LMO solution s
      x: target distribution p
      k: iteration number of Frank-Wolfe
  Returns:
      Computed gamma
  """
  def softmax(v):
      return np.log(1 + np.exp(v))
  # no. of samples to approximate ∇_γ
  N_samples = 10
  # Create current iter q_t
  weights = [weights]
  qt_comps = [
      Normal(
          loc=tf.convert_to_tensor(locs[i]),
          scale=tf.convert_to_tensor(diags[i])) for i in range(len(locs))
  ]
  qt = Mixture(
      cat=Categorical(probs=tf.convert_to_tensor(weights)),
      components=qt_comps,
      sample_shape=N)
  qt = InfiniteMixtureScipy(stats.multivariate_normal)
  qt.weights = weights[0]
  qt.params = list(
      zip([[l] for l in locs], [[softmax(np.dot(d, d))] for d in diags]))
  # samples from q_t
  sample_q = qt.sample_n(N_samples)
  # create and sample from s
  s = stats.multivariate_normal([mu_s],
                                np.dot(np.array([cov_s]), np.array([cov_s])))
  sample_s = s.rvs(N_samples)
  # q_{t+1} is mixture of q_t and s with weights (1 − γ) and γ
  # Set its corresponding parameters and weights
  new_locs = copy.copy(locs)
  new_diags = copy.copy(diags)
  new_locs.append([mu_s])
  new_diags.append([cov_s])
  # initialize γ
  gamma = 2. / (k + 2.)
  # no. steps of gradient ascent
  n_steps = 10
  prog_bar = ed.util.Progbar(n_steps)
  for it in range(n_steps):
  print("line_search iter %d, %.5f" % (it, gamma))
  new_weights = copy.copy(weights)
  new_weights[0] = [(1. - gamma) * w for w in new_weights[0]]
  new_weights[0].append(gamma)
  # create q^γ_{t+1}
  q_next = InfiniteMixtureScipy(stats.multivariate_normal)
  q_next.weights = new_weights[0]
  q_next.params = list(
  zip([[l] for l in new_locs], [[np.dot(d, d)] for d in new_diags]))
  # Computes E[...] ∝ ∑_v log p − log q^γ_{t+1}
  def px_qx_ratio_log_prob(v):
  Lambda = 1.
```

```
ret = x.log_prob([v]).eval()[0] - q_next.log_prob(v)
ret /= Lambda
return ret
# Samples w.r.t s
rez_s = [
px_qx_ratio_log_prob(sample_s[ss]) for ss in range(len(sample_s))
]
# Samples w.r.t q_{t+1}
rez_q = [
px_qx_ratio_log_prob(sample_q[ss]) for ss in range(len(sample_q))
]
# Gradient ascent step, step size decreasing as 1/(it+1)
gamma = gamma + 0.1 * (sum(rez_s) - sum(rez_q)) / (N_samples *
(it + 1.))
# Projecting it back to [0, 1], too small range?
# FIXME(sauravshekhar) if projected to 0, all iterations will be same?
if gamma >= 1 or gamma <= 0:
gamma = max(min(gamma, 1.), 0.)
break
return gamma
```

changes for measuring variance of $\mathbb{E}_s\left[\cdot\right]$ and $\mathbb{E}_{q_{t+1}^{\gamma}}\left[\cdot\right]$.

```
    ...
    grad_gamma = []
    for it in range(n_steps):
    ...
    # Samples w.r.t s
    rez_s = np.asarray([
        px_qx_ratio_log_prob(sample_s[ss]) for ss in range(len(sample_s))
    ])
    # Samples w.r.t q_{t+1}
    rez_q = np.asarray([
        px_qx_ratio_log_prob(sample_q[ss]) for ss in range(len(sample_q))
    ])
    grad_gamma.append({'E_s': rez_s, 'E_q': rez_q, 'gamma': gamma})
    ...
    # Write grad_gamma to outdir/line_search_samples_<n_samples>.npy.<fw_iter>
```

Metrics on original version.

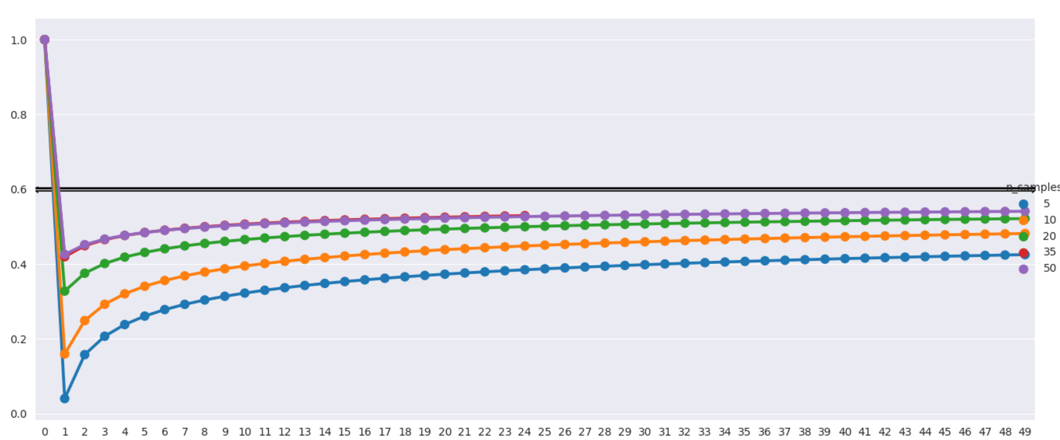

Figure 1: gamma with iterations for different n_samples

1.2 Adaptive Frank-Wolfe from smoothness estimators

Algorithm 1 of [Pedregosa et al., 2018]

Code changes.

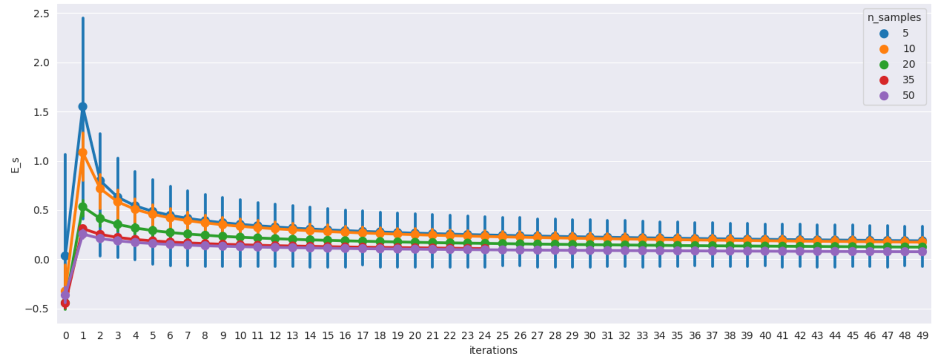

Figure 2: E_s with different n_samples

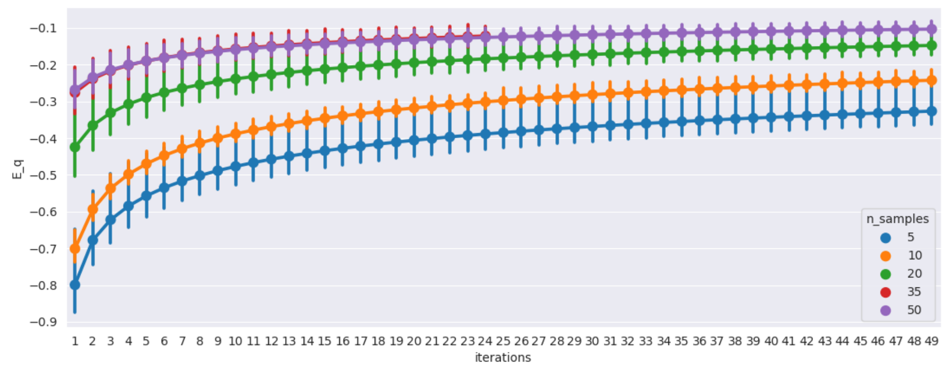

Figure 3: E_q with different n_samples

?? begins with iter 1 as iter 0 has very high variance

1.3 Measuring smoothness

in progress

Computing optimal γ directly from eqn 1 of [Pedregosa et al., 2018]

$$
\begin{aligned}
f(\mathbf{x}_{t+1}) &\leq f(\mathbf{x}_t) + \gamma\langle \nabla f(\mathbf{x}_t), \mathbf{s}_t - \mathbf{x}_t\rangle + \frac{\gamma^2}{2} L_t ||\mathbf{s}_t - \mathbf{x}_t||^2 \\
\Rightarrow L_t &\geq \frac{f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) + \gamma\langle \nabla f(\mathbf{x}_t), \mathbf{s}_t - \mathbf{x}_t\rangle}{\frac{\gamma^2}{2}||\mathbf{s}_t - \mathbf{x}_t||^2} \\
&\frac{f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t) + \gamma\langle \nabla f(\mathbf{x}_t), \mathbf{s}_t - \mathbf{x}_t\rangle}{\frac{\gamma^2}{2}\mathrm{KL}\left(s_t || q_t\right)}
\end{aligned}
$$

Code changes.

```
def kl(mu_q, std_q, mu_p, std_p):
  ...

def func_dot_product():
  ...
```

1.4 Other optimization algorithm

todo

As shown in link **??**, Frank-Wolfe converges slower than Projected Gradient Descent in Practice. See [Locatello et al., 2017] to see why we use FW and if it can be replaced. (will have to derive new convergence proofs and boosting won't be as integrated into the optimization algorithm as before).

Algorithm 1: Adaptive Frank-Wolfe for Boosting BBVI

Input: $q_0 \in \mathcal{D}$, initial Lipschitz estimate L_{-1}, line search parameters $\tau > 1, \eta \in (0, 1]$

for $t = 1 \ldots T$ **do**

  $s_t \leftarrow LMO_{\mathcal{A}}(\nabla f(q^t))$

  $g_t \leftarrow \langle \nabla f(q_t), q_t \rangle - \langle \nabla f(q_t), s_t \rangle$ `// Gap` ≥ 0

  Find smallest integer i s.t

  $f(q_t + \gamma_t(s_t - q_t)) \leq Q_t(\gamma_t)$ Where

  $Q_t(\gamma) := f(q_t) - \gamma g_t + \frac{\gamma^2 L_t}{2} d(s_t, q_t)$ `// Quadratic upper bound ??`

  $L_t \leftarrow \tau^i \eta L_{t-1}$ and $\gamma_t \leftarrow \min\left(\frac{g_t}{L_t d(s_t, q_t)}, 1\right)$

end

return q_T

1.5 Entropy Regularization and Noise addition using Optimal Transport

In LMO, [Locatello et al., 2018] uses Entropy Regularization in place of norm constrained optimization. It can be replaced with something simpler See [Tolstikhin et al., 2017] [Dong Liu, 2018] [Bernton et al., 2017] [Jordan et al., 1998] here And [Peyré et al., 2017] part 4.

1.6 Port code to Tensorflow Probability

see ??. Issue is if `tfp.edward2` will have support for Variational Inference and ELBO etc.

References

[Bernton et al., 2017] Bernton, E., Jacob, P. E., Gerber, M., and Robert, C. P. (2017). Inference in generative models using the wasserstein distance. *arXiv preprint arXiv:1701.05146*.

[Dong Liu, 2018] Dong Liu, Minh Thnh Vu, S. C. L. K. R. (2018). Entropy-regularized optimal transport generative models. *arXiv preprint arXiv:1811.06763*.

[Jordan et al., 1998] Jordan, R., Kinderlehrer, D., and Otto, F. (1998). The variational formulation of the fokker–planck equation. *SIAM journal on mathematical analysis*, 29(1):1–17.

[Locatello et al., 2018] Locatello, F., Dresdner, G., Khanna, R., Valera, I., and Rätsch, G. (2018). Boosting black box variational inference. *arXiv preprint arXiv:1806.02185*.

[Locatello et al., 2017] Locatello, F., Khanna, R., Ghosh, J., and Rätsch, G. (2017). Boosting variational inference: an optimization perspective. *arXiv preprint arXiv:1708.01733*.

[Pedregosa et al., 2018] Pedregosa, F., Askari, A., Negiar, G., and Jaggi, M. (2018). Step-size adaptivity in projection-free optimization. *arXiv preprint arXiv:1806.05123*.

[Peyré et al., 2017] Peyré, G., Cuturi, M., et al. (2017). Computational optimal transport. Technical report, École Normale Supérieure.

[Tolstikhin et al., 2017] Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*.