

Evaluation of baseline LSTM optimizer (denoted "meta" in all figures)

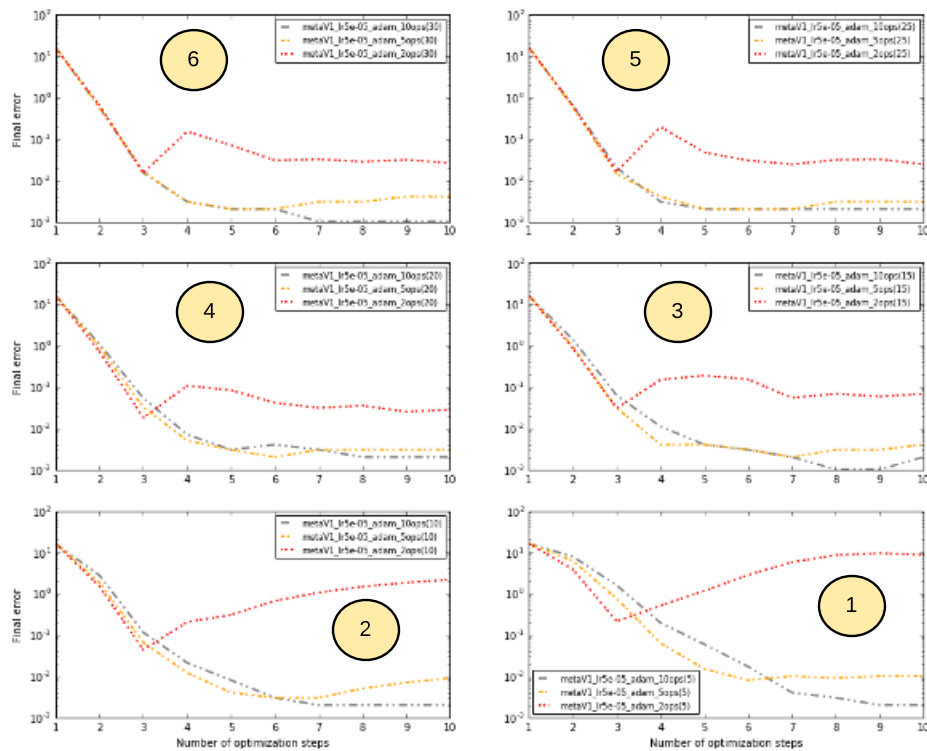
Stochastic training regime stabilizes LSTM optimizer after training horizon T

Left: without stochastic training regime (parameter error)

Figures showing average parameter error for 6 validation runs for 3 different training horizons T: **2, 5 and 10**

Validation **run 1** is shown on bottom right side, **run 2** bottom left side, **run 3** middle row right side, **run 4** middle row left side, **run 5** top row right side, **run 6** top row left side

Final average parameter error (regression)

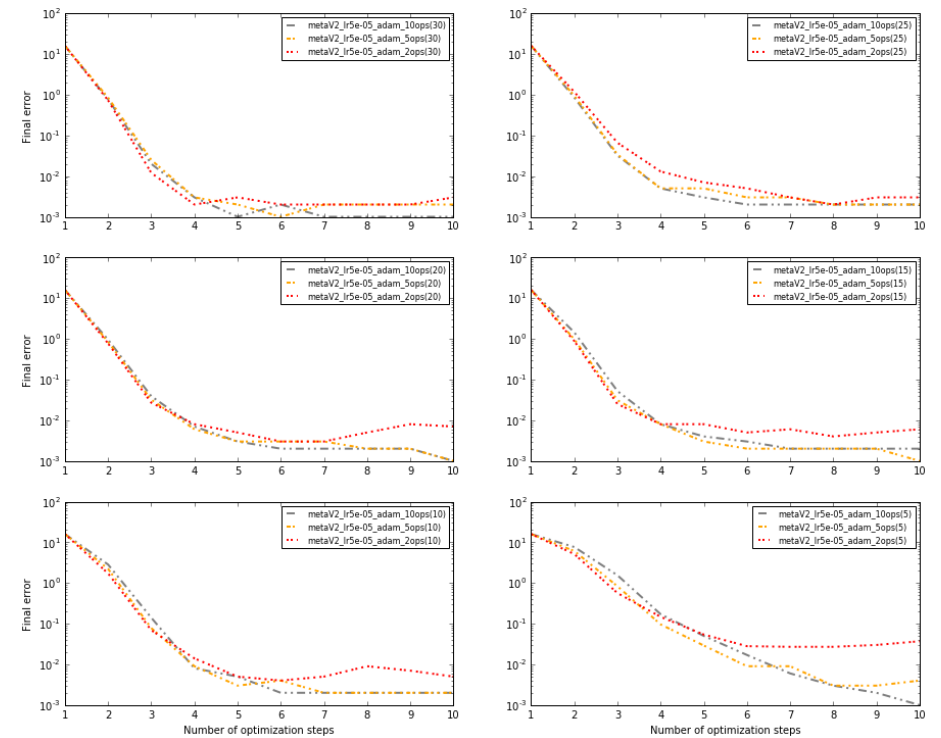


Right: with stochastic training regime (parameter error)

For the stochastic training regime the horizon T is a mean value.

6 validation runs as described on the left side of this slide.

Final average parameter error (regression)

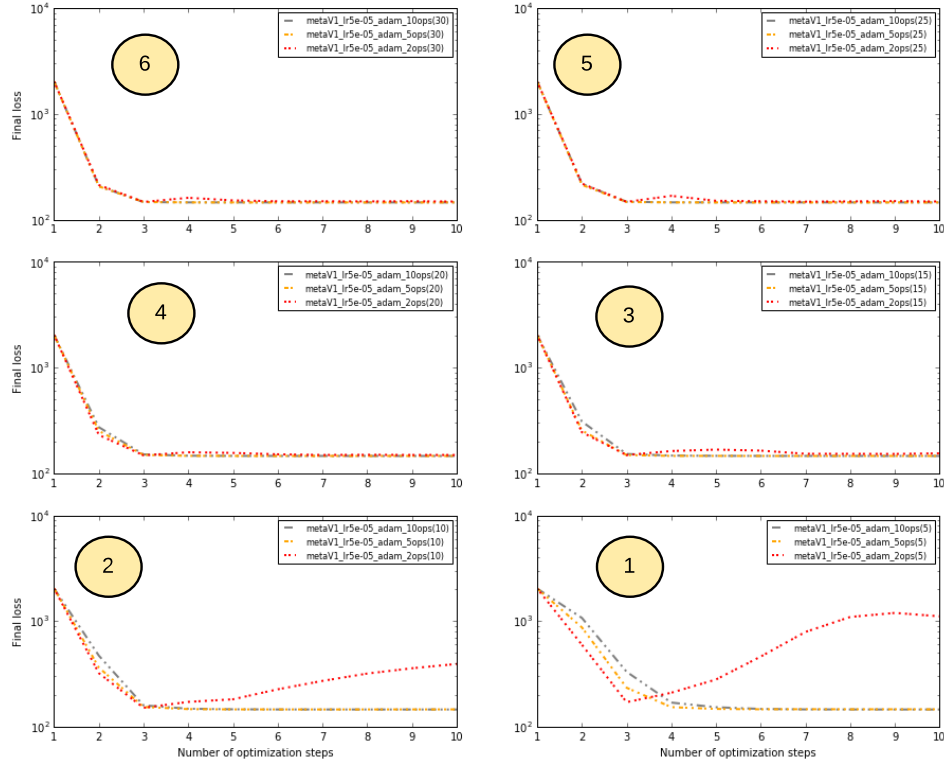


The effect is significantly less when considering the final loss (instead of parameter error)

Left: without stochastic training regime (**loss**)

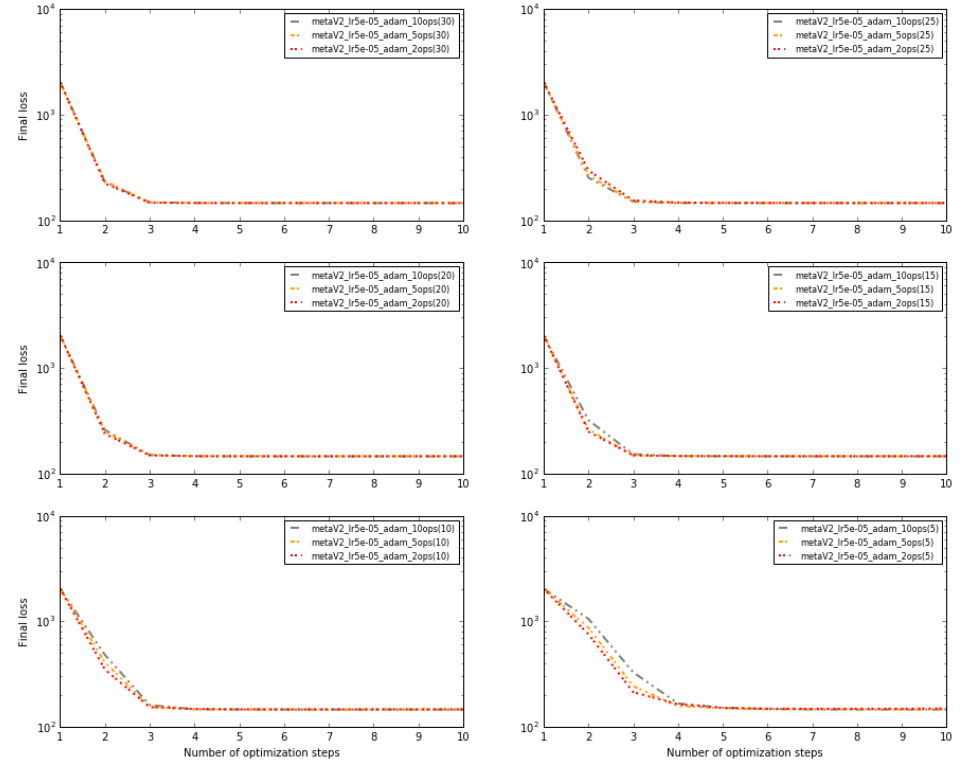
Figures showing average loss during 6 validation runs for 3 different training horizons **T: 2, 5 and 10**

Final average loss (regression)



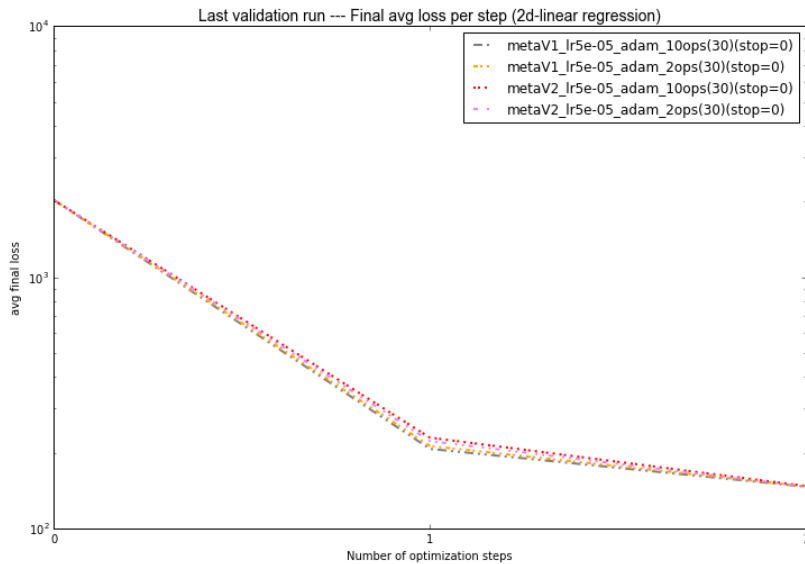
Right: with stochastic training regime (**loss**)

Final average loss (regression)



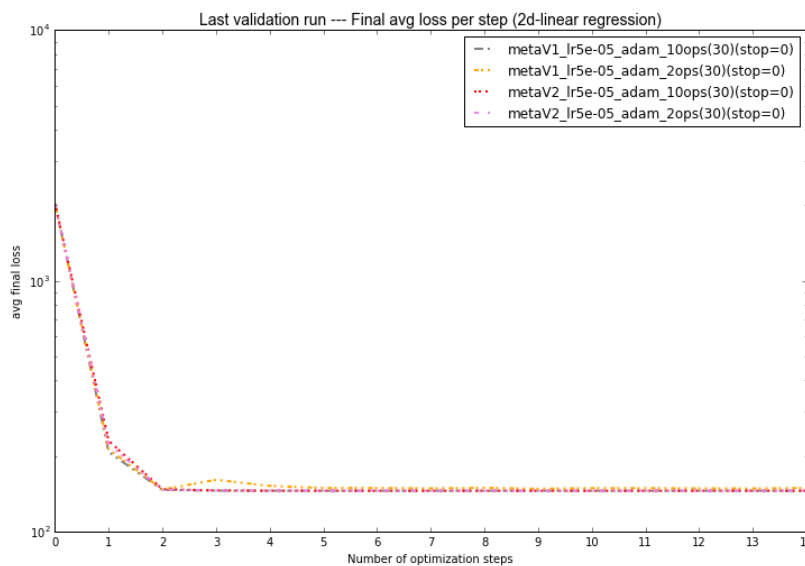
Accept for *instable behavior* after training horizon T, both training regimes perform roughly the same for LSTM optimizer

Left: final average validation **loss** (V1=without stochastic training regime)



Zoom in for
first 2 optimization
steps

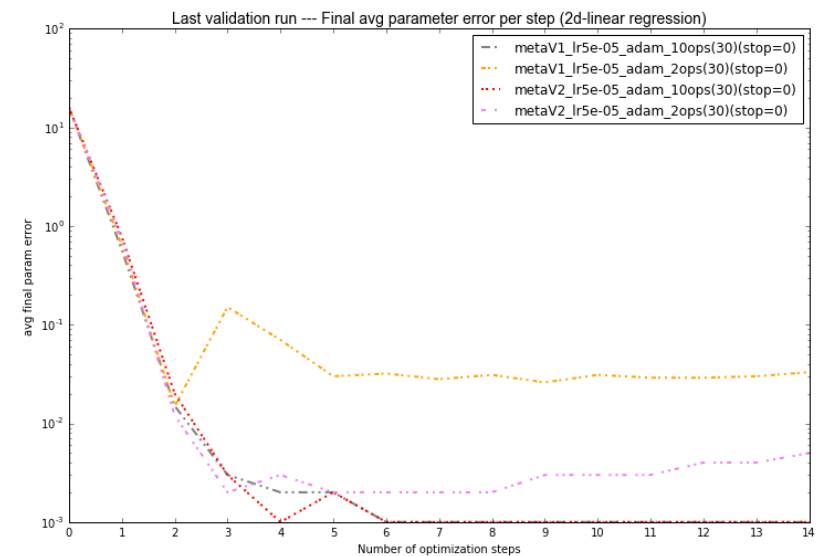
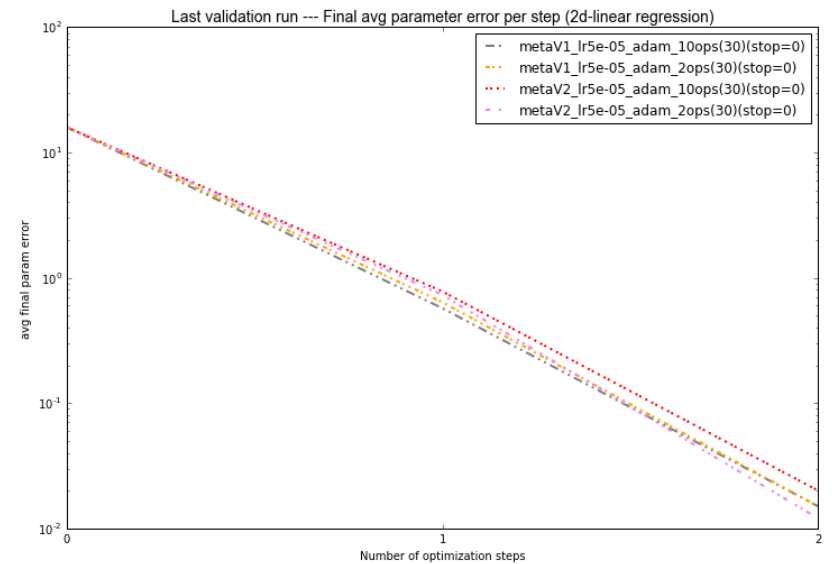
T = 2 and 10



Zoom in for
first 14
optimization steps

T = 2 and 10

Right: final average validation **parameter error** (V1=without stochastic training regime)



Evaluation of ACT optimizer

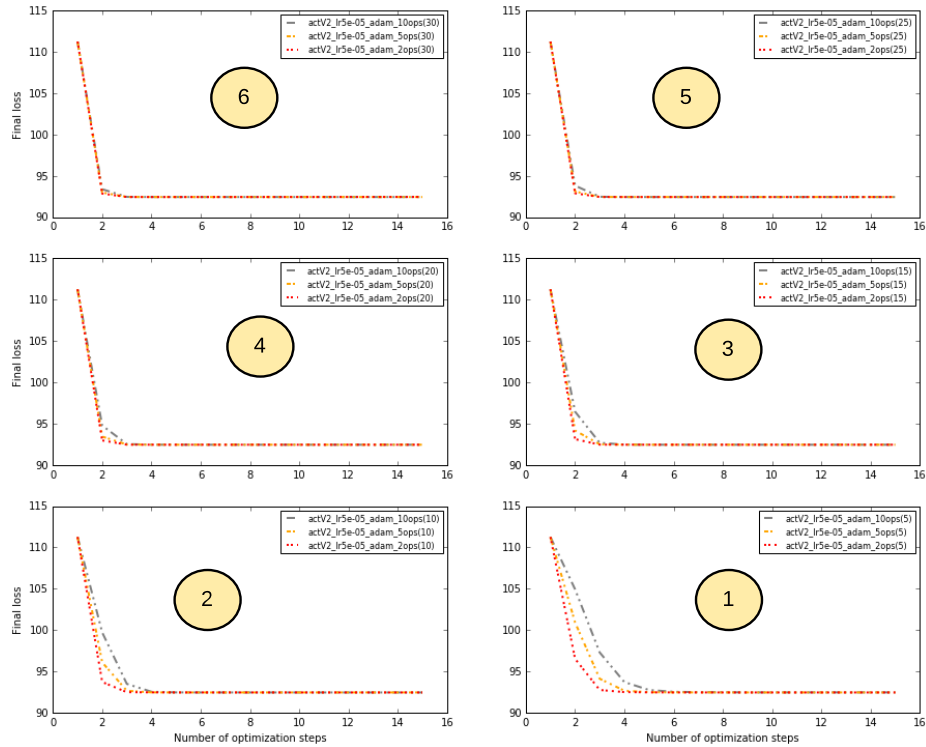
ACT optimizer gets less greedy with increased average training horizon T

Left: average validation loss

Figures showing average loss during 6 validation runs for 3 different training horizons T : 2, 5 and 10

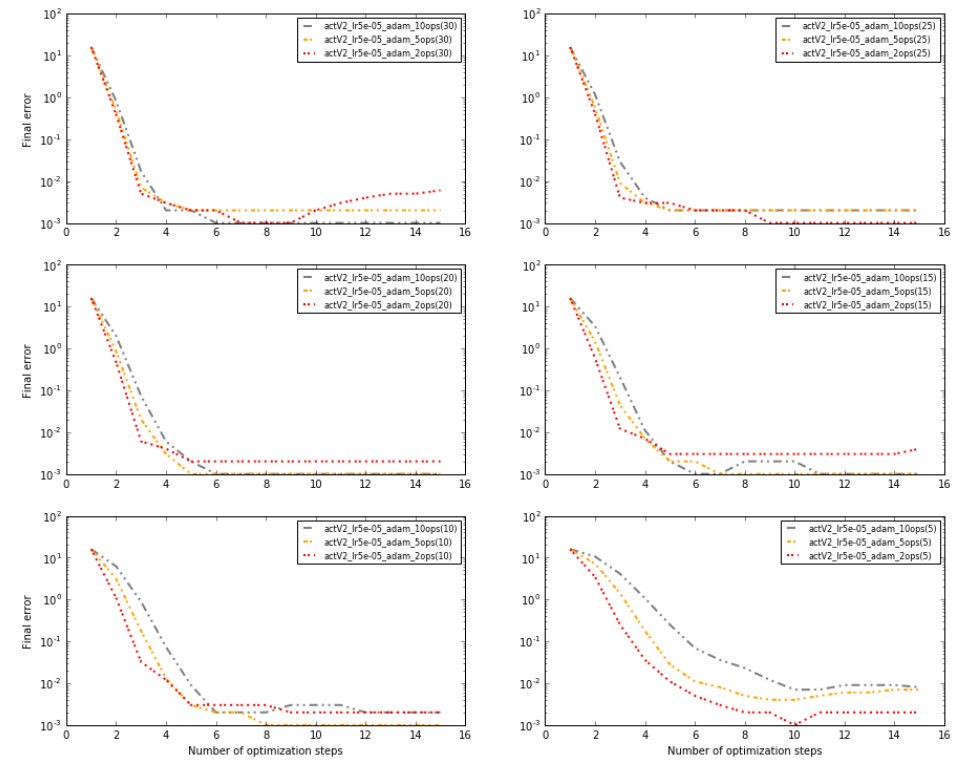
Validation **run 1** is shown on bottom right side, **run 2** on bottom left side, **run 3** middle row right side, **run 4** middle row left side, **run 5** top row right side, **run 6** top row left side

Final average loss (regression)



Right: average validation parameter error

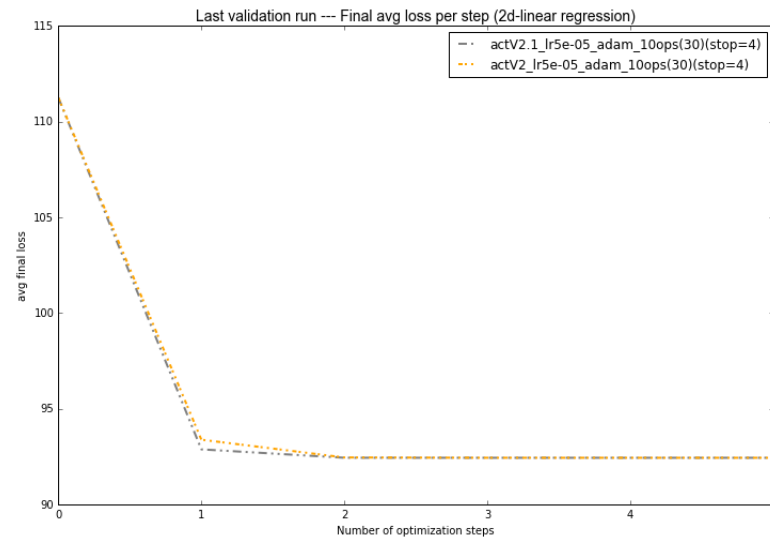
Final average parameter error (regression)



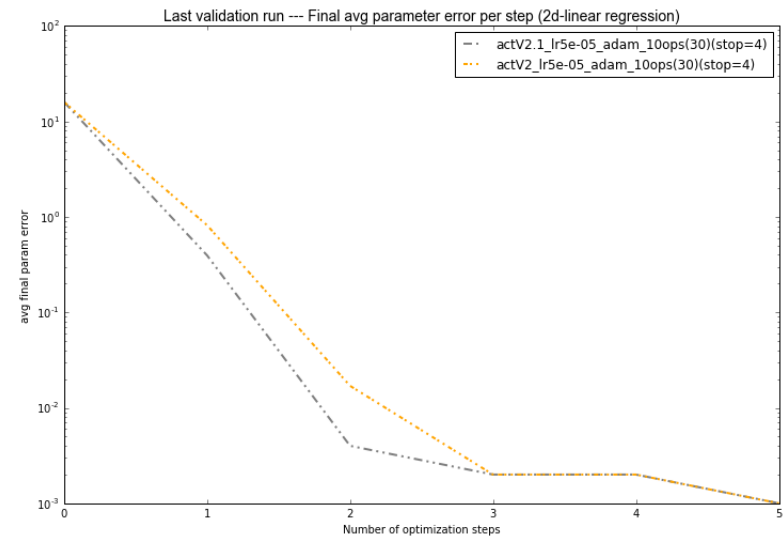
ACT optimizer gets less greedy when model uses linear transformation for producing q_t value compared to linear-followed-by-non-linearity (actV2.1)

Left: average final validation loss

Figures showing the performance of two ACT optimizer models that differ solely in the way they produce the Δq_t value as explained below. Performance is shown for the first 4 optimization steps during validation (on average).



Right: average final validation parameter error



actV2: $\Delta q_t = \mathbf{w}_q^T \mathbf{h}_t$

actV2.1: $\Delta q_t = \lambda \tanh(\mathbf{w}_q^T \mathbf{h}_t)$

What does this exactly mean?

- (1) My hunch is that the model is less constraint when **not using the non-linearity** and explores an easy solution to minimize the total loss by concentrating all the probability mass of the approximated posterior distribution $q(t)$ on the last optimization step (see slides at the end of this document). But this is not helpful for our objective of encouraging a greedy optimization model;
- (2) Or, is this the way we have to *force* the model to more strictly follow the prior distribution? Although I can't see why this should my sense.

Performance comparison between LSTM and ACT optimizer

ACT is less greedy than LSTM baseline especially when trained for larger horizon T

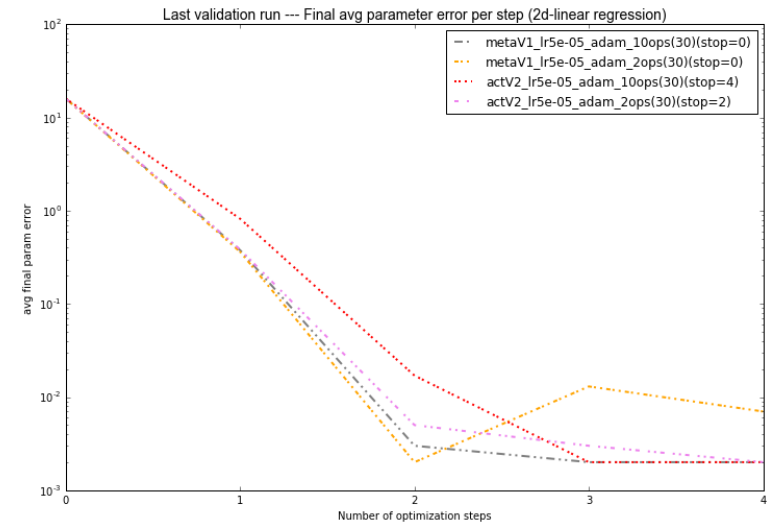
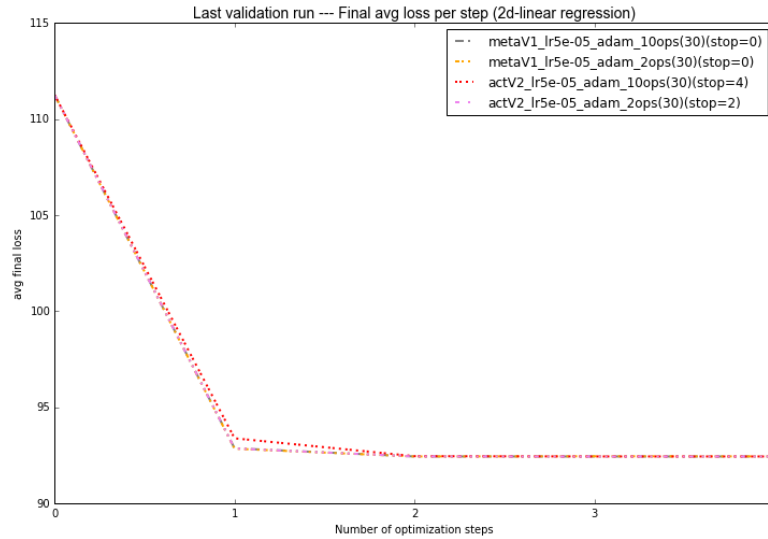
Left: average final validation loss

Figures show LSTM and ACT optimizer model performance during last validation run.
Both models were trained with a horizon of $T = 2$ and 10 optimization steps

Right: average final validation parameter error

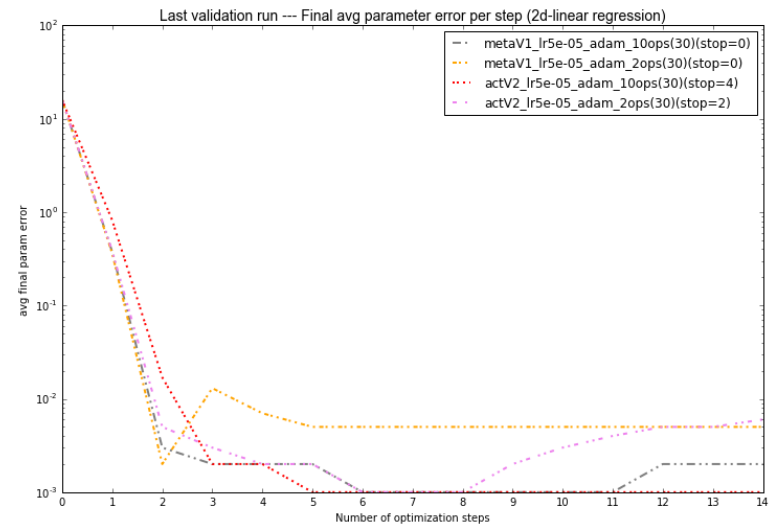
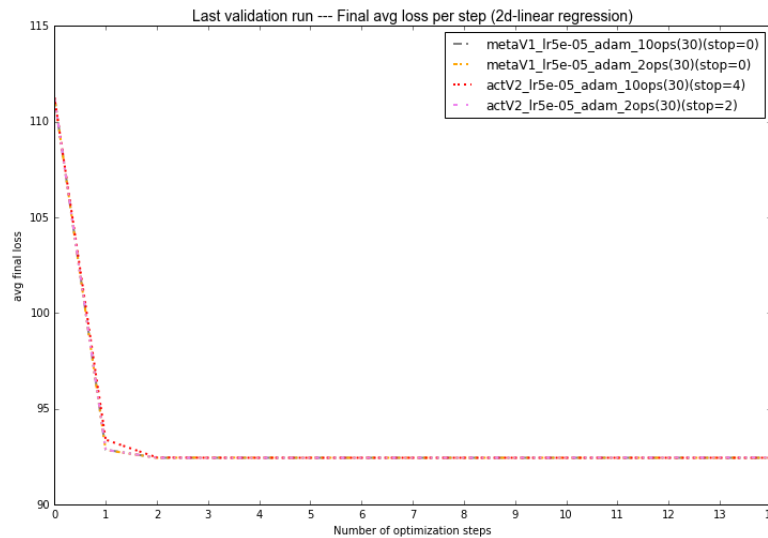
Zoom in for
first 4 optimization
steps

$T = 2$ and 10



Zoom in for
first 14
optimization steps

$T = 2$ and 10

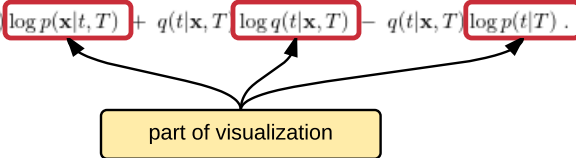


Visualization of components of loss function for ACT optimizer

The goal of the visualization of the loss function components is to get an inside why the model basically ignores the form of the prior distribution. This effect is actually more severe in case the delta q_t values are modelled only by an affine transformation (instead of using a linear transformation followed by a non-linearity plus scaling).

This is the *variational upper bound* the ACT optimizer tries to minimize

$$\log p(\mathbf{x}) \leq \sum_{T=0}^{\infty} p(T) \sum_{t=0}^T -q(t|\mathbf{x}, T) \boxed{\log p(\mathbf{x}|t, T)} + q(t|\mathbf{x}, T) \boxed{\log q(t|\mathbf{x}, T)} - q(t|\mathbf{x}, T) \boxed{\log p(t|T)} .$$



Possible reasons:

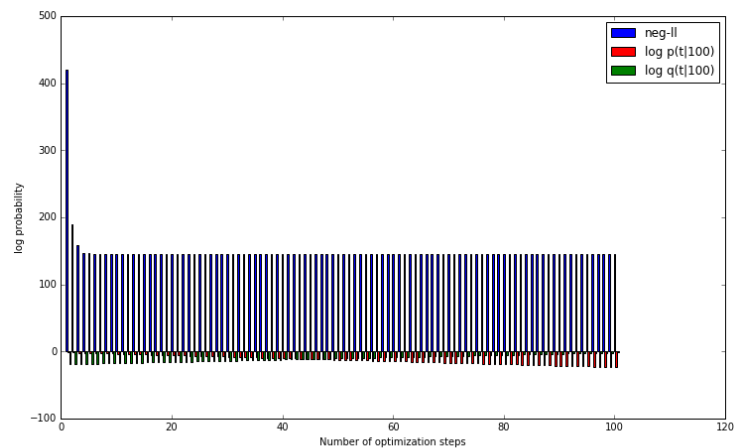
- (1) in case of the regression functions the *final loss* is still substantial and outweighs the increasing divergence between prior and approximate posterior. It is still more beneficial for the model to *center* the probability mass on the last optimization steps. The model generates much smaller $q(t|T)$ values than those in the tail of the prior distribution (even for a large horizon T). This is beneficial because it reduces the loss on the negative log-likelihood (the entropy of $q(t|T)$ is converging to zero anyway).
- (2) In the 2d quadratic experiments the ACT model shows a significantly different performance. Especially when the delta q_t values are produced by the linear transformation only. The approximated posterior distribution does still not exactly match the prior $p(t)$ but compared to the regression experiments the approximation is significantly better. Could this be due to the fact that the $\log p(\mathbf{x}|t, T)$ term which is in the quadratics case just the function evaluation at the current parameter value θ_t , converges to zero after 2-3 optimization steps and the KL terms have more influence in this setting?
- (3) Is the prior probability distribution $p(t)$ correct?

ACT optimizer (2d-regression): approximated posterior distribution $q(t)$ & log probabilities of KL divergence

Left: model uses following transformation on output state h_t $\Delta q_t = \lambda \tanh(\mathbf{w}_q^T \mathbf{h}_t)$

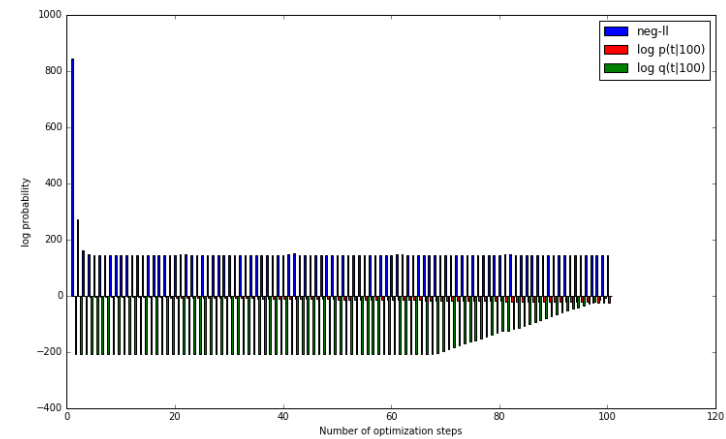
Right: model uses following transformation on output state h_t $\Delta q_t = \mathbf{w}_q^T \mathbf{h}_t$

Separate KL-divergence components per optimization step

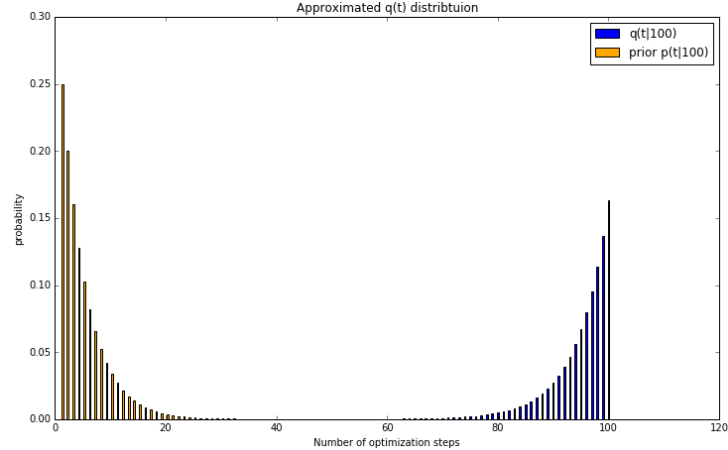


log probabilities
components of
KL-divergence

Separate KL-divergence components per optimization step

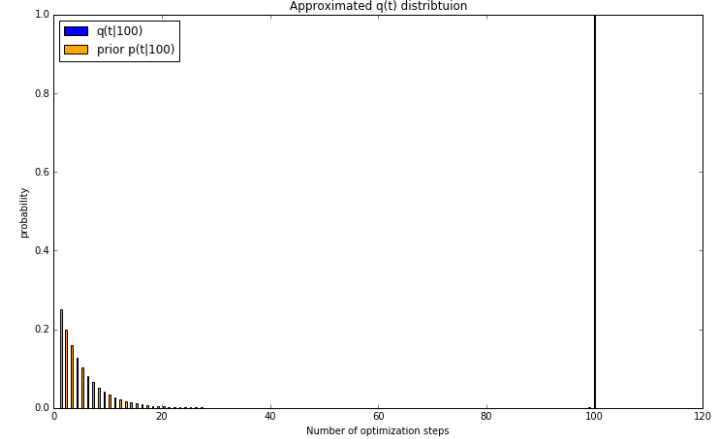


Approximated $q(t)$ distribuion



approximated
posterior $q(t)$ &
prior $p(t)$

Approximated $q(t)$ distribuion

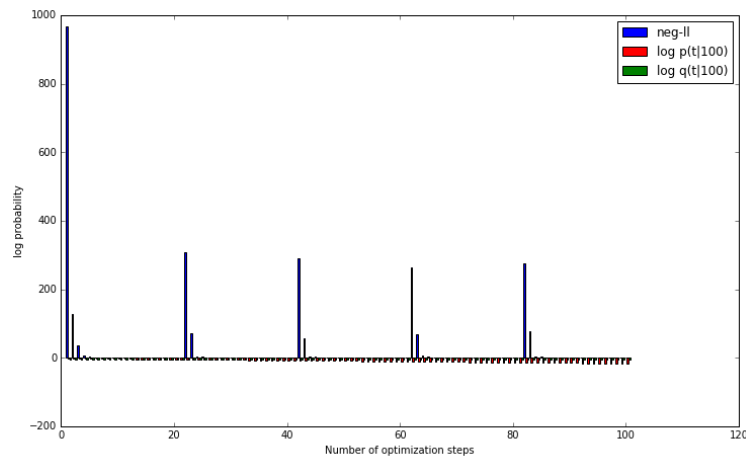


ACT optimizer (2d-quadratics): approximated posterior distribution $q(t)$ (opposed to $p(t)$ prior)

Left: $\Delta q_t = \lambda \tanh(\mathbf{w}_q^T \mathbf{h}_t)$

Right: $\Delta q_t = \mathbf{w}_q^T \mathbf{h}_t$

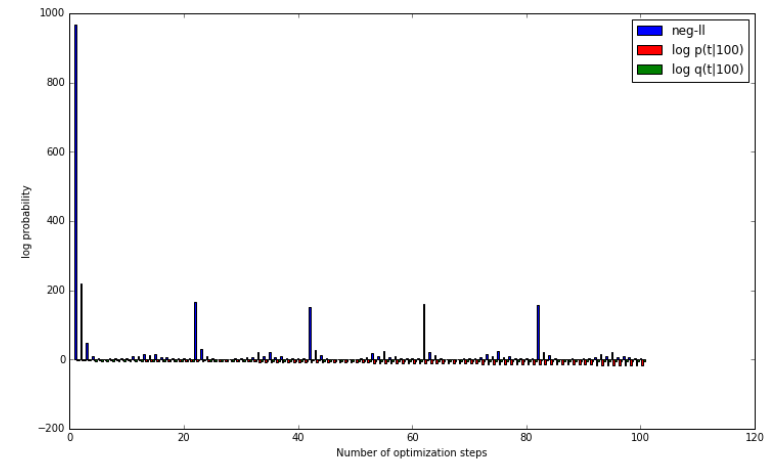
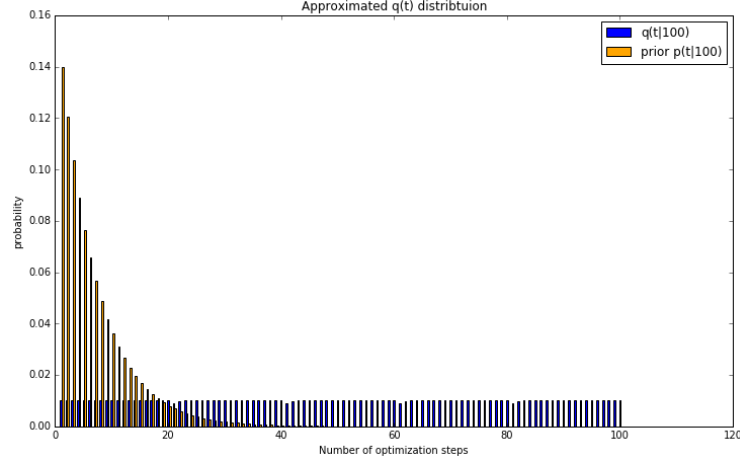
Separate KL-divergence components per optimization step



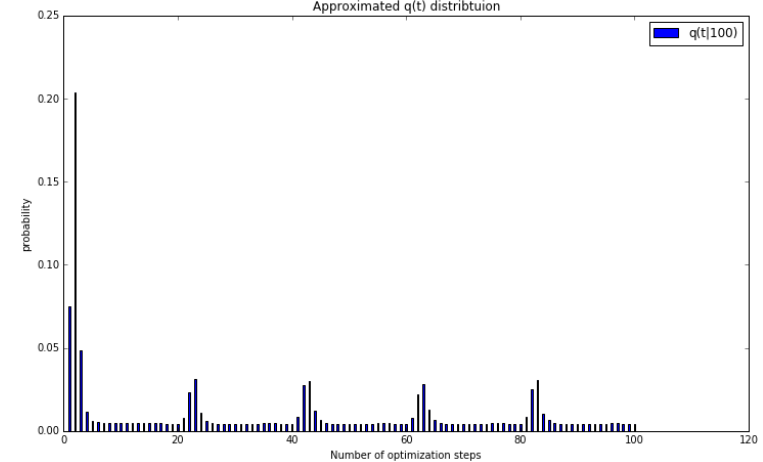
log probabilities
components of
KL-divergence

Please note that in this case the neg-ll is simply the evaluation of the function to be optimized at the current parameter position θ_t . So the term neg-ll is actually wrong in this case,

Separate KL-divergence components per optimization step

Approximated $q(t)$ distribution

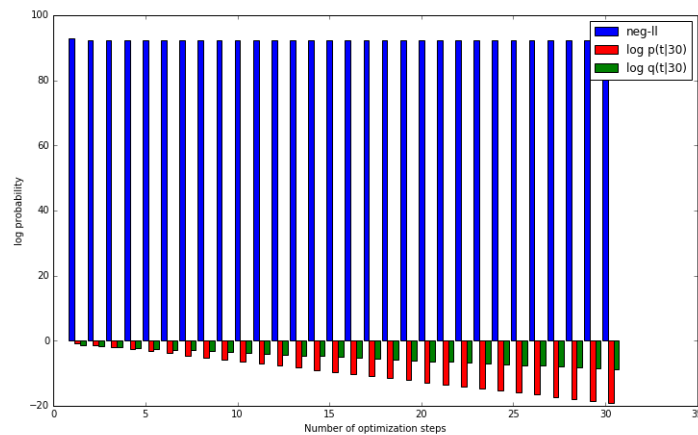
approximated
posterior $q(t)$ &
prior $p(t)$

Approximated $q(t)$ distribution

ACT optimizer (2d-regression): approximated posterior distribution $q(t)$ & log probabilities of KL divergence (model loss function uses **mean loss per sample**)

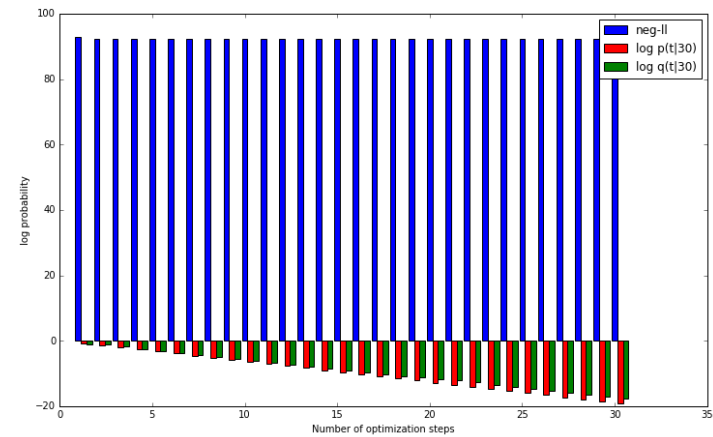
Left: model uses following transformation on output state h_t $\Delta q_t = \lambda \tanh(\mathbf{w}_q^T \mathbf{h}_t)$

Separate KL-divergence components per optimization step

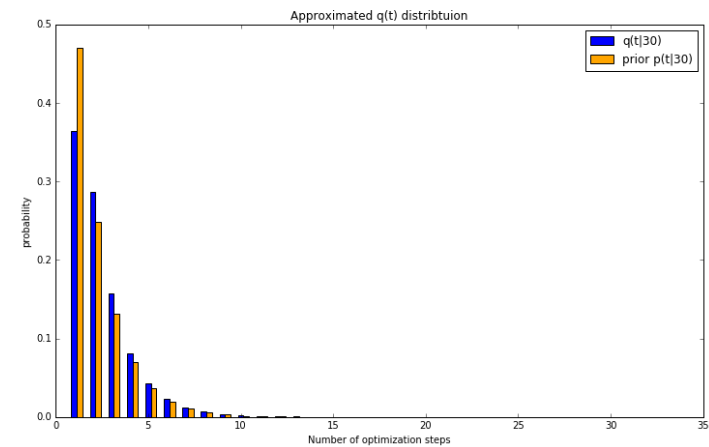
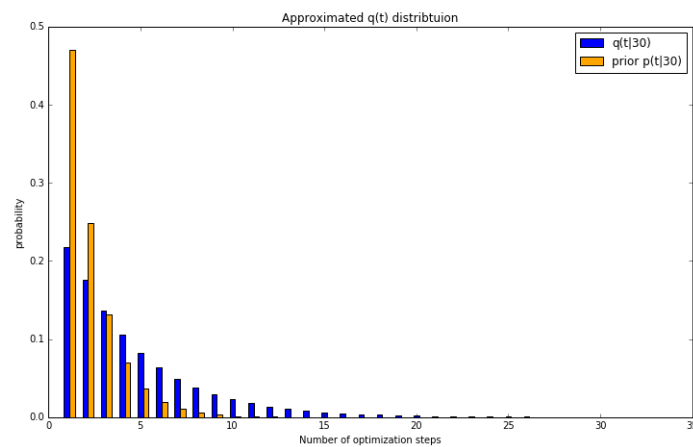


Right: model uses following transformation on output state h_t $\Delta q_t = \mathbf{w}_q^T \mathbf{h}_t$

Separate KL-divergence components per optimization step

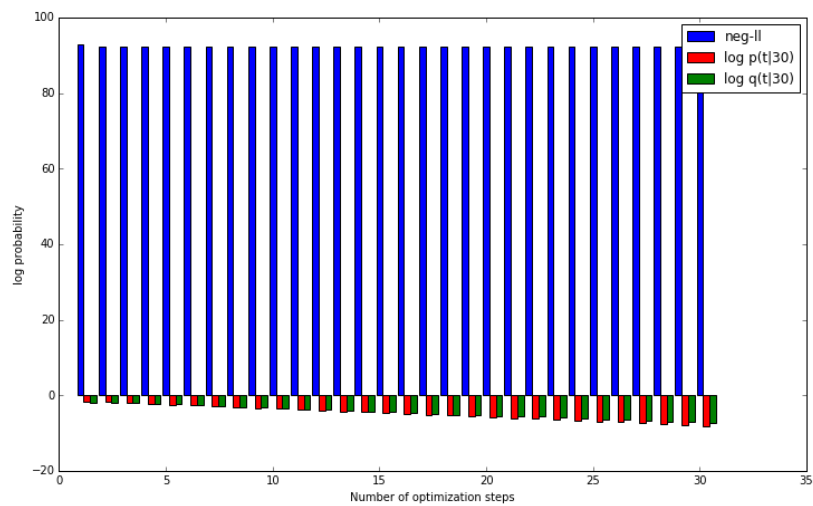


Trained with
horizon $E[T]=2$

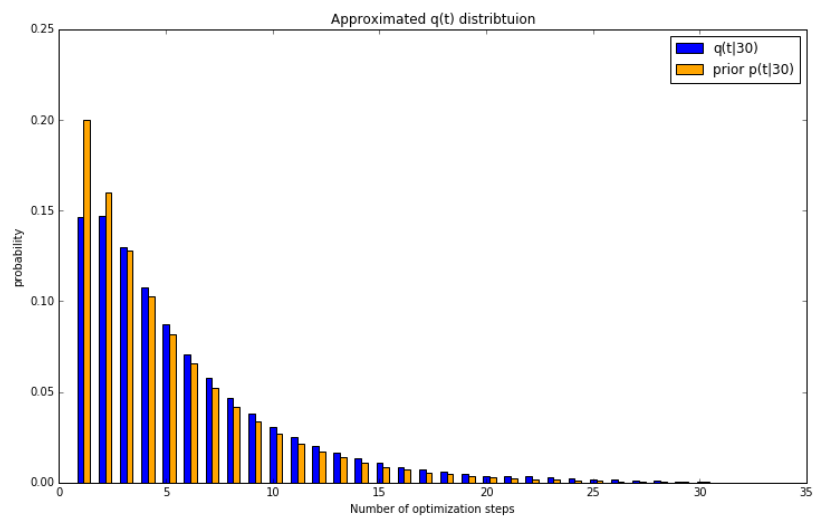


Left: model uses following transformation on output state \mathbf{h}_t $\Delta q_t = \lambda \tanh(\mathbf{w}_q^T \mathbf{h}_t)$

Separate KL-divergence components per optimization step

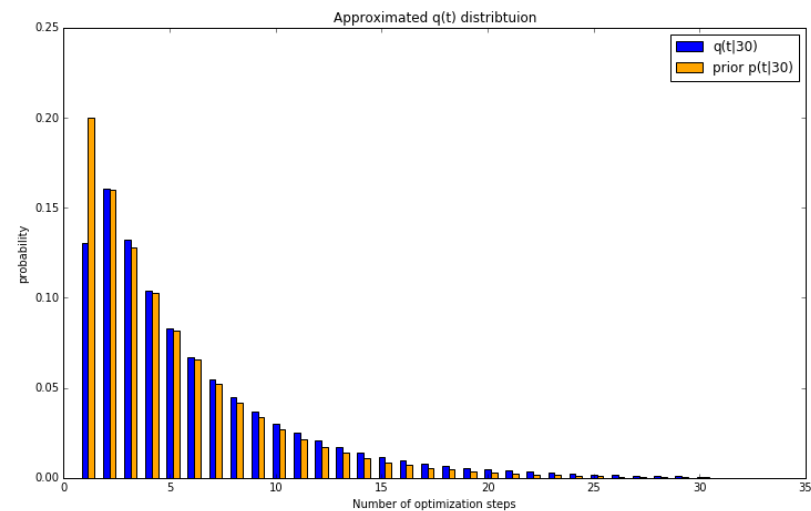
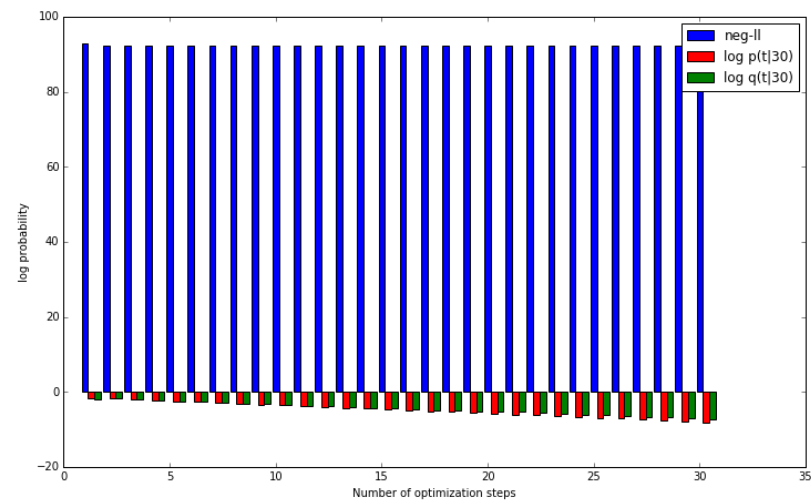


Trained with
horizon $E[T]=5$



Right: model uses following transformation on output state \mathbf{h}_t $\Delta q_t = \mathbf{w}_q^T \mathbf{h}_t$

Separate KL-divergence components per optimization step

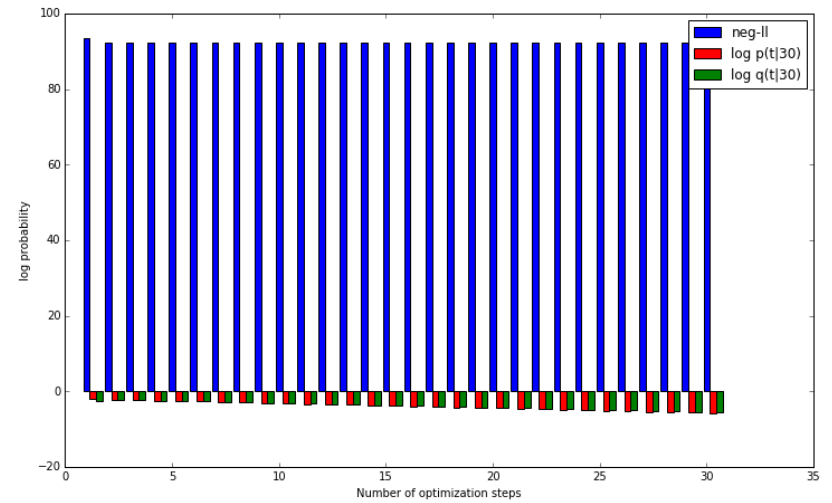
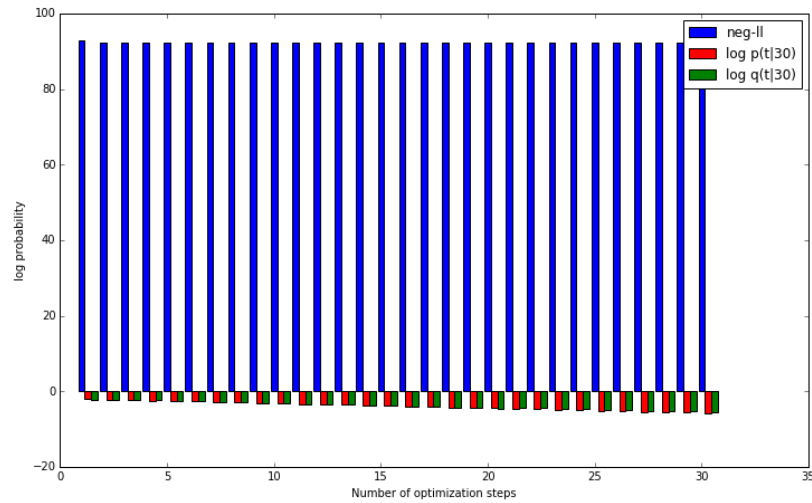


Left: model uses following transformation on output state \mathbf{h}_t $\Delta q_t = \lambda \tanh(\mathbf{w}_q^T \mathbf{h}_t)$

Right: model uses following transformation on output state \mathbf{h}_t $\Delta q_t = \mathbf{w}_q^T \mathbf{h}_t$

Separate KL-divergence components per optimization step

Separate KL-divergence components per optimization step



Trained with
horizon $E[T]=10$

