# A Dualistic View of Activations in Deep Neural Networks

**Ehsan Amid**

**Joint work with:** Rohan Anil, Manfred Warmuth, Abel L. Peirson, Tomer Koren, Yatong Chen, Christopher Fifty, Vlad Fineberg

Google DeepMind

# Overview

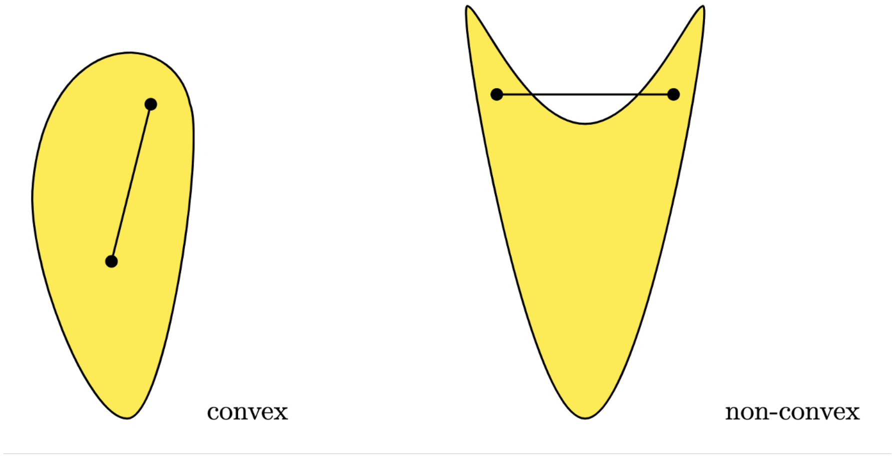
- **Convexity and Duality**
- **A Dualistic View of Activations in DNNs**
- **Application**
  - LocoProp: Local Loss Optimization [AISTATS22]
- **Conclusions and Future Directions**

# Convexity and Duality

**Convex Sets, Convex Functions, Legendre Dual, Bregman Divergence**

# Convex Sets and Convex Functions

A **set** is **convex** if all the points along the line connecting two points inside the set are also belong to the set

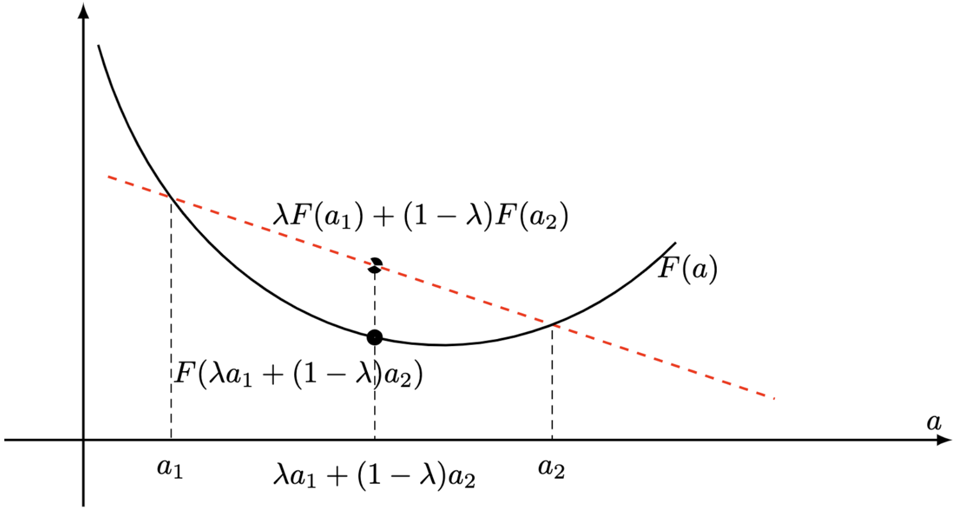



Source code: https://tex.stackexchange.com/questions/639115/how-to-draw-convex-strictly-convex-and-nonconvex-sets

# Convex Sets and Convex Functions

A function is **convex** if it has a convex domain and

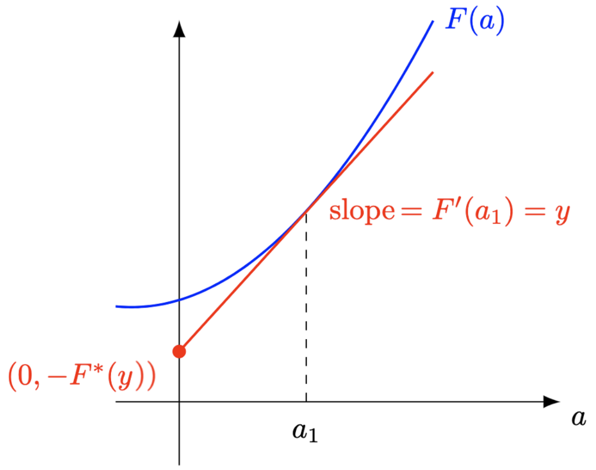
$$F(\lambda\, a_1 + (1-\lambda)\, a_2) \leqslant \lambda\, F(a_1) + (1-\lambda)\, F(a_2)\,, \quad a_1, a_2 \in \mathrm{Dom}(F)\,, \quad \lambda \in [0,1]$$



# Legendre Dual

Given a (convex) function, the **Legendre dual** convex function is given by

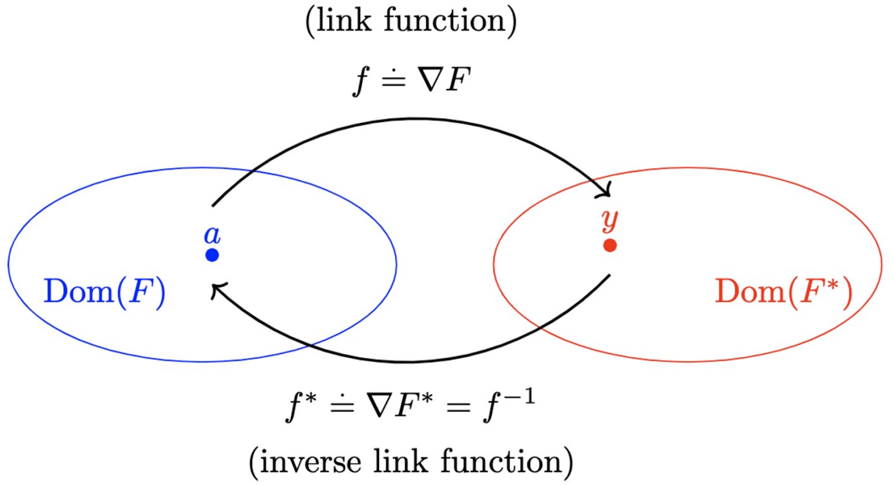
$$F^*(y) = \sup_{\tilde{a} \in \mathrm{Dom}(F)} \{\langle \tilde{a}, y \rangle - F(\tilde{a})\}$$



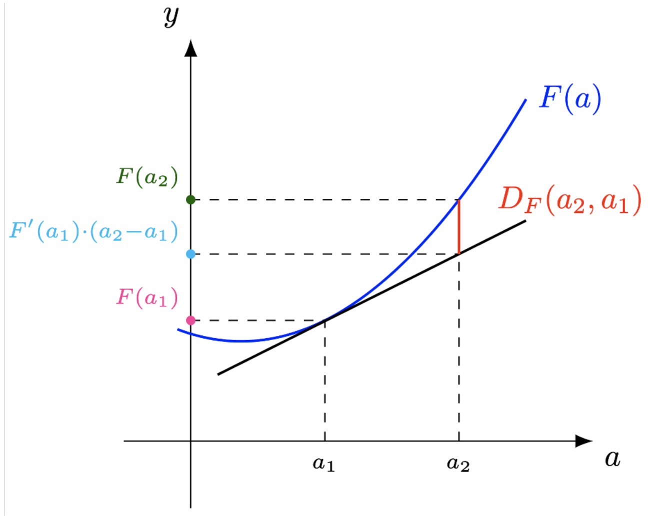
# Legendre Dual

When $F$ is strictly convex, we have the **dually coupled variables**:

$$a = \arg\sup_{\tilde{a} \in \mathrm{Dom}(F)} \{\langle \tilde{a}, y \rangle - F(\tilde{a})\}$$



# Bregman Divergence: Definition




$$D_F(a_2, a_1) = \underbrace{F(a_2) - F(a_1)}_{\text{growth of the convex function}} - \underbrace{\nabla F(a_1)^\top (a_2 - a_1)}_{\text{growth of the linear function}}$$

# Bregman Divergence: Properties

**Convexity:** always in the left argument (not necessarily the right)

**Non-negativity:** $D_F(\hat{\boldsymbol{a}}, \boldsymbol{a}) \geqslant 0$ and $D_F(\hat{\boldsymbol{a}}, \boldsymbol{a}) = 0$ iff $\hat{\boldsymbol{a}} = \boldsymbol{a}$

**Gradient:** $\nabla_{\hat{\boldsymbol{a}}} D_F(\hat{\boldsymbol{a}}, \boldsymbol{a}) = \nabla F(\hat{\boldsymbol{a}}) - \nabla F(\boldsymbol{a})$

**Many well-known cases:**

Squared Euclidean: $D_F(\hat{\boldsymbol{a}}, \boldsymbol{a}) = \frac{1}{2}\|\hat{\boldsymbol{a}} - \boldsymbol{a}\|^2$ (with $F(\boldsymbol{a}) = \frac{1}{2}\|\boldsymbol{a}\|^2$ )

Relative Entropy: $D_F(\hat{\boldsymbol{a}}, \boldsymbol{a}) = \sum_i \{\hat{a}_i \log \frac{\hat{a}_i}{a_i} - \hat{a}_i + a_i\}$ (with $F(\boldsymbol{a}) = \sum_i \{a_i \log a_i - a_i\}$ )

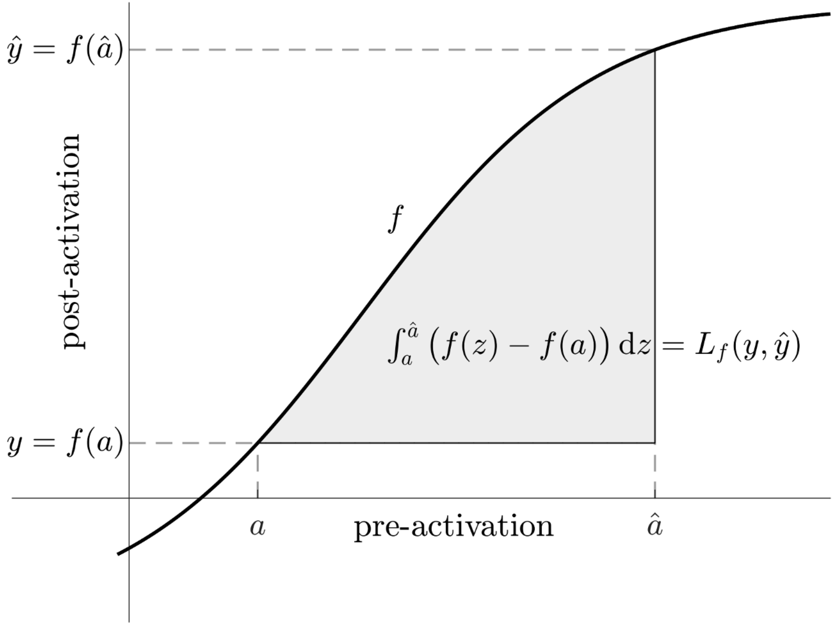
# Matching Loss: Definition

**Definition:** integral under the curve

$$L_f(\boldsymbol{y}, \hat{\boldsymbol{y}}(\hat{\boldsymbol{a}})) \doteq \int_{\boldsymbol{a}}^{\hat{\boldsymbol{a}}} \big(f(\boldsymbol{z}) - f(\boldsymbol{a})\big)^\top \mathrm{d}\boldsymbol{z}$$

… is a Bregman divergence!

$$L_f(\boldsymbol{y}, \hat{\boldsymbol{y}}(\hat{\boldsymbol{a}})) = \Big(F(\boldsymbol{z}) - f(\boldsymbol{a})^\top \boldsymbol{z}\Big)\Big|_{\boldsymbol{a}}^{\hat{\boldsymbol{a}}}$$
$$= F(\hat{\boldsymbol{a}}) - F(\boldsymbol{a}) - f(\boldsymbol{a})^\top(\hat{\boldsymbol{a}} - \boldsymbol{a}) \doteq D_F(\hat{\boldsymbol{a}}, \boldsymbol{a})$$



# Matching Loss Example: Softmax → KL Divergence

Link function:

$$\boldsymbol{y} = f(\boldsymbol{a}) = \text{softmax}(\boldsymbol{a}) = \frac{\exp(\boldsymbol{a})}{\sum_i \exp(a_i)}$$

Integral function:

$$F(\boldsymbol{a}) = \log \sum_i \exp(a_i)$$

Matching loss (in terms of **post-activations**):

$$D_F(\hat{\boldsymbol{a}}, \boldsymbol{a}) = D_{F*}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_i y_i \log \frac{y_i}{\hat{y}_i}$$

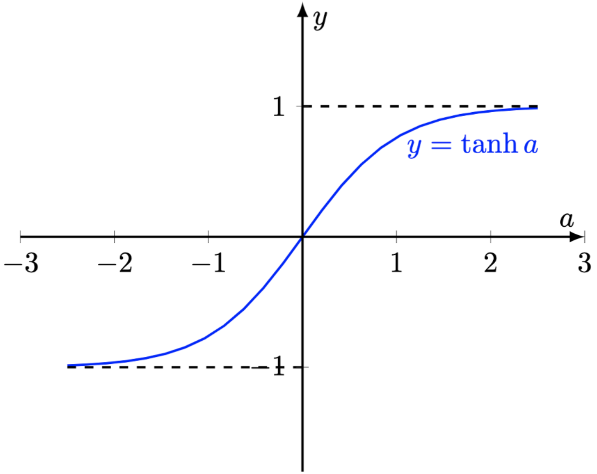
# A Dualistic View of Activations in DNNs

**The Role of Activation Functions**

# Which link functions are valid?

For $f$ to be a "valid" link function (i.e., gradient of a strictly convex function):

- Needs to be **strictly increasing** in 1-D
- And cyclically strictly monotone in higher dimension

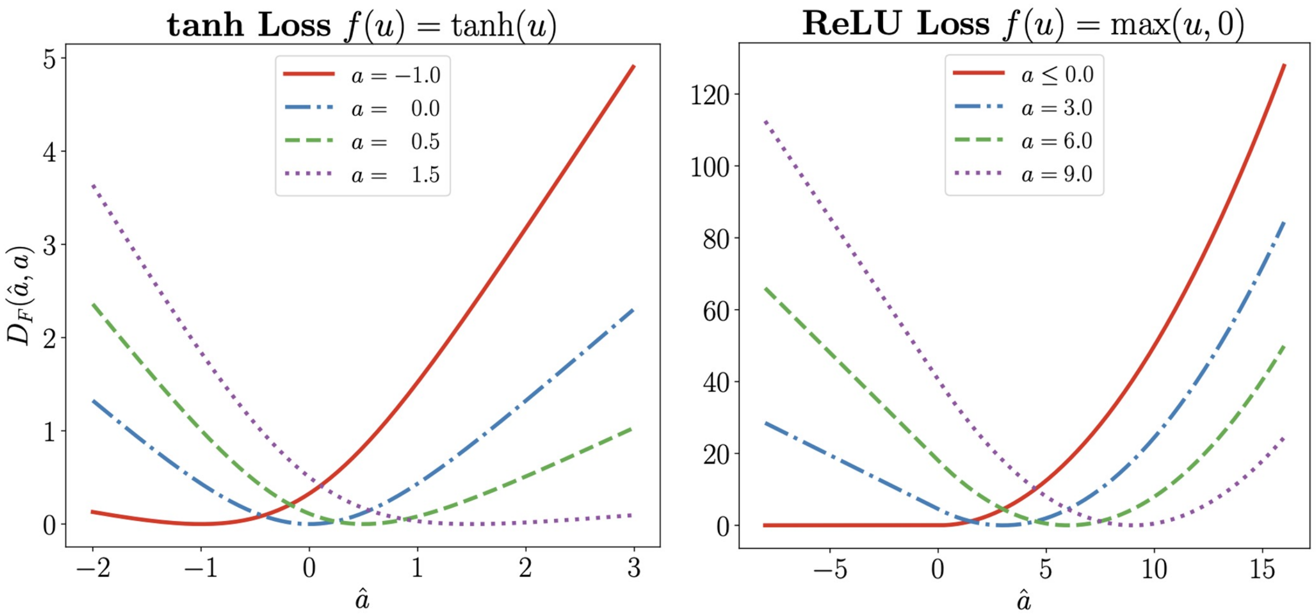



Many **activation functions** in modern neural networks (Linear, Leaky ReLU, tanh, sigmoid, softmax*, etc.) satisfy this property!

# Matching Losses of Common Activation Functions

| Name | Transfer Function $f(\boldsymbol{a})$ | Convex Integral Function $F(\boldsymbol{a})$ | Note |
|---|---|---|---|
| Step Function | $1/2\,(1+\text{sign}(\boldsymbol{a}))$ | $\sum_i \max(a_i, 0)$ | – |
| Linear | $\boldsymbol{a}$ | $1/2\,\|\boldsymbol{a}\|^2$ | – |
| (Leaky) ReLU | $\max(\boldsymbol{a},0) - \beta\max(-\boldsymbol{a},0)$ | $1/2\sum_i a_i\big(\max(a_i,0) - \beta\max(-a_i,0)\big)$ | $\beta \geq 0$ |
| Sigmoid | $(1+\exp(-\boldsymbol{a}))^{-1}$ | $\sum_i \big(a_i + \log(1+\exp(-a_i))\big)$ | – |
| Softmax | $\exp(\boldsymbol{a})/\sum_i \exp(a_i)$ | $\log\sum_i \exp(a_i)$ | – |
| Hyperbolic Tan | $\tanh(\boldsymbol{a})$ | $\sum_i \log\cosh(a_i)$ | – |
| Arc Tan | $\arctan(\boldsymbol{a})$ | $\sum_i \big(a_i \arctan(a_i) - \log\sqrt{1+a_i^2}\big)$ | – |
| SoftPlus | $\log(1+\exp(\boldsymbol{a}))$ | $-\sum_i \text{Li}_2(-\exp(a_i))$ | $\text{Li}_2 :=$ Spence's func. |
| ELU | $[f(\boldsymbol{a})]_i = \begin{cases} a_i & a_i \geq 0 \\ \beta(\exp a_i - 1) & \text{otherwise} \end{cases}$ | $\sum_i \big(a_i^2/2\,\mathbb{I}(a_i \geq 0) + \beta(\exp a_i - a_i - 1)\big)\,\mathbb{I}(a_i < 0)\big)$ | $\beta \geq 0$ |

# Matching Losses of Common Activation Functions



# Application

**LocoProp: Local Loss Optimization**

# Local Loss Optimization

LocoProp conceives neural networks as a **modular composition** of layers

# Local Loss Optimization

**Pre** $\hat{\boldsymbol{a}}_m = \boldsymbol{W}_m \hat{\boldsymbol{y}}_{m-1}$ and **post** $\hat{\boldsymbol{y}}_m = f_m(\hat{\boldsymbol{a}}_m)$ activations in layer $m$

$$\boldsymbol{W}^{\text{new}} = \arg\min_{\widetilde{\boldsymbol{W}}} \Big\{ \underbrace{D(\widetilde{\boldsymbol{W}} \hat{\boldsymbol{y}}_{m-1}, \boldsymbol{a}_m)}_{\text{loss}} + \underbrace{R(\widetilde{\boldsymbol{W}}, \boldsymbol{W})}_{\text{regularizer}} \Big\}$$

target $\boldsymbol{a}_m$ (or $\boldsymbol{y}_m = f_m(\boldsymbol{a}_m)$)

# The Case of Squared Loss

Local regularized squared loss:

$$\tfrac{1}{2}\,\|\widetilde{\boldsymbol{W}}\,\hat{\boldsymbol{y}}_{m-1} - \boldsymbol{a}_m\|^2 + \tfrac{1}{2\eta}\,\|\widetilde{\boldsymbol{W}} - \boldsymbol{W}_m\|^2 \quad \text{where} \quad \boldsymbol{a}_m = \hat{\boldsymbol{a}}_m - \gamma\,\nabla_{\hat{\boldsymbol{a}}_m} L(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

loss to the target — keep the weight close — GD w.r.t. the final loss

Solution: fixed point equation

$$\boldsymbol{W}_m^{\text{new}} = \boldsymbol{W}_m - \eta\,\big(\boldsymbol{W}_m^{\text{new}}\,\hat{\boldsymbol{y}}_{m-1} - \boldsymbol{a}_m\big)\,\hat{\boldsymbol{y}}_{m-1}^\top$$

# The Case of Squared Loss

How about a single iteration?

$$\begin{aligned}
\boxed{\boldsymbol{W}_m^{\text{new}}} &\approx \boxed{\boldsymbol{W}_m} - \eta\,\big(\boxed{\boldsymbol{W}_m}\hat{\boldsymbol{y}}_{m-1} - \boldsymbol{a}_m\big)\,\hat{\boldsymbol{y}}_{m-1}^\top \\
&= \boldsymbol{W}_m - \eta\,\Big(\boldsymbol{W}_m\hat{\boldsymbol{y}}_{m-1} \\
&\qquad - \big(\boldsymbol{W}_m\hat{\boldsymbol{y}}_{m-1} - \gamma\,\nabla_{\hat{\boldsymbol{a}}_m} L(\boldsymbol{y},\hat{\boldsymbol{y}})\big)\Big)\,\hat{\boldsymbol{y}}_{m-1}^\top \\
&= \boldsymbol{W}_m - \eta\,\gamma\,\nabla_{\hat{\boldsymbol{a}}_m} L(\boldsymbol{y},\hat{\boldsymbol{y}})\,\hat{\boldsymbol{y}}_{m-1}^\top \\
&= \boldsymbol{W}_m - \eta_e \boxed{\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\cancel{\partial\hat{\boldsymbol{a}}_m}}\,\frac{\cancel{\partial\hat{\boldsymbol{a}}_m}}{\partial \boldsymbol{W}_m}}\,. \qquad \text{(BackProp)}
\end{aligned}$$

This is just a single step of **BackProp**!

# The Case of Squared Loss

$$\boldsymbol{W}_m^{\text{new}} = \boldsymbol{W}_m - \eta \left(\boldsymbol{W}_m^{\text{new}} \hat{\boldsymbol{y}}_{m-1} - \boldsymbol{a}_m\right) \hat{\boldsymbol{y}}_{m-1}^\top$$

There is a closed-form solution:

$$\boldsymbol{W}_m^{\text{new}} = \boldsymbol{W}_m - \eta_e \nabla_{\boldsymbol{W}_m} L(\boldsymbol{y}, \hat{\boldsymbol{y}}) \left(\boldsymbol{I} + \eta\, \hat{\boldsymbol{y}}_{m-1} \hat{\boldsymbol{y}}_{m-1}^\top\right)^{-1}$$

gradient descent

preconditioner matrix

A **preconditioned** gradient descent!

# Using Matching Losses

Replace the squared loss with the matching loss:

$$\frac{1}{2}\|\widetilde{\boldsymbol{W}}\hat{\boldsymbol{y}}_{m-1} - \boldsymbol{a}_m\|^2 + \frac{1}{2\eta}\|\widetilde{\boldsymbol{W}} - \boldsymbol{W}_m\|^2 \quad \text{where} \quad \boldsymbol{a}_m = \hat{\boldsymbol{a}}_m - \gamma\nabla_{\hat{\boldsymbol{a}}_m} L(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

$$D_{F_m^*}\big(\boldsymbol{y}_m, f_m(\widetilde{\boldsymbol{W}}\hat{\boldsymbol{y}}_{m-1})\big) + \frac{1}{2\eta}\|\widetilde{\boldsymbol{W}} - \boldsymbol{W}_m\|^2 \quad \text{where} \quad \boldsymbol{y}_m = \hat{\boldsymbol{y}}_m - \gamma\nabla_{\hat{\boldsymbol{a}}_m} L(\boldsymbol{y}, \hat{\boldsymbol{y}})$$

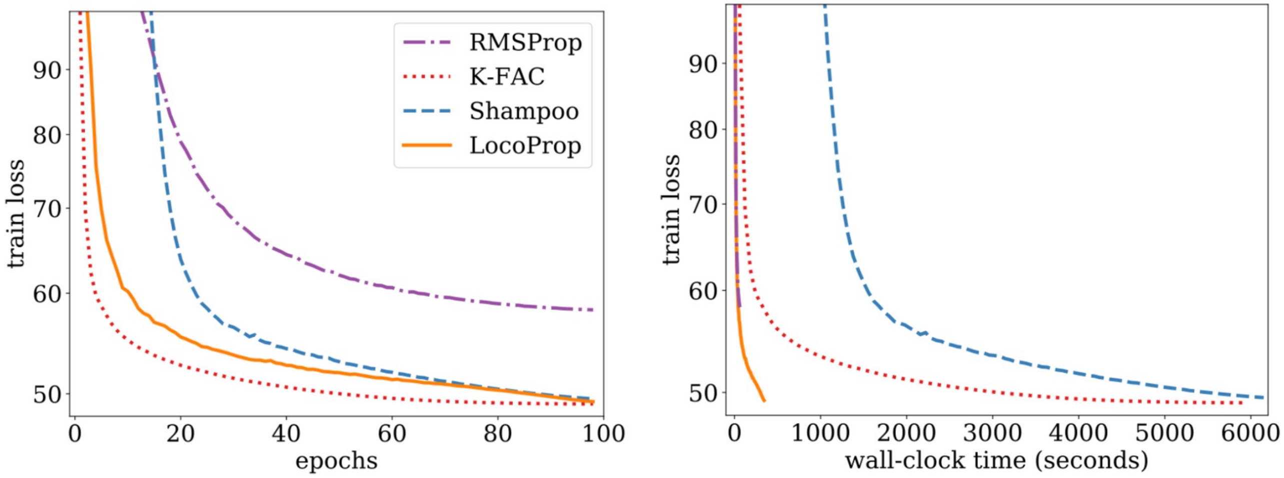
target post-activation

MD w.r.t. the final loss

Still convex in the **weights**, and yields a preconditioned update

# Experiments

Results on the deep autoencoder benchmark




**LocoProp** performs competitive to **second-order** methods

# Conclusions and Future Directions

# Conclusions and Future Directions

Further applications:

- Layerwise Fisher approximation via local sampling [NerIPS22-HITY]
- Bregman knowledge distillation [TMLR23]
- Robust bi-tempered loss [NeurIPS19]

Future directions:

- No-sample layerwise Fisher approximation
- Extension to non-monotonic activations
- Low-rank compression of the weights
- Learning losses and activation functions
- ...