Introduction to deep learning

Andreas Damianou

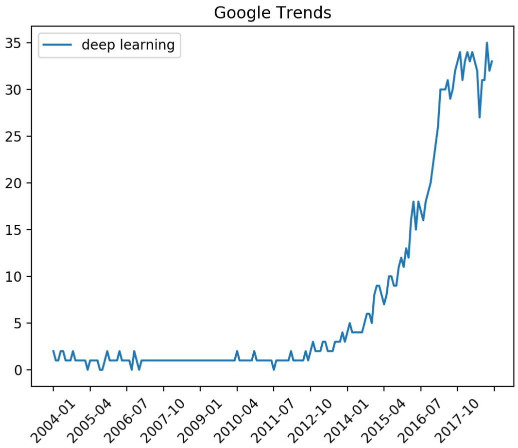
Amazon, Cambridge, UK

Royal Statistical Society, London
13 Dec. 2018

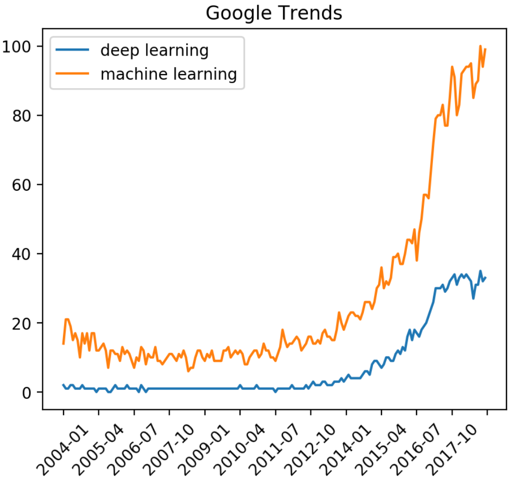
Notebook

http://adamian.github.io/talks/Damianou_DL_tutorial_18.ipynb

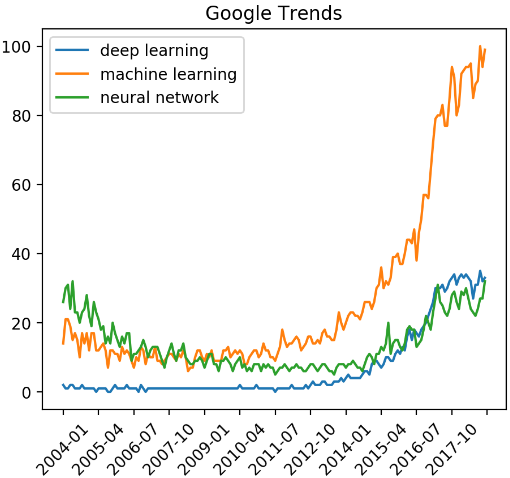
Starting with a cliché...



Starting with a cliché...



Starting with a cliché...



Deep neural networks: hierarchical function definitions

A neural network is a composition of functions (layers), each parameterized with a *weight vector* $\mathbf{w}_l$. E.g. for 2 layers:

$$f_{\text{net}} = h_2(h_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

Generally $f_{\text{net}} : \mathbf{x} \mapsto \mathbf{y}$ with:

$$\begin{aligned}
\mathbf{h}_1 &= \varphi(\mathbf{x}\mathbf{w}_1 + b_1) \\
\mathbf{h}_2 &= \varphi(\mathbf{h}_1\mathbf{w}_2 + b_2) \\
&\cdots \\
\hat{\mathbf{y}} &= \varphi(\mathbf{h}_{L-1}\mathbf{w}_L + b_L)
\end{aligned}$$

ϕ is the (non-linear) activation function.

Deep neural networks: hierarchical function definitions

A neural network is a composition of functions (layers), each parameterized with a *weight vector* $\mathbf{w}_l$. E.g. for 2 layers:

$$f_{\text{net}} = h_2(h_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

Generally $f_{\text{net}} : \mathbf{x} \mapsto \mathbf{y}$ with:

$$\begin{aligned}
\mathbf{h}_1 &= \varphi(\mathbf{x}\mathbf{w}_1 + b_1) \\
\mathbf{h}_2 &= \varphi(\mathbf{h}_1\mathbf{w}_2 + b_2) \\
&\cdots \\
\hat{\mathbf{y}} &= \varphi(\mathbf{h}_{L-1}\mathbf{w}_L + b_L)
\end{aligned}$$

ϕ is the (non-linear) activation function.

Defining the loss

- We have our function approximator $f_{\text{net}}(x) = \hat{y}$
- We have to define our loss (objective function) to relate this function outputs to the observed data.
- E.g. squared difference $\sum_n (y_n - \hat{y}_n)^2$ or cross-entropy

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\text{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \lambda \underbrace{\sum_i \parallel \mathbf{w}_i \parallel}_{\text{regularizer}}$$

- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

- Optimization still done with back-prop (i.e. gradient descent).

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\mathsf{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \lambda \underbrace{\sum_{i} \parallel \mathbf{w}_i \parallel}_{\text{regularizer}}$$

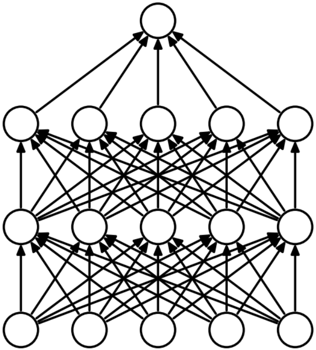
- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

- Optimization still done with back-prop (i.e. gradient descent).

Graphical depiction



Optimization

One layer:

$$Loss = \frac{1}{2}(\mathbf{h} - \mathbf{y})^2$$
$$\mathbf{h} = \phi(\mathbf{xw})$$
$$\frac{\vartheta Loss}{\vartheta \mathbf{w}} = \underbrace{(\mathbf{y} - \mathbf{h})}_{\epsilon} \frac{\vartheta \phi(\mathbf{xw})}{\vartheta \mathbf{w}}$$

Two layers:

$$Loss = \frac{1}{2}(\mathbf{h}_2 - \mathbf{y})^2$$
$$\mathbf{h}_2 = \phi \left[\underbrace{\phi(\mathbf{xw}_0)}_{\mathbf{h}_1} \mathbf{w}_1 \right]$$
$$\frac{\vartheta Loss}{\vartheta \mathbf{w}_0} = \cdots$$
$$\frac{\vartheta Loss}{\vartheta \mathbf{w}_1} = \cdots$$

Derivative w.r.t $\mathbf{w}_1$

$$
\begin{aligned}
\frac{\vartheta(\mathbf{h}_2-\mathbf{y})^2}{\vartheta\mathbf{w}_1} &= -2\frac{1}{2}(\mathbf{h}_2-\mathbf{y})\frac{\vartheta\mathbf{h}_2}{\vartheta\mathbf{w}1} = \\
&= (\mathbf{y}-\mathbf{h}_2)\frac{\vartheta\phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta\mathbf{w}_1} = \\
&= (\mathbf{y}-\mathbf{h}_2)\frac{\vartheta\phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta\mathbf{h}_1\mathbf{w}_1}\frac{\vartheta\mathbf{h}_1\mathbf{w}_1}{\vartheta\mathbf{w}_1} = \\
&= \underbrace{(\mathbf{y}-\mathbf{h}_2)}_{\epsilon_2}\underbrace{\frac{\vartheta\phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta\mathbf{h}_1\mathbf{w}_1}}_{g_1}\mathbf{h}_1^T
\end{aligned}
$$

$\mathbf{h}_1$ is computed during the *forward pass*.

Derivative w.r.t $\mathbf{w}_0$

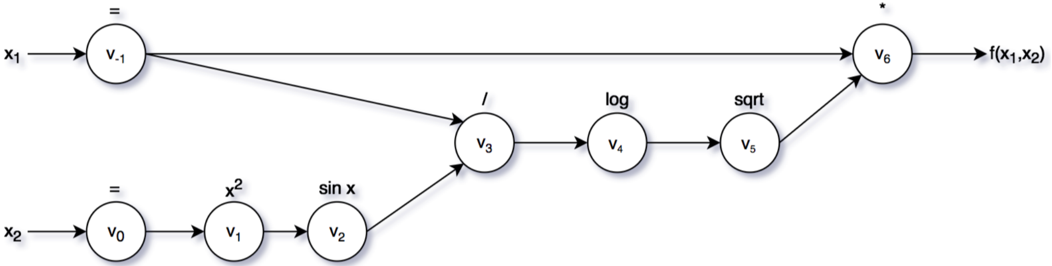
$$
\begin{aligned}
\frac{\vartheta(\mathbf{h}_2-\mathbf{y})^2}{\vartheta\mathbf{w}_0} &= -2\frac{1}{2}(\mathbf{h}_2-\mathbf{y})\frac{\vartheta\mathbf{h}_2}{\vartheta\mathbf{w}_0} = \\
&= (\mathbf{y}-\mathbf{h}_2)\frac{\vartheta\phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta\mathbf{h}_1\mathbf{w}_1}\frac{\vartheta\mathbf{h}_1\mathbf{w}_1}{\vartheta\mathbf{h}_1}\frac{\vartheta\mathbf{h}_1}{\vartheta\mathbf{w}_0} = \\
&= \epsilon_2\ g_1\ \mathbf{w}_1^T\ \frac{\vartheta\phi(\mathbf{x}\mathbf{w}_0)}{\mathbf{x}\mathbf{w}_0}\frac{\vartheta\mathbf{x}\mathbf{w}_0}{\vartheta\mathbf{w}_0} = \\
&= \epsilon_2\ g_1\ \mathbf{w}_1^T\ \underbrace{\frac{\vartheta\phi(\mathbf{x}\mathbf{w}_0)}{\vartheta\mathbf{x}\mathbf{w}_0}}_{g_0}\ \mathbf{x}^T
\end{aligned}
$$

Propagation of error is just the chain rule.

Go to notebook!

Automatic differentiation

Example: $f(x_1, x_2) = x_1\sqrt{\log \frac{x_1}{sin(x_2^2)}}$ has *symbolic graph*:



(image: sanyamkapoor.com)

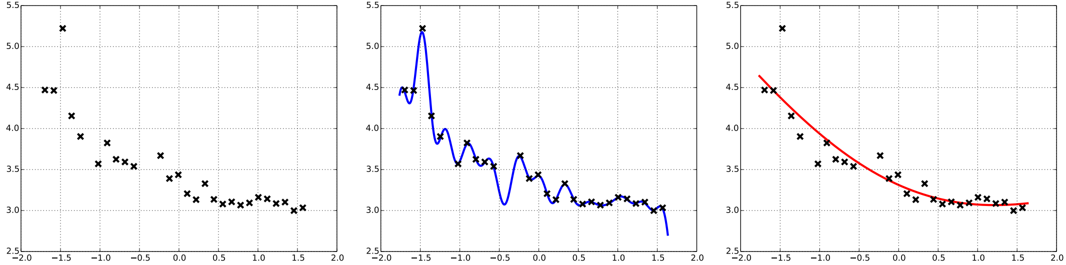
A NN in mxnet

Back to notebook!

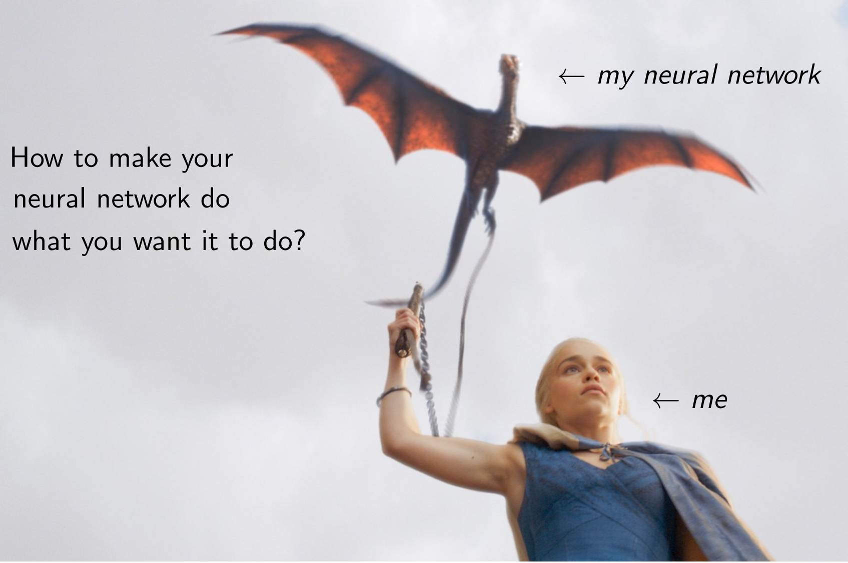
We're far from done...

- How to initialize the (so many) parameters?
- How to pick the right architecture?
- Layers and parameters co-adapt.
- Multiple local optima in optimization surface.
- Numerical problems.
- Bad behaviour of composite function (e.g. problematic gradient distribution).
- *OVERFITTING*

Curve fitting [skip]



Taming the dragon



Lottery ticket hypothesis

Might provide intuition for many of the tricks used.

- Optimization landscape: multiple optima and difficult to navigate
- Over-parameterized networks contain multiple sub-networks ("lottery tickets")
- "Winning ticket": a lucky sub-network found a good solution
- Over-parameterization: more tickets, higher winning probability
- Of course this means we have to prune or at least regularize.

(Frankle and Carbin (2018))

“Tricks”

- Smart initializations
- ReLU: better behaviour of gradients
- Early stopping: prevent overfitting
- Dropout
- Batch-normalization
- Transfer/meta-learning/BO: guide the training with another model
- many other “tricks”

Vanishing and exploding gradients

$$\begin{aligned}\frac{\vartheta(\mathbf{h}_2-\mathbf{y})^2}{\vartheta\mathbf{w}_0} &= -2\frac{1}{2}(\mathbf{h}_2-\mathbf{y})\frac{\vartheta\mathbf{h}_2}{\vartheta\mathbf{w}_0} = \\ &= (\mathbf{y}-\mathbf{h}_2)\frac{\vartheta\phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta\mathbf{h}_1\mathbf{w}_1}\frac{\vartheta\mathbf{h}_1\mathbf{w}_1}{\vartheta\mathbf{h}_1}\frac{\vartheta\mathbf{h}_1}{\vartheta\mathbf{w}_0} = \\ &= \epsilon_2\ g_1\ \mathbf{w}_1^T\ \frac{\vartheta\phi(\mathbf{x}\mathbf{w}_0)}{\mathbf{x}\mathbf{w}_0}\frac{\vartheta\mathbf{x}\mathbf{w}_0}{\vartheta\mathbf{w}_0} = \\ &= \epsilon_2\ g_1\ \mathbf{w}_1^T\underbrace{\frac{\vartheta\phi(\mathbf{x}\mathbf{w}_0)}{\vartheta\mathbf{x}\mathbf{w}_0}}_{g_0}\ \mathbf{x}^T\end{aligned}$$

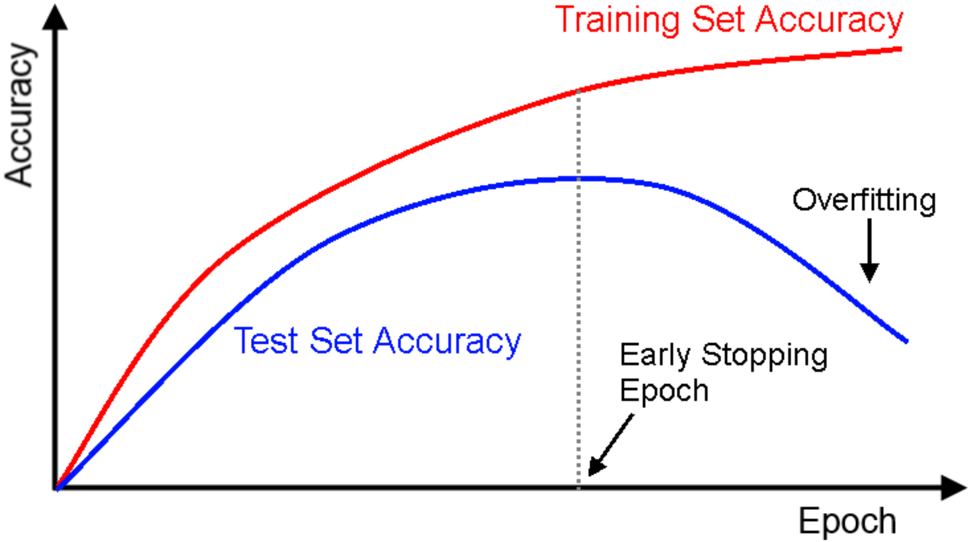
- ReLU: an activation function leading to well-behaved gradients.

Vanishing and exploding gradients

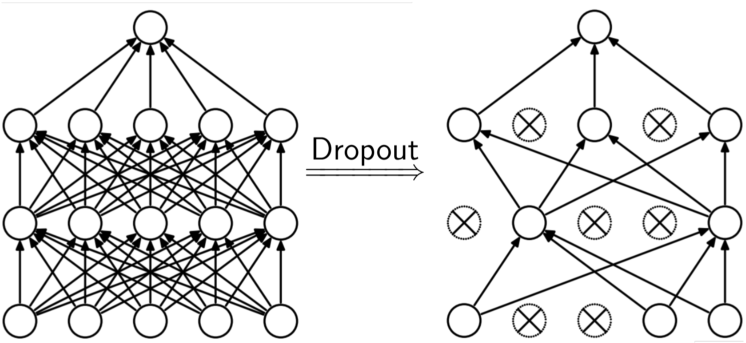
$$
\begin{aligned}
\frac{\vartheta(\mathbf{h}_2 - \mathbf{y})^2}{\vartheta \mathbf{w}_0} &= -2\frac{1}{2}(\mathbf{h}_2 - \mathbf{y})\frac{\vartheta \mathbf{h}_2}{\vartheta \mathbf{w}_0} = \\
&= (\mathbf{y} - \mathbf{h}_2)\frac{\vartheta \phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta \mathbf{h}_1\mathbf{w}_1}\frac{\vartheta \mathbf{h}_1\mathbf{w}_1}{\vartheta \mathbf{h}_1}\frac{\vartheta \mathbf{h}_1}{\vartheta \mathbf{w}_0} = \\
&= \epsilon_2 \; g_1 \; \mathbf{w}_1^T \; \frac{\vartheta \phi(\mathbf{x}\mathbf{w}_0)}{\mathbf{x}\mathbf{w}_0}\frac{\vartheta \mathbf{x}\mathbf{w}_0}{\vartheta \mathbf{w}_0} = \\
&= \epsilon_2 \; g_1 \; \mathbf{w}_1^T \underbrace{\frac{\vartheta \phi(\mathbf{x}\mathbf{w}_0)}{\vartheta \mathbf{x}\mathbf{w}_0}}_{g_0} \; \mathbf{x}^T
\end{aligned}
$$

- ReLU: an activation function leading to well-behaved gradients.

Early stopping



Dropout



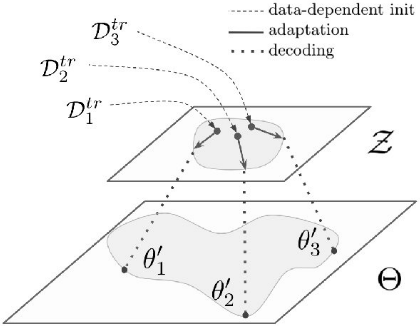
- Randomly drop units during training.
- Prevents units from co-adapting too much and prevents overfitting.

Batch-normalization

- Normalize each layer's output so e.g. $\mu = 0, \sigma = 1$
- Reduces covariate shift (data distribution changes)
- Less co-adaptation of layers
- Overall: faster convergence

Meta-learning

- Optimize the neural network model with the help of another model.
- The helper model might be allowed to learn from multiple datasets.

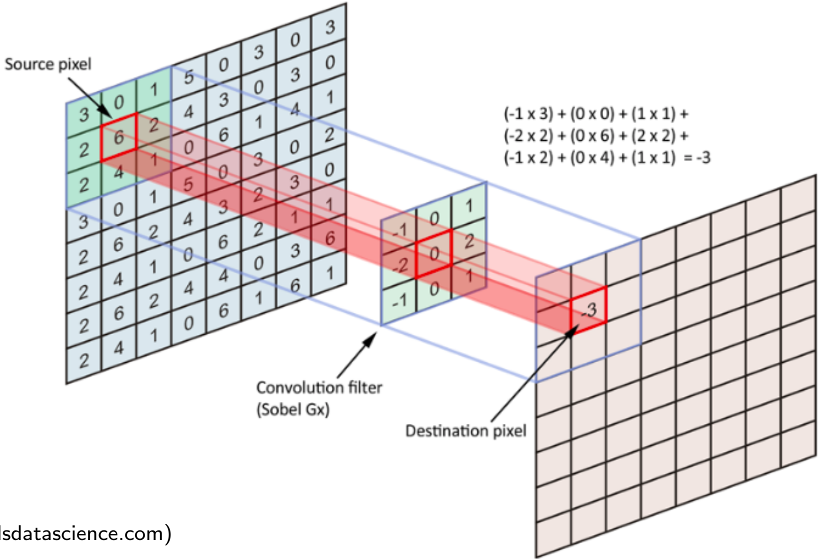


(image: Rusu et al. 2018 - LEO)

Bayesian HPO

- Hyperparameters: learning rate, weight decay, architectures, learning protocols
- Optimize them using Bayesian optimization
- Prediction of learning curves. Can speed up HPO in a bandit setting
- Example: `https://xfer.readthedocs.io/en/master/demos/xfer-hpo.html`

Convolutional NN



(image: towardsdatascience.com)

Recurrent NN

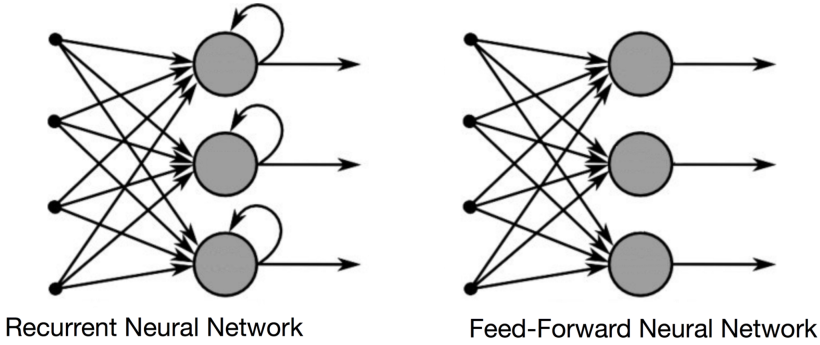


image: towardsdatascience.com

Deployment: Transfer learning

Training neural networks from scratch is not practical as this requires:

- a lot of data
- expertise
- compute (e.g. GPU machines)

Solution:

- Transfer learning. Repurposing pre-trained neural networks to solve new tasks.
- A library for transfer learning: `https://github.com/amzn/xfer`

Go to Notebook!

Deployment: Transfer learning

Training neural networks from scratch is not practical as this requires:

- a lot of data
- expertise
- compute (e.g. GPU machines)

Solution:

- Transfer learning. Repurposing pre-trained neural networks to solve new tasks.
- A library for transfer learning: `https://github.com/amzn/xfer`

Go to Notebook!

Bayesian deep learning

We saw that optimizing the parameters is a challenge.
Why not marginalize them out completely?

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\text{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \lambda \underbrace{\sum_i \| \mathbf{w}_i \|}_{\text{regularizer}}$$

- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

- Optimization still done with back-prop (i.e. gradient descent).

Integrating out weights

$D := (\mathbf{x}, \mathbf{y})$

$$p(w|D) = \frac{p(D|w)p(w)}{p(D) = \int p(D|w)p(w)\mathsf{d}w}$$

Inference

- $p(D)$ (and hence $p(w|D)$) is difficult to compute because of the nonlinear way in which w appears through g.
- Attempt at *variational inference*:

$$\underbrace{\mathrm{KL}\left(q(w;\theta)\,\|\,p(w|D)\right)}_{\text{minimize}} = \log(p(D)) - \underbrace{\mathcal{L}(\theta)}_{\text{maximize}}$$

where

$$\mathcal{L}(\theta) = \underbrace{\mathbb{E}_{q(w;\theta)}[\log p(D,w)]}_{\mathcal{F}} + \mathbb{H}\left[q(w;\theta)\right]$$

- Term in red is still problematic. Solution: MC.
- Such approaches can be formulated as *black-box* inferences.

Inference

- $p(D)$ (and hence $p(w|D)$) is difficult to compute because of the nonlinear way in which w appears through g.

- Attempt at *variational inference*:

$$\underbrace{\mathsf{KL}\left(q(w;\theta)\,\|\,p(w|D)\right)}_{\text{minimize}} = \log(p(D)) - \underbrace{\mathcal{L}(\theta)}_{\text{maximize}}$$

where

$$\mathcal{L}(\theta) = \underbrace{\mathbb{E}_{q(w;\theta)}\left[\log p(D, w)\right]}_{\mathcal{F}} + \mathbb{H}\left[q(w;\theta)\right]$$

- Term in red is still problematic. Solution: MC.

- Such approaches can be formulated as *black-box* inferences.

Inference

- $p(D)$ (and hence $p(w|D)$) is difficult to compute because of the nonlinear way in which w appears through g.

- Attempt at *variational inference*:

$$\underbrace{\mathsf{KL}\left(q(w;\theta)\,\|\,p(w|D)\right)}_{\text{minimize}} = \log(p(D)) - \underbrace{\mathcal{L}(\theta)}_{\text{maximize}}$$

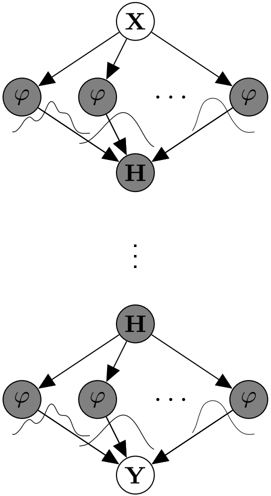
  where

$$\mathcal{L}(\theta) = \underbrace{\mathbb{E}_{q(w;\theta)}\left[\log p(D,w)\right]}_{\mathcal{F}} + \mathbb{H}\left[q(w;\theta)\right]$$

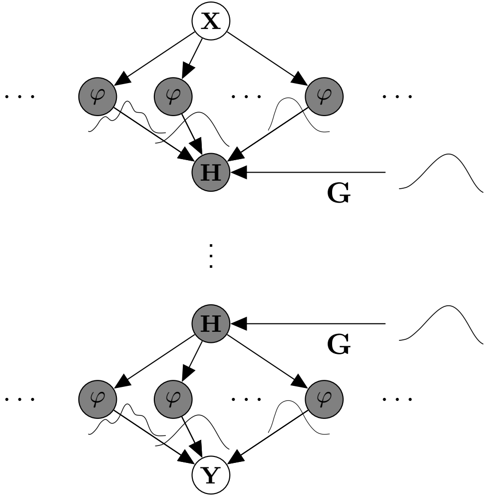
- Term in red is still problematic. Solution: MC.

- Such approaches can be formulated as *black-box* inferences.

Bayesian neural network *(what we saw before)*

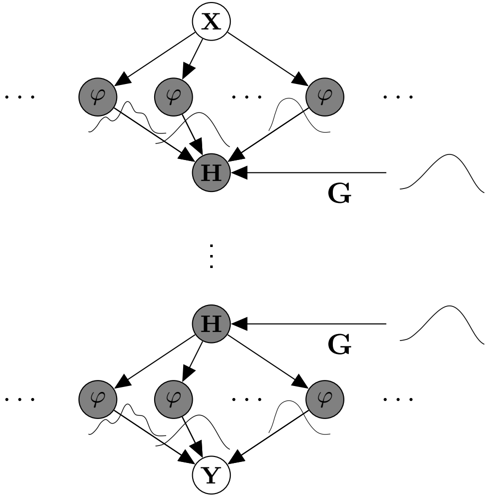


From NN to GP



- NN: $\mathbf{H}_2 = \mathbf{W}_2\phi(\mathbf{H}_1)$
- GP: ϕ is $\infty-$dimensional so:
  $\mathbf{H}_2 = f_2(\mathbf{H}_1; \theta_2) + \epsilon$
- NN: $p(\mathbf{W})$
- GP: $p(f(\cdot))$

From NN to GP



- NN: $\mathbf{H}_2 = \mathbf{W}_2 \phi(\mathbf{H}_1)$
- GP: ϕ is ∞−dimensional so:
  $\mathbf{H}_2 = f_2(\mathbf{H}_1; \theta_2) + \epsilon$
- NN: $p(\mathbf{W})$
- GP: $p(f(\cdot))$

Summary

- Vanilla feedforward NN with backpropabation (chain rule)
- Automatic differentiation
- Practical issues and solutions ("tricks")
- Understanding the challenges: optimization landscape and capacity
- ConvNets and RNNs
- Transfer Learning for practical use
- Bayesian NNs

Conclusions

- NNs are mathematically simple; challenge is in how to optimize them.
- Data efficiency? Uncertainty calibration? Interpretability? Safety? ...