# Introduction to deep learning

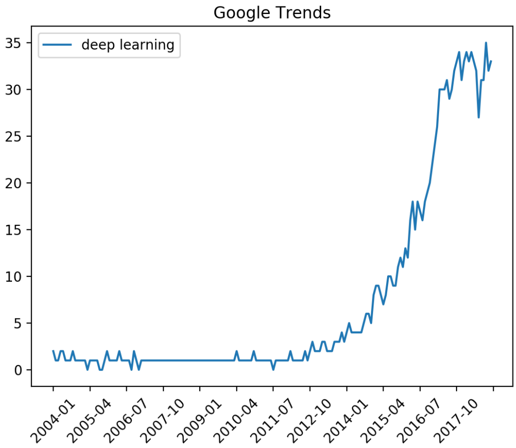
Andreas Damianou

Amazon, Cambridge, UK

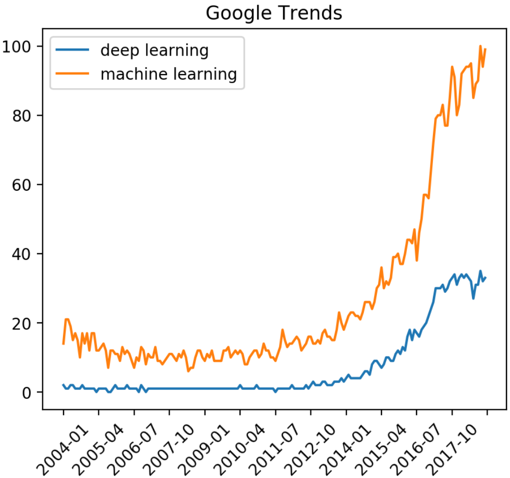
*Royal Statistical Society, London*
13 Dec. 2018

amazon.com

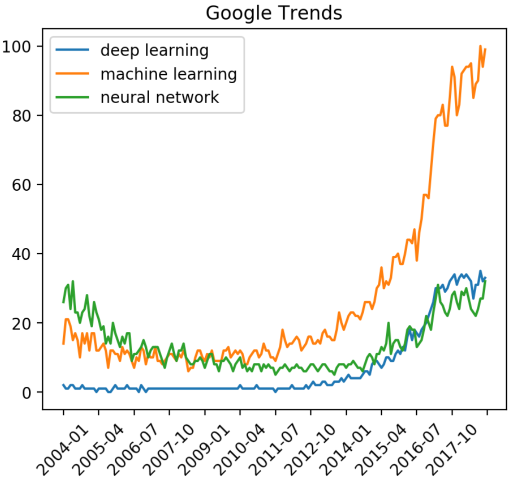
# Starting with a cliché...



# 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* $\mathbf{w}_l$:

$$f_{1,2} = f_2(f_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

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

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

# Deep neural networks: hierarchical function definitions

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

$$f_{1,2} = f_2(f_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

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

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

# Defining the loss

- We have our function approximator $f_net(x) = \hat{y}$
- We have to define our loss (objective function) to relate this function outputs to the observed data.
- Usual choices: softmax (regression) or cross-entropy (classification).

# Probabilistic re-formulation

- Training minimizing loss:

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

- 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).

# Probabilistic re-formulation

- Training minimizing loss:

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

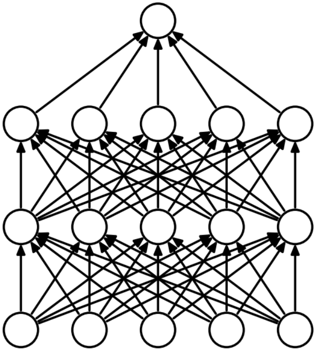
- Equivalent probabilistic view for regression, maximizing posterior probability:

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

  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



# Single layer

One layer:

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

Two layers:

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

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

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

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

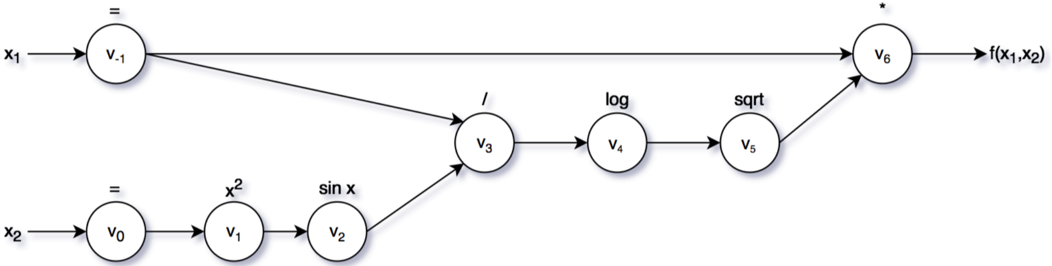
$$
\begin{aligned}
\frac{\vartheta(\mathbf{h}_2 - \mathbf{y})^2}{\vartheta \mathbf{w}_0} &= -2\frac{1}{2}(\mathbf{h}_2 - \mathbf{y})\frac{\vartheta \mathbf{h}_2}{\vartheta \mathbf{w}_0} = \\
&= (\mathbf{y} - \mathbf{h}_2)\frac{\vartheta \phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta \mathbf{h}_1\mathbf{w}_1}\frac{\vartheta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

Propagation of error is just the chain rule.

*Go to notebook!*

# Automatic differentiation

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




*(image: sanyamkapoor.com)*

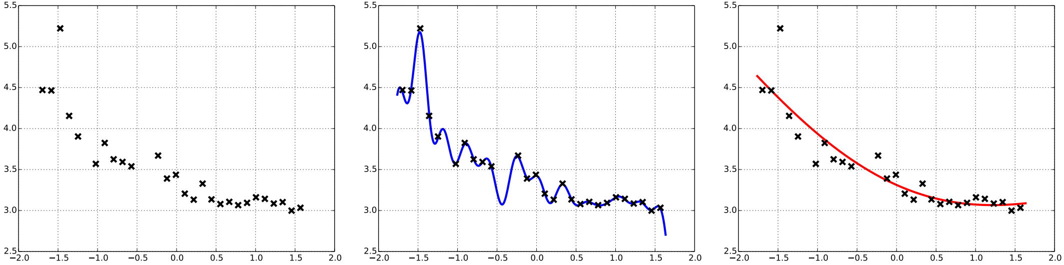
# A NN in mxnet

*Back to notebook!*

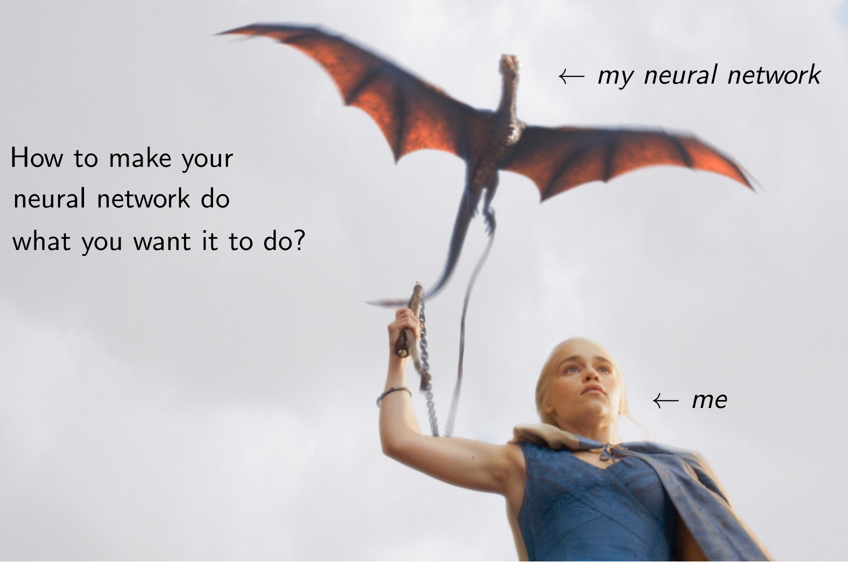
# We're far from done...

- 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

# “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

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

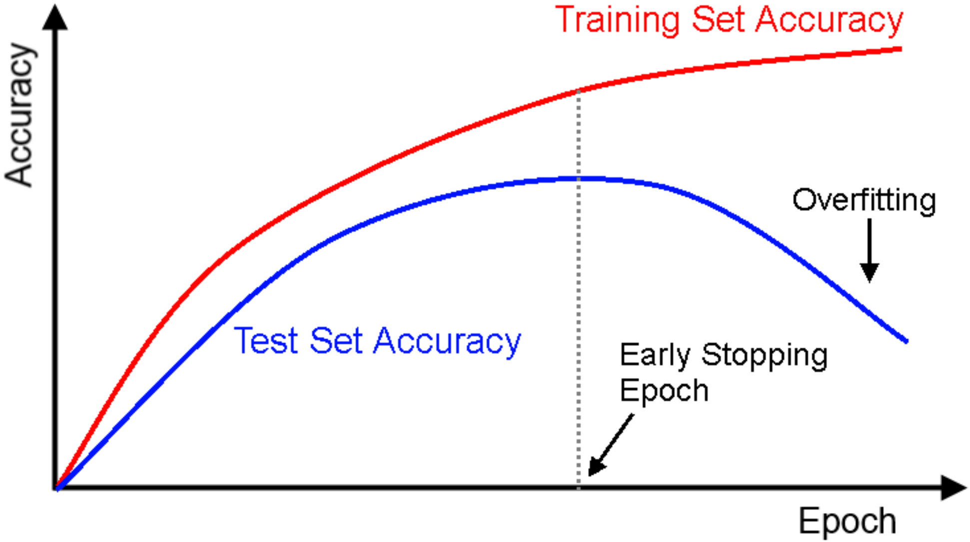
- ReLU: an activation function leading to well-behaved gradients.

# Vanishing and exploding gradients

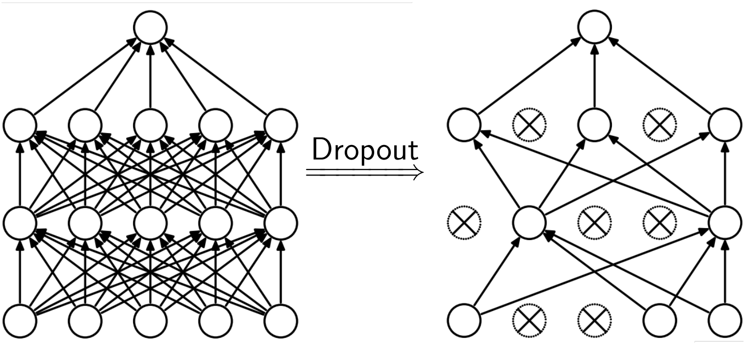
$$
\begin{aligned}
\frac{\vartheta(\mathbf{h}_2-\mathbf{y})^2}{\vartheta\mathbf{w}_0} &= -2\frac{1}{2}(\mathbf{h}_2-\mathbf{y})\frac{\vartheta\mathbf{h}_2}{\vartheta\mathbf{w}_0} = \\
&= (\mathbf{y}-\mathbf{h}_2)\frac{\vartheta\phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta\mathbf{h}_1\mathbf{w}_1}\frac{\vartheta\mathbf{h}_1\mathbf{w}_1}{\vartheta\mathbf{h}_1}\frac{\v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

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

# Early stopping



# Dropout



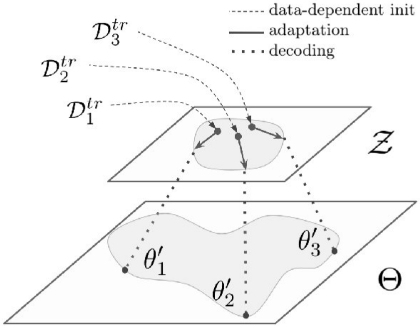

- Randomly drop units during training.
- Prevents units from co-adapting too much and prevents overfitting.

# 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

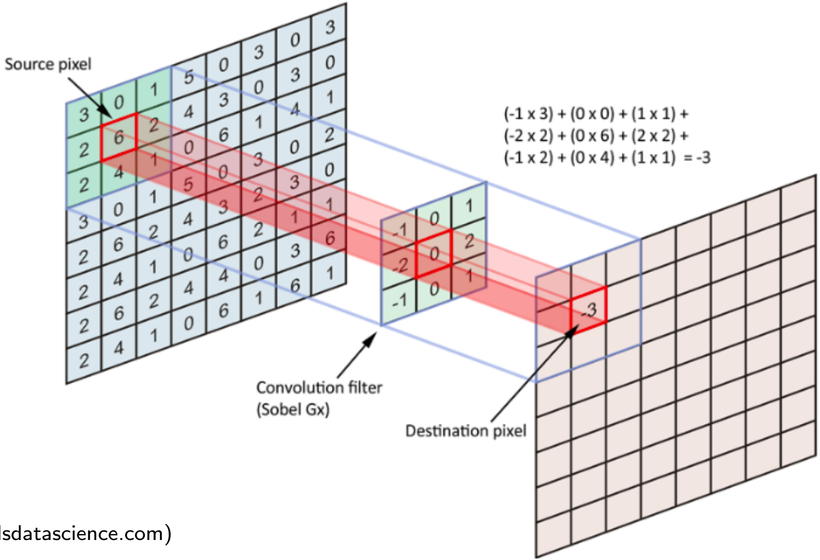
# Meta-learning

- 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)*

# Convolutional NN




(image: towardsdatascience.com)

# Recurrent NN

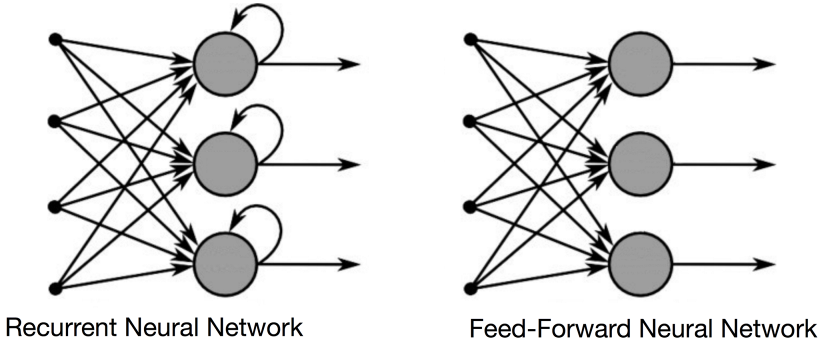



*image: towardsdatascience.com*

# 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.

*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.

*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:

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

- 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*:

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

where

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

- 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

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

- 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*:

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

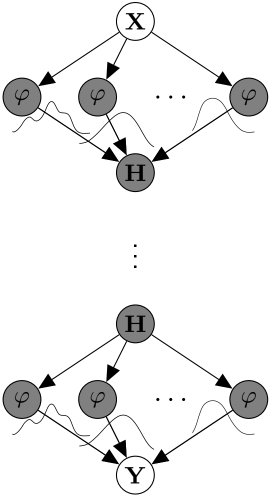
  where

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

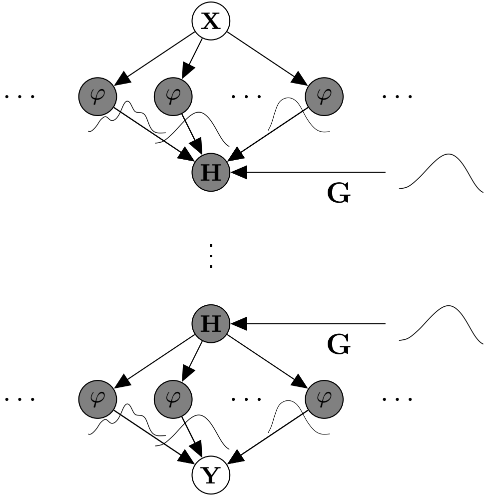
- Term in red is still problematic. Solution: MC.

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

# Bayesian neural network *(what we saw before)*

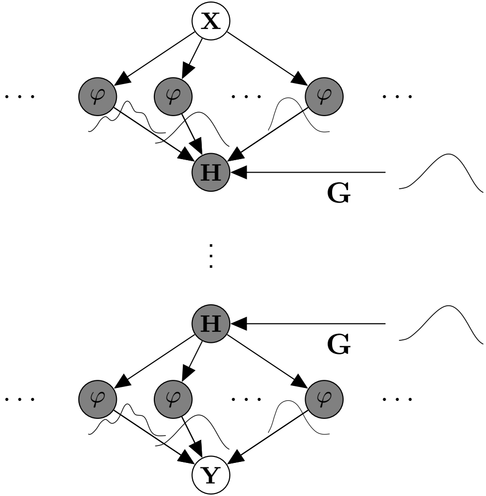


# From NN to GP




- NN: $\mathbf{H}_2 = \mathbf{W}_2 \phi(\mathbf{H}_1)$
- GP: $\phi$ 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: $\phi$ is $\infty-$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? ...

# APPENDIX

# Inference: "Score function method"

$$
\begin{aligned}
\nabla_\theta \mathcal{F} &= \nabla_\theta \mathbb{E}_{q(w;\theta)}[\log p(D, w)] \\
&= \mathbb{E}_{q(w;\theta)}\left[p(D, w)\nabla_\theta \log q(w;\theta)\right] \\
&\approx \frac{1}{K}\sum_{i=1}^{K} p(D, w^{(k)})\nabla_\theta \log q(w^{(k)};\theta), \quad w^{(k)} \overset{iid}{\sim} q(w;\theta)
\end{aligned}
$$

*(Paisley et al., 2012; Ranganath et al., 2014; Mnih and Gregor, 2014, Ruiz et al. 2016)*

# Inference: "Reparameterization gradient"

- Reparametrize $w$ as a transformation $\mathcal{T}$ of a simpler variable $\epsilon$: $w = \mathcal{T}(\epsilon; \theta)$
- $q(\epsilon)$ is now independent of $\theta$

$$\begin{aligned}\nabla_\theta \mathcal{F} &= \nabla_\theta \mathbb{E}_{q(w;\theta)}[\log p(D, w)] \\ &= \mathbb{E}_{q(\epsilon)}\left[\nabla_w p(D, w)|_{w=\mathcal{T}(\epsilon;\theta)} \nabla_\theta \mathcal{T}(\epsilon; \theta)\right]\end{aligned}$$

- For example: $w \sim \mathcal{N}(\mu, \sigma) \xrightarrow{\mathcal{T}} w = \mu + \sigma \cdot \epsilon$, $\epsilon \sim \mathcal{N}(0, 1)$
- MC by sampling from $q(\epsilon)$ (thus obtaining samples from $w$ through $\mathcal{T}$)

*(Salimans and Knowles, 2013; Kingma and Welling, 2014, Ruiz et al. 2016)*

# Inference: “Reparameterization gradient”

- Reparametrize $w$ as a transformation $\mathcal{T}$ of a simpler variable $\epsilon$: $w = \mathcal{T}(\epsilon; \theta)$
- $q(\epsilon)$ is now independent of $\theta$

$$\begin{aligned}\nabla_\theta \mathcal{F} &= \nabla_\theta \mathbb{E}_{q(w;\theta)}[\log p(D, w)] \\ &= \mathbb{E}_{q(\epsilon)}\left[\nabla_w p(D, w)|_{w=\mathcal{T}(\epsilon;\theta)} \nabla_\theta \mathcal{T}(\epsilon; \theta)\right]\end{aligned}$$

- For example: $w \sim \mathcal{N}(\mu, \sigma) \xrightarrow{\mathcal{T}} w = \mu + \sigma \cdot \epsilon,\ \epsilon \sim \mathcal{N}(0, 1)$
- MC by sampling from $q(\epsilon)$ (thus obtaining samples from $w$ through $\mathcal{T}$)

*(Salimans and Knowles, 2013; Kingma and Welling, 2014, Ruiz et al. 2016)*

# Inference: "Reparameterization gradient"

- Reparametrize $w$ as a transformation $\mathcal{T}$ of a simpler variable $\epsilon$: $w = \mathcal{T}(\epsilon; \theta)$

- $q(\epsilon)$ is now independent of $\theta$

$$
\begin{aligned}
\nabla_\theta \mathcal{F} &= \nabla_\theta \mathbb{E}_{q(w;\theta)}[\log p(D, w)] \\
&= \mathbb{E}_{q(\epsilon)} \left[ \nabla_w p(D, w)|_{w=\mathcal{T}(\epsilon;\theta)} \nabla_\theta \mathcal{T}(\epsilon; \theta) \right]
\end{aligned}
$$

- For example: $w \sim \mathcal{N}(\mu, \sigma) \xrightarrow{\mathcal{T}} w = \mu + \sigma \cdot \epsilon,\ \epsilon \sim \mathcal{N}(0, 1)$

- MC by sampling from $q(\epsilon)$ (thus obtaining samples from $w$ through $\mathcal{T}$)

*(Salimans and Knowles, 2013; Kingma and Welling, 2014, Ruiz et al. 2016)*

# Inference: "Reparameterization gradient"

- Reparametrize $w$ as a transformation $\mathcal{T}$ of a simpler variable $\epsilon$: $w = \mathcal{T}(\epsilon; \theta)$

- $q(\epsilon)$ is now independent of $\theta$

$$\begin{aligned} \nabla_\theta \mathcal{F} &= \nabla_\theta \mathbb{E}_{q(w;\theta)}[\log p(D, w)] \\ &= \mathbb{E}_{q(\epsilon)}\left[ \nabla_w p(D, w)|_{w=\mathcal{T}(\epsilon;\theta)} \nabla_\theta \mathcal{T}(\epsilon; \theta) \right] \end{aligned}$$

- For example: $w \sim \mathcal{N}(\mu, \sigma) \xrightarrow{\mathcal{T}} w = \mu + \sigma \cdot \epsilon,\ \epsilon \sim \mathcal{N}(0, 1)$

- MC by sampling from $q(\epsilon)$ (thus obtaining samples from $w$ through $\mathcal{T}$)

*(Salimans and Knowles, 2013; Kingma and Welling, 2014, Ruiz et al. 2016)*

We want the expectation $q(w;\theta)$ to appear on the left of $\nabla_\theta$, otherwise it's difficult. In score function we use a property of the log. In reparam. gradient we just reparameterize the main argument of the problematic $q(w;\theta)$ so then $\nabla_\theta$ does not depend on this argument (the $w$) and can again be pushed.
The reparam. gradient has lower variance in practice, because it's a "richer" estimator, (e.g. has more info, like curvature about true gradient). But it's more restrictive, in that it works when $w$ is continuous and [...].
Notice we can't do the fully naive MC where we bring the $\nabla$ inside the integral and do: $\int q(w;\theta)\nabla_\theta p(D,w)dw$, because this doesn't make sense as the derivative for $\theta$ cannot be applied to $p(D,w)$ that does not contain $\theta$!

$$\nabla_\theta \mathcal{F}(\theta) =$$

$$\nabla_\theta \int_w q(w;\theta) \log p(D,w) = \tag{1}$$

$$\int_w \nabla_\theta q(w;\theta) \log p(D,w)] = \tag{2}$$

$$\int_w q(w;\theta) \nabla_\theta \log q(w;\theta) \log p(D,w) = \tag{3}$$

$$\int_w q(w;\theta) p(D,w) \nabla_\theta \log q(w;\theta) \tag{4}$$

$$\text{setting: } w = \mathcal{T}(\epsilon;\theta) \Rightarrow \tag{5}$$

$$\int_w q(w;\theta)\log p(D,w) = \int_\epsilon q(\epsilon)\log p(D,\mathcal{T}(\epsilon;\theta)) \tag{6}$$

So:

$$\nabla_\theta \mathcal{F}(\theta) =$$

$$\nabla_\theta \int_w q(w;\theta)\log p(D,w) = \tag{7}$$

$$\int_\epsilon \nabla_\theta q(\epsilon)\log p(D,\mathcal{T}(\epsilon;\theta)) = \tag{8}$$

$$\int_\epsilon q(\epsilon)\nabla_\theta \log p(D,\mathcal{T}(\epsilon;\theta)) = \tag{9}$$

$$\int_\epsilon q(\epsilon)\nabla_w \log p(D,\mathcal{T}(\epsilon;\theta))\nabla_\theta \mathcal{T}(\epsilon;\theta) \tag{10}$$

where last equality is from chain rule.