In this lecture series, leading research scientists from leading AI research lab, DeepMind, will give 12 lectures on an exciting selection of topics in Deep Learning, ranging from the fundamentals of training neural networks via advanced ideas around memory, attention, and generative modelling to the important topic of responsible innovation.

Please join us for a deep dive lecture series into Deep Learning!

#UCLxDeepMind
General information

Exits:
At the back, the way you came in

Wifi:
UCL guest
TODAY’S SPEAKER

Wojciech Czarnecki

Wojciech Czarnecki is a Research Scientist at DeepMind. He obtained his phd from the Jagiellonian University in Cracow, during which he worked on the intersection of machine learning, information theory and cheminformatics. Since joining DeepMind in 2016, Wojciech has been mainly working on deep reinforcement learning, with a focus on multi-agent systems, such as recent Capture the Flag project or AlphaStar, the first AI to reach the highest league of human players in a widespread professional esport without simplification of the game.
Neural networks are the models responsible for the deep learning revolution since 2006, but their foundations go back as far as to the 1960s. In this lecture we will go through the basics of how these models operate, learn and solve problems. We will also set various terminology/naming conventions to prepare attendees for further, more advanced talks. Finally, we will briefly touch upon more research oriented directions of neural network design and development.
# Plan for this Lecture

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>01</strong></td>
<td><strong>02</strong></td>
<td><strong>03</strong></td>
</tr>
<tr>
<td>Overview</td>
<td>Neural Networks</td>
<td>Learning</td>
</tr>
<tr>
<td><strong>04</strong></td>
<td><strong>05</strong></td>
<td><strong>06</strong></td>
</tr>
<tr>
<td>Pieces of the puzzle</td>
<td>Practical issues</td>
<td>Bonus: Multiplicative interactions</td>
</tr>
</tbody>
</table>
What is not covered in this lecture

01  “Old school”
- (Restricted) Boltzmann Machines
- Deep Belief Networks
- Hopfield Networks
- Self Organising Maps

02  Biologically plausible
- Spiking networks
- Physical Simulators

03  Other
- Capsules
- Graph networks
- Neural Differential Equations
- Convolutional Networks
- Recurrent Neural Networks
Overview

Extra notes/ideas

I removed most of the "explicit" branding, only first slide has DeepMind name on it, everything else - a small logo in the corner. Also no "confidential" info.
Various successes

Goal

Provides a bit of extra excitement, and makes sure people see applications before getting into technicalities

Computer Vision
In a shocking finding, a scientist discovered a herd of unicorns living in a remote, previously unexplored valley in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

The scientist named the population, after their distinctive horn, Odin's Unicorns. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this old phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.
In a shocking finding, scientists discovered a herd of unicorns living in a remote, previously-unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

The scientist named the population, after their distinctive horn, Odys's Unicorns. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Plata, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had a pool that appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley, "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.
Compute

Content

3 factors behind DL revolution

Goal
Establish elements that we will be referring to throughout the talk, so people see the connection between these, and neural networks.
Compute

Data

3 factors behind DL revolution

Goal

Establish elements that we will be referring to throughout the talk, so people see the connection between these, and neural networks.
Compute Data Modularity

Content 3 factors behind DL revolution

Goal Establish elements that we will be referring to throughout the talk, so people see the connection between these, and neural networks.
The deep learning puzzle

High level, modular vision. Composable blocks, with some dynamical system (often in the form of gradient descent). Mention differentiability.

Goal
Get intuition of where we are going, before getting into details.

Extra notes/ideas
We will use this "puzzle idea" throughout the deck, and keep going back to it. Eventually it will evolve to computational graphs.
The deep learning puzzle

How to adjust this input, if my output needs to change?

What to output?

Data

Node

Target

Node

Node

Node

Node

Loss

Target

Data
The deep learning puzzle

High level, modular vision. Composable blocks, with some dynamical system (often in the form of gradient descent). Mention differentiability.

Goal
Get intuition of where we are going, before getting into details

Differentiable wrt. inputs

What to output?

Data

Node

Node

Node

Loss

Target

Node

@ylecun

Some folks still seem confused about what deep learning is. Here is a definition:

DL is constructing networks of parameterized functional modules & training them from examples using gradient-based optimization. facebook.com/722677142/post...

3:32 PM · Dec 24, 2019 · Facebook

517 Retweets 1.9K Likes

@DeepSphker

Rephrasing @ylecun with my own words: DL is a collection of tools to build complex modular differentiable functions. These tools are devoid of meaning, it is pointless to discuss what DL can or cannot do. What gives meaning to it is how it is trained and how the data is fed to it.

3:43 PM · Dec 25, 2019 · Twitter for iPhone

90 Retweets 464 Likes
2 Neural networks
How do real networks work, or rather a super simplified picture from 50s (?)

Goal
This is how things started, and it is also a good place to show that ANNs are inspired by reality, but not trying to reproduce it.

Human brain is estimated to contain around 86,000,000,000 of such neurons. Each is connected to thousands of other neurons.

Want to learn more?

- Connected to others
- Represents simple computation
- Has inhibition and excitation connections
- Has a state
- Outputs spikes

Connected to others
Represents simple computation
Has inhibition and excitation connections
Has a state
Outputs spikes
Artificial neuron

The goal of simple artificial neurons models is to reflect some neurophysiological observations, not to reproduce their dynamics.

\[ \sum_{i=1}^{d} w_i x_i + b \]

\[ \sum_{i=0}^{d} w_i x_i \quad x_0 := 1 \]

Want to learn more?
McCulloch, Warren S.; Pitts, Walter
A logical calculus of the ideas immanent in nervous activity
Bulletin of Mathematical Biophysics. 5 (4): 115–133. (1943)

- Easy to compose
- Represents simple computation
- Has inhibition and excitation connections
- Is stateless wrt. time
- Outputs real values
The goal of simple artificial neurons models is to reflect some neurophysiological observations, not to reproduce their dynamics.
In Machine Learning **linear** really means **affine**. **Neurons** in a layer are often called **units**. **Parameters** are often called **weights**.
Isn’t this just linear regression?
Note that we stick to color coding, blue inputs, black/dark blue nodes. Losses related things will be orange.
Single layer neural networks
Puzzle view of a single layer network

Goal
Exemplification + reason to go for the next piece - sigmoid

Extra notes/ideas
Grey boxes will be used to indicate that we are “missing” something at this point, and next slide will fill in the missing piece
Puzzle view of a single layer network

Goal: Exemplification + reason to go for the next piece - sigmoid [next slide]

Extra notes/ideas:
Grey boxes will be used to indicate that we are “missing” something at this point, and next slide will fill in the missing piece.
**Sigmoid activation function**

\[ f_\sigma(x) = \frac{1}{1 + e^{-x}} \]

\[ f_\sigma(x) = \frac{e^x}{e^x + 1} \]

*Activation functions* are often called **non-linearities**. Activation functions are applied **point-wise**.

*Want to learn more?*


- Introduces non-linear behaviour
- Produces probability estimate
- Has simple derivatives
- Saturates
- Derivatives vanish
Loss
Target
Data
Sigmoid
Linear

Puzzle view of a single layer network
Goal
Exemplification + reason to go for the next piece - sigmoid [next slide]

Extra notes/ideas
Grey boxes will be used to indicate that we are “missing” something at this point, and next slide will fill in the missing piece
Cross entropy

\[ \ell_{\text{CE}}(p, t) = -[t \log p + (1 - t) \log(1 - p)] \]

Cross entropy loss is also called negative log likelihood or logistic loss.

Want to learn more?

Murphy, Kevin Machine Learning: A Probabilistic Perspective (2012)

- Encodes negation of logarithm of probability of correct classification
- Composable with sigmoid
- Numerically unstable
The simplest “neural” classifier

Cross entropy loss is also called negative log likelihood or logistic loss. Being additive over samples allows for efficient learning.

$$L_{CE}(p, t) = -\sum_{i=1}^{n}[t^{(i)} \log p^{(i)} + (1 - t^{(i)}) \log(1 - p^{(i)})]$$

Want to learn more?


- Encodes negation of logarithm of probability of entirely correct classification
- Equivalent to logistic regression model
- Numerically unstable
Softmax

\[ f_{sm}(x) = \frac{e^x}{\sum_{j=1}^{k} e^{x_j}} \]

\[ f_{sm}([x, 0]) = \left[ \frac{e^x}{e^x + e^0}, \frac{e^0}{e^x + e^0} \right] = [f_\sigma(x), 1 - f_\sigma(x)] \]

Softmax is the most commonly used final activation in classification. It can also be used to have a smooth version of maximum.

Want to learn more?


- Multi-dimensional generalisation of sigmoid
- Produces probability estimate
- Has simple derivatives
- Saturates
- Derivatives vanish
**Softmax + Cross entropy**

\[
\ell_{CE}(f_{sm}(x), t) = -\sum_{j=1}^{k} t_j \log[f_{sm}(x_j)] = -\sum_{j=1}^{k} t_j x_j - \log \sum_{l=1}^{k} e^{x_l}.
\]

Widely used not only in **classification** but also in **RL**. Cannot represent sparse outputs (**sparsemax**). **Does not scale** too well with \(k\).

- Encodes negation of logarithm of probability of entirely correct classification
- Equivalent to multinomial logistic regression model
- Numerically stable combination

Want to learn more?

Uses

Handwritten digits recognition at 92% level.

Highly dimensional spaces are surprisingly easy to shatter with hyperplanes.

Widely used in commercial applications.

For a long time a crucial model for Natural Language Processing under the name of MaxEnt (Maximum Entropy Classifier).
... and limitations

What they are not good for - complex things like Go or Chess

Goal

Justifies deep learning. Provides us with long-term goal.
... and limitations
Two layer neural networks
Loss
Linear
Node
Cross entropy
Target
Data
Linear
Softmax
Sigmoid
Cross entropy
Target
Extra notes/ideas

We will go step by step through the xor solution with a simple neural net, we don't need any text for that.
Extra notes/ideas

- Introducing extra colors for simpler visual identification of projections.
- Lines are hidden neurons.
Introducing extra colors for simpler visual identification of projections. Lines are hidden neurons.
Extra notes/ideas
We add projections

\[
W = \begin{bmatrix}-1 & 1 \\ 1 & -1\end{bmatrix} \quad b = \begin{bmatrix}-4 \\ -4\end{bmatrix}
\]
Extra notes/ideas

Bend the space with sigmoids

\[ W = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad b = \begin{bmatrix} -4 \\ -4 \end{bmatrix} \]
\[ W = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad b = \begin{bmatrix} -4 \\ -4 \end{bmatrix} \]
Extra notes/ideas

And separate linearly now. Done!

\[
W = \begin{bmatrix}
-1 & 1 \\
1 & -1
\end{bmatrix}
\quad b = \begin{bmatrix}
-4 \\
-4
\end{bmatrix}
\]
1-hidden layer network vs XOR

With just 2 hidden neurons we solve XOR.

Hidden layer allows us to bend and twist input space.

We use linear model on top, to do the classification.

Want to learn more?

Hidden layer provides non-linear input space transformation so that final linear layer can classify.
### Data
Which dataset do you want to use?
- [ ] Iris
- [ ] MNIST
- [ ] Fashion MNIST
- [ ] Imagenet
- [ ] Other

Ratio of training to test data: 50%
Noise: 0
Batch size: 10

### Features
Which properties do you want to feed in?
- [ ] $X_1$
- [ ] $X_2$
- [ ] $X_1^2$
- [ ] $X_2^2$
- [ ] $X_1X_2$
- [ ] $\sin(X_1)$
- [ ] $\sin(X_2)$

### Output
- Test loss: 0.049
- Training loss: 0.036

Colors show data, neuron and weight values.

- [ ] Show test data
- [ ] Discretize output

---

**Epoch**: 000,218  
**Learning rate**: 0.03  
**Activation**: Sigmoid  
**Regularization**: None  
**Regularization rate**: 0  
**Problem type**: Classification
Universal Approximation Theorem

For any continuous function from a hypercube \([0,1]^d\) to real numbers, and every positive epsilon, there exists a sigmoid-based, 1-hidden layer neural network that obtains at most epsilon error in functional space.

Big enough network can approximate, but not represent any smooth function. The math trick is to show that networks are dense in the space of target functions.

One of the most important theoretical results for Neural Networks
- Shows, that they are extremely expressive
- Tells us nothing about learning
- Size of network grows exponentially

Want to learn more?

Universal Approximation Theorem

For any continuous function from a hypercube $[0,1]^d$ to real numbers, non-constant, bounded and continuous activation function $f$, and every positive epsilon, there exists a 1-hidden layer neural network using $f$ that obtains at most epsilon error in functional space.

Big enough network can approximate, but not represent any smooth function. The math trick is to show that networks are dense in the space of target functions.

One of the most important theoretical results for Neural Networks
- Shows, that they are extremely expressive
- Tells us nothing about learning
- Size of network grows exponentially

Want to learn more?

Kurt Hornik Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2), 251–25 (1991)
Universal Approximation Theorem Intuition

Building 1D UAT visual intuition.

Goal: Make people "feel" what UAT really says, without going into technicalities of the proof itself.
Universal Approximation Theorem Intuition

Content
Building 1D UAT visual intuition.

Goal
Make people “feel” what UAT really says, without going into technicalities of the proof itself.
Universal Approximation Theorem Intuition

Building 1D UAT visual intuition.

Goal

Make people "feel" what UAT really says, without going into technicalities of the proof itself.
Universal Approximation Theorem Intuition

Building 1D UAT visual intuition.

Goal

Make people “feel” what UAT really says, without going into technicalities of the proof itself.
Universal Approximation Theorem Intuition

Content

Building 1D UAT visual intuition.

Goal

Make people “feel” what UAT really says, without going into technicalities of the proof itself.
Universal Approximation Theorem Intuition

Building 1D UAT visual intuition.

Goal: Make people “feel” what UAT really says, without going into technicalities of the proof itself.
Universal Approximation Theorem

For any continuous function from a hypercube $[0,1]^d$ to real numbers, non-constant, bounded and continuous activation function $f$, and every positive epsilon, there exists a 1-hidden layer neural network using $f$ that obtains at most epsilon error in functional space.

Want to learn more?
Kurt Hornik
Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2), 251–25 (1991)

One of the most important theoretical results for Neural Networks - Shows, that they are extremely expressive - Tells us nothing about learning - Size of network grows exponentially

Big enough network can approximate, but not represent any smooth function.

The math trick is to show that networks are dense in the space of target functions.

http://playground.tensorflow.org/ by Daniel Smilkov and Shan Carter
Deep neural networks
Rectified Linear Unit (ReLU)

One of the most commonly used activation functions. Made math analysis of networks much simpler.

\[ f_{\text{relu}}(x) = \max(0, x) \]

\[ f_{\text{sp}}(x) = \log(1 + e^x) \]

Introduces non-linear behaviour
Creates piecewise linear functions
Derivatives do not vanish
Dead neurons can occur
Technically not differentiable at 0

Want to learn more?
The "old school" view/premise of DL, hierarchical representations.

Goal: Technical content
Expressing symmetries and regularities is much easier with deep model than wide one.

Deep model means many non-linear composition and thus harder learning.

Number of linear regions grows exponentially with depth, and polynomially with width.
Neural networks as computational graphs

Goal
Introduce a toolbox we are all working with.

Extra notes/ideas
We keep the color coding, computational nodes have exactly the same semantics as equations and puzzles from previous slides.
Neural networks as computational graphs
Neural networks as computational graphs

Show things can get messy/non-linear

Goal

Make people understand flexibility
Neural networks as computational graphs

Show things we can have many losses [deep supervision, aux]

Goal

Make people understand flexibility
Neural networks as computational graphs

Show losses don't have to be end nodes

[Sobolev, Aux]

Goal

Make people understand flexibility
Neural networks as computational graphs

Show weights can be shared [convnet, rnn]

Goal

Make people understand flexibility
Linear algebra recap

**Gradient**

\[
y = f(x) : \mathbb{R}^d \rightarrow \mathbb{R} \\
\frac{\partial y}{\partial x} = \nabla_x f(x) = \left[ \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d} \right]
\]

**Jacobian**

\[
y = f(x) : \mathbb{R}^d \rightarrow \mathbb{R}^k \\
\frac{\partial y}{\partial x} = J_x f(x) = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_d} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_k}{\partial x_1} & \cdots & \frac{\partial f_k}{\partial x_d}
\end{bmatrix}
\]
Gradient descent recap

\[ \theta_{t+1} := \theta_t - \alpha_t \nabla_\theta L(\theta_t) \]

\[ \nabla_\theta L(\theta_t) = \nabla_\theta \sum_i \ell(g(x^{(i)}, \theta_t), t^{(i)}) \]

\[ = \sum_i \nabla_\theta \ell(g(x^{(i)}, \theta_t), t^{(i)}) \]

Choice of learning rate is critical. Main learning algorithm behind deep learning. Many modifications: Adam, RMSProp, ...

Want to learn more?


- Works for any “smooth enough” function
- Can be used on non-smooth targets but with less guarantees
- Converges to local optimum
Neural networks as computational graphs - API

- **Forward pass**: $f(x)$
- **Backward pass**: $J_x f(x)$
Neural networks as computational graphs - API

**Forward pass**

\[ f(x) \]

**Backward pass**

\[ \frac{\partial L}{\partial y} J_x f(x) \]
Gradient descent and computational graph

\[ \theta_{t+1} := \theta_t - \alpha_t \nabla_\theta L(\theta_t) \]

\[ \nabla_\theta L(\theta_t) = \nabla_\theta \sum_i \ell(g(x^{(i)}, \theta_t), t^{(i)}) = \sum_i \nabla_\theta \ell(g(x^{(i)}, \theta_t), t^{(i)}) \]

Want to learn more?
\[ \theta_{t+1} := \theta_t - \alpha_t \nabla_{\theta} L(\theta_t) \]

\[ \nabla_{\theta} L(\theta_t) = \nabla_{\theta} \sum_i \ell(g(x^{(i)}, \theta_t), t^{(i)}) = \sum_i \nabla_{\theta} \ell(g(x^{(i)}, \theta_t), t^{(i)}) \]
Chain rule, backprop and automatic differentiation

\[ y = f(g(x)) \]

\[ \frac{\partial y}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \]

\[ y = f(g(x)) \]

\[ \frac{\partial y}{\partial x} = \sum_{i=1}^{m} \frac{\partial f}{\partial g(i)} \frac{\partial g(i)}{\partial x} \]
Chain rule, backprop and automatic differentiation

\[ y = f(g(x)) \quad \frac{\partial y}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \]

\[ y = f(g(x)) \quad \frac{\partial y}{\partial x} = \sum_{i=1}^{m} \frac{\partial f}{\partial g^{(i)}} \frac{\partial g^{(i)}}{\partial x} \]
Chain rule, backprop and automatic differentiation

\[ y = f(g(x)) \]
\[ \frac{\partial y}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \]

\[ y = f(g(x)) \]
\[ \frac{\partial y}{\partial x} = \sum_{i=1}^{m} \frac{\partial f}{\partial g^{(i)}} \frac{\partial g^{(i)}}{\partial x} \]
Chain rule, backprop and automatic differentiation

\[ y = f(g(x)) \quad \frac{\partial y}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \]

\[ y = f(g(x)) \quad \frac{\partial y}{\partial x} = \sum_{i=1}^{m} \frac{\partial f}{\partial g^{(i)}} \frac{\partial g^{(i)}}{\partial x} \]
Chain rule, backprop and automatic differentiation

\[ y = f(g(x)) \quad \frac{\partial y}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \]

\[ y = f(g(x)) \quad \frac{\partial y}{\partial x} = \sum_{i=1}^{m} \frac{\partial f}{\partial g^{(i)}} \frac{\partial g^{(i)}}{\partial x} \]
Linear layer as a computational graph

\[ f_{\text{linear}}(x, W, b) = Wx + b \]

Note that backward pass is a computational graph itself.

\[
\begin{align*}
\frac{\partial L}{\partial x} &= \frac{\partial L}{\partial y} W \\
\frac{\partial L}{\partial W} &= \left( \frac{\partial L}{\partial y} \right)^T x^T \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial y}
\end{align*}
\]

Symmetry between weights and inputs

Biases are adjusted proportional to error
ReLU as a computational graph

We usually put “gradient” at zero to be equal to zero.

\[
f_{\text{relu}}(x) = \max(0, x)
\]

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \odot 1_{y > 0}
\]

Can be seen as gating the incoming gradients. The ones going through neurons that were active are passed through, and the rest zeroed.
Softmax as a computational graph

\[ f_{\text{sm}}(\mathbf{x}) = \frac{e^x}{\sum_{j=1}^{k} e^{x_j}} \]

Since exponents of big numbers will cause overflow, it is rarely explicitly written like this.

\[
\frac{\partial L}{\partial x_j} = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} y_i (\delta_{ij} - y_j)
\]

\[
= \frac{\partial L}{\partial y} - y \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}
\]

Backwards pass is essentially a difference between incoming gradient and our output.
Cross entropy as a computational graph

Even though it is a loss, we could still multiply its backwards by another incoming errors.

\[ \ell_{CE}(p, t) = -t^T \log p \]

\[ \frac{\partial L}{\partial p} = -t \odot p \quad \text{Dividing by } p \text{ can be numerically unstable} \]

\[ \frac{\partial L}{\partial t} = -\log p \quad \text{We can also backprop into labels themselves} \]
Cross entropy with logits as a computational graph

For numerical stability it is usually a single operation in a computational graph.

\[ \ell_{CE}(f_{sm}(x), t) = - \sum_{j=1}^{k} t \log f_{sm}(x) \]

\[ \frac{\partial L}{\partial x} = t - x \quad \text{Simplifies extremely!} \]

\[ \frac{\partial L}{\partial t} = - \log f_{sm}(x) \quad \text{We can also backprop into labels themselves} \]
Example - 3 layer MLP with ReLU activations
Example - 3 layer MLP with ReLU activations

\[ x \times W_3 + b_3 \] + \[ \text{relu}(x \times W_2 + b_2) \] + \[ \text{relu}(x \times W_1 + b_1) \] + \[ \exp(\sum \text{slice}(\text{slice}(\text{slice}(\text{slice}(\text{slice}(\text{slice}(\text{slice}(\theta))))))) \] 

Out

\[ \log \text{div} (t \times \text{neg} L) \]
Example - 3 layer MLP with ReLU activations

\begin{align*}
\theta_{t+1} &= \theta_t - \alpha_t \sum_i \nabla_{\theta} \ell(g(x^{(i)}, \theta_t), t^{(i)})
\end{align*}
4. Pieces of the puzzle
Max as a computational graph

$\mathbf{f}_{\text{max}}(\mathbf{x}) = \max_i x_i$

$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \odot 1_{x=f_{\text{max}}(x)}$

Gradients only flow through the selected element. Consequently we are not learning how to select.

Used in max pooling.
Let's assume \( p \) is a probability distribution (e.g., one-hot).

**Conditional execution as a computational graph**

\[ f_{\text{cond}}(x, p) = x \odot p \]

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \odot p^T \quad \text{Backwards pass is gated in the same way forward one is} \]

\[ \frac{\partial L}{\partial p} = \frac{\partial L}{\partial y} \odot x^T \quad \text{We can learn conditionals themselves too, just use softmax.} \]
Quadratic loss as a computational graph

Typical loss for all regression problems (e.g. Value function fitting)

\[ \ell_2(x, t) = \| t - x \|^2 \]

\[ \frac{\partial L}{\partial x} = 2(x - t)^T \]

\[ \frac{\partial L}{\partial t} = 2(t - x)^T \]

Backwards pass is just a difference in predictions.

Learning targets is analogous.
Practical issues
Overfitting and regularisation

Classical results from statistics and **Statistical Learning Theory** which analyses the **worst case scenario**.

Want to learn more?


As your model gets more powerful, it can create extremely complex hypotheses, even if they are not needed.

Keeping things simple guarantees that if the training error is small, so will the test be.

- $L_p$ regularisation
- Dropout
- Noising data
- Early stopping
- Batch/Layer norm

Figure from Belkin et al. (2019)
Overfitting and regularisation

As models grow, their learning dynamics changes, and they become less prone to overfitting.

New, exciting theoretical results, also mapping these huge networks onto Gaussian Processes.

New results, that take into consideration learning effects.

Want to learn more?


Figure from Belkin et al. (2019)
Overfitting and regularisation

Model complexity is not as simple as number of parameters.

Want to learn more?

Even big models still need (can benefit from) regularisation techniques.
We need new notions of effective complexity of our hypotheses classes.
Diagnosing and debugging

- **Initialisation** matters
- **Overfit** small sample
- **Monitor** training **loss**
- **Monitor** weights **norms** and **NaNs**
- Add **shape asserts**
- Start with **Adam**
- **Change one thing** at the time

**Want to learn more?**

Karpathy A. A Recipe for Training Neural Networks

- **It is always worth spending time on verifying correctness.**
- Be suspicious of good results more than bad ones.
- Experience is key, just keep trying!
6 Bonus: Multiplicative interactions
What MLPs cannot do?
What MLPs cannot do?

\[ f(x,z) = \langle x, z \rangle \]
Multiplicative interactions

\[ f(x, z, W, U, V, b) = x^T W z + z^T U + V x + b \]

Being able to **approximate** something is not the same as **represent** it.

Want to learn more?

Siddhant M. Jayakumar et al. 

- Multiplicative units unify attention, metric learning and many others
- They enrich the hypothesis space of regular neural networks in a meaningful way
If you want to do research in fundamental building blocks of Neural Networks, **do not seek** to marginally improve the way they behave by finding new activation function.

Ask yourself what current modules cannot represent or guarantee right now, and propose a module that can.
Thank you
Questions